# HLPSL Code Introduction

## Am Beispiel des Needham-Schroeder-Protokoll

André Karge

Bauhaus-Universität Weimar

17. Juni 2014

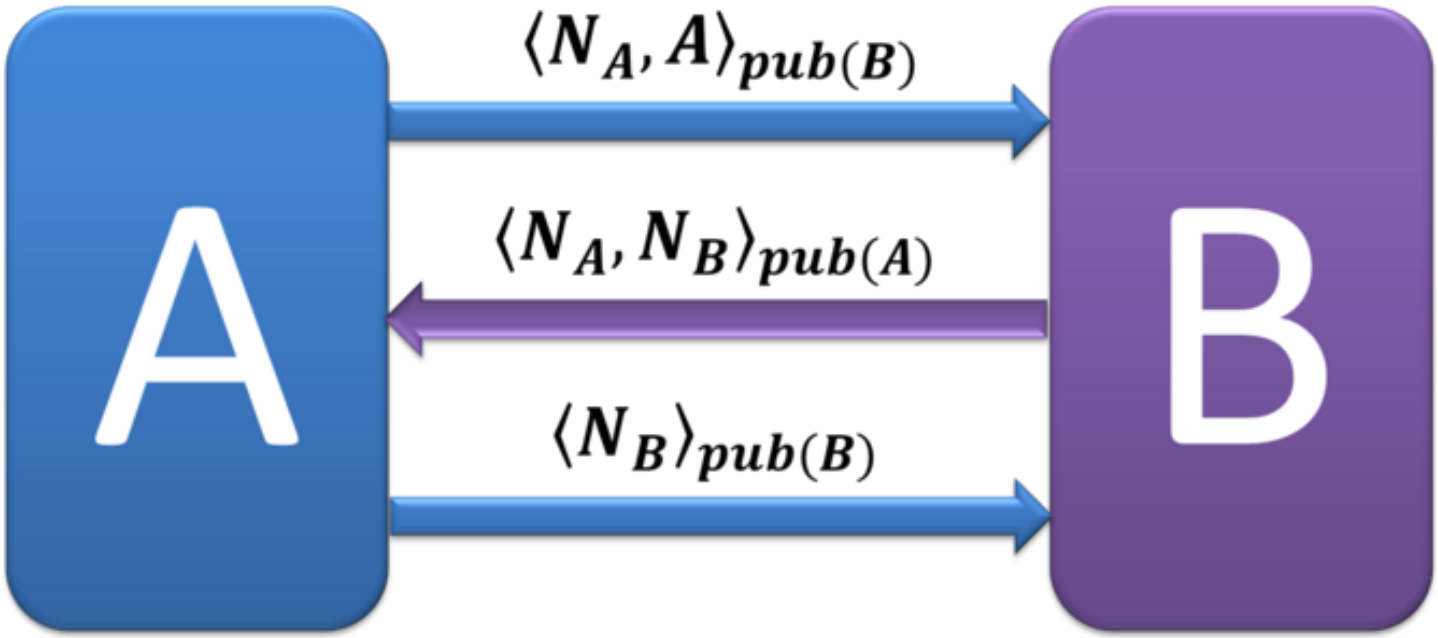
# Agenda

1. Grundaufbau
2. Konventionen
3. Aufbau eines role-Objektes einer beteiligten Person
4. Aufbau eines role-Objektes einer Session
5. Aufbau des Enviroments

HLPSL Dokument besteht aus:

- Rollen (zu vergleichen mit Klassen)
  - Personen (Alice, Bob, Charles)
  - Sessions
  - Enviroment (Protokollumgebung - zu vergleichen mit Main-Funktion)
- Sicherheitszielsetzung (Was soll der Proofer überprüfen)
- Aufruf der Umgebung

# Needham-Schroeder Protokoll



## Grundgerüst Needham-Shroeder

```
role alice          % Beteiligte Person 1
   ...
end role
role bob            % Beteiligte Person 2
   ...
end role
role session        % Session eines Durchlaufs
   ...
end role
role enviroment     % Testumgebung
   ...
end role
goal                % Angabe zu pruefender
   ...              % Parameter
end goal
enviroment()        % Aufruf der Testumgebung
```

- Variablen beginnen mit Großbuchstaben
- Konstanten beginnen mit Kleinbuchstaben
- Variablen müssen getypt sein
- Kennzeichnung von Sicherheitsparametern in der Rolle, in der diese erzeugt werden

```
role alice(
    A, B : agent,
    Ka, Kb : public_key,
    SND, RCV : channel(dy)
  )
  played_by A def=
  local State : nat, Na, Nb : text
  init State := 0
  transition
  0. State = 0 /\ RCV(start)
    =|>
    State' := 2 /\ Na' := new() /\ SND({Na'.A}_Kb)
    /\ secret(Na',na,{A,B})
  2. State = 2 /\ RCV({Na.Nb'}_Ka)
    =|>
    State' := 4 /\ SND({Nb'}_Kb)
end role
```

```
role alice(
    A, B : agent,
    Ka, Kb : public_key,
    SND, RCV : channel(dy)
  )
  ...
end role
```

- Definition von Protokollparametern, die Alice bekannt sind
- Ka und Kb sind öffentliche Schlüssel
- SND und RCV sind Kommunikationskanäle
- channel(dy) steht für Intruder Model (dy = Dolev-Yao)

```
role alice
  ...
  played_by A def=
  local State : nat,
  Na, Nb : text
  init State := 0
  ...
end role
```

- A bezeichnet den Beteiligten, den Alice spielt (alice played by A)
- dazu werden weitere Parameter definiert
- local State : nat = lokale natürliche Zahl von Alice mit Namen State
- Na und Nb sind Texte
- State wird mit 0 initialisiert

```
role alice
  ...
  transition
  0. State = 0 /\ RCV(start) =|>
    State' := 2 /\ Na' := new() /\ SND({Na'.A}_Kb)
    /\ secret(Na',na,{A,B})
  2. State = 2 /\ RCV({Na.Nb'}_Ka) =|> State' := 4
    /\ SND({Nb'}_Kb)
end role
```

- Spezifikation von Transitionen, die von Alice ausgeführt werden
- 0. und 2. sind die Namen der Transitionen (können bel. gewählt werden)
- Eine Transition wird erst ausgeführt, wenn bestimmte Bedingungen erfüllt sind

```
role alice
  ...
  0. State = 0 /\ RCV(start) =|>
    State' := 2 /\ Na' := new() /\ SND({Na'.A}_Kb)
    /\ secret(Na',na,{A,B})
  ...
end role
```

- 0. Wenn State = 0 und auf dem RCV channel der Startbefehl liegt, führe folgendes aus:
- erstelle neue Nonce und belege den SND channel mit dieser und der Signatur von A verschlüsselt mit dem public key von Bob
- dazu werden Sicherheitsziele angegeben: Na' muss zwischen A und B geheim gehalten werden (na ist protocoll id und wird später im enviroment deklariert)

```
role alice
   ...
   2. State = 2 /\ RCV({Na.Nb'}_Ka) =|>
      State' := 4 /\ SND({Nb'}_Kb)
end role
```

- 2. Wenn State = 2 und auf dem RCV channel eine Message mit 2 Noncen verschlüsselt mit dem public key von alice liegt, führe folgendes aus:
- setze State auf 4 und schreibe die Nonce von b verschlüsselt mit dem public key von b auf den SND channel

```
role bob(
        A, B : agent,
        Ka, Kb : public_key,
        SND, RCV : channel(dy)
    )
    played_by B def=
    local State : nat, Na, Nb : text
    init State := 1
    transition
    1. State = 1 /\ RCV({Na'.A}_Kb)
      =|>
      State' := 3 /\ Nb' := new() /\ SND({Na'.Nb'}_Ka)
      /\ secret(Nb',nb,{A,B})
    3. State = 3 /\ RCV({Nb'}_Kb)
      =|>
      State' := 4
end role
```

```
role session(
        A, B : agent,
        Ka, Kb : public_key
    )
    def=
    local SA, RA, SB, RB : channel(dy)
    composition
    alice(A,B,Ka,Kb,SA,RA) /\ bob(A,B,Ka,Kb,SB,RB)
end role
```

```
role session(
        A, B : agent,
        Ka, Kb : public_key
    )
    ...
end role
```

- A und B sind beteiligte einer Session
- Ka und Kb sind öffentliche Schlüssel

```
role session
    ...
    def=
    local SA, RA, SB, RB : channel(dy)
    composition
    alice(A,B,Ka,Kb,SA,RA) /\ bob(A,B,Ka,Kb,SB,RB)
end role
```

- lokale Variablen einer Session sind die jeweiligen Kanäle von A und B
- composition bedeutet, dass die Rollen initialisiert werden
- alice und bob werden jeweils mit allen notwendigen Parametern initialisiert

```
role enviroment() def=
    const a, b : agent,
    ka, kb, ki : public_key,
    na, nb : protocol_id
    intruder_knowledge = {a, b, ka, kb, ki, inv(ki)}
    composition
    session(a,b,ka,kb) /\ session(a,i,ka,ki)
      /\ session(i,b,ki,kb)
end role
```

```
role enviroment() def=
    const a, b : agent,
    ka, kb, ki : public_key,
    na, nb : protocol_id
    ...
end role
```

- Enviroment ist die Top-Level Rolle (mit globalen Einschränkungen und einer oder mehreren Sessions)
- a und b sind die beteiligten des Protokolls
- ka, kb und ki sind öffentliche Schlüssel
- na und nb sind die Protokoll IDs der Sicherheitsziele (Verweisen auf die kritischen Variablen der Beteiligten)

```
role enviroment() def=
    ...
    intruder_knowledge = {a, b, ka, kb, ki, inv(ki)}
    composition
    session(a,b,ka,kb) /\ session(a,i,ka,ki)
      /\ session(i,b,ki,kb)
end role
```

- intruder_knowledge ist alles, was Eve wissen kann
- sie kennt a und b, deren öffentliche Schlüssel, ihren eigenen Schlüssel und dessen Inverses zum entschlüsseln
- danach werden Sessions initialisiert, wobei Eve auch einen legitimen Nutzer spielt

```
goal
     secrecy_of na, nb
end goal

enviroment()
```

- goal gibt an, was der Proofer prüfen soll
- es soll getestet werden, ob na und nb vor dem Eingriff eines Intruders sicher sind
- enviroment wird am ende des codes aufgerufen, um das Szenario durchzuspielen

## Intruder Model dy

- dy = Dolev-Yao
- Zur Zeit das einzige Model für Avispa
- Intruder hat volle Kontrolle über das Netzwerk
- Er kann Nachrichten Mitschneiden, Analysieren oder Modifizieren (solang er die Schlüssel kennt)
- Er kann sich beliebige Messages erstellen und an egal wen verschicken um sich als beteiligte Person auszugeben