Model checking secrecy

Sjouke Mauw
joint work with Cas Cremers

ECSS group
Eindhoven University of Technology
The Netherlands

TU/e

Outline

- Security protocols and secrecy.
- Example: Bilateral Key Exchange.
- Model checking algorithm.
- Comparison.
- Concluding remarks.

TU/e

Motivation

"*Security protocols are three-line programs that people still manage to get wrong*"

(Roger Needham)

- Security protocol = set of interaction rules to guarantee security property (plus some intended functionality).
- Formal validation is imperative and feasible.
- Security properties: secrecy, authentication, non-repudiation, availability, . . .

TU/e

Model checking secrecy

- Well-understood property.
- Several tools available.
- Often: general purpose model checker instantiated for this problem.

Conjecture.

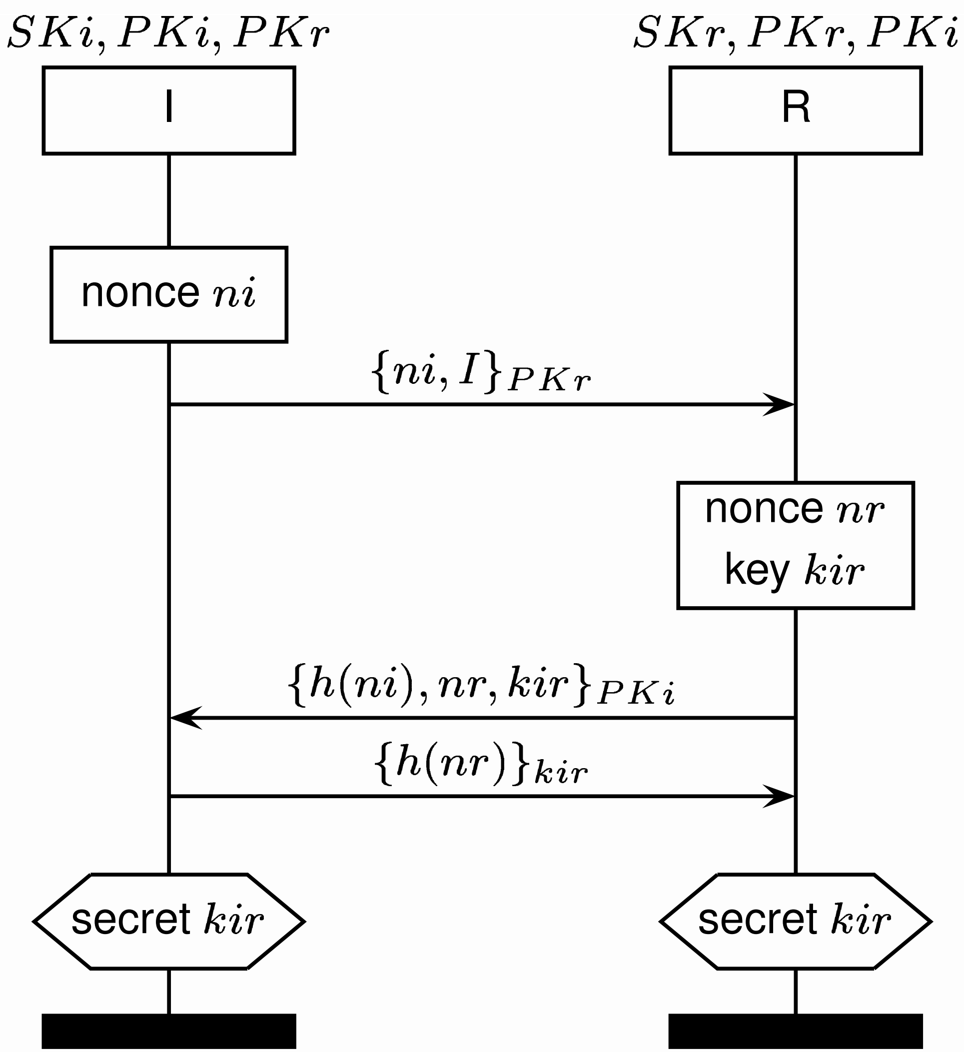
A model checker dedicated to verifying secrecy in security protocols will outperform general purpose model checkers applied to this problem.

Example

Bilateral Key Exchange (BKE):

Given a Public Key Infrastructure, two agents should agree upon the value of a freshly generated symmetric key. This key should remain secret.

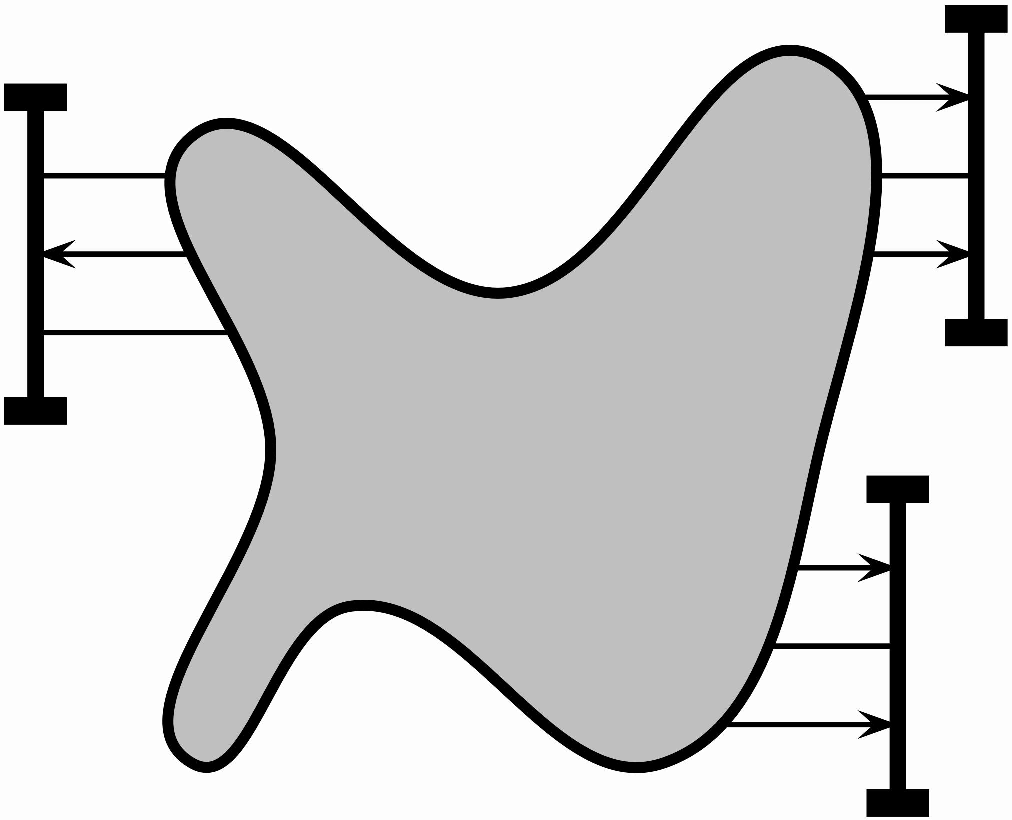
BKE



SKi, PKi, PKr
SKr, PKr, PKi
I
R
nonce ni
$\{ni, I\}_{PKr}$
nonce nr
key kir
$\{h(ni), nr, kir\}_{PKi}$
$\{h(nr)\}_{kir}$
secret kir
secret kir

Intruder model (Dolev-Yao)

- Intruder has complete control over network.



Intruder model (Dolev-Yao)

- Intruder has complete control over network.
- Intruder can pack/unpack messages as long as he knows the cryptographic key.

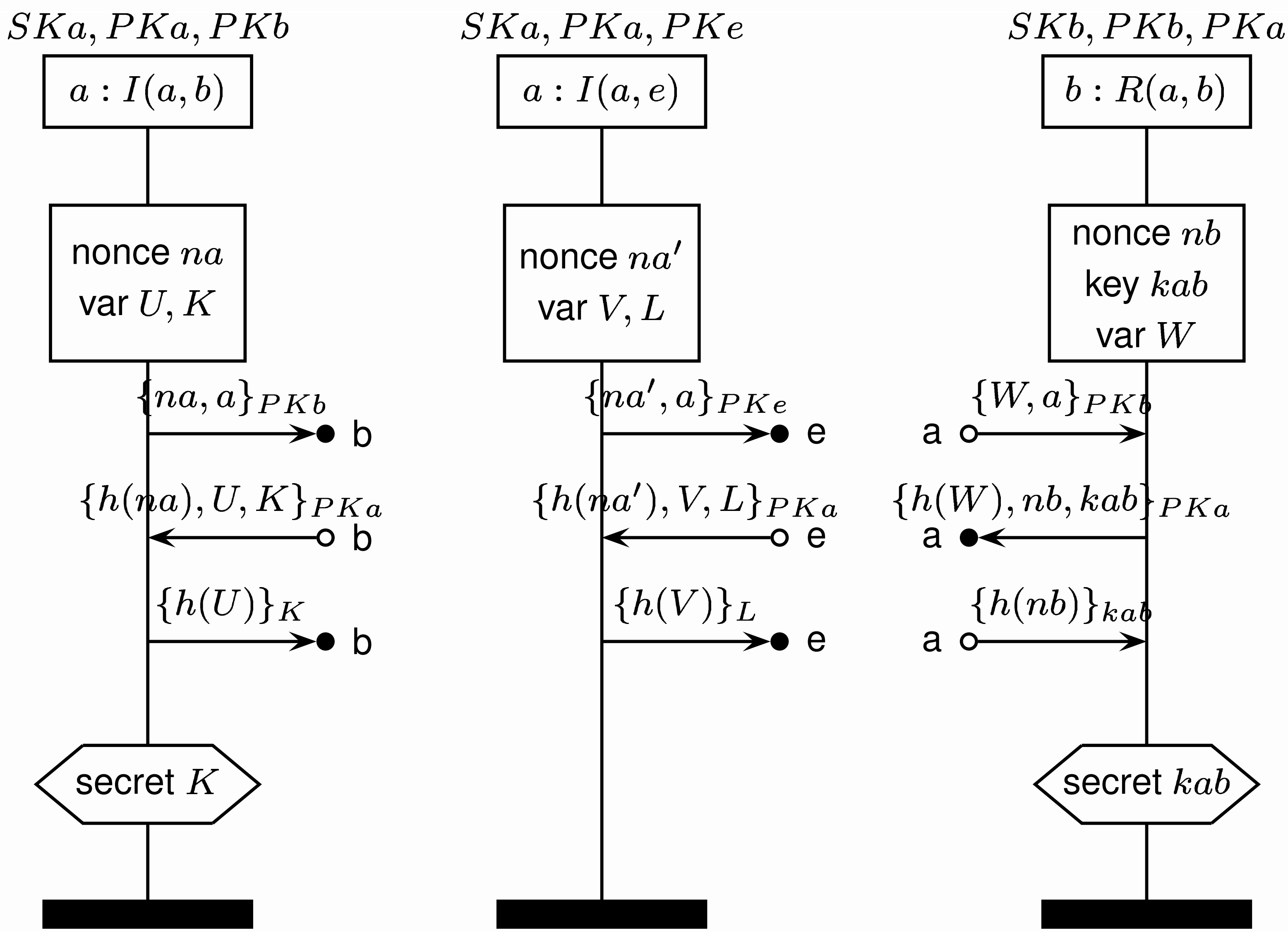
TU/e

Intruder model (Dolev-Yao)

- Intruder has complete control over network.
- Intruder can pack/unpack messages as long as he knows the cryptographic key.
- Possibly conspiring agents, i.e. intruder knows their secret keys.

TU/e

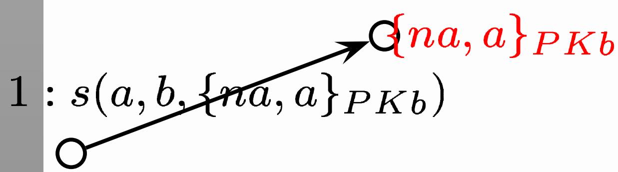
Finite scenario



TU/e

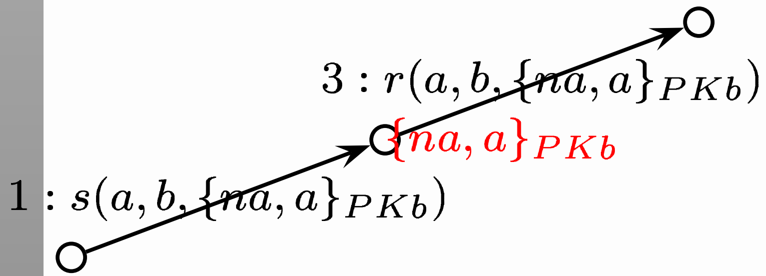
State space

Initial intruder knowledge: PKa, PKb, PKe, SKe



State space

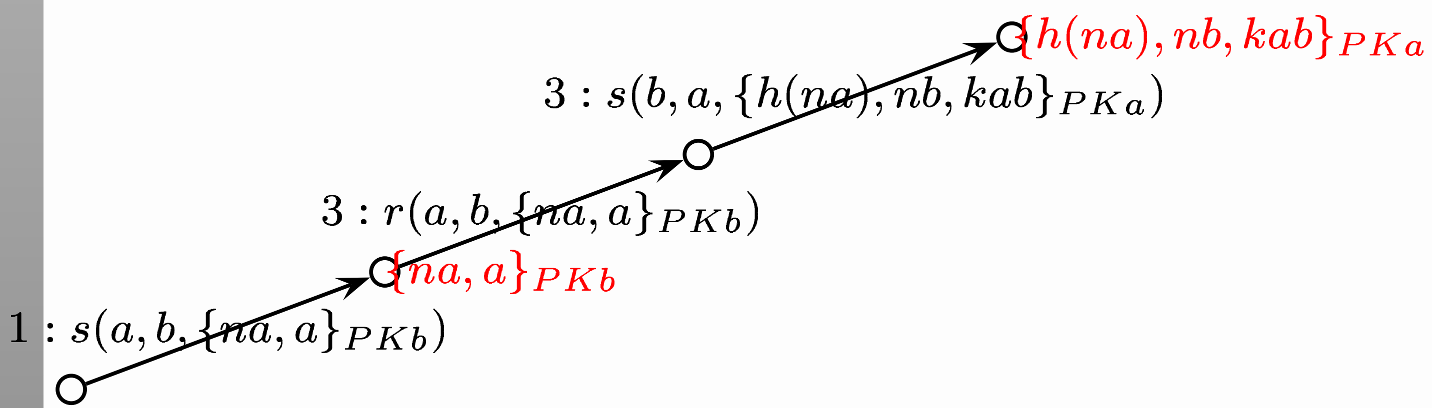
Initial intruder knowledge: PKa, PKb, PKe, SKe



TU/e

State space

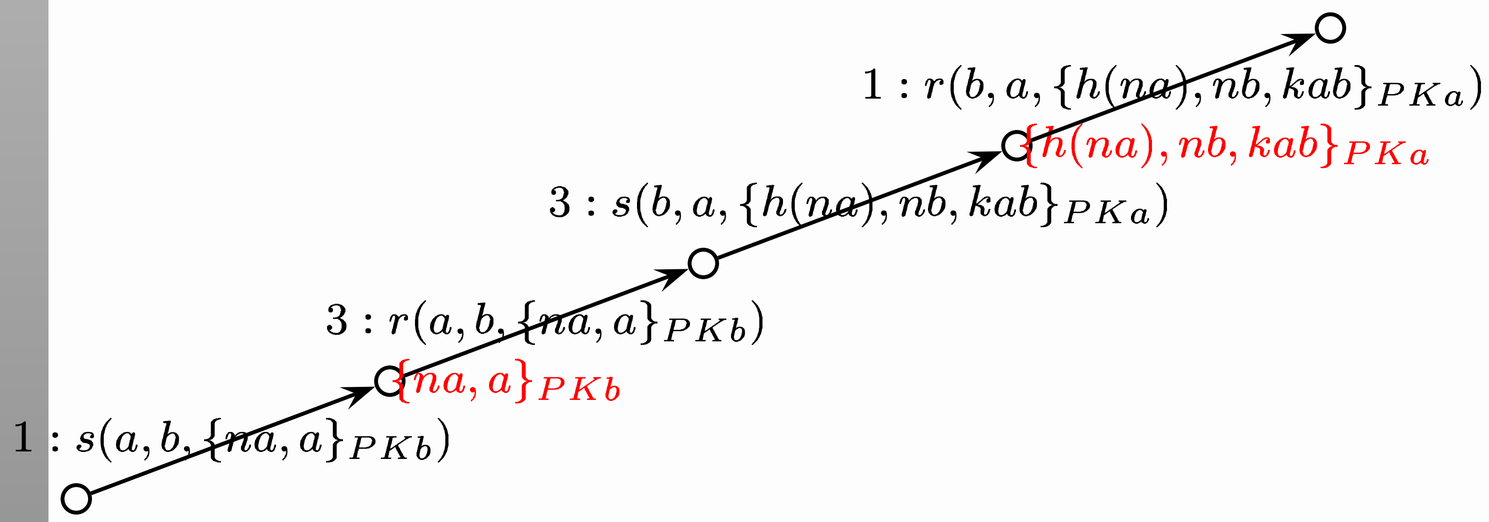
Initial intruder knowledge: PKa, PKb, PKe, SKe



TU/e

State space

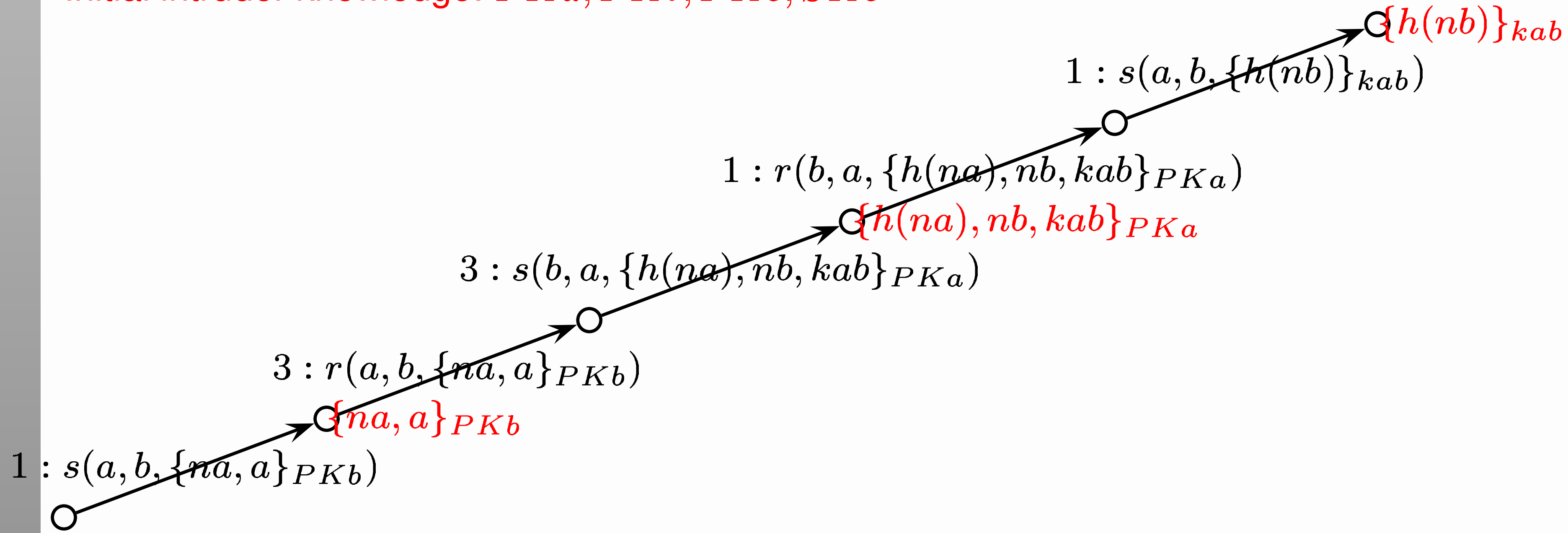
Initial intruder knowledge: PKa, PKb, PKe, SKe



TU/e

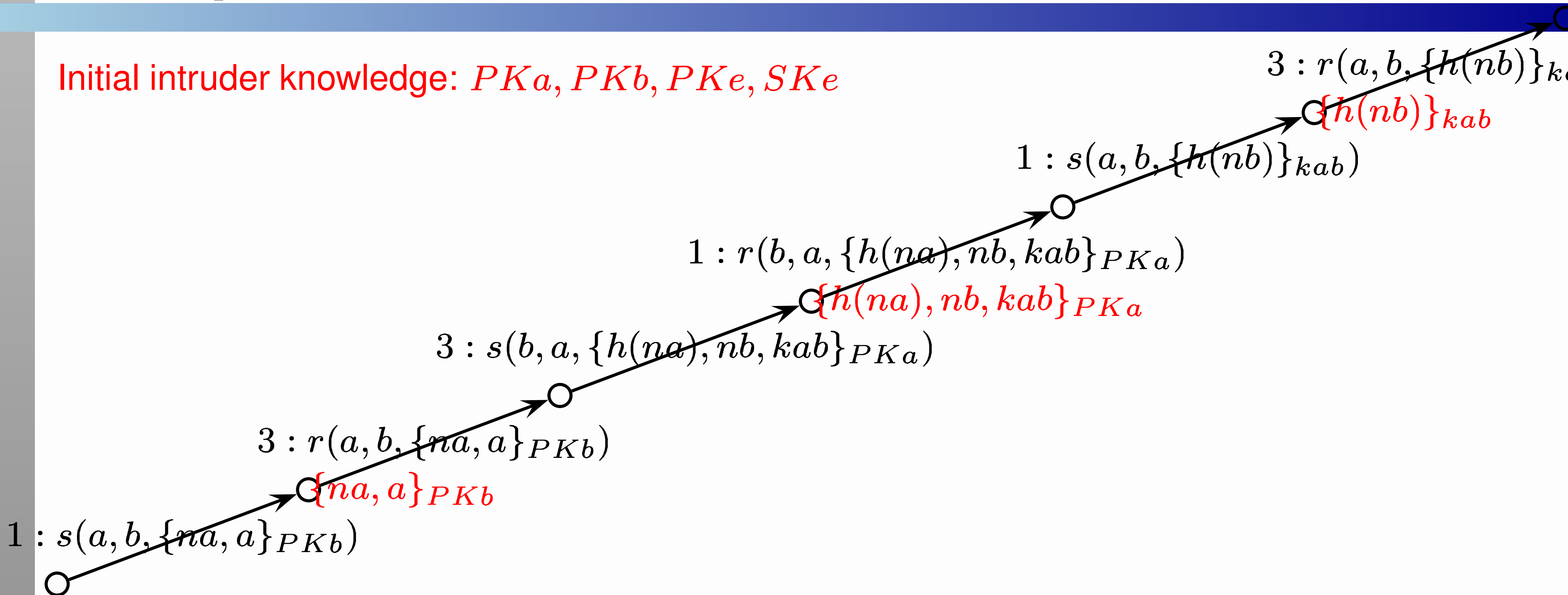
State space

Initial intruder knowledge: PKa, PKb, PKe, SKe



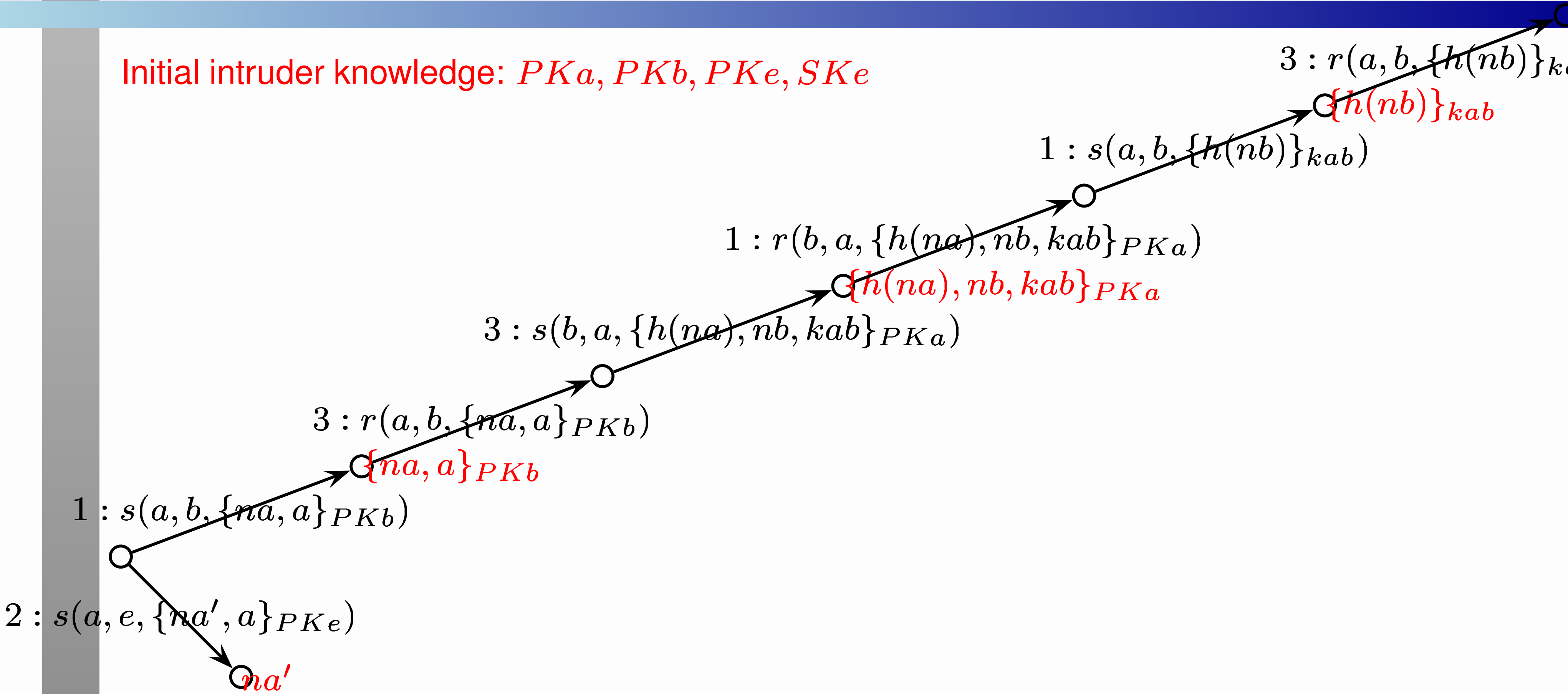
State space

Initial intruder knowledge: PKa, PKb, PKe, SKe



State space

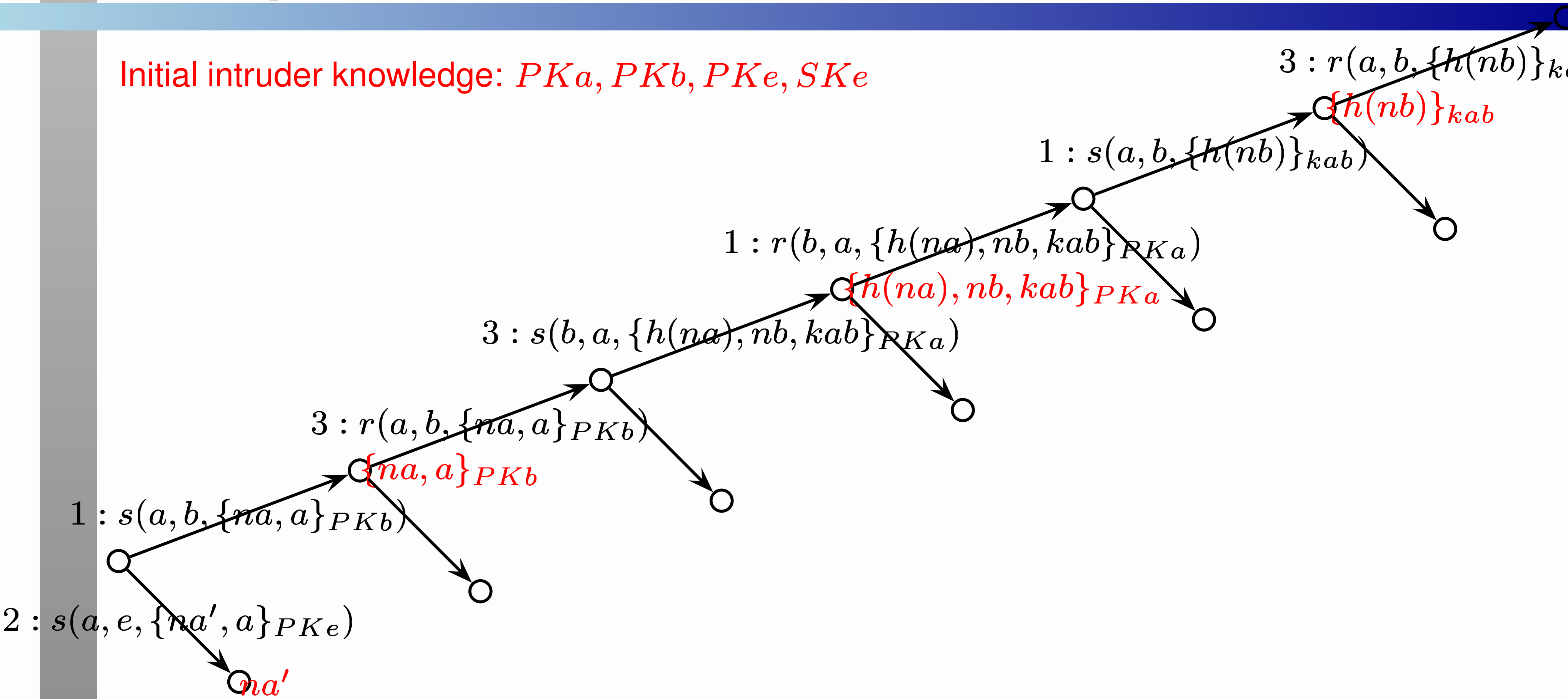
Initial intruder knowledge: PKa, PKb, PKe, SKe



TU/e

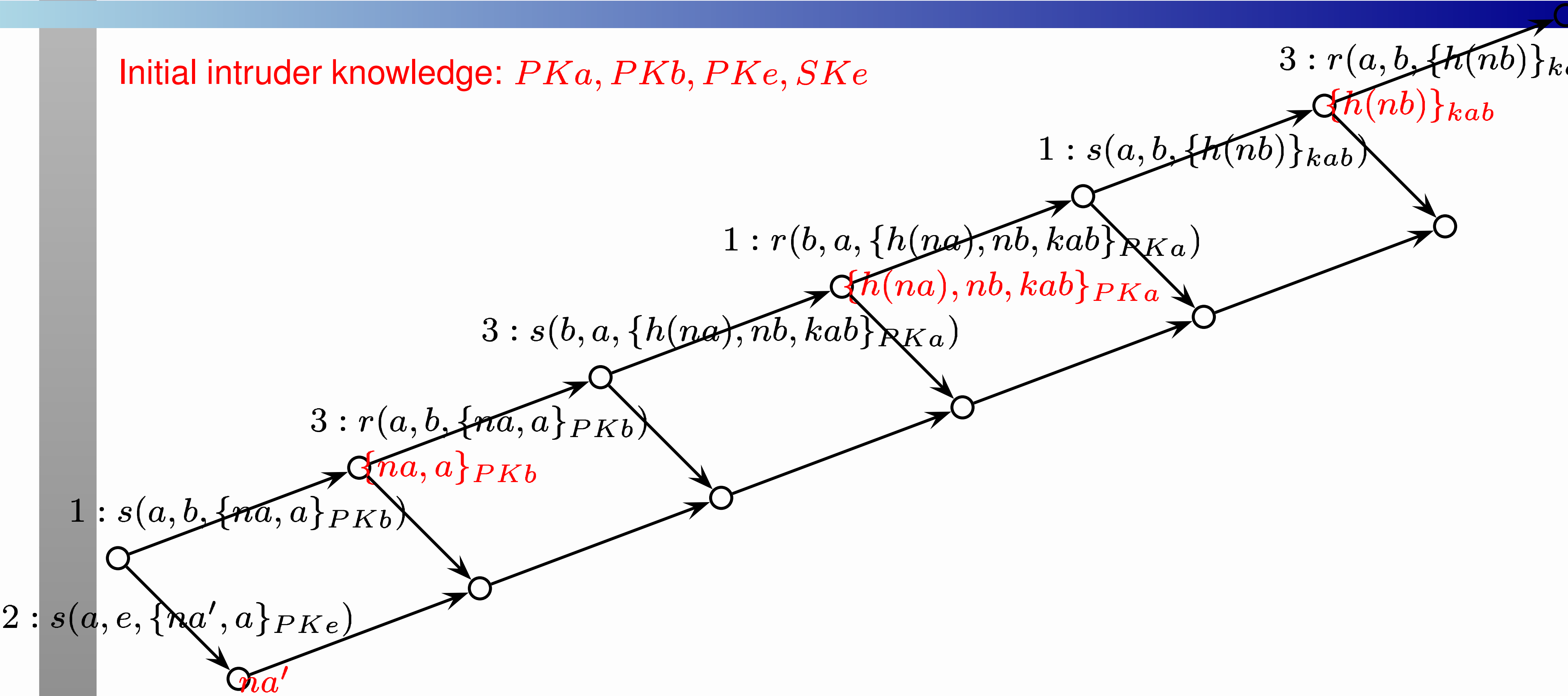
State space

Initial intruder knowledge: PKa, PKb, PKe, SKe



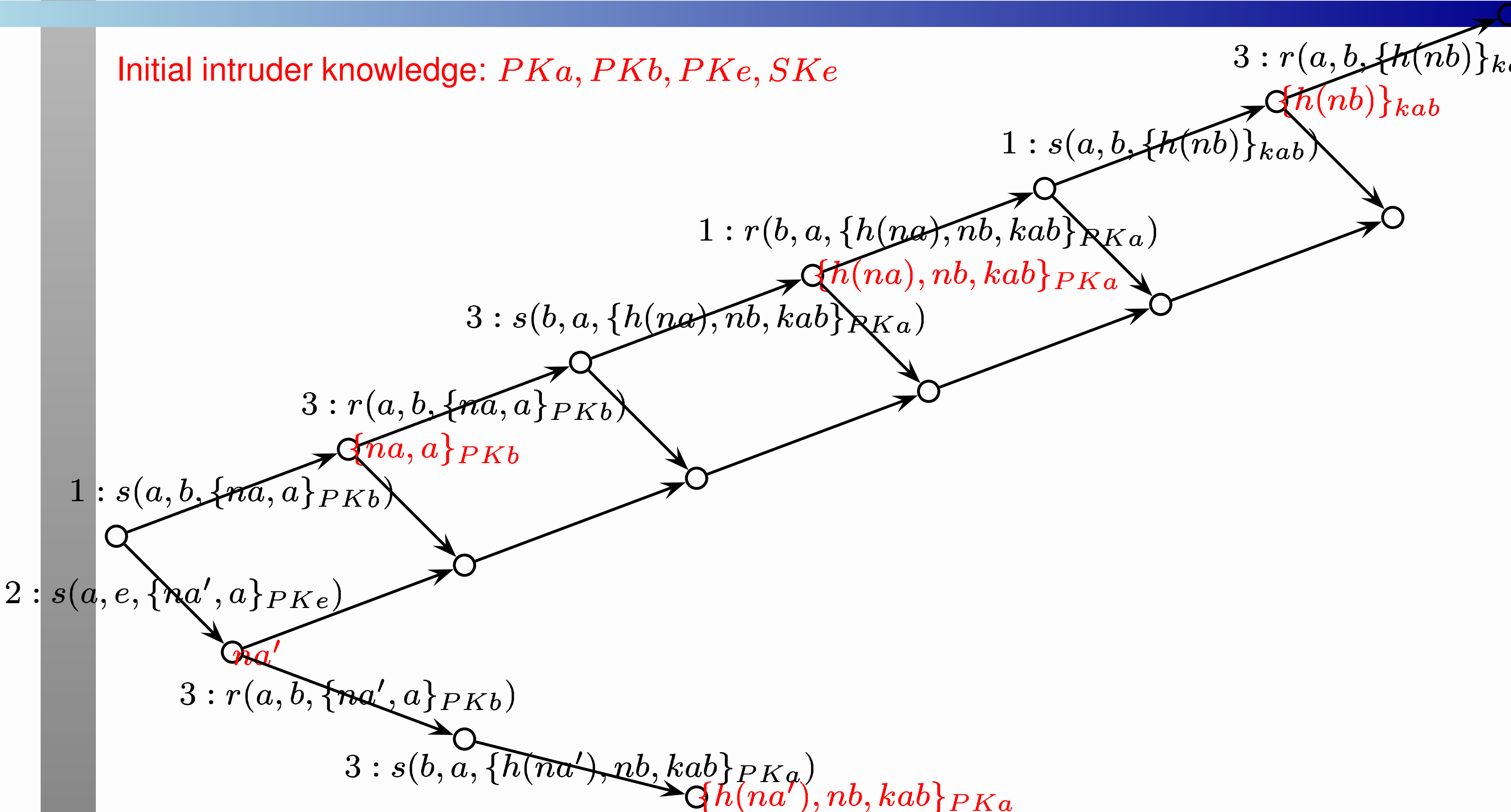
State space

Initial intruder knowledge: PKa, PKb, PKe, SKe



State space

Initial intruder knowledge: PKa, PKb, PKe, SKe



TU/e

Correctness criterion

A security protocol is correct w.r.t. secrecy if

- for every finite scenario,
- for every possible trace of that scenario, under control of the intruder,
- whenever an agent reaches a secrecy claim,
- the claimed secret will never occur in the intruder knowledge.

TU/e

Auxilliary definitions

match The match function determines whether the intruder can satisfy the required message format.

enabled An event is enabled if it is the first to be executed in a run and in case it is a read it must have a match with the intruder knowledge.

after The after function returns the new system state after executing a run.

General model checking algorithm

```
modelcheck (σ) =
if σ does not satisfy property then
    exit ("property fails") ;
else
    for all ev ∈ enabled(σ) do
        modelcheck(after(σ, ev))
    end
end
```

Correct if state space forms directed acyclic graph.

TU/e

traverseFull (runs,know,secrets)

```
if any secret in know then
|   exit ("attack") ;
else
|   for all ev ∈ enabled(runs, know) do
|   |   if ev = secret(m) then
|   |   |   traverseFull(after(runs, ev), know, secrets ∪ {m} ) ;
|   |   end
|   |   if ev = send(m) then
|   |   |   traverseFull(after(runs, ev), know ⊕ m, secrets ) ;
|   |   end
|   |   if ev = read(m) then
|   |   |   for all m' ∈ match(know, m) do
|   |   |   |   traverseFull(after(runs, read(m')), know, secrets ) ;
|   |   |   end
|   |   end
|   end
end
```

TU/e

traverseFull (runs,know,secrets)

if *any secret in know* **then**
    exit ("attack") ;
else
    for *all* $ev \in \texttt{enabled}(\textit{runs}, \textit{know})$ **do**
        if $ev =$ *secret(*m*)* **then**
            traverseFull($\texttt{after}$(runs, ev), know, secrets $\cup \{m\}$) ;
        end
        if $ev =$ *send(*m*)* **then**
            traverseFull($\texttt{after}$(runs, ev), know $\oplus\, m$, secrets) ;
        end
        if $ev =$ *read(*m*)* **then**
            for *all* $m' \in \texttt{match}($[illegible]$)$ **do**
                traverseFu[illegible](runs, $read(m')$), know, secrets) ;
            end
        end
    end
end

Correct but slow

traverseFull (runs,know,secrets)

```
if any secret in know then
    exit ("attack") ;
else
  Choose ev ∈ enabled(runs, know) do
        if ev = secret(m) then
            traverseFull(after(runs, ev), know, secrets ∪ {m} ) ;
        end
        if ev = send(m) then
            traverseFull(after(runs, ev), know ⊕ m, secrets ) ;
        end
        if ev = read(m) then
            for all m' ∈ match(know, m) do
                traverseFull(after(runs, read(m')), know, secrets ) ;
            end
        end
    end
end
```

TU/e

traverseFull (runs,know,secrets)

if *any secret in know* **then**
  exit ("attack") ;
else
  Choose $ev \in \texttt{enabled}(runs, know)$ **do**
    if $ev =$ *secret(m)* **then**
      traverseFull($\texttt{after}$(runs, ev), know, secrets $\cup \{m\}$) ;
    end
    if $ev =$ *send(m)* **then**
      traverseFull($\texttt{after}$(runs [illegible] w $\oplus\, m$, secrets) ;
    end
    if $ev =$ *read(m)* **th**[illegible]
      for *all* $m' \in$ [illegible]$(know, m)$ **do**
        traverseFu[illegible]($\texttt{after}$(runs, $read(m')$), know, secrets) ;
      end
    end
  end
end

Fast but incorrect

TU/e

General model checking algorithm with tail recursion

```
modelcheck (σ, except) =
if σ does not satisfy property then
  | exit ("property fails") ;
else
  | if enabled(σ) \ except ≠ ∅ then
  |   | ev = choose(enabled(σ) \ except) ;
  |   | modelcheck(after(σ, ev), ∅) ;
  |   | modelcheck(σ, except ∪ {ev})) ;
  | end
end
```

traverseFull2 (runs,know,secrets,except)

For-loop replaced by tail recursion

if *any secret in know* **then**
  exit ("attack") ;
else
  if $\mathtt{enabled2}(\textit{runs}, \textit{know}, \textit{except}) \neq \emptyset$ **then**
    $ev = \mathtt{choose}(\mathtt{enabled2}(\text{runs}, \text{know}, \text{except}))$;
    if $ev = secret(m)$ **then**
      traverseFull2($\mathtt{after}$(runs, ev), know, secrets $\cup \{m\}$, $\emptyset$) ;
      traverseFull2(runs, know, secrets, except $\cup \{ev\}$) ;
    end
    if $ev = send(m)$ **then**
      traverseFull2($\mathtt{after}$(runs, ev), know $\oplus\, m$, secrets, $\emptyset$) ;
      traverseFull2(runs, know, secrets, except $\cup \{ev\}$) ;
    end
    if $ev = read(m)$ **then**
      for *all* $m' \in \mathtt{match}(know, m)$ **do**
        traverseFull2($\mathtt{after}$(runs, $read(m')$), know, secrets, $\emptyset$) ;
      end
      traverseFull2(runs, know, secrets, except $\cup \{ev\}$) ;
    end
  end
end

traverseFull2 (runs,know,secrets,except)

$\mathtt{enabled2}(\text{runs}, \text{know}, \text{except}) = \mathtt{enabled}(\text{runs}, \text{know}) \setminus \text{except}$

```
if any secret in know then
  | exit ("attack") ;
else
  | if enabled2(runs, know, except) ≠ ∅ then
  |   | ev = choose(enabled2(runs, know, except)) ;
  |   | if ev = secret(m) then
  |   |   | traverseFull2(after(runs, ev), know, secrets ∪ {m}, ∅) ;
  |   |   | traverseFull2(runs, know, secrets, except ∪ {ev})
  |   | end
  |   | if ev = send(m) then
  |   |   | traverseFull2(after(runs, ev), know ⊕ m, secrets, ∅) ;
  |   |   | traverseFull2(runs, know, secrets, except ∪ {ev})
  |   | end
  |   | if ev = read(m) then
  |   |   | for all m′ ∈ match(know, m) do
  |   |   |   | traverseFull2(after(runs, read(m′)), know, secrets, ∅) ;
  |   |   | end
  |   |   | traverseFull2(runs, know, secrets, except ∪ {ev})
  |   | end
  | end
end
```

Partial order reduction

Lemma.
If at a given state closed events e and f from different runs can be executed, then.

- after executing event e, event f can still be executed;
- after executing event f, event e can still be executed;
- the states reached after ef and fe are both equal.

Example:

$e_1; e_2; e_3; send_1; send_2; e_4; e_5, \ldots$
$e_1; e_2; e_3; send_2; send_1; e_4; e_5, \ldots$
$e_1; e_2; e_3; send_2; e_4; send_1; e_5, \ldots$

All result in the same state, so we only have to traverse one of these.

TU/e

traverseFull2 (runs,know,secrets,except)

```
if any secret in know then
  | exit ("attack") ;
else
  | if enabled2(runs, know, except) ≠ ∅ then
  |   | ev = choose(enabled2(runs, know, except)) ;
  |   | if ev = secret(m) then
  |   |   | traverseFull2(after(runs, ev), know, secrets ∪ {m}, ∅) ;
  |   |   | traverseFull2(runs, know, secrets, except ∪ {ev})
  |   | end
  |   | if ev = send(m)  then
  |   |   | traverseFull2(after(runs, ev), know ⊕ m, secrets, ∅) ;
  |   |   | traverseFull2(runs, know, secrets, except ∪ {ev})
  |   | end
  |   | if ev = read(m)  then
  |   |   | for all m' ∈ match(know, m)  do
  |   |   |   | traverseFull2(after(runs, read(m')), know, secrets, ∅) ;
  |   |   | end
  |   |   | traverseFull2(runs, know, secrets, except ∪ {ev})
  |   | end
  | end
end
```

TU/e

traverseFull2 (runs,know,secrets,except)

if *any secret in know* **then**
    exit ("attack") ;
else
    if $\mathtt{enabled}2(runs, know, except) \neq \emptyset$ **then**
        $ev = \mathtt{choose}(\mathtt{enabled}2(\mathrm{runs, know, except}))$;
        if $ev = secret(m)$ **then**
            traverseFull2($\mathtt{after}$(runs, ev), know, secrets $\cup \{m\}$, $\emptyset$) ;
            traverseFull2(runs, know, secrets, except $\cup \{ev\}$) ;
        end
        if $ev = send(m)$ **then**
            traverseFull2($\mathtt{after}$(runs, ev), know $\oplus\, m$, secrets, $\emptyset$) ;
            traverseFull2(runs, know, secrets, except $\cup \{ev\}$)
        end
        if $ev = read(m)$ **then**
            for *all* $m' \in \mathtt{match}(know, m)$ **do**
                traverseFull2($\mathtt{after}$(runs, $read(m')$), know, secrets, $\emptyset$) ;
            end
            traverseFull2(runs, know, secrets, except $\cup \{ev\}$)
        end
    end
end

Not needed

TU/e

traverseFull2 (runs,know,secrets,except)

if *any secret in know* **then**
    exit ("attack") ;
else
    if $\texttt{enabled}2(runs, know, except) \neq \emptyset$ **then**
        $ev = \texttt{choose}(\texttt{enabled}2(\text{runs, know, except}))$;
        if $ev = secret(m)$ **then**
            traverseFull2($\texttt{after}$(runs, ev), know, secrets $\cup \{m\}$, $\emptyset$) ;
            traverseFull2(runs, know, secrets, except $\cup \{ev\}$)
        end
        if $ev = send(m)$ **then**
            traverseFull2($\texttt{after}$(runs, ev), know $\oplus\, m$, secrets, $\emptyset$) ;
            traverseFull2(runs, know, secrets, except $\cup \{ev\}$) ;

Not needed

        end
        if $ev = read(m)$ **then**
            for *all* $m' \in \texttt{match}(know, m)$ **do**
                traverseFull2($\texttt{after}$(runs, $read(m')$), know, secrets, $\emptyset$) ;
            end
            traverseFull2(runs, know, secrets, except $\cup \{ev\}$)
        end
    end
end

traverseFull2 (runs,know,secrets,except)

Reduced

if *any secret in know* **then**
    exit ("attack") ;
else
    if $\mathtt{enabled2}(\mathit{runs}, \mathit{know}, \mathit{except}) \neq \emptyset$ **then**
        $ev = \mathtt{choose}(\mathtt{enabled2}(\text{runs}, \text{know}, \text{except}))$;
        if $ev = secret(m)$ **then**
            traverseFull2($\mathtt{after}$(runs, ev), know, secrets $\cup \{m\}$, $\emptyset$) ;
            traverseFull2(runs, know, secrets, except $\cup \{ev\}$)
        end
        if $ev = send(m)$ **then**
            traverseFull2($\mathtt{after}$(runs, ev), know $\oplus\, m$, secrets, $\emptyset$) ;
            traverseFull2(runs, know, secrets, except $\cup \{ev\}$)
        end
        if $ev = read(m)$ **then**
            for *all* $m' \in \mathtt{match}(know, m)$ **do**
                traverseFull2($\mathtt{after}$(runs, $read(m')$), know, secrets, $\emptyset$) ;
            end
            traverseFull2(runs, know, secrets, except $\cup \{ev\}$) ;
        end
    end
end

traverse (runs,know,secrets,forbidden)

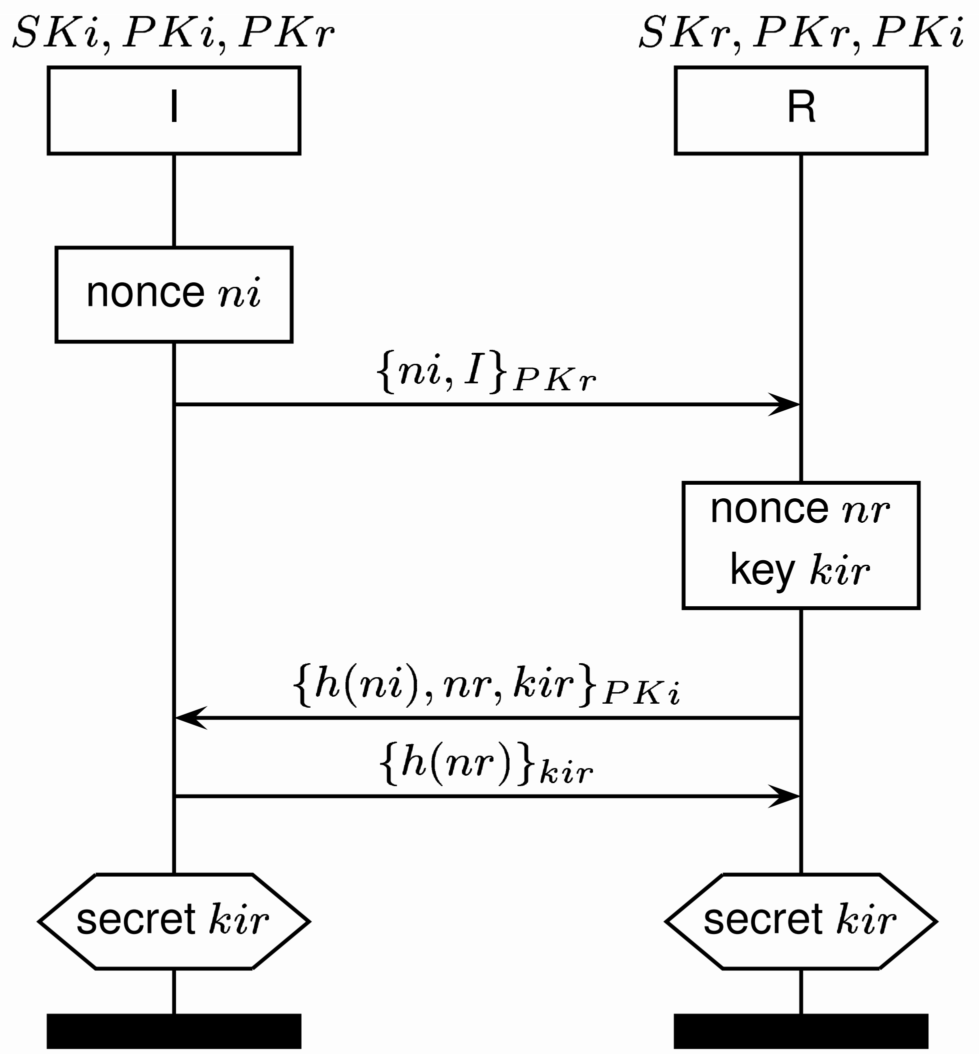
```
if any secret in know then
    exit ("attack") ;
else
    if enabled3(runs, know, forbidden) ≠ ∅ then
        ev = choose(enabled3(runs, know, forbidden)) ;
        if ev = secret(m) then
            traverse(after(runs, ev), know, secrets ∪ {m}, forbidden) ;
        end
        if ev = send(m) then
            traverse(after(runs, ev), know ⊕ m, secrets, forbidden) ;
        end
        if ev = read(m) then
            for all m' ∈ match(know, m) ∧ m' ∉ forbidden(read(m)) do
                traverse(after(runs, read(m')), know, secrets, forbidden) ;
            end
            traverse(runs, know, secrets, forbidden[ev → know]) ;
        end
    end
end
```

traverse (runs,know,secrets,forbidden)

$\texttt{enabled3}(\text{runs}, \text{know}, \text{forbidden}) =$
$\{ev \in \texttt{enabled}(\text{runs}, \text{know}) \mid ev = read(m) \Rightarrow$
$\exists_{m' \in \texttt{match}(know,m)} m' \notin \text{forbidden}(ev)\}$

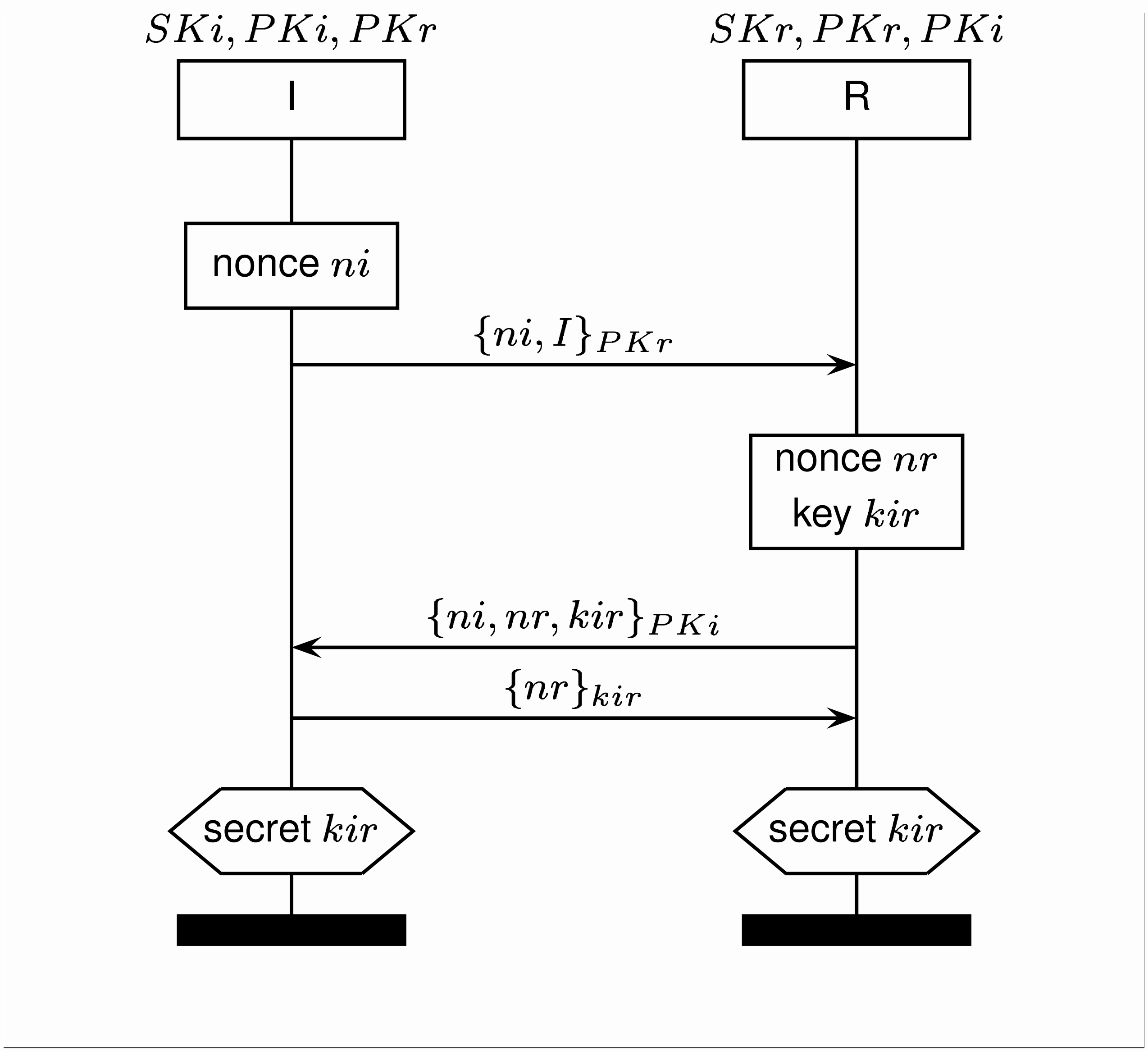
if *any secret in know* **t**
  exit ("attack") ;
else
  if $\texttt{enabled3}(runs, know, forbidden) \neq \emptyset$ **then**
    $ev = \texttt{choose}(\texttt{enabled3}(\text{runs}, \text{know}, \text{forbidden}))$;
    if $ev = secret(m)$ **then**
      traverse($\texttt{after}$(runs, ev), know, secrets $\cup \{m\}$, forbidden) ;
    end
    if $ev = send(m)$ **then**
      traverse($\texttt{after}$(runs, ev), know $\oplus\, m$, secrets, forbidden) ;
    end
    if ev = *read*(m) **then**
      for *all* $m' \in \texttt{match}(know, m) \wedge m' \notin$ *forbidden*$(read(m))$ **do**
        traverse($\texttt{after}$(runs, $read(m')$), know, secrets, forbidden) ;
      end
      traverse(runs, know, secrets, forbidden$[ev \to$ know$]$) ;
    end
  end
end

BKE

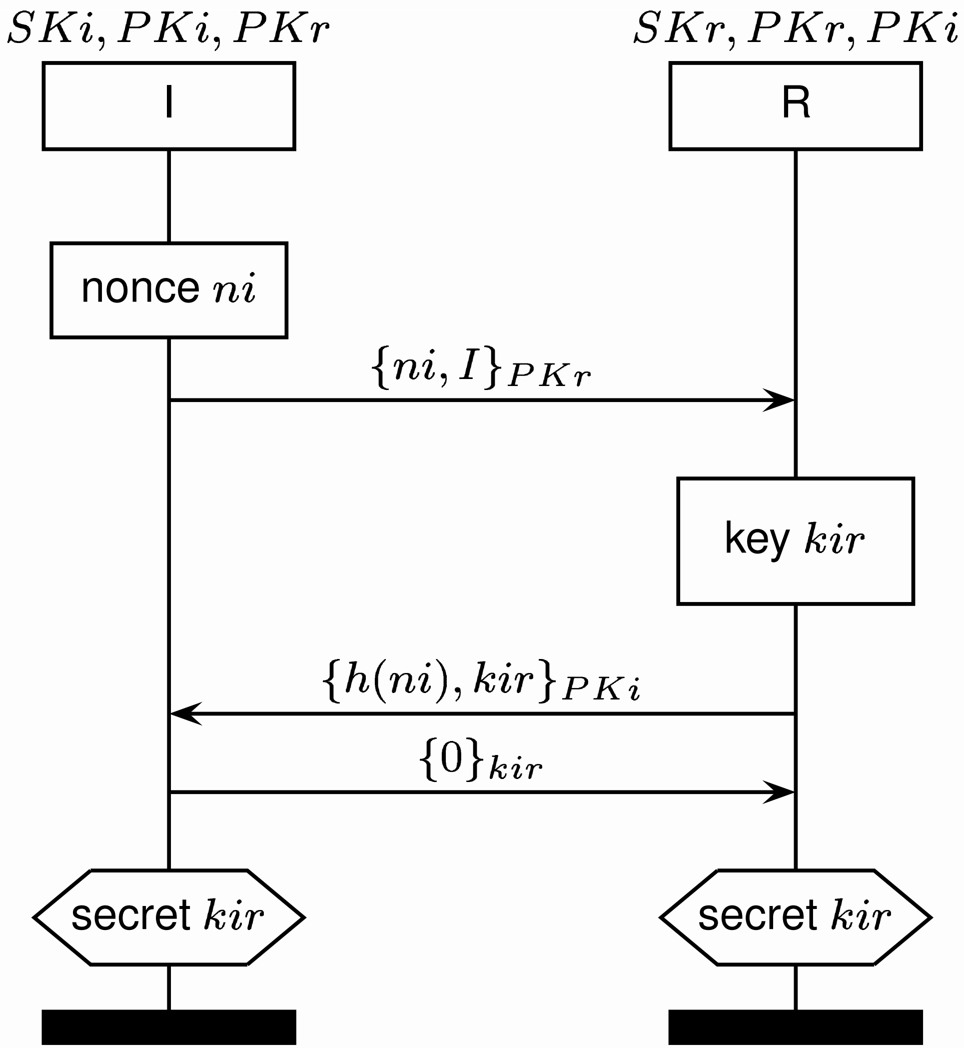


SKi, PKi, PKr
SKr, PKr, PKi
I
R
nonce ni
$\{ni, I\}_{PKr}$
nonce nr
key kir
$\{h(ni), nr, kir\}_{PKi}$
$\{h(nr)\}_{kir}$
secret kir
secret kir

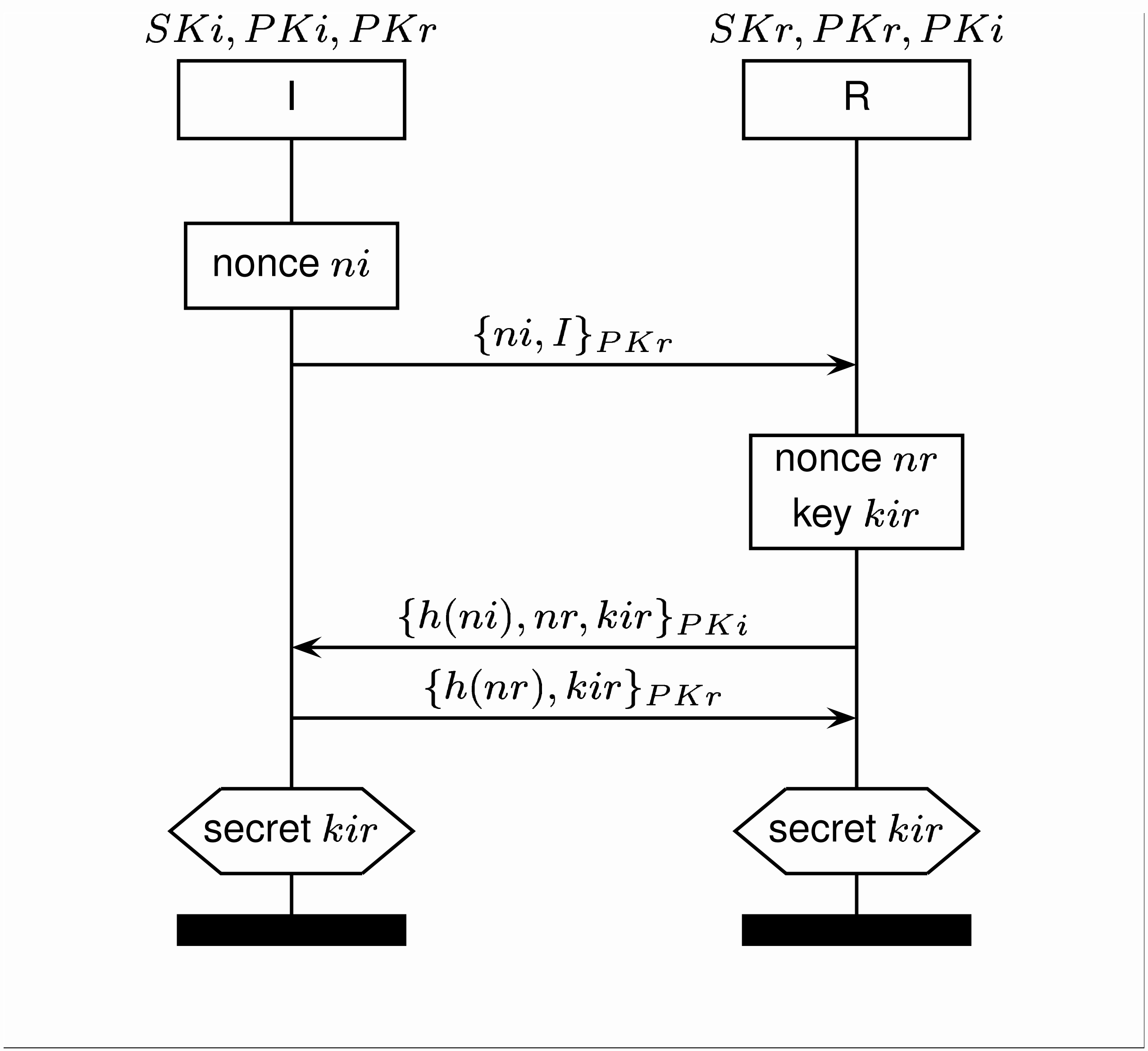
BKE without hash



BKE without nr

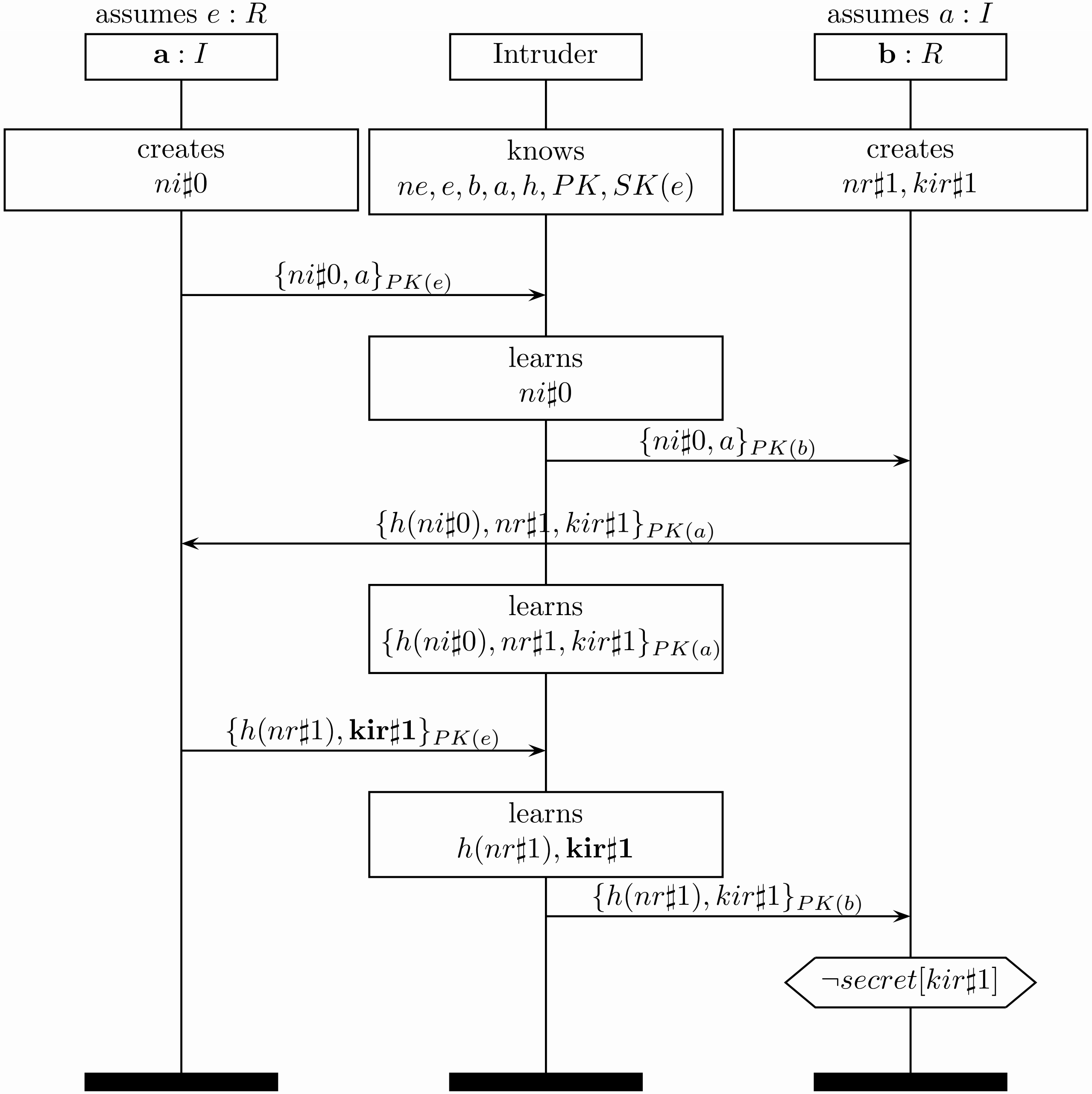


BKE kir within encryption

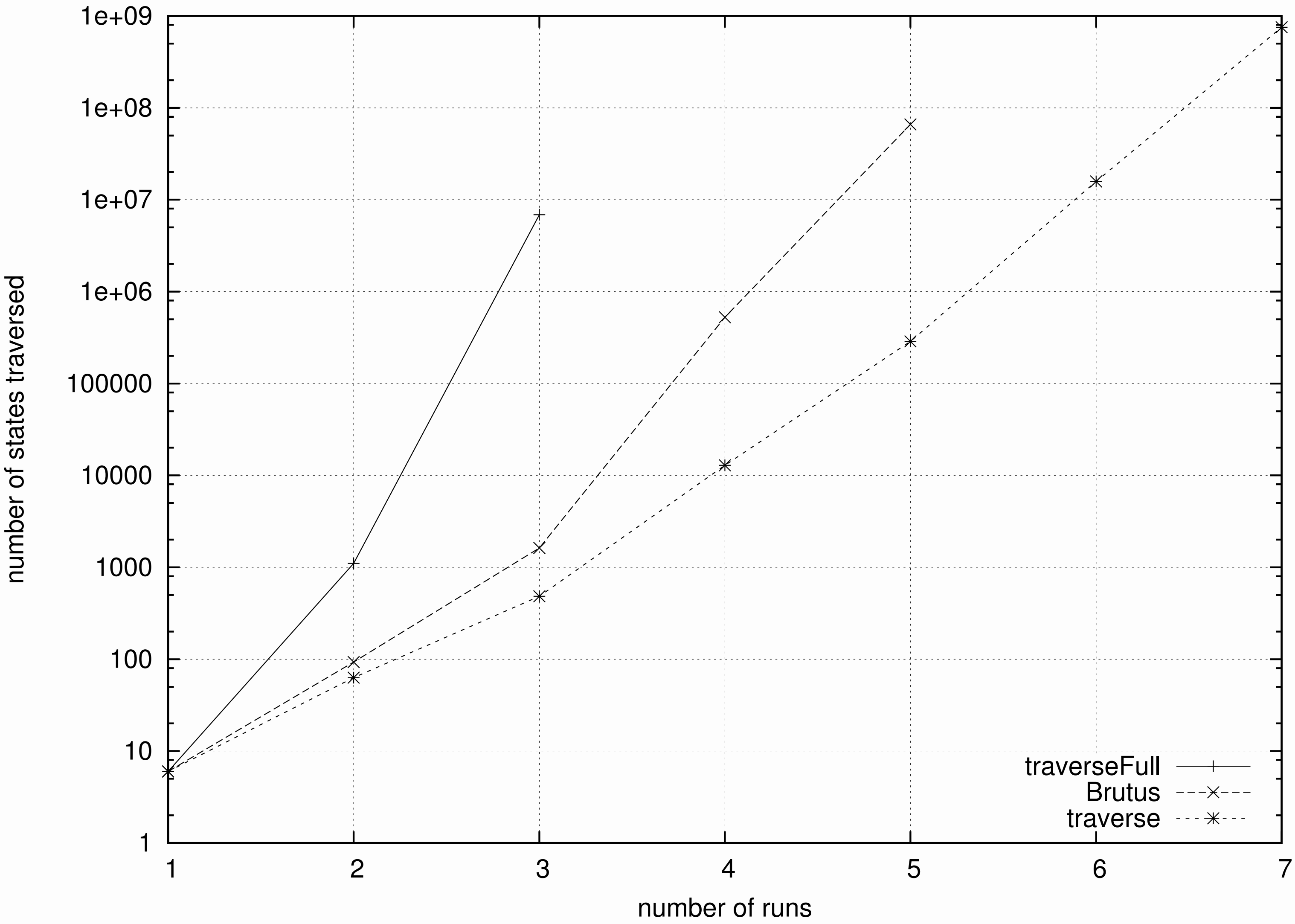


TU/e

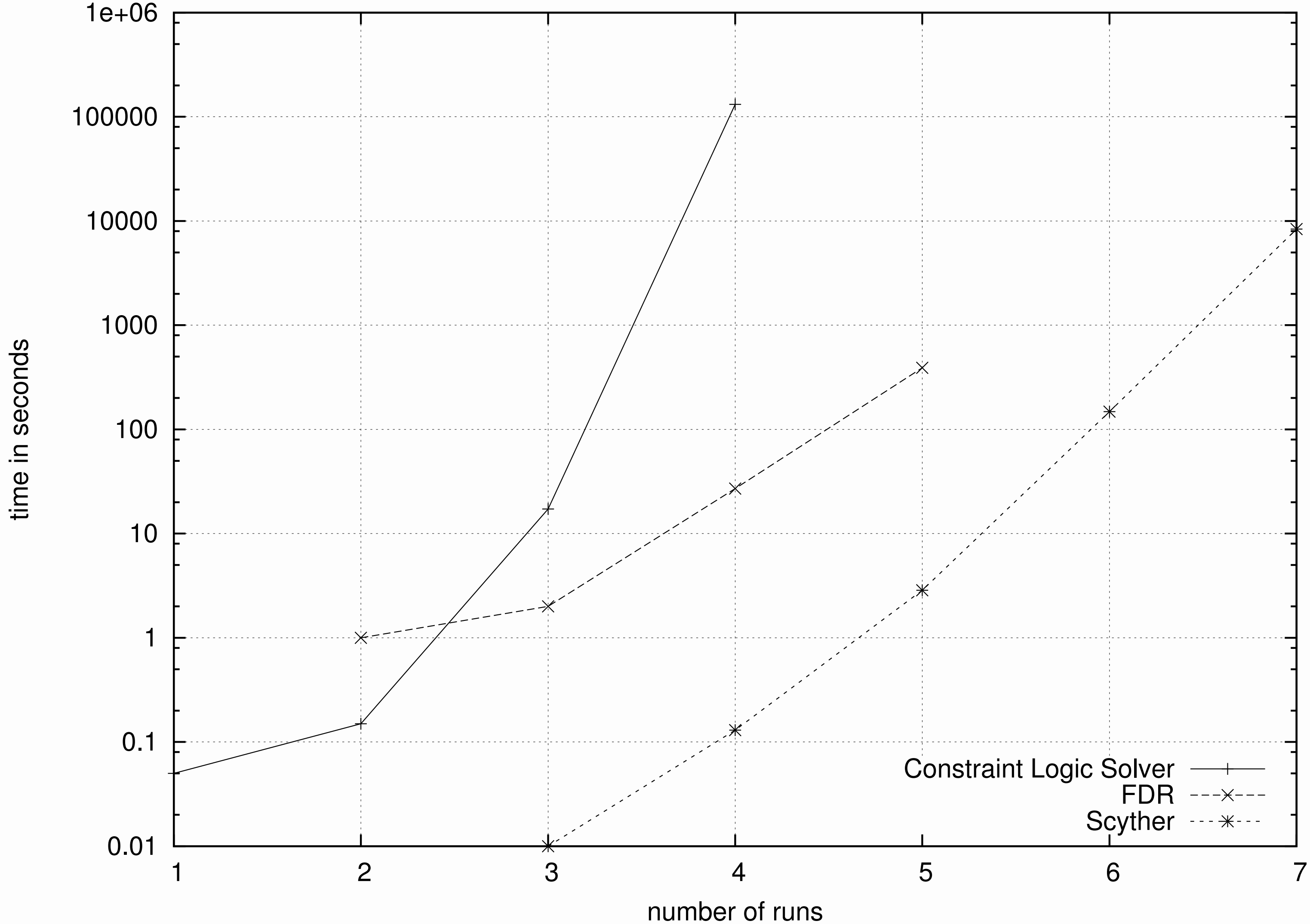
Attack visualization



Tool comparison: number of states



Tool comparison: execution time



TU/e

Conclusions

- Fastest algorithm that we know of (only basic type flaw attacks).
- Tool produces visual attack trees.
- Possible improvements:
  1. Exploit symmetry in scenario's.
  2. Combine with Constraint Logic approach $\Rightarrow$ hybrid model checker.
- Extend algorithm to different intruder models.
- Similar algorithm for authentication properties.