Verifying Security Protocols in Tamarin

Ralf Sasse

Institute of Information Security
ETH Zurich

Tamarin Day 3, v.1

Jan 27, 2016

Roadmap

1. Protocol Security Goals
2. Secrecy
3. Authentication
4. Key-related properties
5. Automated Verification
6. Decidability

Outline

1. Protocol Security Goals
2. Secrecy
3. Authentication
4. Key-related properties
5. Automated Verification
6. Decidability

Protocol Goals

Goals what the protocol should achieve, e.g.,

- Authenticate messages, binding them to their originator
- Ensure timeliness of messages (recent, fresh, ...)
- Guarantee secrecy of certain items (e.g., generated keys)

Protocol Goals

Goals what the protocol should achieve, e.g.,

- Authenticate messages, binding them to their originator
- Ensure timeliness of messages (recent, fresh, ...)
- Guarantee secrecy of certain items (e.g., generated keys)

Most common goals

- secrecy
- authentication (many different forms)

Other goals

- anonymity, non-repudiation (of receipt, submission, delivery), fairness, availability, sender invariance, ...

Protocol Properties and Correctness

What does it mean?

Properties

- Semantics of a security protocol P is a set of traces $\|P\| = \mathit{traces}(P)$. (Traces may be finite or infinite, state- or event-based.)
- Security goal / property ϕ also denotes a set of traces $\|\phi\|$.

Protocol Properties and Correctness

What does it mean?

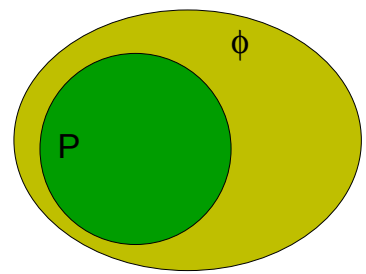
Properties

- Semantics of a security protocol P is a set of traces $\|P\| = \mathit{traces}(P)$. (Traces may be finite or infinite, state- or event-based.)
- Security goal / property ϕ also denotes a set of traces $\|\phi\|$.

Correctness has an exact meaning

- Protocol P satisfies property ϕ, written $P \models \phi$, iff

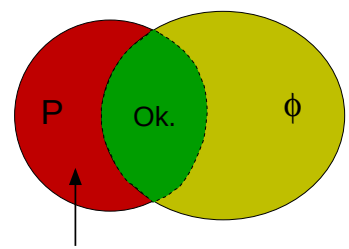
$$\|P\| \subseteq \|\phi\|$$



Ok, no attacks.

- Attack traces are those in

$$\|P\| - \|\phi\|$$



Attacks.

- Every correctness statement is either true or false.
- Later: how do we find attacks or prove correctness?

Formalizing Security Properties

Two approaches

Direct formulation

- Formulate property ϕ directly in terms of send and receive events occurring in protocol traces, i.e., as a set of (or predicate on) traces.
- Drawback: standard properties like secrecy and authentication become highly protocol-dependent, since they need to refer to the concrete protocol messages.

Formalizing Security Properties

Two approaches

Direct formulation

- Formulate property ϕ directly in terms of send and receive events occurring in protocol traces, i.e., as a set of (or predicate on) traces.
- Drawback: standard properties like secrecy and authentication become highly protocol-dependent, since they need to refer to the concrete protocol messages.

Protocol instrumentation

- Idea: insert special claim events into the protocol roles:

$$\text{Claim\_}\texttt{claimtype}(R, t)$$

  where R is the executing role, `claimtype` indicates the type of claim, and t is a message term.
- Serve as interface to express properties independently of protocol.
- Example: $\text{Claim\_}\texttt{secret}(A, N_A)$ claims that N_A is a secret for role A, i.e., not known to the intruder.

Claim Events

Claim events are part of the protocol rules as actions.

Properties of claim events

- Their only effect is to record facts or claims in the protocol trace.
- They cannot be observed, modified, or generated by the intruder.

Claim Events

Claim events are part of the protocol rules as actions.

Properties of claim events

- Their only effect is to record facts or claims in the protocol trace.
- They cannot be observed, modified, or generated by the intruder.

Expressing properties using claim events

- Properties of traces *tr* are expressed in terms of claim events and other actions (e.g., adversary knowledge K) occuring in *tr*.
- Properties are formulated from the point of view of a given role, thus yielding security guarantees for that role.
- We concentrate on secrecy and various forms of authentication, though the approach is not limited to these properties.

Outline

1 Protocol Security Goals

2 Secrecy

3 Authentication

4 Key-related properties

5 Automated Verification

6 Decidability

Role Instrumentation for Secrecy

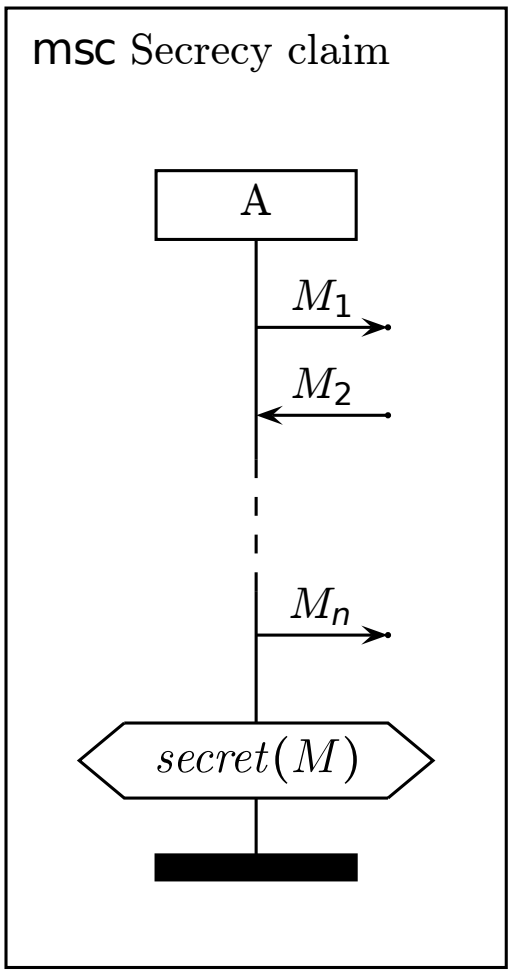
Definition (Secrecy, informally)

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 run remains secret.
- Position: At the end of the role.
- For instance, in NSPK, the 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

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

- Let $tr = tr_1; tr_2; \ldots; tr_k$. We write $x@k$ as a shorthand for $x \in tr_k$.

Formalization of Secrecy

Definition (Secrecy, first attempt)

The secrecy property consists of all traces tr satisfying

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

- Let $tr = tr_1; tr_2; \ldots; tr_k$. We write $x@k$ as a shorthand for $x \in tr_k$.
- Can only require M to remain secret if A 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 rightfully knows 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 adversary control, i.e., sharing all its information with the adversary and participating in protocols upon its direction. We model this by having the agent give its initial secret information to the adversary, which can then mimic the agent's actions.

We note the fact that an agent is compromised by a Rev event in the trace, attached to the rule that passes its initial secrets to the adversary (compare to the creation rule):

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

Exercise: convince yourself that, given the agent's secret, the adversary is capable of performing all of the agent's send and receive steps.

Formalization of Secrecy

Definition (Honesty)

An agent A is honest in a trace tr when $\text{Rev}(A) \notin tr$.
When making a claim in a rule action, all parties B that are expected to be honest need to be listed with a $\text{Honest}(B)$ action in that rule.

Formalization of Secrecy

Definition (Honesty)

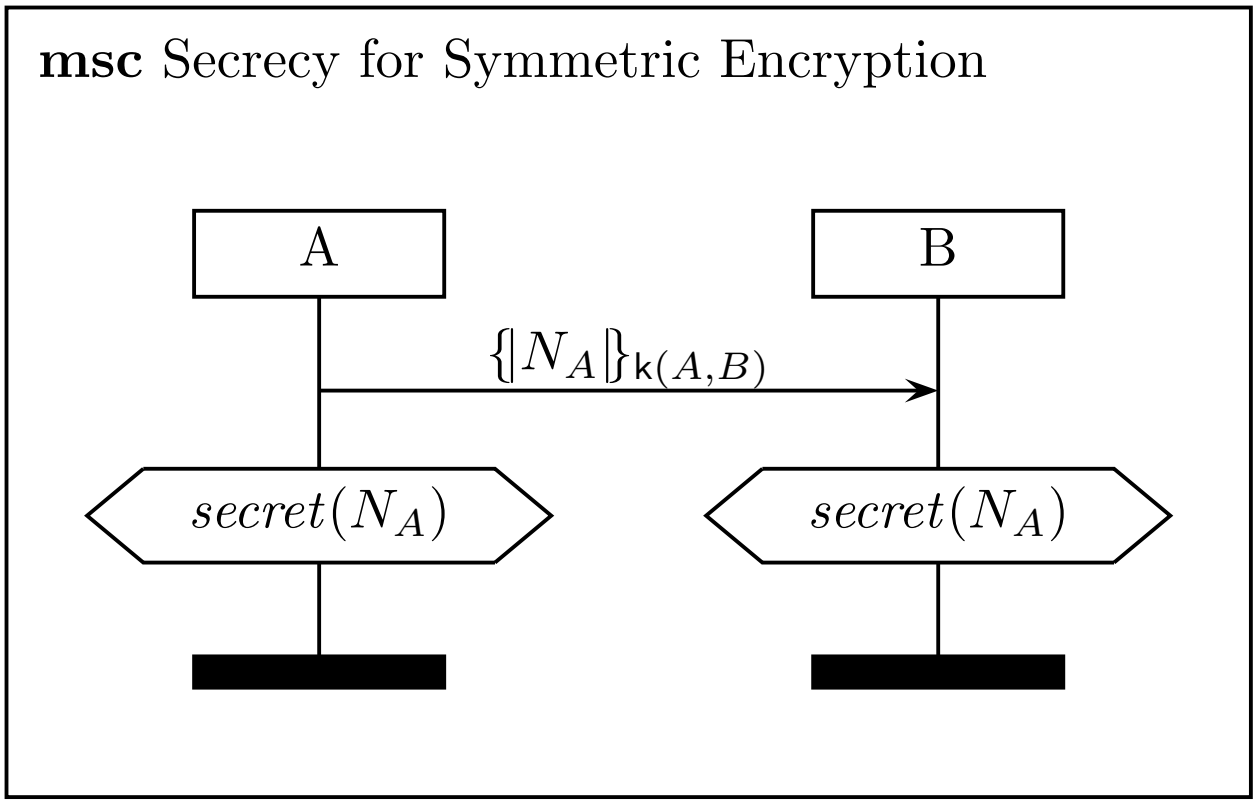
An agent A is honest in a trace tr when $\mathrm{Rev}(A) \notin tr$.
When making a claim in a rule action, all parties B that are expected to be honest need to be listed with a $\mathrm{Honest}(B)$ action in that rule.

Definition (Secrecy)

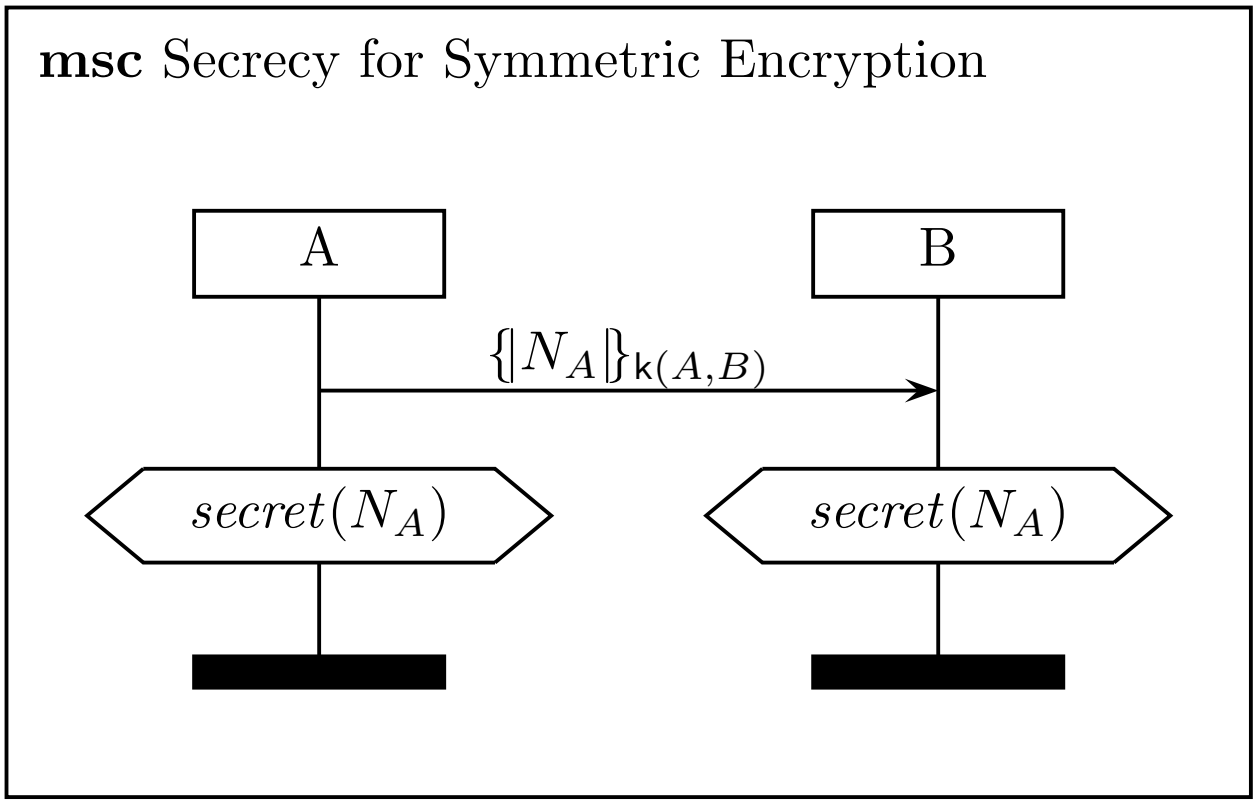
The secrecy property consists of all traces tr satisfying

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

Secrecy Example #1

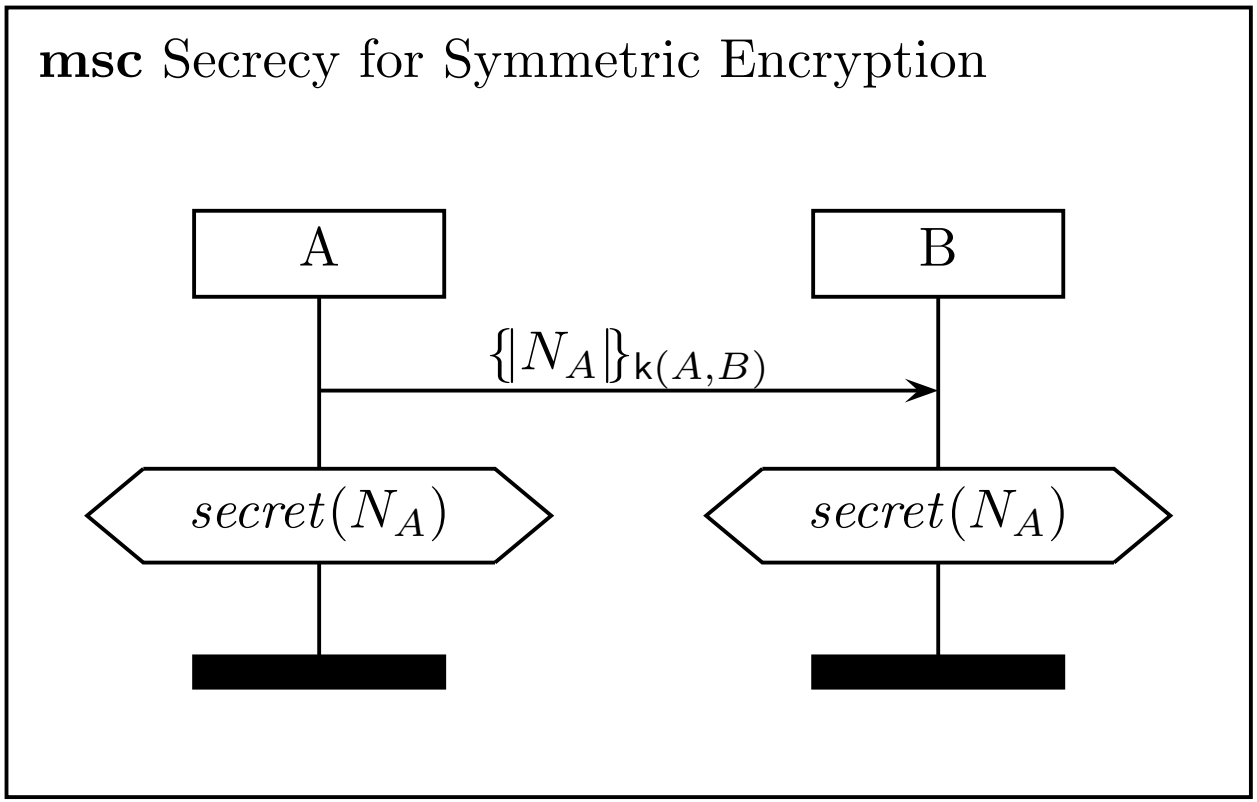


Secrecy Example #1



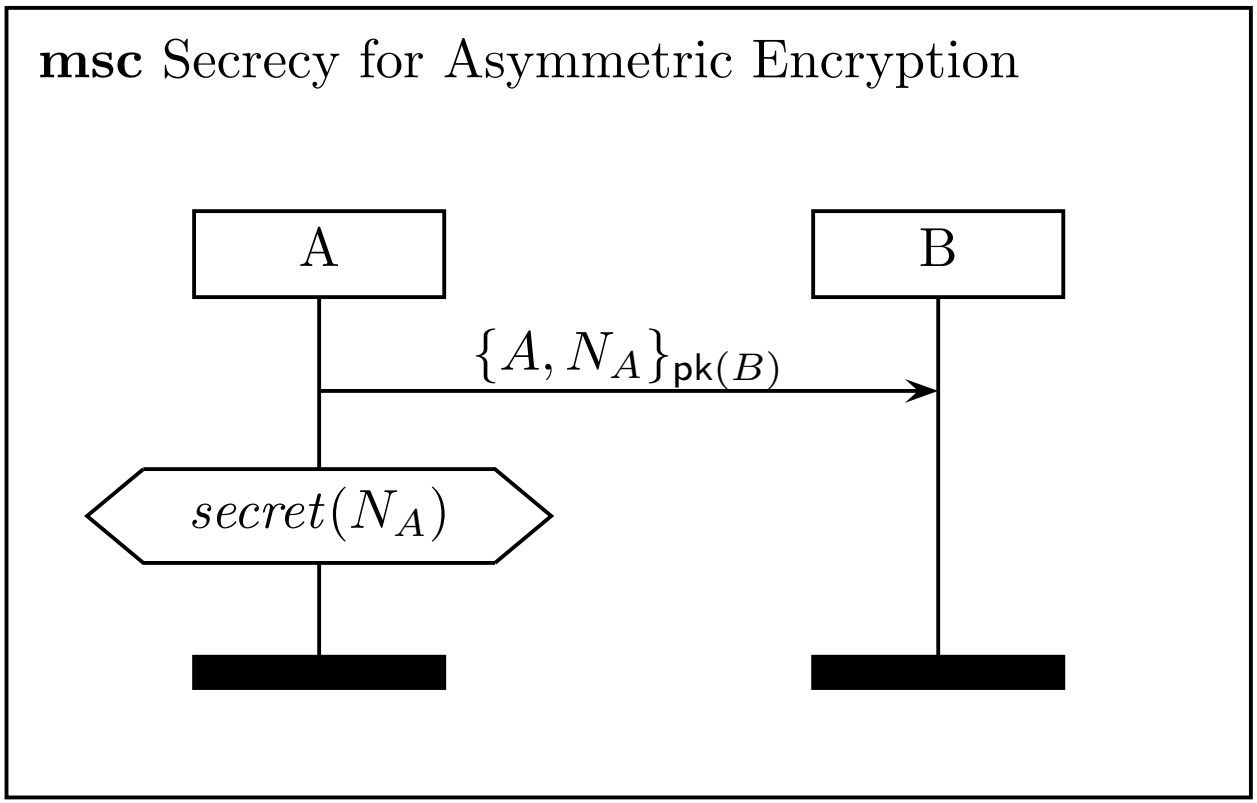
- This is fine: secrecy holds for both A and B.

Secrecy Example #1

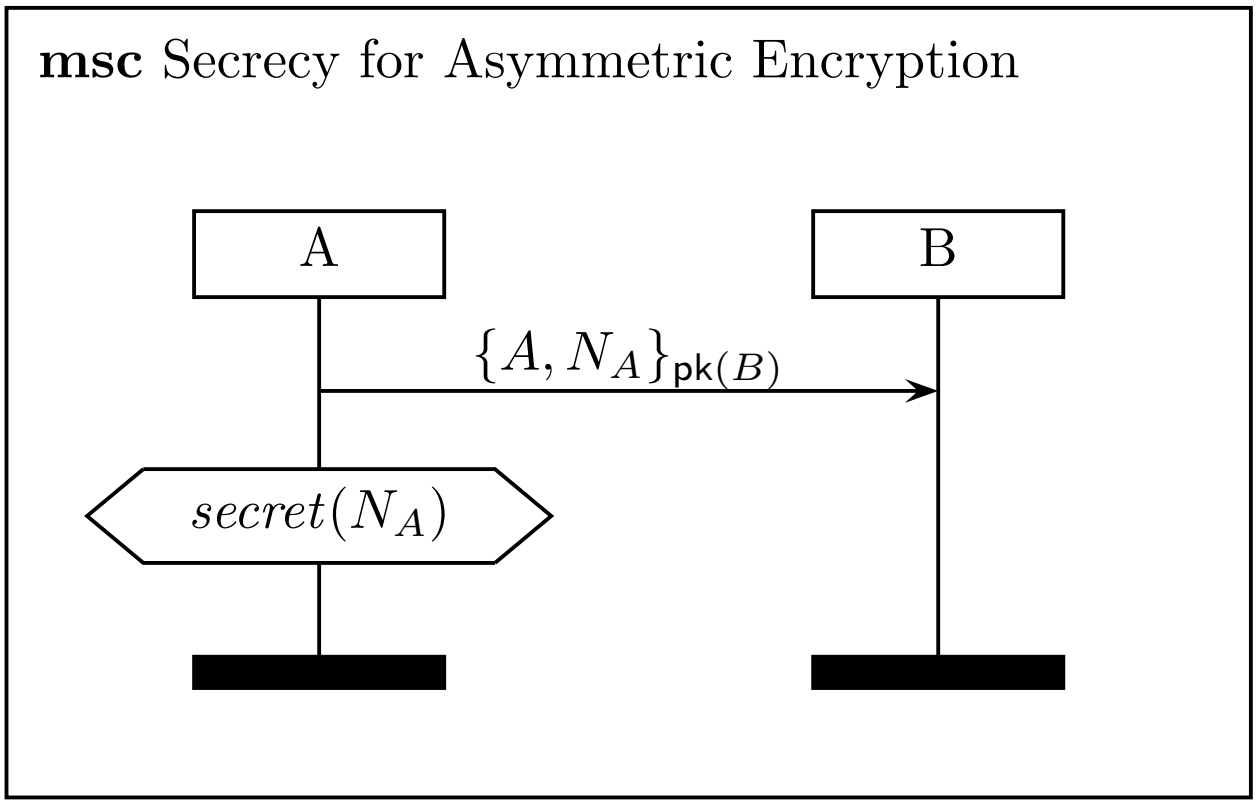


- This is fine: secrecy holds for both A and B.
- We omit the obvious annotations Honest(A), Honest(B) in message sequence charts for 2-party protocols.

Secrecy Example #2

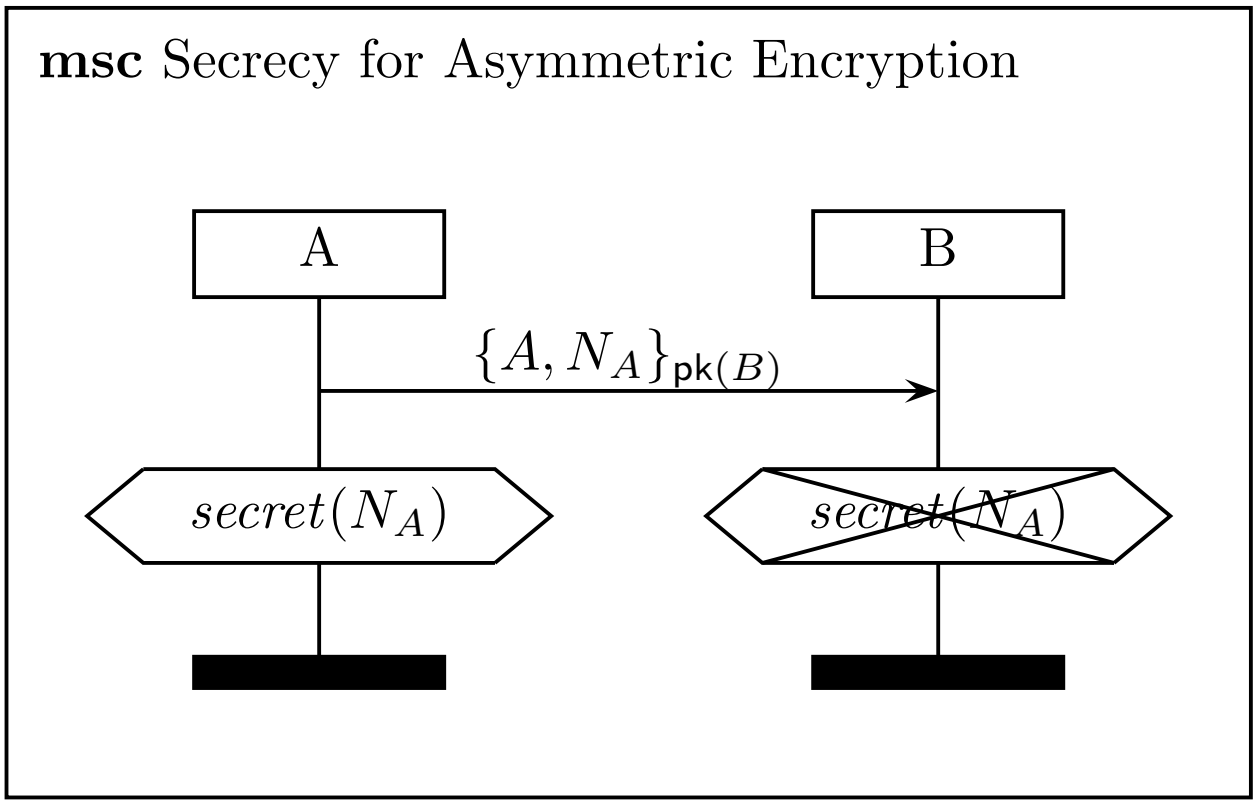


Secrecy Example #2



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

Secrecy Example #2



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

Outline

1 Protocol Security Goals

2 Secrecy

3 **Authentication**

4 Key-related properties

5 Automated Verification

6 Decidability

Authentication

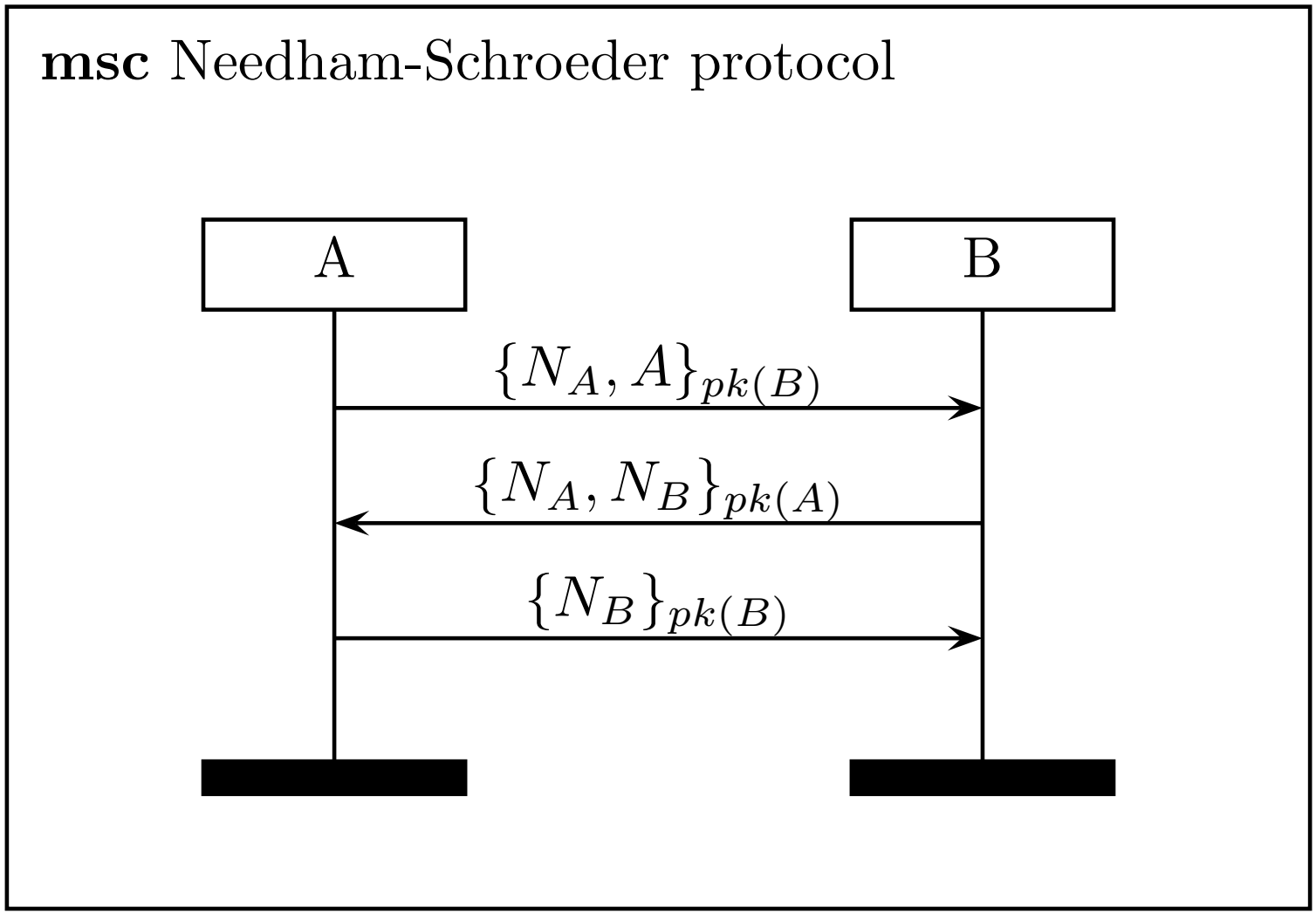
Which authentication are you talking about?

- No unique definition of authentication, but a variety of different forms.
- Considerable effort has been devoted to specifying and classifying, semi-formally or formally, different forms of authentication (e.g., by Cervesato/Syverson, Clark/Jacob, Gollmann, Lowe, Cremers et al.).

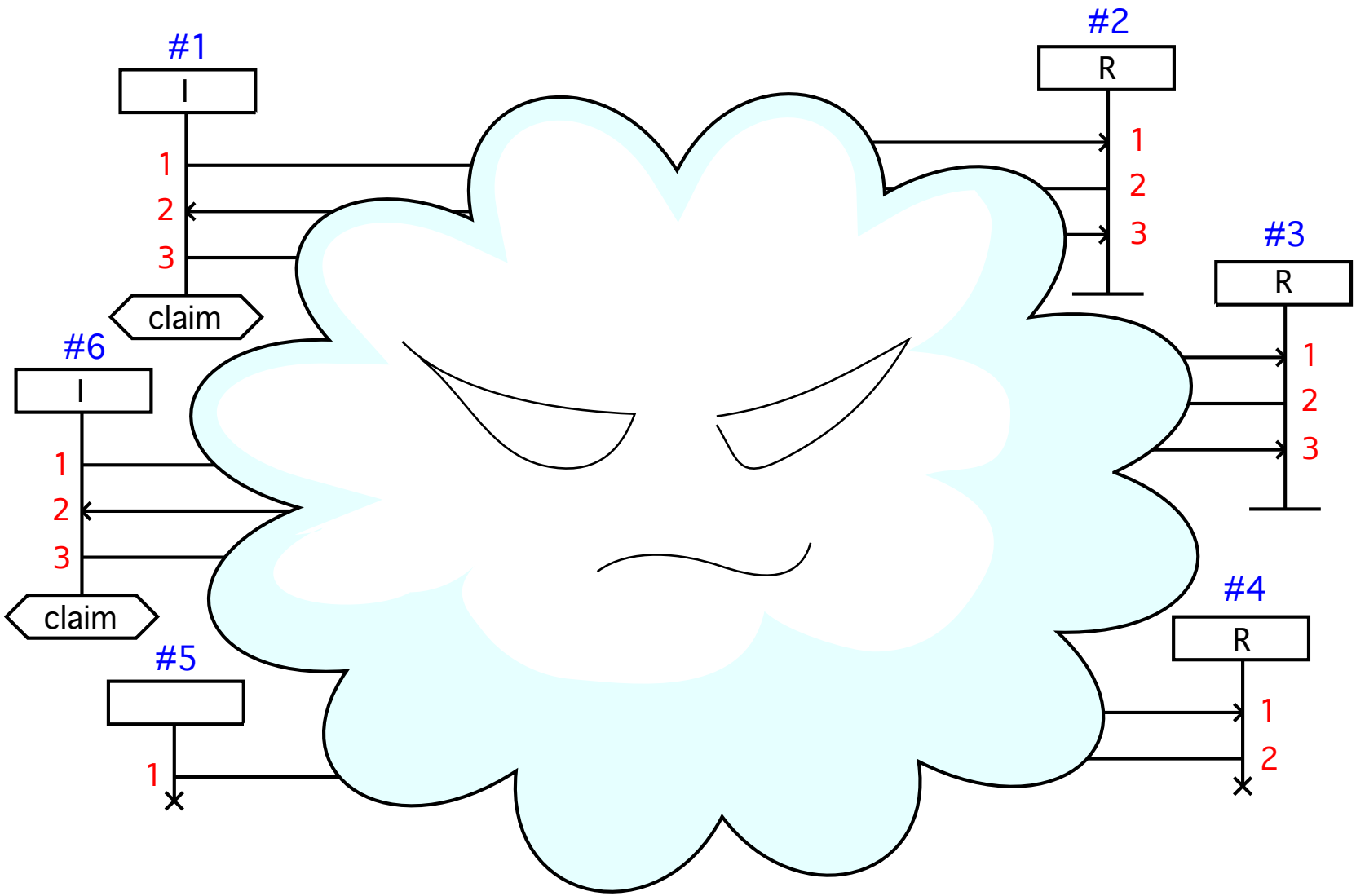
Examples

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

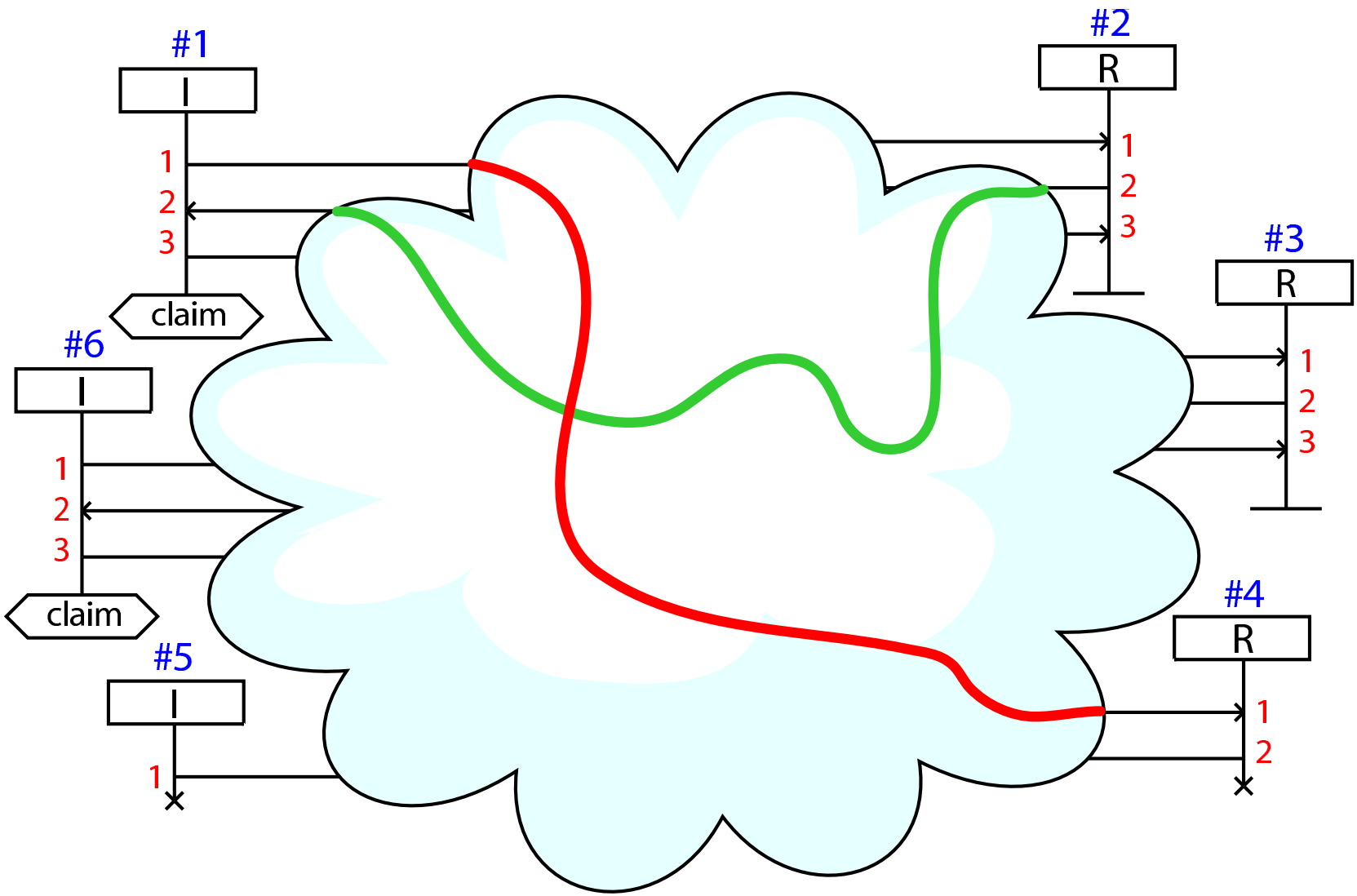
A Perfect (Picture of the) World



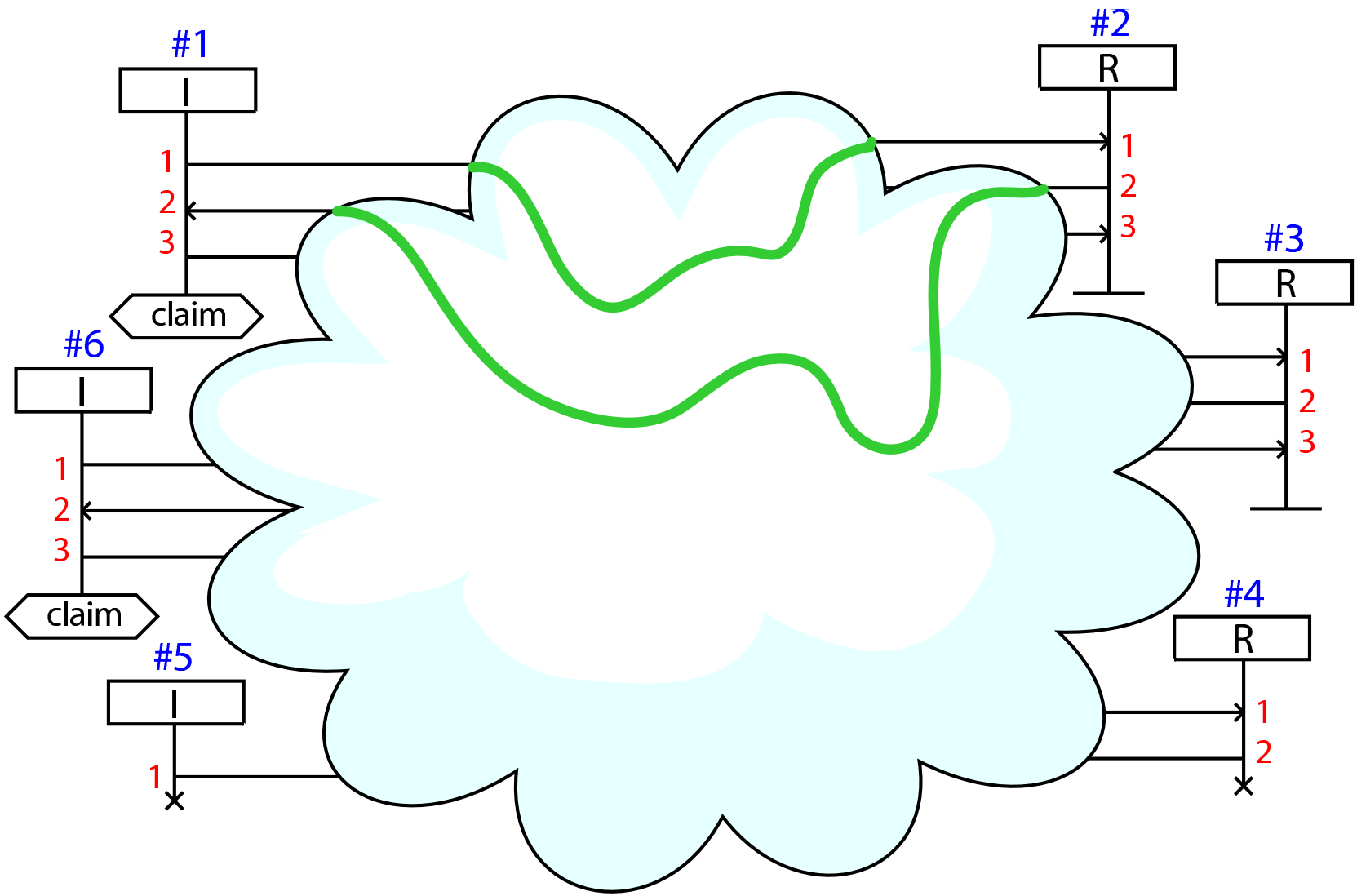
A More Realistic Picture



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.

Also 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.

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

These are quite complex properties. How can we formalize them?

Role Instrumentation for Authentication

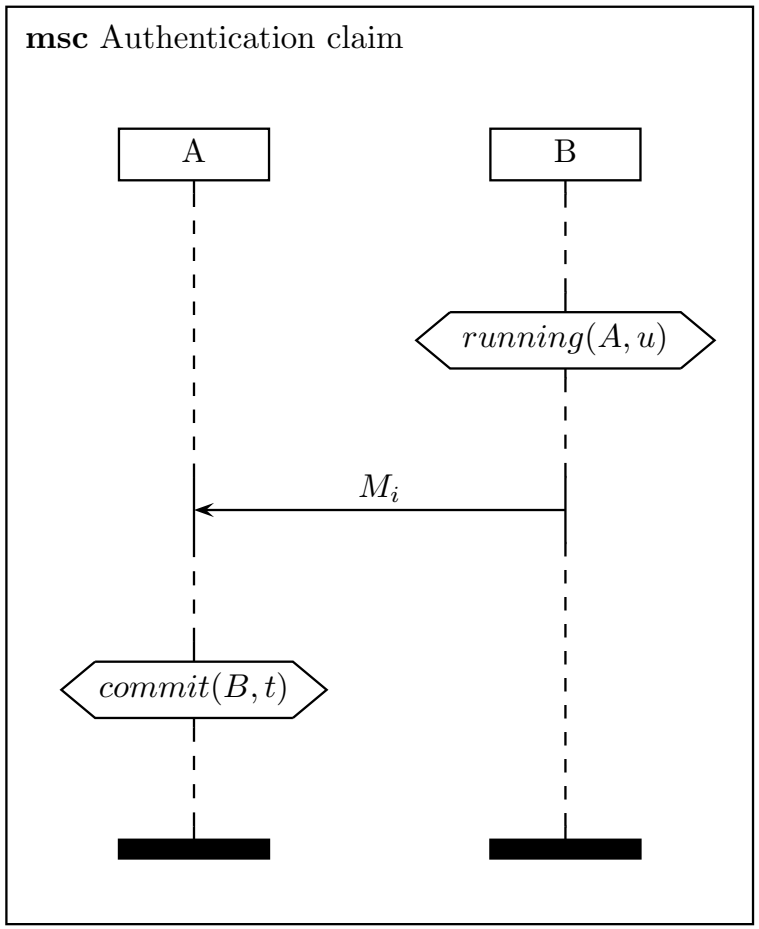
We use two claims to express that role A authenticates role B on t:

In role A:

- Insert a commit claim event $\texttt{Claim\_commit}(A, B, t)$.
- Position: after A can construct t. Typically, at end of A's role.

In role B:

- Insert a running claim event $\texttt{Claim\_running}(B, A, u)$.
- Term u is B's view of t.
- Position: after B can construct u and causally preceding $\texttt{Claim\_commit}(A, B, t)$.



Formalizing Authentication

Definition (Non-injective agreement)

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

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

- Whenever a commit claim is made with honest agents a and b, 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)

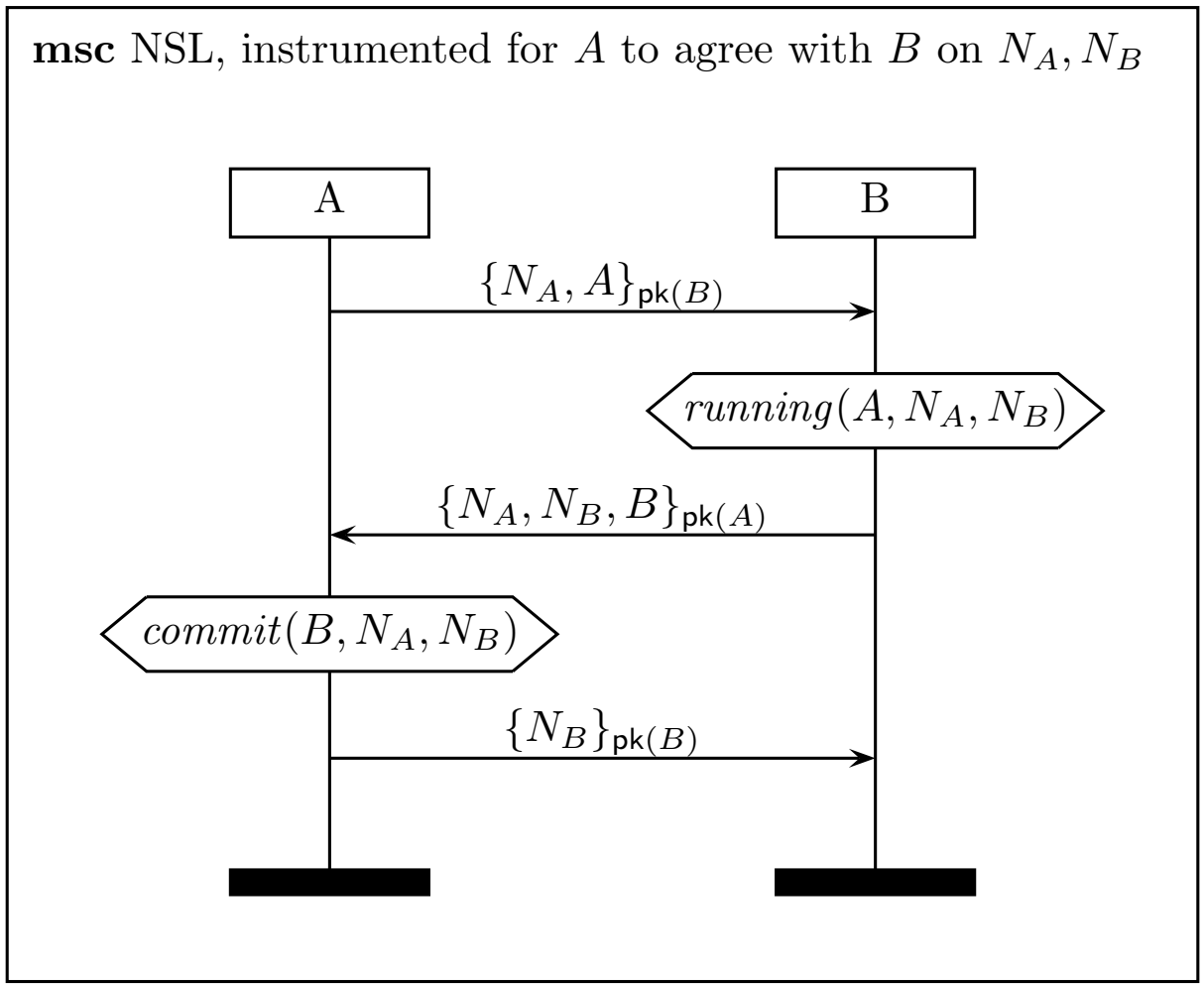
The property $Agreement_{NI}(A, B, t)$ consists of all traces satisfying

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

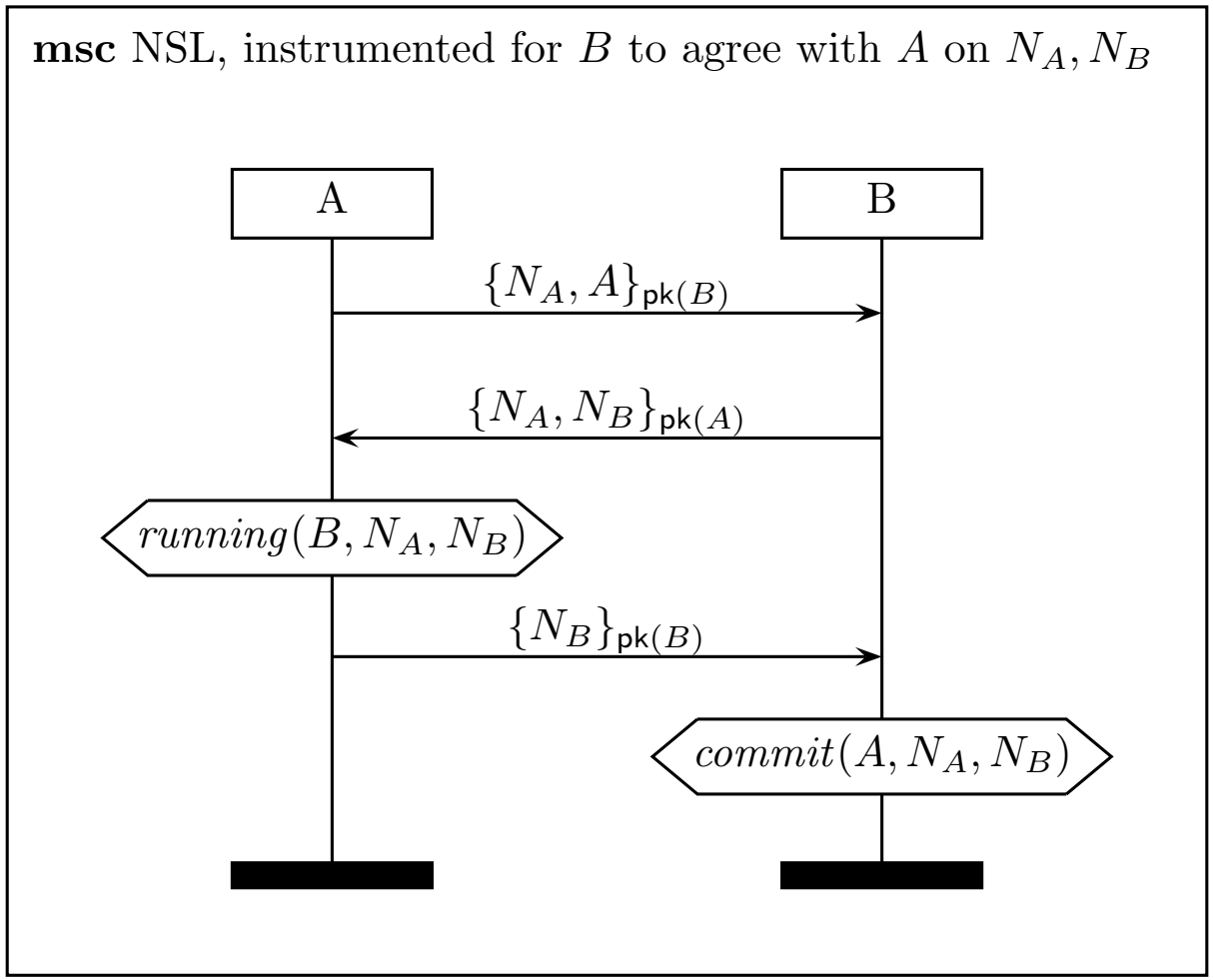
- Whenever a commit claim is made with honest agents a and b, then the peer b must be running with the same parameter t, or the adversary has compromised at least one of the two agents.

Faithfulness What about the ordering of the claims in the trace? This holds even if the running claim succeeds the commit claim!

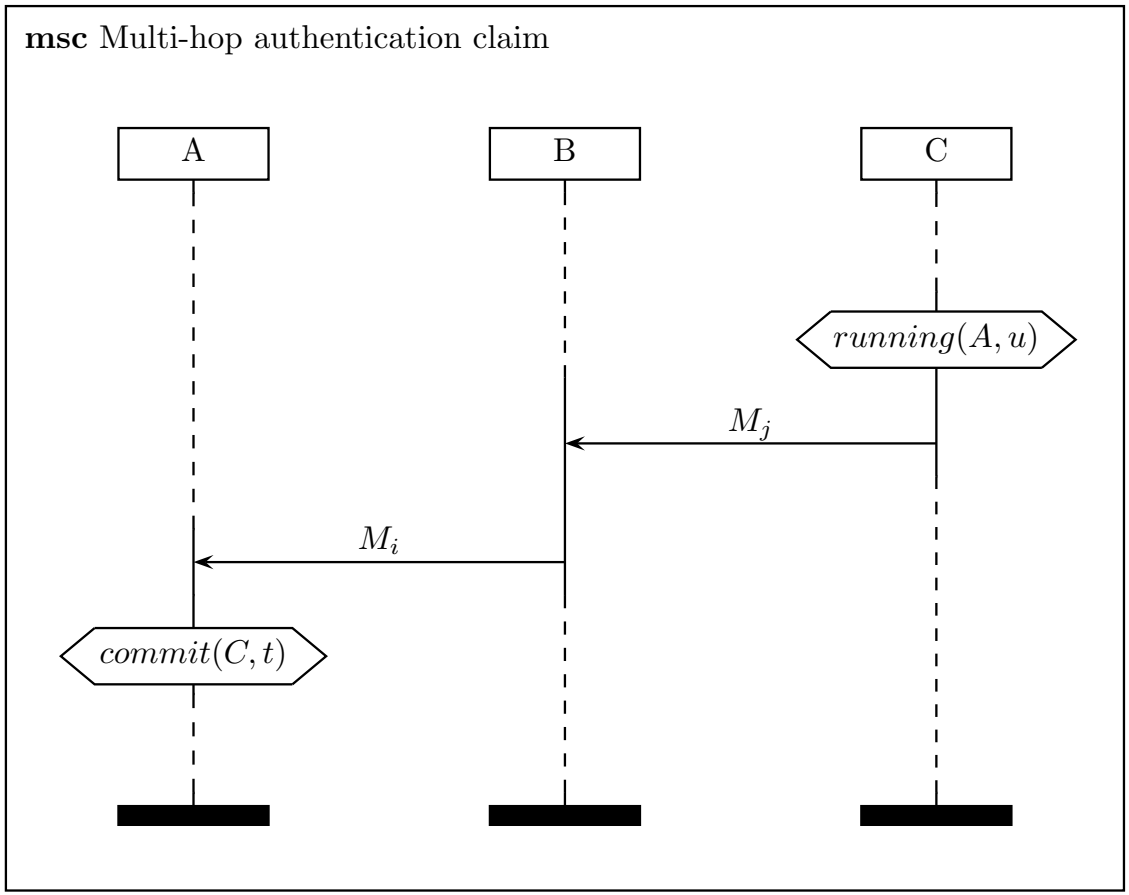
Example: NSL Protocol (1/2)



Example: NSL Protocol (2/2)

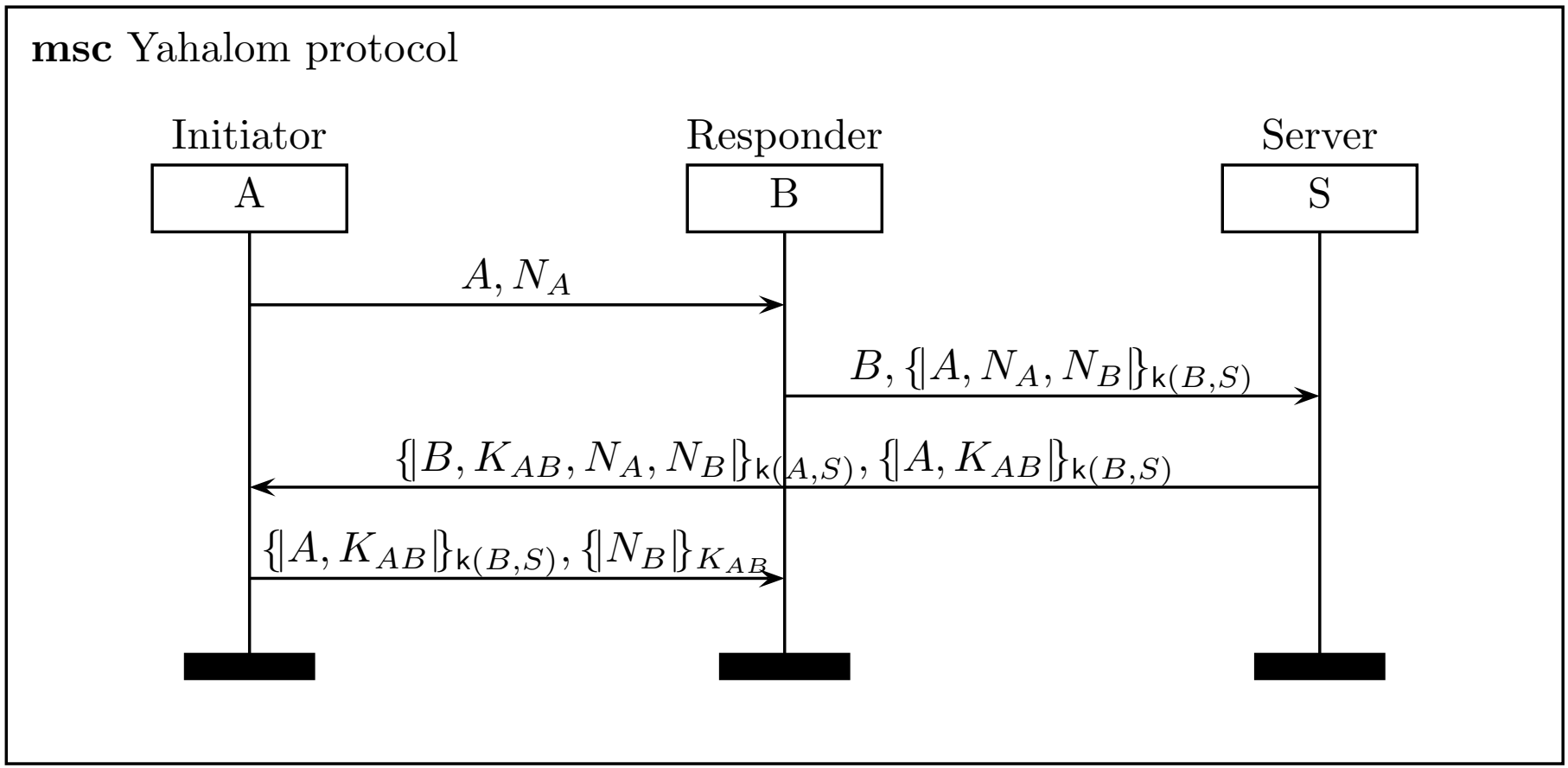


Role Instrumentation for Authentication (cont.)

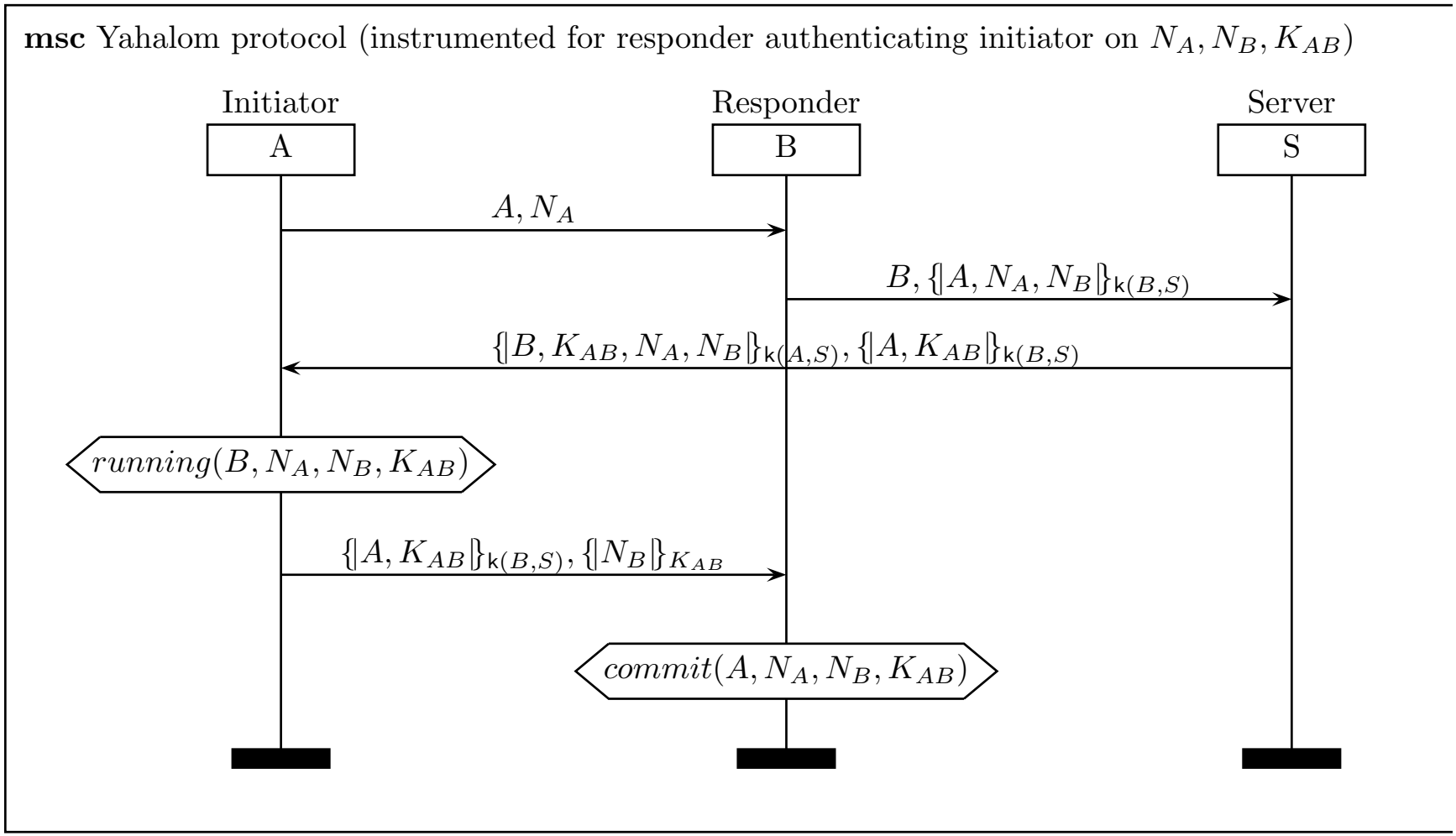


Event causality in multi-hop authentication claims: The *running* event must causally precede the *commit* event and the messages t and u must be known at the position of the claim event in the respective role.

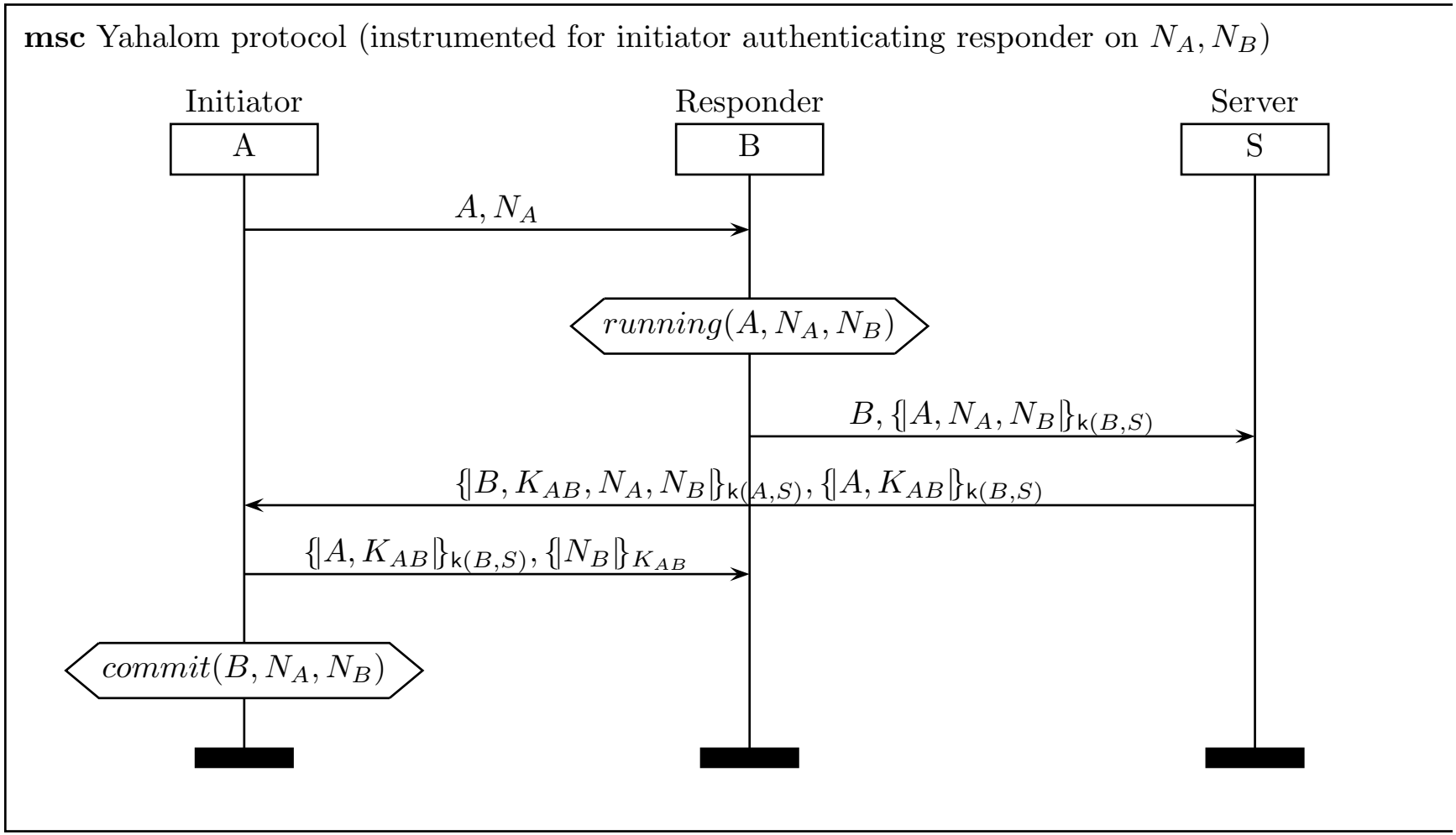
Example: Yahalom Protocol (1/3)



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)

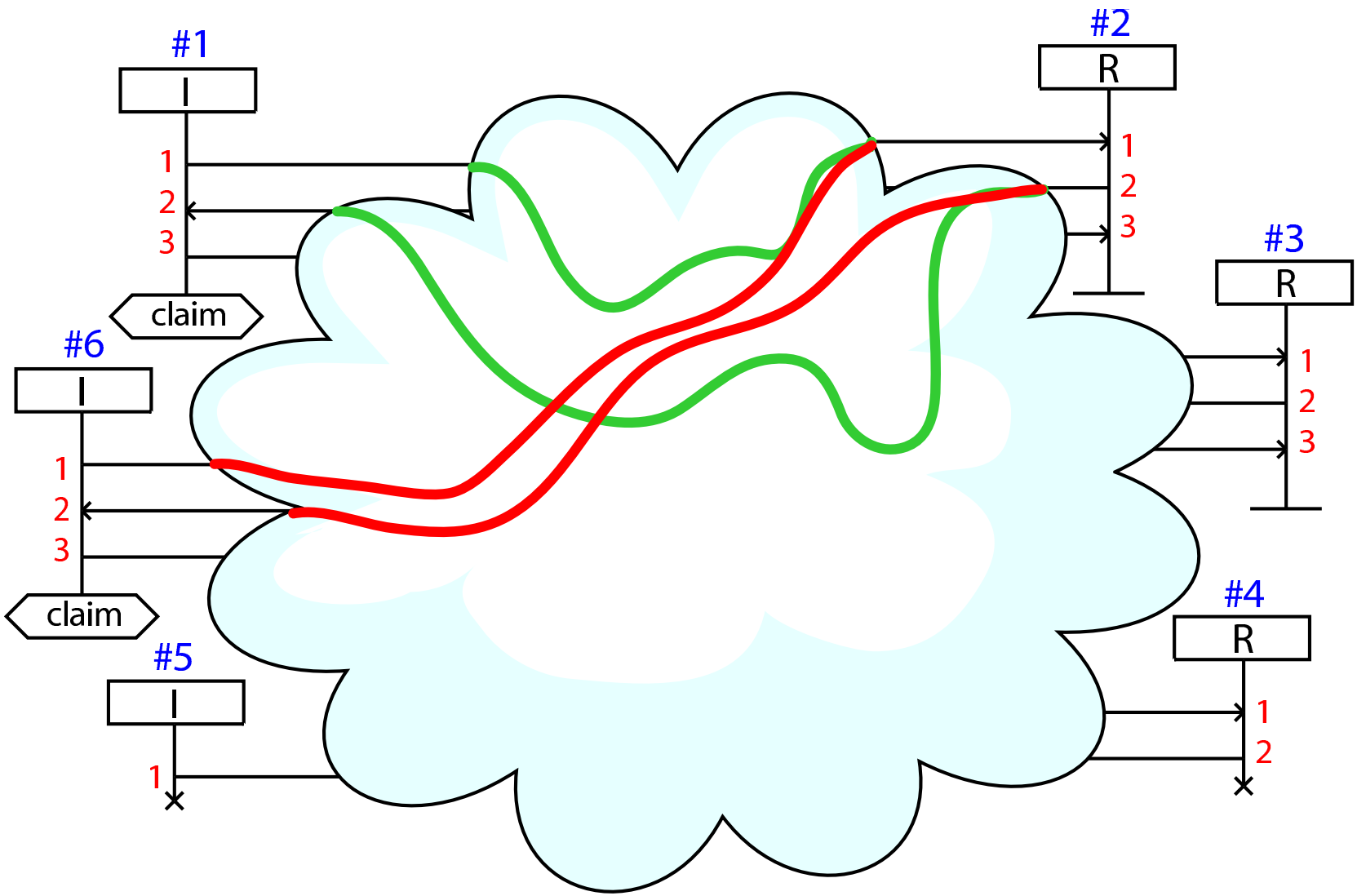
The property $Agreement(A, B, t)$ consists of all traces satisfying:

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

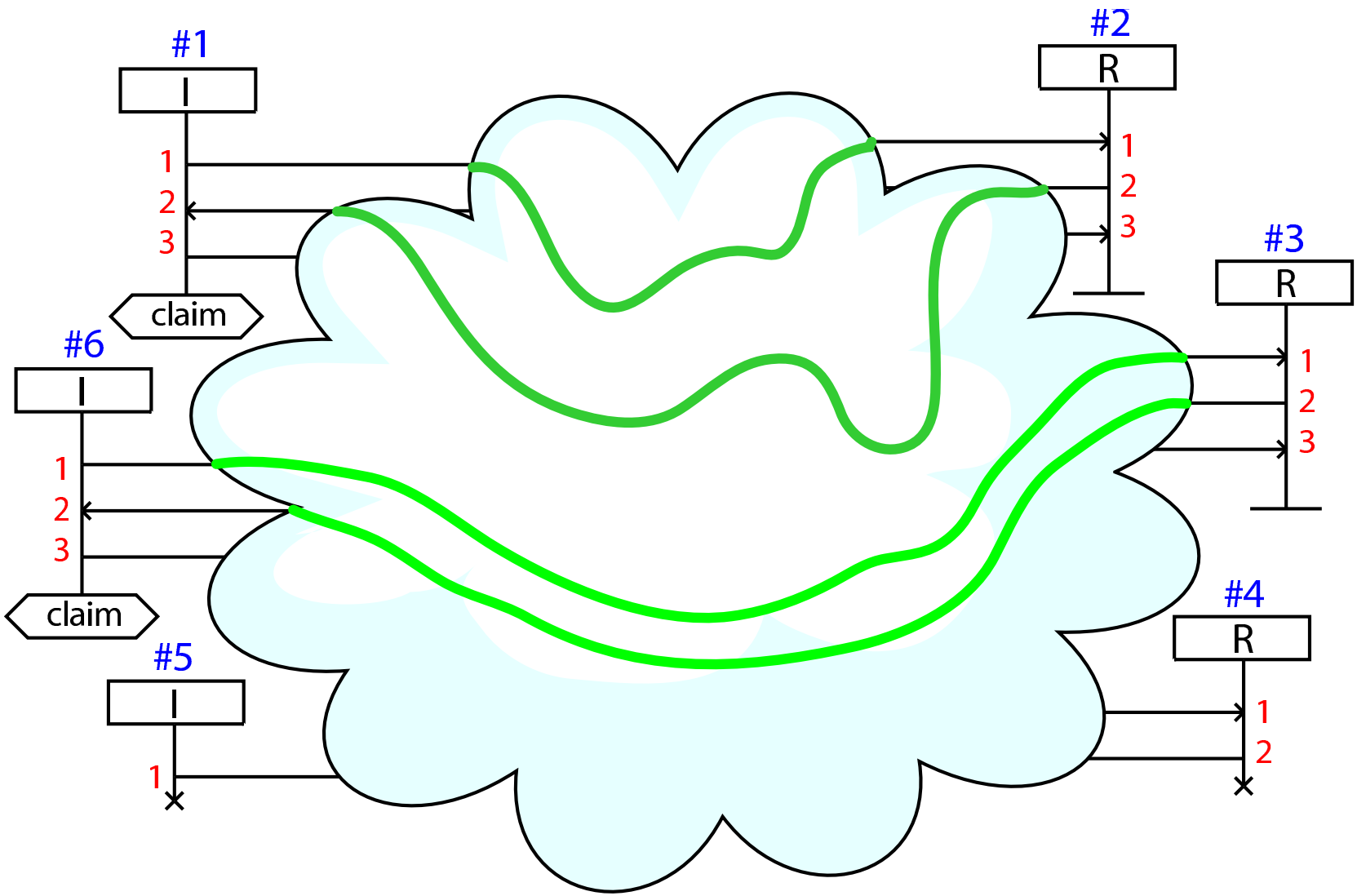
Remarks

- For each commit by a in role A on the trace there is a unique matching b executing role B.

Failed Injective Authentication

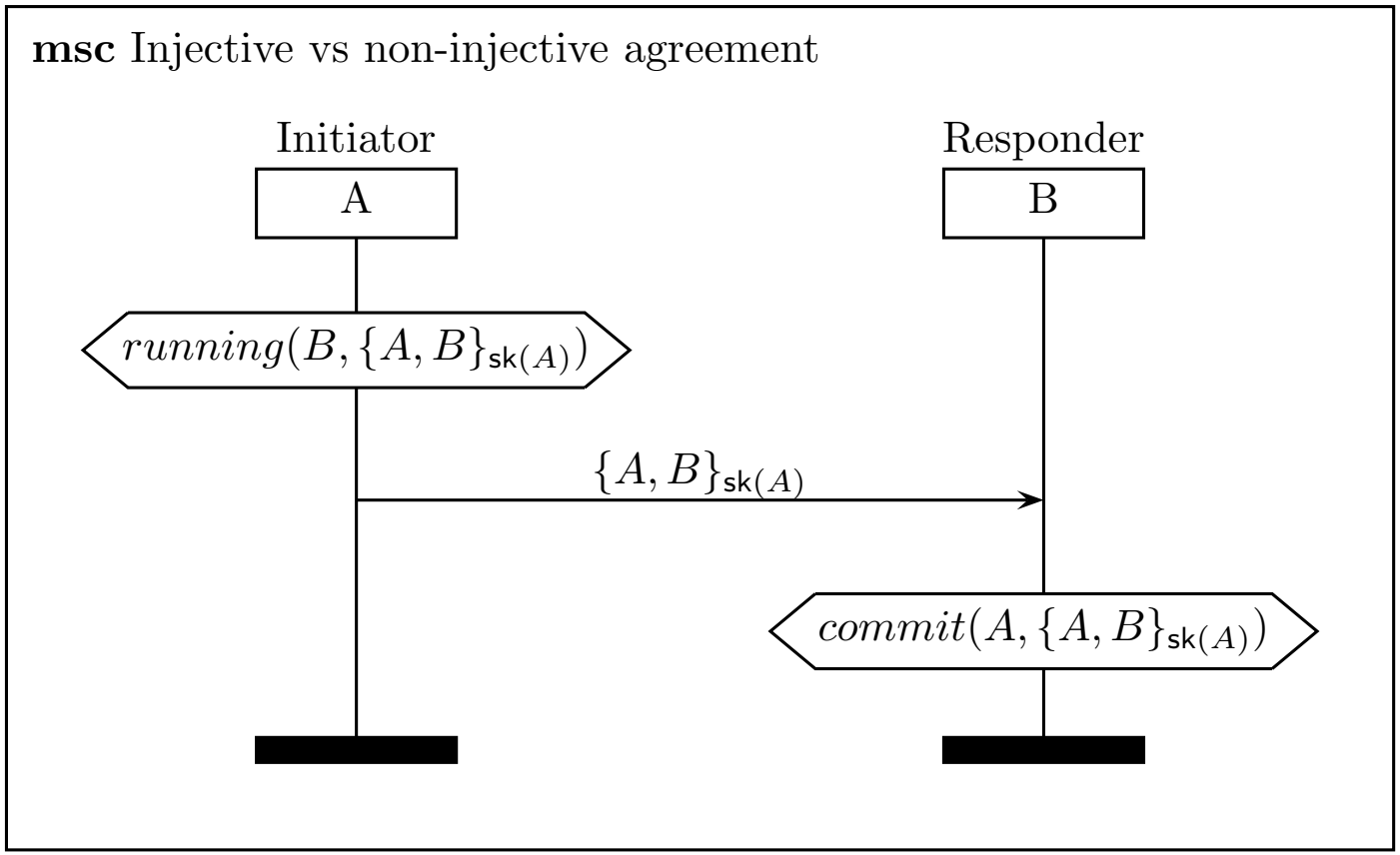


Successful Injective Authentication



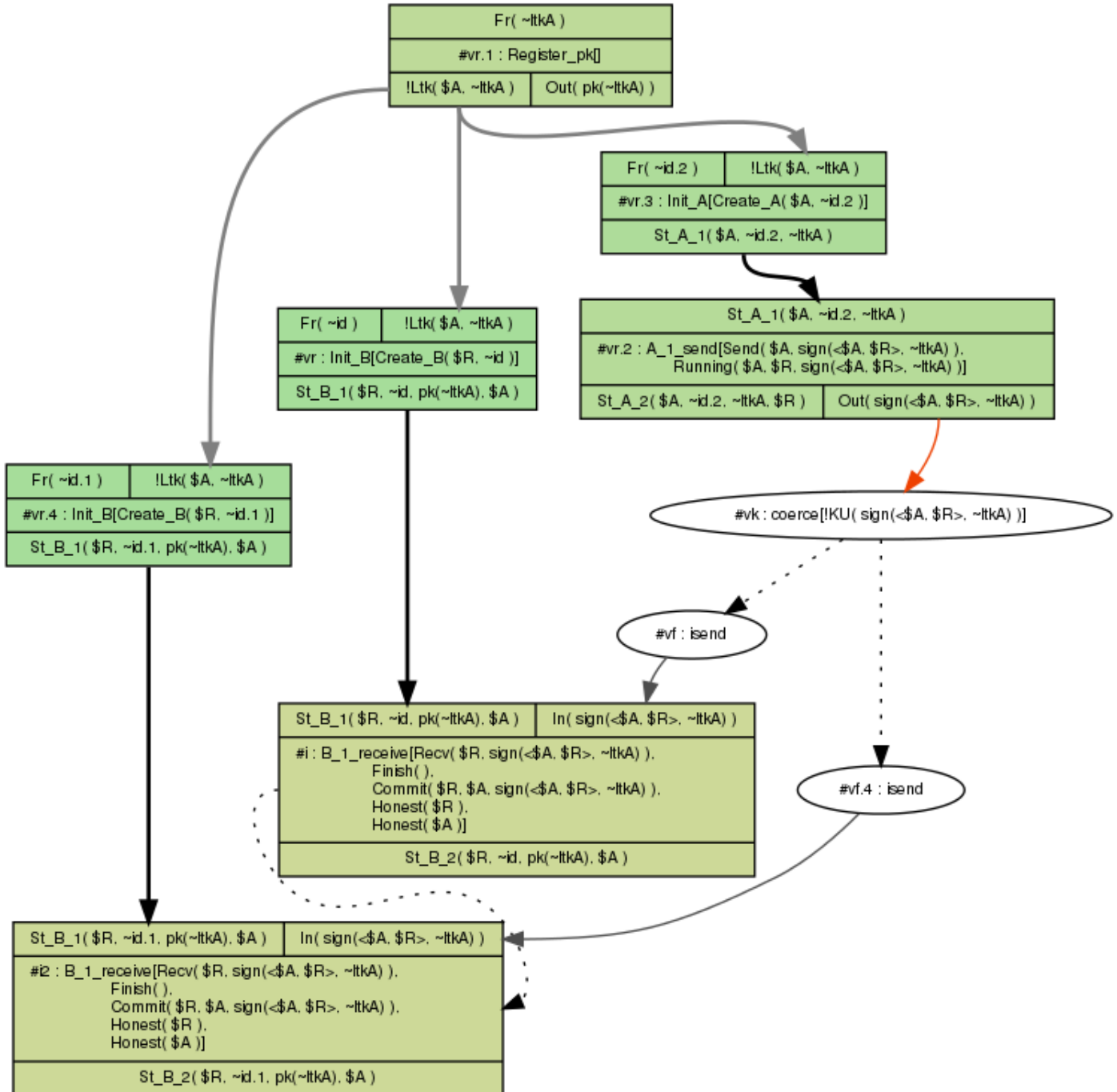
Injective vs Non-injective Agreement

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 $WeakAgreement(A, B)$ iff

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

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)

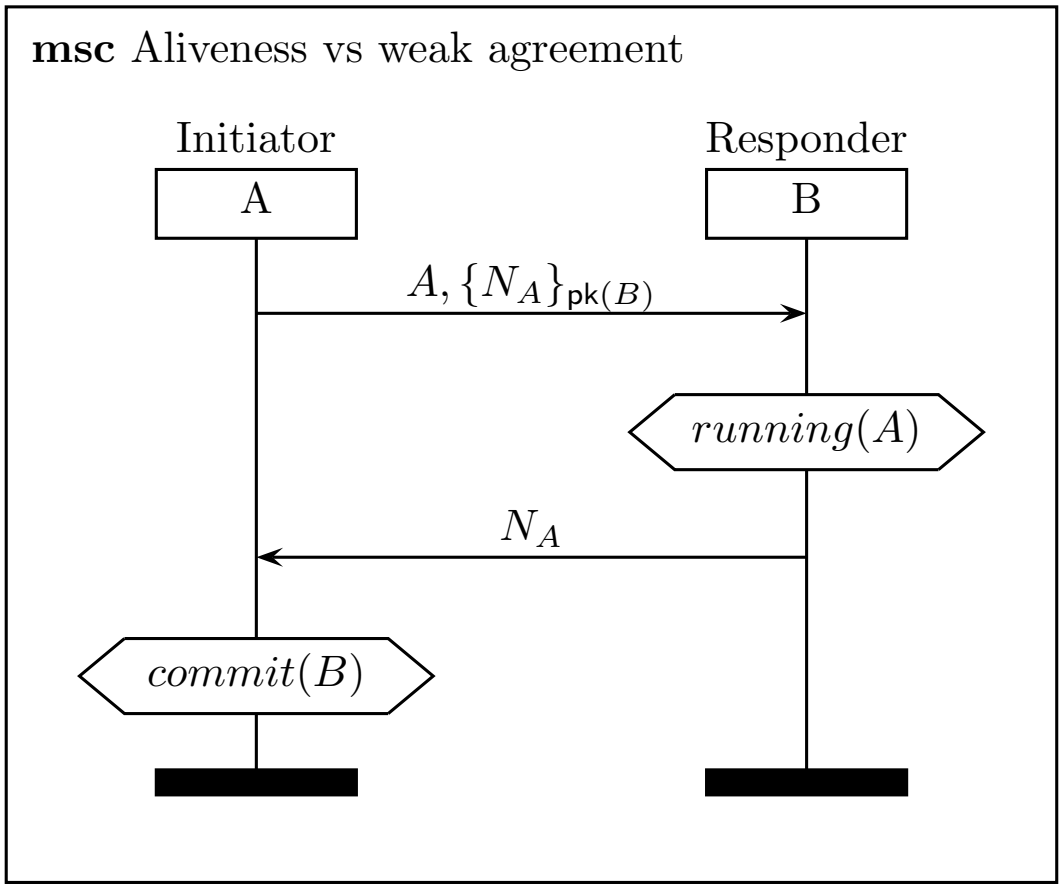
A trace tr satisfies the property $Alive(A, B)$ iff

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

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.

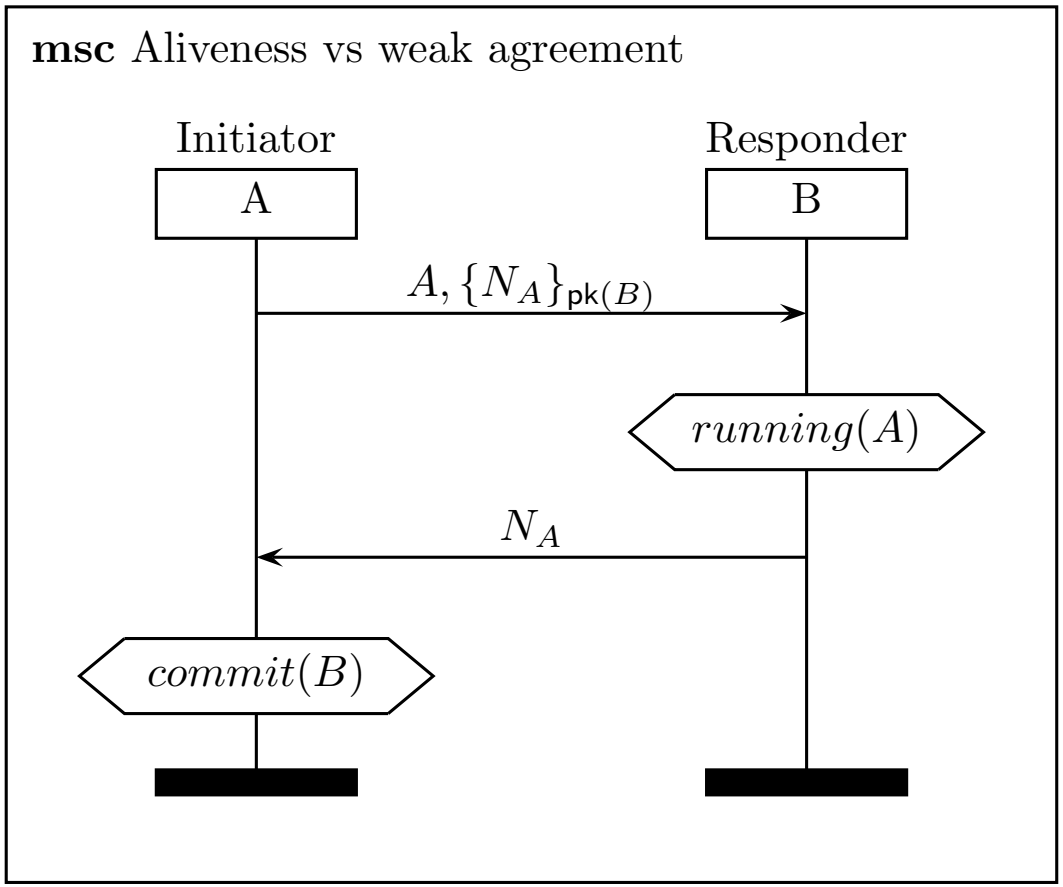
Aliveness vs Weak Agreement

Separating Example



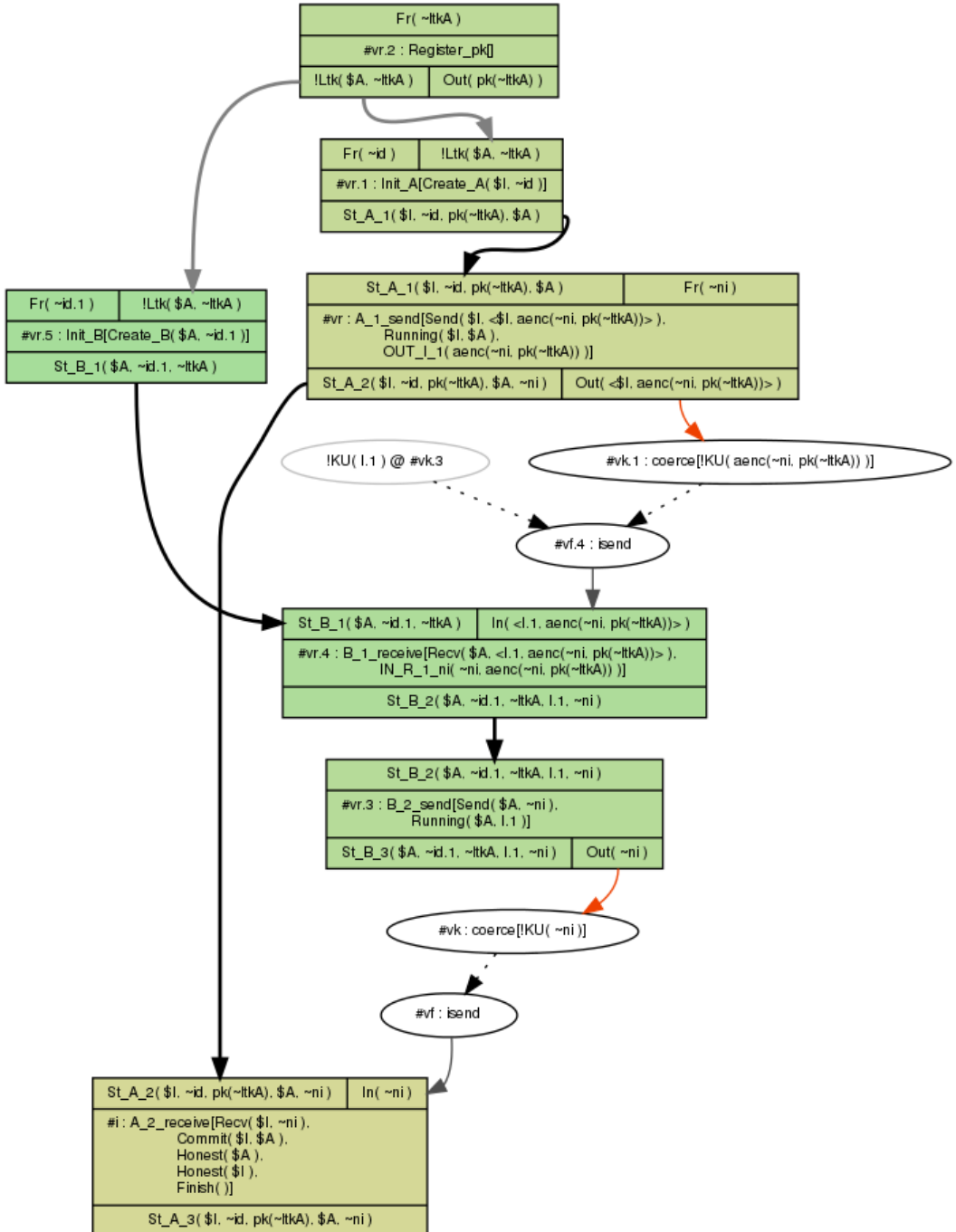
Aliveness vs Weak Agreement

Separating Example

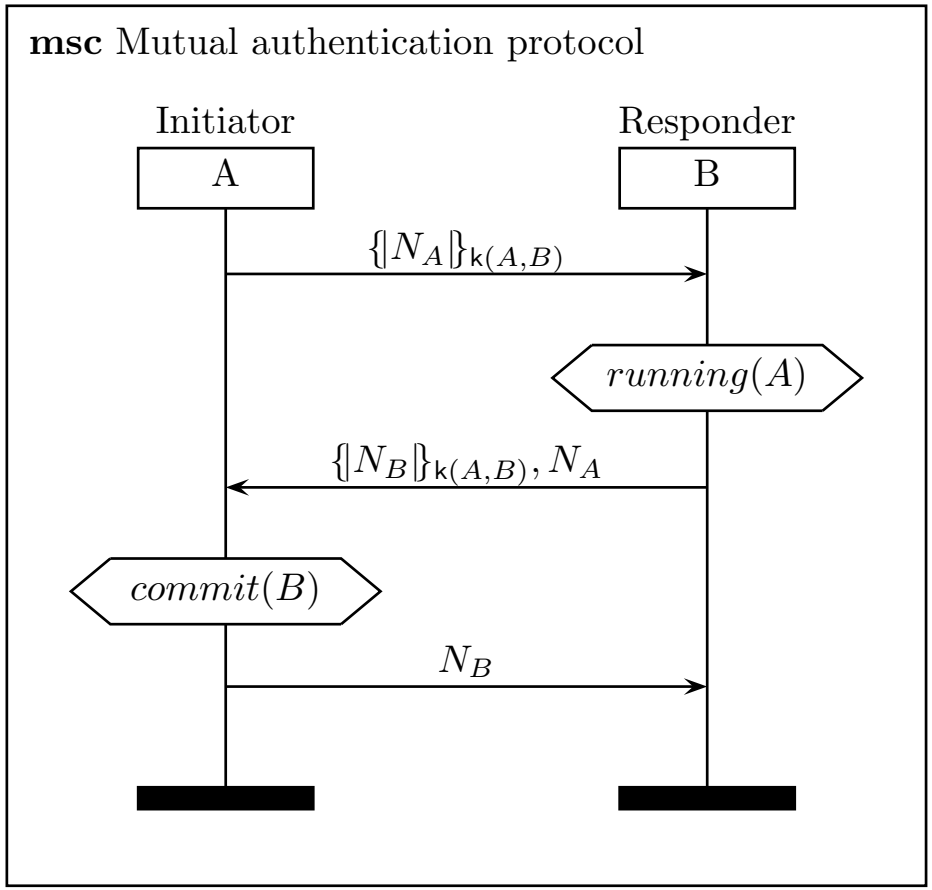


- Aliveness holds: only B can have decrypted the fresh nonce N_A.
- Weak agreement fails, since adversary may modify unprotected identity A to C in first message so that B thinks he is talking to C.

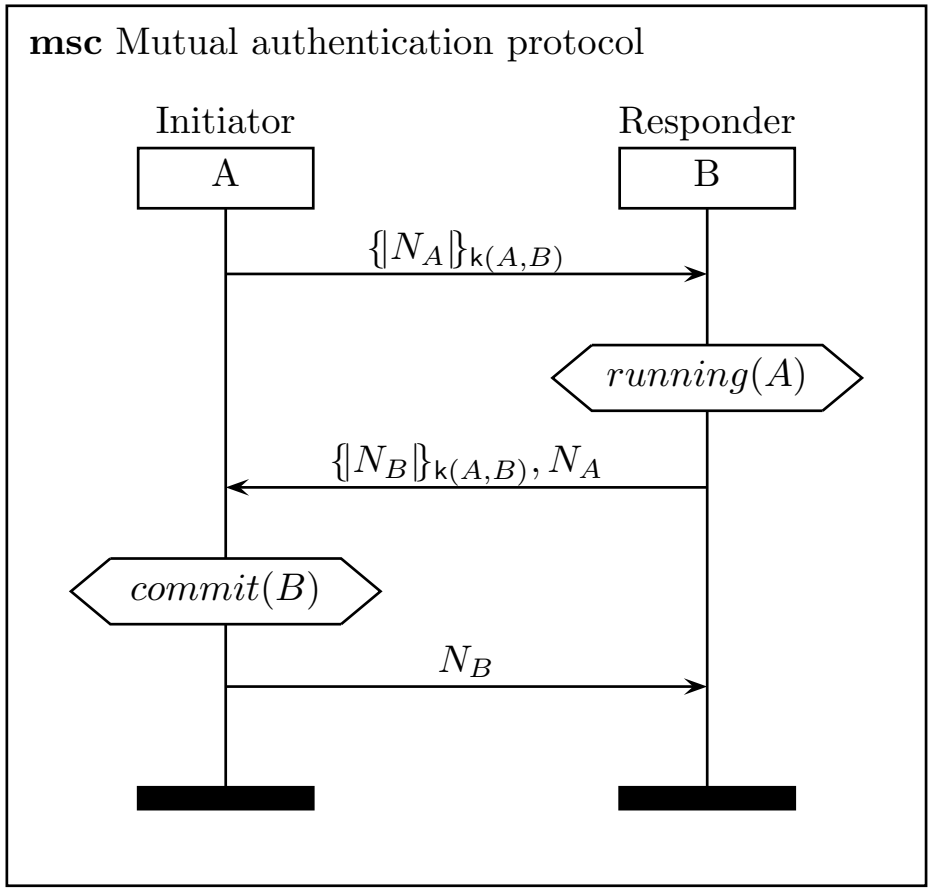
Weak Agreement counter-example



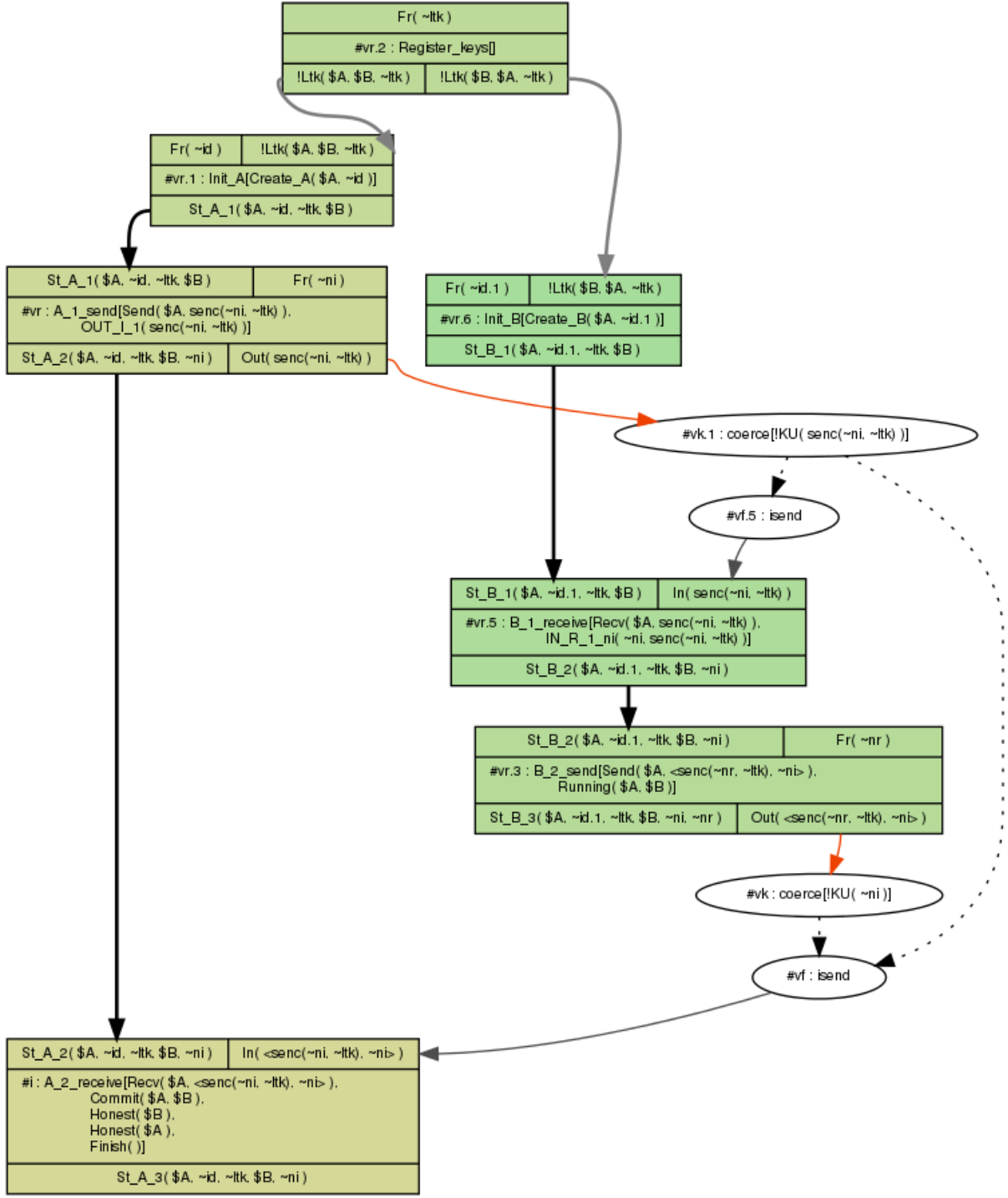
When Even Aliveness Fails ...



When Even Aliveness Fails ...



- Reflection attack: A may complete run without B's participation.
- 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_I_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_R_1_ni(~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), ~ni>).
Running($A, $B)]
St_B_3($A, ~id.1, ~ltk, $B, ~ni, ~nr)
Out(<senc(~nr, ~ltk), ~ni>)
#vk : coerce[!KU(~ni)]
#vf : isend
St_A_2($A, ~id, ~ltk, $B, ~ni)
In(<senc(~ni, ~ltk), ~ni>)
#i : A_2_receive[Recv($A, <senc(~ni, ~ltk), ~ni>).
Commit($A, $B).
Honest($B).
Honest($A).
Finish( )]
St_A_3($A, ~id, ~ltk, $B, ~ni)

Outline

1 Protocol Security Goals

2 Secrecy

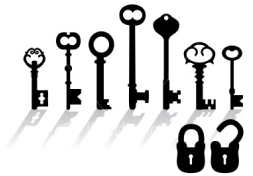
3 Authentication

4 Key-related properties

5 Automated Verification

6 Decidability

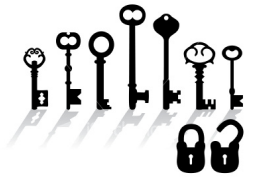
Key-related Properties



Basic key-oriented goals

- key freshness
- (implicit) key authentication: a key is only known to the communicating agents A and B and mutually trusted parties
- key confirmation of A to B is provided if B has assurance that agent A has possession of key K
- explicit key authentication = key authentication + key confirmation
  $\Rightarrow$ expressible in terms of secrecy and agreement

Key-related Properties



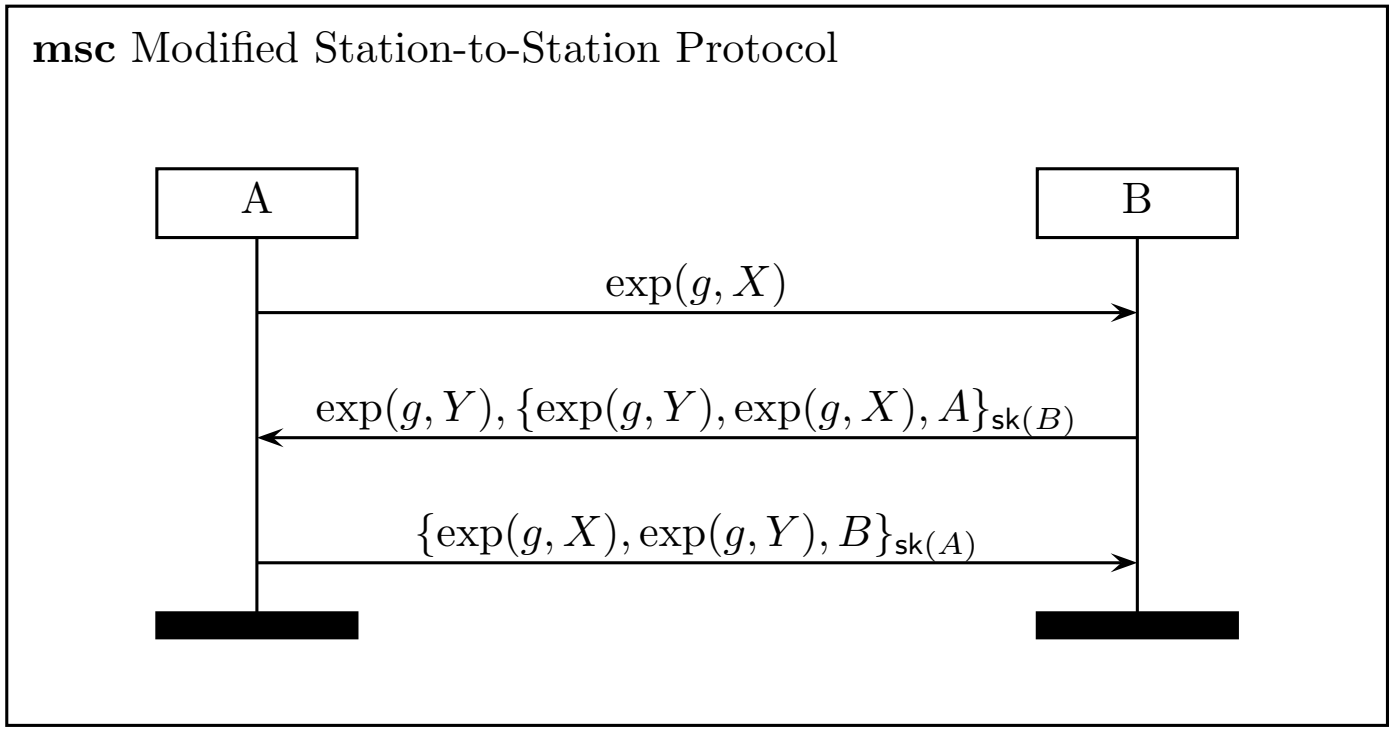
Basic key-oriented goals

- key freshness
- (implicit) key authentication: a key is only known to the communicating agents A and B and mutually trusted parties
- key confirmation of A to B is provided if B has assurance that agent A has possession of key K
- explicit key authentication = key authentication + key confirmation
  $\Rightarrow$ expressible in terms of secrecy and agreement

Goals concerning compromised keys

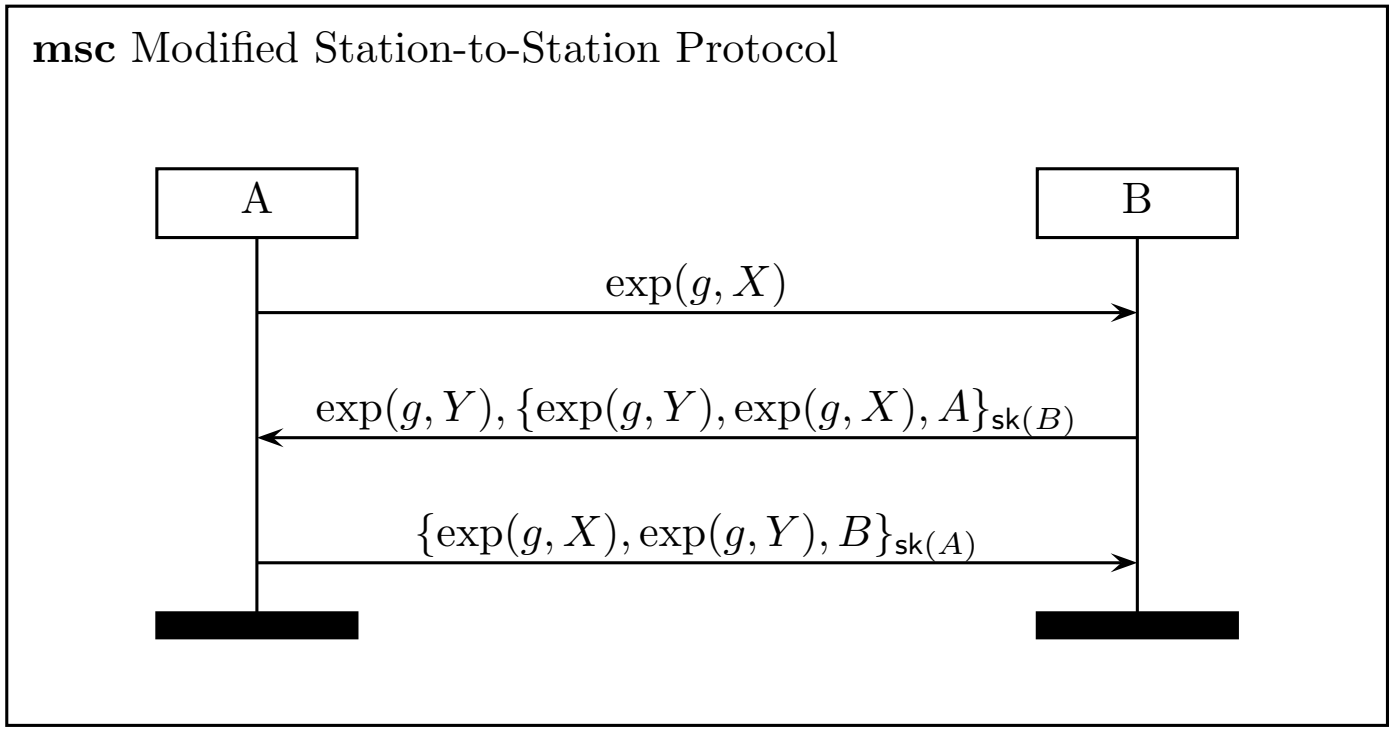
- (perfect) forward secrecy: compromise of long-term keys of a set of principals does not compromise the session keys established in previous protocol runs involving those principals
- resistance to key-compromise impersonation: compromise of long-term key of an agent A does not allow the adversary to masquerade to A as a different principal.

Forward Secrecy: Example 1



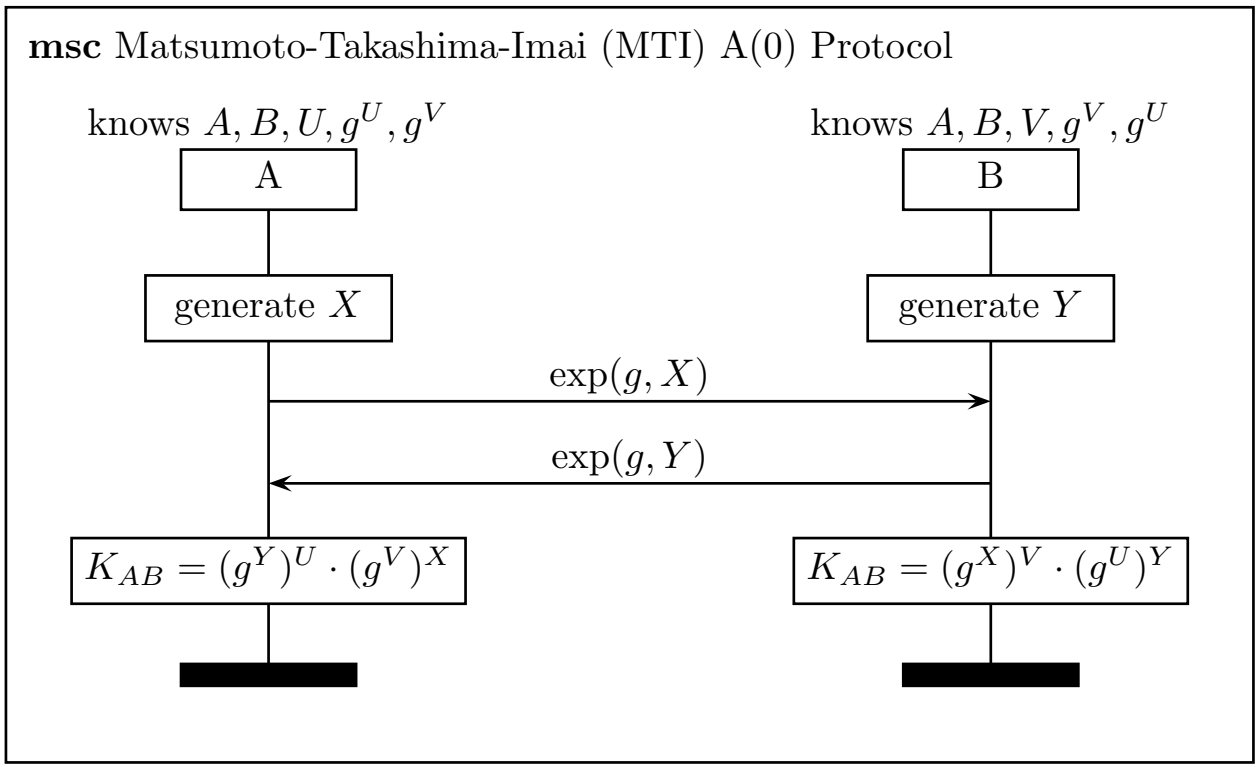
- Signatures are used to authenticate the Diffie-Hellman public keys $\exp(g, X)$ and $\exp(g, Y)$.

Forward Secrecy: Example 1



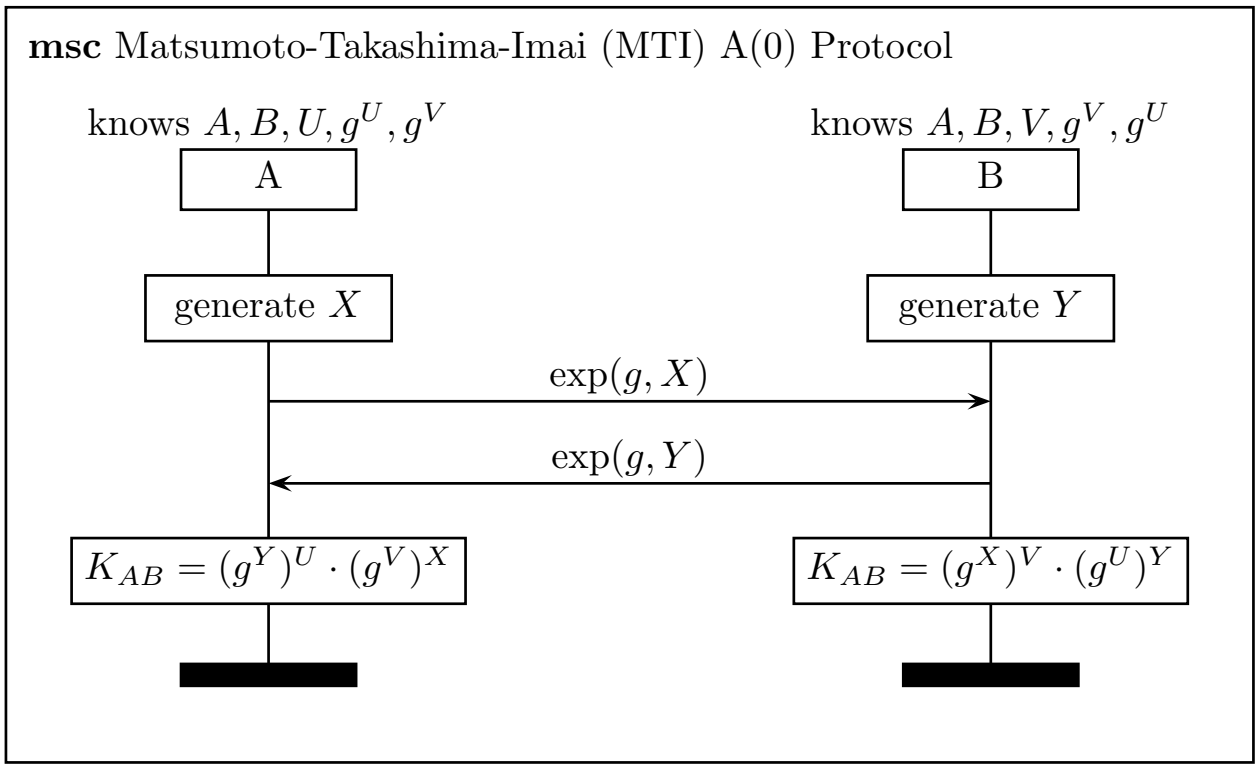
- Signatures are used to authenticate the Diffie-Hellman public keys $\exp(g, X)$ and $\exp(g, Y)$.
- Protocol provides forward secrecy: The adversary cannot derive session key $K_{AB} = \exp(\exp(g, X), Y)$ by compromise of signing keys.

Forward Secrecy: Example 2



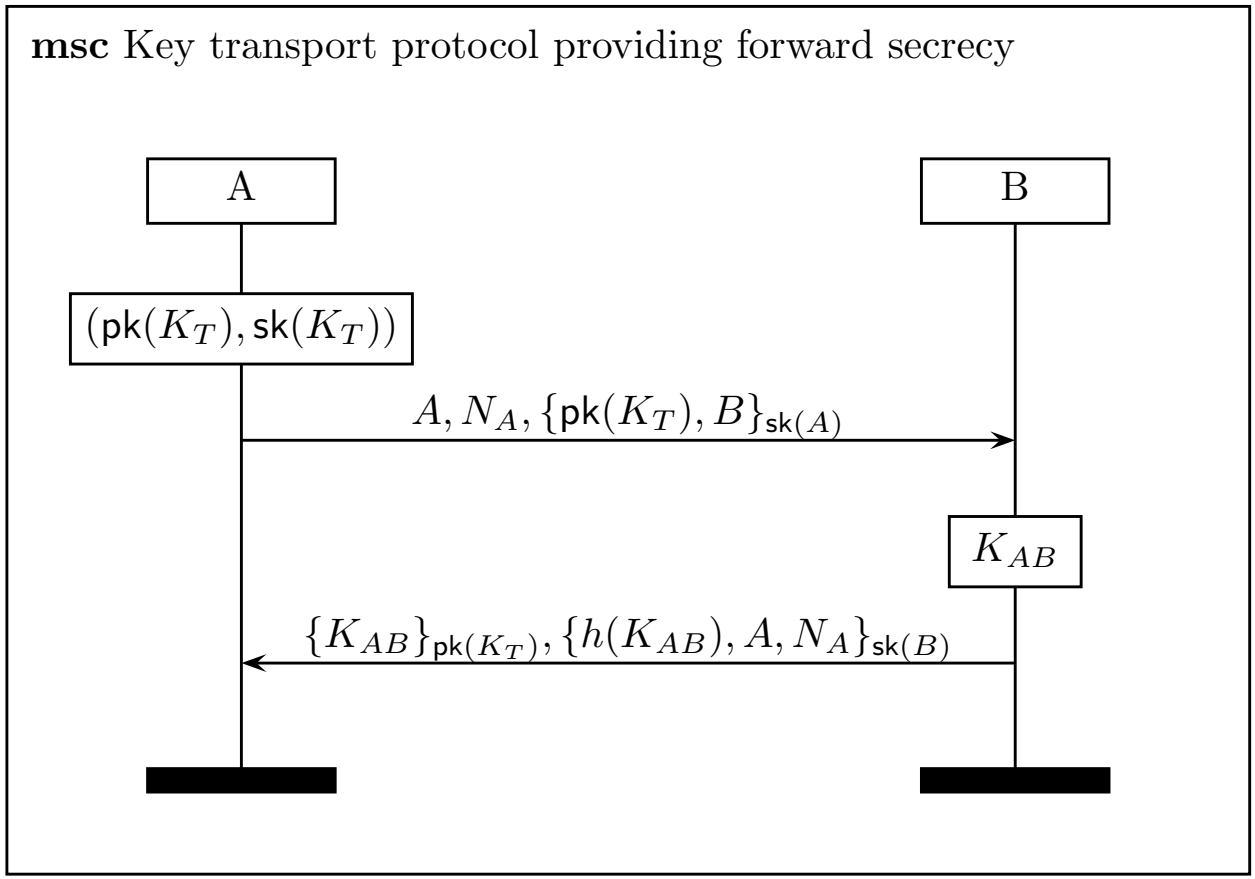
- Message exchange as in basic DH; protocol combines long-term and ephemeral DH keys to authenticate exchanged DH public keys.

Forward Secrecy: Example 2



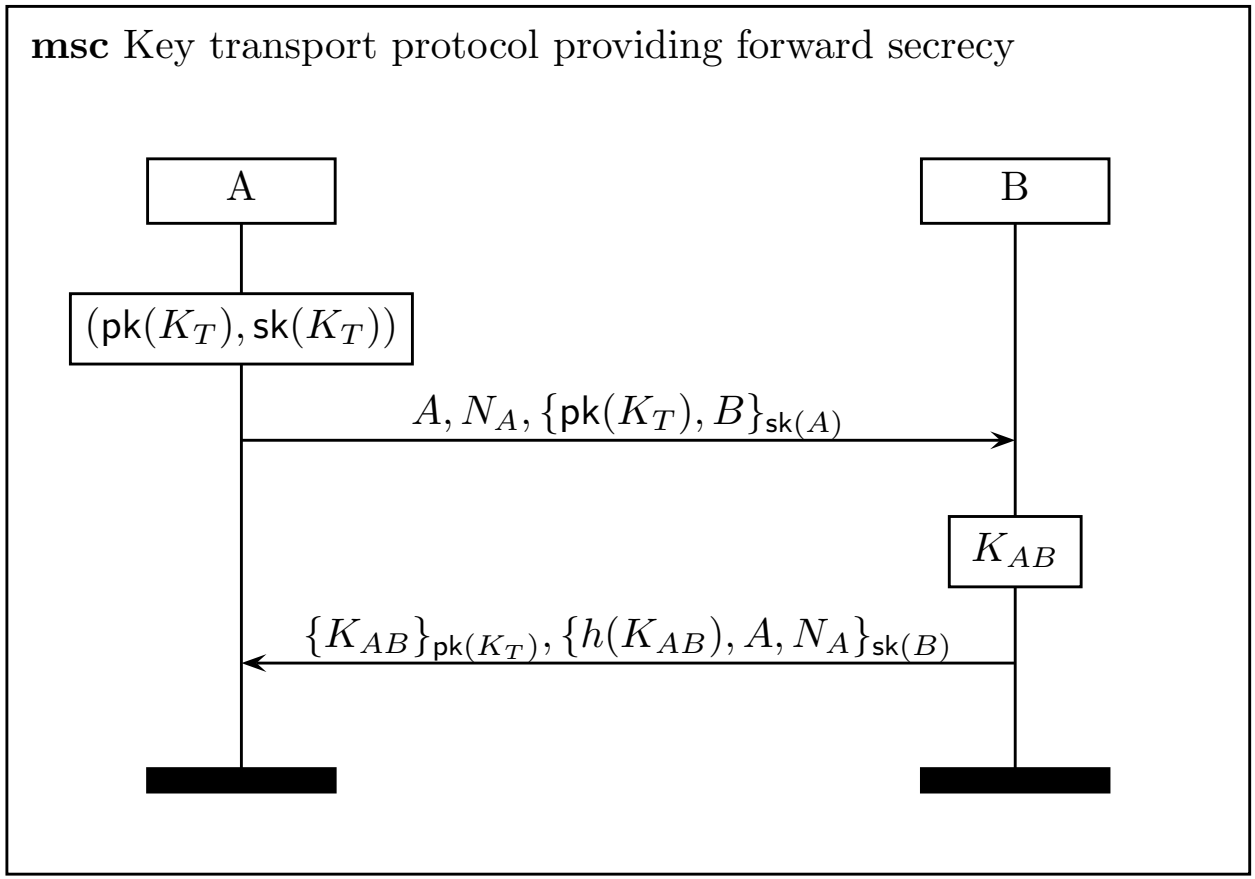
- Message exchange as in basic DH; protocol combines long-term and ephemeral DH keys to authenticate exchanged DH public keys.
- Protocol does not provide forward secrecy: The adversary can construct the session key $K_{AB} = g^{VX+UY}$ as $(g^X)^V \cdot (g^Y)^U$ from observed messages and long-term private keys U and V.

Forward Secrecy: Example 3



- A generates an ephemeral asymmetric key pair $(\mathsf{pk}(K_T), \mathsf{sk}(K_T))$.

Forward Secrecy: Example 3



- A generates an ephemeral asymmetric key pair $(\mathsf{pk}(K_T), \mathsf{sk}(K_T))$.
- Protocol provides forward secrecy without using Diffie-Hellman keys: Adversary cannot learn session key by compromise of signing keys.

Outline

1 Protocol Security Goals

2 Secrecy

3 Authentication

4 Key-related properties

5 **Automated Verification**

6 Decidability

Automated Verification and Decidability

We would like to have a program V with ...

- Input:
  - ★ some description of a program P
  - ★ some description of a functional specification S
- Output: *Yes* if P satisfies S, and *No* otherwise.
- Optional extra: in the *No* case, give a counter-example, i.e. an input on which P violates the specification.

Automated Verification and Decidability

We would like to have a program V with ...

- Input:
  - ★ some description of a program P
  - ★ some description of a functional specification S
- Output: *Yes* if P satisfies S, and *No* otherwise.
- Optional extra: in the *No* case, give a counter-example, i.e. an input on which P violates the specification.

Forget it:

Theorem (Rice)

Let S be any non-empty, proper subset of the computable functions. Then the verification problem for S (the set of programs P that compute a function in S) is undecidable.

The Sources of Infinity

For security protocols, the state space can be infinite for (at least) the following reasons:

Messages The intruder can compose arbitrarily complex messages from his knowledge, e.g., $i, h(i), h(h(i)), \ldots$.

Sessions Any number of sessions (or threads) may be executed.

Nonces Unbounded number of fresh nonces generated.

The Sources of Infinity

For security protocols, the state space can be infinite for (at least) the following reasons:

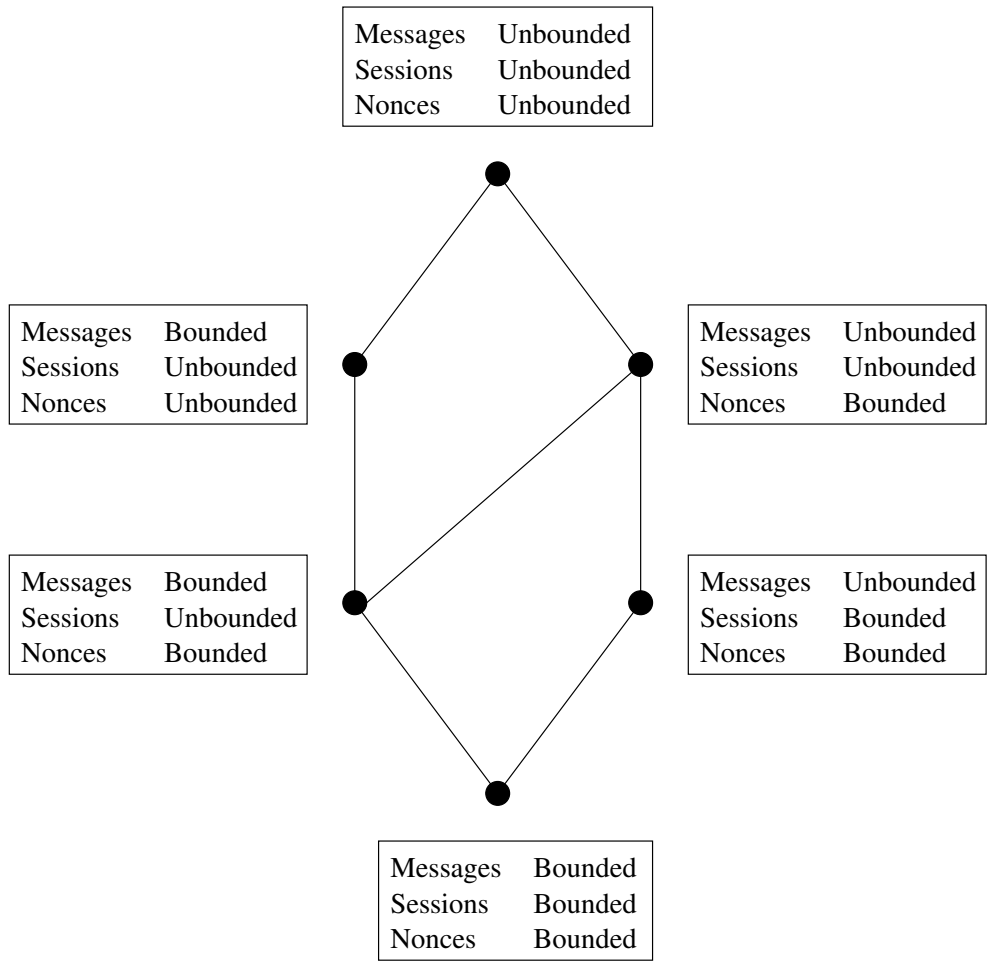
Messages The intruder can compose arbitrarily complex messages from his knowledge, e.g., $i, h(i), h(h(i)), \ldots$.

Sessions Any number of sessions (or threads) may be executed.

Nonces Unbounded number of fresh nonces generated.

NB: For finite-length threads (as considered in this lecture), we can have unbounded threads with bounded nonces, but not vice versa.

Decidability Roadmap



Outline

1 Protocol Security Goals

2 Secrecy

3 Authentication

4 Key-related properties

5 Automated Verification

6 Decidability

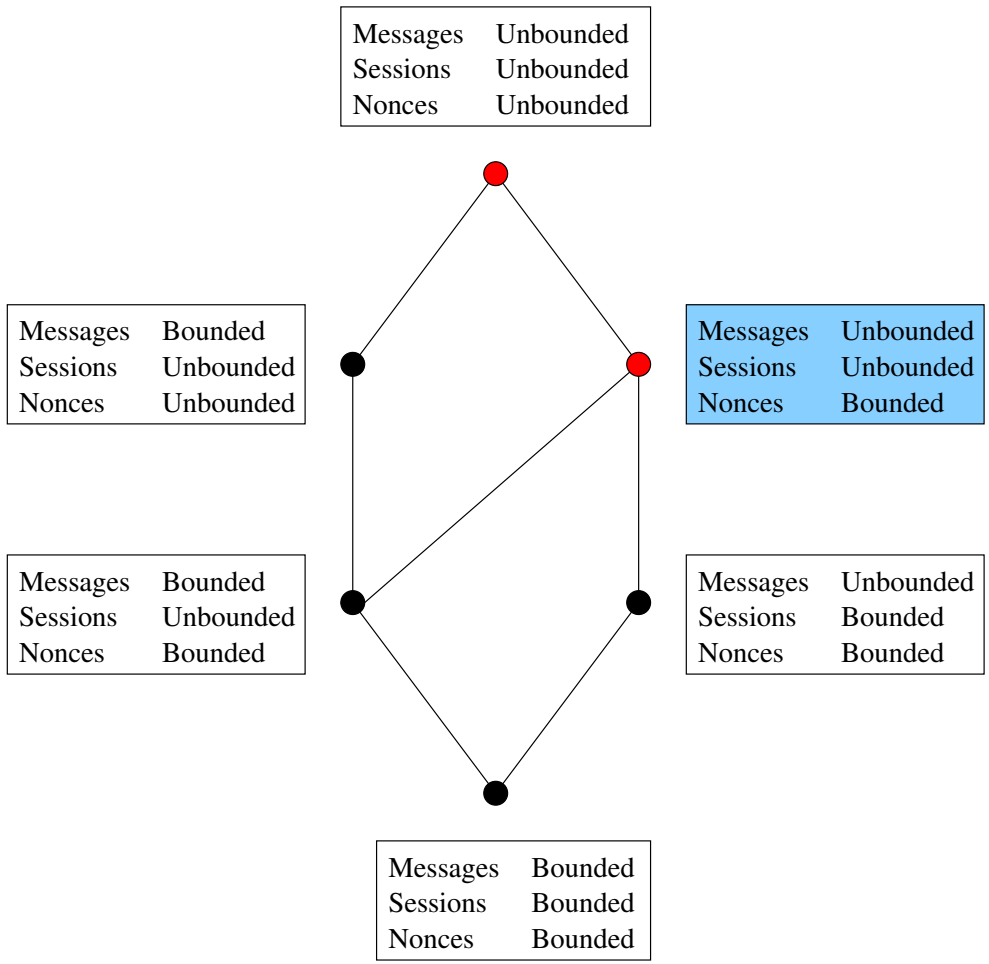
Undecidability of Secrecy

- By a reduction from PCP, first shown by [Even/Goldreich 1983].
- Basic idea of the proof: we let the intruder "guess" a solution, use the honest agents as a machine to "check" the solution, and reveal a secret if the solution is correct.
- As an attack on secrecy is equivalent to existence of a solution, we can use the protocol verifier to solve the PCP problem.

Undecidability of Secrecy

- By a reduction from PCP, first shown by [Even/Goldreich 1983].
- Basic idea of the proof: we let the intruder "guess" a solution, use the honest agents as a machine to "check" the solution, and reveal a secret if the solution is correct.
- As an attack on secrecy is equivalent to existence of a solution, we can use the protocol verifier to solve the PCP problem.
- However, the protocols generated for PCP are very artificial:
  - ★ They are not even executable without an intruder.
  - ★ [Comon et al. 2001]: even when restricting to executable protocols, secrecy is undecidable.
- The proof requires unbounded messages and unbounded threads (as the length of the solution and its check cannot be bounded).

Decidability Roadmap



- Secrecy is undecidable for UUB case [Even/Goldreich 1983]
- Hence also for UUU case.

Bounding

Reduction to PCP required unbounded threads and message size.

Realistic assumptions?

Maybe real-world attacks can be modeled without unbounded threads and message sizes.

Bound	Justification
Number of threads Term size	

Bounding

Reduction to PCP required unbounded threads and message size.

Realistic assumptions?

Maybe real-world attacks can be modeled without unbounded threads and message sizes.

Bound	Justification
Number of threads	*Usually* attacks don't involve 100 threads
Term size	

Bounding

Reduction to PCP required unbounded threads and message size.

Realistic assumptions?

Maybe real-world attacks can be modeled without unbounded threads and message sizes.

Bound	Justification
Number of threads	*Usually* attacks don't involve 100 threads
Term size	Protocol checks type of incoming messages

Bounding

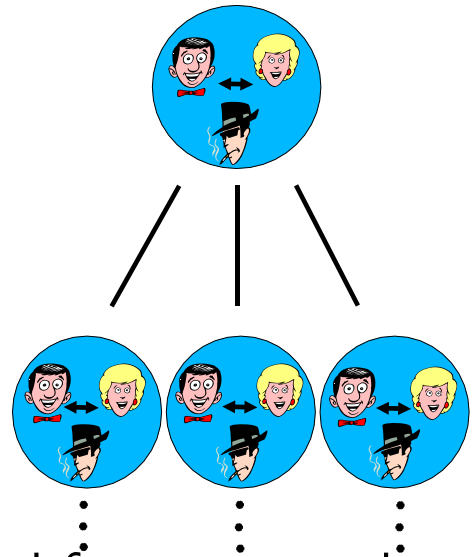
Reduction to PCP required unbounded threads and message size.

Realistic assumptions?

Maybe real-world attacks can be modeled without unbounded threads and message sizes.

Bound	Justification
Number of threads	*Usually* attacks don't involve 100 threads
Term size	Protocol checks type of incoming messages - What about $h(h(h(h(A))))$?

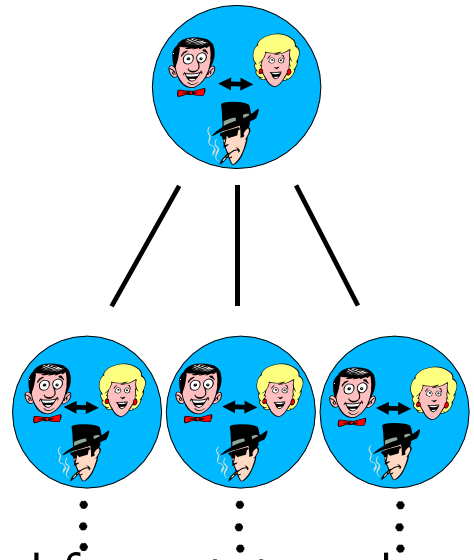
Search Tree



We can consider a search tree, where

- each node is a state,
- the root node is the initial state,
- node n is a child of node m iff state n can be reached from state m by one transition justified by a rule in our labeled transition system.
- we can check for the trace leading to each state/node whether it violates our secrecy or authentication goals.
- we can use the standard search techniques to browse that tree, e.g., depth first, breadth first, iterative deepening.

Search Tree



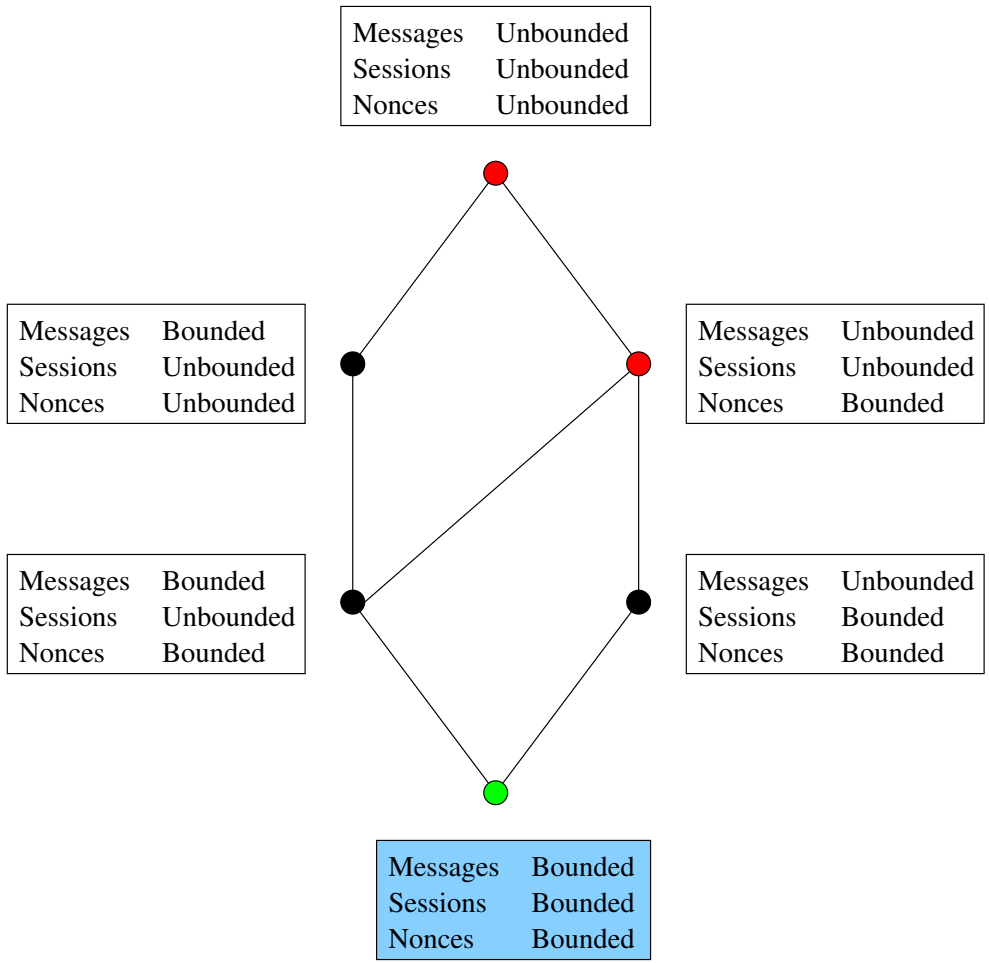
We can consider a search tree, where

- each node is a state,
- the root node is the initial state,
- node n is a child of node m iff state n can be reached from state m by one transition justified by a rule in our labeled transition system.
- we can check for the trace leading to each state/node whether it violates our secrecy or authentication goals.
- we can use the standard search techniques to browse that tree, e.g., depth first, breadth first, iterative deepening.

Exercise: formalize and prove:

- When bounding everything, this yields a decision procedure.
- Otherwise, we can give a semi-decision procedure, i.e., one that is guaranteed to terminate with an attack if there is one.

Decidability Roadmap



- Bounding everything trivially yields decidability

Lazy Intruder: Summary

- The constraint reduction produces finitely many simple constraints by a terminating algorithm.
- If the number of threads is bounded, we now have a decision procedure even without bounding the messages:

Theorem (Rusinowitch & Turuani 2001)

Protocol insecurity for a bounded number of threads is NP-complete.

Lazy Intruder: Summary

- The constraint reduction produces finitely many simple constraints by a terminating algorithm.
- If the number of threads is bounded, we now have a decision procedure even without bounding the messages:

Theorem (Rusinowitch & Turuani 2001)

Protocol insecurity for a bounded number of threads is NP-complete.

Proof Sketch.

In NP: Guess a symbolic trace *tr* with no more than a given number of threads and a sequence of reduction steps for the resulting constraints. Check whether *tr* is an attack trace. All these steps can be polynomially bounded.

NP-hard: Polynomial reduction from boolean formula satisfaction (3-SAT) such that formula satisfiable iff protocol has an attack.

□

Lazy Intruder: Summary

- The constraint reduction produces finitely many simple constraints by a terminating algorithm.
- If the number of threads is bounded, we now have a decision procedure even without bounding the messages:

Theorem (Rusinowitch & Turuani 2001)

Protocol insecurity for a bounded number of threads is NP-complete.

Proof Sketch.

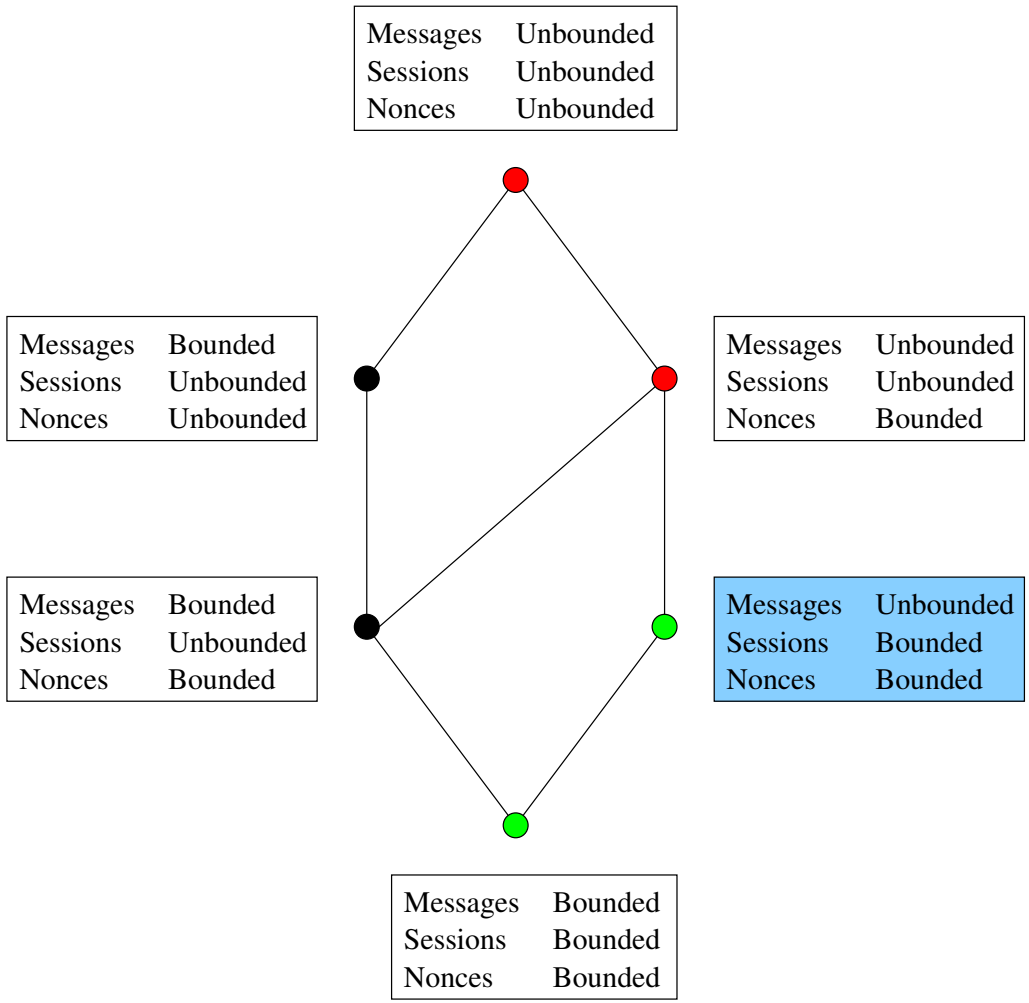
In NP: Guess a symbolic trace *tr* with no more than a given number of threads and a sequence of reduction steps for the resulting constraints. Check whether *tr* is an attack trace. All these steps can be polynomially bounded.

NP-hard: Polynomial reduction from boolean formula satisfaction (3-SAT) such that formula satisfiable iff protocol has an attack.

□

It follows that protocol security for a bounded number of threads is decidable.

Roadmap


Messages Unbounded
Sessions Unbounded
Nonces Unbounded
Messages Bounded
Sessions Unbounded
Nonces Unbounded
Messages Unbounded
Sessions Unbounded
Nonces Bounded
Messages Bounded
Sessions Unbounded
Nonces Bounded
Messages Unbounded
Sessions Bounded
Nonces Bounded
Messages Bounded
Sessions Bounded
Nonces Bounded

To complete the picture

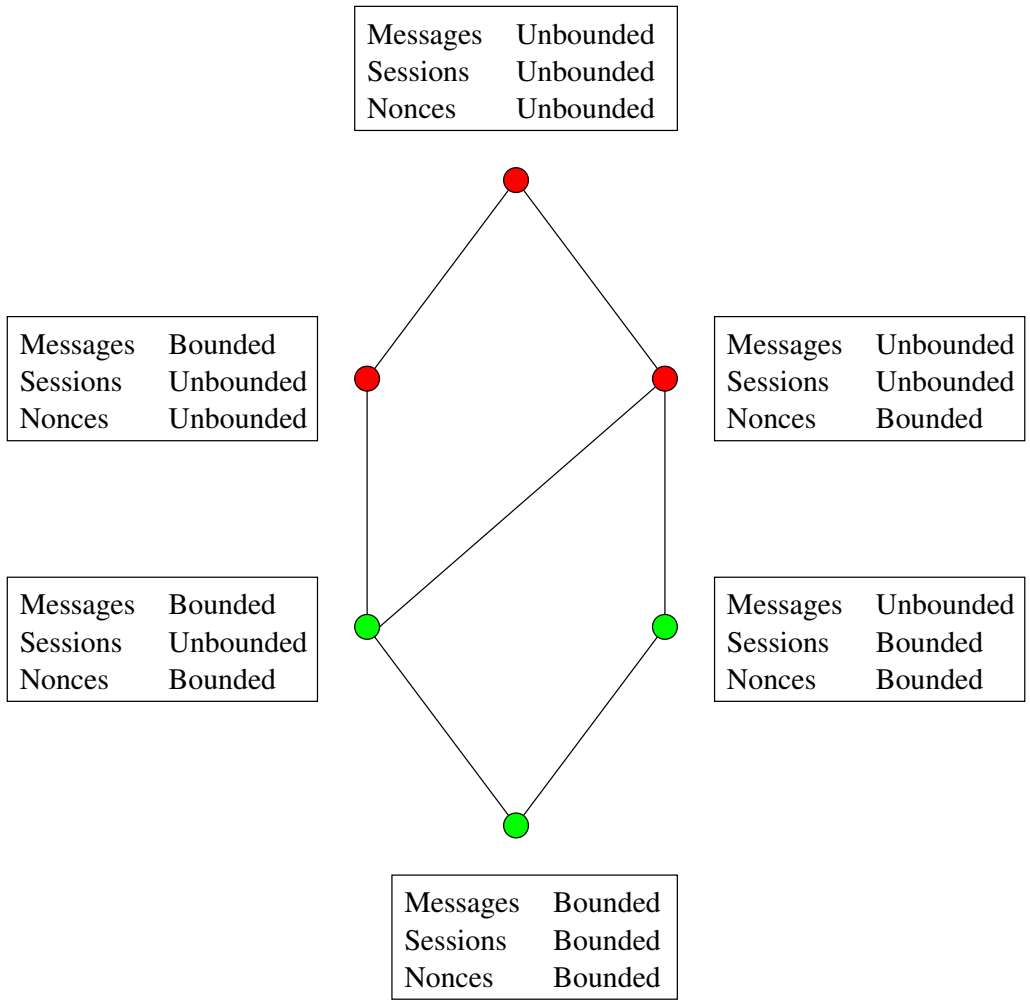
Theorem (Durgin et al., 1999)

For an unbounded number of threads and an unbounded number of nonces, protocol security is undecidable, even when bounding messages.

Theorem (Durgin et al., 1999)

For bounded messages and a bounded number of nonces, protocol security is DEXPTIME-complete.

(Un)decidability: Complete picture



Bottom line: need at least two bounded parameters for decidability.

Tamarin overview

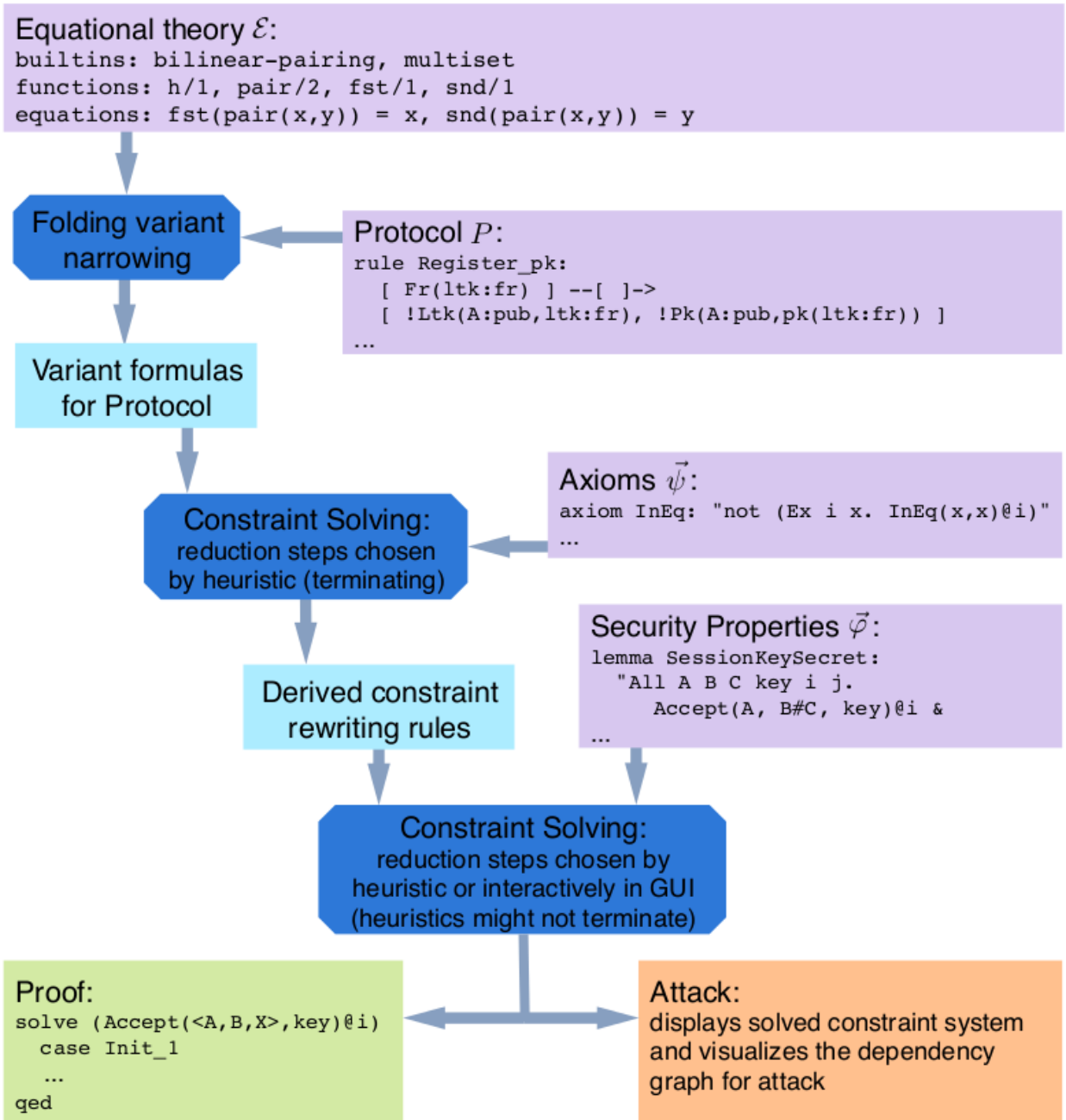
- Uses multiset rewriting to represent protocol
- Adversary message deduction rules given as multiset rewriting rules
- Properties specified in first-order logic
  - ★ Allows quantification over messages and timepoints
- Algorithm is proven sound and complete

Tamarin overview

- Uses multiset rewriting to represent protocol
- Adversary message deduction rules given as multiset rewriting rules
- Properties specified in first-order logic
  - ★ Allows quantification over messages and timepoints
- Algorithm is proven sound and complete

- Backwards reachability analysis – searching for insecure states
  - ★ Negate security property, search for *solutions*
- Constraint solving
  - ★ Uses dependency graphs
  - ★ Normal dependency graphs for state-space reduction – efficiency

Tamarin workflow

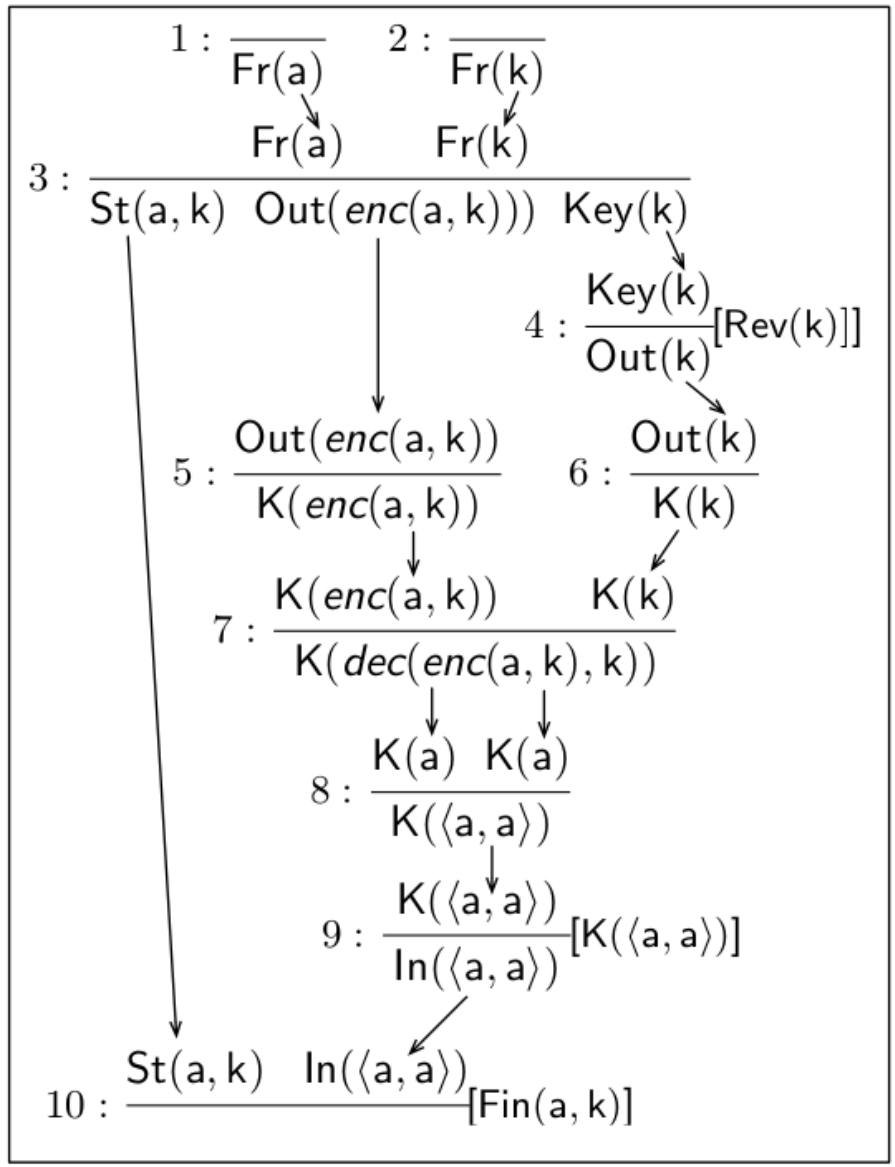


Tamarin's constraint solving algorithm

1: **function** SOLVE($P \vDash_{E_{DH}} \varphi$)
2: $\hat{\varphi} \leftarrow \neg\varphi$ rewritten into negation normal form
3: $\mathbf{\Omega} \leftarrow \{\{\hat{\varphi}\}\}$
4: **while** $\mathbf{\Omega} \neq \emptyset$ and $solved(\mathbf{\Omega}) = \emptyset$ **do**
5: choose $\Gamma \rightsquigarrow_P \{\Gamma_1, \ldots, \Gamma_k\}$ such that $\Gamma \in \mathbf{\Omega}$
6: $\mathbf{\Omega} \leftarrow (\mathbf{\Omega} \setminus \{\Gamma\}) \cup \{\Gamma_1, \ldots, \Gamma_k\}$
7: **if** $solved(\mathbf{\Omega}) \neq \emptyset$
8: **then return** "attack(s) found: ", $solved(\mathbf{\Omega})$
9: **else return** "verification successful"

Dependency graph example

$$P = \{\ [\mathsf{Fr}(x), \mathsf{Fr}(k)] -[]\!\!\rightarrow [\mathsf{St}(x,k), \mathsf{Out}(\mathrm{enc}(x,k)), \mathsf{Key}(k)]$$
$$,\ [\mathsf{St}(x,k), \mathsf{In}(\langle x, x\rangle)] -[\ \mathsf{Fin}(x,k)\ ]\!\!\rightarrow []$$
$$,\ [\mathsf{Key}(k)] -[\ \mathsf{Rev}(k)\ ]\!\!\rightarrow [\mathsf{Out}(k)]\ \}\ .$$



Adversary rules

Coerce rule: $\textsc{Coerce}\ \dfrac{\mathsf{K}^{\downarrow}_{e}(x)}{\mathsf{K}^{\uparrow}_{e}(x)}$ *Communication rules*: $\textsc{IRecv}\ \dfrac{\mathsf{Out}(x)}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)}$ $\textsc{ISend}\ \dfrac{\mathsf{K}^{\uparrow}_{e}(x)}{\mathsf{In}(x)}[\mathsf{K}(x)]$

Construction rules:

$$\frac{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(x)\ \ \mathsf{K}^{\uparrow}_{e}(y)}{\mathsf{K}^{\uparrow}_{\mathsf{noexp}}(x \hat{}\ y)} \qquad \frac{}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(x{:}\mathit{pub})} \qquad \frac{\mathsf{Fr}(x{:}\mathit{fresh})}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(x{:}\mathit{fresh})} \qquad \frac{\mathsf{K}^{\uparrow}_{e}(x)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(x^{-1})} \qquad \frac{}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(1)} \qquad \frac{\mathsf{K}^{\uparrow}_{e_1}(x)\ \ \mathsf{K}^{\uparrow}_{e_2}(y)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathrm{enc}(x,y))} \qquad \frac{\mathsf{K}^{\uparrow}_{e_1}(x)\ \ \mathsf{K}^{\uparrow}_{e_2}(y)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathrm{dec}(x,y))}$$

$$\frac{\mathsf{K}^{\uparrow}_{e}(x)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathrm{h}(x))} \qquad \frac{\mathsf{K}^{\uparrow}_{e}(x)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathrm{fst}(x))} \qquad \frac{\mathsf{K}^{\uparrow}_{e}(x)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\mathrm{snd}(x))} \qquad \frac{\mathsf{K}^{\uparrow}_{e_1}(x)\ \ \mathsf{K}^{\uparrow}_{e_2}(y)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}(\langle x,y\rangle)} \qquad \frac{\mathsf{K}^{\uparrow}_{e_1}(x_1)\ \ldots\ \mathsf{K}^{\uparrow}_{e_n}(x_n) \qquad \mathsf{K}^{\uparrow}_{e_{n+1}}(x_{n+1})\ \ldots\ \mathsf{K}^{\uparrow}_{e_l}(x_l)}{\mathsf{K}^{\uparrow}_{\mathsf{exp}}((x_1 * \ldots * x_n) * (x_{n+1} * \ldots * x_l)^{-1})}$$

Deconstruction rules:

$$\frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x \hat{}\ y)\ \ \mathsf{K}^{\uparrow}_{e}(y^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x)} \qquad \frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x \hat{}\ y^{-1})\ \ \mathsf{K}^{\uparrow}_{e}(y)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x)} \qquad \frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x \hat{}\ (y * z^{-1}))\ \ \mathsf{K}^{\uparrow}_{e}(y^{-1} * z)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x)}$$

$$\frac{\mathsf{K}^{\downarrow}_{e}(\langle x,y\rangle)}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)} \qquad \frac{\mathsf{K}^{\downarrow}_{e}(\langle x,y\rangle)}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(y)} \qquad \frac{\mathsf{K}^{\downarrow}_{e}(x^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)} \qquad \frac{\mathsf{K}^{\downarrow}_{e_1}(\mathrm{enc}(x,y))\ \ \mathsf{K}^{\uparrow}_{e_2}(y)}{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x)}$$

Exponentiation rules:

$$\frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x \hat{}\ y)\ \ \mathsf{K}^{\uparrow}_{e}(z)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x \hat{}\ (y * z))} \qquad \frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x \hat{}\ y)\ \ \mathsf{K}^{\uparrow}_{e}(y^{-1} * z)}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x \hat{}\ z)} \qquad \ldots \qquad \frac{\mathsf{K}^{\downarrow}_{\mathsf{exp}}(x \hat{}\ (y * z^{-1}))\ \ \mathsf{K}^{\uparrow}_{e}(a * b^{-1})}{\mathsf{K}^{\downarrow}_{\mathsf{noexp}}(x \hat{}\ (y * a * (z * b)^{-1}))}$$

Figure 9. Normal message deduction rules *ND*. Rules containing variables e or e_i denote all variants where these are replaced by noexp or exp. Rules containing n and l denote all variants for $n \geq 1$ and $l \geq 2$. There are 42 exponentiation rules computed from the DH,AC-variants of the exponentiation rule.

Bibliography

- David Basin, Sebastian Mödersheim, and Luca Viganò. *OFMC: A symbolic model checker for security protocols.* International Journal of Information Security, 4(3), 2005.
- Hubert Comon, Véronique Cortier, John Mitchell. *Tree automata with one memory, set constraints, and ping-pong protocols*. ICALP 2001.
- Shimon Even and Oded Goldreich. *On the security of multi-party ping-pong protocols*, Symposium on Foundations of Computer Science, IEEE Computer Society, 1983.
- N.Durgin, P.Lincoln, J.Mitchell, and A.Scedrov. *Undecidability of bounded security protocols.* In Workshop on Formal Methods and Security Protocols (FMSP '99), 1999.
- J. Millen and V. Shmatikov. *Constraint Solving for Bounded-Process Cryptographic Protocol Analysis.* CCS 2001.
- Michaël Rusinowitch and Mathieu Turuani. *Protocol Insecurity with Finite Number of Sessions is NP-complete.* CSFW, 2001.