02244 Logic for Security
Security Protocols
Symbolic Analysis: The Lazy Intruder

Sebastian Mödersheim

February 16, 2026

Dolev-Yao Closure: Summary

Dolev-Yao rules

$$\frac{}{M \vdash m} \text{ if } m \in M \text{ (Axiom)}$$

$$\frac{M \vdash m_1 \quad \ldots \quad M \vdash m_n}{M \vdash f(m_1, \ldots, m_n)} \text{ if } f/n \in \Sigma_p \text{ (Compose)}$$

$$\frac{M \vdash \langle m_1, m_2 \rangle}{M \vdash m_i} \text{ (Proj}_i\text{)} \qquad \frac{M \vdash \{\!|m|\!\}_k \quad M \vdash k}{M \vdash m} \text{ (DecSym)}$$

$$\frac{M \vdash \{m\}_k \quad M \vdash \mathsf{inv}(k)}{M \vdash m} \text{ (DecAsym)} \qquad \frac{M \vdash \{m\}_{\mathsf{inv}(k)}}{M \vdash m} \text{ (OpenSig)}$$

The compose rule is for all public functions Σ_p,
including $\{\!| \cdot |\!\}_{\cdot}$ $\quad \{\cdot\}_{\cdot}$ $\quad \langle \cdot, \cdot \rangle$

Automation

Goal: design (in pseudocode) a decision procedure for Dolev-Yao:

- Given a finite set M of messages (the intruder knowledge)
- and given a message m (the goal)
- Output whether $M \vdash m$ holds.
    - ★ additionally, in the positive case, give the proof.

Automating Dolev-Yao

Step 1: Composition only

Consider first the following simpler problem:

- $M \vdash_c m$ are those deductions where the intruder does not apply any analysis steps ("composition only"):

Composition Only

$$\frac{}{M \vdash_c m} \text{ if } m \in M \text{ (Axiom)}$$

$$\frac{M \vdash_c m_1 \quad \ldots \quad M \vdash_c m_n}{M \vdash_c f(m_1, \ldots, m_n)} \text{ if } f \in \Sigma_p \text{ (Compose)}$$

Example

$M = \{k_1, k_2, \{\!|m|\!\}_{h(k_1,k_2)}\}$ where $h \in \Sigma_p$

- $M \vdash_c h(k_1, k_2)$
- $M \nvdash_c m$
- $M \vdash m$

Automating Dolev-Yao

Step 1: Composition only—Solution

Composition Only

$$\frac{}{M \vdash_c m} \text{ if } m \in M \text{ (Axiom)}$$

$$\frac{M \vdash_c m_1 \quad \ldots \quad M \vdash_c m_n}{M \vdash_c f(m_1, \ldots, m_n)} \text{ if } f \in \Sigma_p \text{ (Compose)}$$

Challenge: decision procedure for $\vdash_c$:

- Input: M and m
- Output yes if $M \vdash_c m$, and no otherwise.

Idea: backwards search for a derivation.

Automating Dolev-Yao

Step 1: Composition only—Solution

Composition Only

$$\frac{}{M \vdash_c m} \text{ if } m \in M \text{ (Axiom)}$$

$$\frac{M \vdash_c m_1 \quad \ldots \quad M \vdash_c m_n}{M \vdash_c f(m_1, \ldots, m_n)} \text{ if } f \in \Sigma_p \text{ (Compose)}$$

Example

$M = \{k_1, k_2, \{\!|\langle n_1, k_3\rangle|\!\}_{h(k_1,k_2)}, \{\!|n_2|\!\}_{k_3}\}$ where $h \in \Sigma_p$

$$\frac{??}{M \vdash_c h(k_1, k_2)}$$

Check Axiom: $h(k_1, k_2) \in M$? (No: we cannot apply Axiom)

Automating Dolev-Yao

Step 1: Composition only—Solution

Composition Only

$$\frac{}{M \vdash_c m} \text{ if } m \in M \text{ (Axiom)}$$

$$\frac{M \vdash_c m_1 \quad \ldots \quad M \vdash_c m_n}{M \vdash_c f(m_1, \ldots, m_n)} \text{ if } f \in \Sigma_p \text{ (Compose)}$$

Example

$M = \{k_1, k_2, \{\!|\langle n_1, k_3\rangle|\!\}_{h(k_1,k_2)}, \{\!|n_2|\!\}_{k_3}\}$ where $h \in \Sigma_p$

$$\frac{??}{M \vdash_c h(k_1, k_2)}$$

Check Compose: h public? (Yes: so we can try Compose)

Automating Dolev-Yao

Step 1: Composition only—Solution

Composition Only

$$\frac{}{M \vdash_c m} \text{ if } m \in M \text{ (Axiom)}$$

$$\frac{M \vdash_c m_1 \quad \ldots \quad M \vdash_c m_n}{M \vdash_c f(m_1, \ldots, m_n)} \text{ if } f \in \Sigma_p \text{ (Compose)}$$

Example

$M = \{k_1, k_2, \{\!|\langle n_1, k_3\rangle|\!\}_{h(k_1,k_2)}, \{\!|n_2|\!\}_{k_3}\}$ where $h \in \Sigma_p$

$$\frac{\dfrac{??}{M \vdash_c k_1} \quad \dfrac{??}{M \vdash_c k_2}}{M \vdash_c h(k_1, k_2)} \; \textit{Compose}$$

Try to find proofs for the new sub-goals – recursively.

Automating Dolev-Yao

Step 1: Composition only—Solution

Composition Only

$$\frac{}{M \vdash_c m} \text{ if } m \in M \text{ (Axiom)}$$

$$\frac{M \vdash_c m_1 \quad \ldots \quad M \vdash_c m_n}{M \vdash_c f(m_1, \ldots, m_n)} \text{ if } f \in \Sigma_p \text{ (Compose)}$$

Example

$M = \{k_1, k_2, \{\!|\langle n_1, k_3\rangle|\!\}_{h(k_1,k_2)}, \{\!|n_2|\!\}_{k_3}\}$ where $h \in \Sigma_p$

$$\frac{\dfrac{}{M \vdash_c k_1}\,\mathit{Axiom} \qquad \dfrac{}{M \vdash_c k_2}\,\mathit{Axiom}}{M \vdash_c h(k_1, k_2)}\,\mathit{Compose}$$

$k_1, k_2 \in M$, so we can solve that with Axiom – done!

Automating Dolev-Yao

Step 1: Composition only—Solution

Composition Only

$$\frac{}{M \vdash_c m} \text{ if } m \in M \text{ (Axiom)}$$

$$\frac{M \vdash_c m_1 \quad \ldots \quad M \vdash_c m_n}{M \vdash_c f(m_1, \ldots, m_n)} \text{ if } f \in \Sigma_p \text{ (Compose)}$$

Decision procedure for $M \vdash_c m$

1. Check if $m \in M$; if so return yes.
2. Otherwise, let $m = f(t_1, \ldots, t_n)$
   - ★ If f is not public return no.
   - ★ Otherwise recursively check whether:

     $$M \vdash_c t_1 \text{ and } \ldots \text{and } M \vdash_c t_n$$

     Return yes if all these return yes, and no otherwise.

Automating Dolev-Yao

Step 2: Analysis—solution

Analysis Steps

- If $\{|m|\}_k \in M$ and $M \vdash_c k$ then add m to M.
- If $\{m\}_k \in M$ and $M \vdash_c \mathsf{inv}(k)$ then add m to M.
- if $\{m\}_{\mathsf{inv}(k)} \in M$ then add m to M.
- If $\langle m_1, m_2 \rangle \in M$ then add m_1 and m_2 to M.
- Repeat until no new messages can be added.

Automating Dolev-Yao

To check $M \vdash m$:

- Perform the Analysis Steps procedure, augmenting M with all derivable messages.
- Now it suffices to check $M \vdash_c m$.

Properties of the algorithm for checking $M \vdash m$:

- Soundness: if algorithm says “yes”, then $M \vdash m$.
- Completeness: if $M \vdash m$, then the algorithm says “yes”.
  - ★ This is quite tricky to prove.
- Termination: the algorithm never runs into an infinite loop.

Automating Dolev-Yao

To check $M \vdash m$:

- Perform the Analysis Steps procedure, augmenting M with all derivable messages.
- Now it suffices to check $M \vdash_c m$.

Properties of the algorithm for checking $M \vdash m$:

- Soundness: if algorithm says "yes", then $M \vdash m$.
- Completeness: if $M \vdash m$, then the algorithm says "yes".
  - ★ This is quite tricky to prove.
- Termination: the algorithm never runs into an infinite loop.
  - ★ Procedure for $\vdash_c$ terminates since it goes to smaller terms

Automating Dolev-Yao

To check $M \vdash m$:

- Perform the Analysis Steps procedure, augmenting M with all derivable messages.
- Now it suffices to check $M \vdash_c m$.

Properties of the algorithm for checking $M \vdash m$:

- Soundness: if algorithm says “yes”, then $M \vdash m$.
- Completeness: if $M \vdash m$, then the algorithm says “yes”.
  - ★ This is quite tricky to prove.
- Termination: the algorithm never runs into an infinite loop.
  - ★ Procedure for $\vdash_c$ terminates since it goes to smaller terms
  - ★ Analysis terminates because it only adds proper subterms of an existing term. This cannot go on forever, since there are only finitely many subterms.

Negative Question

Can we thus prove also statements of the form $M \nvdash m$
. . . that a m cannot be derived from M?

Example

$$M = \{\, k_1, \{\!|m_1|\!\}_{k_1}, m_2, \{\!|m_3|\!\}_{k_2} \,\} \nvdash m_3$$

Negative Question

Can we thus prove also statements of the form $M \nvdash m$
. . . that a m cannot be derived from M?

Example

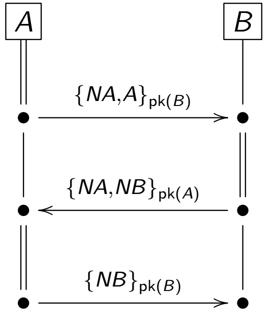
$$M = \{\, k_1, \{\!|m_1|\!\}_{k_1}, m_2, \{\!|m_3|\!\}_{k_2} \,\} \nvdash m_3$$

- Yes, due to completeness when our algorithm answers “no”, we know there is no derivation for m.

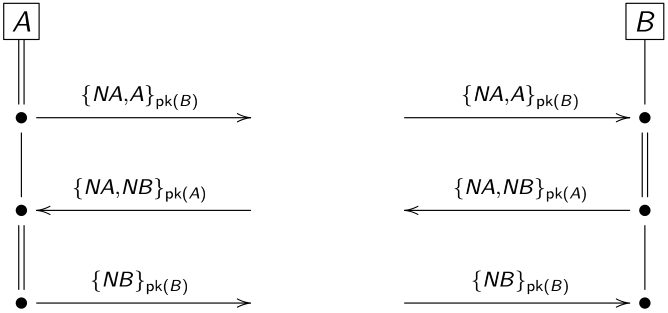
Needham-Schroeder Public-Key Protocol [1978]

```
Protocol: NSPK

Types: Agent A,B;
       Number NA,NB;
       Function pk,h

Knowledge: A: A,pk(A),inv(pk(A)),B,pk(B),h;
           B: B,pk(B),inv(pk(B)),A,pk(A),h
Actions:
A->B: {NA,A}(pk(B))  # A generates NA
B->A: {NA,NB}(pk(A)) # B generates NB
A->B: {NB}(pk(B))

Goals:
h(NA,NB) secret between A,B
```

NSPK as A Message Sequence Chart

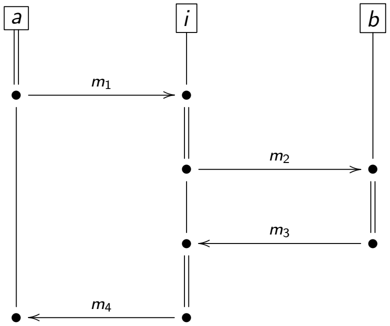

A
B
$\{NA,A\}_{\mathsf{pk}(B)}$
$\{NA,NB\}_{\mathsf{pk}(A)}$
$\{NB\}_{\mathsf{pk}(B)}$

NSPK as Roles / Strands


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

- For each Role of the protocol, a program that sends and receives messages (over possibly insecure network)
- Strand: concrete execution of a role: all variables (here A, B, NA, NB) instantiated with concrete values
  - ★ or a prefix thereof (an agent might not finish)

Attacks

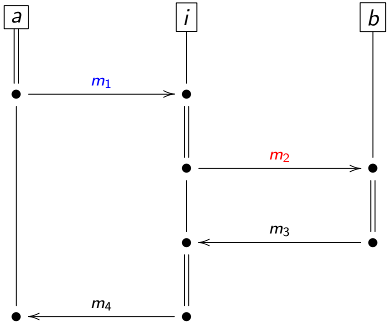


An attack is a strand space where the following conditions are met:

- Messages sent by honest agents are received by *i*
- Messages received by honest agents are sent by *i* who can compose the message from the messages he has received so far.

- The successful completion violates a goal of the protocol.

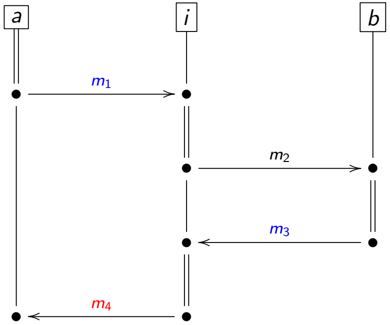
Attacks



An attack is a strand space where the following conditions are met:

- Messages sent by honest agents are received by i
- Messages received by honest agents are sent by i who can compose the message from the messages he has received so far.
  - ★ In the example: $\{m_1\} \vdash m_2$
- The successful completion violates a goal of the protocol.

Attacks



An attack is a strand space where the following conditions are met:

- Messages sent by honest agents are received by i
- Messages received by honest agents are sent by i who can compose the message from the messages he has received so far.
  - ★ In the example: $\{m_1\} \vdash m_2$ and $\{m_1, m_3\} \vdash m_4$.
- The successful completion violates a goal of the protocol.

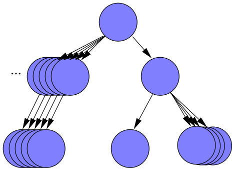
Infinite State Space

Problem 1: at any time, any number of people can run the protocol in parallel. (Think of TLS...)

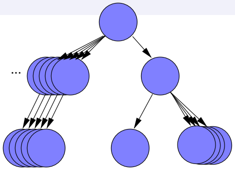
- For now we bound the number of sessions: only finitely many strands of honest agents
- Later: how to verify for unbounded sessions

Problem 2: at any time the intruder has an inifinite choice of message they can construct and send to an agent.

- We will solve this problem with a constraint approach: the lazy intruder.



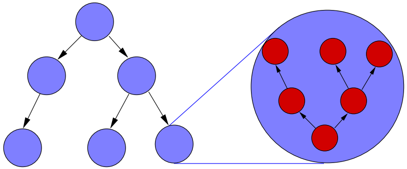
Lazy Intruder: Overview



Even for bounded sessions we have an infinite tree of reachable states, i.e., how the intruder can interact with honest agents.

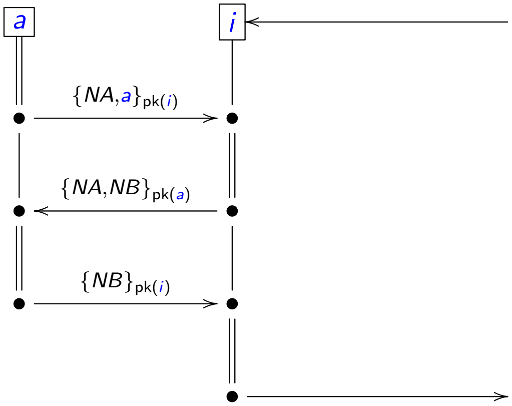
Idea: symbolic states

- Each state is an attack scenario: a sequence of interactions with honest agents, leaving undetermined what exactly the intruder sends.



- Then each state is a constraint solving problem: "Can the intruder generate all messages that the attack scenario has?"
- This will be a backward search: we start at the complete attack scenario and try to see how the intruder could have constructed this.

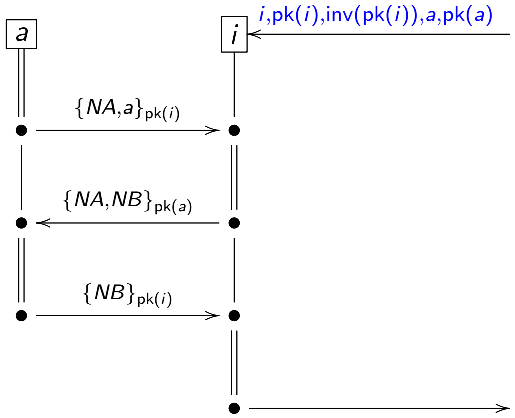
Example: Can the Intruder Play a Protocol Role?


a
i
$\{NA,a\}_{\mathrm{pk}(i)}$
$\{NA,NB\}_{\mathrm{pk}(a)}$
$\{NB\}_{\mathrm{pk}(i)}$

Example:

- NSPK with $A = a$ and $B = i$

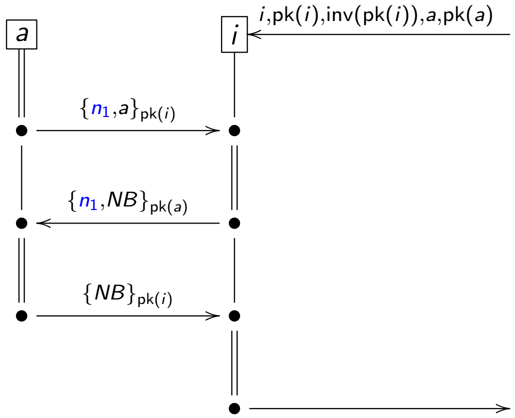
Example: Can the Intruder Play a Protocol Role?


i,pk(i),inv(pk(i)),a,pk(a)
a
i
$\{NA,a\}_{\mathsf{pk}(i)}$
$\{NA,NB\}_{\mathsf{pk}(a)}$
$\{NB\}_{\mathsf{pk}(i)}$

Example:

- NSPK with $A = a$ and $B = i$
- i needs corrsp. knowledge of role B: $i, \mathsf{pk}(i), \mathsf{inv}(\mathsf{pk}(i)), a, \mathsf{pk}(a)$

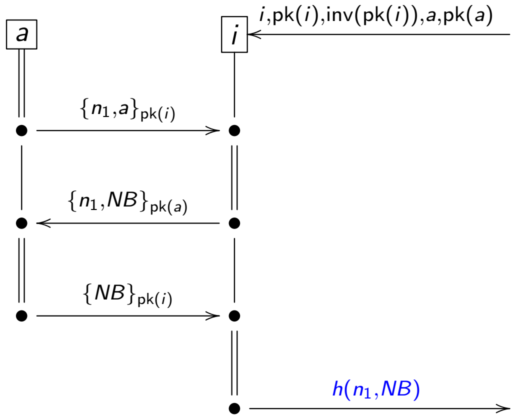
Example: Can the Intruder Play a Protocol Role?


a
i
i,pk(i),inv(pk(i)),a,pk(a)
$\{n_1,a\}_{\mathrm{pk}(i)}$
$\{n_1,NB\}_{\mathrm{pk}(a)}$
$\{NB\}_{\mathrm{pk}(i)}$

Example:

- NSPK with $A = a$ and $B = i$
- i needs corrsp. knowledge of role B: $i, \mathrm{pk}(i), \mathrm{inv}(\mathrm{pk}(i)), a, \mathrm{pk}(a)$
- a uses a fresh $NA = n_1$.

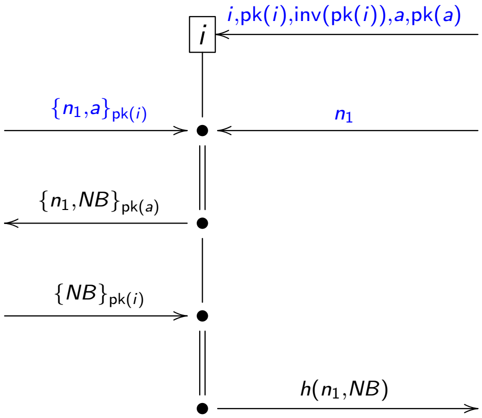
Example: Can the Intruder Play a Protocol Role?


a
i
i,pk(i),inv(pk(i)),a,pk(a)
$\{n_1,a\}_{\mathrm{pk}(i)}$
$\{n_1,NB\}_{\mathrm{pk}(a)}$
$\{NB\}_{\mathrm{pk}(i)}$
$h(n_1,NB)$

Example:

- NSPK with $A = a$ and $B = i$
- i needs corrsp. knowledge of role B: $i, \mathrm{pk}(i), \mathrm{inv}(\mathrm{pk}(i)), a, \mathrm{pk}(a)$
- a uses a fresh $NA = n_1$.
- Afterwards, i should be able to construct the shared key $h(NA, NB) = h(n_1, NB)$

Can the Intruder Produce all Outgoing Messages?

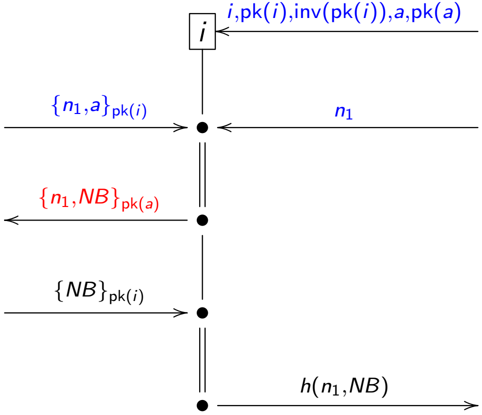


i
i,pk(i),inv(pk(i)),a,pk(a)
$\{n_1,a\}_{\mathrm{pk}(i)}$
n_1
$\{n_1,NB\}_{\mathrm{pk}(a)}$
$\{NB\}_{\mathrm{pk}(i)}$
$h(n_1,NB)$

Analysis of the first incoming messages:

- i can decrypt $\{n_1, a\}_{\mathrm{pk}(a)}$ and obtain n_1.

Can the Intruder Produce all Outgoing Messages?

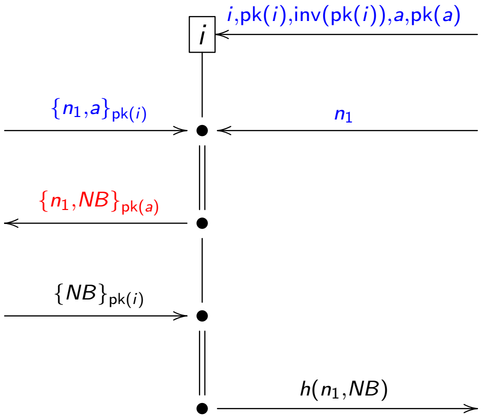


i
$i, \mathsf{pk}(i), \mathsf{inv}(\mathsf{pk}(i)), a, \mathsf{pk}(a)$
$\{n_1, a\}_{\mathsf{pk}(i)}$
n_1
$\{n_1, NB\}_{\mathsf{pk}(a)}$
$\{NB\}_{\mathsf{pk}(i)}$
$h(n_1, NB)$

Can the intruder produce $\{n_1, NB\}_{\mathsf{pk}(a)}$ from his current knowledge?

- Axiom: does not work, no fitting message known.

Can the Intruder Produce all Outgoing Messages?

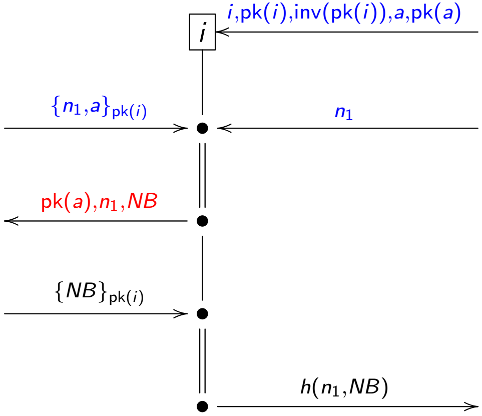


i
i,pk(i),inv(pk(i)),a,pk(a)
$\{n_1,a\}_{\mathrm{pk}(i)}$
n_1
$\{n_1,NB\}_{\mathrm{pk}(a)}$
$\{NB\}_{\mathrm{pk}(i)}$
$h(n_1,NB)$

Can the intruder produce $\{n_1, NB\}_{\mathrm{pk}(a)}$ from his current knowledge?

- Axiom: does not work, no fitting message known.
- Compose: public key encryption is public operator.
  - ★ Idea: replace the composed term by its subterms in the strand!

Can the Intruder Produce all Outgoing Messages?

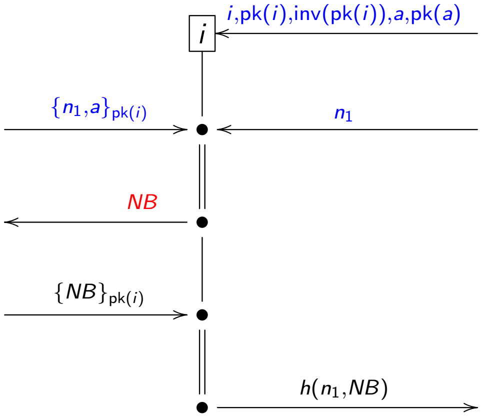


i
$i, \mathsf{pk}(i), \mathsf{inv}(\mathsf{pk}(i)), a, \mathsf{pk}(a)$
$\{n_1, a\}_{\mathsf{pk}(i)}$
n_1
$\mathsf{pk}(a), n_1, NB$
$\{NB\}_{\mathsf{pk}(i)}$
$h(n_1, NB)$

Can the intruder produce $\{n_1, NB\}_{\mathsf{pk}(a)}$ from his current knowledge?

- it sufficies to produce $\mathsf{pk}(a)$, n_1 and NB

Can the Intruder Produce all Outgoing Messages?

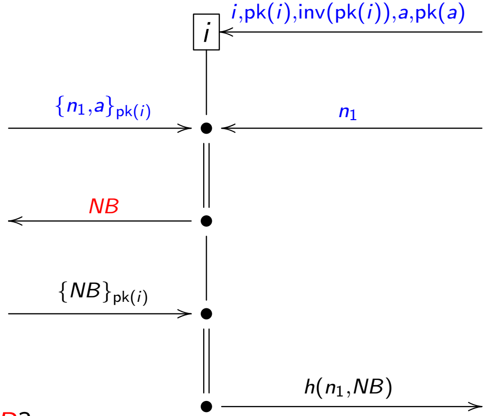


i
i,pk(i),inv(pk(i)),a,pk(a)
$\{n_1,a\}_{\mathrm{pk}(i)}$
n_1
NB
$\{NB\}_{\mathrm{pk}(i)}$
$h(n_1,NB)$

Can the intruder produce $\{n_1, NB\}_{\mathrm{pk}(a)}$ from his current knowledge?

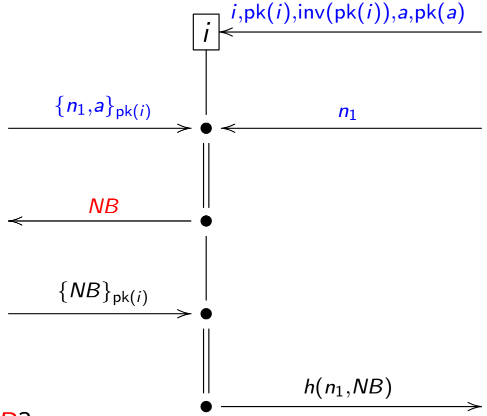
- it sufficies to produce $\mathrm{pk}(a)$, n_1 and NB
- $\mathrm{pk}(a)$ and n_1 are in the knowledge: done with axiom.

Can the Intruder Produce all Outgoing Messages?


i
$i, \mathsf{pk}(i), \mathsf{inv}(\mathsf{pk}(i)), a, \mathsf{pk}(a)$
$\{n_1, a\}_{\mathsf{pk}(i)}$
n_1
NB
$\{NB\}_{\mathsf{pk}(i)}$
$h(n_1, NB)$

What about NB?

Can the Intruder Produce all Outgoing Messages?

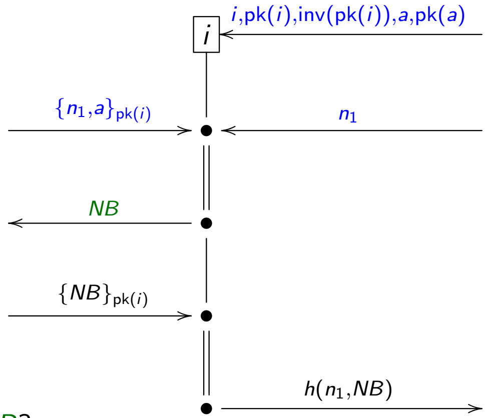


i
i,pk(i),inv(pk(i)),a,pk(a)
$\{n_1,a\}_{\text{pk}(i)}$
n_1
NB
$\{NB\}_{\text{pk}(i)}$
$h(n_1,NB)$

What about NB?

- Intruder's choice: he can send any message he can compose from his knowledge. There are infinitely many possibilities!

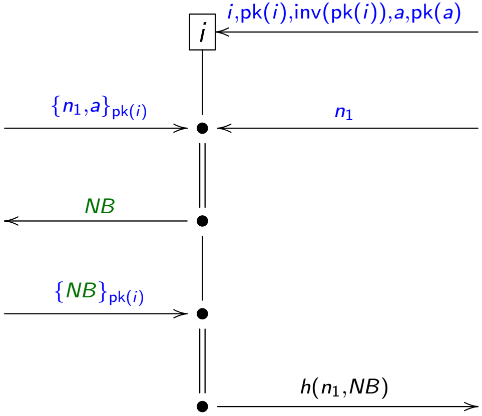
Can the Intruder Produce all Outgoing Messages?


i,pk(i),inv(pk(i)),a,pk(a)
i
$\{n_1,a\}_{\mathrm{pk}(i)}$
n_1
NB
$\{NB\}_{\mathrm{pk}(i)}$
$h(n_1,NB)$

What about NB?

- Intruder's choice: he can send any message he can compose from his knowledge. There are infinitely many possibilities!
- Lazy intruder: just leave the variable NB as is.
  - ★ It does not matter what NB for now.
    So we do not decide what NB is – until it matters.

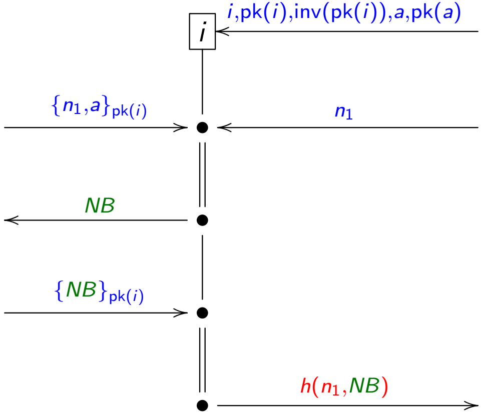
Can the Intruder Produce all Outgoing Messages?


i
i,pk(i),inv(pk(i)),a,pk(a)
$\{n_1,a\}_{\mathsf{pk}(i)}$
n_1
NB
$\{NB\}_{\mathsf{pk}(i)}$
$h(n_1,NB)$

Analyzing the next incoming message: $\{NB\}_{\mathsf{pk}(i)}$

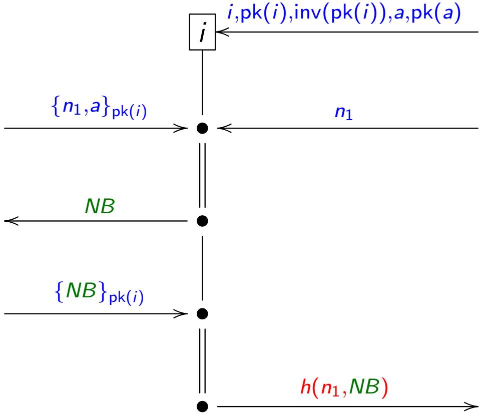
- i can decrypt it, but that yields just NB.
- Whatever NB is, it came from i, so nothing new!

Can the Intruder Produce all Outgoing Messages?


i,pk(i),inv(pk(i)),a,pk(a)
i
$\{n_1,a\}_{\mathrm{pk}(i)}$
n_1
NB
$\{NB\}_{\mathrm{pk}(i)}$
$h(n_1,NB)$

Can i generate the session key $h(n_1, NB)$?

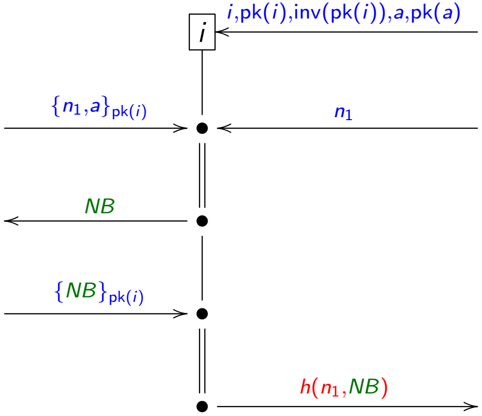
Can the Intruder Produce all Outgoing Messages?


i
i,pk(i),inv(pk(i)),a,pk(a)
$\{n_1,a\}_{\mathrm{pk}(i)}$
n_1
NB
$\{NB\}_{\mathrm{pk}(i)}$
$h(n_1,NB)$

Can i generate the session key $h(n_1, NB)$?

- Axiom is not possible

Can the Intruder Produce all Outgoing Messages?

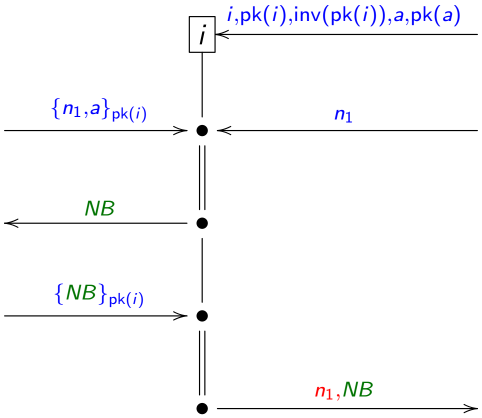


i
$i, \mathsf{pk}(i), \mathsf{inv}(\mathsf{pk}(i)), a, \mathsf{pk}(a)$
$\{n_1, a\}_{\mathsf{pk}(i)}$
n_1
NB
$\{NB\}_{\mathsf{pk}(i)}$
$h(n_1, NB)$

Can i generate the session key $h(n_1, NB)$?

- Axiom is not possible
- Compose is possible, since h is public.

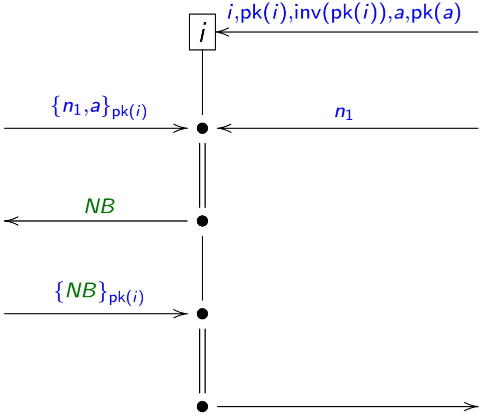
Can the Intruder Produce all Outgoing Messages?



Can i generate the session key $h(n_1, NB)$?

- Axiom is not possible
- Compose is possible, since h is public.
- both n_1 and NB are known by i.

Can the Intruder Produce all Outgoing Messages?


i
i,pk(i),inv(pk(i)),a,pk(a)
$\{n_1,a\}_{\mathrm{pk}(i)}$
n_1
NB
$\{NB\}_{\mathrm{pk}(i)}$

Can i generate the session key $h(n_1, NB)$?

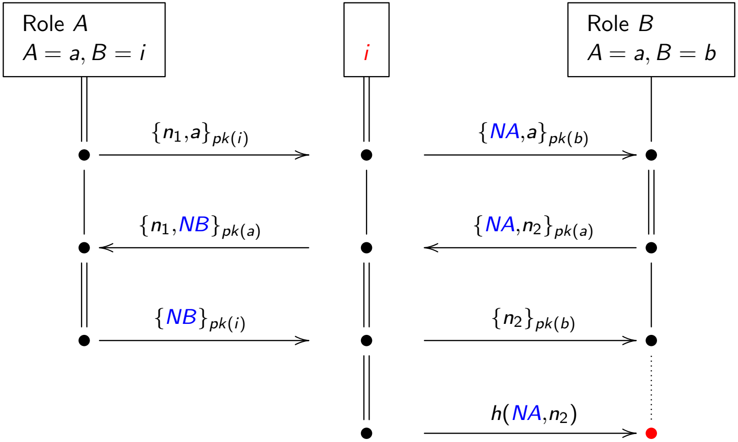
- Axiom is not possible
- Compose is possible, since h is public.
- both n_1 and NB are known by i.
- Nothing else to do – just choose an arbitrary NB.

The Lazy Intruder Attacking

- So far, we have only looked at how the intruder can play a role of the protocol under his real name.
  - ★ We have with this a check that the role is even executable: the initial knowledge is sufficient to perform all steps of a "normal" protocol execution.
- We want to use this technique for checking an attack instead.
- For the NSPK we consider the following attack scenario:
  - ★ a in role A wants to talk to i in role B
    - ▶ She uses fresh n_1 as NA in her strand
  - ★ b in role B is willing to talk to a in role A;
    - ▶ He uses fresh n_2 as NB in his strand
- To prove security, one has to consider any possible attack scenario, i.e., any number of sessions between agents in roles A and B.
  - ★ If we limit ourselves to a bounded number of sessions, there are finitely many scenarios. (E.g. OFMC for two sessions)
- We want to show that $b : B$ can be attacked: the intruder can find out the session key that $b : B$ believes to have with $a : A$.

The Lazy Intruder Attacking

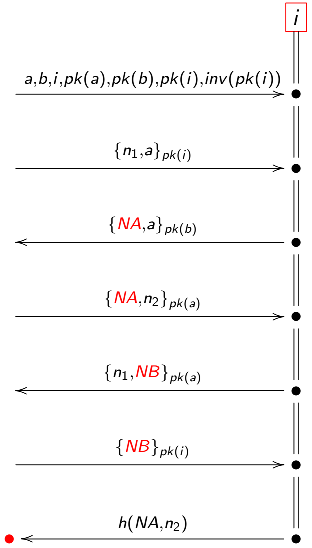
Scenario



The challenge labeled ●:

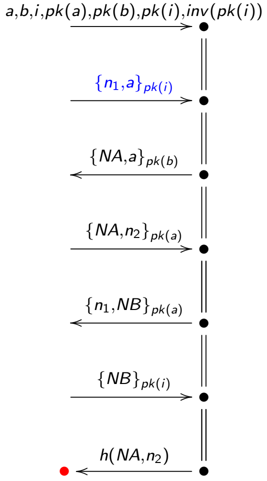
- Can the intruder produce the secret session key $h(NA, n2)$?

Choose and Check



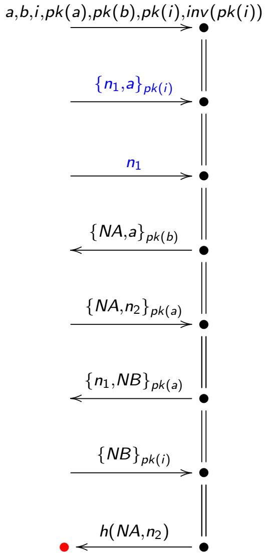
- For simplicity we display here only outgoing and incoming messages of the intruder.

Lazy Intruder Constraint Solving – Example



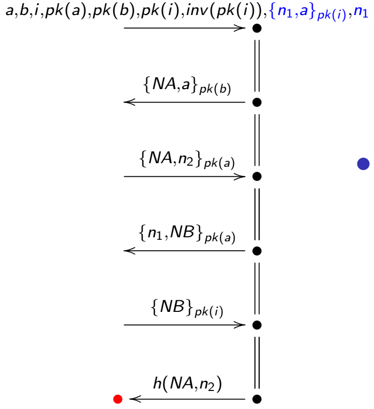
- Here we can decrypt $\{n_1, a\}_{pk(i)}$ since we know the private key $inv(pk(i))$
- Since a is already known, we only learn n_1 from this.

Lazy Intruder Constraint Solving – Example



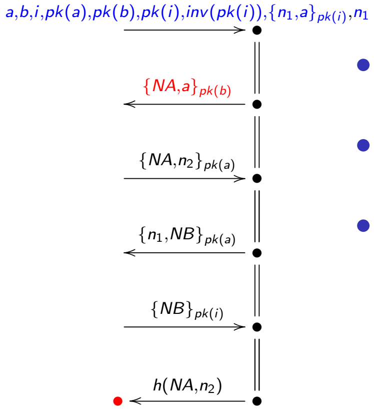
- Here we can decrypt $\{n_1, a\}_{pk(i)}$ since we know the private key $inv(pk(i))$
- Since a is already known, we only learn n_1 from this.

Lazy Intruder Constraint Solving – Example



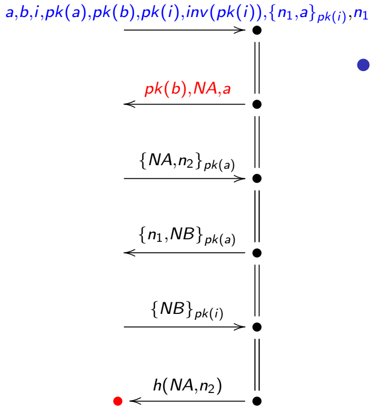
- Compressing notation a bit

Lazy Intruder Constraint Solving – Example



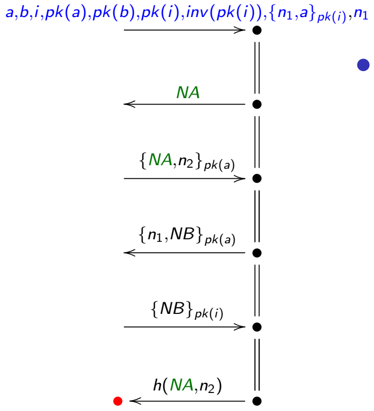
- Consider the first outgoing message $\{NA,a\}_{pk(b)}$
- Axiom does not work (no matching message known)
- So let us go for composition.

Lazy Intruder Constraint Solving – Example



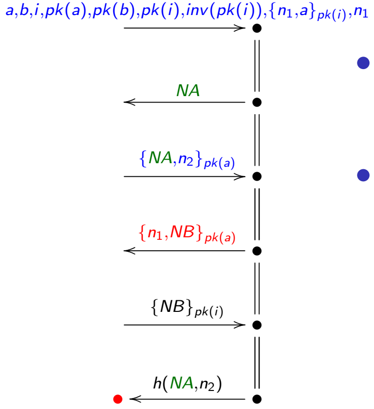
- pk(b) and a are known – done with Axiom

Lazy Intruder Constraint Solving – Example



- NA is a variable: here we are lazy.

Lazy Intruder Constraint Solving – Example

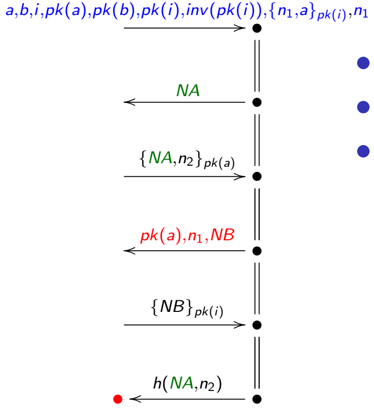


- The next incoming message $\{NA, n_2\}_{pk(a)}$ cannot be decrypted since i does not have inv(pk(a)).
- The next outgoing message $\{n_1, NB\}_{pk(a)}$
  - ★ Axiom is possible
  - ★ Compose is possible

  We need to check both.

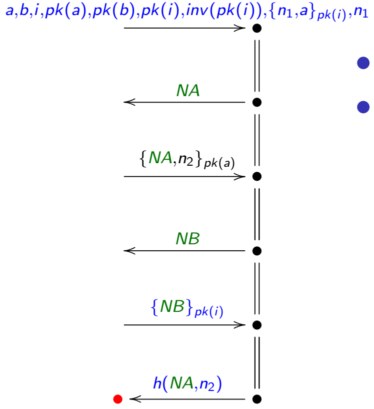
  Let's try compose first.

Lazy Intruder Constraint Solving – Example



- Composing $\{n_1, NB\}_{pk(a)}$
- $pk(a)$ and n_1 are known.
- NB is a variable and we are lazy again.

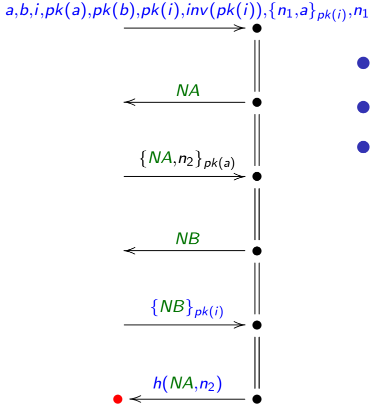
Lazy Intruder Constraint Solving – Example



$a, b, i, pk(a), pk(b), pk(i), inv(pk(i)), \{n_1, a\}_{pk(i)}, n_1$
NA
$\{NA, n_2\}_{pk(a)}$
NB
$\{NB\}_{pk(i)}$
$h(NA, n_2)$

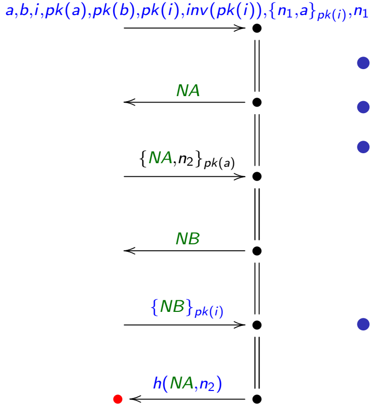
- We can decrypt $\{NB\}_{pk(i)}$
- but NB is already known.

Lazy Intruder Constraint Solving – Example



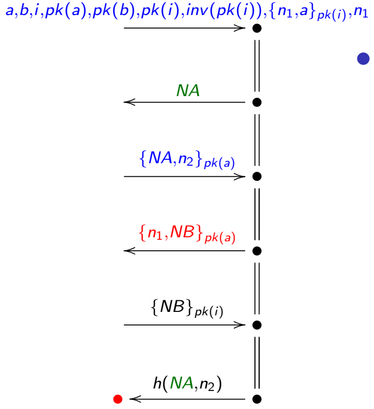
- We can decrypt $\{NB\}_{pk(i)}$
- but NB is already known.
- For challenge $h(NA, n_2)$:
  - ★ Axiom is not possible (no matching message known)
  - ★ Composition requires us to produce n_2 ... which do not have.

Lazy Intruder Constraint Solving – Example



- We can decrypt $\{NB\}_{pk(i)}$
- but NB is already known.
- For challenge $h(NA, n_2)$:
  - ★ Axiom is not possible (no matching message known)
  - ★ Composition requires us to produce n_2 ... which do not have.
- So: backtrack to the last point with an alternative.

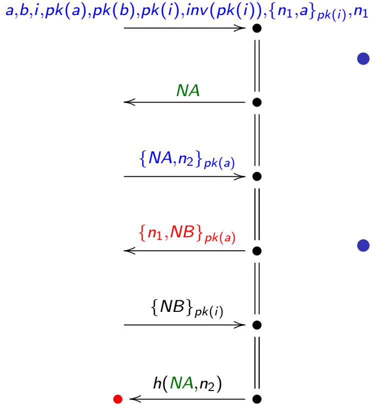
Lazy Intruder Constraint Solving – Example



- The next outgoing message $\{n_1, NB\}_{pk(a)}$
  - ★ Axiom is possible
  - ★ Composition Failed

  Let's try Axiom then!

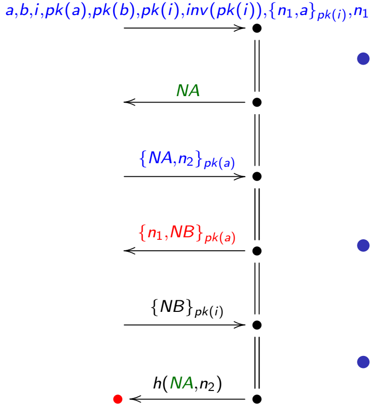
Lazy Intruder Constraint Solving – Example



- The next outgoing message $\{n_1, NB\}_{pk(a)}$
  - ★ Axiom is possible
  - ★ Composition Failed

  Let's try Axiom then!
- The known message $\{NA, n_2\}_{pk(a)}$ and the target $\{n_1, NB\}_{pk(a)}$ are equal if $NA = n_1$ and $NB = n_2$

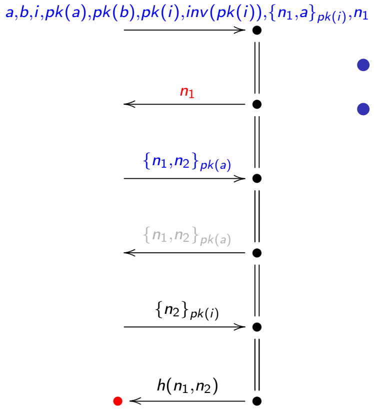
Lazy Intruder Constraint Solving – Example



- The next outgoing message $\{n_1, NB\}_{pk(a)}$
  - ★ Axiom is possible
  - ★ Composition Failed

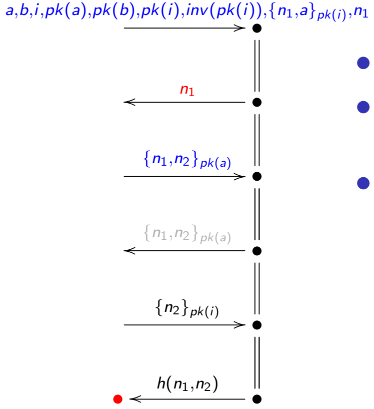
  Let's try Axiom then!
- The known message $\{NA, n_2\}_{pk(a)}$ and the target $\{n_1, NB\}_{pk(a)}$ are equal if $NA = n_1$ and $NB = n_2$
- Under that unifier we can apply Axiom!

Lazy Intruder Constraint Solving – Example



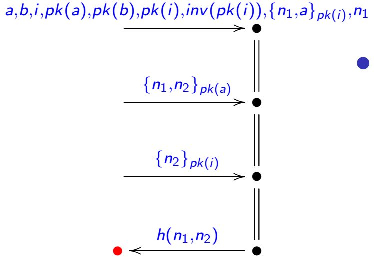
- Replacing $NA = n_1$ and $NB = n_2$.
- Now we can handle $\{n_1, n_2\}_{pk(a)}$ by Axiom.

Lazy Intruder Constraint Solving – Example



- Replacing $NA = n_1$ and $NB = n_2$.
- Now we can handle $\{n_1, n_2\}_{pk(a)}$ by Axiom.
- The NA where we were lazy is replaced by the concrete n_1 now.
  - ★ We cannot be lazy about that – but n_1 is known
  - ★ Handled by Axiom

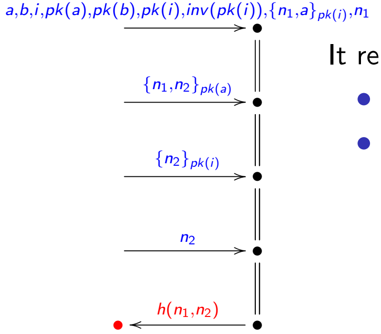
Lazy Intruder Constraint Solving – Example



$a, b, i, pk(a), pk(b), pk(i), inv(pk(i)), \{n_1, a\}_{pk(i)}, n_1$
$\{n_1, n_2\}_{pk(a)}$
$\{n_2\}_{pk(i)}$
$h(n_1, n_2)$

- The message $\{n_2\}_{pk(i)}$ can be decrypted, giving us n_2.

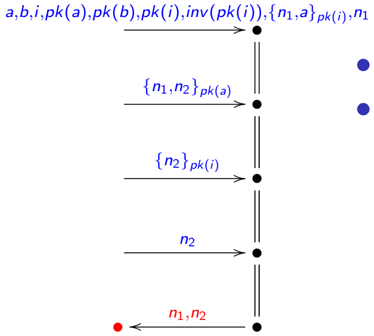
Lazy Intruder Constraint Solving – Example



It remains to solve the challenge $h(n_1, n_2)$:

- Axiom cannot be applied
- Try Composition

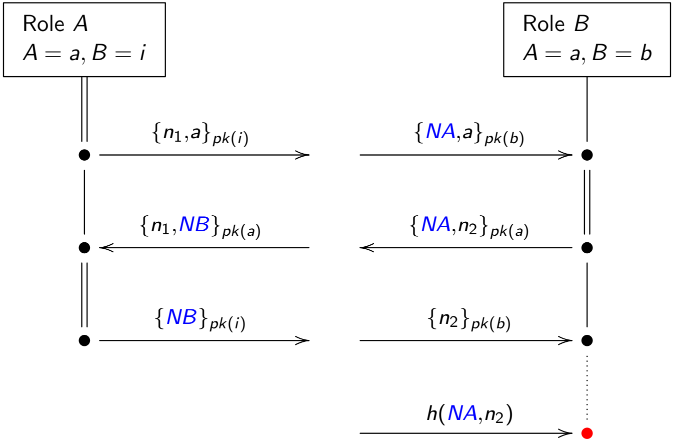
Lazy Intruder Constraint Solving – Example


$a, b, i, pk(a), pk(b), pk(i), inv(pk(i)), \{n_1, a\}_{pk(i)}, n_1$
$\{n_1, n_2\}_{pk(a)}$
$\{n_2\}_{pk(i)}$
n_2
n_1, n_2

- Both n_1 and n_2 are known – Axiom.
- SOLVED!

The Lazy Intruder Attacking

Review of the Example

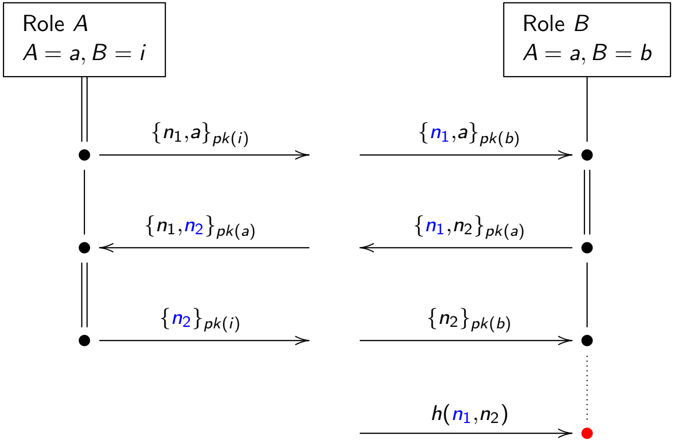


We found an attack:

- For $NA = n_1$ and $NB = n_2$, the intruder can attack b.

The Lazy Intruder Attacking

Review of the Example



We found an attack:

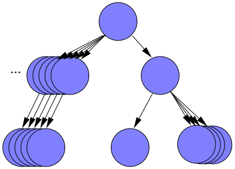
- For $NA = n_1$ and $NB = n_2$, the intruder can attack b.
- This attack was first found by Lowe [1996]

Needham-Schroeder-Lowe [1996]

```
Protocol: NSL

Types: Agent A,B;
       Number NA,NB;
       Function pk,h

Knowledge: A: A,pk(A),inv(pk(A)),B,pk(B),h;
           B: B,pk(B),inv(pk(B)),A,pk(A),h
Actions:
A->B: {NA,A}(pk(B))
B->A: {NA,NB,B}(pk(A))
      # Inserted B's name into the message
A->B: {NB}(pk(B))

Goals:
h(NA,NB) secret between A,B
```

Lazy Intruder: Summary

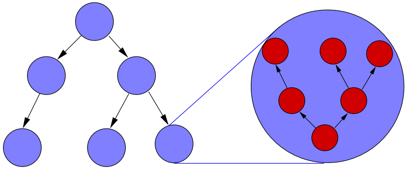
- Without the lazy approach, we would get an infinite search tree because the intruder has often an infinite choice of messages to send.



- We avoid this by using the lazy intruder:

  Layer 1: a symbolic search tree

  Layer 2: constraint solving



Lazy Intruder: Summary

Constraint Solving

Go step by step through the constraint.

- For incoming messages: can a decryption rule be applied?
  If so, add the decrypted message also as an incoming message
  - ★ When the key contains variables this can be handled by adding the key as an outgoing message, i.e., require that the intruder can produce it.
- For outgoing messages
  - ★ If it is a variable: be lazy for now.
    - ► If it gets replaced, you need to come back here.
  - ★ Otherwise: check all following possibilities (backtracking!):
    - ► Compose: can a compose rule be applied?
    - ► Axiom: can the axiom rule be applied?
      This may require instantiating variables

Lazy Intruder: Summary

Theorem (Rusinowitch & Turuani 2001)

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

Proof Sketch.

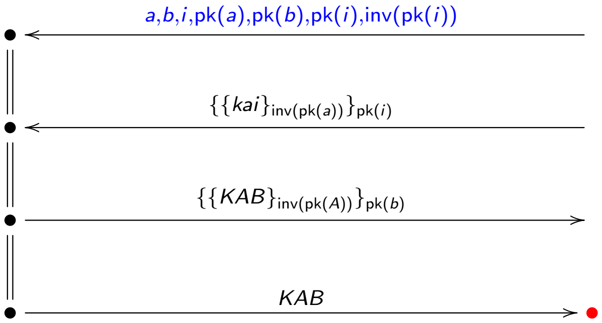
In NP: Guess a symbolic attack trace for the given strands and a sequence of reduction steps for the resulting constraints. Check that this sequence of reduction steps solves the constraint.

NP-hard: Polynomial reduction for boolean formulae to security protocols such that formula satisfiable iff protocol has an attack. □

Relevant Research Papers

- David Basin, Sebastian Mödersheim, and Luca Viganò. *OFMC: A symbolic model checker for security protocols.* International Journal of Information Security, 4(3), 2005.
- Gavin Lowe. *Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR*. Software Concepts Tools, 17(3), 1996.
- Jonathan K. Millen and Vitaly Shmatikov. *Constraint solving for bounded-process cryptographic protocol analysis*. Computer and Communications Security, 2001,
- Roger Needham and Michael Schroeder. *Using Encryption for Authentication in Large Networks of Computers.* Communications of the ACM, 21(12), 1978.
- Michaël Rusinowitch and Mathieu Turuani. *Protocol Insecurity with Finite Number of Sessions is NP-complete.* Computer Security Foundations Workshop, 2001.

Exercise


$a, b, i, \mathsf{pk}(a), \mathsf{pk}(b), \mathsf{pk}(i), \mathsf{inv}(\mathsf{pk}(i))$
$\{\{kai\}_{\mathsf{inv}(\mathsf{pk}(a))}\}_{\mathsf{pk}(i)}$
$\{\{KAB\}_{\mathsf{inv}(\mathsf{pk}(A))}\}_{\mathsf{pk}(b)}$
KAB

Exercise: show that this constraint has both

- a solution where $A = i$ (i.e., a normal execution)
- a solution where $A = a$ (i.e., an attack)