

# Development of Security Protocols by Refinement

Christoph Sprenger and Ivano Somaini  
ETH Zurich, Switzerland

April 10, 2026

## **Abstract**

We propose a development method for security protocols based on stepwise refinement. Our refinement strategy transforms abstract security goals into protocols that are secure when operating over an insecure channel controlled by a Dolev-Yao-style intruder. As intermediate levels of abstraction, we employ messageless guard protocols and channel protocols communicating over channels with security properties. These abstractions provide insights on why protocols are secure and foster the development of families of protocols sharing common structure and properties. We have implemented our method in Isabelle/HOL and used it to develop different entity authentication and key establishment protocols, including realistic features such as key confirmation, replay caches, and encrypted tickets. Our development highlights that guard protocols and channel protocols provide fundamental abstractions for bridging the gap between security properties and standard protocol descriptions based on cryptographic messages. It also shows that our refinement approach scales to protocols of nontrivial size and complexity.

# Contents

<b>1</b>	<b>Protocol Modeling and Refinement Infrastructure</b>	<b>5</b>
1.1	Proving infrastructure	5
1.1.1	Prover configuration	5
1.1.2	Forward reasoning ("attributes")	5
1.1.3	General results	6
1.2	Models, Invariants and Refinements	7
1.2.1	Specifications, reachability, and behaviours.	7
1.2.2	Invariants	11
1.2.3	Refinement	13
1.3	Atomic messages	19
1.3.1	Agents	19
1.3.2	Nonces	20
1.4	Symmetric and Assymmetric Keys	20
1.4.1	Asymmetric Keys	21
1.4.2	Basic properties of <i>pubK</i> and <i>priK</i>	21
1.4.3	"Image" equations that hold for injective functions	22
1.4.4	Symmetric Keys	22
1.5	Atomic messages	23
1.5.1	Atoms datatype	23
1.5.2	Long-term key setup (abstractly)	23
1.6	Protocol runs	24
1.6.1	Runs	24
1.6.2	Run abstraction	24
1.7	Channel Messages	25
1.7.1	Channel messages	25
1.7.2	Keys used in dynamic channel messages	26
1.7.3	Atoms in a set of channel messages	26
1.7.4	Intruder knowledge (atoms)	27
1.7.5	Faking messages	29
1.8	Theory of Agents and Messages for Security Protocols	30
1.8.1	keysFor operator	31
1.8.2	Inductive relation "parts"	32
1.8.3	Inductive relation "analz"	34
1.8.4	Inductive relation "synth"	38
1.8.5	HPair: a combination of Hash and MPair	40
1.9	Secrecy with Leaking (global version)	43

1.9.1	State . . . . .	43
1.9.2	Invariant definitions . . . . .	43
1.9.3	Events . . . . .	44
1.9.4	Specification . . . . .	45
1.9.5	Invariant proofs . . . . .	45
1.9.6	inv1: Secrecy . . . . .	45
1.9.7	inv2: Authorized and leaked data is known to someone . . . . .	46
1.10	Non-injective Agreement . . . . .	46
1.10.1	State . . . . .	46
1.10.2	Events . . . . .	47
1.10.3	Invariants . . . . .	48
1.10.4	inv1: non-injective agreement . . . . .	48
1.11	Injective Agreement . . . . .	49
1.11.1	State . . . . .	49
1.11.2	Events . . . . .	49
1.11.3	Invariants . . . . .	51
1.11.4	Refinement . . . . .	51
1.11.5	Derived invariants . . . . .	52
<b>2</b>	<b>Unidirectional Authentication Protocols</b>	<b>53</b>
2.1	Refinement 1: Abstract Protocol . . . . .	53
2.1.1	State . . . . .	53
2.1.2	Events . . . . .	53
2.1.3	Simulation relation . . . . .	55
2.1.4	Refinement . . . . .	56
2.2	Refinement 2a: Authentic Channel Protocol . . . . .	57
2.2.1	State . . . . .	57
2.2.2	Events . . . . .	57
2.2.3	Invariants . . . . .	59
2.2.4	Refinement . . . . .	60
2.3	Refinement 2b: Confidential Channel Protocol . . . . .	61
2.3.1	State and observations . . . . .	61
2.3.2	Events . . . . .	62
2.3.3	Invariants . . . . .	63
2.3.4	Refinement . . . . .	65
2.4	Refinement 3a: Signature-based Dolev-Yao Protocol (Variant A) . . . . .	66
2.4.1	State . . . . .	67
2.4.2	Events . . . . .	67
2.4.3	Invariants . . . . .	69
2.4.4	Refinement . . . . .	71
2.5	Refinement 3b: Encryption-based Dolev-Yao Protocol (Variant A) . . . . .	74
2.5.1	State and observations . . . . .	74
2.5.2	Events . . . . .	75
2.5.3	Invariants . . . . .	76
2.5.4	Simulation relation . . . . .	77
2.5.5	Misc lemmas . . . . .	78
2.5.6	Refinement proof . . . . .	79

<b>3</b>	<b>Key Establishment Protocols</b>	<b>81</b>
3.1	Basic abstract key distribution (L1)	81
3.1.1	State	81
3.1.2	Events	84
3.1.3	Specification	86
3.1.4	Invariants	86
3.1.5	Refinement of $s0g$	87
3.1.6	Derived invariants	89
3.2	Abstract (i/n)-authenticated key transport (L1)	89
3.2.1	State	89
3.2.2	Events	90
3.2.3	Specification	91
3.2.4	Invariants	92
3.2.5	Refinement of $m1x$	92
3.2.6	Refinement of $a0i$ for initiator/server	95
3.2.7	Refinement of $a0n$ for responder/server	99
3.3	Abstract (n/n)-authenticated key transport (L1)	102
3.3.1	State	102
3.3.2	Events	103
3.3.3	Specification	104
3.3.4	Invariants	105
3.3.5	Refinement of $m1x$	105
3.3.6	Refinement of $a0n$ for initiator/server	107
3.3.7	Refinement of $a0n$ for responder/server	110
3.4	Abstract Kerberos core protocol (L1)	113
3.4.1	State	114
3.4.2	Events	114
3.4.3	Specification	117
3.4.4	Invariants	117
3.4.5	Refinement of $m1a$	118
3.4.6	Refinement of $a0i$ for responder/initiator	122
3.4.7	Refinement of $a0i$ for initiator/responder	127
3.5	Abstract Kerberos core protocol (L2)	131
3.5.1	State	132
3.5.2	Events	132
3.5.3	Transition system	135
3.5.4	Invariants and simulation relation	136
3.5.5	Refinement	143
3.5.6	Inherited and derived invariants	145
3.6	Core Kerberos, "parallel" variant (L3)	146
3.6.1	Setup	146
3.6.2	State	146
3.6.3	Events	147
3.6.4	Transition system	150
3.6.5	Invariants	151
3.6.6	Refinement	152
3.6.7	Inherited invariants	156

3.7	Core Kerberos 5 (L3)	157
3.7.1	Setup	157
3.7.2	State	158
3.7.3	Events	158
3.7.4	Transition system	161
3.7.5	Invariants	162
3.7.6	Refinement	163
3.7.7	Inherited invariants	167
3.8	Core Kerberos 4 (L3)	168
3.8.1	Setup	168
3.8.2	State	169
3.8.3	Events	169
3.8.4	Transition system	172
3.8.5	Invariants	173
3.8.6	Refinement	176
3.8.7	Inherited invariants	181
3.9	Abstract Needham-Schroeder Shared Key (L1)	181
3.9.1	State	182
3.9.2	Events	182
3.9.3	Specification	184
3.9.4	Invariants	185
3.9.5	Refinement of $m1a$	185
3.9.6	Refinement of $a0i$ for initiator/responder	189
3.9.7	Refinement of $a0i$ for responder/initiator	193
3.10	Abstract Needham-Schroeder Shared Key (L2)	197
3.10.1	State	197
3.10.2	Events	198
3.10.3	Transition system	200
3.10.4	Invariants	201
3.10.5	Refinement	212
3.10.6	Inherited and derived invariants	214
3.11	Needham-Schroeder Shared Key, "parallel" variant (L3)	214
3.11.1	Setup	215
3.11.2	State	215
3.11.3	Events	216
3.11.4	Transition system	218
3.11.5	Invariants	219
3.11.6	Refinement	221
3.11.7	Inherited invariants	225
3.12	Needham-Schroeder Shared Key (L3)	225
3.12.1	Setup	226
3.12.2	State	226
3.12.3	Events	226
3.12.4	Transition system	229
3.12.5	Invariants	230
3.12.6	Refinement	233
3.12.7	Inherited invariants	237

3.13	Abstract Denning-Sacco protocol (L1)	238
3.13.1	State	238
3.13.2	Events	238
3.13.3	Specification	239
3.13.4	Invariants	240
3.13.5	Refinement of $m1a$	241
3.14	Abstract Denning-Sacco protocol (L2)	242
3.14.1	State	243
3.14.2	Events	243
3.14.3	Transition system	246
3.14.4	Invariants and simulation relation	246
3.14.5	Refinement	251
3.14.6	Inherited and derived invariants	252
3.15	Denning-Sacco, direct variant (L3)	253
3.15.1	Setup	253
3.15.2	State	253
3.15.3	Events	253
3.15.4	Transition system	256
3.15.5	Invariants	256
3.15.6	Refinement	258
3.16	Denning-Sacco protocol (L3)	261
3.16.1	Setup	262
3.16.2	State	262
3.16.3	Events	262
3.16.4	Transition system	265
3.16.5	Invariants	265
3.16.6	Refinement	268

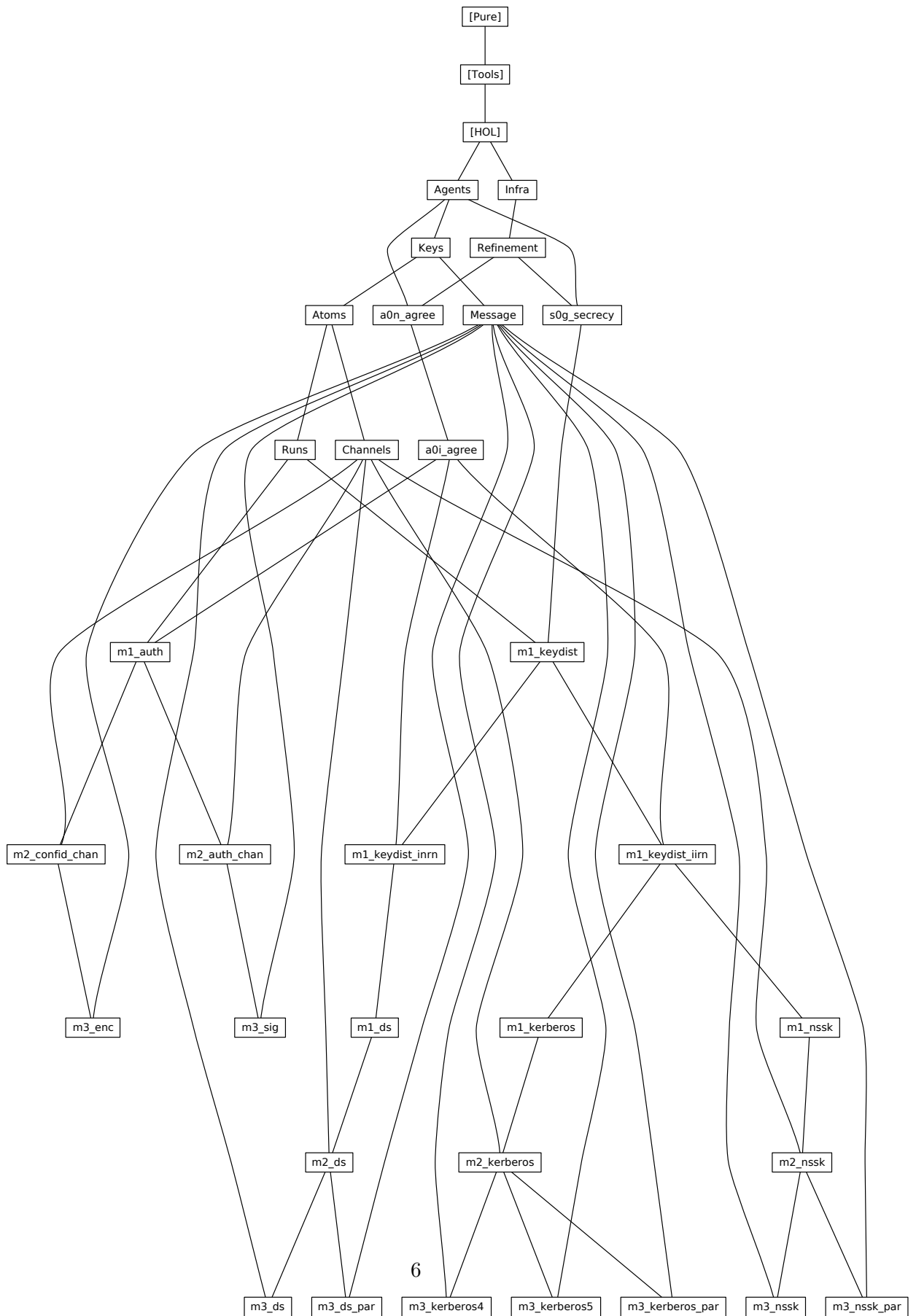


Figure 1: Theory dependencies

## Preamble

### Related Publications

The following papers describe our results in more detail:

- Christoph Sprenger and David Basin, *Developing Security Protocols by Refinement*, CCS 2010.
- Christoph Sprenger and David Basin, *Refining Key Establishment*, CSF 2012.
- Christoph Sprenger and David Basin, *Refining Security Protocols*, Journal of Computer Security (in submission), 2017.

Note: The Isabelle/HOL sources in this distribution also include the treatment of session key compromise. This is described in our journal paper (see above), which subsumes the CCS 2010 and CSF 2012 papers.

### Mapping the model names in our papers to the Isabelle/HOL theories

For the sake of the presentation, the papers use shorter names for the models than the Isabelle theories. Here is a mapping of the names. On the left you find the model name used in the papers and on the right the corresponding Isabelle/HOL theory name. Note that the Isabelle theories contain a separate lemma or theorem for each invariant and refinement result.

#### Level 0

	Refinement/
s0	s0g_secrecy
a0n	a0n_agree
a0i	a0i_agree

#### Level 1

	Auth_simple/
a1	m1_auth

	Key_establish/
kt1	m1_keydist
kt1in	m1_keydist_iirn
kt1nn	m1_keydist_inrn
nssk1	m1_nssk
krb1	m1_kerberos
ds1	m1_ds

#### Level 2

	Auth_simple/
a2	m2_auth_chan
c2	m2_confid_chan

	Key_establish/
nssk2	m2_nssk
krb2	m2_kerberos
ds2	m2_ds

### Level 3

	Auth_simple/
iso3	m3_sig
nsl3	m3_enc

	Key_establish/
nssk3d	m3_nssk_par
nssk3	m3_nssk
krb3d	m3_kerberos_par
krb3v	m3_kerberos5
krb3iv	m3_kerberos4
ds3d	m3_ds_par
ds3	m3_ds

# Chapter 1

## Protocol Modeling and Refinement Infrastructure

This chapter sets up our theory of refinement and the protocol modeling infrastructure.

### 1.1 Proving infrastructure

```
theory Infra imports Main
begin
```

#### 1.1.1 Prover configuration

```
declare if-split-asm [split]
```

#### 1.1.2 Forward reasoning ("attributes")

The following lemmas are used to produce intro/elim rules from set definitions and relation definitions.

```
lemmas set-def-to-intro = meta-eq-to-obj-eq [THEN eqset-imp-iff, THEN iffD2]
```

```
lemmas set-def-to-dest = meta-eq-to-obj-eq [THEN eqset-imp-iff, THEN iffD1]
```

```
lemmas set-def-to-elim = set-def-to-dest [elim-format]
```

```
lemmas setc-def-to-intro =
  set-def-to-intro [where  $B = \{x. P\ x\}$  for  $P$ , to-pred]
```

```
lemmas setc-def-to-dest =
  set-def-to-dest [where  $B = \{x. P\ x\}$  for  $P$ , to-pred]
```

```
lemmas setc-def-to-elim = setc-def-to-dest [elim-format]
```

```
lemmas rel-def-to-intro = setc-def-to-intro [where  $x = (s, t)$  for  $s\ t$ ]
```

```
lemmas rel-def-to-dest = setc-def-to-dest [where  $x = (s, t)$  for  $s\ t$ ]
```

```
lemmas rel-def-to-elim = rel-def-to-dest [elim-format]
```

### 1.1.3 General results

#### Maps

We usually remove *domIff* from the simpset and clasets due to annoying behavior. Sometimes the lemmas below are more well-behaved than *domIff*. Usually to be used as "dest: dom\_lemmas". However, adding them as permanent dest rules slows down proofs too much, so we refrain from doing this.

**lemma** *map-definedness*:

$f x = \text{Some } y \implies x \in \text{dom } f$   
*<proof>*

**lemma** *map-definedness-contra*:

$\llbracket f x = \text{Some } y; z \notin \text{dom } f \rrbracket \implies x \neq z$   
*<proof>*

**lemmas** *dom-lemmas* = *map-definedness map-definedness-contra*

#### Set

**lemma** *vimage-image-subset*:  $A \subseteq f^{-1}(f \cdot A)$

*<proof>*

#### Relations

**lemma** *Image-compose* [*simp*]:

$(R1 \circ R2) \cdot A = R2 \cdot (R1 \cdot A)$   
*<proof>*

#### Lists

**lemma** *map-id*:  $\text{map } id = id$

*<proof>*

**lemma** *map-comp*:  $\text{map } (g \circ f) = \text{map } g \circ \text{map } f$

*<proof>*

**declare** *map-comp-map* [*simp del*]

**lemma** *take-prefix*:  $\llbracket \text{take } n \ l = xs \rrbracket \implies \exists xs'. l = xs @ xs'$

*<proof>*

#### Finite sets

Cardinality.

**declare** *arg-cong* [**where** *f=card, intro*]

**lemma** *finite-positive-cardI* [*intro!*]:

$\llbracket A \neq \{\}; \text{finite } A \rrbracket \implies 0 < \text{card } A$   
*<proof>*

**lemma** *finite-positive-cardD* [*dest!*]:

$\llbracket 0 < \text{card } A; \text{finite } A \rrbracket \implies A \neq \{\}$

*<proof>*

**lemma** *finite-zero-cardI* [*intro!*]:

$\llbracket A = \{\}; \text{finite } A \rrbracket \implies \text{card } A = 0$

*<proof>*

**lemma** *finite-zero-cardD* [*dest!*]:

$\llbracket \text{card } A = 0; \text{finite } A \rrbracket \implies A = \{\}$

*<proof>*

**end**

## 1.2 Models, Invariants and Refinements

**theory** *Refinement* **imports** *Infra*

**begin**

### 1.2.1 Specifications, reachability, and behaviours.

Transition systems are multi-pointed graphs.

**record** *'s TS* =

*init* :: *'s set*

*trans* :: (*'s* × *'s*) *set*

The inductive set of reachable states.

**inductive-set**

*reach* :: (*'s*, *'a*) *TS-scheme*  $\Rightarrow$  *'s set*

**for** *T* :: (*'s*, *'a*) *TS-scheme*

**where**

*r-init* [*intro*]:  $s \in \text{init } T \implies s \in \text{reach } T$

| *r-trans* [*intro*]:  $\llbracket (s, t) \in \text{trans } T; s \in \text{reach } T \rrbracket \implies t \in \text{reach } T$

### Finite behaviours

Note that behaviours grow at the head of the list, i.e., the initial state is at the end.

**inductive-set**

*beh* :: (*'s*, *'a*) *TS-scheme*  $\Rightarrow$  (*'s list*) *set*

**for** *T* :: (*'s*, *'a*) *TS-scheme*

**where**

*b-empty* [*iff*]:  $\llbracket \in \text{beh } T \rrbracket$

| *b-init* [*intro*]:  $s \in \text{init } T \implies [s] \in \text{beh } T$

| *b-trans* [*intro*]:  $\llbracket s \# b \in \text{beh } T; (s, t) \in \text{trans } T \rrbracket \implies t \# s \# b \in \text{beh } T$

**inductive-cases** *beh-non-empty*:  $s \# b \in \text{beh } T$

Behaviours are prefix closed.

**lemma** *beh-immediate-prefix-closed*:

$s \# b \in \text{beh } T \implies b \in \text{beh } T$

*<proof>*

**lemma** *beh-prefix-closed*:

$$c @ b \in \text{beh } T \implies b \in \text{beh } T$$

$\langle \text{proof} \rangle$

States in behaviours are exactly reachable.

**lemma** *beh-in-reach* [*rule-format*]:

$$b \in \text{beh } T \implies (\forall s \in \text{set } b. s \in \text{reach } T)$$

$\langle \text{proof} \rangle$

**lemma** *reach-in-beh*:

$$\text{assumes } s \in \text{reach } T \text{ shows } \exists b \in \text{beh } T. s \in \text{set } b$$

$\langle \text{proof} \rangle$

**lemma** *reach-equiv-beh-states*:  $\text{reach } T = \bigcup (\text{set}'(\text{beh } T))$

$\langle \text{proof} \rangle$

## Specifications, observability, and implementation

Specifications add an observer function to transition systems.

**record** (*'s, 'o*) *spec* = *'s* *TS* +

$$\text{obs} :: 's \Rightarrow 'o$$

**lemma** *beh-obs-upd* [*simp*]:  $\text{beh } (S(| \text{obs} := x |)) = \text{beh } S$

$\langle \text{proof} \rangle$

**lemma** *reach-obs-upd* [*simp*]:  $\text{reach } (S(| \text{obs} := x |)) = \text{reach } S$

$\langle \text{proof} \rangle$

Observable behaviour and reachability.

**definition**

$$\text{obeh} :: ('s, 'o) \text{spec} \Rightarrow ('o \text{ list}) \text{set} \text{ where}$$

$$\text{obeh } S \equiv (\text{map } (\text{obs } S))'(\text{beh } S)$$

**definition**

$$\text{oreach} :: ('s, 'o) \text{spec} \Rightarrow 'o \text{ set} \text{ where}$$

$$\text{oreach } S \equiv (\text{obs } S)'(\text{reach } S)$$

**lemma** *oreach-equiv-obeh-states*:

$$\text{oreach } S = \bigcup (\text{set}'(\text{obeh } S))$$

$\langle \text{proof} \rangle$

**lemma** *obeh-pi-translation*:

$$(\text{map } \text{pi})'(\text{obeh } S) = \text{obeh } (S(| \text{obs} := \text{pi } o (\text{obs } S) |))$$

$\langle \text{proof} \rangle$

**lemma** *oreach-pi-translation*:

$$\text{pi}'(\text{oreach } S) = \text{oreach } (S(| \text{obs} := \text{pi } o (\text{obs } S) |))$$

$\langle \text{proof} \rangle$

A predicate  $P$  on the states of a specification is *observable* if it cannot distinguish between

states yielding the same observation. Equivalently,  $P$  is observable if it is the inverse image under the observation function of a predicate on observations.

**definition**

$observable :: ['s \Rightarrow 'o, 's\ set] \Rightarrow bool$

**where**

$observable\ ob\ P \equiv \forall s\ s'.\ ob\ s = ob\ s' \longrightarrow s' \in P \longrightarrow s \in P$

**definition**

$observable2 :: ['s \Rightarrow 'o, 's\ set] \Rightarrow bool$

**where**

$observable2\ ob\ P \equiv \exists Q.\ P = ob-'Q$

**definition**

$observable3 :: ['s \Rightarrow 'o, 's\ set] \Rightarrow bool$

**where**

$observable3\ ob\ P \equiv ob-'ob'P \subseteq P \quad \text{— other direction holds trivially}$

**lemma** *observableE* [elim]:

$\llbracket observable\ ob\ P; ob\ s = ob\ s'; s' \in P \rrbracket \Longrightarrow s \in P$

$\langle proof \rangle$

**lemma** *observable2-equiv-observable*:  $observable2\ ob\ P = observable\ ob\ P$

$\langle proof \rangle$

**lemma** *observable3-equiv-observable2*:  $observable3\ ob\ P = observable2\ ob\ P$

$\langle proof \rangle$

**lemma** *observable-id* [simp]:  $observable\ id\ P$

$\langle proof \rangle$

The set extension of a function  $ob$  is the left adjoint of a Galois connection on the powerset lattices over domain and range of  $ob$  where the right adjoint is the inverse image function.

**lemma** *image-vimage-adjoints*:  $(ob'P \subseteq Q) = (P \subseteq ob-'Q)$

$\langle proof \rangle$

**declare** *image-vimage-subset* [simp, intro]

**declare** *vimage-image-subset* [simp, intro]

Similar but "reversed" (wrt to adjointness) relationships only hold under additional conditions.

**lemma** *image-r-vimage-l*:  $\llbracket Q \subseteq ob'P; observable\ ob\ P \rrbracket \Longrightarrow ob-'Q \subseteq P$

$\langle proof \rangle$

**lemma** *vimage-l-image-r*:  $\llbracket ob-'Q \subseteq P; Q \subseteq range\ ob \rrbracket \Longrightarrow Q \subseteq ob'P$

$\langle proof \rangle$

Internal and external invariants

**lemma** *external-from-internal-invariant*:

$\llbracket reach\ S \subseteq P; (obs\ S)'P \subseteq Q \rrbracket$

$\Longrightarrow oreach\ S \subseteq Q$

$\langle proof \rangle$

**lemma** *external-from-internal-invariant-vimage:*

$$\llbracket \text{reach } S \subseteq P; P \subseteq (\text{obs } S) \text{' } Q \rrbracket \\ \implies \text{oreach } S \subseteq Q$$

$\langle \text{proof} \rangle$

**lemma** *external-to-internal-invariant-vimage:*

$$\llbracket \text{oreach } S \subseteq Q; (\text{obs } S) \text{' } Q \subseteq P \rrbracket \\ \implies \text{reach } S \subseteq P$$

$\langle \text{proof} \rangle$

**lemma** *external-to-internal-invariant:*

$$\llbracket \text{oreach } S \subseteq Q; Q \subseteq (\text{obs } S) \text{' } P; \text{observable } (\text{obs } S) P \rrbracket \\ \implies \text{reach } S \subseteq P$$

$\langle \text{proof} \rangle$

**lemma** *external-equiv-internal-invariant-vimage:*

$$\llbracket P = (\text{obs } S) \text{' } Q \rrbracket \\ \implies (\text{oreach } S \subseteq Q) = (\text{reach } S \subseteq P)$$

$\langle \text{proof} \rangle$

**lemma** *external-equiv-internal-invariant:*

$$\llbracket (\text{obs } S) \text{' } P = Q; \text{observable } (\text{obs } S) P \rrbracket \\ \implies (\text{oreach } S \subseteq Q) = (\text{reach } S \subseteq P)$$

$\langle \text{proof} \rangle$

Our notion of implementation is inclusion of observable behaviours.

**definition**

$$\text{implements} :: [ 'p \Rightarrow 'o, ('s, 'o) \text{ spec}, ('t, 'p) \text{ spec} ] \Rightarrow \text{bool} \textbf{ where} \\ \text{implements } \pi \text{ } S \text{ } S \equiv (\text{map } \pi) \text{' } (\text{obeh } S \text{ } c) \subseteq \text{obeh } S \text{ } a$$

Reflexivity and transitivity

**lemma** *implements-refl:* *implements id S S*

$\langle \text{proof} \rangle$

**lemma** *implements-trans:*

$$\llbracket \text{implements } \pi 1 \text{ } S 1 \text{ } S 2; \text{implements } \pi 2 \text{ } S 2 \text{ } S 3 \rrbracket \\ \implies \text{implements } (\pi 1 \text{ } o \text{ } \pi 2) \text{ } S 1 \text{ } S 3$$

$\langle \text{proof} \rangle$

Preservation of external invariants

**lemma** *implements-oreach:*

$$\text{implements } \pi \text{ } S \text{ } a \text{ } S \text{ } c \implies \pi \text{' } (\text{oreach } S \text{ } c) \subseteq \text{oreach } S \text{ } a$$

$\langle \text{proof} \rangle$

**lemma** *external-invariant-preservation:*

$$\llbracket \text{oreach } S \text{ } a \subseteq Q; \text{implements } \pi \text{ } S \text{ } a \text{ } S \text{ } c \rrbracket \\ \implies \pi \text{' } (\text{oreach } S \text{ } c) \subseteq Q$$

$\langle \text{proof} \rangle$

**lemma** *external-invariant-translation:*

$$\llbracket \text{oreach } Sa \subseteq Q; \text{pi-}'Q \subseteq P; \text{implements } \text{pi } Sa \text{ } Sc \rrbracket$$

$$\implies \text{oreach } Sc \subseteq P$$
 <proof>

Preservation of internal invariants

**lemma** *internal-invariant-translation*:

$$\llbracket \text{reach } Sa \subseteq Pa; Pa \subseteq \text{obs } Sa \text{ -}' Qa; \text{pi -}' Qa \subseteq Q; \text{obs } S \text{ -}' Q \subseteq P; \text{implements } \text{pi } Sa \text{ } S \rrbracket$$

$$\implies \text{reach } S \subseteq P$$
 <proof>

## 1.2.2 Invariants

First we define Hoare triples over transition relations and then we derive proof rules to establish invariants.

### Hoare triples

**definition**

$$PO\text{-hoare} :: ['s \text{ set}, ('s \times 's) \text{ set}, 's \text{ set}] \Rightarrow \text{bool}$$

$$(\langle \exists \{-\} - \{> \{-\} \rangle, [0, 0, 0] \ 90)$$

**where**

$$\{pre\} R \{> post\} \equiv R \text{'pre} \subseteq post$$

**lemmas** *PO-hoare-defs = PO-hoare-def Image-def*

**lemma**  $\{P\} R \{> Q\} = (\forall s \ t. s \in P \longrightarrow (s, t) \in R \longrightarrow t \in Q)$   
 <proof>

Some essential facts about Hoare triples.

**lemma** *hoare-conseq-left* [*intro*]:

$$\llbracket \{P'\} R \{> Q\}; P \subseteq P' \rrbracket$$

$$\implies \{P\} R \{> Q\}$$
 <proof>

**lemma** *hoare-conseq-right*:

$$\llbracket \{P\} R \{> Q'\}; Q' \subseteq Q \rrbracket$$

$$\implies \{P\} R \{> Q\}$$
 <proof>

**lemma** *hoare-false-left* [*simp*]:

$$\{\{\}\} R \{> Q\}$$
 <proof>

**lemma** *hoare-true-right* [*simp*]:

$$\{P\} R \{> UNIV\}$$
 <proof>

**lemma** *hoare-conj-right* [*intro!*]:

$$\llbracket \{P\} R \{> Q1\}; \{P\} R \{> Q2\} \rrbracket$$

$$\implies \{P\} R \{> Q1 \cap Q2\}$$
 <proof>

Special transition relations.

**lemma** *hoare-stop* [*simp, intro!*]:

$$\{P\} \{\}\ \{>\ Q\}$$

*<proof>*

**lemma** *hoare-skip* [*simp, intro!*]:

$$P \subseteq Q \implies \{P\} \text{Id} \{>\ Q\}$$

*<proof>*

**lemma** *hoare-trans-Un* [*iff*]:

$$\{P\} R1 \cup R2 \{>\ Q\} = (\{P\} R1 \{>\ Q\} \wedge \{P\} R2 \{>\ Q\})$$

*<proof>*

**lemma** *hoare-trans-UN* [*iff*]:

$$\{P\} \cup x. R\ x \{>\ Q\} = (\forall x. \{P\} R\ x \{>\ Q\})$$

*<proof>*

### Characterization of reachability

**lemma** *reach-init*:  $\text{reach } T \subseteq I \implies \text{init } T \subseteq I$

*<proof>*

**lemma** *reach-trans*:  $\text{reach } T \subseteq I \implies \{\text{reach } T\} \text{trans } T \{>\ I\}$

*<proof>*

Useful consequences.

**corollary** *init-reach* [*iff*]:  $\text{init } T \subseteq \text{reach } T$

*<proof>*

**corollary** *trans-reach* [*iff*]:  $\{\text{reach } T\} \text{trans } T \{>\ \text{reach } T\}$

*<proof>*

### Invariant proof rules

Basic proof rule for invariants.

**lemma** *inv-rule-basic*:

$$\llbracket \text{init } T \subseteq P; \{P\} (\text{trans } T) \{>\ P\} \rrbracket$$

$$\implies \text{reach } T \subseteq P$$

*<proof>*

General invariant proof rule. This rule is complete (set  $I = \text{reach } T$ ).

**lemma** *inv-rule*:

$$\llbracket \text{init } T \subseteq I; I \subseteq P; \{I\} (\text{trans } T) \{>\ I\} \rrbracket$$

$$\implies \text{reach } T \subseteq P$$

*<proof>*

The following rule is equivalent to the previous one.

**lemma** *INV-rule*:

$$\llbracket \text{init } T \subseteq I; \{I \cap \text{reach } T\} (\text{trans } T) \{>\ I\} \rrbracket$$

$$\implies \text{reach } T \subseteq I$$

*<proof>*

Proof of equivalence.

**lemma** *inv-rule-from-INV-rule*:

$$\llbracket \text{init } T \subseteq I; I \subseteq P; \{I\} (\text{trans } T) \{> I\} \rrbracket \\ \implies \text{reach } T \subseteq P$$

$\langle \text{proof} \rangle$

**lemma** *INV-rule-from-inv-rule*:

$$\llbracket \text{init } T \subseteq I; \{I \cap \text{reach } T\} (\text{trans } T) \{> I\} \rrbracket \\ \implies \text{reach } T \subseteq I$$

$\langle \text{proof} \rangle$

Incremental proof rule for invariants using auxiliary invariant(s). This rule might have become obsolete by addition of *INV\_rule*.

**lemma** *inv-rule-incr*:

$$\llbracket \text{init } T \subseteq I; \{I \cap J\} (\text{trans } T) \{> I\}; \text{reach } T \subseteq J \rrbracket \\ \implies \text{reach } T \subseteq I$$

$\langle \text{proof} \rangle$

### 1.2.3 Refinement

Our notion of refinement is simulation. We first define a general notion of relational Hoare tuple, which we then use to define the refinement proof obligation. Finally, we show that observation-consistent refinement of specifications implies the implementation relation between them.

#### Relational Hoare tuples

Relational Hoare tuples formalize the following generalized simulation diagram:

$$\begin{array}{ccc} \circ & \text{-- } Ra & \text{-->} \circ \\ | & & | \\ \text{pre} & & \text{post} \\ | & & | \\ \vee & & \vee \\ \circ & \text{-- } Rc & \text{-->} \circ \end{array}$$

Here, *Ra* and *Rc* are the abstract and concrete transition relations, and *pre* and *post* are the pre- and post-relations. (In the definition below, the operator (*O*) stands for relational composition, which is defined as follows: (*O*)  $\equiv \lambda r s. \{(xa, x). ((\lambda x xa. (x, xa) \in r) O O (\lambda x xa. (x, xa) \in s)) xa x\}.$ )

**definition**

$$PO\text{-rhoare} :: \\ [(\text{'s} \times \text{'t}) \text{ set}, (\text{'s} \times \text{'s}) \text{ set}, (\text{'t} \times \text{'t}) \text{ set}, (\text{'s} \times \text{'t}) \text{ set}] \Rightarrow \text{bool} \\ (\langle \{ \{ - \} \} -, - \{ > - \} \rangle [0, 0, 0] 90)$$

**where**

$$\{pre\} Ra, Rc \{> post\} \equiv pre O Rc \subseteq Ra O post$$

**lemmas** *PO-rhoare-defs = PO-rhoare-def relcomp-unfold*

Facts about relational Hoare tuples.

**lemma** *relhoare-conseq-left* [*intro*]:  

$$\llbracket \{pre\} Ra, Rc \{> post\}; pre \subseteq pre' \rrbracket$$

$$\implies \{pre\} Ra, Rc \{> post\}$$
 $\langle proof \rangle$

**lemma** *relhoare-conseq-right*: — do NOT declare [*intro*]  

$$\llbracket \{pre\} Ra, Rc \{> post'\}; post' \subseteq post \rrbracket$$

$$\implies \{pre\} Ra, Rc \{> post\}$$
 $\langle proof \rangle$

**lemma** *relhoare-false-left* [*simp*]: — do NOT declare [*intro*]  

$$\{ \{ \} \} Ra, Rc \{> post\}$$
 $\langle proof \rangle$

**lemma** *relhoare-true-right* [*simp*]: — not true in general  

$$\{pre\} Ra, Rc \{> UNIV\} = (Domain (pre \ O \ Rc) \subseteq Domain \ Ra)$$
 $\langle proof \rangle$

**lemma** *Domain-rel-comp* [*intro*]:  

$$Domain \ pre \subseteq R \implies Domain (pre \ O \ Rc) \subseteq R$$
 $\langle proof \rangle$

**lemma** *rel-hoare-skip* [*iff*]:  $\{R\} Id, Id \{> R\}$   
 $\langle proof \rangle$

Reflexivity and transitivity.

**lemma** *relhoare-refl* [*simp*]:  $\{Id\} R, R \{> Id\}$   
 $\langle proof \rangle$

**lemma** *rhoare-trans*:  

$$\llbracket \{R1\} T1, T2 \{> R1\}; \{R2\} T2, T3 \{> R2\} \rrbracket$$

$$\implies \{R1 \ O \ R2\} T1, T3 \{> R1 \ O \ R2\}$$
 $\langle proof \rangle$

Conjunction in the post-relation cannot be split in general. However, here are two useful special cases. In the first case the abstract transtition relation is deterministic and in the second case one conjunct is a cartesian product of two state predicates.

**lemma** *relhoare-conj-right-det*:  

$$\llbracket \{pre\} Ra, Rc \{> post1\}; \{pre\} Ra, Rc \{> post2\};$$

$$single-valued \ Ra \rrbracket$$
 — only for deterministic *Ra*!  

$$\implies \{pre\} Ra, Rc \{> post1 \cap post2\}$$
 $\langle proof \rangle$

**lemma** *relhoare-conj-right-cartesian* [*intro*]:  

$$\llbracket \{Domain \ pre\} Ra \{> I\}; \{Range \ pre\} Rc \{> J\};$$

$$\{pre\} Ra, Rc \{> post\} \rrbracket$$

$$\implies \{pre\} Ra, Rc \{> post \cap I \times J\}$$
 $\langle proof \rangle$

Separate rule for cartesian products.

**corollary** *relhoare-cartesian*:

$$\llbracket \{ \text{Domain } pre \} Ra \{> I\}; \{ \text{Range } pre \} Rc \{> J\};$$

$$\{pre\} Ra, Rc \{> post\} \rrbracket \quad \text{— any post, including } UNIV!$$

$$\implies \{pre\} Ra, Rc \{> I \times J\}$$

$$\langle \text{proof} \rangle$$

Unions of transition relations.

**lemma** *relhoare-concrete-Un* [*simp*]:  

$$\{pre\} Ra, Rc1 \cup Rc2 \{> post\}$$

$$= (\{pre\} Ra, Rc1 \{> post\} \wedge \{pre\} Ra, Rc2 \{> post\})$$

$$\langle \text{proof} \rangle$$

**lemma** *relhoare-concrete-UN* [*simp*]:  

$$\{pre\} Ra, \bigcup x. Rc x \{> post\} = (\forall x. \{pre\} Ra, Rc x \{> post\})$$

$$\langle \text{proof} \rangle$$

**lemma** *relhoare-abstract-Un-left* [*intro*]:  

$$\llbracket \{pre\} Ra1, Rc \{> post\} \rrbracket$$

$$\implies \{pre\} Ra1 \cup Ra2, Rc \{> post\}$$

$$\langle \text{proof} \rangle$$

**lemma** *relhoare-abstract-Un-right* [*intro*]:  

$$\llbracket \{pre\} Ra2, Rc \{> post\} \rrbracket$$

$$\implies \{pre\} Ra1 \cup Ra2, Rc \{> post\}$$

$$\langle \text{proof} \rangle$$

**lemma** *relhoare-abstract-UN* [*intro!*]: — might be too aggressive?  

$$\llbracket \{pre\} Ra x, Rc \{> post\} \rrbracket$$

$$\implies \{pre\} \bigcup x. Ra x, Rc \{> post\}$$

$$\langle \text{proof} \rangle$$

## Refinement proof obligations

A transition system refines another one if the initial states and the transitions are refined. Initial state refinement means that for each concrete initial state there is a related abstract one. Transition refinement means that the simulation relation is preserved (as expressed by a relational Hoare tuple).

### definition

$$PO\text{-refines} ::$$

$$[(s \times t) \text{ set}, (s, 'a) \text{ TS-scheme}, (t, 'b) \text{ TS-scheme}] \Rightarrow \text{bool}$$

### where

$$PO\text{-refines } R \text{ } Ta \text{ } Tc \equiv ($$

$$\quad \text{init } Tc \subseteq R''(\text{init } Ta)$$

$$\quad \wedge \{R\} (\text{trans } Ta), (\text{trans } Tc) \{> R\}$$

$$)$$

### lemma

*PO-refinesI*:

$$\llbracket \text{init } Tc \subseteq R''(\text{init } Ta); \{R\} (\text{trans } Ta), (\text{trans } Tc) \{> R\} \rrbracket \implies PO\text{-refines } R \text{ } Ta \text{ } Tc$$

$$\langle \text{proof} \rangle$$

### lemma

*PO-refinesE* [*elim*]:

$$\llbracket PO\text{-refines } R \text{ } Ta \text{ } Tc; \llbracket \text{init } Tc \subseteq R''(\text{init } Ta); \{R\} (\text{trans } Ta), (\text{trans } Tc) \{> R\} \rrbracket \implies P \rrbracket$$

$$\implies P$$

*<proof>*

Basic refinement rule. This is just an introduction rule for the definition.

**lemma** *refine-basic:*

$$\begin{aligned} & \llbracket \text{init } Tc \subseteq R \text{''}(\text{init } Ta); \{R\} (\text{trans } Ta), (\text{trans } Tc) \{> R\} \rrbracket \\ & \implies \text{PO-refines } R \text{ } Ta \text{ } Tc \end{aligned}$$

*<proof>*

The following proof rule uses individual invariants  $I$  and  $J$  of the concrete and abstract systems to strengthen the simulation relation  $R$ .

The hypotheses state that these state predicates are indeed invariants. Note that the precondition of the invariant preservation hypotheses for  $I$  and  $J$  are strengthened by adding the predicates  $\text{Domain } (R \cap \text{UNIV} \times J)$  and  $\text{Range } (R \cap I \times \text{UNIV})$ , respectively. In particular, the latter predicate may be essential, if a concrete invariant depends on the simulation relation and an abstract invariant, i.e. to "transport" abstract invariants to the concrete system.

**lemma** *refine-init-using-invariants:*

$$\begin{aligned} & \llbracket \text{init } Tc \subseteq R \text{''}(\text{init } Ta); \text{init } Ta \subseteq I; \text{init } Tc \subseteq J \rrbracket \\ & \implies \text{init } Tc \subseteq (R \cap I \times J) \text{''}(\text{init } Ta) \end{aligned}$$

*<proof>*

**lemma** *refine-trans-using-invariants:*

$$\begin{aligned} & \llbracket \{R \cap I \times J\} (\text{trans } Ta), (\text{trans } Tc) \{> R\}; \\ & \quad \{I \cap \text{Domain } (R \cap \text{UNIV} \times J)\} (\text{trans } Ta) \{> I\}; \\ & \quad \{J \cap \text{Range } (R \cap I \times \text{UNIV})\} (\text{trans } Tc) \{> J\} \rrbracket \\ & \implies \{R \cap I \times J\} (\text{trans } Ta), (\text{trans } Tc) \{> R \cap I \times J\} \end{aligned}$$

*<proof>*

This is our main rule for refinements.

**lemma** *refine-using-invariants:*

$$\begin{aligned} & \llbracket \{R \cap I \times J\} (\text{trans } Ta), (\text{trans } Tc) \{> R\}; \\ & \quad \{I \cap \text{Domain } (R \cap \text{UNIV} \times J)\} (\text{trans } Ta) \{> I\}; \\ & \quad \{J \cap \text{Range } (R \cap I \times \text{UNIV})\} (\text{trans } Tc) \{> J\}; \\ & \quad \text{init } Tc \subseteq R \text{''}(\text{init } Ta); \\ & \quad \text{init } Ta \subseteq I; \text{init } Tc \subseteq J \rrbracket \\ & \implies \text{PO-refines } (R \cap I \times J) \text{ } Ta \text{ } Tc \end{aligned}$$

*<proof>*

## Deriving invariants from refinements

Some invariants can only be proved after the simulation has been established, because they depend on the simulation relation and some abstract invariants. Here is a rule to derive invariant theorems from the refinement.

**lemma** *PO-refines-implies-Range-init:*

$$\text{PO-refines } R \text{ } Ta \text{ } Tc \implies \text{init } Tc \subseteq \text{Range } R$$

*<proof>*

**lemma** *PO-refines-implies-Range-trans:*

$$\text{PO-refines } R \text{ } Ta \text{ } Tc \implies \{\text{Range } R\} \text{ trans } Tc \{> \text{Range } R\}$$

*<proof>*

**lemma** *PO-refines-implies-Range-invariant*:  
 $PO\text{-refines } R \text{ } Ta \text{ } Tc \implies reach \ Tc \subseteq Range \ R$   
 ⟨proof⟩

The following rules are more useful in proofs.

**corollary** *INV-init-from-refinement*:  
 $\llbracket PO\text{-refines } R \text{ } Ta \text{ } Tc; Range \ R \subseteq I \rrbracket$   
 $\implies init \ Tc \subseteq I$   
 ⟨proof⟩

**corollary** *INV-trans-from-refinement*:  
 $\llbracket PO\text{-refines } R \text{ } Ta \text{ } Tc; K \subseteq Range \ R; Range \ R \subseteq I \rrbracket$   
 $\implies \{K\} \text{ trans } Tc \ \{> \ I\}$   
 ⟨proof⟩

**corollary** *INV-from-refinement*:  
 $\llbracket PO\text{-refines } R \text{ } Ta \text{ } Tc; Range \ R \subseteq I \rrbracket$   
 $\implies reach \ Tc \subseteq I$   
 ⟨proof⟩

## Refinement of specifications

Lift relation membership to finite sequences

**inductive-set**  
 $seq\text{-lift} :: ('s \times 't) \text{ set} \Rightarrow ('s \text{ list} \times 't \text{ list}) \text{ set}$   
**for**  $R :: ('s \times 't) \text{ set}$   
**where**  
 $sl\text{-nil} \ [iff]: ([], []) \in seq\text{-lift } R$   
 $| \ sl\text{-cons} \ [intro]:$   
 $\llbracket (xs, ys) \in seq\text{-lift } R; (x, y) \in R \rrbracket \implies (x\#\!xs, y\#\!ys) \in seq\text{-lift } R$

**inductive-cases**  $sl\text{-cons-right-invert}: (ba', t \# bc) \in seq\text{-lift } R$

For each concrete behaviour there is a related abstract one.

**lemma** *behaviour-refinement*:  
 $\llbracket PO\text{-refines } R \text{ } Ta \text{ } Tc; bc \in beh \ Tc \rrbracket$   
 $\implies \exists ba \in beh \ Ta. (ba, bc) \in seq\text{-lift } R$   
 ⟨proof⟩

Observation consistency of a relation is defined using a mediator function  $pi$  to abstract the concrete observation. This allows us to also refine the observables as we move down a refinement branch.

**definition**  
 $obs\text{-consistent} ::$   
 $\llbracket ('s \times 't) \text{ set}, 'p \Rightarrow 'o, ('s, 'o) \text{ spec}, ('t, 'p) \text{ spec} \rrbracket \Rightarrow bool$   
**where**  
 $obs\text{-consistent } R \ pi \ Sa \ Sc \equiv (\forall s \ t. (s, t) \in R \longrightarrow pi \ (obs \ Sc \ t) = obs \ Sa \ s)$

**lemma** *obs-consistent-refl* [iff]:  $obs\text{-consistent } Id \ id \ S \ S$   
 ⟨proof⟩

**lemma** *obs-consistent-trans* [intro]:  
 $\llbracket \text{obs-consistent } R1 \text{ } \pi 1 \text{ } S1 \text{ } S2; \text{obs-consistent } R2 \text{ } \pi 2 \text{ } S2 \text{ } S3 \rrbracket$   
 $\implies \text{obs-consistent } (R1 \text{ } O \text{ } R2) (\pi 1 \text{ } o \text{ } \pi 2) S1 \text{ } S3$   
 $\langle \text{proof} \rangle$

**lemma** *obs-consistent-empty*:  $\text{obs-consistent } \{ \} \pi \text{ } Sa \text{ } Sc$   
 $\langle \text{proof} \rangle$

**lemma** *obs-consistent-conj1* [intro]:  
 $\text{obs-consistent } R \pi \text{ } Sa \text{ } Sc \implies \text{obs-consistent } (R \cap R') \pi \text{ } Sa \text{ } Sc$   
 $\langle \text{proof} \rangle$

**lemma** *obs-consistent-conj2* [intro]:  
 $\text{obs-consistent } R \pi \text{ } Sa \text{ } Sc \implies \text{obs-consistent } (R' \cap R) \pi \text{ } Sa \text{ } Sc$   
 $\langle \text{proof} \rangle$

**lemma** *obs-consistent-behaviours*:  
 $\llbracket \text{obs-consistent } R \pi \text{ } Sa \text{ } Sc; bc \in \text{beh } Sc; ba \in \text{beh } Sa; (ba, bc) \in \text{seq-lift } R \rrbracket$   
 $\implies \text{map } \pi (\text{map } (\text{obs } Sc) bc) = \text{map } (\text{obs } Sa) ba$   
 $\langle \text{proof} \rangle$

Definition of refinement proof obligations.

**definition**  
 $\text{refines} ::$   
 $\llbracket ('s \times 't) \text{ set}, 'p \Rightarrow 'o, ('s, 'o) \text{ spec}, ('t, 'p) \text{ spec} \rrbracket \Rightarrow \text{bool}$

**where**  
 $\text{refines } R \pi \text{ } Sa \text{ } Sc \equiv \text{obs-consistent } R \pi \text{ } Sa \text{ } Sc \wedge \text{PO-refines } R \text{ } Sa \text{ } Sc$

**lemmas** *refines-defs* =  
 $\text{refines-def } \text{PO-refines-def}$

**lemma** *refinesI*:  
 $\llbracket \text{PO-refines } R \text{ } Sa \text{ } Sc; \text{obs-consistent } R \pi \text{ } Sa \text{ } Sc \rrbracket$   
 $\implies \text{refines } R \pi \text{ } Sa \text{ } Sc$   
 $\langle \text{proof} \rangle$

**lemma** *refinesE* [elim]:  
 $\llbracket \text{refines } R \pi \text{ } Sa \text{ } Sc; \llbracket \text{PO-refines } R \text{ } Sa \text{ } Sc; \text{obs-consistent } R \pi \text{ } Sa \text{ } Sc \rrbracket \implies P \rrbracket$   
 $\implies P$   
 $\langle \text{proof} \rangle$

Reflexivity and transitivity of refinement.

**lemma** *refinement-reflexive*:  $\text{refines } Id \text{ } id \text{ } S \text{ } S$   
 $\langle \text{proof} \rangle$

**lemma** *refinement-transitive*:  
 $\llbracket \text{refines } R1 \text{ } \pi 1 \text{ } S1 \text{ } S2; \text{refines } R2 \text{ } \pi 2 \text{ } S2 \text{ } S3 \rrbracket$   
 $\implies \text{refines } (R1 \text{ } O \text{ } R2) (\pi 1 \text{ } o \text{ } \pi 2) S1 \text{ } S3$   
 $\langle \text{proof} \rangle$

Soundness of refinement for proving implementation

**lemma** *observable-behaviour-refinement*:

$\llbracket \text{refines } R \text{ pi } Sa \text{ Sc}; bc \in \text{obeh } Sc \rrbracket \implies \text{map pi } bc \in \text{obeh } Sa$   
 <proof>

**theorem** *refinement-soundness:*

$\text{refines } R \text{ pi } Sa \text{ Sc} \implies \text{implements pi } Sa \text{ Sc}$

<proof>

Extended versions of refinement proof rules including observations

**lemmas** *Refinement-basic = refine-basic [THEN refinesI]*

**lemmas** *Refinement-using-invariants = refine-using-invariants [THEN refinesI]*

**lemma** *refines-reachable-strengthening:*

$\text{refines } R \text{ pi } Sa \text{ Sc} \implies \text{refines } (R \cap \text{reach } Sa \times \text{reach } Sc) \text{ pi } Sa \text{ Sc}$

<proof>

Inheritance of internal invariants through refinements

**lemma** *INV-init-from-Refinement:*

$\llbracket \text{refines } R \text{ pi } Sa \text{ Sc}; \text{Range } R \subseteq I \rrbracket \implies \text{init } Sc \subseteq I$

<proof>

**lemma** *INV-trans-from-Refinement:*

$\llbracket \text{refines } R \text{ pi } Sa \text{ Sc}; K \subseteq \text{Range } R; \text{Range } R \subseteq I \rrbracket \implies \{K\} \text{TS.trans } Sc \{> I\}$

<proof>

**lemma** *INV-from-Refinement-basic:*

$\llbracket \text{refines } R \text{ pi } Sa \text{ Sc}; \text{Range } R \subseteq I \rrbracket \implies \text{reach } Sc \subseteq I$

<proof>

**lemma** *INV-from-Refinement-using-invariants:*

**assumes**  $\text{refines } R \text{ pi } Sa \text{ Sc}$   $\text{Range } (R \cap I \times J) \subseteq K$  — EQUIV:  $R \text{''} I \cap J$   
 $\text{reach } Sa \subseteq I$   $\text{reach } Sc \subseteq J$

**shows**  $\text{reach } Sc \subseteq K$

<proof>

end

## 1.3 Atomic messages

**theory** *Agents imports Main*

**begin**

The definitions below are moved here from the message theory, since the higher levels of protocol abstraction do not know about cryptographic messages.

### 1.3.1 Agents

**datatype** — We allow any number of agents plus an honest server.

$\text{agent} = \text{Server} \mid \text{Agent } \text{nat}$

**consts**  
*bad* :: *agent set* — compromised agents

**specification** (*bad*)  
*Server-not-bad* [iff]: *Server*  $\notin$  *bad*  
*<proof>*

**abbreviation**  
*good* :: *agent set*

**where**  
*good*  $\equiv$   $\neg$ *bad*

**abbreviation**  
*Sv* :: *agent*

**where**  
*Sv*  $\equiv$  *Server*

### 1.3.2 Nonces

We have an unspecified type of freshness identifiers. For executability, we may need to assume that this type is infinite.

**typedecl** *fid-t*

**datatype** *fresh-t* =  
*mk-fresh fid-t nat* (infixr  $\langle \$ \rangle$  65)

**fun** *fid* :: *fresh-t*  $\Rightarrow$  *fid-t* **where**  
*fid* (*f* \$ *n*) = *f*

**fun** *num* :: *fresh-t*  $\Rightarrow$  *nat* **where**  
*num* (*f* \$ *n*) = *n*

Nonces

**type-synonym**  
*nonce* = *fresh-t*

**end**

## 1.4 Symmetric and Assymmetric Keys

**theory** *Keys* imports *Agents* **begin**

Divide keys into session and long-term keys. Define different kinds of long-term keys in second step.

**datatype** *ltkey* = — long-term keys  
*sharK agent* — key shared with server  
| *publK agent* — agent's public key  
| *privK agent* — agent's private key

**datatype** *key* =

*sesK fresh-t* — session key  
| *ltK ltkey* — long-term key

**abbreviation**

*shrK* :: *agent*  $\Rightarrow$  *key* **where**  
*shrK* *A*  $\equiv$  *ltK* (*sharK* *A*)

**abbreviation**

*pubK* :: *agent*  $\Rightarrow$  *key* **where**  
*pubK* *A*  $\equiv$  *ltK* (*publK* *A*)

**abbreviation**

*priK* :: *agent*  $\Rightarrow$  *key* **where**  
*priK* *A*  $\equiv$  *ltK* (*privK* *A*)

The inverse of a symmetric key is itself; that of a public key is the private key and vice versa

**fun** *invKey* :: *key*  $\Rightarrow$  *key* **where**

*invKey* (*ltK* (*publK* *A*)) = *priK* *A*  
| *invKey* (*ltK* (*privK* *A*)) = *pubK* *A*  
| *invKey* *K* = *K*

**definition**

*symKeys* :: *key set* **where**  
*symKeys*  $\equiv$  {*K*. *invKey* *K* = *K*}

**lemma** *invKey-K*:  $K \in \text{symKeys} \implies \text{invKey } K = K$

*<proof>*

Most lemmas we need come for free with the inductive type definition: injectiveness and distinctness.

**lemma** *invKey-invKey-id* [*simp*]: *invKey* (*invKey* *K*) = *K*

*<proof>*

**lemma** *invKey-eq* [*simp*]: (*invKey* *K* = *invKey* *K'*) = (*K*=*K'*)

*<proof>*

We get most lemmas below for free from the inductive definition of type *key*. Many of these are just proved as a reality check.

### 1.4.1 Asymmetric Keys

No private key equals any public key (essential to ensure that private keys are private!). A similar statement an axiom in Paulson's theory!

**lemma** *privateKey-neq-publicKey*: *priK* *A*  $\neq$  *pubK* *A'*

*<proof>*

**lemma** *publicKey-neq-privateKey*: *pubK* *A*  $\neq$  *priK* *A'*

*<proof>*

### 1.4.2 Basic properties of *pubK* and *priK*

**lemma** *publicKey-inject* [*iff*]: (*pubK* *A* = *pubK* *A'*) = (*A* = *A'*)

$\langle proof \rangle$

**lemma** *not-symKeys-pubK* [iff]:  $pubK\ A \notin symKeys$

$\langle proof \rangle$

**lemma** *not-symKeys-priK* [iff]:  $priK\ A \notin symKeys$

$\langle proof \rangle$

**lemma** *symKey-neq-priK*:  $K \in symKeys \implies K \neq priK\ A$

$\langle proof \rangle$

**lemma** *symKeys-neq-imp-neq*:  $(K \in symKeys) \neq (K' \in symKeys) \implies K \neq K'$

$\langle proof \rangle$

**lemma** *symKeys-invKey-iff* [iff]:  $(invKey\ K \in symKeys) = (K \in symKeys)$

$\langle proof \rangle$

### 1.4.3 "Image" equations that hold for injective functions

**lemma** *invKey-image-eq* [simp]:  $(invKey\ x \in invKey\ A) = (x \in A)$

$\langle proof \rangle$

**lemma** *invKey-pubK-image-priK-image* [simp]:  $invKey\ ' pubK\ ' AS = priK\ ' AS$

$\langle proof \rangle$

**lemma** *publicKey-notin-image-privateKey*:  $pubK\ A \notin priK\ ' AS$

$\langle proof \rangle$

**lemma** *privateKey-notin-image-publicKey*:  $priK\ x \notin pubK\ ' AA$

$\langle proof \rangle$

**lemma** *publicKey-image-eq* [simp]:  $(pubK\ x \in pubK\ ' AA) = (x \in AA)$

$\langle proof \rangle$

**lemma** *privateKey-image-eq* [simp]:  $(priK\ A \in priK\ ' AS) = (A \in AS)$

$\langle proof \rangle$

### 1.4.4 Symmetric Keys

The following was stated as an axiom in Paulson's theory.

**lemma** *sym-sesK*:  $sesK\ f \in symKeys$  — All session keys are symmetric

$\langle proof \rangle$

**lemma** *sym-shrK*:  $shrK\ X \in symKeys$  — All shared keys are symmetric

$\langle proof \rangle$

Symmetric keys and inversion

**lemma** *symK-eq-invKey*:  $\llbracket SK = invKey\ K; SK \in symKeys \rrbracket \implies K = SK$

$\langle proof \rangle$

Image-related lemmas.

**lemma** *publicKey-notin-image-shrK*:  $\text{pubK } x \notin \text{shrK } 'AA$   
*<proof>*

**lemma** *privateKey-notin-image-shrK*:  $\text{priK } x \notin \text{shrK } 'AA$   
*<proof>*

**lemma** *shrK-notin-image-publicKey*:  $\text{shrK } x \notin \text{pubK } 'AA$   
*<proof>*

**lemma** *shrK-notin-image-privateKey*:  $\text{shrK } x \notin \text{priK } 'AA$   
*<proof>*

**lemma** *sesK-notin-image-shrK [simp]*:  $\text{sesK } K \notin \text{shrK } 'AA$   
*<proof>*

**lemma** *shrK-notin-image-sesK [simp]*:  $\text{shrK } K \notin \text{sesK } 'AA$   
*<proof>*

**lemma** *sesK-image-eq [simp]*:  $(\text{sesK } x \in \text{sesK } 'AA) = (x \in AA)$   
*<proof>*

**lemma** *shrK-image-eq [simp]*:  $(\text{shrK } x \in \text{shrK } 'AA) = (x \in AA)$   
*<proof>*

**end**

## 1.5 Atomic messages

**theory** *Atoms* **imports** *Keys*  
**begin**

### 1.5.1 Atoms datatype

**datatype** *atom* =  
  *aAgt agent*  
| *aNon nonce*  
| *aKey key*  
| *aNum nat*

### 1.5.2 Long-term key setup (abstractly)

Suppose an initial long-term key setup without looking into the structure of long-term keys.  
Remark: This setup is agnostic with respect to the structure of the type *ltkey*. Ideally, the type *ltkey* should be a parameter of the type *key*, which is instantiated only at Level 3.

**consts**  
*ltkeySetup* ::  $(\text{ltkey} \times \text{agent}) \text{ set}$  — LT key setup, for now unspecified

The initial key setup contains static, long-term keys.

**definition**  
*keySetup* ::  $(\text{key} \times \text{agent}) \text{ set}$  **where**

$keySetup \equiv \{(ltK K, A) \mid K A. (K, A) \in ltkeySetup\}$

Corrupted keys are the long-term keys known by bad agents.

**definition**

$corrKey :: key\ set$  **where**  
 $corrKey \equiv keySetup^{-1} \text{ “ bad}$

**lemma** *corrKey-Dom-keySetup* [simp, intro]:  $K \in corrKey \implies K \in Domain\ keySetup$   
 ⟨proof⟩

**lemma** *keySetup-noSessionKeys* [simp]:  $(sesK K, A) \notin keySetup$   
 ⟨proof⟩

**lemma** *corrKey-noSessionKeys* [simp]:  $sesK K \notin corrKey$   
 ⟨proof⟩

**end**

## 1.6 Protocol runs

**theory** *Runs* **imports** *Atoms*  
**begin**

### 1.6.1 Runs

Define some typical roles.

**datatype** *role-t* = *Init* | *Resp* | *Serv*

**fun**

$roleIdx :: role-t \Rightarrow nat$

**where**

$roleIdx\ Init = 0$

|  $roleIdx\ Resp = 1$

|  $roleIdx\ Serv = 2$

The type of runs is a partial function from run identifiers to a triple consisting of a role, a list of agents, and a list of atomic messages recorded during the run’s execution.

The type of roles could be made a parameter for more flexibility.

**type-synonym**

$rid-t = fid-t$

**type-synonym**

$runs-t = rid-t \rightarrow role-t \times agent\ list \times atom\ list$

### 1.6.2 Run abstraction

Define a function that lifts a function on roles and atom lists to a function on runs.

**definition**

$map-runs :: (role-t, atom\ list) \Rightarrow atom\ list \Rightarrow runs-t \Rightarrow runs-t$

**where**

*map-runs h runz rid*  $\equiv$  *case runz rid of*  
  *None*  $\Rightarrow$  *None*  
  | *Some (rol, agts, al)*  $\Rightarrow$  *Some (rol, agts, h rol al)*

**lemma** *map-runs-empty* [*simp*]: *map-runs h Map.empty = Map.empty*  
*<proof>*

**lemma** *map-runs-dom* [*simp*]: *dom (map-runs h runz) = dom runz*  
*<proof>*

**lemma** *map-runs-update* [*simp*]:  
  *map-runs h (runz(R  $\mapsto$  (rol, agts, al)))*  
  = *(map-runs h runz)(R  $\mapsto$  (rol, agts, h rol al))*  
*<proof>*

**end**

## 1.7 Channel Messages

**theory** *Channels* **imports** *Atoms*  
**begin**

### 1.7.1 Channel messages

**datatype** *secprop* = *auth* | *confid*

**type-synonym**  
*chtyp* = *secprop set*

**abbreviation**  
*secure* :: *chtyp* **where**  
*secure*  $\equiv$  {*auth*, *confid*}

**datatype** *payload* = *Msg atom list*

**datatype** *chmsg* =  
  *StatCh chtyp agent agent payload*  
  | *DynCh chtyp key payload*

Abbreviations for use in protocol defs

**abbreviation**  
*Insec* :: [*agent, agent, payload*]  $\Rightarrow$  *chmsg* **where**  
*Insec*  $\equiv$  *StatCh {}*

**abbreviation**  
*Confid* :: [*agent, agent, payload*]  $\Rightarrow$  *chmsg* **where**  
*Confid*  $\equiv$  *StatCh {confid}*

**abbreviation**

$Auth :: [agent, agent, payload] \Rightarrow chmsg$  **where**  
 $Auth \equiv StatCh \{auth\}$

**abbreviation**

$Secure :: [agent, agent, payload] \Rightarrow chmsg$  **where**  
 $Secure \equiv StatCh \{auth, confid\}$

**abbreviation**

$dConfid :: [key, payload] \Rightarrow chmsg$  **where**  
 $dConfid \equiv DynCh \{confid\}$

**abbreviation**

$dAuth :: [key, payload] \Rightarrow chmsg$  **where**  
 $dAuth \equiv DynCh \{auth\}$

**abbreviation**

$dSecure :: [key, payload] \Rightarrow chmsg$  **where**  
 $dSecure \equiv DynCh \{auth, confid\}$

### 1.7.2 Keys used in dynamic channel messages

**definition**

$keys-for :: chmsg\ set \Rightarrow key\ set$  **where**  
 $keys-for\ H \equiv \{K. \exists c\ M. DynCh\ c\ K\ M \in H\}$

**lemma**  $keys-forI$   $[dest]: DynCh\ c\ K\ M \in H \implies K \in keys-for\ H$   
 $\langle proof \rangle$

**lemma**  $keys-for-empty$   $[simp]: keys-for\ \{\} = \{\}$   
 $\langle proof \rangle$

**lemma**  $keys-for-monotone: G \subseteq H \implies keys-for\ G \subseteq keys-for\ H$   
 $\langle proof \rangle$

**lemmas**  $keys-for-mono$   $[elim] = keys-for-monotone$   $[THEN\ [2]\ rev-subsetD]$

**lemma**  $keys-for-insert-StatCh$   $[simp]:$   
 $keys-for\ (insert\ (StatCh\ c\ A\ B\ M)\ H) = keys-for\ H$   
 $\langle proof \rangle$

**lemma**  $keys-for-insert-DynCh$   $[simp]:$   
 $keys-for\ (insert\ (DynCh\ c\ K\ M)\ H) = insert\ K\ (keys-for\ H)$   
 $\langle proof \rangle$

### 1.7.3 Atoms in a set of channel messages

The set of atoms contained in a set of channel messages. We also include the public atoms, i.e., the agent names, numbers, and corrupted keys.

**inductive-set**

$atoms :: chmsg\ set \Rightarrow atom\ set$

**for**  $H :: \text{chmsg set}$   
**where**  
 $\text{at-StatCh}: \llbracket \text{StatCh } c \ A \ B \ (\text{Msg } M) \in H; \text{At} \in \text{set } M \rrbracket \implies \text{At} \in \text{atoms } H$   
 $\text{at-DynCh}: \llbracket \text{DynCh } c \ K \ (\text{Msg } M) \in H; \text{At} \in \text{set } M \rrbracket \implies \text{At} \in \text{atoms } H$

**declare**  $\text{atoms.intros}$  [intro]

**lemma**  $\text{atoms-empty}$  [simp]:  $\text{atoms } \{\} = \{\}$   
 ⟨proof⟩

**lemma**  $\text{atoms-monotone}$ :  $G \subseteq H \implies \text{atoms } G \subseteq \text{atoms } H$   
 ⟨proof⟩

**lemmas**  $\text{atoms-mono}$  [elim] =  $\text{atoms-monotone}$  [THEN [2] rev-subsetD]

**lemma**  $\text{atoms-insert-StatCh}$  [simp]:  
 $\text{atoms } (\text{insert } (\text{StatCh } c \ A \ B \ (\text{Msg } M)) \ H) = \text{set } M \cup \text{atoms } H$   
 ⟨proof⟩

**lemma**  $\text{atoms-insert-DynCh}$  [simp]:  
 $\text{atoms } (\text{insert } (\text{DynCh } c \ K \ (\text{Msg } M)) \ H) = \text{set } M \cup \text{atoms } H$   
 ⟨proof⟩

## 1.7.4 Intruder knowledge (atoms)

Atoms that the intruder can extract from a set of channel messages.

**inductive-set**

$\text{extr} :: \text{atom set} \Rightarrow \text{chmsg set} \Rightarrow \text{atom set}$

**for**  $T :: \text{atom set}$

**and**  $H :: \text{chmsg set}$

**where**

$\text{extr-Inj}: \text{At} \in T \implies \text{At} \in \text{extr } T \ H$

|  $\text{extr-StatCh}$ :

$\llbracket \text{StatCh } c \ A \ B \ (\text{Msg } M) \in H; \text{At} \in \text{set } M; \text{confid} \notin c \vee A \in \text{bad} \vee B \in \text{bad} \rrbracket$   
 $\implies \text{At} \in \text{extr } T \ H$

|  $\text{extr-DynCh}$ :

$\llbracket \text{DynCh } c \ K \ (\text{Msg } M) \in H; \text{At} \in \text{set } M; \text{confid} \notin c \vee a\text{Key } K \in \text{extr } T \ H \rrbracket$   
 $\implies \text{At} \in \text{extr } T \ H$

**declare**  $\text{extr.intros}$  [intro]

**declare**  $\text{extr.cases}$  [elim]

Typical parameter describing initial intruder knowledge.

**definition**

$\text{ik0} :: \text{atom set}$  **where**

$\text{ik0} \equiv \text{range } a\text{Agt} \cup \text{range } a\text{Num} \cup a\text{Key}'\text{corrKey}$

**lemma**  $\text{ik0-aAgt}$  [iff]:  $a\text{Agt } A \in \text{ik0}$   
 ⟨proof⟩

**lemma**  $\text{ik0-aNum}$  [iff]:  $a\text{Num } T \in \text{ik0}$

$\langle \text{proof} \rangle$

**lemma** *ik0-aNon* [*iff*]:  $aNon\ N \notin ik0$

$\langle \text{proof} \rangle$

**lemma** *ik0-aKey-corr* [*simp*]:  $(aKey\ K \in ik0) = (K \in corrKey)$

$\langle \text{proof} \rangle$

### Basic lemmas

**lemma** *extr-empty* [*simp*]:  $extr\ T\ \{\} = T$

$\langle \text{proof} \rangle$

**lemma** *extr-monotone* [*dest*]:  $G \subseteq H \implies extr\ T\ G \subseteq extr\ T\ H$

$\langle \text{proof} \rangle$

**lemmas** *extr-mono* [*elim*] = *extr-monotone* [*THEN* [2] *rev-subsetD*]

**lemma** *extr-monotone-param* [*dest*]:  $T \subseteq U \implies extr\ T\ H \subseteq extr\ U\ H$

$\langle \text{proof} \rangle$

**lemmas** *extr-mono-param* [*elim*] = *extr-monotone-param* [*THEN* [2] *rev-subsetD*]

**lemma** *extr-insert* [*intro*]:  $At \in extr\ T\ H \implies At \in extr\ T\ (insert\ C\ H)$

$\langle \text{proof} \rangle$

**lemma** *extr-into-atoms* [*dest*]:  $At \in extr\ T\ H \implies At \in T \cup atoms\ H$

$\langle \text{proof} \rangle$

### Insertion lemmas for atom parameters

**lemma** *extr-insert-non-key-param* [*simp*]:

**assumes**  $At \in range\ aNon \cup range\ aAgt \cup range\ aNum$

**shows**  $extr\ (insert\ At\ T)\ H = insert\ At\ (extr\ T\ H)$

$\langle \text{proof} \rangle$

**lemma** *extr-insert-unused-key-param* [*simp*]:

**assumes**  $K \notin keys\ for\ H$

**shows**  $extr\ (insert\ (aKey\ K)\ T)\ H = insert\ (aKey\ K)\ (extr\ T\ H)$

$\langle \text{proof} \rangle$

### Insertion lemmas for each type of channel message

Note that the parameter accumulates the extracted atoms. In particular, these may include keys that may open further dynamically confidential messages.

**lemma** *extr-insert-StatCh* [*simp*]:

$extr\ T\ (insert\ (StatCh\ c\ A\ B\ (Msg\ M))\ H)$

$= (if\ confid\ \notin\ c \vee A \in bad \vee B \in bad\ then\ extr\ (set\ M \cup T)\ H\ else\ extr\ T\ H)$

$\langle \text{proof} \rangle$

**lemma** *extr-insert-DynCh* [*simp*]:

$extr\ T\ (insert\ (DynCh\ c\ K\ (Msg\ M))\ H)$

$= (\text{if } \text{confid} \notin c \vee \text{aKey } K \in \text{extr } T \ H \text{ then } \text{extr } (\text{set } M \cup T) \ H \text{ else } \text{extr } T \ H)$   
 <proof>

**declare** *extr.cases* [rule del, elim]

### 1.7.5 Faking messages

Channel messages that are fakeable from a given set of channel messages. Parameters are a set of atoms and a set of freshness identifiers.

For faking messages on dynamic non-authentic channels, we cannot allow the intruder to use arbitrary keys. Otherwise, we would lose the possibility to generate fresh values in our model. Therefore, the chosen keys must correspond to session keys associated with existing runs (i.e., from set *rkeys* *U*).

#### abbreviation

*rkeys* :: *fid-t set*  $\Rightarrow$  *key set* **where**  
*rkeys* *U*  $\equiv$  *sesK*'( $\lambda(x, y). x \ \$ \ y$ )'(U  $\times$  (UNIV::*nat set*))

**lemma** *rkeys-sesK* [*simp*, *dest*]: *sesK* (*R*\$*i*)  $\in$  *rkeys* *U*  $\Longrightarrow$  *R*  $\in$  *U*  
 <proof>

#### inductive-set

*fake* :: *atom set*  $\Rightarrow$  *fid-t set*  $\Rightarrow$  *chmsg set*  $\Rightarrow$  *chmsg set*  
**for** *T* :: *atom set*  
**and** *U* :: *fid-t set*  
**and** *H* :: *chmsg set*

#### where

*fake-Inj*:  
 $M \in H \Longrightarrow M \in \text{fake } T \ U \ H$   
 | *fake-StatCh*:  
 $\llbracket \text{set } M \subseteq \text{extr } T \ H; \text{auth} \notin c \vee A \in \text{bad} \vee B \in \text{bad} \rrbracket$   
 $\Longrightarrow \text{StatCh } c \ A \ B \ (\text{Msg } M) \in \text{fake } T \ U \ H$   
 | *fake-DynCh*:  
 $\llbracket \text{set } M \subseteq \text{extr } T \ H; \text{auth} \notin c \wedge K \in \text{rkeys } U \vee \text{aKey } K \in \text{extr } T \ H \rrbracket$   
 $\Longrightarrow \text{DynCh } c \ K \ (\text{Msg } M) \in \text{fake } T \ U \ H$

**declare** *fake.cases* [*elim*]

**declare** *fake.intros* [*intro*]

**lemmas** *fake-intros* = *fake-StatCh fake-DynCh*

**lemma** *fake-expanding* [*intro*]:  $H \subseteq \text{fake } T \ U \ H$   
 <proof>

**lemma** *fake-monotone* [*intro*]:  $G \subseteq H \Longrightarrow \text{fake } T \ U \ G \subseteq \text{fake } T \ U \ H$   
 <proof>

**lemma** *fake-monotone-param1* [*intro*]:  
 $T \subseteq T' \Longrightarrow \text{fake } T \ U \ H \subseteq \text{fake } T' \ U \ H$   
 <proof>

**lemmas** *fake-mono* [elim] = *fake-monotone* [THEN [2] rev-subsetD]  
**lemmas** *fake-mono-param1* [elim] = *fake-monotone-param1* [THEN [2] rev-subsetD]

### Atoms and extr together with fake

**lemma** *atoms-fake* [simp]:  $atoms (fake T U H) = T \cup atoms H$   
 ⟨proof⟩

**lemma** *extr-fake* [simp]:  
**assumes**  $T' \subseteq T$  **shows**  $extr T (fake T' U H) = extr T H$   
 ⟨proof⟩

end

## 1.8 Theory of Agents and Messages for Security Protocols

**theory** *Message* **imports** *Keys* **begin**

**lemma** *Un-idem-collapse* [simp]:  $A \cup (B \cup A) = B \cup A$   
 ⟨proof⟩

### datatype

*msg* = *Agent agent* — Agent names  
 | *Number nat* — Ordinary integers, timestamps, ...  
 | *Nonce nonce* — Unguessable nonces  
 | *Key key* — Crypto keys  
 | *Hash msg* — Hashing  
 | *MPair msg msg* — Compound messages  
 | *Crypt key msg* — Encryption, public- or shared-key

Concrete syntax: messages appear as  $\{A, B, NA\}$ , etc...

### syntax

*-MTuple* ::  $[ 'a, args ] \Rightarrow 'a * 'b$  ( $\langle \langle \text{indent}=2 \text{ notation}=\langle \text{mixfix message tuple} \rangle \{ -, / - \} \rangle \rangle$ )

### syntax-consts

*-MTuple* == *MPair*

### translations

$\{x, y, z\} == \{x, \{y, z\}\}$   
 $\{x, y\} == \text{CONST MPair } x y$

### definition

*HPair* ::  $[msg, msg] \Rightarrow msg$  ( $\langle \langle 4Hash[-] /- \rangle \rangle [0, 1000]$ )

### where

— Message Y paired with a MAC computed with the help of X  
 $Hash[X] Y \equiv \{Hash\{X, Y\}, Y\}$

### definition

$keysFor :: msg\ set \Rightarrow key\ set$

**where**

— Keys useful to decrypt elements of a message set

$keysFor\ H \equiv invKey\ \{K. \exists X. Crypt\ K\ X \in H\}$

## Inductive Definition of All Parts" of a Message

**inductive-set**

$parts :: msg\ set \Rightarrow msg\ set$

**for**  $H :: msg\ set$

**where**

$Inj\ [intro]: \quad X \in H \implies X \in parts\ H$   
 $| Fst: \quad \{X, Y\} \in parts\ H \implies X \in parts\ H$   
 $| Snd: \quad \{X, Y\} \in parts\ H \implies Y \in parts\ H$   
 $| Body: \quad Crypt\ K\ X \in parts\ H \implies X \in parts\ H$

Monotonicity

**lemma**  $parts\ mono: G \subseteq H \implies parts(G) \subseteq parts(H)$

$\langle proof \rangle$

Equations hold because constructors are injective.

**lemma**  $Other\ image\ eq\ [simp]: (Agent\ x \in Agent\ A) = (x:A)$

$\langle proof \rangle$

**lemma**  $Key\ image\ eq\ [simp]: (Key\ x \in Key\ A) = (x \in A)$

$\langle proof \rangle$

**lemma**  $Nonce\ Key\ image\ eq\ [simp]: (Nonce\ x \notin Key\ A)$

$\langle proof \rangle$

### 1.8.1 keysFor operator

**lemma**  $keysFor\ empty\ [simp]: keysFor\ \{\} = \{\}$

$\langle proof \rangle$

**lemma**  $keysFor\ Un\ [simp]: keysFor\ (H \cup H') = keysFor\ H \cup keysFor\ H'$

$\langle proof \rangle$

**lemma**  $keysFor\ UN\ [simp]: keysFor\ (\bigcup_{i \in A} H\ i) = (\bigcup_{i \in A} keysFor\ (H\ i))$

$\langle proof \rangle$

Monotonicity

**lemma**  $keysFor\ mono: G \subseteq H \implies keysFor(G) \subseteq keysFor(H)$

$\langle proof \rangle$

**lemma**  $keysFor\ insert\ Agent\ [simp]: keysFor\ (insert\ (Agent\ A)\ H) = keysFor\ H$

$\langle proof \rangle$

**lemma**  $keysFor\ insert\ Nonce\ [simp]: keysFor\ (insert\ (Nonce\ N)\ H) = keysFor\ H$

$\langle proof \rangle$

**lemma**  $keysFor\ insert\ Number\ [simp]: keysFor\ (insert\ (Number\ N)\ H) = keysFor\ H$

$\langle \text{proof} \rangle$

**lemma** *keysFor-insert-Key* [simp]:  $\text{keysFor} (\text{insert} (\text{Key } K) H) = \text{keysFor } H$   
 $\langle \text{proof} \rangle$

**lemma** *keysFor-insert-Hash* [simp]:  $\text{keysFor} (\text{insert} (\text{Hash } X) H) = \text{keysFor } H$   
 $\langle \text{proof} \rangle$

**lemma** *keysFor-insert-MPair* [simp]:  $\text{keysFor} (\text{insert} \{X, Y\} H) = \text{keysFor } H$   
 $\langle \text{proof} \rangle$

**lemma** *keysFor-insert-Crypt* [simp]:  
 $\text{keysFor} (\text{insert} (\text{Crypt } K X) H) = \text{insert} (\text{invKey } K) (\text{keysFor } H)$   
 $\langle \text{proof} \rangle$

**lemma** *keysFor-image-Key* [simp]:  $\text{keysFor} (\text{Key } E) = \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *Crypt-imp-invKey-keysFor*:  $\text{Crypt } K X \in H \implies \text{invKey } K \in \text{keysFor } H$   
 $\langle \text{proof} \rangle$

## 1.8.2 Inductive relation "parts"

**lemma** *MPair-parts*:  
 $\llbracket \{X, Y\} \in \text{parts } H; \llbracket X \in \text{parts } H; Y \in \text{parts } H \rrbracket \implies P \rrbracket \implies P$   
 $\langle \text{proof} \rangle$

**declare** *MPair-parts* [elim!] *parts.Body* [dest!]

NB These two rules are UNSAFE in the formal sense, as they discard the compound message. They work well on THIS FILE. *MPair-parts* is left as SAFE because it speeds up proofs. The *Crypt* rule is normally kept UNSAFE to avoid breaking up certificates.

**lemma** *parts-increasing*:  $H \subseteq \text{parts}(H)$   
 $\langle \text{proof} \rangle$

**lemmas** *parts-insertI = subset-insertI* [THEN *parts-mono*, THEN *subsetD*]

**lemma** *parts-empty* [simp]:  $\text{parts}\{\} = \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *parts-emptyE* [elim!]:  $X \in \text{parts}\{\} \implies P$   
 $\langle \text{proof} \rangle$

WARNING: loops if  $H = Y$ , therefore must not be repeated!

**lemma** *parts-singleton*:  $X \in \text{parts } H \implies \exists Y \in H. X \in \text{parts } \{Y\}$   
 $\langle \text{proof} \rangle$

## Unions

**lemma** *parts-Un-subset1*:  $\text{parts}(G) \cup \text{parts}(H) \subseteq \text{parts}(G \cup H)$   
 $\langle \text{proof} \rangle$

**lemma** *parts-Un-subset2*:  $parts(G \cup H) \subseteq parts(G) \cup parts(H)$   
*<proof>*

**lemma** *parts-Un [simp]*:  $parts(G \cup H) = parts(G) \cup parts(H)$   
*<proof>*

**lemma** *parts-insert*:  $parts(insert\ X\ H) = parts\ \{X\} \cup parts\ H$   
*<proof>*

TWO inserts to avoid looping. This rewrite is better than nothing. Not suitable for Addsimps: its behaviour can be strange.

**lemma** *parts-insert2*:  
 $parts(insert\ X\ (insert\ Y\ H)) = parts\ \{X\} \cup parts\ \{Y\} \cup parts\ H$   
*<proof>*

Added to simplify arguments to parts, analz and synth.

This allows *blast* to simplify occurrences of  $parts(G \cup H)$  in the assumption.

**lemmas** *in-parts-UnE* = *parts-Un [THEN equalityD1, THEN subsetD, THEN UnE]*  
**declare** *in-parts-UnE [elim!]*

**lemma** *parts-insert-subset*:  $insert\ X\ (parts\ H) \subseteq parts(insert\ X\ H)$   
*<proof>*

### Idempotence and transitivity

**lemma** *parts-partsD [dest!]*:  $X \in parts(parts\ H) ==> X \in parts\ H$   
*<proof>*

**lemma** *parts-idem [simp]*:  $parts(parts\ H) = parts\ H$   
*<proof>*

**lemma** *parts-subset-iff [simp]*:  $(parts\ G \subseteq parts\ H) = (G \subseteq parts\ H)$   
*<proof>*

**lemma** *parts-trans*:  $[| X \in parts\ G; G \subseteq parts\ H |] ==> X \in parts\ H$   
*<proof>*

Cut

**lemma** *parts-cut*:  
 $[| Y \in parts(insert\ X\ G); X \in parts\ H |] ==> Y \in parts(G \cup H)$   
*<proof>*

**lemma** *parts-cut-eq [simp]*:  $X \in parts\ H ==> parts(insert\ X\ H) = parts\ H$   
*<proof>*

### Rewrite rules for pulling out atomic messages

**lemmas** *parts-insert-eq-I = equalityI [OF subsetI parts-insert-subset]*

**lemma** *parts-insert-Agent* [simp]:  
 $parts (insert (Agent\ agt)\ H) = insert (Agent\ agt)\ (parts\ H)$   
 <proof>

**lemma** *parts-insert-Nonce* [simp]:  
 $parts (insert (Nonce\ N)\ H) = insert (Nonce\ N)\ (parts\ H)$   
 <proof>

**lemma** *parts-insert-Number* [simp]:  
 $parts (insert (Number\ N)\ H) = insert (Number\ N)\ (parts\ H)$   
 <proof>

**lemma** *parts-insert-Key* [simp]:  
 $parts (insert (Key\ K)\ H) = insert (Key\ K)\ (parts\ H)$   
 <proof>

**lemma** *parts-insert-Hash* [simp]:  
 $parts (insert (Hash\ X)\ H) = insert (Hash\ X)\ (parts\ H)$   
 <proof>

**lemma** *parts-insert-Crypt* [simp]:  
 $parts (insert (Crypt\ K\ X)\ H) = insert (Crypt\ K\ X)\ (parts\ (insert\ X\ H))$   
 <proof>

**lemma** *parts-insert-MPair* [simp]:  
 $parts (insert \{X, Y\}\ H) =$   
 $insert \{X, Y\}\ (parts (insert\ X\ (insert\ Y\ H)))$   
 <proof>

**lemma** *parts-image-Key* [simp]:  $parts (Key'N) = Key'N$   
 <proof>

In any message, there is an upper bound N on its greatest nonce.

### 1.8.3 Inductive relation "analz"

Inductive definition of "analz" – what can be broken down from a set of messages, including keys. A form of downward closure. Pairs can be taken apart; messages decrypted with known keys.

#### inductive-set

*analz* :: *msg set* ==> *msg set*

**for** *H* :: *msg set*

**where**

*Inj* [intro,simp] :  $X \in H \implies X \in analz\ H$

| *Fst*:  $\{X, Y\} \in analz\ H \implies X \in analz\ H$

| *Snd*:  $\{X, Y\} \in analz\ H \implies Y \in analz\ H$

| *Decrypt* [dest]:

$[(Crypt\ K\ X \in analz\ H; Key(invKey\ K): analz\ H)] \implies X \in analz\ H$

Monotonicity; Lemma 1 of Lowe's paper

**lemma** *analz-mono*:  $G \subseteq H \implies \text{analz}(G) \subseteq \text{analz}(H)$   
*<proof>*

**lemmas** *analz-monotonic* = *analz-mono* [THEN [2] *rev-subsetD*]

Making it safe speeds up proofs

**lemma** *MPair-analz* [elim!]:  
[[  $\{X, Y\} \in \text{analz } H$ ;  
[[  $X \in \text{analz } H$ ;  $Y \in \text{analz } H$  ]]  $\implies P$  ]]  
[[  $\implies P$  ]]  
*<proof>*

**lemma** *analz-increasing*:  $H \subseteq \text{analz}(H)$   
*<proof>*

**lemma** *analz-subset-parts*:  $\text{analz } H \subseteq \text{parts } H$   
*<proof>*

**lemmas** *analz-into-parts* = *analz-subset-parts* [THEN *subsetD*]

**lemmas** *not-parts-not-analz* = *analz-subset-parts* [THEN *contra-subsetD*]

**lemma** *parts-analz* [simp]:  $\text{parts } (\text{analz } H) = \text{parts } H$   
*<proof>*

**lemma** *analz-parts* [simp]:  $\text{analz } (\text{parts } H) = \text{parts } H$   
*<proof>*

**lemmas** *analz-insertI* = *subset-insertI* [THEN *analz-mono*, THEN [2] *rev-subsetD*]

### General equational properties

**lemma** *analz-empty* [simp]:  $\text{analz}\{\} = \{\}$   
*<proof>*

Converse fails: we can *analz* more from the union than from the separate parts, as a key in one might decrypt a message in the other

**lemma** *analz-Un*:  $\text{analz}(G) \cup \text{analz}(H) \subseteq \text{analz}(G \cup H)$   
*<proof>*

**lemma** *analz-insert*:  $\text{insert } X (\text{analz } H) \subseteq \text{analz}(\text{insert } X H)$   
*<proof>*

### Rewrite rules for pulling out atomic messages

**lemmas** *analz-insert-eq-I* = *equalityI* [OF *subsetI analz-insert*]

**lemma** *analz-insert-Agent* [simp]:  
 $\text{analz } (\text{insert } (\text{Agent } \text{agt}) H) = \text{insert } (\text{Agent } \text{agt}) (\text{analz } H)$   
*<proof>*

**lemma** *analz-insert-Nonce* [simp]:

$analz (insert (Nonce N) H) = insert (Nonce N) (analz H)$   
 $\langle proof \rangle$

**lemma** *analz-insert-Number* [simp]:  
 $analz (insert (Number N) H) = insert (Number N) (analz H)$   
 $\langle proof \rangle$

**lemma** *analz-insert-Hash* [simp]:  
 $analz (insert (Hash X) H) = insert (Hash X) (analz H)$   
 $\langle proof \rangle$

Can only pull out Keys if they are not needed to decrypt the rest

**lemma** *analz-insert-Key* [simp]:  
 $K \notin keysFor (analz H) ==>$   
 $analz (insert (Key K) H) = insert (Key K) (analz H)$   
 $\langle proof \rangle$

**lemma** *analz-insert-MPair* [simp]:  
 $analz (insert \{X, Y\} H) =$   
 $insert \{X, Y\} (analz (insert X (insert Y H)))$   
 $\langle proof \rangle$

Can pull out enCrypted message if the Key is not known

**lemma** *analz-insert-Crypt*:  
 $Key (invKey K) \notin analz H$   
 $==> analz (insert (Crypt K X) H) = insert (Crypt K X) (analz H)$   
 $\langle proof \rangle$

**lemma** *lemma1*:  $Key (invKey K) \in analz H ==>$   
 $analz (insert (Crypt K X) H) \subseteq$   
 $insert (Crypt K X) (analz (insert X H))$   
 $\langle proof \rangle$

**lemma** *lemma2*:  $Key (invKey K) \in analz H ==>$   
 $insert (Crypt K X) (analz (insert X H)) \subseteq$   
 $analz (insert (Crypt K X) H)$   
 $\langle proof \rangle$

**lemma** *analz-insert-Decrypt*:  
 $Key (invKey K) \in analz H ==>$   
 $analz (insert (Crypt K X) H) =$   
 $insert (Crypt K X) (analz (insert X H))$   
 $\langle proof \rangle$

Case analysis: either the message is secure, or it is not! Effective, but can cause subgoals to blow up! Use with *split-if*; apparently *split-tac* does not cope with patterns such as  $analz (insert (Crypt K X) H)$

**lemma** *analz-Crypt-if* [simp]:  
 $analz (insert (Crypt K X) H) =$   
 $(if (Key (invKey K) \in analz H)$   
 $then insert (Crypt K X) (analz (insert X H))$   
 $else insert (Crypt K X) (analz H))$

$\langle proof \rangle$

This rule supposes "for the sake of argument" that we have the key.

**lemma** *analz-insert-Crypt-subset*:

$$\begin{aligned} \text{analz } (\text{insert } (\text{Crypt } K \ X) \ H) &\subseteq \\ &\text{insert } (\text{Crypt } K \ X) \ (\text{analz } (\text{insert } X \ H)) \end{aligned}$$

$\langle proof \rangle$

**lemma** *analz-image-Key [simp]*:  $\text{analz } (\text{Key}'N) = \text{Key}'N$

$\langle proof \rangle$

### Idempotence and transitivity

**lemma** *analz-analzD [dest!]*:  $X \in \text{analz } (\text{analz } H) \implies X \in \text{analz } H$

$\langle proof \rangle$

**lemma** *analz-idem [simp]*:  $\text{analz } (\text{analz } H) = \text{analz } H$

$\langle proof \rangle$

**lemma** *analz-subset-iff [simp]*:  $(\text{analz } G \subseteq \text{analz } H) = (G \subseteq \text{analz } H)$

$\langle proof \rangle$

**lemma** *analz-trans*:  $[| X \in \text{analz } G; G \subseteq \text{analz } H |] \implies X \in \text{analz } H$

$\langle proof \rangle$

Cut; Lemma 2 of Lowe

**lemma** *analz-cut*:  $[| Y \in \text{analz } (\text{insert } X \ H); X \in \text{analz } H |] \implies Y \in \text{analz } H$

$\langle proof \rangle$

This rewrite rule helps in the simplification of messages that involve the forwarding of unknown components (X). Without it, removing occurrences of X can be very complicated.

**lemma** *analz-insert-eq*:  $X \in \text{analz } H \implies \text{analz } (\text{insert } X \ H) = \text{analz } H$

$\langle proof \rangle$

A congruence rule for "analz"

**lemma** *analz-subset-cong*:

$$\begin{aligned} [| \text{analz } G \subseteq \text{analz } G'; \text{analz } H \subseteq \text{analz } H' |] \\ \implies \text{analz } (G \cup H) \subseteq \text{analz } (G' \cup H') \end{aligned}$$

$\langle proof \rangle$

**lemma** *analz-cong*:

$$\begin{aligned} [| \text{analz } G = \text{analz } G'; \text{analz } H = \text{analz } H' |] \\ \implies \text{analz } (G \cup H) = \text{analz } (G' \cup H') \end{aligned}$$

$\langle proof \rangle$

**lemma** *analz-insert-cong*:

$$\text{analz } H = \text{analz } H' \implies \text{analz}(\text{insert } X \ H) = \text{analz}(\text{insert } X \ H')$$

$\langle proof \rangle$

If there are no pairs or encryptions then analz does nothing

**lemma** *analz-trivial*:

$\langle proof \rangle$   
 $[[ \forall X Y. \{X, Y\} \notin H; \forall X K. Crypt\ K\ X \notin H ]] ==> analz\ H = H$

### 1.8.4 Inductive relation "synth"

Inductive definition of "synth" – what can be built up from a set of messages. A form of upward closure. Pairs can be built, messages encrypted with known keys. Agent names are public domain. Numbers can be guessed, but Nonces cannot be.

**inductive-set**

*synth* :: msg set => msg set

for *H* :: msg set

where

*Inj* [intro]:  $X \in H ==> X \in synth\ H$

| *Agent* [intro]:  $Agent\ agt \in synth\ H$

| *Number* [intro]:  $Number\ n \in synth\ H$

| *Hash* [intro]:  $X \in synth\ H ==> Hash\ X \in synth\ H$

| *MPair* [intro]:  $[[X \in synth\ H; Y \in synth\ H]] ==> \{X, Y\} \in synth\ H$

| *Crypt* [intro]:  $[[X \in synth\ H; Key(K) \in H]] ==> Crypt\ K\ X \in synth\ H$

Monotonicity

**lemma** *synth-mono*:  $G \subseteq H ==> synth(G) \subseteq synth(H)$

$\langle proof \rangle$

NO *Agent-synth*, as any Agent name can be synthesized. The same holds for *Number*

**inductive-cases** *Nonce-synth* [elim!]:  $Nonce\ n \in synth\ H$

**inductive-cases** *Key-synth* [elim!]:  $Key\ K \in synth\ H$

**inductive-cases** *Hash-synth* [elim!]:  $Hash\ X \in synth\ H$

**inductive-cases** *MPair-synth* [elim!]:  $\{X, Y\} \in synth\ H$

**inductive-cases** *Crypt-synth* [elim!]:  $Crypt\ K\ X \in synth\ H$

**lemma** *synth-increasing*:  $H \subseteq synth(H)$

$\langle proof \rangle$

### Unions

Converse fails: we can synth more from the union than from the separate parts, building a compound message using elements of each.

**lemma** *synth-Un*:  $synth(G) \cup synth(H) \subseteq synth(G \cup H)$

$\langle proof \rangle$

**lemma** *synth-insert*:  $insert\ X\ (synth\ H) \subseteq synth(insert\ X\ H)$

$\langle proof \rangle$

### Idempotence and transitivity

**lemma** *synth-synthD* [dest!]:  $X \in synth\ (synth\ H) ==> X \in synth\ H$

$\langle proof \rangle$

**lemma** *synth-idem*:  $synth\ (synth\ H) = synth\ H$

$\langle proof \rangle$

**lemma** *synth-subset-iff* [simp]:  $(\text{synth } G \subseteq \text{synth } H) = (G \subseteq \text{synth } H)$   
<proof>

**lemma** *synth-trans*:  $[| X \in \text{synth } G; G \subseteq \text{synth } H |] ==> X \in \text{synth } H$   
<proof>

Cut; Lemma 2 of Lowe

**lemma** *synth-cut*:  $[| Y \in \text{synth } (\text{insert } X H); X \in \text{synth } H |] ==> Y \in \text{synth } H$   
<proof>

**lemma** *Agent-synth* [simp]:  $\text{Agent } A \in \text{synth } H$   
<proof>

**lemma** *Number-synth* [simp]:  $\text{Number } n \in \text{synth } H$   
<proof>

**lemma** *Nonce-synth-eq* [simp]:  $(\text{Nonce } N \in \text{synth } H) = (\text{Nonce } N \in H)$   
<proof>

**lemma** *Key-synth-eq* [simp]:  $(\text{Key } K \in \text{synth } H) = (\text{Key } K \in H)$   
<proof>

**lemma** *Crypt-synth-eq* [simp]:  
 $\text{Key } K \notin H ==> (\text{Crypt } K X \in \text{synth } H) = (\text{Crypt } K X \in H)$   
<proof>

**lemma** *keysFor-synth* [simp]:  
 $\text{keysFor } (\text{synth } H) = \text{keysFor } H \cup \text{invKey}\{K. \text{Key } K \in H\}$   
<proof>

## Combinations of parts, analz and synth

**lemma** *parts-synth* [simp]:  $\text{parts } (\text{synth } H) = \text{parts } H \cup \text{synth } H$   
<proof>

**lemma** *analz-analz-Un* [simp]:  $\text{analz } (\text{analz } G \cup H) = \text{analz } (G \cup H)$   
<proof>

**lemma** *analz-synth-Un* [simp]:  $\text{analz } (\text{synth } G \cup H) = \text{analz } (G \cup H) \cup \text{synth } G$   
<proof>

**lemma** *analz-synth* [simp]:  $\text{analz } (\text{synth } H) = \text{analz } H \cup \text{synth } H$   
<proof>

chsp: added

**lemma** *analz-Un-analz* [simp]:  $\text{analz } (G \cup \text{analz } H) = \text{analz } (G \cup H)$   
<proof>

**lemma** *analz-synth-Un2* [simp]:  $\text{analz } (G \cup \text{synth } H) = \text{analz } (G \cup H) \cup \text{synth } H$   
<proof>

## For reasoning about the Fake rule in traces

**lemma** *parts-insert-subset-Un*:  $X \in G \implies \text{parts}(\text{insert } X \ H) \subseteq \text{parts } G \cup \text{parts } H$   
*<proof>*

More specifically for Fake. Very occasionally we could do with a version of the form  $\text{parts } \{X\} \subseteq \text{synth } (\text{analz } H) \cup \text{parts } H$

**lemma** *Fake-parts-insert*:  
 $X \in \text{synth } (\text{analz } H) \implies$   
 $\text{parts } (\text{insert } X \ H) \subseteq \text{synth } (\text{analz } H) \cup \text{parts } H$   
*<proof>*

**lemma** *Fake-parts-insert-in-Un*:  
 $[[Z \in \text{parts } (\text{insert } X \ H); X \in \text{synth } (\text{analz } H)]]$   
 $\implies Z \in \text{synth } (\text{analz } H) \cup \text{parts } H$   
*<proof>*

$H$  is sometimes *Key* ‘ $KK \cup \text{spies } \text{evs}$ , so can't put  $G = H$ .

**lemma** *Fake-analz-insert*:  
 $X \in \text{synth } (\text{analz } G) \implies$   
 $\text{analz } (\text{insert } X \ H) \subseteq \text{synth } (\text{analz } G) \cup \text{analz } (G \cup H)$   
*<proof>*

**lemma** *analz-conj-parts [simp]*:  
 $(X \in \text{analz } H \ \& \ X \in \text{parts } H) = (X \in \text{analz } H)$   
*<proof>*

**lemma** *analz-disj-parts [simp]*:  
 $(X \in \text{analz } H \ | \ X \in \text{parts } H) = (X \in \text{parts } H)$   
*<proof>*

Without this equation, other rules for *synth* and *analz* would yield redundant cases

**lemma** *MPair-synth-analz [iff]*:  
 $(\{X, Y\} \in \text{synth } (\text{analz } H)) =$   
 $(X \in \text{synth } (\text{analz } H) \ \& \ Y \in \text{synth } (\text{analz } H))$   
*<proof>*

**lemma** *Crypt-synth-analz*:  
 $[[ \text{Key } K \in \text{analz } H; \ \text{Key } (\text{invKey } K) \in \text{analz } H \ ]]$   
 $\implies (\text{Crypt } K \ X \in \text{synth } (\text{analz } H)) = (X \in \text{synth } (\text{analz } H))$   
*<proof>*

**lemma** *Hash-synth-analz [simp]*:  
 $X \notin \text{synth } (\text{analz } H)$   
 $\implies (\text{Hash } \{X, Y\} \in \text{synth } (\text{analz } H)) = (\text{Hash } \{X, Y\} \in \text{analz } H)$   
*<proof>*

## 1.8.5 HPair: a combination of Hash and MPair

### Freeness

**lemma** *Agent-neq-HPair*:  $\text{Agent } A \sim = \text{Hash}[X] \ Y$

$\langle proof \rangle$

**lemma** *Nonce-neq-HPair*:  $Nonce\ N \sim = Hash[X]\ Y$

$\langle proof \rangle$

**lemma** *Number-neq-HPair*:  $Number\ N \sim = Hash[X]\ Y$

$\langle proof \rangle$

**lemma** *Key-neq-HPair*:  $Key\ K \sim = Hash[X]\ Y$

$\langle proof \rangle$

**lemma** *Hash-neq-HPair*:  $Hash\ Z \sim = Hash[X]\ Y$

$\langle proof \rangle$

**lemma** *Crypt-neq-HPair*:  $Crypt\ K\ X' \sim = Hash[X]\ Y$

$\langle proof \rangle$

**lemmas** *HPair-neqs* = *Agent-neq-HPair* *Nonce-neq-HPair* *Number-neq-HPair*  
*Key-neq-HPair* *Hash-neq-HPair* *Crypt-neq-HPair*

**declare** *HPair-neqs* [*iff*]

**declare** *HPair-neqs* [*symmetric, iff*]

**lemma** *HPair-eq* [*iff*]:  $(Hash[X]\ Y' = Hash[X]\ Y) = (X' = X \ \& \ Y' = Y)$

$\langle proof \rangle$

**lemma** *MPair-eq-HPair* [*iff*]:

$(\{X', Y'\} = Hash[X]\ Y) = (X' = Hash\{X, Y\} \ \& \ Y' = Y)$

$\langle proof \rangle$

**lemma** *HPair-eq-MPair* [*iff*]:

$(Hash[X]\ Y = \{X', Y'\}) = (X' = Hash\{X, Y\} \ \& \ Y' = Y)$

$\langle proof \rangle$

### Specialized laws, proved in terms of those for Hash and MPair

**lemma** *keysFor-insert-HPair* [*simp*]:  $keysFor\ (insert\ (Hash[X]\ Y)\ H) = keysFor\ H$

$\langle proof \rangle$

**lemma** *parts-insert-HPair* [*simp*]:

$parts\ (insert\ (Hash[X]\ Y)\ H) =$   
 $insert\ (Hash[X]\ Y)\ (insert\ (Hash\{X, Y\})\ (parts\ (insert\ Y\ H)))$

$\langle proof \rangle$

**lemma** *analz-insert-HPair* [*simp*]:

$analz\ (insert\ (Hash[X]\ Y)\ H) =$   
 $insert\ (Hash[X]\ Y)\ (insert\ (Hash\{X, Y\})\ (analz\ (insert\ Y\ H)))$

$\langle proof \rangle$

**lemma** *HPair-synth-analz* [*simp*]:

$X \notin synth\ (analz\ H)$   
 $==> (Hash[X]\ Y \in synth\ (analz\ H)) =$   
 $(Hash\{X, Y\} \in analz\ H \ \& \ Y \in synth\ (analz\ H))$

*<proof>*

We do NOT want Crypt... messages broken up in protocols!!

**declare** *parts.Body* [rule del]

Rewrites to push in Key and Crypt messages, so that other messages can be pulled out using the *analz-insert* rules

**lemmas** *pushKeys* =

*insert-commute* [of Key K Agent C for K C]  
*insert-commute* [of Key K Nonce N for K N]  
*insert-commute* [of Key K Number N for K N]  
*insert-commute* [of Key K Hash X for K X]  
*insert-commute* [of Key K MPair X Y for K X Y]  
*insert-commute* [of Key K Crypt X K' for K K' X]

**lemmas** *pushCrypts* =

*insert-commute* [of Crypt X K Agent C for X K C]  
*insert-commute* [of Crypt X K Agent C for X K C]  
*insert-commute* [of Crypt X K Nonce N for X K N]  
*insert-commute* [of Crypt X K Number N for X K N]  
*insert-commute* [of Crypt X K Hash X' for X K X']  
*insert-commute* [of Crypt X K MPair X' Y for X K X' Y]

Cannot be added with [simp] – messages should not always be re-ordered.

**lemmas** *pushes* = *pushKeys pushCrypts*

By default only *o-apply* is built-in. But in the presence of eta-expansion this means that some terms displayed as  $f \circ g$  will be rewritten, and others will not!

**declare** *o-def* [simp]

**lemma** *Crypt-notin-image-Key* [simp]:  $Crypt\ K\ X \notin Key\ 'A$   
*<proof>*

**lemma** *Hash-notin-image-Key* [simp]:  $Hash\ X \notin Key\ 'A$   
*<proof>*

**lemma** *synth-analz-mono*:  $G \subseteq H \implies synth\ (analz\ (G)) \subseteq synth\ (analz\ (H))$   
*<proof>*

**lemma** *Fake-analz-eq* [simp]:  
 $X \in synth\ (analz\ H) \implies synth\ (analz\ (insert\ X\ H)) = synth\ (analz\ H)$   
*<proof>*

Two generalizations of *analz-insert-eq*

**lemma** *gen-analz-insert-eq* [rule-format]:  
 $X \in analz\ H \implies ALL\ G.\ H \subseteq G \implies analz\ (insert\ X\ G) = analz\ G$   
*<proof>*

**lemma** *synth-analz-insert-eq* [rule-format]:  
 $X \in synth\ (analz\ H)$

$\implies \text{ALL } G. H \subseteq G \dashrightarrow (\text{Key } K \in \text{analz } (\text{insert } X \ G)) = (\text{Key } K \in \text{analz } G)$   
 $\langle \text{proof} \rangle$

**lemma** *Fake-parts-sing*:

$X \in \text{synth } (\text{analz } H) \implies \text{parts}\{X\} \subseteq \text{synth } (\text{analz } H) \cup \text{parts } H$   
 $\langle \text{proof} \rangle$

**lemmas** *Fake-parts-sing-imp-Un* = *Fake-parts-sing* [THEN [2] *rev-subsetD*]

For some reason, moving this up can make some proofs loop!

**declare** *invKey-K* [*simp*]

**end**

## 1.9 Secrecy with Leaking (global version)

**theory** *s0g-secrecy* **imports** *Refinement Agents*  
**begin**

This model extends the global secrecy model by adding a *leak* event, which models that the adversary can learn messages through leaks of some (unspecified) kind.

Proof tool configuration. Avoid annoying automatic unfolding of *dom*.

**declare** *domIff* [*simp, iff del*]

### 1.9.1 State

The only state variable is a knowledge relation, an authorization relation, and a leakage relation.

$(d, A) \in \text{kn } s$  means that the agent  $A$  knows data  $d$ .  $(d, A) \in \text{az } s$  means that the agent  $A$  is authorized to know data  $d$ .  $(d, A) \in \text{lk } s$  means that data  $d$  has leaked to agent  $A$ . Leakage models potential unauthorized knowledge.

**record** *'d s0g-state* =  
*kn* :: (*'d* × *agent*) *set*  
*az* :: (*'d* × *agent*) *set*  
*lk* :: *'d set* — leaked data

**type-synonym**

*'d s0g-obs* = *'d s0g-state*

**abbreviation**

*lkr s* ≡ *lk s* × *UNIV*

### 1.9.2 Invariant definitions

Global secrecy is stated as an invariant.

**definition**

*s0g-secrecy* :: *'d s0g-state set*

**where**

$s0g\text{-secrecy} \equiv \{s. kn\ s \subseteq az\ s \cup lkr\ s\}$

**lemmas**  $s0g\text{-secrecy}I = s0g\text{-secrecy}\text{-def}$  [*THEN setc-def-to-intro, rule-format*]

**lemmas**  $s0g\text{-secrecy}E$  [*elim*] =  
 $s0g\text{-secrecy}\text{-def}$  [*THEN setc-def-to-elim, rule-format*]

Data that someone is authorized to know and leaked data is known by someone.

**definition**

$s0g\text{-dom} :: 'd\ s0g\text{-state}\ set$

**where**

$s0g\text{-dom} \equiv \{s. Domain\ (az\ s \cup lkr\ s) \subseteq Domain\ (kn\ s)\}$

**lemmas**  $s0g\text{-dom}I = s0g\text{-dom}\text{-def}$  [*THEN setc-def-to-intro, rule-format*]

**lemmas**  $s0g\text{-dom}E$  [*elim*] =  $s0g\text{-dom}\text{-def}$  [*THEN setc-def-to-elim, rule-format*]

### 1.9.3 Events

New secrets may be generated anytime.

**definition**

$s0g\text{-gen} :: ['d, agent, agent\ set] \Rightarrow ('d\ s0g\text{-state} \times 'd\ s0g\text{-state})\ set$

**where**

$s0g\text{-gen}\ d\ A\ G \equiv \{(s, s1).$

— guards:

$A \in G \wedge$

$d \notin Domain\ (kn\ s) \wedge$  — fresh item

— actions:

$s1 = s(|$

$kn := insert\ (d, A)\ (kn\ s),$

$az := az\ s \cup \{d\} \times (if\ G \cap bad = \{\} then\ G\ else\ UNIV)$

$|)$

$\}$

Learning secrets.

**definition**

$s0g\text{-learn} ::$

$['d, agent] \Rightarrow ('d\ s0g\text{-state} \times 'd\ s0g\text{-state})\ set$

**where**

$s0g\text{-learn}\ d\ B \equiv \{(s, s1).$

— guards:

—  $d \in Domain\ (kn\ s) \wedge$  someone knows  $d$  (follows from authorization)

— check authorization or leakage to preserve secrecy

$(d, B) \in az\ s \cup lkr\ s \wedge$

— actions:

$s1 = s(|\ kn := insert\ (d, B)\ (kn\ s)\ |)$

$\}$

Leaking secrets.

**definition**

$s0g\text{-leak} ::$   
 $'d \Rightarrow ('d\ s0g\text{-state} \times 'd\ s0g\text{-state})\ set$   
**where**  
 $s0g\text{-leak}\ d \equiv \{(s, s1).$   
— guards:  
 $d \in Domain\ (kn\ s) \wedge$  — someone knows  $d$   
— actions:  
 $s1 = s(\ lk := insert\ d\ (lk\ s) \ )$   
 $\}$

### 1.9.4 Specification

**definition**

$s0g\text{-init} :: 'd\ s0g\text{-state}\ set$

**where**

$s0g\text{-init} \equiv s0g\text{-secrecy} \cap s0g\text{-dom}$  — any state satisfying invariants

**definition**

$s0g\text{-trans} :: ('d\ s0g\text{-state} \times 'd\ s0g\text{-state})\ set$  **where**

$s0g\text{-trans} \equiv (\bigcup d\ A\ B\ G.$

$s0g\text{-gen}\ d\ A\ G \cup$

$s0g\text{-learn}\ d\ B \cup$

$s0g\text{-leak}\ d \cup$

$Id$

$\})$

**definition**

$s0g :: ('d\ s0g\text{-state}, 'd\ s0g\text{-obs})\ spec$  **where**

$s0g \equiv (\$

$init = s0g\text{-init},$

$trans = s0g\text{-trans},$

$obs = id$

$\})$

**lemmas**  $s0g\text{-defs} =$

$s0g\text{-def}\ s0g\text{-init}\text{-def}\ s0g\text{-trans}\text{-def}$

$s0g\text{-gen}\text{-def}\ s0g\text{-learn}\text{-def}\ s0g\text{-leak}\text{-def}$

**lemma**  $s0g\text{-obs}\text{-id}$  [simp]:  $obs\ s0g = id$

$\langle proof \rangle$

All state predicates are trivially observable.

**lemma**  $s0g\text{-any}\text{-}P\text{-observable}$  [iff]:  $observable\ (obs\ s0g)\ P$

$\langle proof \rangle$

### 1.9.5 Invariant proofs

#### 1.9.6 inv1: Secrecy

**lemma**  $PO\text{-}s0g\text{-secrecy}\text{-init}$  [iff]:

$init\ s0g \subseteq s0g\text{-secrecy}$

$\langle proof \rangle$

**lemma** *PO-s0g-secrecy-trans* [iff]:  
 $\{s0g\text{-secrecy}\} \text{ trans } s0g \{> s0g\text{-secrecy}\}$   
 ⟨proof⟩

**lemma** *PO-s0g-secrecy* [iff]:  $\text{reach } s0g \subseteq s0g\text{-secrecy}$   
 ⟨proof⟩

As an external invariant.

**lemma** *PO-s0g-obs-secrecy* [iff]:  $\text{oreach } s0g \subseteq s0g\text{-secrecy}$   
 ⟨proof⟩

### 1.9.7 inv2: Authorized and leaked data is known to someone

**lemma** *PO-s0g-dom-init* [iff]:  
 $\text{init } s0g \subseteq s0g\text{-dom}$   
 ⟨proof⟩

**lemma** *PO-s0g-dom-trans* [iff]:  
 $\{s0g\text{-dom}\} \text{ trans } s0g \{> s0g\text{-dom}\}$   
 ⟨proof⟩

**lemma** *PO-s0g-dom* [iff]:  $\text{reach } s0g \subseteq s0g\text{-dom}$   
 ⟨proof⟩

As an external invariant.

**lemma** *PO-s0g-obs-dom* [iff]:  $\text{oreach } s0g \subseteq s0g\text{-dom}$   
 ⟨proof⟩

**end**

## 1.10 Non-injective Agreement

**theory** *a0n-agree* **imports** *Refinement Agents*  
**begin**

The initial model abstractly specifies entity authentication, where one agent/role authenticates another. More precisely, this property corresponds to non-injective agreement on a data set  $ds$ . We use Running and Commit signals to obtain a protocol-independent extensional specification.

Proof tool configuration. Avoid annoying automatic unfolding of  $dom$ .

**declare**  $domIff$  [simp, iff del]

### 1.10.1 State

Signals. At this stage there are no protocol runs yet. All we model are the signals that indicate a certain progress of a protocol run by one agent/role (Commit signal) and the other role (Running signal). The signals contain a list of agents that are assumed to be honest and a polymorphic data set to be agreed upon, which is instantiated later.

Usually, the agent list will contain the names of the two agents who want to agree on the data, but sometimes one of the agents is honest by assumption (e.g., the server) or the honesty of additional agents needs to be assumed for the agreement to hold.

**datatype**  $'ds$  *signal* =  
*Running agent list 'ds*  
| *Commit agent list 'ds*

**record**  $'ds$  *a0n-state* =  
*signals* ::  $'ds$  *signal*  $\Rightarrow$  *nat* — multi-set of signals  
*corrupted* ::  $'ds$  *set* — set of corrupted data

**type-synonym**  
 $'ds$  *a0n-obs* =  $'ds$  *a0n-state*

### 1.10.2 Events

**definition**

$a0n-init$  ::  $'ds$  *a0n-state set*

**where**

$a0n-init \equiv \{s. \exists ds. s = ()$   
*signals* =  $\lambda s. 0$ ,  
*corrupted* =  $ds$   
 $\} \}$

Running signal, indicating end of responder run.

**definition**

$a0n-running$  :: [*agent list*,  $'ds$ ]  $\Rightarrow$  ( $'ds$  *a0n-state*  $\times$   $'ds$  *a0n-state*) *set*

**where**

$a0n-running$   $h$   $d \equiv \{(s, s')$ .  
— actions:  
 $s' = s()$   
*signals* := (*signals*  $s$ )(*Running*  $h$   $d$  := *signals*  $s$  (*Running*  $h$   $d$ ) + 1)  
 $\}$

Commit signal, marking end of initiator run.

**definition**

$a0n-commit$  :: [*agent list*,  $'ds$ ]  $\Rightarrow$  ( $'ds$  *a0n-state*  $\times$   $'ds$  *a0n-state*) *set*

**where**

$a0n-commit$   $h$   $d \equiv \{(s, s')$ .  
— guards:  
(*set*  $h \subseteq$  *good*  $\longrightarrow$   $d \notin$  *corrupted*  $s \longrightarrow$  *signals*  $s$  (*Running*  $h$   $d$ ) > 0)  $\wedge$   
— actions:  
 $s' = s()$   
*signals* := (*signals*  $s$ )(*Commit*  $h$   $d$  := *signals*  $s$  (*Commit*  $h$   $d$ ) + 1)  
 $\}$

Data corruption.

**definition**

$a0n\text{-corrupt} :: 'ds \text{ set} \Rightarrow ('ds \text{ a0n-state} \times 'ds \text{ a0n-state}) \text{ set}$

**where**

$a0n\text{-corrupt } ds \equiv \{(s, s')\}.$

— actions:

$s' = s \langle$

$\text{corrupted} := \text{corrupted } s \cup ds$

$\rangle$

$\}$

Transition system.

**definition**

$a0n\text{-trans} :: ('ds \text{ a0n-state} \times 'ds \text{ a0n-state}) \text{ set}$  **where**

$a0n\text{-trans} \equiv (\bigcup h \ d \ ds.$

$a0n\text{-running } h \ d \cup$

$a0n\text{-commit } h \ d \cup$

$a0n\text{-corrupt } ds \cup$

$Id$

$)$

**definition**

$a0n :: ('ds \text{ a0n-state}, 'ds \text{ a0n-obs}) \text{ spec}$  **where**

$a0n \equiv \langle$

$init = a0n\text{-init},$

$trans = a0n\text{-trans},$

$obs = id$

$\rangle$

**lemmas**  $a0n\text{-defs} =$

$a0n\text{-def } a0n\text{-init-def } a0n\text{-trans-def}$

$a0n\text{-running-def } a0n\text{-commit-def } a0n\text{-corrupt-def}$

Any property is trivially observable.

**lemma**  $a0n\text{-obs [simp]: } obs \ a0n = id$

$\langle proof \rangle$

**lemma**  $a0n\text{-anyP-observable [iff]: } observable \ (obs \ a0n) \ P$

$\langle proof \rangle$

### 1.10.3 Invariants

#### 1.10.4 inv1: non-injective agreement

This is an extensional variant of Lowe's *non-injective agreement* of the first with the second agent (by convention) in  $h$  on data  $d$  [Lowe97].

**definition**

$a0n\text{-inv1-niagree} :: 'ds \text{ a0n-state} \text{ set}$

**where**

$a0n\text{-inv1-niagree} \equiv \{s. \forall h \ d.$

$set \ h \subseteq good \longrightarrow d \notin corrupted \ s \longrightarrow$

$signals \ s \ (Commit \ h \ d) > 0 \longrightarrow signals \ s \ (Running \ h \ d) > 0$

$\}$

**lemmas** *a0n-inv1-niagreeI* =  
*a0n-inv1-niagree-def* [*THEN setc-def-to-intro, rule-format*]  
**lemmas** *a0n-inv1-niagreeE* [*elim*] =  
*a0n-inv1-niagree-def* [*THEN setc-def-to-elim, rule-format*]  
**lemmas** *a0n-inv1-niagreeD* =  
*a0n-inv1-niagree-def* [*THEN setc-def-to-dest, rule-format, rotated 2*]

Invariance proof.

**lemma** *PO-a0n-inv1-niagree-init* [*iff*]:  
*init a0n*  $\subseteq$  *a0n-inv1-niagree*  
*<proof>*

**lemma** *PO-a0n-inv1-niagree-trans* [*iff*]:  
 $\{a0n\text{-inv1-niagree}\}$  *trans a0n*  $\{> a0n\text{-inv1-niagree}\}$   
*<proof>*

**lemma** *PO-a0n-inv1-niagree* [*iff*]: *reach a0n*  $\subseteq$  *a0n-inv1-niagree*  
*<proof>*

This is also an external invariant.

**lemma** *a0n-obs-inv1-niagree* [*iff*]:  
*oreach a0n*  $\subseteq$  *a0n-inv1-niagree*  
*<proof>*

**end**

## 1.11 Injective Agreement

**theory** *a0i-agree* **imports** *a0n-agree*  
**begin**

This refinement adds injectiveness to the agreement property.

### 1.11.1 State

The state and observations are the same as in the previous model.

**type-synonym**  
*'d a0i-state* = *'d a0n-state*

**type-synonym**  
*'d a0i-obs* = *'d a0n-obs*

### 1.11.2 Events

We just refine the commit event. Everything else remains the same.

**abbreviation**  
*a0i-init* :: *'ds a0n-state set*  
**where**  
*a0i-init*  $\equiv$  *a0n-init*

**abbreviation**

$a0i\text{-running} :: [\text{agent list}, 'ds] \Rightarrow ('ds\ a0i\text{-state} \times 'ds\ a0i\text{-state})\ \text{set}$

**where**

$a0i\text{-running} \equiv a0n\text{-running}$

**definition**

$a0i\text{-commit} ::$

$[\text{agent list}, 'ds] \Rightarrow ('ds\ a0i\text{-state} \times 'ds\ a0i\text{-state})\ \text{set}$

**where**

$a0i\text{-commit}\ h\ d \equiv \{(s, s')\}.$

— guards:

$(\text{set}\ h \subseteq \text{good} \longrightarrow d \notin \text{corrupted}\ s \longrightarrow$   
 $\text{signals}\ s\ (\text{Commit}\ h\ d) < \text{signals}\ s\ (\text{Running}\ h\ d)) \wedge$

— actions:

$s' = s\{$   
 $\text{signals} := (\text{signals}\ s)(\text{Commit}\ h\ d := \text{signals}\ s\ (\text{Commit}\ h\ d) + 1)$   
 $\}$

**abbreviation**

$a0i\text{-corrupt} :: 'ds\ \text{set} \Rightarrow ('ds\ a0i\text{-state} \times 'ds\ a0i\text{-state})\ \text{set}$

**where**

$a0i\text{-corrupt} \equiv a0n\text{-corrupt}$

Transition system.

**definition**

$a0i\text{-trans} :: ('ds\ a0i\text{-state} \times 'ds\ a0i\text{-state})\ \text{set}$  **where**

$a0i\text{-trans} \equiv (\bigcup\ h\ d\ ds.$

$a0i\text{-running}\ h\ d \cup$

$a0i\text{-commit}\ h\ d \cup$

$a0i\text{-corrupt}\ ds \cup$

$Id$

)

**definition**

$a0i :: ('ds\ a0i\text{-state}, 'ds\ a0i\text{-obs})\ \text{spec}$  **where**

$a0i \equiv \{$

$\text{init} = a0i\text{-init},$

$\text{trans} = a0i\text{-trans},$

$\text{obs} = id$

$\}$

**lemmas**  $a0i\text{-defs} =$

$a0n\text{-defs}\ a0i\text{-def}\ a0i\text{-trans}\text{-def}\ a0i\text{-commit}\text{-def}$

Any property is trivially observable.

**lemma**  $a0i\text{-obs}$  [simp]:  $\text{obs}\ a0i = id$

$\langle\text{proof}\rangle$

**lemma**  $a0i\text{-anyP}\text{-observable}$  [iff]:  $\text{observable}\ (\text{obs}\ a0i)\ P$

$\langle\text{proof}\rangle$

### 1.11.3 Invariants

#### Injective agreement.

##### definition

$a0i\text{-inv1}\text{-iagree} :: 'ds\ a0i\text{-state}\ set$

##### where

$$\begin{aligned} a0i\text{-inv1}\text{-iagree} &\equiv \{s. \forall h\ d. \\ &\quad \text{set } h \subseteq \text{good} \longrightarrow d \notin \text{corrupted } s \longrightarrow \\ &\quad \text{signals } s\ (\text{Commit } h\ d) \leq \text{signals } s\ (\text{Running } h\ d) \\ &\} \end{aligned}$$

**lemmas**  $a0i\text{-inv1}\text{-iagree}I =$

$a0i\text{-inv1}\text{-iagree}\text{-def}\ [THEN\ \text{setc}\text{-def}\text{-to}\text{-intro},\ \text{rule}\text{-format}]$

**lemmas**  $a0i\text{-inv1}\text{-iagree}E\ [elim] =$

$a0i\text{-inv1}\text{-iagree}\text{-def}\ [THEN\ \text{setc}\text{-def}\text{-to}\text{-elim},\ \text{rule}\text{-format}]$

**lemmas**  $a0i\text{-inv1}\text{-iagree}D =$

$a0i\text{-inv1}\text{-iagree}\text{-def}\ [THEN\ \text{setc}\text{-def}\text{-to}\text{-dest},\ \text{rule}\text{-format},\ \text{rotated}\ 1]$

**lemma**  $PO\text{-}a0i\text{-inv1}\text{-iagree}\text{-init}\ [iff]:$

$\text{init } a0i \subseteq a0i\text{-inv1}\text{-iagree}$

$\langle\text{proof}\rangle$

**lemma**  $PO\text{-}a0i\text{-inv1}\text{-iagree}\text{-trans}\ [iff]:$

$\{a0i\text{-inv1}\text{-iagree}\}\ \text{trans } a0i\ \{>\ a0i\text{-inv1}\text{-iagree}\}$

$\langle\text{proof}\rangle$

**lemma**  $PO\text{-}a0i\text{-inv1}\text{-iagree}\ [iff]:\ \text{reach } a0i \subseteq a0i\text{-inv1}\text{-iagree}$

$\langle\text{proof}\rangle$

As an external invariant.

**lemma**  $PO\text{-}a0i\text{-obs}\text{-inv1}\text{-iagree}\ [iff]:\ \text{oreach } a0i \subseteq a0i\text{-inv1}\text{-iagree}$

$\langle\text{proof}\rangle$

### 1.11.4 Refinement

##### definition

$\text{med}0n0i :: 'd\ a0i\text{-obs} \Rightarrow 'd\ a0i\text{-obs}$

##### where

$\text{med}0n0i \equiv \text{id}$

##### definition

$R0n0i :: ('d\ a0n\text{-state} \times 'd\ a0i\text{-state})\ set$

##### where

$R0n0i \equiv \text{Id}$

**lemma**  $PO\text{-}a0i\text{-running}\text{-refines}\text{-}a0n\text{-running}:$

$\{R0n0i\}$

$(a0n\text{-running } h\ d), (a0i\text{-running } h\ d)$

$\{>\ R0n0i\}$

$\langle\text{proof}\rangle$

**lemma**  $PO\text{-}a0i\text{-commit}\text{-refines}\text{-}a0n\text{-commit}:$

$\{R0n0i\}$   
 $(a0n\text{-commit } h \ d), (a0i\text{-commit } h \ d)$   
 $\{> R0n0i\}$   
 $\langle\text{proof}\rangle$

**lemma** *PO-a0i-corrupt-refines-a0n-corrupt*:  
 $\{R0n0i\}$   
 $(a0n\text{-corrupt } d), (a0i\text{-corrupt } d)$   
 $\{> R0n0i\}$   
 $\langle\text{proof}\rangle$

**lemmas** *PO-a0i-trans-refines-a0n-trans =*  
*PO-a0i-running-refines-a0n-running*  
*PO-a0i-commit-refines-a0n-commit*  
*PO-a0i-corrupt-refines-a0n-corrupt*

All together now...

**lemma** *PO-m1-refines-init-a0n [iff]*:  
 $\text{init } a0i \subseteq R0n0i \text{“}(\text{init } a0n)$   
 $\langle\text{proof}\rangle$

**lemma** *PO-m1-refines-trans-a0n [iff]*:  
 $\{R0n0i\}$   
 $(\text{trans } a0n), (\text{trans } a0i)$   
 $\{> R0n0i\}$   
 $\langle\text{proof}\rangle$

**lemma** *PO-obs-consistent [iff]*:  
 $\text{obs-consistent } R0n0i \ \text{med}0n0i \ a0n \ a0i$   
 $\langle\text{proof}\rangle$

**lemma** *PO-a0i-refines-a0n*:  
 $\text{refines } R0n0i \ \text{med}0n0i \ a0n \ a0i$   
 $\langle\text{proof}\rangle$

### 1.11.5 Derived invariants

**lemma** *iagree-implies-niagree [iff]*:  $a0i\text{-inv1-iagree} \subseteq a0n\text{-inv1-niagree}$   
 $\langle\text{proof}\rangle$

Non-injective agreement as internal and external invariants.

**lemma** *PO-a0i-a0n-inv1-niagree [iff]*:  $\text{reach } a0i \subseteq a0n\text{-inv1-niagree}$   
 $\langle\text{proof}\rangle$

**lemma** *PO-a0i-obs-a0n-inv1-niagree [iff]*:  $\text{oreach } a0i \subseteq a0n\text{-inv1-niagree}$   
 $\langle\text{proof}\rangle$

**end**

## Chapter 2

# Unidirectional Authentication Protocols

In this chapter, we derive some simple unilateral authentication protocols. We have a single abstract model at Level 1. We then refine this model into two channel protocols (Level 2), one using authentic channels and one using confidential channels. We then refine these in turn into cryptographic protocols (Level 3) respectively using signatures and public-key encryption.

### 2.1 Refinement 1: Abstract Protocol

```
theory m1-auth imports ../Refinement/Runs ../Refinement/a0i-agree
begin
```

```
declare domIff [simp, iff del]
```

#### 2.1.1 State

We introduce protocol runs.

```
record m1-state =
  runs :: runs-t
```

```
type-synonym
  m1-obs = m1-state
```

```
definition
  m1-init :: m1-state set where
  m1-init  $\equiv$  { (
    runs = Map.empty
  ) }
```

#### 2.1.2 Events

```
definition — refines skip
  m1-step1 :: [rid-t, agent, agent, nonce]  $\Rightarrow$  (m1-state  $\times$  m1-state) set
where
```

$m1\text{-step1 } Ra \ A \ B \ Na \equiv \{(s, s1)\}.$

— guards  
 $Ra \notin \text{dom } (runs \ s) \wedge$  — new initiator run  
 $Na = Ra\$0 \wedge$  — generated nonce

— actions  
 $s1 = s[$   
 $runs := (runs \ s)($   
 $Ra \mapsto (Init, [A, B], [])$   
 $)$   
 $]$   
 $\}$

**definition** — refines *a0i-running*

$m1\text{-step2} :: [rid\text{-}t, agent, agent, nonce, nonce] \Rightarrow (m1\text{-state} \times m1\text{-state}) \text{ set}$

**where**

$m1\text{-step2 } Rb \ A \ B \ Na \ Nb \equiv \{(s, s1)\}.$  —  $Ni$  is completely arbitrary

— guards  
 $Rb \notin \text{dom } (runs \ s) \wedge$  — new responder run  
 $Nb = Rb\$0 \wedge$  — generated nonce

— actions  
 $s1 = s[$   
 $runs := (runs \ s)(Rb \mapsto (Resp, [A, B], [aNon \ Na]))$   
 $]$   
 $\}$

**definition** — refines *a0i-commit*

$m1\text{-step3} ::$

$[rid\text{-}t, agent, agent, nonce, nonce] \Rightarrow (m1\text{-state} \times m1\text{-state}) \text{ set}$

**where**

$m1\text{-step3 } Ra \ A \ B \ Na \ Nb \equiv \{(s, s1)\}.$

— guards  
 $runs \ s \ Ra = \text{Some } (Init, [A, B], []) \wedge$   
 $Na = Ra\$0 \wedge$

— authentication guard:  
 $(A \notin bad \wedge B \notin bad \longrightarrow (\exists Rb.$   
 $Nb = Rb\$0 \wedge runs \ s \ Rb = \text{Some } (Resp, [A, B], [aNon \ Na]))) \wedge$

— actions  
 $s1 = s[$   
 $runs := (runs \ s)(Ra \mapsto (Init, [A, B], [aNon \ Nb]))$   
 $]$   
 $\}$

Transition system.

**definition**

$m1\text{-trans} :: (m1\text{-state} \times m1\text{-state}) \text{ set}$  **where**

$m1\text{-trans} \equiv (\bigcup A \ B \ Ra \ Rb \ Na \ Nb.$

```

    m1-step1 Ra A B Na  ∪
    m1-step2 Rb A B Na Nb ∪
    m1-step3 Ra A B Na Nb ∪
    Id
  )

```

**definition**

```

m1 :: (m1-state, m1-obs) spec where
m1 ≡ ⟨
  init = m1-init,
  trans = m1-trans,
  obs = id
⟩

```

**lemmas** *m1-defs* =

```

m1-def m1-init-def m1-trans-def
m1-step1-def m1-step2-def m1-step3-def

```

### 2.1.3 Simulation relation

We define two auxiliary functions to reconstruct the signals of the initial model from completed initiator and responder runs of the current one.

**type-synonym**

```

irsig = nonce × nonce

```

**fun**

```

runs2sigs :: runs-t ⇒ irsig signal ⇒ nat

```

**where**

```

runs2sigs runz (Commit [A, B] (Ra$0, Nb)) =
  (if runz Ra = Some (Init, [A, B], [aNon Nb]) then 1 else 0)

```

```

| runs2sigs runz (Running [A, B] (Na, Rb$0)) =
  (if runz Rb = Some (Resp, [A, B], [aNon Na]) then 1 else 0)

```

```

| runs2sigs runz - = 0

```

Simulation relation and mediator function. We map completed initiator and responder runs to commit and running signals, respectively.

**definition**

```

med10 :: m1-obs ⇒ irsig a0i-obs where
med10 o1 ≡ ⟨ signals = runs2sigs (runs o1), corrupted = {} ⟩

```

**definition**

```

R01 :: (irsig a0i-state × m1-state) set where
R01 ≡ {(s, t). signals s = runs2sigs (runs t) ∧ corrupted s = {} }

```

**lemmas** *R01-defs* = *R01-def med10-def*

### Lemmas about the auxiliary functions

Basic lemmas

**lemma** *runs2sigs-empty* [*simp*]:  
 $runz = Map.empty \implies runs2sigs\ runz = (\lambda x. 0)$   
 ⟨*proof*⟩

Update lemmas

**lemma** *runs2sigs-upd-init-none* [*simp*]:  
 $\llbracket Ra \notin dom\ runz \rrbracket$   
 $\implies runs2sigs\ (runz(Ra \mapsto (Init, [A, B], []))) = runs2sigs\ runz$   
 ⟨*proof*⟩

**lemma** *runs2sigs-upd-init-some* [*simp*]:  
 $\llbracket runz\ Ra = Some\ (Init, [A, B], []) \rrbracket$   
 $\implies runs2sigs\ (runz(Ra \mapsto (Init, [A, B], [aNon\ Nb]))) =$   
 $(runs2sigs\ runz)(Commit\ [A, B]\ (Ra\$0, Nb)\ :=\ 1)$   
 ⟨*proof*⟩

**lemma** *runs2sigs-upd-resp* [*simp*]:  
 $\llbracket Rb \notin dom\ runz \rrbracket$   
 $\implies runs2sigs\ (runz(Rb \mapsto (Resp, [A, B], [aNon\ Na]))) =$   
 $(runs2sigs\ runz)(Running\ [A, B]\ (Na, Rb\$0)\ :=\ 1)$   
 ⟨*proof*⟩

## 2.1.4 Refinement

**lemma** *PO-m1-step1-refines-skip*:  
 $\{R01\}$   
 $Id, (m1-step1\ Ra\ A\ B\ Na)$   
 $\{>\ R01\}$   
 ⟨*proof*⟩

**lemma** *PO-m1-step2-refines-a0i-running*:  
 $\{R01\}$   
 $(a0i-running\ [A, B]\ (Na, Nb)), (m1-step2\ Rb\ A\ B\ Na\ Nb)$   
 $\{>\ R01\}$   
 ⟨*proof*⟩

**lemma** *PO-m1-step3-refines-a0i-commit*:  
 $\{R01\}$   
 $(a0i-commit\ [A, B]\ (Na, Nb)), (m1-step3\ Ra\ A\ B\ Na\ Nb)$   
 $\{>\ R01\}$   
 ⟨*proof*⟩

**lemmas** *PO-m1-trans-refines-a0i-trans* =  
 $PO-m1-step1-refines-skip\ PO-m1-step2-refines-a0i-running$   
 $PO-m1-step3-refines-a0i-commit$

All together now...

**lemma** *PO-m1-refines-init-a0i* [*iff*]:  
 $init\ m1 \subseteq R01 \iff (init\ a0i)$   
 ⟨*proof*⟩

**lemma** *PO-m1-refines-trans-a0i* [*iff*]:  
 $\{R01\}$

$(trans\ a0i), (trans\ m1)$   
 $\{> R01\}$   
 $\langle proof \rangle$

**lemma** *PO-obs-consistent* [iff]:  
*obs-consistent R01 med10 a0i m1*  
 $\langle proof \rangle$

**lemma** *PO-m1-refines-a0i*:  
*refines R01 med10 a0i m1*  
 $\langle proof \rangle$

end

## 2.2 Refinement 2a: Authentic Channel Protocol

**theory** *m2-auth-chan* **imports** *m1-auth* *../Refinement/Channels*  
**begin**

We refine the abstract authentication protocol to a version of the ISO/IEC 9798-3 protocol using abstract channels. In standard protocol notation, the original protocol is specified as follows.

- M1.  $A \rightarrow B$  :  $A, B, N_A$   
M2.  $B \rightarrow A$  :  $\{N_B, N_A, A\}_{K^{-1}(B)}$

We introduce insecure channels between pairs of agents for the first message and authentic channels for the second.

**declare** *domIff* [*simp, iff del*]

### 2.2.1 State

State: we extend the state with insecure and authentic channels defined above.

**record** *m2-state* = *m1-state* +  
*chan* :: *chmsg set*

Observations.

**type-synonym**  
*m2-obs* = *m1-state*

**definition**  
*m2-obs* :: *m2-state*  $\Rightarrow$  *m2-obs* **where**  
*m2-obs* *s*  $\equiv$  (  
*runs* = *runs s*  
 $)$

### 2.2.2 Events

**definition**  
*m2-step1* :: [*rid-t, agent, agent, nonce*]  $\Rightarrow$  (*m2-state*  $\times$  *m2-state*) *set*  
**where**

$m2\text{-step1 } Ra \ A \ B \ Na \equiv \{(s, s1)\}.$   
 — guards  
 $Ra \notin \text{dom } (\text{runs } s) \wedge$   
 $Na = Ra\$0 \wedge$   
 — actions  
 $s1 = s\langle$   
 $\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [])),$   
 $\text{chan} := \text{insert } (\text{Insec } A \ B \ (\text{Msg } [aNon \ Na])) \ (\text{chan } s)$   
 $\rangle$   
 $\}$

**definition**

$m2\text{-step2} :: [\text{rid-t}, \text{agent}, \text{agent}, \text{nonce}, \text{nonce}] \Rightarrow (m2\text{-state} \times m2\text{-state}) \text{ set}$

**where**

$m2\text{-step2 } Rb \ A \ B \ Na \ Nb \equiv \{(s, s1)\}.$

— guards

$Rb \notin \text{dom } (\text{runs } s) \wedge$

$Nb = Rb\$0 \wedge$

$\text{Insec } A \ B \ (\text{Msg } [aNon \ Na]) \in \text{chan } s \wedge$

— rcv M1

— actions

$s1 = s\langle$

$\text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [aNon \ Na])),$

$\text{chan} := \text{insert } (\text{Auth } B \ A \ (\text{Msg } [aNon \ Nb, aNon \ Na])) \ (\text{chan } s) \text{ — snd } M2$

$\rangle$

$\}$

**definition**

$m2\text{-step3} :: [\text{rid-t}, \text{agent}, \text{agent}, \text{nonce}, \text{nonce}] \Rightarrow (m2\text{-state} \times m2\text{-state}) \text{ set}$

**where**

$m2\text{-step3 } Ra \ A \ B \ Na \ Nb \equiv \{(s, s1)\}.$

— guards

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], []) \wedge$

$Na = Ra\$0 \wedge$

$\text{Auth } B \ A \ (\text{Msg } [aNon \ Nb, aNon \ Na]) \in \text{chan } s \wedge$

— rcv M2

— actions

$s1 = s\langle$

$\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [aNon \ Nb]))$

$\rangle$

$\}$

Intruder fake event.

**definition** — refines Id

$m2\text{-fake} :: (m2\text{-state} \times m2\text{-state}) \text{ set}$

**where**

$m2\text{-fake} \equiv \{(s, s1)\}.$

— actions:

$s1 = s\langle \text{chan} := \text{fake ik0 } (\text{dom } (\text{runs } s)) \ (\text{chan } s) \rangle$

}

Transition system.

**definition**

$m2-init :: m2-state\ set$

**where**

$m2-init \equiv \{ (\mid$   
 $runs = Map.empty,$   
 $chan = \{\}$   
 $\mid \}$

**definition**

$m2-trans :: (m2-state \times m2-state)\ set$  **where**

$m2-trans \equiv (\bigcup A\ B\ Ra\ Rb\ Na\ Nb.$   
 $(m2-step1\ Ra\ A\ B\ Na) \cup$   
 $(m2-step2\ Rb\ A\ B\ Na\ Nb) \cup$   
 $(m2-step3\ Ra\ A\ B\ Na\ Nb) \cup$   
 $m2-fake \cup$   
 $Id$   
 $)$

**definition**

$m2 :: (m2-state, m2-obs)\ spec$  **where**

$m2 \equiv (\mid$   
 $init = m2-init,$   
 $trans = m2-trans,$   
 $obs = m2-obs$   
 $\mid)$

**lemmas**  $m2-defs =$

$m2-def\ m2-init-def\ m2-trans-def\ m2-obs-def$   
 $m2-step1-def\ m2-step2-def\ m2-step3-def\ m2-fake-def$

## 2.2.3 Invariants

### Authentic channel and responder

This property relates the messages in the authentic channel to the responder run frame.

**definition**

$m2-inv1-auth :: m2-state\ set$  **where**

$m2-inv1-auth \equiv \{s. \forall A\ B\ Na\ Nb.$   
 $Auth\ B\ A\ (Msg\ [aNon\ Nb, aNon\ Na]) \in chan\ s \longrightarrow B \notin bad \longrightarrow A \notin bad \longrightarrow$   
 $(\exists Rb. runs\ s\ Rb = Some\ (Resp, [A, B], [aNon\ Na]) \wedge Nb = Rb\$0)$   
 $\}$

**lemmas**  $m2-inv1-authI =$

$m2-inv1-auth-def\ [THEN\ setc-def-to-intro, rule-format]$

**lemmas**  $m2-inv1-authE [elim] =$

$m2-inv1-auth-def\ [THEN\ setc-def-to-elim, rule-format]$

**lemmas**  $m2-inv1-authD [dest] =$

$m2-inv1-auth-def\ [THEN\ setc-def-to-dest, rule-format, rotated\ 1]$

Invariance proof.

**lemma** *PO-m2-inv2-init* [iff]:

$init\ m2 \subseteq m2\text{-inv1-auth}$

*<proof>*

**lemma** *PO-m2-inv2-trans* [iff]:

$\{m2\text{-inv1-auth}\}\ trans\ m2\ \{>\ m2\text{-inv1-auth}\}$

*<proof>*

**lemma** *PO-m2-inv2* [iff]:  $reach\ m2 \subseteq m2\text{-inv1-auth}$

*<proof>*

## 2.2.4 Refinement

Simulation relation and mediator function. This is a pure superposition refinement.

**definition**

$R12 :: (m1\text{-state} \times m2\text{-state})\ set\ \mathbf{where}$

$R12 \equiv \{(s, t). runs\ s = runs\ t\}$  — That's it!

**definition**

$med21 :: m2\text{-obs} \Rightarrow m1\text{-obs}\ \mathbf{where}$

$med21 \equiv id$

Refinement proof

**lemma** *PO-m2-step1-refines-m1-step1*:

$\{R12\}$

$(m1\text{-step1}\ Ra\ A\ B\ Na), (m2\text{-step1}\ Ra\ A\ B\ Na)$

$\{>\ R12\}$

*<proof>*

**lemma** *PO-m2-step2-refines-m1-step2*:

$\{R12\}$

$(m1\text{-step2}\ Ra\ A\ B\ Na\ Nb), (m2\text{-step2}\ Ra\ A\ B\ Na\ Nb)$

$\{>\ R12\}$

*<proof>*

**lemma** *PO-m2-step3-refines-m1-step3*:

$\{R12 \cap UNIV \times m2\text{-inv1-auth}\}$

$(m1\text{-step3}\ Ra\ A\ B\ Na\ Nb), (m2\text{-step3}\ Ra\ A\ B\ Na\ Nb)$

$\{>\ R12\}$

*<proof>*

New fake event refines skip.

**lemma** *PO-m2-fake-refines-m1-skip*:

$\{R12\}\ Id, m2\text{-fake}\ \{>\ R12\}$

*<proof>*

**lemmas** *PO-m2-trans-refines-m1-trans* =

*PO-m2-step1-refines-m1-step1 PO-m2-step2-refines-m1-step2*

*PO-m2-step3-refines-m1-step3 PO-m2-fake-refines-m1-skip*

All together now...

**lemma** *PO-m2-refines-init-m1* [iff]:

$init\ m2 \subseteq R12 \llbracket (init\ m1) \rrbracket$   
 $\langle proof \rangle$

**lemma** *PO-m2-refines-trans-m1* [iff]:  
 $\{R12 \cap UNIV \times m2\text{-inv1-auth}\}$   
 $(trans\ m1), (trans\ m2)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-obs-consistent* [iff]:  
 $obs\text{-consistent}\ R12\ med21\ m1\ m2$   
 $\langle proof \rangle$

**lemma** *m2-refines-m1*:  
 $refines$   
 $(R12 \cap UNIV \times m2\text{-inv1-auth})$   
 $med21\ m1\ m2$   
 $\langle proof \rangle$

**end**

## 2.3 Refinement 2b: Confidential Channel Protocol

**theory** *m2-confid-chan* **imports** *m1-auth* *../Refinement/Channels*  
**begin**

We refine the abstract authentication protocol to the first two steps of the Needham-Schroeder-Lowe protocol, which we call NSL/2. In standard protocol notation, the original protocol is specified as follows.

$$\begin{aligned} \text{M1. } A \rightarrow B & : \{N_A, A\}_{K(B)} \\ \text{M2. } B \rightarrow A & : \{N_A, N_B, B\}_{K(A)} \end{aligned}$$

At this refinement level, we abstract the encrypted messages to non-cryptographic messages transmitted on confidential channels.

**declare** *domIff* [*simp, iff del*]

### 2.3.1 State and observations

**record** *m2-state* = *m1-state* +  
 $chan :: chmsg\ set$  — channels

**type-synonym**  
 $m2\text{-obs} = m1\text{-state}$

**definition**  
 $m2\text{-obs} :: m2\text{-state} \Rightarrow m2\text{-obs}$  **where**  
 $m2\text{-obs}\ s \equiv ()$   
 $runs = runs\ s$   
 $\})$



— guards  
 $runs\ s\ Ra = Some\ (Init,\ [A,\ B],\ []) \wedge$   
 $Na = Ra\$0 \wedge$

$Confid\ B\ A\ (Msg\ [aNon\ Na,\ aNon\ Nb]) \in chan\ s \wedge$  — receive M2

— actions  
 $s1 = s(|$   
 $runs := (runs\ s)(Ra \mapsto (Init,\ [A,\ B],\ [aNon\ Nb]))$   
 $|)$   
 $}$

Intruder fake event.

**definition** — refines  $Id$

$m2-fake :: (m2-state \times m2-state)\ set$

**where**

$m2-fake \equiv \{(s,\ s1).\$

— actions:

$s1 = s(|\ chan := fake\ ik0\ (dom\ (runs\ s))\ (chan\ s)\ |)$

$\}$

Transition system.

**definition**

$m2-trans :: (m2-state \times m2-state)\ set$  **where**

$m2-trans \equiv (\bigcup\ A\ B\ Ra\ Rb\ Na\ Nb.$

$m2-step1\ Ra\ A\ B\ Na \cup$

$m2-step2\ Rb\ A\ B\ Na\ Nb \cup$

$m2-step3\ Ra\ A\ B\ Na\ Nb \cup$

$m2-fake \cup$

$Id$

$)$

**definition**

$m2 :: (m2-state,\ m2-obs)\ spec$  **where**

$m2 \equiv (|$

$init = m2-init,$

$trans = m2-trans,$

$obs = m2-obs$

$|)$

**lemmas**  $m2-defs =$

$m2-def\ m2-init-def\ m2-trans-def\ m2-obs-def$

$m2-step1-def\ m2-step2-def\ m2-step3-def\ m2-fake-def$

### 2.3.3 Invariants

**Invariant 1: Messages only contains generated nonces.**

**definition**

$m2-inv1-nonces :: m2-state\ set$  **where**

$m2-inv1-nonces \equiv \{s.\ \forall R.$

$aNon\ (R\$0) \in atoms\ (chan\ s) \longrightarrow R \in dom\ (runs\ s)$

$\}$

**lemmas**  $m2\text{-inv1}\text{-nonces}I =$   
 $m2\text{-inv1}\text{-nonces}\text{-def}$  [*THEN setc-def-to-intro, rule-format*]  
**lemmas**  $m2\text{-inv1}\text{-nonces}E$  [*elim*] =  
 $m2\text{-inv1}\text{-nonces}\text{-def}$  [*THEN setc-def-to-elim, rule-format*]  
**lemmas**  $m2\text{-inv1}\text{-nonces}D =$   
 $m2\text{-inv1}\text{-nonces}\text{-def}$  [*THEN setc-def-to-dest, rule-format, rotated 1*]

**lemma**  $PO\text{-}m2\text{-inv1}\text{-init}$  [*iff*]:  $\text{init } m2 \subseteq m2\text{-inv1}\text{-nonces}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m2\text{-inv1}\text{-trans}$  [*iff*]:  
 $\{m2\text{-inv1}\text{-nonces}\}$   $\text{trans } m2 \{> m2\text{-inv1}\text{-nonces}\}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m2\text{-inv012}$  [*iff*]:  
 $\text{reach } m2 \subseteq m2\text{-inv1}\text{-nonces}$   
 $\langle\text{proof}\rangle$

### Invariant 3: relates message 2 with the responder run

It is needed, together with initiator nonce secrecy, in proof obligation REF/*m2-step2*.

#### definition

$m2\text{-inv3}\text{-msg2} :: m2\text{-state}$  set **where**  
 $m2\text{-inv3}\text{-msg2} \equiv \{s. \forall A B Na Nb.$   
 $\text{Confid } B A (\text{Msg } [a\text{Non } Na, a\text{Non } Nb]) \in \text{chan } s \longrightarrow$   
 $a\text{Non } Na \notin \text{extr ik0 } (\text{chan } s) \longrightarrow$   
 $(\exists Rb. Nb = Rb\$0 \wedge \text{runs } s Rb = \text{Some } (\text{Resp}, [A, B], [a\text{Non } Na]))$   
 $\}$

**lemmas**  $m2\text{-inv3}\text{-msg2}I = m2\text{-inv3}\text{-msg2}\text{-def}$  [*THEN setc-def-to-intro, rule-format*]  
**lemmas**  $m2\text{-inv3}\text{-msg2}E$  [*elim*] =  $m2\text{-inv3}\text{-msg2}\text{-def}$  [*THEN setc-def-to-elim, rule-format*]  
**lemmas**  $m2\text{-inv3}\text{-msg2}D = m2\text{-inv3}\text{-msg2}\text{-def}$  [*THEN setc-def-to-dest, rule-format, rotated 1*]

**lemma**  $PO\text{-}m2\text{-inv4}\text{-init}$  [*iff*]:  
 $\text{init } m2 \subseteq m2\text{-inv3}\text{-msg2}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m2\text{-inv4}\text{-trans}$  [*iff*]:  
 $\{m2\text{-inv3}\text{-msg2}\}$   $\text{trans } m2 \{> m2\text{-inv3}\text{-msg2}\}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m2\text{-inv4}$  [*iff*]:  $\text{reach } m2 \subseteq m2\text{-inv3}\text{-msg2}$   
 $\langle\text{proof}\rangle$

### Invariant 4: Initiator nonce secrecy.

It is needed in the proof obligation REF/*m2-step2*. It would be sufficient to prove the invariant for the case  $x = \text{None}$ , but we have generalized it here.

#### definition

$m2\text{-inv4-inon-secret} :: m2\text{-state set where}$   
 $m2\text{-inv4-inon-secret} \equiv \{s. \forall A B Ra al.$   
 $runs\ s\ Ra = Some\ (Init, [A, B], al) \longrightarrow$   
 $A \notin bad \longrightarrow B \notin bad \longrightarrow$   
 $aNon\ (Ra\$0) \notin extr\ ik0\ (chan\ s)$   
 $\}$

**lemmas**  $m2\text{-inv4-inon-secret}I =$   
 $m2\text{-inv4-inon-secret-def}\ [THEN\ setc\text{-def-to-intro},\ rule\text{-format}]$   
**lemmas**  $m2\text{-inv4-inon-secret}E\ [elim] =$   
 $m2\text{-inv4-inon-secret-def}\ [THEN\ setc\text{-def-to-elim},\ rule\text{-format}]$   
**lemmas**  $m2\text{-inv4-inon-secret}D =$   
 $m2\text{-inv4-inon-secret-def}\ [THEN\ setc\text{-def-to-dest},\ rule\text{-format},\ rotated\ 1]$

**lemma**  $PO\text{-}m2\text{-inv3-init}\ [iff]:$   
 $init\ m2 \subseteq m2\text{-inv4-inon-secret}$   
 $\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv3-trans}\ [iff]:$   
 $\{m2\text{-inv4-inon-secret} \cap m2\text{-inv1-nonces}\}$   
 $trans\ m2$   
 $\{>\ m2\text{-inv4-inon-secret}\}$   
 $\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv3}\ [iff]:\ reach\ m2 \subseteq m2\text{-inv4-inon-secret}$   
 $\langle proof \rangle$

### 2.3.4 Refinement

#### definition

$R12 :: (m1\text{-state} \times m2\text{-state})\ set\ where$   
 $R12 \equiv \{(s, t). runs\ s = runs\ t\}$

#### abbreviation

$med21 :: m2\text{-obs} \Rightarrow m1\text{-obs}\ where$   
 $med21 \equiv id$

Proof obligations.

**lemma**  $PO\text{-}m2\text{-step1-refines-m1-step1}:$   
 $\{R12\}$   
 $(m1\text{-step1}\ Ra\ A\ B\ Na), (m2\text{-step1}\ Ra\ A\ B\ Na)$   
 $\{>\ R12\}$   
 $\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-step2-refines-m1-step2}:$   
 $\{R12\}$   
 $(m1\text{-step2}\ Rb\ A\ B\ Na\ Nb), (m2\text{-step2}\ Rb\ A\ B\ Na\ Nb)$   
 $\{>\ R12\}$   
 $\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-step3-refines-m1-step3}:$   
 $\{R12 \cap UNIV \times (m2\text{-inv4-inon-secret} \cap m2\text{-inv3-msg2})\}$

$(m1\text{-step3 } Ra \ A \ B \ Na \ Nb), (m2\text{-step3 } Ra \ A \ B \ Na \ Nb)$   
 $\{> R12\}$   
 $\langle proof \rangle$

New fake events refine skip.

**lemma** *PO-m2-fake-refines-skip*:  
 $\{R12\} \text{ Id}, m2\text{-fake } \{> R12\}$   
 $\langle proof \rangle$

**lemmas** *PO-m2-trans-refines-m1-trans =*  
*PO-m2-step1-refines-m1-step1 PO-m2-step2-refines-m1-step2*  
*PO-m2-step3-refines-m1-step3 PO-m2-fake-refines-skip*

All together now...

**lemma** *PO-m2-refines-init-m1 [iff]*:  
 $init \ m2 \subseteq R12 \text{“}(init \ m1)$   
 $\langle proof \rangle$

**lemma** *PO-m2-refines-trans-m1 [iff]*:  
 $\{R12 \cap$   
 $UNIV \times (m2\text{-inv4-inon-secret} \cap m2\text{-inv3-msg2})\}$   
 $(trans \ m1), (trans \ m2)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-R12-obs-consistent [iff]*:  
 $obs\text{-consistent } R12 \ med21 \ m1 \ m2$   
 $\langle proof \rangle$

**lemma** *PO-m3-refines-m2*:  
*refines*  
 $(R12 \cap$   
 $UNIV \times (m2\text{-inv4-inon-secret} \cap m2\text{-inv3-msg2} \cap m2\text{-inv1-nonces}))$   
 $med21 \ m1 \ m2$   
 $\langle proof \rangle$

**end**

## 2.4 Refinement 3a: Signature-based Dolev-Yao Protocol (Variant A)

**theory** *m3-sig imports m2-auth-chan ../Refinement/Message*  
**begin**

We implement the channel protocol of the previous refinement with signatures and add a full-fledged Dolev-Yao adversary. In this variant, the adversary is realized using Paulson’s closure operators for message derivation (as opposed to a collection of one-step derivation events a la Strand spaces).

Proof tool configuration. Avoid annoying automatic unfolding of *dom* (again).

**declare** *domIff* [*simp, iff del*]  
**declare** *analz-into-parts* [*dest*]

### 2.4.1 State

We extend the state of *m1* with insecure and authentic channels between each pair of agents.

**record** *m3-state* = *m1-state* +  
*IK* :: *msg set* — intruder knowledge

**type-synonym**

*m3-obs* = *m1-state*

**definition**

*m3-obs* :: *m3-state*  $\Rightarrow$  *m3-obs* **where**  
*m3-obs* *s*  $\equiv$  (  
*runs* = *runs s*  
 $\mid$ )

### 2.4.2 Events

**definition**

*m3-step1* :: [*rid-t, agent, agent, nonce*]  $\Rightarrow$  (*m3-state*  $\times$  *m3-state*) *set*

**where**

*m3-step1* *Ra A B Na*  $\equiv$   $\{(s, s1)\}$ .

— guards

*Ra*  $\notin$  *dom (runs s)*  $\wedge$   
*Na* = *Ra*\$0  $\wedge$

— actions

*s1* = *s*(  
*runs* := (*runs s*)(*Ra*  $\mapsto$  (*Init*, [*A*, *B*], [])),  
*IK* := *insert*  $\{\{Agent\ A, Agent\ B, Nonce\ Na\}\}$  (*IK s*) — send msg 1  
 $\mid$ )  
 $\}$

**definition**

*m3-step2* :: [*rid-t, agent, agent, nonce, nonce*]  $\Rightarrow$  (*m3-state*  $\times$  *m3-state*) *set*

**where**

*m3-step2* *Rb A B Na Nb*  $\equiv$   $\{(s, s1)\}$ .

— guards

*Rb*  $\notin$  *dom (runs s)*  $\wedge$   
*Nb* = *Rb*\$0  $\wedge$

$\{\{Agent\ A, Agent\ B, Nonce\ Na\}\} \in IK\ s$   $\wedge$  — receive msg 1

— actions

*s1* = *s*(  
*runs* := (*runs s*)(*Rb*  $\mapsto$  (*Resp*, [*A*, *B*], [*aNon* *Na*])),  
— send msg 2  
*IK* := *insert* (*Crypt* (*priK* *B*)  $\{\{Nonce\ Nb, Nonce\ Na, Agent\ A\}\}$ ) (*IK s*)  
 $\mid$ )

$\Downarrow$   
 $\}$

**definition**

$m3\text{-step3} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{nonce}] \Rightarrow (m3\text{-state} \times m3\text{-state}) \text{ set}$

**where**

$m3\text{-step3 } Ra \ A \ B \ Na \ Nb \equiv \{(s, s1)\}.$

— guards

$runs \ s \ Ra = \text{Some } (Init, [A, B], []) \wedge$

$Na = Ra\$0 \wedge$

$Crypt \ (priK \ B) \ \{\{Nonce \ Nb, Nonce \ Na, Agent \ A\} \in IK \ s \wedge \text{— recv msg 2}$

— actions

$s1 = s\{$

$runs := (runs \ s)(Ra \mapsto (Init, [A, B], [aNon \ Nb]))$

$\Downarrow$

$\}$

The intruder messages are now generated by a full-fledged Dolev-Yao intruder.

**definition**

$m3\text{-DY-fake} :: (m3\text{-state} \times m3\text{-state}) \text{ set}$

**where**

$m3\text{-DY-fake} \equiv \{(s, s1)\}.$

— actions:

$s1 = s\{$

$IK := synth \ (analz \ (IK \ s))$

$\Downarrow$

$\}$

Transition system.

**definition**

$m3\text{-init} :: m3\text{-state} \text{ set}$

**where**

$m3\text{-init} \equiv \{ \{$

$runs = Map.empty,$

$IK = (Key'priK'bad) \cup (Key'range \ pubK) \cup (Key'shrK'bad)$

$\Downarrow \}$

**definition**

$m3\text{-trans} :: (m3\text{-state} \times m3\text{-state}) \text{ set}$  **where**

$m3\text{-trans} \equiv (\bigcup \ A \ B \ Ra \ Rb \ Na \ Nb.$

$m3\text{-step1 } Ra \ A \ B \ Na \ \cup$

$m3\text{-step2 } Rb \ A \ B \ Na \ Nb \ \cup$

$m3\text{-step3 } Ra \ A \ B \ Na \ Nb \ \cup$

$m3\text{-DY-fake} \ \cup$

$Id$

$\})$

**definition**

$m3 :: (m3\text{-state}, m3\text{-obs}) \text{ spec where}$   
 $m3 \equiv \langle$   
 $\text{init} = m3\text{-init},$   
 $\text{trans} = m3\text{-trans},$   
 $\text{obs} = m3\text{-obs}$   
 $\rangle$

**lemmas**  $m3\text{-defs} =$   
 $m3\text{-def } m3\text{-init-def } m3\text{-trans-def } m3\text{-obs-def}$   
 $m3\text{-step1-def } m3\text{-step2-def } m3\text{-step3-def}$   
 $m3\text{-DY-fake-def}$

### 2.4.3 Invariants

Specialize injectiveness of parts to enable aggressive application.

**lemmas**  $parts\text{-Inj}\text{-IK} = parts.Inj \text{ [where } H=IK \text{ s for s]}$   
**lemmas**  $analz\text{-Inj}\text{-IK} = analz.Inj \text{ [where } H=IK \text{ s for s]}$

The following invariants do not depend on the protocol messages. We want to keep this compilation refinement from channel protocols to full-fledged Dolev-Yao protocols as generic as possible.

#### inv1: Long-term key secrecy

Private signing keys are secret, that is, the intruder only knows private keys of corrupted agents.

The invariant uses the weaker *parts* operator instead of the perhaps more intuitive *analz* in its premise. This strengthens the invariant and potentially simplifies its proof.

#### definition

$m3\text{-inv1-lkeysec} :: m3\text{-state set where}$   
 $m3\text{-inv1-lkeysec} \equiv \{s. \forall A.$   
 $\text{Key } (priK A) \in analz (IK s) \longrightarrow A \in bad$   
 $\}$

**lemmas**  $m3\text{-inv1-lkeysecI} =$   
 $m3\text{-inv1-lkeysec-def [THEN setc-def-to-intro, rule-format]}$   
**lemmas**  $m3\text{-inv1-lkeysecE [elim] =$   
 $m3\text{-inv1-lkeysec-def [THEN setc-def-to-elim, rule-format]}$   
**lemmas**  $m3\text{-inv1-lkeysecD} =$   
 $m3\text{-inv1-lkeysec-def [THEN setc-def-to-dest, rule-format, rotated 1]}$

**lemma**  $PO\text{-}m3\text{-inv1-lkeysec-init [iff]:$   
 $\text{init } m3 \subseteq m3\text{-inv1-lkeysec}$   
 $\langle proof \rangle$

**lemma**  $PO\text{-}m3\text{-inv1-lkeysec-trans [iff]:$   
 $\{m3\text{-inv1-lkeysec}\ trans } m3 \{> m3\text{-inv1-lkeysec}\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-inv1-lkeysec* [iff]:  $reach\ m3 \subseteq m3\text{-inv1-lkeysec}$   
 ⟨proof⟩

### inv2: Intruder knows long-term keys of bad guys

**definition**

$m3\text{-inv2-badkeys} :: m3\text{-state set}$

**where**

$m3\text{-inv2-badkeys} \equiv \{s. \forall C.$   
 $C \in bad \longrightarrow Key\ (priK\ C) \in analz\ (IK\ s)$   
 $\}$

**lemmas**  $m3\text{-inv2-badkeysI} =$

$m3\text{-inv2-badkeys-def}\ [THEN\ setc\text{-def-to-intro},\ rule\text{-format}]$

**lemmas**  $m3\text{-inv2-badkeysE}\ [elim] =$

$m3\text{-inv2-badkeys-def}\ [THEN\ setc\text{-def-to-elim},\ rule\text{-format}]$

**lemmas**  $m3\text{-inv2-badkeysD}\ [dest] =$

$m3\text{-inv2-badkeys-def}\ [THEN\ setc\text{-def-to-dest},\ rule\text{-format},\ rotated\ 1]$

Invariance proof.

**lemma** *PO-m3-inv2-badkeys-init* [iff]:

$init\ m3 \subseteq m3\text{-inv2-badkeys}$

⟨proof⟩

**lemma** *PO-m3-inv2-badkeys-trans* [iff]:

$\{m3\text{-inv2-badkeys}\}\ trans\ m3\ \{>\ m3\text{-inv2-badkeys}\}$

⟨proof⟩

**lemma** *PO-m3-inv2-badkeys* [iff]:  $reach\ m3 \subseteq m3\text{-inv2-badkeys}$

⟨proof⟩

### inv3: Intruder knows all public keys (NOT USED)

This invariant is only needed with equality in *R23-msgs*.

**definition**

$m3\text{-inv3-pubkeys} :: m3\text{-state set}$

**where**

$m3\text{-inv3-pubkeys} \equiv \{s. \forall C.$   
 $Key\ (pubK\ C) \in analz\ (IK\ s)$   
 $\}$

**lemmas**  $m3\text{-inv3-pubkeysI} =$

$m3\text{-inv3-pubkeys-def}\ [THEN\ setc\text{-def-to-intro},\ rule\text{-format}]$

**lemmas**  $m3\text{-inv3-pubkeysE}\ [elim] =$

$m3\text{-inv3-pubkeys-def}\ [THEN\ setc\text{-def-to-elim},\ rule\text{-format}]$

**lemmas**  $m3\text{-inv3-pubkeysD}\ [dest] =$

$m3\text{-inv3-pubkeys-def}\ [THEN\ setc\text{-def-to-dest},\ rule\text{-format},\ rotated\ 1]$

Invariance proof.

**lemma** *PO-m3-inv3-pubkeys-init* [iff]:

$init\ m3 \subseteq m3\text{-inv3-pubkeys}$

⟨proof⟩

**lemma** *PO-m3-inv3-pubkeys-trans* [iff]:  
 $\{m3\text{-inv3-pubkeys}\} \text{ trans } m3 \{> m3\text{-inv3-pubkeys}\}$   
 ⟨proof⟩

**lemma** *PO-m3-inv3-pubkeys* [iff]: *reach*  $m3 \subseteq m3\text{-inv3-pubkeys}$   
 ⟨proof⟩

#### 2.4.4 Refinement

Automatic tool tuning. Tame too-aggressive pair decomposition, which is declared as a safe elim rule ([elim!]).

**lemmas** *MPair-parts* [rule del, elim]

**lemmas** *MPair-analz* [rule del, elim]

#### Simulation relation

##### abbreviation

$nonces :: msg \text{ set} \Rightarrow nonce \text{ set}$

##### where

$nonces \ H \equiv \{N. \ Nonce \ N \in \text{analz } H\}$

##### abbreviation

$ink :: chmsg \text{ set} \Rightarrow nonce \text{ set}$

##### where

$ink \ H \equiv \{N. \ aNon \ N \in \text{extr } ik0 \ H\}$

Abstraction function on sets of messages.

##### inductive-set

$abs\text{-msg} :: msg \text{ set} \Rightarrow chmsg \text{ set}$

**for**  $H :: msg \text{ set}$

##### where

*am-M1*:

$\{\{Agent \ A, \ Agent \ B, \ Nonce \ Na\} \in H$

$\implies Insec \ A \ B \ (Msg \ [aNon \ Na]) \in abs\text{-msg } H$

| *am-M2*:

$Crypt \ (priK \ B) \ \{\{Nonce \ Nb, \ Nonce \ Na, \ Agent \ A\} \in H$

$\implies Auth \ B \ A \ (Msg \ [aNon \ Nb, \ aNon \ Na]) \in abs\text{-msg } H$

The simulation relation is canonical. It states that the protocol messages in the intruder knowledge refine the abstract messages appearing in the channels *Insec* and *Auth*.

##### definition

$R23\text{-msgs} :: (m2\text{-state} \times m3\text{-state}) \text{ set}$  **where**

$R23\text{-msgs} \equiv \{(s, t). \ abs\text{-msg} \ (parts \ (IK \ t)) \subseteq \text{chan } s\}$  — with *parts!*

##### definition

$R23\text{-ink} :: (m2\text{-state} \times m3\text{-state}) \text{ set}$  **where**

$R23\text{-ink} \equiv \{(s, t). \ nonces \ (IK \ t) \subseteq \text{ink} \ (\text{chan } s)\}$

##### definition

$R23\text{-preserved} :: (m2\text{-state} \times m3\text{-state}) \text{ set}$  **where**

$R23\text{-preserved} \equiv \{(s, t). \text{runs } s = \text{runs } t\}$

**definition**

$R23 :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23 \equiv R23\text{-msgs} \cap R23\text{-ink} \cap R23\text{-preserved}$

**lemmas**  $R23\text{-defs} = R23\text{-def } R23\text{-msgs-def } R23\text{-ink-def } R23\text{-preserved-def}$

Mediator function: nothing new.

**definition**

$med32 :: m3\text{-obs} \Rightarrow m2\text{-obs where}$   
 $med32 \equiv id$

**lemmas**  $R23\text{-msgsI} =$

$R23\text{-msgs-def [THEN rel-def-to-intro, simplified, rule-format]}$

**lemmas**  $R23\text{-msgsE [elim] =$

$R23\text{-msgs-def [THEN rel-def-to-elim, simplified, rule-format]}$

**lemmas**  $R23\text{-msgsE' [elim] =$

$R23\text{-msgs-def [THEN rel-def-to-dest, simplified, rule-format, THEN subsetD]}$

**lemmas**  $R23\text{-inkI} =$

$R23\text{-ink-def [THEN rel-def-to-intro, simplified, rule-format]}$

**lemmas**  $R23\text{-inkE [elim] =$

$R23\text{-ink-def [THEN rel-def-to-elim, simplified, rule-format]}$

**lemmas**  $R23\text{-preservedI} =$

$R23\text{-preserved-def [THEN rel-def-to-intro, simplified, rule-format]}$

**lemmas**  $R23\text{-preservedE [elim] =$

$R23\text{-preserved-def [THEN rel-def-to-elim, simplified, rule-format]}$

**lemmas**  $R23\text{-intros} = R23\text{-msgsI } R23\text{-inkI } R23\text{-preservedI}$

**Facts about the abstraction function**

**declare**  $abs\text{-msg.intros [intro!]$

**declare**  $abs\text{-msg.cases [elim!]$

**lemma**  $abs\text{-msg-empty}: abs\text{-msg } \{\} = \{\}$

$\langle proof \rangle$

**lemma**  $abs\text{-msg-Un [simp]:$

$abs\text{-msg } (G \cup H) = abs\text{-msg } G \cup abs\text{-msg } H$

$\langle proof \rangle$

**lemma**  $abs\text{-msg-mono [elim]:$

$\llbracket m \in abs\text{-msg } G; G \subseteq H \rrbracket \Longrightarrow m \in abs\text{-msg } H$

$\langle proof \rangle$

**lemma**  $abs\text{-msg-insert-mono [intro]:$

$\llbracket m \in abs\text{-msg } H \rrbracket \Longrightarrow m \in abs\text{-msg } (\text{insert } m' H)$

$\langle proof \rangle$

Abstraction of concretely fakeable message yields abstractly fakeable messages. This is the key lemma for the refinement of the intruder.

**lemma** *abs-msg-DY-subset-fakeable*:

$$\begin{aligned} & \llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-ink}; t \in m3\text{-inv1-lkeysec} \rrbracket \\ & \implies \text{abs-msg} (\text{synth} (\text{analz} (IK\ t))) \subseteq \text{fake ik0} (\text{dom} (\text{runs}\ s)) (\text{chan}\ s) \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *absmsg-parts-subset-fakeable*:

$$\begin{aligned} & \llbracket (s, t) \in R23\text{-msgs} \rrbracket \\ & \implies \text{abs-msg} (\text{parts} (IK\ t)) \subseteq \text{fake ik0} (-\text{dom} (\text{runs}\ s)) (\text{chan}\ s) \\ & \langle \text{proof} \rangle \end{aligned}$$

**declare** *abs-msg-DY-subset-fakeable* [*simp*, *intro!*]

**declare** *absmsg-parts-subset-fakeable* [*simp*, *intro!*]

## Refinement proof

**lemma** *PO-m3-step1-refines-m2-step1*:

$$\begin{aligned} & \{R23\} \\ & (m2\text{-step1}\ Ra\ A\ B\ Na), (m3\text{-step1}\ Ra\ A\ B\ Na) \\ & \{>\ R23\} \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *PO-m3-step2-refines-m2-step2*:

$$\begin{aligned} & \{R23 \cap UNIV \times (m3\text{-inv1-lkeysec} \cap m3\text{-inv2-badkeys})\} \\ & (m2\text{-step2}\ Rb\ A\ B\ Na\ Nb), (m3\text{-step2}\ Rb\ A\ B\ Na\ Nb) \\ & \{>\ R23\} \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *PO-m3-step3-refines-m2-step3*:

$$\begin{aligned} & \{R23\} \\ & (m2\text{-step3}\ Ra\ A\ B\ Na\ Nb), (m3\text{-step3}\ Ra\ A\ B\ Na\ Nb) \\ & \{>\ R23\} \\ & \langle \text{proof} \rangle \end{aligned}$$

The Dolev-Yao fake event refines the abstract fake event.

**lemma** *PO-m3-DY-fake-refines-m2-fake*:

$$\begin{aligned} & \{R23 \cap UNIV \times (m3\text{-inv1-lkeysec} \cap m3\text{-inv2-badkeys})\} \\ & \quad m2\text{-fake}, m3\text{-DY-fake} \\ & \{>\ R23\} \\ & \langle \text{proof} \rangle \end{aligned}$$

All together now...

**lemmas** *PO-m3-trans-refines-m2-trans* =

$$\begin{aligned} & PO\text{-m3-step1-refines-m2-step1}\ PO\text{-m3-step2-refines-m2-step2} \\ & PO\text{-m3-step3-refines-m2-step3}\ PO\text{-m3-DY-fake-refines-m2-fake} \end{aligned}$$

**lemma** *PO-m3-refines-init-m2* [*iff*]:

$$\begin{aligned} & \text{init}\ m3 \subseteq R23 \text{“(init}\ m2) \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma** *PO-m3-refines-trans-m2* [*iff*]:

```

    {R23 ∩ UNIV × (m3-inv2-badkeys ∩ m3-inv1-lkeysec)}
      (trans m2), (trans m3)
    {> R23}
  ⟨proof⟩

```

**lemma** *PO-obs-consistent* [iff]:  
*obs-consistent R23 med32 m2 m3*  
 ⟨proof⟩

**lemma** *PO-m3-refines-m2*:  
*refines*  
 (*R23 ∩ UNIV × (m3-inv2-badkeys ∩ m3-inv1-lkeysec)*)  
*med32 m2 m3*  
 ⟨proof⟩

end

## 2.5 Refinement 3b: Encryption-based Dolev-Yao Protocol (Variant A)

**theory** *m3-enc* **imports** *m2-confid-chan* ../Refinement/Message  
**begin**

This refines the channel protocol using public-key encryption and adds a full-fledged Dolev-Yao adversary. In this variant, the adversary is realized using Paulson’s message derivation closure operators (as opposed to a collection of one-step message construction and decomposition events a la Strand spaces).

Proof tool configuration. Avoid annoying automatic unfolding of *dom* (again).

**declare** *domIff* [*simp, iff del*]

A general lemma about *parts* (move?!).

**lemmas** *parts-insertD = parts-insert* [*THEN equalityD1, THEN subsetD*]

### 2.5.1 State and observations

We extend the state of *m1* with two confidential channels between each pair of agents, one channel for each protocol message.

**record** *m3-state* = *m1-state* +  
*IK* :: *msg set* — intruder knowledge

Observations: local agent states.

**type-synonym**  
*m3-obs* = *m1-obs*

**definition**  
*m3-obs* :: *m3-state* ⇒ *m3-obs* **where**  
*m3-obs* *s* ≡ (  
   *runs* = *runs s*  
 )

## 2.5.2 Events

### definition

$m3\text{-step1} :: [rid\text{-}t, agent, agent, nonce] \Rightarrow (m3\text{-state} \times m3\text{-state}) \text{ set}$

### where

$m3\text{-step1 } Ra \ A \ B \ Na \equiv \{(s, s1)\}.$

— guards:

$Ra \notin \text{dom}(\text{runs } s) \wedge$

$Na = Ra\$0 \wedge$

— actions:

$s1 = s\{$

$\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [])),$

$IK := \text{insert}(\text{Crypt}(\text{pubK } B) \{ \text{Nonce } Na, \text{Agent } A \}) (IK \ s)$

$\}$

}

### definition

$m3\text{-step2} ::$

$[rid\text{-}t, agent, agent, nonce, nonce] \Rightarrow (m3\text{-state} \times m3\text{-state}) \text{ set}$

### where

$m3\text{-step2 } Rb \ A \ B \ Na \ Nb \equiv \{(s, s1)\}.$

— guards

$Rb \notin \text{dom}(\text{runs } s) \wedge$

$Nb = Rb\$0 \wedge$

$\text{Crypt}(\text{pubK } B) \{ \text{Nonce } Na, \text{Agent } A \} \in IK \ s \wedge$  — receive msg 1

— actions

$s1 = s\{$

$\text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [a\text{Non } Na])),$

$IK := \text{insert}(\text{Crypt}(\text{pubK } A) \{ \text{Nonce } Na, \text{Nonce } Nb, \text{Agent } B \}) (IK \ s)$

$\}$

}

### definition

$m3\text{-step3} :: [rid\text{-}t, agent, agent, nonce, nonce] \Rightarrow (m3\text{-state} \times m3\text{-state}) \text{ set}$

### where

$m3\text{-step3 } Ra \ A \ B \ Na \ Nb \equiv \{(s, s1)\}.$

— guards

$\text{runs } s \ Ra = \text{Some}(\text{Init}, [A, B], []) \wedge$

$Na = Ra\$0 \wedge$

$\text{Crypt}(\text{pubK } A) \{ \text{Nonce } Na, \text{Nonce } Nb, \text{Agent } B \} \in IK \ s \wedge$  — recv msg2

— actions

$s1 = s\{$

$\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [a\text{Non } Nb]))$

$\}$

}

Standard Dolev-Yao intruder.

**definition**

$m3-DY-fake :: (m3-state \times m3-state) \text{ set}$

**where**

$m3-DY-fake \equiv \{(s, s1).$

— actions:

$s1 = s \{ IK := synth (analz (IK s)) \}$

Transition system.

**definition**

$m3-init :: m3-state \text{ set}$

**where**

$m3-init \equiv \{ \{$   
 $runs = Map.empty,$   
 $IK = (Key'priK'bad) \cup (Key'range pubK) \cup (Key'shrK'bad)$   
 $\} \}$

**definition**

$m3-trans :: (m3-state \times m3-state) \text{ set}$  **where**

$m3-trans \equiv (\bigcup A B Ra Rb Na Nb.$

$m3-step1 Ra A B Na \cup$

$m3-step2 Rb A B Na Nb \cup$

$m3-step3 Ra A B Na Nb \cup$

$m3-DY-fake \cup$

$Id$

$)$

**definition**

$m3 :: (m3-state, m3-obs) \text{ spec}$  **where**

$m3 \equiv \{$

$init = m3-init,$

$trans = m3-trans,$

$obs = m3-obs$

$\}$

**lemmas**  $m3-defs =$

$m3-def m3-init-def m3-trans-def m3-obs-def$

$m3-step1-def m3-step2-def m3-step3-def$

$m3-DY-fake-def$

### 2.5.3 Invariants

Automatic tool tuning. Tame too-aggressive pair decomposition, which is declared as a safe elim rule (`[elim!]`).

**lemmas**  $MPair-parts$  [`rule del, elim`]

**lemmas**  $MPair-analz$  [`rule del, elim`]

Specialize injectiveness of `parts` and `analz` to enable aggressive application.

**lemmas**  $parts-Inj-IK = parts.Inj$  [**where**  $H=IK$  **for**  $s$ ]

**lemmas** *analz-Inj-IK* = *analz.Inj* [**where**  $H=IK$  *s* **for** *s*]

**declare** *analz-into-parts* [*dest*]

### inv1: Key secrecy

Decryption keys are secret, that is, the intruder only knows private keys of corrupted agents.

#### definition

*m3-inv1-keys* :: *m3-state set* **where**  
*m3-inv1-keys*  $\equiv \{s. \forall A.$   
 $\text{Key } (\text{priK } A) \in \text{parts } (IK \ s) \longrightarrow A \in \text{bad}$   
 $\}$

**lemmas** *m3-inv1-keysI* = *m3-inv1-keys-def* [*THEN setc-def-to-intro, rule-format*]

**lemmas** *m3-inv1-keysE* [*elim*] =

*m3-inv1-keys-def* [*THEN setc-def-to-elim, rule-format*]

**lemmas** *m3-inv1-keysD* [*dest*] =

*m3-inv1-keys-def* [*THEN setc-def-to-dest, rule-format, rotated 1*]

**lemma** *PO-m3-inv1-keys-init* [*iff*]:

$\text{init } m3 \subseteq m3\text{-inv1-keys}$

*<proof>*

**lemma** *PO-m3-inv1-keys-trans* [*iff*]:

$\{m3\text{-inv1-keys}\} \text{ trans } m3 \{> m3\text{-inv1-keys}\}$

*<proof>*

**lemma** *PO-m3-inv1-keys* [*iff*]:  $\text{reach } m3 \subseteq m3\text{-inv1-keys}$

*<proof>*

## 2.5.4 Simulation relation

Simulation relation is canonical. It states that the protocol messages appearing in the intruder knowledge refine those occurring on the abstract confidential channels. Moreover, if the concrete intruder knows a nonce then so does the abstract one (as defined by *ink*).

Abstraction function on sets of messages.

#### inductive-set

*abs-msg* :: *msg set*  $\Rightarrow$  *chmsg set*

**for** *H* :: *msg set*

#### where

*am-msg1*:

$\text{Crypt } (\text{pubK } B) \{ \text{Nonce } Na, \text{Agent } A \} \in H$   
 $\Longrightarrow \text{Confid } A \ B \ (\text{Msg } [a\text{Non } Na]) \in \text{abs-msg } H$

| *am-msg2*:

$\text{Crypt } (\text{pubK } A) \{ \text{Nonce } Na, \text{Nonce } Nb, \text{Agent } B \} \in H$   
 $\Longrightarrow \text{Confid } B \ A \ (\text{Msg } [a\text{Non } Na, a\text{Non } Nb]) \in \text{abs-msg } H$

**declare** *abs-msg.intros* [*intro!*]

**declare** *abs-msg.cases* [elim!]

The simulation relation is canonical. It states that the protocol messages in the intruder knowledge refine the abstract messages appearing on the confidential channels.

**definition**

$R23\text{-msgs} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23\text{-msgs} \equiv \{(s, t). \text{abs-msg} (\text{parts} (IK\ t)) \subseteq \text{chan } s\} \quad \text{— with } \text{parts!}$

**definition**

$R23\text{-non} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23\text{-non} \equiv \{(s, t). \forall N. \text{Nonce } N \in \text{analz} (IK\ t) \longrightarrow a\text{Non } N \in \text{extr ik0} (\text{chan } s)\}$

**definition**

$R23\text{-pres} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23\text{-pres} \equiv \{(s, t). \text{runs } s = \text{runs } t\}$

**definition**

$R23 :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23 \equiv R23\text{-msgs} \cap R23\text{-non} \cap R23\text{-pres}$

**lemmas**  $R23\text{-defs} =$

$R23\text{-def } R23\text{-msgs-def } R23\text{-non-def } R23\text{-pres-def}$

**lemmas**  $R23\text{-msgsI} =$

$R23\text{-msgs-def} [THEN\ \text{rel-def-to-intro}, \text{simplified}, \text{rule-format}]$

**lemmas**  $R23\text{-msgsE} [elim] =$

$R23\text{-msgs-def} [THEN\ \text{rel-def-to-elim}, \text{simplified}, \text{rule-format}]$

**lemmas**  $R23\text{-msgsE}' [elim] =$

$R23\text{-msgs-def} [THEN\ \text{rel-def-to-dest}, \text{simplified}, \text{rule-format}, THEN\ \text{subsetD}]$

**lemmas**  $R23\text{-nonI} =$

$R23\text{-non-def} [THEN\ \text{rel-def-to-intro}, \text{simplified}, \text{rule-format}]$

**lemmas**  $R23\text{-nonE} [elim] =$

$R23\text{-non-def} [THEN\ \text{rel-def-to-elim}, \text{simplified}, \text{rule-format}]$

**lemmas**  $R23\text{-presI} =$

$R23\text{-pres-def} [THEN\ \text{rel-def-to-intro}, \text{simplified}, \text{rule-format}]$

**lemmas**  $R23\text{-presE} [elim] =$

$R23\text{-pres-def} [THEN\ \text{rel-def-to-elim}, \text{simplified}, \text{rule-format}]$

**lemmas**  $R23\text{-intros} = R23\text{-msgsI } R23\text{-nonI } R23\text{-presI}$

Mediator function.

**abbreviation**

$\text{med32} :: m3\text{-obs} \Rightarrow m2\text{-obs} \text{ where}$   
 $\text{med32} \equiv \text{id}$

## 2.5.5 Misc lemmas

General facts about *abs-msg*

**lemma**  $\text{abs-msg-empty}: \text{abs-msg } \{\} = \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *abs-msg-Un* [*simp*]:  
 $abs\text{-}msg (G \cup H) = abs\text{-}msg G \cup abs\text{-}msg H$   
 ⟨*proof*⟩

**lemma** *abs-msg-mono* [*elim*]:  
 $\llbracket m \in abs\text{-}msg G; G \subseteq H \rrbracket \implies m \in abs\text{-}msg H$   
 ⟨*proof*⟩

**lemma** *abs-msg-insert-mono* [*intro*]:  
 $\llbracket m \in abs\text{-}msg H \rrbracket \implies m \in abs\text{-}msg (insert\ m'\ H)$   
 ⟨*proof*⟩

Abstraction of concretely fakeable message yields abstractly fakeable messages. This is the key lemma for the refinement of the intruder.

**lemma** *abs-msg-DY-subset-fake*:  
 $\llbracket (s, t) \in R23\text{-}msgs; (s, t) \in R23\text{-}non; t \in m3\text{-}inv1\text{-}keys \rrbracket$   
 $\implies abs\text{-}msg (synth (analz (IK\ t))) \subseteq fake\ ik0 (dom (runs\ s)) (chan\ s)$   
 ⟨*proof*⟩

**lemma** *abs-msg-parts-subset-fake*:  
 $\llbracket (s, t) \in R23\text{-}msgs \rrbracket$   
 $\implies abs\text{-}msg (parts (IK\ t)) \subseteq fake\ ik0 (-dom (runs\ s)) (chan\ s)$   
 ⟨*proof*⟩

**declare** *abs-msg-DY-subset-fake* [*simp*, *intro!*]  
**declare** *abs-msg-parts-subset-fake* [*simp*, *intro!*]

## 2.5.6 Refinement proof

Proofs obligations.

**lemma** *PO-m3-step1-refines-m2-step1*:  
 $\{R23 \cap UNIV \times m3\text{-}inv1\text{-}keys\}$   
 $(m2\text{-}step1\ Ra\ A\ B\ Na), (m3\text{-}step1\ Ra\ A\ B\ Na)$   
 $\{>\ R23\}$   
 ⟨*proof*⟩

**lemma** *PO-m3-step2-refines-m2-step2*:  
 $\{R23 \cap UNIV \times m3\text{-}inv1\text{-}keys\}$   
 $(m2\text{-}step2\ Rb\ A\ B\ Na\ Nb), (m3\text{-}step2\ Rb\ A\ B\ Na\ Nb)$   
 $\{>\ R23\}$   
 ⟨*proof*⟩

**lemma** *PO-m3-step3-refines-m2-step3*:  
 $\{R23\}$   
 $(m2\text{-}step3\ Ra\ A\ B\ Na\ Nb), (m3\text{-}step3\ Ra\ A\ B\ Na\ Nb)$   
 $\{>\ R23\}$   
 ⟨*proof*⟩

Dolev-Yao fake event refines abstract fake event.

**lemma** *PO-m3-DY-fake-refines-m2-fake*:  
 $\{R23 \cap UNIV \times m3\text{-}inv1\text{-}keys\}$

$(m2\text{-fake}), (m3\text{-DY-fake})$   
 $\{> R23\}$   
 $\langle\text{proof}\rangle$

All together now...

**lemmas**  $PO\text{-}m3\text{-trans-refines-}m2\text{-trans} =$   
 $PO\text{-}m3\text{-step1-refines-}m2\text{-step1}$   $PO\text{-}m3\text{-step2-refines-}m2\text{-step2}$   
 $PO\text{-}m3\text{-step3-refines-}m2\text{-step3}$   $PO\text{-}m3\text{-DY-fake-refines-}m2\text{-fake}$

**lemma**  $PO\text{-}m3\text{-refines-init-}m2$  [*iff*]:  
 $init\ m3 \subseteq R23''(init\ m2)$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m3\text{-refines-trans-}m2$  [*iff*]:  
 $\{R23 \cap UNIV \times m3\text{-inv1-keys}\}$   
 $(trans\ m2), (trans\ m3)$   
 $\{> R23\}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}R23\text{-obs-consistent}$  [*iff*]:  
 $obs\text{-consistent}\ R23\ med32\ m2\ m3$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m3\text{-refines-}m2$  [*iff*]:  
 $refines$   
 $(R23 \cap UNIV \times m3\text{-inv1-keys})$   
 $med32\ m2\ m3$   
 $\langle\text{proof}\rangle$

**end**

## Chapter 3

# Key Establishment Protocols

In this chapter, we develop several key establishment protocols:

- Needham-Schroeder Shared Key (NSSK)
- core Kerberos IV and V, and
- Denning-Sacco.

### 3.1 Basic abstract key distribution (L1)

```
theory m1-keydist imports ../Refinement/Runs ../Refinement/s0g-secrecy
begin
```

The first refinement introduces the protocol roles, local memory of the agents and the communication structure of the protocol. For actual communication, the "receiver" directly reads the memory of the "sender".

It captures the core of essentials of server-based key distribution protocols: The server generates a key that the clients read from his memory. At this stage we are only interested in secrecy preservation, not in authentication.

```
declare option.split-asm [split]
declare domIff [simp, iff del]
```

```
consts
```

```
  sk :: nat           — identifier used for session keys
```

#### 3.1.1 State

Runs record the protocol participants (initiator, responder) and the keys learned during the execution. In later refinements, we will also add nonces and timestamps to the run record.

The variables *kn* and *az* from *s0g-secrecy-leak* are replaced by runs using a data refinement. Variable *lk* is concretized into variable *leak*.

We define the state in two separate record definitions. The first one has just a runs field and the second extends this with a leak field. Later refinements may define different state for leaks (e.g. to record more context).

**record** *m1r-state* =  
*runs* :: *runs-t*

**record** *m1x-state* = *m1r-state* +  
*leak* :: *key set* — keys leaked to attacker

**type-synonym** *m1x-obs* = *m1x-state*

Predicate types for invariants and transition relation types. Use the r-version for invariants and transitions if there is no reference to the leak variable. This improves reusability in later refinements.

**type-synonym** *'x m1r-pred* = *'x m1r-state-scheme set*

**type-synonym** *'x m1x-pred* = *'x m1x-state-scheme set*

**type-synonym** *'x m1r-trans* = (*'x m1r-state-scheme* × *'x m1r-state-scheme*) *set*

**type-synonym** *'x m1x-trans* = (*'x m1x-state-scheme* × *'x m1x-state-scheme*) *set*

### Key knowledge and authorization (reconstruction)

Key knowledge and authorization relations, reconstructed from the runs and an unspecified initial key setup. These auxiliary definitions are used in some event guards and in the simulation relation (see below).

Knowledge relation (reconstructed)

**inductive-set**

*knC* :: *runs-t* ⇒ (*key* × *agent*) *set* **for** *runz* :: *runs-t*

**where**

*knC-init*:

*runz Ra* = *Some (Init, [A, B], aKey K # al)* ⇒ (*K, A*) ∈ *knC runz*

| *knC-resp*:

*runz Rb* = *Some (Resp, [A, B], aKey K # al)* ⇒ (*K, B*) ∈ *knC runz*

| *knC-serv*:

[[ *Rs* ∈ *dom runz*; *fst (the (runz Rs))* = *Serv* ]] ⇒ (*sesK (Rs\$sk), Sv*) ∈ *knC runz*

| *knC-0*:

(*K, A*) ∈ *keySetup* ⇒ (*K, A*) ∈ *knC runz*

Authorization relation (reconstructed)

**inductive-set**

*azC* :: *runs-t* ⇒ (*key* × *agent*) *set* **for** *runz* :: *runs-t*

**where**

*azC-good*:

[[ *runz Rs* = *Some (Serv, [A, B], al)*; *C* ∈ {*A, B, Sv*} ]]  
⇒ (*sesK (Rs\$sk), C*) ∈ *azC runz*

| *azC-bad*:

[[ *runz Rs* = *Some (Serv, [A, B], al)*; *A* ∈ *bad* ∨ *B* ∈ *bad* ]]  
⇒ (*sesK (Rs\$sk), C*) ∈ *azC runz*

| *azC-0*:

[[ (*K, C*) ∈ *keySetup* ]] ⇒ (*K, C*) ∈ *azC runz*

**declare** *knC.intros* [*intro*]

**declare** *azC.intros* [*intro*]

Misc lemmas: empty state, projections, ...

**lemma** *knC-empty* [*simp*]: *knC Map.empty* = *keySetup*  
⟨*proof*⟩

**lemma** *azC-empty* [*simp*]: *azC Map.empty* = *keySetup*  
⟨*proof*⟩

*azC* and run abstraction

**lemma** *azC-map-runs* [*simp*]: *azC (map-runs h runz)* = *azC runz*  
⟨*proof*⟩

Update lemmas for *knC*

**lemma** *knC-upd-Init-Resp-None*:  
[[ *R* ∉ *dom runz*; *rol* ∈ {*Init*, *Resp*} ]]  
⇒ *knC (runz(R ↦ (rol, [A, B], [])))* = *knC runz*  
⟨*proof*⟩

**lemma** *knC-upd-Init-Some*:  
[[ *runz Ra* = *Some (Init, [A, B], [])* ]]  
⇒ *knC (runz(Ra ↦ (Init, [A, B], [aKey Kab])))* = *insert (Kab, A) (knC runz)*  
⟨*proof*⟩

**lemma** *knC-upd-Resp-Some*:  
[[ *runz Ra* = *Some (Resp, [A, B], [])* ]]  
⇒ *knC (runz(Ra ↦ (Resp, [A, B], [aKey Kab])))* = *insert (Kab, B) (knC runz)*  
⟨*proof*⟩

**lemma** *knC-upd-Server*:  
[[ *Rs* ∉ *dom runz* ]]  
⇒ *knC (runz(Rs ↦ (Serv, [A, B], [])))* = *insert (sesK (Rs\$sk), Sv) (knC runz)*  
⟨*proof*⟩

**lemmas** *knC-upd-lemmas* [*simp*] =  
*knC-upd-Init-Resp-None knC-upd-Init-Some knC-upd-Resp-Some*  
*knC-upd-Server*

Update lemmas for *azC*

**lemma** *azC-upd-Init-None*:  
[[ *Ra* ∉ *dom runz* ]]  
⇒ *azC (runz(Ra ↦ (Init, [A, B], [])))* = *azC runz*  
⟨*proof*⟩

**lemma** *azC-upd-Resp-None*:  
[[ *Rb* ∉ *dom runz* ]]  
⇒ *azC (runz(Rb ↦ (Resp, [A, B], [])))* = *azC runz*  
⟨*proof*⟩

**lemma** *azC-upd-Init-Some*:  
[[ *runz Ra* = *Some (Init, [A, B], [])* ]]  
⇒ *azC (runz(Ra ↦ (Init, [A, B], al)))* = *azC runz*

$\langle \text{proof} \rangle$

**lemma** *azC-upd-Resp-Some*:

$$\begin{aligned} & \llbracket \text{runz } Rb = \text{Some } (\text{Resp}, [A, B], []) \rrbracket \\ & \implies \text{azC } (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], al))) = \text{azC } \text{runz} \end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *azC-upd-Serv-bad*:

$$\begin{aligned} & \llbracket Rs \notin \text{dom } \text{runz}; A \in \text{bad} \vee B \in \text{bad} \rrbracket \\ & \implies \text{azC } (\text{runz}(Rs \mapsto (\text{Serv}, [A, B], al))) = \text{azC } \text{runz} \cup \{\text{sesK } (Rs\$sk)\} \times \text{UNIV} \end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *azC-upd-Serv-good*:

$$\begin{aligned} & \llbracket Rs \notin \text{dom } \text{runz}; K = \text{sesK } (Rs\$sk); A \notin \text{bad}; B \notin \text{bad} \rrbracket \\ & \implies \text{azC } (\text{runz}(Rs \mapsto (\text{Serv}, [A, B], al))) \\ & = \text{azC } \text{runz} \cup \{(K, A), (K, B), (K, Sv)\} \end{aligned}$$

$\langle \text{proof} \rangle$

**lemma** *azC-upd-Serv*:

$$\begin{aligned} & \llbracket Rs \notin \text{dom } \text{runz}; K = \text{sesK } (Rs\$sk) \rrbracket \\ & \implies \text{azC } (\text{runz}(Rs \mapsto (\text{Serv}, [A, B], al))) = \\ & \quad \text{azC } \text{runz} \cup \{K\} \times (\text{if } A \notin \text{bad} \wedge B \notin \text{bad} \text{ then } \{A, B, Sv\} \text{ else } \text{UNIV}) \end{aligned}$$

$\langle \text{proof} \rangle$

**lemmas** *azC-upd-lemmas [simp]* =

$$\begin{aligned} & \text{azC-upd-Init-None } \text{azC-upd-Resp-None} \\ & \text{azC-upd-Init-Some } \text{azC-upd-Resp-Some } \text{azC-upd-Serv} \end{aligned}$$

### 3.1.2 Events

**definition** — by  $A$ , refines skip

$$m1x\text{-step1} :: [\text{rid-}t, \text{agent}, \text{agent}] \Rightarrow 'x \text{ m1r-trans}$$

**where**

$$m1x\text{-step1 } Ra \ A \ B \equiv \{(s, s1)\}.$$

— guards:

$$Ra \notin \text{dom } (\text{runs } s) \wedge \quad \text{— } Ra \text{ is fresh}$$

— actions:

— create initiator thread

$$s1 = s \langle \text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [])) \rangle$$

**definition** — by  $B$ , refines skip

$$m1x\text{-step2} :: [\text{rid-}t, \text{agent}, \text{agent}] \Rightarrow 'x \text{ m1r-trans}$$

**where**

$$m1x\text{-step2 } Rb \ A \ B \equiv \{(s, s1)\}.$$

— guards:

$$Rb \notin \text{dom } (\text{runs } s) \wedge \quad \text{— } Rb \text{ is fresh}$$

— actions:

— create responder thread

$s1 = s \{ \text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [])) \}$

**definition** — by  $Sv$ , refines  $s0g\text{-gen}$   
 $m1x\text{-step3} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}] \Rightarrow 'x \text{ m1r-trans}$   
**where**  
 $m1x\text{-step3 } Rs \ A \ B \ Kab \equiv \{(s, s1)\}.$

— guards:  
 $Rs \notin \text{dom } (\text{runs } s) \wedge$  —  $Rs$  is fresh  
 $Kab = \text{sesK } (Rs\$sk) \wedge$  — generate session key

— actions:  
 $s1 = s \{ \text{runs} := (\text{runs } s)(Rs \mapsto (\text{Serv}, [A, B], [])) \}$

**definition** — by  $A$ , refines  $s0g\text{-learn}$   
 $m1x\text{-step4} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}] \Rightarrow 'x \text{ m1x-trans}$   
**where**  
 $m1x\text{-step4 } Ra \ A \ B \ Kab \equiv \{(s, s1)\}.$

— guards:  
 $\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], []) \wedge$   
 $(Kab \notin \text{leak } s \longrightarrow (Kab, A) \in \text{azC } (\text{runs } s)) \wedge$  — authorization guard

— actions:  
 $s1 = s \{ \text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [aKey \ Kab])) \}$

**definition** — by  $B$ , refines  $s0g\text{-learn}$   
 $m1x\text{-step5} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}] \Rightarrow 'x \text{ m1x-trans}$   
**where**  
 $m1x\text{-step5 } Rb \ A \ B \ Kab \equiv \{(s, s1)\}.$

— guards:  
 $\text{runs } s \ Rb = \text{Some } (\text{Resp}, [A, B], []) \wedge$   
 $(Kab \notin \text{leak } s \longrightarrow (Kab, B) \in \text{azC } (\text{runs } s)) \wedge$  — authorization guard

— actions:  
 $s1 = s \{ \text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [aKey \ Kab])) \}$

**definition** — by attacker, refines  $s0g\text{-leak}$   
 $m1x\text{-leak} :: \text{rid-}t \Rightarrow 'x \text{ m1x-trans}$   
**where**  
 $m1x\text{-leak } Rs \equiv \{(s, s1)\}.$

— guards:  
 $Rs \in \text{dom } (\text{runs } s) \wedge$   
 $\text{fst } (\text{the } (\text{runs } s \ Rs)) = \text{Serv} \wedge$  — compromise server run  $Rs$

— actions:  
 $s1 = s \{ \text{leak} := \text{insert } (\text{sesK } (Rs\$sk)) (\text{leak } s) \}$

### 3.1.3 Specification

#### definition

$m1x-init :: m1x-state\ set$

#### where

$m1x-init \equiv \{ \langle \langle$   
 $runs = Map.empty,$   
 $leak = corrKey \quad \text{— statically corrupted keys initially leaked}$   
 $\rangle \rangle \}$

#### definition

$m1x-trans :: 'x\ m1x-trans\ \mathbf{where}$   
 $m1x-trans \equiv (\bigcup A\ B\ Ra\ Rb\ Rs\ Kab.$   
 $m1x-step1\ Ra\ A\ B \cup$   
 $m1x-step2\ Rb\ A\ B \cup$   
 $m1x-step3\ Rs\ A\ B\ Kab \cup$   
 $m1x-step4\ Ra\ A\ B\ Kab \cup$   
 $m1x-step5\ Rb\ A\ B\ Kab \cup$   
 $m1x-leak\ Rs \cup$   
 $Id$   
 $)$

#### definition

$m1x :: (m1x-state, m1x-obs)\ spec\ \mathbf{where}$   
 $m1x \equiv \langle \langle$   
 $init = m1x-init,$   
 $trans = m1x-trans,$   
 $obs = id$   
 $\rangle \rangle$

#### lemmas $m1x-defs =$

$m1x-def\ m1x-init-def\ m1x-trans-def$   
 $m1x-step1-def\ m1x-step2-def\ m1x-step3-def\ m1x-step4-def\ m1x-step5-def$   
 $m1x-leak-def$

**lemma**  $m1x-obs-id$  [simp]:  $obs\ m1x = id$   
 $\langle proof \rangle$

### 3.1.4 Invariants

#### inv1: Key definedness

Only run identifiers or static keys can be (concretely) known or authorized keys. (This reading corresponds to the contraposition of the property expressed below.)

#### definition

$m1x-inv1-key :: m1x-state\ set$

#### where

$m1x-inv1-key \equiv \{s. \forall Rs\ A.$   
 $Rs \notin dom\ (runs\ s) \longrightarrow$   
 $(sesK\ (Rs\$sk), A) \notin knC\ (runs\ s) \wedge$   
 $(sesK\ (Rs\$sk), A) \notin azC\ (runs\ s) \wedge$   
 $sesK\ (Rs\$sk) \notin leak\ s$   
 $\}$

**lemmas**  $m1x\text{-inv1}\text{-key}I = m1x\text{-inv1}\text{-key}\text{-def}$  [*THEN setc-def-to-intro, rule-format*]

**lemmas**  $m1x\text{-inv1}\text{-key}E$  [*elim*] =  
 $m1x\text{-inv1}\text{-key}\text{-def}$  [*THEN setc-def-to-elim, rule-format*]

**lemmas**  $m1x\text{-inv1}\text{-key}D$  [*dest*] =  
 $m1x\text{-inv1}\text{-key}\text{-def}$  [*THEN setc-def-to-dest, rule-format, rotated 1*]

Invariance proof.

**lemma**  $PO\text{-}m1x\text{-inv1}\text{-key}\text{-init}$  [*iff*]:

$init\ m1x \subseteq m1x\text{-inv1}\text{-key}$

*<proof>*

**lemma**  $PO\text{-}m1x\text{-inv1}\text{-key}\text{-trans}$  [*iff*]:

$\{m1x\text{-inv1}\text{-key}\}\ trans\ m1x \{>\ m1x\text{-inv1}\text{-key}\}$

*<proof>*

**lemma**  $PO\text{-}m1x\text{-inv1}\text{-key}$  [*iff*]:  $reach\ m1x \subseteq m1x\text{-inv1}\text{-key}$

*<proof>*

### 3.1.5 Refinement of s0g

med10: The mediator function maps a concrete observation to an abstract one.

**definition**

$med01x :: m1x\text{-obs} \Rightarrow key\ s0g\text{-obs}$

**where**

$med01x\ t \equiv (\ | kn = knC\ (runs\ t), az = azC\ (runs\ t), lk = leak\ t\ |)$

R01: The simulation relation expresses key knowledge and authorization in terms of the client and server run information.

**definition**

$R01x :: (key\ s0g\text{-state} \times m1x\text{-state})\ set$  **where**

$R01x \equiv \{(s, t). s = med01x\ t\}$

**lemmas**  $R01x\text{-defs} = R01x\text{-def}\ med01x\text{-def}$

Refinement proof.

**lemma**  $PO\text{-}m1x\text{-step1}\text{-refines}\text{-skip}$ :

$\{R01x\}$

$Id, (m1x\text{-step1}\ Ra\ A\ B)$

$\{>\ R01x\}$

*<proof>*

**lemma**  $PO\text{-}m1x\text{-step2}\text{-refines}\text{-skip}$ :

$\{R01x\}$

$Id, (m1x\text{-step2}\ Rb\ A\ B)$

$\{>\ R01x\}$

*<proof>*

**lemma**  $PO\text{-}m1x\text{-step3}\text{-refines}\text{-s0g}\text{-gen}$ :

$\{R01x \cap UNIV \times m1x\text{-inv1}\text{-key}\}$

$(s0g\text{-gen}\ Kab\ Sv\ \{Sv, A, B\}), (m1x\text{-step3}\ Rs\ A\ B\ Kab)$

$\{> R01x\}$   
 $\langle proof \rangle$

**lemma** *PO-m1x-step4-refines-s0g-learn*:

$\{R01x\}$   
 $(s0g-learn\ Kab\ A), (m1x-step4\ Ra\ A\ B\ Kab)$   
 $\{> R01x\}$   
 $\langle proof \rangle$

**lemma** *PO-m1x-step5-refines-s0g-learn*:

$\{R01x\}$   
 $(s0g-learn\ Kab\ B), (m1x-step5\ Rb\ A\ B\ Kab)$   
 $\{> R01x\}$   
 $\langle proof \rangle$

**lemma** *PO-m1x-leak-refines-s0g-leak*:

$\{R01x\}$   
 $(s0g-leak\ (sesK\ (Rs\$sk))), (m1x-leak\ Rs)$   
 $\{> R01x\}$   
 $\langle proof \rangle$

All together now...

**lemmas** *PO-m1x-trans-refines-s0g-trans =*

*PO-m1x-step1-refines-skip PO-m1x-step2-refines-skip*  
*PO-m1x-step3-refines-s0g-gen PO-m1x-step4-refines-s0g-learn*  
*PO-m1x-step5-refines-s0g-learn PO-m1x-leak-refines-s0g-leak*

**lemma** *PO-m1x-refines-init-s0g [iff]*:

$init\ m1x \subseteq R01x \text{“}(init\ s0g)$   
 $\langle proof \rangle$

**lemma** *PO-m1x-refines-trans-s0g [iff]*:

$\{R01x \cap UNIV \times m1x-inv1-key\}$   
 $(trans\ s0g), (trans\ m1x)$   
 $\{> R01x\}$   
 $\langle proof \rangle$

Observation consistency.

**lemma** *obs-consistent-med01x [iff]*:

$obs-consistent\ R01x\ med01x\ s0g\ m1x$   
 $\langle proof \rangle$

Refinement result.

**lemma** *PO-m1x-refines-s0g [iff]*:

*refines*  
 $(R01x \cap UNIV \times m1x-inv1-key)$   
 $med01x\ s0g\ m1x$   
 $\langle proof \rangle$

**lemma** *m1x-implements-s0g [iff]: implements med01x s0g m1x*

$\langle proof \rangle$

### 3.1.6 Derived invariants

#### inv2: Secrecy

Secrecy, expressed in terms of runs.

##### definition

$m1x\text{-secrecy} :: 'x\ m1x\text{-pred}$

##### where

$m1x\text{-secrecy} \equiv \{s.\ knC\ (runs\ s) \subseteq azC\ (runs\ s) \cup leak\ s \times UNIV\}$

**lemmas**  $m1x\text{-secrecy}I = m1x\text{-secrecy}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

**lemmas**  $m1x\text{-secrecy}E\ [elim] = m1x\text{-secrecy}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

Invariance proof.

**lemma**  $PO\text{-}m1x\text{-obs}\text{-secrecy}\ [iff]:\ oreach\ m1x \subseteq m1x\text{-secrecy}$   
*<proof>*

**lemma**  $PO\text{-}m1x\text{-secrecy}\ [iff]:\ reach\ m1x \subseteq m1x\text{-secrecy}$   
*<proof>*

end

## 3.2 Abstract (i/n)-authenticated key transport (L1)

**theory**  $m1\text{-keydist}\text{-iirn}$  **imports**  $m1\text{-keydist}\ .. /Refinement/a0i\text{-agree}$   
**begin**

We add authentication for the initiator and responder to the basic server-based key transport protocol:

1. the initiator injectively agrees with the server on the key and some additional data
2. the responder non-injectively agrees with the server on the key and some additional data.

The "additional data" is a parameter of this model.

**declare**  $option.\text{split}\ [split]$

##### consts

$na :: nat$

### 3.2.1 State

The state type remains the same, but in this model we will record nonces and timestamps in the run frame.

**type-synonym**  $m1a\text{-state} = m1x\text{-state}$

**type-synonym**  $m1a\text{-obs} = m1x\text{-obs}$

**type-synonym**  $'x\ m1a\text{-pred} = 'x\ m1x\text{-pred}$

**type-synonym**  $'x\ m1a\text{-trans} = 'x\ m1x\text{-trans}$

We need some parameters regarding the list of freshness values stored by the server. These should be defined in further refinements.

**consts**

$is\text{-len} :: nat$  — num of agreeing list elements for initiator-server  
 $rs\text{-len} :: nat$  — num of agreeing list elements for responder-server

### 3.2.2 Events

**definition** — by  $A$ , refines  $m1x\text{-step1}$

$m1a\text{-step1} :: [rid\text{-}t, agent, agent, nonce] \Rightarrow 'x\ m1r\text{-trans}$

**where**

$m1a\text{-step1}\ Ra\ A\ B\ Na \equiv \{(s, s1)\}.$

— guards:

$Ra \notin dom\ (runs\ s) \wedge$  —  $Ra$  is fresh  
 $Na = Ra\$na \wedge$  — NEW: generate a nonce

— actions:

— create initiator thread

$s1 = s\{ runs := (runs\ s)(Ra \mapsto (Init, [A, B], [])) \}$

**definition** — by  $B$ , refines  $m1x\text{-step2}$

$m1a\text{-step2} :: [rid\text{-}t, agent, agent] \Rightarrow 'x\ m1r\text{-trans}$

**where**

$m1a\text{-step2} \equiv m1x\text{-step2}$

**definition** — by  $Sv$ , refines  $m1x\text{-step3}$

$m1a\text{-step3} :: [rid\text{-}t, agent, agent, key, nonce, atom\ list] \Rightarrow 'x\ m1r\text{-trans}$

**where**

$m1a\text{-step3}\ Rs\ A\ B\ Kab\ Na\ al \equiv \{(s, s1)\}.$

— guards:

$Rs \notin dom\ (runs\ s) \wedge$  — fresh run id  
 $Kab = sesK\ (Rs\$sk) \wedge$  — generate session key

— actions:

$s1 = s\{ runs := (runs\ s)(Rs \mapsto (Serv, [A, B], aNon\ Na\ \# al)) \}$

**definition** — by  $A$ , refines  $m1x\text{-step4}$

$m1a\text{-step4} :: [rid\text{-}t, agent, agent, nonce, key, atom\ list] \Rightarrow 'x\ m1a\text{-trans}$

**where**

$m1a\text{-step4}\ Ra\ A\ B\ Na\ Kab\ nla \equiv \{(s, s')\}.$

— guards:

$runs\ s\ Ra = Some\ (Init, [A, B], []) \wedge$   
 $(Kab \notin leak\ s \longrightarrow (Kab, A) \in azC\ (runs\ s)) \wedge$  — authorization guard  
 $Na = Ra\$na \wedge$  — fix parameter

— new guard for agreement with server on  $(Kab, B, Na, isl)$ ,

— where  $isl = take\ is\text{-len}\ nla$ ; injectiveness by including  $Na$

$(A \notin bad \longrightarrow (\exists Rs. Kab = sesK\ (Rs\$sk) \wedge$   
 $runs\ s\ Rs = Some\ (Serv, [A, B], aNon\ Na\ \# take\ is\text{-len}\ nla))) \wedge$

— actions:  
 $s' = s \langle \text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], aKey \text{ Kab } \# \text{ nla})) \rangle$   
 $\}$

**definition** — by  $B$ , refines  $m1x\text{-step5}$   
 $m1a\text{-step5} :: [\text{rid-}t, \text{agent}, \text{key}, \text{atom list}] \Rightarrow 'x \text{ m1a-trans}$

**where**

$m1a\text{-step5 } Rb \ A \ B \ Kab \ nlb \equiv \{(s, s1).$

— guards:

$\text{runs } s \ Rb = \text{Some } (\text{Resp}, [A, B], []) \wedge$

$(Kab \notin \text{leak } s \longrightarrow (Kab, B) \in \text{azC } (\text{runs } s)) \wedge$  — authorization guard

— guard for showing agreement with server on  $(Kab, A, \text{rsl})$ ,

— where  $\text{rsl} = \text{take } \text{rs-len } \text{nlb}$ ; this agreement is non-injective

$(B \notin \text{bad} \longrightarrow (\exists Rs \ Na. Kab = \text{sesK } (Rs\$sk) \wedge$

$\text{runs } s \ Rs = \text{Some } (\text{Serv}, [A, B], aNon \ Na \ \# \ \text{take } \text{rs-len } \text{nlb}))) \wedge$

— actions:

$s1 = s \langle \text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], aKey \ \text{Kab} \ \# \ \text{nlb})) \rangle$   
 $\}$

**definition** — by attacker, refines  $m1x\text{-leak}$

$m1a\text{-leak} :: \text{rid-}t \Rightarrow 'x \text{ m1x-trans}$

**where**

$m1a\text{-leak} = m1x\text{-leak}$

### 3.2.3 Specification

**definition**

$m1a\text{-init} :: m1a\text{-state set}$

**where**

$m1a\text{-init} \equiv m1x\text{-init}$

**definition**

$m1a\text{-trans} :: 'x \text{ m1a-trans}$  **where**

$m1a\text{-trans} \equiv (\bigcup A \ B \ Ra \ Rb \ Rs \ Na \ Kab \ nls \ nla \ nlb.$

$m1a\text{-step1 } Ra \ A \ B \ Na \cup$

$m1a\text{-step2 } Rb \ A \ B \cup$

$m1a\text{-step3 } Rs \ A \ B \ Kab \ Na \ nls \cup$

$m1a\text{-step4 } Ra \ A \ B \ Na \ Kab \ nla \cup$

$m1a\text{-step5 } Rb \ A \ B \ Kab \ nlb \cup$

$m1a\text{-leak } Rs \cup$

$Id$

)

**definition**

$m1a :: (m1a\text{-state}, m1a\text{-obs}) \text{ spec}$  **where**

$m1a \equiv \langle$

$\text{init} = m1a\text{-init},$

$\text{trans} = m1a\text{-trans},$

$\text{obs} = \text{id}$

)

**lemma** *init-m1a*: *init m1a = m1a-init*  
*<proof>*

**lemma** *trans-m1a*: *trans m1a = m1a-trans*  
*<proof>*

**lemma** *obs-m1a [simp]*: *obs m1a = id*  
*<proof>*

**lemmas** *m1a-loc-defs =*  
*m1a-def m1a-init-def m1a-trans-def*  
*m1a-step1-def m1a-step2-def m1a-step3-def m1a-step4-def m1a-step5-def*  
*m1a-leak-def*

**lemmas** *m1a-defs = m1a-loc-defs m1x-defs*

### 3.2.4 Invariants

#### inv0: Finite domain

There are only finitely many runs. This is needed to establish the responder/initiator agreement.

#### definition

*m1a-inv0-fin* :: 'x m1r-pred

#### where

*m1a-inv0-fin*  $\equiv$  {s. finite (dom (runs s))}

**lemmas** *m1a-inv0-finI = m1a-inv0-fin-def [THEN setc-def-to-intro, rule-format]*

**lemmas** *m1a-inv0-finE [elim] = m1a-inv0-fin-def [THEN setc-def-to-elim, rule-format]*

**lemmas** *m1a-inv0-finD = m1a-inv0-fin-def [THEN setc-def-to-dest, rule-format]*

Invariance proof.

**lemma** *PO-m1a-inv0-fin-init [iff]*:

*init m1a*  $\subseteq$  *m1a-inv0-fin*

*<proof>*

**lemma** *PO-m1a-inv0-fin-trans [iff]*:

{*m1a-inv0-fin*} *trans m1a*  $\{>$  *m1a-inv0-fin*

*<proof>*

**lemma** *PO-m1a-inv0-fin [iff]*: *reach m1a*  $\subseteq$  *m1a-inv0-fin*

*<proof>*

### 3.2.5 Refinement of m1x

#### Simulation relation

Define run abstraction.

#### fun

*rm1x1a* :: *role-t*  $\Rightarrow$  *atom list*  $\Rightarrow$  *atom list*

**where**

$rm1x1a\ Init = take\ 1$  — take  $Kab$  from  $Kab \# nla$   
|  $rm1x1a\ Resp = take\ 1$  — take  $Kab$  from  $Kab \# nlb$   
|  $rm1x1a\ Serv = take\ 0$  — drop all from  $[Na]$

**abbreviation**

$runs1x1a :: runs\ t \Rightarrow runs\ t$  **where**  
 $runs1x1a \equiv map\ runs\ rm1x1a$

med1x1: The mediator function maps a concrete observation to an abstract one.

**definition**

$med1x1a :: m1a\ obs \Rightarrow m1x\ obs$  **where**  
 $med1x1a\ t \equiv (\ runs = runs1x1a\ (runs\ t), leak = leak\ t \ )$

R1x1a: The simulation relation is defined in terms of the mediator function.

**definition**

$R1x1a :: (m1x\ state \times m1a\ state)$  set **where**  
 $R1x1a \equiv \{(s, t). s = med1x1a\ t\}$

**lemmas**  $R1x1a\ defs =$

$R1x1a\ def\ med1x1a\ def$

**Refinement proof**

**lemma**  $PO\ m1a\ step1\ refines\ m1x\ step1$ :

$\{R1x1a\}$   
 $(m1x\ step1\ Ra\ A\ B), (m1a\ step1\ Ra\ A\ B\ Na)$   
 $\{>\ R1x1a\}$   
 $\langle proof \rangle$

**lemma**  $PO\ m1a\ step2\ refines\ m1x\ step2$ :

$\{R1x1a\}$   
 $(m1x\ step2\ Rb\ A\ B), (m1a\ step2\ Rb\ A\ B)$   
 $\{>\ R1x1a\}$   
 $\langle proof \rangle$

**lemma**  $PO\ m1a\ step3\ refines\ m1x\ step3$ :

$\{R1x1a\}$   
 $(m1x\ step3\ Rs\ A\ B\ Kab), (m1a\ step3\ Rs\ A\ B\ Kab\ Na\ nls)$   
 $\{>\ R1x1a\}$   
 $\langle proof \rangle$

**lemma**  $PO\ m1a\ step4\ refines\ m1x\ step4$ :

$\{R1x1a\}$   
 $(m1x\ step4\ Ra\ A\ B\ Kab), (m1a\ step4\ Ra\ A\ B\ Na\ Kab\ nla)$   
 $\{>\ R1x1a\}$   
 $\langle proof \rangle$

**lemma**  $PO\ m1a\ step5\ refines\ m1x\ step5$ :

$\{R1x1a\}$   
 $(m1x\ step5\ A\ B\ Rb\ Kab), (m1a\ step5\ A\ B\ Rb\ Kab\ nlb)$   
 $\{>\ R1x1a\}$   
 $\langle proof \rangle$

**lemma** *PO-m1a-leak-refines-m1x-leak*:

$\{R1x1a\}$   
 $(m1x-leak\ Rs), (m1a-leak\ Rs)$   
 $\{> R1x1a\}$   
 $\langle proof \rangle$

All together now...

**lemmas** *PO-m1a-trans-refines-m1x-trans* =

*PO-m1a-step1-refines-m1x-step1 PO-m1a-step2-refines-m1x-step2*  
*PO-m1a-step3-refines-m1x-step3 PO-m1a-step4-refines-m1x-step4*  
*PO-m1a-step5-refines-m1x-step5 PO-m1a-leak-refines-m1x-leak*

**lemma** *PO-m1a-refines-init-m1x* [iff]:

$init\ m1a \subseteq R1x1a \text{“}(init\ m1x)$   
 $\langle proof \rangle$

**lemma** *PO-m1a-refines-trans-m1x* [iff]:

$\{R1x1a\}$   
 $(trans\ m1x), (trans\ m1a)$   
 $\{> R1x1a\}$   
 $\langle proof \rangle$

Observation consistency.

**lemma** *obs-consistent-med1x1a* [iff]:

$obs-consistent\ R1x1a\ med1x1a\ m1x\ m1a$   
 $\langle proof \rangle$

Refinement result.

**lemma** *PO-m1a-refines-m1x* [iff]:

$refines\ R1x1a\ med1x1a\ m1x\ m1a$   
 $\langle proof \rangle$

**lemma** *m1a-implements-m1x* [iff]:  $implements\ med1x1a\ m1x\ m1a$

$\langle proof \rangle$

By transitivity:

**lemma** *m1a-implements-s0g* [iff]:  $implements\ (med01x\ o\ med1x1a)\ s0g\ m1a$

$\langle proof \rangle$

**inv (inherited): Secrecy**

Secrecy preserved from *m1x*.

**lemma** *knC-runs1x1a* [simp]:  $knC\ (runs1x1a\ runz) = knC\ runz$

$\langle proof \rangle$

**lemma** *PO-m1a-obs-secrecy* [iff]:  $oreach\ m1a \subseteq m1x-secrecy$

$\langle proof \rangle$

**lemma** *PO-m1a-secrecy* [iff]:  $reach\ m1a \subseteq m1x-secrecy$

$\langle proof \rangle$

### 3.2.6 Refinement of $a0i$ for initiator/server

For the initiator, we get an injective agreement with the server on the session key, the responder name, the initiator's nonce and the list of freshness values  $isl$ .

#### Simulation relation

We define two auxiliary functions to reconstruct the signals of the initial model from completed initiator and server runs.

#### type-synonym

$issig = key \times agent \times nonce \times atom\ list$

#### fun

$is-runs2sigs :: runs\ t \Rightarrow issig\ signal \Rightarrow nat$

#### where

$is-runs2sigs\ runz\ (Running\ [A,\ Sv]\ (Kab,\ B,\ Na,\ nl)) =$   
 $(if\ \exists\ Rs.\ Kab = sesK\ (Rs\$sk) \wedge$   
 $runz\ Rs = Some\ (Serv,\ [A,\ B],\ aNon\ Na\ \# \ nl)$   
 $then\ 1\ else\ 0)$

|  $is-runs2sigs\ runz\ (Commit\ [A,\ Sv]\ (Kab,\ B,\ Na,\ nl)) =$   
 $(if\ \exists\ Ra\ nla.\ Na = Ra\$na \wedge$   
 $runz\ Ra = Some\ (Init,\ [A,\ B],\ aKey\ Kab\ \# \ nla) \wedge$   
 $take\ is-len\ nla = nl$   
 $then\ 1\ else\ 0)$

|  $is-runs2sigs\ runz\ - = 0$

Simulation relation and mediator function. We map completed initiator and responder runs to commit and running signals, respectively.

#### definition

$med-a0m1a-is :: m1a-obs \Rightarrow issig\ a0i-obs\ \mathbf{where}$   
 $med-a0m1a-is\ o1 \equiv \{\ signals = is-runs2sigs\ (runs\ o1),\ corrupted = \{\} \}$

#### definition

$R-a0m1a-is :: (issig\ a0i-state \times m1a-state)\ set\ \mathbf{where}$   
 $R-a0m1a-is \equiv \{(s,\ t).\ signals\ s = is-runs2sigs\ (runs\ t) \wedge\ corrupted\ s = \{\} \}$

**lemmas**  $R-a0m1a-is-defs = R-a0m1a-is-def\ med-a0m1a-is-def$

#### Lemmas about the auxiliary functions

**lemma**  $is-runs2sigs-empty\ [simp]:$

$runz = Map.empty \Longrightarrow is-runs2sigs\ runz = (\lambda s.\ 0)$   
 $\langle proof \rangle$

Update lemmas

**lemma**  $is-runs2sigs-upd-init-none\ [simp]:$

$\llbracket Ra \notin dom\ runz \rrbracket$   
 $\Longrightarrow is-runs2sigs\ (runz(Ra \mapsto (Init,\ [A,\ B],\ []))) = is-runs2sigs\ runz$   
 $\langle proof \rangle$

**lemma** *is-runs2sigs-upd-resp-none* [simp]:  
 $\llbracket Rb \notin \text{dom runz} \rrbracket$   
 $\implies \text{is-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], []))) = \text{is-runs2sigs runz}$   
 ⟨proof⟩

**lemma** *is-runs2sigs-upd-serv* [simp]:  
 $\llbracket Rs \notin \text{dom runz} \rrbracket$   
 $\implies \text{is-runs2sigs} (\text{runz}(Rs \mapsto (\text{Serv}, [A, B], \text{aNon } Na \# \text{ ils}))) =$   
 $(\text{is-runs2sigs runz})(\text{Running } [A, Sv] (\text{sesK } (Rs\$sk), B, Na, \text{ ils}) := 1)$   
 ⟨proof⟩

**lemma** *is-runs2sigs-upd-init-some* [simp]:  
 $\llbracket \text{runz } Ra = \text{Some } (\text{Init}, [A, B], []); \text{ ils} = \text{take is-len nla} \rrbracket$   
 $\implies \text{is-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], \text{aKey } Kab \# \text{ nla}))) =$   
 $(\text{is-runs2sigs runz})(\text{Commit } [A, Sv] (Kab, B, Ra\$na, \text{ ils}) := 1)$   
 ⟨proof⟩

**lemma** *is-runs2sigs-upd-resp-some* [simp]:  
 $\llbracket \text{runz } Rb = \text{Some } (\text{Resp}, [A, B], []) \rrbracket$   
 $\implies \text{is-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], \text{aKey } Kab \# \text{ nlb}))) =$   
 $\text{is-runs2sigs runz}$   
 ⟨proof⟩

## Refinement proof

**lemma** *PO-m1a-step1-refines-a0-is-skip*:  
 $\{R\text{-a0m1a-is}\}$   
 $\text{Id}, (m1a\text{-step1 } Ra \ A \ B \ Na)$   
 $\{> R\text{-a0m1a-is}\}$   
 ⟨proof⟩

**lemma** *PO-m1a-step2-refines-a0-is-skip*:  
 $\{R\text{-a0m1a-is}\}$   
 $\text{Id}, (m1a\text{-step2 } Rb \ A \ B)$   
 $\{> R\text{-a0m1a-is}\}$   
 ⟨proof⟩

**lemma** *PO-m1a-step3-refines-a0-is-running*:  
 $\{R\text{-a0m1a-is}\}$   
 $(a0i\text{-running } [A, Sv] (Kab, B, Na, nls)),$   
 $(m1a\text{-step3 } Rs \ A \ B \ Kab \ Na \ nls)$   
 $\{> R\text{-a0m1a-is}\}$   
 ⟨proof⟩

**lemma** *PO-m1a-step4-refines-a0-is-commit*:  
 $\{R\text{-a0m1a-is} \cap \text{UNIV} \times m1a\text{-inv0-fin}\}$   
 $(a0i\text{-commit } [A, Sv] (Kab, B, Na, \text{take is-len nla})),$   
 $(m1a\text{-step4 } Ra \ A \ B \ Na \ Kab \ nla)$   
 $\{> R\text{-a0m1a-is}\}$   
 ⟨proof⟩

**lemma** *PO-m1a-step5-refines-a0-is-skip*:

$\{R\text{-}a0m1a\text{-}is\}$   
 $Id, (m1a\text{-}step5\ A\ B\ Rb\ Kab\ nlb)$   
 $\{>\ R\text{-}a0m1a\text{-}is\}$   
 $\langle proof \rangle$

**lemma** *PO-m1a-leak-refines-a0-is-skip*:  
 $\{R\text{-}a0m1a\text{-}is\}$   
 $Id, (m1a\text{-}leak\ Rs)$   
 $\{>\ R\text{-}a0m1a\text{-}is\}$   
 $\langle proof \rangle$

All together now...

**lemmas** *PO-m1a-trans-refines-a0-is-trans* =  
 $PO\text{-}m1a\text{-}step1\text{-}refines\text{-}a0\text{-}is\text{-}skip\ PO\text{-}m1a\text{-}step2\text{-}refines\text{-}a0\text{-}is\text{-}skip$   
 $PO\text{-}m1a\text{-}step3\text{-}refines\text{-}a0\text{-}is\text{-}running\ PO\text{-}m1a\text{-}step4\text{-}refines\text{-}a0\text{-}is\text{-}commit$   
 $PO\text{-}m1a\text{-}step5\text{-}refines\text{-}a0\text{-}is\text{-}skip\ PO\text{-}m1a\text{-}leak\text{-}refines\text{-}a0\text{-}is\text{-}skip$

**lemma** *PO-m1a-refines-init-a0-is* [iff]:  
 $init\ m1a \subseteq R\text{-}a0m1a\text{-}is''(init\ a0i)$   
 $\langle proof \rangle$

**lemma** *PO-m1a-refines-trans-a0-is* [iff]:  
 $\{R\text{-}a0m1a\text{-}is \cap a0i\text{-}inv1\text{-}iagree \times m1a\text{-}inv0\text{-}fin\}$   
 $(trans\ a0i), (trans\ m1a)$   
 $\{>\ R\text{-}a0m1a\text{-}is\}$   
 $\langle proof \rangle$

**lemma** *obs-consistent-med-a0m1a-is* [iff]:  
 $obs\text{-}consistent\ R\text{-}a0m1a\text{-}is\ med\text{-}a0m1a\text{-}is\ a0i\ m1a$   
 $\langle proof \rangle$

Refinement result.

**lemma** *PO-m1a-refines-a0-is* [iff]:  
 $refines\ (R\text{-}a0m1a\text{-}is \cap a0i\text{-}inv1\text{-}iagree \times m1a\text{-}inv0\text{-}fin)\ med\text{-}a0m1a\text{-}is\ a0i\ m1a$   
 $\langle proof \rangle$

**lemma** *m1a-implements-a0-is: implements med-a0m1a-is a0i m1a*  
 $\langle proof \rangle$

## inv2i (inherited): Initiator and server

This is a translation of the agreement property to Level 1. It follows from the refinement and is needed to prove inv1.

### definition

$m1a\text{-}inv2i\text{-}serv :: 'x\ m1x\text{-}state\text{-}scheme\ set$

where

$m1a\text{-}inv2i\text{-}serv \equiv \{s. \forall A\ B\ Ra\ Kab\ nla.$

$A \notin bad \longrightarrow$

$runs\ s\ Ra = Some\ (Init, [A, B], aKey\ Kab\ \# \ nla) \longrightarrow$

$(\exists Rs. Kab = sesK\ (Rs\$sk) \wedge$

$runs\ s\ Rs = Some\ (Serv, [A, B], aNon\ (Ra\$na)\ \# \ take\ is\text{-}len\ nla))$

}

**lemmas**  $m1a\text{-}inv2i\text{-}servI = m1a\text{-}inv2i\text{-}serv\text{-}def [THEN setc\text{-}def\text{-}to\text{-}intro, rule\text{-}format]$

**lemmas**  $m1a\text{-}inv2i\text{-}servE = m1a\text{-}inv2i\text{-}serv\text{-}def [THEN setc\text{-}def\text{-}to\text{-}elim, rule\text{-}format]$

**lemmas**  $m1a\text{-}inv2i\text{-}servD = m1a\text{-}inv2i\text{-}serv\text{-}def [THEN setc\text{-}def\text{-}to\text{-}dest, rule\text{-}format, rotated -1]$

Invariance proof, see below after init/serv authentication proof.

**lemma**  $PO\text{-}m1a\text{-}inv2i\text{-}serv [iff]:$

$reach\ m1a \subseteq m1a\text{-}inv2i\text{-}serv$

$\langle proof \rangle$

### inv1: Key freshness for initiator

The initiator obtains key freshness from the injective agreement with the server AND the fact that there is only one server run with a given key.

#### definition

$m1a\text{-}inv1\text{-}ifresh :: 'x\ m1a\text{-}pred$

#### where

$m1a\text{-}inv1\text{-}ifresh \equiv \{s. \forall A\ A'\ B\ B'\ Ra\ Ra'\ Kab\ nl\ nl'. \}$

$runs\ s\ Ra = Some\ (Init, [A, B], aKey\ Kab\ \# nl) \longrightarrow$

$runs\ s\ Ra' = Some\ (Init, [A', B'], aKey\ Kab\ \# nl') \longrightarrow$

$A \notin bad \longrightarrow B \notin bad \longrightarrow Kab \notin leak\ s \longrightarrow$

$Ra = Ra'$

}

**lemmas**  $m1a\text{-}inv1\text{-}ifreshI = m1a\text{-}inv1\text{-}ifresh\text{-}def [THEN setc\text{-}def\text{-}to\text{-}intro, rule\text{-}format]$

**lemmas**  $m1a\text{-}inv1\text{-}ifreshE [elim] = m1a\text{-}inv1\text{-}ifresh\text{-}def [THEN setc\text{-}def\text{-}to\text{-}elim, rule\text{-}format]$

**lemmas**  $m1a\text{-}inv1\text{-}ifreshD = m1a\text{-}inv1\text{-}ifresh\text{-}def [THEN setc\text{-}def\text{-}to\text{-}dest, rule\text{-}format, rotated 1]$

Invariance proof

**lemma**  $PO\text{-}m1a\text{-}inv1\text{-}ifresh\text{-}init [iff]:$

$init\ m1a \subseteq m1a\text{-}inv1\text{-}ifresh$

$\langle proof \rangle$

**lemma**  $PO\text{-}m1a\text{-}inv1\text{-}ifresh\text{-}step4:$

$\{m1a\text{-}inv1\text{-}ifresh \cap m1a\text{-}inv2i\text{-}serv \cap m1x\text{-}secrecy\}$

$m1a\text{-}step4\ Ra\ A\ B\ Na\ Kab\ nla$

$\{>\ m1a\text{-}inv1\text{-}ifresh\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m1a\text{-}inv1\text{-}ifresh\text{-}trans [iff]:$

$\{m1a\text{-}inv1\text{-}ifresh \cap m1a\text{-}inv2i\text{-}serv \cap m1x\text{-}secrecy\}\ trans\ m1a\ \{>\ m1a\text{-}inv1\text{-}ifresh\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m1a\text{-}inv1\text{-}ifresh [iff]: reach\ m1a \subseteq m1a\text{-}inv1\text{-}ifresh$

$\langle proof \rangle$

### 3.2.7 Refinement of $a0n$ for responder/server

For the responder, we get a non-injective agreement with the server on the session key, the initiator's name, and additional data.

#### Simulation relation

We define two auxiliary functions to reconstruct the signals of the initial model from completed responder and server runs.

#### type-synonym

$rssig = key \times agent \times atom\ list$

#### abbreviation

$rs-commit :: [runs-t, agent, agent, key, atom\ list] \Rightarrow rid-t\ set$

#### where

$rs-commit\ runz\ A\ B\ Kab\ rsl \equiv \{Rb.\ \exists\ nlb.\$   
 $\quad runz\ Rb = Some\ (Resp,\ [A,\ B],\ aKey\ Kab\ \# \ nlb) \wedge take\ rs-len\ nlb = rsl$   
 $\}$

#### fun

$rs-runs2sigs :: runs-t \Rightarrow rssig\ signal \Rightarrow nat$

#### where

$rs-runs2sigs\ runz\ (Running\ [B,\ Sv]\ (Kab,\ A,\ rsl)) =$   
 $\quad (if\ (\exists\ Rs\ Na.\ Kab = sesK\ (Rs\$sk) \wedge$   
 $\quad\quad runz\ Rs = Some\ (Serv,\ [A,\ B],\ aNon\ Na\ \# \ rsl))$   
 $\quad then\ 1\ else\ 0)$

$| rs-runs2sigs\ runz\ (Commit\ [B,\ Sv]\ (Kab,\ A,\ rsl)) =$   
 $\quad card\ (rs-commit\ runz\ A\ B\ Kab\ rsl)$

$| rs-runs2sigs\ runz\ - = 0$

Simulation relation and mediator function. We map completed initiator and responder runs to commit and running signals, respectively.

#### definition

$med-a0m1a-rs :: m1a-obs \Rightarrow rssig\ a0n-obs$  **where**  
 $med-a0m1a-rs\ o1 \equiv (\ signals = rs-runs2sigs\ (runs\ o1),\ corrupted = \{\} )$

#### definition

$R-a0m1a-rs :: (rssig\ a0n-state \times m1a-state)\ set$  **where**  
 $R-a0m1a-rs \equiv \{(s,\ t).\ signals\ s = rs-runs2sigs\ (runs\ t) \wedge corrupted\ s = \{\} \}$

**lemmas**  $R-a0m1a-rs-defs = R-a0m1a-rs-def\ med-a0m1a-rs-def$

#### Lemmas about the auxiliary functions

Other lemmas

**lemma**  $rs-runs2sigs-empty\ [simp]:$

$runz = Map.empty \Longrightarrow rs-runs2sigs\ runz = (\lambda s.\ 0)$

$\langle proof \rangle$

**lemma** *rs-commit-finite* [*simp*, *intro*]:  
 $finite (dom\ runz) \implies finite (rs\ commit\ runz\ A\ B\ Kab\ nls)$   
 $\langle proof \rangle$

Update lemmas

**lemma** *rs-runs2sigs-upd-init-none* [*simp*]:  
 $\llbracket Ra \notin dom\ runz \rrbracket$   
 $\implies rs\ runs2sigs (runz(Ra \mapsto (Init, [A, B], []))) = rs\ runs2sigs\ runz$   
 $\langle proof \rangle$

**lemma** *rs-runs2sigs-upd-resp-none* [*simp*]:  
 $\llbracket Rb \notin dom\ runz \rrbracket$   
 $\implies rs\ runs2sigs (runz(Rb \mapsto (Resp, [A, B], []))) = rs\ runs2sigs\ runz$   
 $\langle proof \rangle$

**lemma** *rs-runs2sigs-upd-serv* [*simp*]:  
 $\llbracket Rs \notin dom\ runz \rrbracket$   
 $\implies rs\ runs2sigs (runz(Rs \mapsto (Serv, [A, B], aNon\ Na\ \# \ nls))) =$   
 $(rs\ runs2sigs\ runz)(Running\ [B, Sv] (sesK (Rs\$sk), A, nls) := 1)$   
 $\langle proof \rangle$

**lemma** *rs-runs2sigs-upd-init-some* [*simp*]:  
 $\llbracket runz\ Ra = Some\ (Init, [A, B], []) \rrbracket$   
 $\implies rs\ runs2sigs (runz(Ra \mapsto (Init, [A, B], aKey\ Kab\ \# \ nl))) =$   
 $rs\ runs2sigs\ runz$   
 $\langle proof \rangle$

**lemma** *rs-runs2sigs-upd-resp-some* [*simp*]:  
 $\llbracket runz\ Rb = Some\ (Resp, [A, B], []); finite\ (dom\ runz);$   
 $rsl = take\ rs\ len\ nlb \rrbracket$   
 $\implies rs\ runs2sigs (runz(Rb \mapsto (Resp, [A, B], aKey\ Kab\ \# \ nlb))) =$   
 $(rs\ runs2sigs\ runz)($   
 $Commit\ [B, Sv] (Kab, A, rsl) := Suc (card (rs\ commit\ runz\ A\ B\ Kab\ rsl)))$   
 $\langle proof \rangle$

## Refinement proof

**lemma** *PO-m1a-step1-refines-a0-rs-skip*:  
 $\{R\ a0m1a\ rs\}$   
 $Id, (m1a\ step1\ Ra\ A\ B\ Na)$   
 $\{>\ R\ a0m1a\ rs\}$   
 $\langle proof \rangle$

**lemma** *PO-m1a-step2-refines-a0-rs-skip*:  
 $\{R\ a0m1a\ rs\}$   
 $Id, (m1a\ step2\ Rb\ A\ B)$   
 $\{>\ R\ a0m1a\ rs\}$   
 $\langle proof \rangle$

**lemma** *PO-m1a-step3-refines-a0-rs-running*:  
 $\{R\ a0m1a\ rs\}$   
 $(a0n\ running\ [B, Sv] (Kab, A, nls)),$   
 $(m1a\ step3\ Rs\ A\ B\ Kab\ Na\ nls)$

{>  $R\text{-}a0m1a\text{-}rs$ }  
 ⟨proof⟩

**lemma** *PO-m1a-step4-refines-a0-rs-skip*:

{ $R\text{-}a0m1a\text{-}rs$ }  
 $Id, (m1a\text{-}step4\ Ra\ A\ B\ Na\ Kab\ nla)$   
 {>  $R\text{-}a0m1a\text{-}rs$ }  
 ⟨proof⟩

**lemma** *PO-m1a-step5-refines-a0-rs-commit*:

{ $R\text{-}a0m1a\text{-}rs \cap UNIV \times m1a\text{-}inv0\text{-}fin$ }  
 $(a0n\text{-}commit\ [B, Sv]\ (Kab, A, take\ rs\text{-}len\ nlb)),$   
 $(m1a\text{-}step5\ Rb\ A\ B\ Kab\ nlb)$   
 {>  $R\text{-}a0m1a\text{-}rs$ }  
 ⟨proof⟩

**lemma** *PO-m1a-leak-refines-a0-rs-skip*:

{ $R\text{-}a0m1a\text{-}rs$ }  
 $Id, (m1a\text{-}leak\ Rs)$   
 {>  $R\text{-}a0m1a\text{-}rs$ }  
 ⟨proof⟩

All together now...

**lemmas** *PO-m1a-trans-refines-a0-rs-trans =*

*PO-m1a-step1-refines-a0-rs-skip PO-m1a-step2-refines-a0-rs-skip*  
*PO-m1a-step3-refines-a0-rs-running PO-m1a-step4-refines-a0-rs-skip*  
*PO-m1a-step5-refines-a0-rs-commit PO-m1a-leak-refines-a0-rs-skip*

**lemma** *PO-m1a-refines-init-ra0n [iff]*:

$init\ m1a \subseteq R\text{-}a0m1a\text{-}rs \text{“}(init\ a0n)$   
 ⟨proof⟩

**lemma** *PO-m1a-refines-trans-ra0n [iff]*:

{ $R\text{-}a0m1a\text{-}rs \cap a0n\text{-}inv1\text{-}niagree \times m1a\text{-}inv0\text{-}fin$ }  
 $(trans\ a0n), (trans\ m1a)$   
 {>  $R\text{-}a0m1a\text{-}rs$ }  
 ⟨proof⟩

**lemma** *obs-consistent-med-a0m1a-rs [iff]*:

*obs-consistent*  
 $(R\text{-}a0m1a\text{-}rs \cap a0n\text{-}inv1\text{-}niagree \times m1a\text{-}inv0\text{-}fin)$   
 $med\text{-}a0m1a\text{-}rs\ a0n\ m1a$   
 ⟨proof⟩

Refinement result.

**lemma** *PO-m1a-refines-a0-rs [iff]*:

*refines*  $(R\text{-}a0m1a\text{-}rs \cap a0n\text{-}inv1\text{-}niagree \times m1a\text{-}inv0\text{-}fin)\ med\text{-}a0m1a\text{-}rs\ a0n\ m1a$   
 ⟨proof⟩

**lemma** *m1a-implements-ra0n: implements med-a0m1a-rs a0n m1a*

⟨proof⟩

## inv2r (inherited): Responder and server

This is a translation of the agreement property to Level 1. It follows from the refinement and not needed here but later.

### definition

*m1a-inv2r-serv* :: 'x m1x-state-scheme set

### where

*m1a-inv2r-serv*  $\equiv$  {s.  $\forall A B Rb Kab nlb$ .  
  $B \notin bad \longrightarrow$   
  $runs\ s\ Rb = Some\ (Resp, [A, B], aKey\ Kab\ \# \ nlb) \longrightarrow$   
  $(\exists Rs\ Na. Kab = sesK\ (Rs\$sk) \wedge$   
  $runs\ s\ Rs = Some\ (Serv, [A, B], aNon\ Na\ \# \ take\ rs-len\ nlb))$   
}

**lemmas** *m1a-inv2r-servI* =

*m1a-inv2r-serv-def* [THEN setc-def-to-intro, rule-format]

**lemmas** *m1a-inv2r-servE* [elim] =

*m1a-inv2r-serv-def* [THEN setc-def-to-elim, rule-format]

**lemmas** *m1a-inv2r-servD* =

*m1a-inv2r-serv-def* [THEN setc-def-to-dest, rule-format, rotated -1]

Invariance proof

**lemma** *PO-m1a-inv2r-serv* [iff]:

*reach m1a*  $\subseteq$  *m1a-inv2r-serv*

<proof>

end

## 3.3 Abstract (n/n)-authenticated key transport (L1)

**theory** *m1-keydist-inrn* **imports** *m1-keydist* ../Refinement/a0i-agree

**begin**

We add authentication for the initiator and responder to the basic server-based key transport protocol:

1. the initiator injectively agrees with the server on the key and some additional data
2. the responder non-injectively agrees with the server on the key and some additional data.

The "additional data" is a parameter of this model.

**declare** *option.split* [split]

### 3.3.1 State

The state type remains the same, but in this model we will record nonces and timestamps in the run frame.

**type-synonym** *m1a-state* = *m1x-state*

**type-synonym**  $m1a\text{-obs} = m1x\text{-obs}$

**type-synonym**  $'x\ m1a\text{-pred} = 'x\ m1x\text{-pred}$

**type-synonym**  $'x\ m1a\text{-trans} = 'x\ m1x\text{-trans}$

We need some parameters regarding the list of freshness values stored by the server. These should be defined in further refinements.

**consts**

$is\text{-len} :: nat$  — num of agreeing list elements for initiator-server

$rs\text{-len} :: nat$  — num of agreeing list elements for responder-server

### 3.3.2 Events

**definition** — by  $A$ , refines  $m1x\text{-step1}$

$m1a\text{-step1} :: [rid\text{-}t, agent, agent] \Rightarrow 'x\ m1r\text{-trans}$

**where**

$m1a\text{-step1} \equiv m1x\text{-step1}$

**definition** — by  $B$ , refines  $m1x\text{-step2}$

$m1a\text{-step2} :: [rid\text{-}t, agent, agent] \Rightarrow 'x\ m1r\text{-trans}$

**where**

$m1a\text{-step2} \equiv m1x\text{-step2}$

**definition** — by  $Sv$ , refines  $m1x\text{-step3}$

$m1a\text{-step3} :: [rid\text{-}t, agent, agent, key, atom\ list] \Rightarrow 'x\ m1r\text{-trans}$

**where**

$m1a\text{-step3}\ Rs\ A\ B\ Kab\ al \equiv \{(s, s1).\}$

— guards:

$Rs \notin dom\ (runs\ s) \wedge$  — fresh run id

$Kab = sesK\ (Rs\$sk) \wedge$  — generate session key

— actions:

$s1 = s \langle runs := (runs\ s)(Rs \mapsto (Serv, [A, B], al)) \rangle$

}

**definition** — by  $A$ , refines  $m1x\text{-step4}$

$m1a\text{-step4} :: [rid\text{-}t, agent, agent, key, atom\ list] \Rightarrow 'x\ m1a\text{-trans}$

**where**

$m1a\text{-step4}\ Ra\ A\ B\ Kab\ nla \equiv \{(s, s').\}$

— guards:

$runs\ s\ Ra = Some\ (Init, [A, B], []) \wedge$

$(Kab \notin leak\ s \longrightarrow (Kab, A) \in azC\ (runs\ s)) \wedge$  — authorization guard

— new guard for non-injective agreement with server on  $(Kab, B, isl)$ ,

— where  $isl = take\ is\text{-len}\ nla$

$(A \notin bad \longrightarrow (\exists Rs. Kab = sesK\ (Rs\$sk) \wedge$

$runs\ s\ Rs = Some\ (Serv, [A, B], take\ is\text{-len}\ nla))) \wedge$

— actions:

$s' = s \langle runs := (runs\ s)(Ra \mapsto (Init, [A, B], aKey\ Kab\ \# nla)) \rangle$

}

**definition** — by  $B$ , refines  $m1x\text{-step5}$

$m1a\text{-step5} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}, \text{atom list}] \Rightarrow 'x \text{ m1a-trans}$

**where**

$m1a\text{-step5 } Rb \ A \ B \ Kab \ nlb \equiv \{(s, s1).$

— guards:

$\text{runs } s \ Rb = \text{Some } (\text{Resp}, [A, B], []) \wedge$

$(Kab \notin \text{leak } s \longrightarrow (Kab, B) \in \text{azC } (\text{runs } s)) \wedge$  — authorization guard

— guard for non-injective agreement with server on  $(Kab, A, \text{rsl})$

— where  $\text{rsl} = \text{take } \text{rs-len } \text{nlb}$

$(B \notin \text{bad} \longrightarrow (\exists Rs. Kab = \text{sesK } (Rs\$sk) \wedge$

$\text{runs } s \ Rs = \text{Some } (\text{Serv}, [A, B], \text{take } \text{rs-len } \text{nlb}))) \wedge$

— actions:

$s1 = s \langle \text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], \text{aKey } Kab \ \# \ \text{nlb})) \rangle$

$\}$

**definition** — by attacker, refines  $m1x\text{-leak}$

$m1a\text{-leak} :: \text{rid-}t \Rightarrow 'x \text{ m1x-trans}$

**where**

$m1a\text{-leak} = m1x\text{-leak}$

### 3.3.3 Specification

**definition**

$m1a\text{-init} :: m1a\text{-state set}$

**where**

$m1a\text{-init} \equiv m1x\text{-init}$

**definition**

$m1a\text{-trans} :: 'x \text{ m1a-trans}$  **where**

$m1a\text{-trans} \equiv (\bigcup A \ B \ Ra \ Rb \ Rs \ Kab \ nls \ nla \ nlb.$

$m1a\text{-step1 } Ra \ A \ B \cup$

$m1a\text{-step2 } Rb \ A \ B \cup$

$m1a\text{-step3 } Rs \ A \ B \ Kab \ nls \cup$

$m1a\text{-step4 } Ra \ A \ B \ Kab \ nla \cup$

$m1a\text{-step5 } Rb \ A \ B \ Kab \ nlb \cup$

$m1a\text{-leak } Rs \cup$

$Id$

$)$

**definition**

$m1a :: (m1a\text{-state}, m1a\text{-obs}) \text{ spec}$  **where**

$m1a \equiv \langle$

$\text{init} = m1a\text{-init},$

$\text{trans} = m1a\text{-trans},$

$\text{obs} = \text{id}$

$\rangle$

**lemma**  $\text{init-}m1a: \text{init } m1a = m1a\text{-init}$

$\langle \text{proof} \rangle$

**lemma**  $\text{trans-}m1a: \text{trans } m1a = m1a\text{-trans}$

$\langle \text{proof} \rangle$

**lemma** *obs-m1a* [*simp*]: *obs m1a = id*  
 ⟨*proof*⟩

**lemmas** *m1a-loc-defs* =  
*m1a-def m1a-init-def m1a-trans-def*  
*m1a-step1-def m1a-step2-def m1a-step3-def m1a-step4-def m1a-step5-def*  
*m1a-leak-def*

**lemmas** *m1a-defs* = *m1a-loc-defs m1x-defs*

### 3.3.4 Invariants

#### inv0: Finite domain

There are only finitely many runs. This is needed to establish the responder/initiator agreement.

**definition**

*m1a-inv0-fin* :: 'x *m1r-pred*

**where**

*m1a-inv0-fin* ≡ {*s*. *finite* (*dom* (*runs s*))}

**lemmas** *m1a-inv0-finI* = *m1a-inv0-fin-def* [*THEN setc-def-to-intro, rule-format*]

**lemmas** *m1a-inv0-finE* [*elim*] = *m1a-inv0-fin-def* [*THEN setc-def-to-elim, rule-format*]

**lemmas** *m1a-inv0-finD* = *m1a-inv0-fin-def* [*THEN setc-def-to-dest, rule-format*]

Invariance proof.

**lemma** *PO-m1a-inv0-fin-init* [*iff*]:

*init m1a* ⊆ *m1a-inv0-fin*

⟨*proof*⟩

**lemma** *PO-m1a-inv0-fin-trans* [*iff*]:

{*m1a-inv0-fin*} *trans m1a* {> *m1a-inv0-fin*}

⟨*proof*⟩

**lemma** *PO-m1a-inv0-fin* [*iff*]: *reach m1a* ⊆ *m1a-inv0-fin*

⟨*proof*⟩

### 3.3.5 Refinement of *m1x*

#### Simulation relation

Define run abstraction.

**fun**

*rm1x1a* :: *role-t* ⇒ *atom list* ⇒ *atom list*

**where**

*rm1x1a Init* = *take 1* — take *Kab* from *Kab # nla*  
 | *rm1x1a Resp* = *take 1* — take *Kab* from *Kab # nlb*  
 | *rm1x1a Serv* = *take 0* — drop all from *nls*

**abbreviation**

*runs1x1a* :: *runs-t* ⇒ *runs-t* **where**

$runs1x1a \equiv map-runs\ rm1x1a$

med1x1: The mediator function maps a concrete observation to an abstract one.

**definition**

$med1x1a :: m1a-obs \Rightarrow m1x-obs$  **where**  
 $med1x1a\ t \equiv (\mid runs = runs1x1a\ (runs\ t), leak = leak\ t \mid)$

R1x1a: The simulation relation is defined in terms of the mediator function.

**definition**

$R1x1a :: (m1x-state \times m1a-state)$  *set* **where**  
 $R1x1a \equiv \{(s, t). s = med1x1a\ t\}$

**lemmas**  $R1x1a-defs =$

$R1x1a-def\ med1x1a-def$

**Refinement proof**

**lemma**  $PO-m1a-step1-refines-m1x-step1:$

$\{R1x1a\}$   
 $(m1x-step1\ Ra\ A\ B), (m1a-step1\ Ra\ A\ B)$   
 $\{>\ R1x1a\}$   
 $\langle proof \rangle$

**lemma**  $PO-m1a-step2-refines-m1x-step2:$

$\{R1x1a\}$   
 $(m1x-step2\ Rb\ A\ B), (m1a-step2\ Rb\ A\ B)$   
 $\{>\ R1x1a\}$   
 $\langle proof \rangle$

**lemma**  $PO-m1a-step3-refines-m1x-step3:$

$\{R1x1a\}$   
 $(m1x-step3\ Rs\ A\ B\ Kab), (m1a-step3\ Rs\ A\ B\ Kab\ nls)$   
 $\{>\ R1x1a\}$   
 $\langle proof \rangle$

**lemma**  $PO-m1a-step4-refines-m1x-step4:$

$\{R1x1a\}$   
 $(m1x-step4\ Ra\ A\ B\ Kab), (m1a-step4\ Ra\ A\ B\ Kab\ nla)$   
 $\{>\ R1x1a\}$   
 $\langle proof \rangle$

**lemma**  $PO-m1a-step5-refines-m1x-step5:$

$\{R1x1a\}$   
 $(m1x-step5\ Rb\ A\ B\ Kab), (m1a-step5\ Rb\ A\ B\ Kab\ nlb)$   
 $\{>\ R1x1a\}$   
 $\langle proof \rangle$

**lemma**  $PO-m1a-leak-refines-m1x-leak:$

$\{R1x1a\}$   
 $(m1x-leak\ Rs), (m1a-leak\ Rs)$   
 $\{>\ R1x1a\}$   
 $\langle proof \rangle$

All together now...

**lemmas** *PO-m1a-trans-refines-m1x-trans* =  
*PO-m1a-step1-refines-m1x-step1 PO-m1a-step2-refines-m1x-step2*  
*PO-m1a-step3-refines-m1x-step3 PO-m1a-step4-refines-m1x-step4*  
*PO-m1a-step5-refines-m1x-step5 PO-m1a-leak-refines-m1x-leak*

**lemma** *PO-m1a-refines-init-m1x* [iff]:  
 $init\ m1a \subseteq R1x1a''(init\ m1x)$   
 ⟨proof⟩

**lemma** *PO-m1a-refines-trans-m1x* [iff]:  
 $\{R1x1a\}$   
 $(trans\ m1x), (trans\ m1a)$   
 $\{>\ R1x1a\}$   
 ⟨proof⟩

Observation consistency.

**lemma** *obs-consistent-med1x1a* [iff]:  
 $obs-consistent\ R1x1a\ med1x1a\ m1x\ m1a$   
 ⟨proof⟩

Refinement result.

**lemma** *PO-m1a-refines-m1x* [iff]:  
 $refines\ R1x1a\ med1x1a\ m1x\ m1a$   
 ⟨proof⟩

**lemma** *m1a-implements-m1x* [iff]: *implements med1x1a m1x m1a*  
 ⟨proof⟩

### 3.3.6 Refinement of *a0n* for initiator/server

For the initiator, we get an non-injective agreement with the server on the session key, the responder name, and the atom list *isl*.

#### Simulation relation

We define two auxiliary functions to reconstruct the signals of the initial model from completed initiator and server runs.

#### type-synonym

$issig = key \times agent \times atom\ list$

#### abbreviation

$is-commit :: [runs-t, agent, agent, key, atom\ list] \Rightarrow rid-t\ set$

#### where

$is-commit\ runz\ A\ B\ Kab\ sl \equiv \{Ra. \exists nla.$   
 $runz\ Ra = Some\ (Init, [A, B], aKey\ Kab\ \# nla) \wedge take\ is-len\ nla = sl$   
 $\}$

#### fun

$is\text{-runs2sigs} :: runs\text{-}t \Rightarrow issig\ signal \Rightarrow nat$

**where**

$is\text{-runs2sigs}\ runz\ (Running\ [A,\ Sv]\ (Kab,\ B,\ sl)) =$   
 $(if\ \exists\ Rs\ nls.\ Kab = sesK\ (Rs\$sk) \wedge$   
 $runz\ Rs = Some\ (Serv,\ [A,\ B],\ nls) \wedge take\ is\text{-}len\ nls = sl$   
 $then\ 1\ else\ 0)$

$| is\text{-runs2sigs}\ runz\ (Commit\ [A,\ Sv]\ (Kab,\ B,\ sl)) =$   
 $card\ (is\text{-}commit\ runz\ A\ B\ Kab\ sl)$

$| is\text{-runs2sigs}\ runz\ - = 0$

Simulation relation and mediator function. We map completed initiator and responder runs to commit and running signals, respectively.

**definition**

$med\text{-}a0m1a\text{-}is :: m1a\text{-}obs \Rightarrow issig\ a0i\text{-}obs$  **where**  
 $med\text{-}a0m1a\text{-}is\ o1 \equiv (\ signals = is\text{-}runs2sigs\ (runs\ o1),\ corrupted = \{\} )$

**definition**

$R\text{-}a0m1a\text{-}is :: (issig\ a0i\text{-}state \times m1a\text{-}state)$  **set** **where**  
 $R\text{-}a0m1a\text{-}is \equiv \{(s,\ t).\ signals\ s = is\text{-}runs2sigs\ (runs\ t) \wedge corrupted\ s = \{\} \}$

**lemmas**  $R\text{-}a0m1a\text{-}is\text{-}defs = R\text{-}a0m1a\text{-}is\text{-}def\ med\text{-}a0m1a\text{-}is\text{-}def$

## Lemmas about the auxiliary functions

**lemma**  $is\text{-runs2sigs}\text{-}empty$   $[simp]$ :

$runz = Map.empty \implies is\text{-runs2sigs}\ runz = (\lambda s.\ 0)$   
 $\langle proof \rangle$

**lemma**  $is\text{-}commit\text{-}finite$   $[simp,\ intro]$ :

$finite\ (dom\ runz) \implies finite\ (is\text{-}commit\ runz\ A\ B\ Kab\ nls)$   
 $\langle proof \rangle$

Update lemmas

**lemma**  $is\text{-runs2sigs}\text{-}upd\text{-}init\text{-}none$   $[simp]$ :

$\llbracket Ra \notin dom\ runz \rrbracket$   
 $\implies is\text{-runs2sigs}\ (runz(Ra \mapsto (Init,\ [A,\ B],\ []))) = is\text{-runs2sigs}\ runz$   
 $\langle proof \rangle$

**lemma**  $is\text{-runs2sigs}\text{-}upd\text{-}resp\text{-}none$   $[simp]$ :

$\llbracket Rb \notin dom\ runz \rrbracket$   
 $\implies is\text{-runs2sigs}\ (runz(Rb \mapsto (Resp,\ [A,\ B],\ []))) = is\text{-runs2sigs}\ runz$   
 $\langle proof \rangle$

**lemma**  $is\text{-runs2sigs}\text{-}upd\text{-}serv$   $[simp]$ :

$\llbracket Rs \notin dom\ runz \rrbracket$   
 $\implies is\text{-runs2sigs}\ (runz(Rs \mapsto (Serv,\ [A,\ B],\ nls))) =$   
 $(is\text{-runs2sigs}\ runz)(Running\ [A,\ Sv]\ (sesK\ (Rs\$sk),\ B,\ take\ is\text{-}len\ nls) := 1)$   
 $\langle proof \rangle$

**lemma**  $is\text{-runs2sigs}\text{-}upd\text{-}init\text{-}some$   $[simp]$ :

$$\llbracket \text{runz } Ra = \text{Some } (\text{Init}, [A, B], []) ; \text{finite } (\text{dom } \text{runz}) ;$$

$$\text{ils} = \text{take } \text{is-len } \text{nla} \rrbracket$$

$$\implies \text{is-runs2sigs } (\text{runz}(Ra \mapsto (\text{Init}, [A, B], \text{aKey } Kab \# \text{nla}))) =$$

$$(\text{is-runs2sigs } \text{runz})($$

$$\text{Commit } [A, Sv] (Kab, B, \text{ils}) :=$$

$$\text{Suc } (\text{card } (\text{is-commit } \text{runz } A \ B \ Kab \ \text{ils})))$$

$$\langle \text{proof} \rangle$$

**lemma** *is-runs2sigs-upd-resp-some [simp]:*  

$$\llbracket \text{runz } Rb = \text{Some } (\text{Resp}, [A, B], []) \rrbracket$$

$$\implies \text{is-runs2sigs } (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], \text{aKey } Kab \# \text{nlb}))) =$$

$$\text{is-runs2sigs } \text{runz}$$

$$\langle \text{proof} \rangle$$

## Refinement proof

**lemma** *PO-m1a-step1-refines-a0-is-skip:*  

$$\{R\text{-a0m1a-is}\}$$

$$\text{Id}, (\text{m1a-step1 } Ra \ A \ B)$$

$$\{> R\text{-a0m1a-is}\}$$

$$\langle \text{proof} \rangle$$

**lemma** *PO-m1a-step2-refines-a0-is-skip:*  

$$\{R\text{-a0m1a-is}\}$$

$$\text{Id}, (\text{m1a-step2 } Rb \ A \ B)$$

$$\{> R\text{-a0m1a-is}\}$$

$$\langle \text{proof} \rangle$$

**lemma** *PO-m1a-step3-refines-a0-is-running:*  

$$\{R\text{-a0m1a-is}\}$$

$$(\text{a0n-running } [A, Sv] (Kab, B, \text{take } \text{is-len } \text{nls})),$$

$$(\text{m1a-step3 } Rs \ A \ B \ Kab \ \text{nls})$$

$$\{> R\text{-a0m1a-is}\}$$

$$\langle \text{proof} \rangle$$

**lemma** *PO-m1a-step4-refines-a0-is-commit:*  

$$\{R\text{-a0m1a-is} \cap \text{UNIV} \times \text{m1a-inv0-fin}\}$$

$$(\text{a0n-commit } [A, Sv] (Kab, B, \text{take } \text{is-len } \text{nla})),$$

$$(\text{m1a-step4 } Ra \ A \ B \ Kab \ \text{nla})$$

$$\{> R\text{-a0m1a-is}\}$$

$$\langle \text{proof} \rangle$$

**lemma** *PO-m1a-step5-refines-a0-is-skip:*  

$$\{R\text{-a0m1a-is}\}$$

$$\text{Id}, (\text{m1a-step5 } Rb \ A \ B \ Kab \ \text{nlb})$$

$$\{> R\text{-a0m1a-is}\}$$

$$\langle \text{proof} \rangle$$

**lemma** *PO-m1a-leak-refines-a0-is-skip:*  

$$\{R\text{-a0m1a-is}\}$$

$$\text{Id}, (\text{m1a-leak } Rs)$$

$$\{> R\text{-a0m1a-is}\}$$

$$\langle \text{proof} \rangle$$

All together now...

**lemmas**  $PO\text{-}m1a\text{-}trans\text{-}refines\text{-}a0\text{-}is\text{-}trans =$   
 $PO\text{-}m1a\text{-}step1\text{-}refines\text{-}a0\text{-}is\text{-}skip$   $PO\text{-}m1a\text{-}step2\text{-}refines\text{-}a0\text{-}is\text{-}skip$   
 $PO\text{-}m1a\text{-}step3\text{-}refines\text{-}a0\text{-}is\text{-}running$   $PO\text{-}m1a\text{-}step4\text{-}refines\text{-}a0\text{-}is\text{-}commit$   
 $PO\text{-}m1a\text{-}step5\text{-}refines\text{-}a0\text{-}is\text{-}skip$   $PO\text{-}m1a\text{-}leak\text{-}refines\text{-}a0\text{-}is\text{-}skip$

**lemma**  $PO\text{-}m1a\text{-}refines\text{-}init\text{-}a0\text{-}is$  [iff]:  
 $init\ m1a \subseteq R\text{-}a0m1a\text{-}is''(init\ a0n)$   
 ⟨proof⟩

**lemma**  $PO\text{-}m1a\text{-}refines\text{-}trans\text{-}a0\text{-}is$  [iff]:  
 $\{R\text{-}a0m1a\text{-}is \cap UNIV \times m1a\text{-}inv0\text{-}fin\}$   
 $(trans\ a0n), (trans\ m1a)$   
 $\{> R\text{-}a0m1a\text{-}is\}$   
 ⟨proof⟩

**lemma**  $obs\text{-}consistent\text{-}med\text{-}a0m1a\text{-}is$  [iff]:  
 $obs\text{-}consistent\ R\text{-}a0m1a\text{-}is\ med\text{-}a0m1a\text{-}is\ a0n\ m1a$   
 ⟨proof⟩

Refinement result.

**lemma**  $PO\text{-}m1a\text{-}refines\text{-}a0\text{-}is$  [iff]:  
 $refines\ (R\text{-}a0m1a\text{-}is \cap UNIV \times m1a\text{-}inv0\text{-}fin)\ med\text{-}a0m1a\text{-}is\ a0n\ m1a$   
 ⟨proof⟩

**lemma**  $m1a\text{-}implements\text{-}a0\text{-}is$ :  $implements\ med\text{-}a0m1a\text{-}is\ a0n\ m1a$   
 ⟨proof⟩

### 3.3.7 Refinement of $a0n$ for responder/server

For the responder, we get a non-injective agreement with the server on the session key, the initiator's name, and additional data.

#### Simulation relation

We define two auxiliary functions to reconstruct the signals of the initial model from completed responder and server runs.

#### type-synonym

$rssig = key \times agent \times atom\ list$

#### abbreviation

$rs\text{-}commit :: [runs\text{-}t, agent, agent, key, atom\ list] \Rightarrow rid\text{-}t\ set$

#### where

$rs\text{-}commit\ runz\ A\ B\ Kab\ rsl \equiv \{Rb. \exists nlb.$

$runz\ Rb = Some\ (Resp, [A, B], aKey\ Kab\ \# nlb) \wedge take\ rs\text{-}len\ nlb = rsl$

$\}$

#### fun

$rs\text{-}runs2sigs :: runs\text{-}t \Rightarrow rssig\ signal \Rightarrow nat$

#### where

$rs\text{-runs2sigs runz (Running [B, Sv] (Kab, A, rsl)) =$   
 $(if \exists Rs nls. Kab = sesK (Rs\$sk) \wedge$   
 $runz Rs = Some (Serv, [A, B], nls) \wedge take\ rs\text{-len}\ nls = rsl$   
 $then\ 1\ else\ 0)$

$| rs\text{-runs2sigs runz (Commit [B, Sv] (Kab, A, rsl)) =$   
 $card (rs\text{-commit runz A B Kab rsl)}$

$| rs\text{-runs2sigs runz - = 0$

Simulation relation and mediator function. We map completed initiator and responder runs to commit and running signals, respectively.

**definition**

$med\text{-}a0m1a\text{-}rs :: m1a\text{-}obs \Rightarrow rssid\ a0n\text{-}obs$  **where**  
 $med\text{-}a0m1a\text{-}rs\ o1 \equiv (\ signals = rs\text{-runs2sigs (runs\ o1), corrupted = \{\} )$

**definition**

$R\text{-}a0m1a\text{-}rs :: (rssid\ a0n\text{-}state \times m1a\text{-}state)$  **set** **where**  
 $R\text{-}a0m1a\text{-}rs \equiv \{(s, t). signals\ s = rs\text{-runs2sigs (runs\ t) \wedge corrupted\ s = \{\} \}$

**lemmas**  $R\text{-}a0m1a\text{-}rs\text{-}defs = R\text{-}a0m1a\text{-}rs\text{-}def\ med\text{-}a0m1a\text{-}rs\text{-}def$

## Lemmas about the auxiliary functions

Other lemmas

**lemma**  $rs\text{-runs2sigs}\text{-}empty$  [simp]:

$runz = Map.empty \Longrightarrow rs\text{-runs2sigs runz = (\lambda s. 0)$

$\langle proof \rangle$

**lemma**  $rs\text{-commit}\text{-}finite$  [simp, intro]:

$finite (dom runz) \Longrightarrow finite (rs\text{-commit runz A B Kab nls)$

$\langle proof \rangle$

Update lemmas

**lemma**  $rs\text{-runs2sigs}\text{-}upd\text{-}init\text{-}none$  [simp]:

$\llbracket Ra \notin dom runz \rrbracket$

$\Longrightarrow rs\text{-runs2sigs (runz(Ra \mapsto (Init, [A, B], []))) = rs\text{-runs2sigs runz}$

$\langle proof \rangle$

**lemma**  $rs\text{-runs2sigs}\text{-}upd\text{-}resp\text{-}none$  [simp]:

$\llbracket Rb \notin dom runz \rrbracket$

$\Longrightarrow rs\text{-runs2sigs (runz(Rb \mapsto (Resp, [A, B], []))) = rs\text{-runs2sigs runz}$

$\langle proof \rangle$

**lemma**  $rs\text{-runs2sigs}\text{-}upd\text{-}serv$  [simp]:

$\llbracket Rs \notin dom runz \rrbracket$

$\Longrightarrow rs\text{-runs2sigs (runz(Rs \mapsto (Serv, [A, B], nls))) =$

$(rs\text{-runs2sigs runz)(Running [B, Sv] (sesK (Rs\$sk), A, take\ rs\text{-len}\ nls) := 1)$

$\langle proof \rangle$

**lemma**  $rs\text{-runs2sigs}\text{-}upd\text{-}init\text{-}some$  [simp]:

$\llbracket runz Ra = Some (Init, [A, B], []) \rrbracket$

$\implies rs\text{-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], aKey\ Kab \# nl))) =$   
 $rs\text{-runs2sigs} \text{runz}$   
 <proof>

**lemma** *rs-runs2sigs-upd-resp-some [simp]:*  
 $\llbracket \text{runz } Rb = \text{Some } (Resp, [A, B], []); \text{finite } (\text{dom } \text{runz});$   
 $\text{rsl} = \text{take } rs\text{-len } nlb \rrbracket$   
 $\implies rs\text{-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], aKey\ Kab \# nlb))) =$   
 $(rs\text{-runs2sigs } \text{runz})($   
 $\text{Commit } [B, Sv] (Kab, A, rsl) := \text{Suc } (\text{card } (rs\text{-commit } \text{runz } A\ B\ Kab\ rsl)))$   
 <proof>

## Refinement proof

**lemma** *PO-m1a-step1-refines-a0-rs-skip:*  
 $\{R\text{-a0m1a-rs}\}$   
 $\text{Id}, (m1a\text{-step1 } Ra\ A\ B)$   
 $\{> R\text{-a0m1a-rs}\}$   
 <proof>

**lemma** *PO-m1a-step2-refines-a0-rs-skip:*  
 $\{R\text{-a0m1a-rs}\}$   
 $\text{Id}, (m1a\text{-step2 } Rb\ A\ B)$   
 $\{> R\text{-a0m1a-rs}\}$   
 <proof>

**lemma** *PO-m1a-step3-refines-a0-rs-running:*  
 $\{R\text{-a0m1a-rs}\}$   
 $(a0n\text{-running } [B, Sv] (Kab, A, \text{take } rs\text{-len } nls)),$   
 $(m1a\text{-step3 } Rs\ A\ B\ Kab\ nls)$   
 $\{> R\text{-a0m1a-rs}\}$   
 <proof>

**lemma** *PO-m1a-step4-refines-a0-rs-skip:*  
 $\{R\text{-a0m1a-rs}\}$   
 $\text{Id}, (m1a\text{-step4 } Ra\ A\ B\ Kab\ nla)$   
 $\{> R\text{-a0m1a-rs}\}$   
 <proof>

**lemma** *PO-m1a-step5-refines-a0-rs-commit:*  
 $\{R\text{-a0m1a-rs} \cap UNIV \times m1a\text{-inv0-fin}\}$   
 $(a0n\text{-commit } [B, Sv] (Kab, A, \text{take } rs\text{-len } nlb)),$   
 $(m1a\text{-step5 } Rb\ A\ B\ Kab\ nlb)$   
 $\{> R\text{-a0m1a-rs}\}$   
 <proof>

**lemma** *PO-m1a-leak-refines-a0-rs-skip:*  
 $\{R\text{-a0m1a-rs}\}$   
 $\text{Id}, (m1a\text{-leak } Rs)$   
 $\{> R\text{-a0m1a-rs}\}$   
 <proof>

All together now...

**lemmas** *PO-m1a-trans-refines-a0-rs-trans* =  
*PO-m1a-step1-refines-a0-rs-skip PO-m1a-step2-refines-a0-rs-skip*  
*PO-m1a-step3-refines-a0-rs-running PO-m1a-step4-refines-a0-rs-skip*  
*PO-m1a-step5-refines-a0-rs-commit PO-m1a-leak-refines-a0-rs-skip*

**lemma** *PO-m1a-refines-init-ra0n* [iff]:  
 $init\ m1a \subseteq R\text{-}a0m1a\text{-}rs \text{“}(init\ a0n)$   
 ⟨proof⟩

**lemma** *PO-m1a-refines-trans-ra0n* [iff]:  
 $\{R\text{-}a0m1a\text{-}rs \cap UNIV \times m1a\text{-}inv0\text{-}fin\}$   
 $(trans\ a0n), (trans\ m1a)$   
 $\{>\ R\text{-}a0m1a\text{-}rs\}$   
 ⟨proof⟩

**lemma** *obs-consistent-med-a0m1a-rs* [iff]:  
 $obs\text{-}consistent\ (R\text{-}a0m1a\text{-}rs \cap UNIV \times m1a\text{-}inv0\text{-}fin)\ med\text{-}a0m1a\text{-}rs\ a0n\ m1a$   
 ⟨proof⟩

Refinement result.

**lemma** *PO-m1a-refines-a0-rs* [iff]:  
 $refines\ (R\text{-}a0m1a\text{-}rs \cap UNIV \times m1a\text{-}inv0\text{-}fin)\ med\text{-}a0m1a\text{-}rs\ a0n\ m1a$   
 ⟨proof⟩

**lemma** *m1a-implements-ra0n: implements med-a0m1a-rs a0n m1a*  
 ⟨proof⟩

**end**

### 3.4 Abstract Kerberos core protocol (L1)

**theory** *m1-kerberos* **imports** *m1-keydist-iirn*  
**begin**

We augment the basic abstract key distribution model such that the server sends a timestamp along with the session key. We use a cache to guard against replay attacks and timestamp validity checks to ensure recentness of the session key.

We establish three refinements for this model, namely that this model refines

1. the authenticated key distribution model *m1-keydist-iirn*,
2. the injective agreement model *a0i*, instantiated such that the responder agrees with the initiator on the session key, its timestamp and the initiator’s authenticator timestamp.
3. the injective agreement model *a0i*, instantiated such that the initiator agrees with the responder on the session key, its timestamp and the initiator’s authenticator timestamp.

### 3.4.1 State

We extend the basic key distribution by adding timestamps. We add a clock variable modeling the current time and an authenticator replay cache recording triples  $(A, Kab, Ta)$  of agents, session keys, and authenticator timestamps. The inclusion of the session key avoids false replay rejections for different keys with identical authenticator timestamps.

The frames, runs, and observations remain the same as in the previous model, but we will use the *nat list*'s to store timestamps.

#### type-synonym

$time = nat$  — for clock and timestamps

#### consts

$Ls :: time$  — life time for session keys

$La :: time$  — life time for authenticators

State and observations

#### record

$m1-state = m1r-state +$

$leak :: (key \times agent \times agent \times nonce \times time) set$  — key leaked plus context

$clk :: time$

$cache :: (agent \times key \times time) set$

**type-synonym**  $m1-obs = m1-state$

**type-synonym**  $'x m1-pred = 'x m1-state-scheme set$

**type-synonym**  $'x m1-trans = ('x m1-state-scheme \times 'x m1-state-scheme) set$

#### consts

$END :: atom$  — run end marker (for initiator)

### 3.4.2 Events

**definition** — by  $A$ , refines  $m1x-step1$

$m1-step1 :: [rid-t, agent, agent, nonce] \Rightarrow 'x m1-trans$

**where**

$m1-step1 \equiv m1a-step1$

**definition** — by  $B$ , refines  $m1x-step2$

$m1-step2 :: [rid-t, agent, agent] \Rightarrow 'x m1-trans$

**where**

$m1-step2 \equiv m1a-step2$

**definition** — by  $Sv$ , refines  $m1x-step3$

$m1-step3 :: [rid-t, agent, agent, key, nonce, time] \Rightarrow 'x m1-trans$

**where**

$m1-step3 Rs A B Kab Na Ts \equiv \{(s, s')\}.$

— new guards:

$Ts = clk s \wedge$  — fresh timestamp

— rest as before:

$(s, s') \in m1a-step3 Rs A B Kab Na [aNum Ts]$   
}

**definition** — by  $A$ , refines  $m1x\text{-step5}$

$m1\text{-step4} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}, \text{time}, \text{time}] \Rightarrow 'x\ m1\text{-trans}$

**where**

$m1\text{-step4}\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta \equiv \{(s, s')\}.$

— previous guards:

$runs\ s\ Ra = \text{Some}\ (Init, [A, B], []) \wedge$

$(Kab \notin \text{Domain}\ (\text{leak}\ s) \longrightarrow (Kab, A) \in \text{azC}\ (runs\ s)) \wedge$  — authorization guard

$Na = Ra\$na \wedge$  — fix parameter

— guard for agreement with server on  $(Kab, B, Na, isl)$ ,

— where  $isl = \text{take}\ is\text{-len}\ nla$ ; injectiveness by including  $Na$

$(A \notin \text{bad} \longrightarrow (\exists Rs. Kab = \text{sesK}\ (Rs\$sk) \wedge$

$runs\ s\ Rs = \text{Some}\ (Serv, [A, B], [aNon\ Na, aNum\ Ts]))) \wedge$

— new guards:

$Ta = \text{clk}\ s \wedge$  — fresh timestamp

$\text{clk}\ s < Ts + Ls \wedge$  — ensure session key recentness

— actions:

$s' = s\{ runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])) \}$

**definition** — by  $B$ , refines  $m1x\text{-step4}$

$m1\text{-step5} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}, \text{time}, \text{time}] \Rightarrow 'x\ m1\text{-trans}$

**where**

$m1\text{-step5}\ Rb\ A\ B\ Kab\ Ts\ Ta \equiv \{(s, s')\}.$

— previous guards:

$runs\ s\ Rb = \text{Some}\ (Resp, [A, B], []) \wedge$

$(Kab \notin \text{Domain}\ (\text{leak}\ s) \longrightarrow (Kab, B) \in \text{azC}\ (runs\ s)) \wedge$  — authorization guard

— guard for showing agreement with server on  $(Kab, A, rsl)$ ,

— where  $rsl = \text{take}\ rs\text{-len}\ nlb$ ; this agreement is non-injective

$(B \notin \text{bad} \longrightarrow (\exists Rs\ Na. Kab = \text{sesK}\ (Rs\$sk) \wedge$

$runs\ s\ Rs = \text{Some}\ (Serv, [A, B], [aNon\ Na, aNum\ Ts]))) \wedge$

— new guards:

— guard for showing agreement with initiator  $A$  on  $(Kab, Ts, Ta)$

$(A \notin \text{bad} \longrightarrow B \notin \text{bad} \longrightarrow$

$(\exists Ra\ nl. runs\ s\ Ra = \text{Some}\ (Init, [A, B], [aKey\ Kab \# aNum\ Ts \# aNum\ Ta \# nl]))) \wedge$

— ensure recentness of session key

$\text{clk}\ s < Ts + Ls \wedge$

— check validity of authenticator and prevent its replay

— 'replays' with fresh authenticator ok!

$\text{clk}\ s < Ta + La \wedge$

$(B, Kab, Ta) \notin \text{cache}\ s \wedge$

— actions:

$s' = s\{$

$runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])),$

$\text{cache} := \text{insert}\ (B, Kab, Ta)\ (\text{cache}\ s)$

$\Downarrow$   
 $\}$

**definition** — by  $A$ , refines  $skip$

$m1\text{-step6} :: [rid\text{-}t, agent, agent, nonce, key, time, time] \Rightarrow 'x\ m1\text{-trans}$

**where**

$m1\text{-step6}\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta \equiv \{(s, s')\}.$

$runs\ s\ Ra = Some\ (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta]) \wedge$  — key recv'd before  
 $Na = Ra\$na \wedge$  — fix parameter

— check key's freshness [NEW]

—  $clk\ s < Ts + Ls \wedge$

— guard for showing agreement with  $B$  on  $Kab$ ,  $Ts$ , and  $Ta$

$(A \notin bad \longrightarrow B \notin bad \longrightarrow$

$(\exists Rb. runs\ s\ Rb = Some\ (Resp, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta]))) \wedge$

— actions: (redundant) update local state marks successful termination

$s' = s\langle$

$runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta, END]))$

$\rangle$

$\}$

**definition** — by attacker, refines  $m1a\text{-leak}$

$m1\text{-leak} :: [rid\text{-}t, agent, agent, nonce, time] \Rightarrow 'x\ m1\text{-trans}$

**where**

$m1\text{-leak}\ Rs\ A\ B\ Na\ Ts \equiv \{(s, s1)\}.$

— guards:

$runs\ s\ Rs = Some\ (Serv, [A, B], [aNon\ Na, aNum\ Ts]) \wedge$

$(clk\ s \geq Ts + Ls) \wedge$  — only compromise 'old' session keys

— actions:

— record session key as leaked;

$s1 = s\langle leak := insert\ (sesK\ (Rs\$sk), A, B, Na, Ts)\ (leak\ s) \rangle$

$\}$

Clock tick event

**definition** — refines  $skip$

$m1\text{-tick} :: time \Rightarrow 'x\ m1\text{-trans}$

**where**

$m1\text{-tick}\ T \equiv \{(s, s')\}.$

$s' = s\langle clk := clk\ s + T \rangle$

$\}$

Purge event: purge cache of expired timestamps

**definition** — refines  $skip$

$m1\text{-purge} :: agent \Rightarrow 'x\ m1\text{-trans}$

**where**

$m1\text{-purge}\ A \equiv \{(s, s')\}.$

$s' = s\langle$

$cache := cache\ s - \{(A, K, T) \mid A\ K\ T.$

$(A, K, T) \in cache\ s \wedge T + La \leq clk\ s$

```

    }
  })
}

```

### 3.4.3 Specification

**definition**

*m1-init* :: *m1-state set*

**where**

*m1-init*  $\equiv$  { ( $\emptyset$  runs = *Map.empty*, leak = *corrKey*  $\times$  {*undefined*}, clk = 0, cache = {} ) }

**definition**

*m1-trans* :: 'x *m1-trans* **where**

*m1-trans*  $\equiv$  ( $\bigcup$  *A B Ra Rb Rs Na Kab Ts Ta T*.

*m1-step1 Ra A B Na*  $\cup$

*m1-step2 Rb A B*  $\cup$

*m1-step3 Rs A B Kab Na Ts*  $\cup$

*m1-step4 Ra A B Na Kab Ts Ta*  $\cup$

*m1-step5 Rb A B Kab Ts Ta*  $\cup$

*m1-step6 Ra A B Na Kab Ts Ta*  $\cup$

*m1-leak Rs A B Na Ts*  $\cup$

*m1-tick T*  $\cup$

*m1-purge A*  $\cup$

*Id*

)

**definition**

*m1* :: (*m1-state*, *m1-obs*) *spec* **where**

*m1*  $\equiv$  ( $\emptyset$

*init* = *m1-init*,

*trans* = *m1-trans*,

*obs* = *id*

)

**lemmas** *m1-loc-defs* =

*m1-def m1-init-def m1-trans-def*

*m1-step1-def m1-step2-def m1-step3-def m1-step4-def m1-step5-def*

*m1-step6-def m1-leak-def m1-purge-def m1-tick-def*

**lemmas** *m1-defs* = *m1-loc-defs m1a-defs*

**lemma** *m1-obs-id* [*simp*]: *obs m1* = *id*

*<proof>*

### 3.4.4 Invariants

**inv0: Finite domain**

There are only finitely many runs. This is needed to establish the responder/initiator agreement.

**definition**

*m1-inv0-fin* :: 'x *m1-pred*

where

$$m1\text{-inv0-fin} \equiv \{s. \text{finite} (\text{dom} (\text{runs } s))\}$$

**lemmas**  $m1\text{-inv0-fin}I = m1\text{-inv0-fin-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m1\text{-inv0-fin}E$  [elim] =  $m1\text{-inv0-fin-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m1\text{-inv0-fin}D = m1\text{-inv0-fin-def}$  [THEN setc-def-to-dest, rule-format]

Invariance proofs.

**lemma**  $PO\text{-}m1\text{-inv0-fin-init}$  [iff]:

$$\text{init } m1 \subseteq m1\text{-inv0-fin}$$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m1\text{-inv0-fin-trans}$  [iff]:

$$\{m1\text{-inv0-fin}\} \text{ trans } m1 \{> m1\text{-inv0-fin}\}$$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m1\text{-inv0-fin}$  [iff]:  $\text{reach } m1 \subseteq m1\text{-inv0-fin}$

$\langle \text{proof} \rangle$

### inv1: Caching invariant for responder

**definition**

$$m1\text{-inv1r-cache} :: 'x \text{ m1-pred}$$

where

$$m1\text{-inv1r-cache} \equiv \{s. \forall Rb \ A \ B \ Kab \ Ts \ Ta \ nl.$$

$$\text{runs } s \ Rb = \text{Some} (\text{Resp}, [A, B], aKey \ Kab \ \# \ aNum \ Ts \ \# \ aNum \ Ta \ \# \ nl) \longrightarrow$$

$$\text{clk } s < Ta + La \longrightarrow$$

$$(B, Kab, Ta) \in \text{cache } s$$

}

**lemmas**  $m1\text{-inv1r-cache}I = m1\text{-inv1r-cache-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m1\text{-inv1r-cache}E$  [elim] =  $m1\text{-inv1r-cache-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m1\text{-inv1r-cache}D = m1\text{-inv1r-cache-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proof

**lemma**  $PO\text{-}m1\text{-inv1r-cache-init}$  [iff]:

$$\text{init } m1 \subseteq m1\text{-inv1r-cache}$$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m1\text{-inv1r-cache-trans}$  [iff]:

$$\{m1\text{-inv1r-cache}\} \text{ trans } m1 \{> m1\text{-inv1r-cache}\}$$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m1\text{-inv1r-cache}$  [iff]:  $\text{reach } m1 \subseteq m1\text{-inv1r-cache}$

$\langle \text{proof} \rangle$

### 3.4.5 Refinement of $m1a$

**Simulation relation**

The abstraction removes all but the first freshness identifiers (corresponding to  $Kab$  and  $Ts$ ) from the initiator and responder frames and leaves the server's freshness ids untouched.

**overloading**  $is-len' \equiv is-len$   $rs-len' \equiv rs-len$  **begin**

**definition**  $is-len-def$  [*simp*]:  $is-len' \equiv 1::nat$

**definition**  $rs-len-def$  [*simp*]:  $rs-len' \equiv 1::nat$

**end**

**fun**

$rm1a1 :: role-t \Rightarrow atom\ list \Rightarrow atom\ list$

**where**

$rm1a1\ Init = take\ (Suc\ is-len)$  — take  $Kab, Ts$ ; drop  $Ta$

|  $rm1a1\ Resp = take\ (Suc\ rs-len)$  — take  $Kab, Ts$ ; drop  $Ta$

|  $rm1a1\ Serv = id$  — take  $Na, Ts$

**abbreviation**

$runs1a1 :: runs-t \Rightarrow runs-t$  **where**

$runs1a1 \equiv map-runs\ rm1a1$

**lemma**  $knC-runs1a1$  [*simp*]:

$knC\ (runs1a1\ runz) = knC\ runz$

$\langle proof \rangle$

med1a1: The mediator function maps a concrete observation (i.e., run) to an abstract one.

R1a1: The simulation relation is defined in terms of the mediator function.

**definition**

$med1a1 :: m1-obs \Rightarrow m1a-obs$  **where**

$med1a1\ s \equiv \langle runs = runs1a1\ (runs\ s), m1x-state.leak = Domain\ (leak\ s) \rangle$

**definition**

$R1a1 :: (m1a-state \times m1-state)\ set$  **where**

$R1a1 \equiv \{(s, t). s = med1a1\ t\}$

**lemmas**  $R1a1-defs = R1a1-def\ med1a1-def$

## Refinement proof

**lemma**  $PO-m1-step1-refines-m1a-step1$ :

$\{R1a1\}$

$(m1a-step1\ Ra\ A\ B\ Na), (m1-step1\ Ra\ A\ B\ Na)$

$\{>\ R1a1\}$

$\langle proof \rangle$

**lemma**  $PO-m1-step2-refines-m1a-step2$ :

$\{R1a1\}$

$(m1a-step2\ Rb\ A\ B), (m1-step2\ Rb\ A\ B)$

$\{>\ R1a1\}$

$\langle proof \rangle$

**lemma**  $PO-m1-step3-refines-m1a-step3$ :

$\{R1a1\}$

$(m1a-step3\ Rs\ A\ B\ Kab\ Na\ [aNum\ Ts]), (m1-step3\ Rs\ A\ B\ Kab\ Na\ Ts)$

$\{>\ R1a1\}$

$\langle proof \rangle$

**lemma** *PO-m1-step4-refines-m1a-step4*:

{*R1a1*}  
(*m1a-step4 Ra A B Na Kab [aNum Ts]*), (*m1-step4 Ra A B Na Kab Ts Ta*)  
{> *R1a1*}  
<proof>

**lemma** *PO-m1-step5-refines-m1a-step5*:

{*R1a1*}  
(*m1a-step5 Rb A B Kab [aNum Ts]*), (*m1-step5 Rb A B Kab Ts Ta*)  
{> *R1a1*}  
<proof>

**lemma** *PO-m1-step6-refines-m1a-skip*:

{*R1a1*}  
*Id*, (*m1-step6 Ra A B Na Kab Ts Ta*)  
{> *R1a1*}  
<proof>

**lemma** *PO-m1-leak-refines-m1a-leak*:

{*R1a1*}  
(*m1a-leak Rs*), (*m1-leak Rs A B Na Ts*)  
{> *R1a1*}  
<proof>

**lemma** *PO-m1-tick-refines-m1a-skip*:

{*R1a1*}  
*Id*, (*m1-tick T*)  
{> *R1a1*}  
<proof>

**lemma** *PO-m1-purge-refines-m1a-skip*:

{*R1a1*}  
*Id*, (*m1-purge A*)  
{> *R1a1*}  
<proof>

All together now...

**lemmas** *PO-m1-trans-refines-m1a-trans* =

*PO-m1-step1-refines-m1a-step1 PO-m1-step2-refines-m1a-step2*  
*PO-m1-step3-refines-m1a-step3 PO-m1-step4-refines-m1a-step4*  
*PO-m1-step5-refines-m1a-step5 PO-m1-step6-refines-m1a-skip*  
*PO-m1-leak-refines-m1a-leak PO-m1-tick-refines-m1a-skip*  
*PO-m1-purge-refines-m1a-skip*

**lemma** *PO-m1-refines-init-m1a [iff]*:

*init m1*  $\subseteq$  *R1a1*“(*init m1a*)  
<proof>

**lemma** *PO-m1-refines-trans-m1a [iff]*:

{*R1a1*}  
(*trans m1a*), (*trans m1*)  
{> *R1a1*}  
<proof>

Observation consistency.

**lemma** *obs-consistent-med1a1* [iff]:  
*obs-consistent R1a1 med1a1 m1a m1*  
 ⟨proof⟩

Refinement result.

**lemma** *PO-m1-refines-m1a* [iff]:  
*refines R1a1 med1a1 m1a m1*  
 ⟨proof⟩

**lemma** *m1-implements-m1a* [iff]: *implements med1a1 m1a m1*  
 ⟨proof⟩

### inv (inherited): Secrecy

Secrecy, as external and internal invariant

#### definition

*m1-secrecy* :: 'x *m1-pred* **where**  
*m1-secrecy* ≡ {s. *knC* (runs s) ⊆ *azC* (runs s) ∪ *Domain* (*leak* s) × *UNIV*}

**lemmas** *m1-secrecyI* = *m1-secrecy-def* [THEN *setc-def-to-intro*, *rule-format*]

**lemmas** *m1-secrecyE* [elim] = *m1-secrecy-def* [THEN *setc-def-to-elim*, *rule-format*]

**lemma** *PO-m1-obs-secrecy* [iff]: *oreach m1* ⊆ *m1-secrecy*  
 ⟨proof⟩

**lemma** *PO-m1-secrecy* [iff]: *reach m1* ⊆ *m1-secrecy*  
 ⟨proof⟩

### inv (inherited): Responder auth server.

#### definition

*m1-inv2r-serv* :: 'x *m1r-pred*

#### where

*m1-inv2r-serv* ≡ {s. ∀ A B Rb Kab Ts nlb.  
 B ∉ *bad* →  
 runs s Rb = *Some* (*Resp*, [A, B], *aKey* Kab # *aNum* Ts # *nlb*) →  
 (∃ Rs Na. Kab = *sesK* (Rs\$sk) ∧  
 runs s Rs = *Some* (*Serv*, [A, B], [*aNon* Na, *aNum* Ts]))  
 }

**lemmas** *m1-inv2r-servI* = *m1-inv2r-serv-def* [THEN *setc-def-to-intro*, *rule-format*]

**lemmas** *m1-inv2r-servE* [elim] = *m1-inv2r-serv-def* [THEN *setc-def-to-elim*, *rule-format*]

**lemmas** *m1-inv2r-servD* = *m1-inv2r-serv-def* [THEN *setc-def-to-dest*, *rule-format*, *rotated -1*]

Proof of invariance.

**lemma** *PO-m1-inv2r-serv* [iff]: *reach m1* ⊆ *m1-inv2r-serv*  
 ⟨proof⟩

### inv (inherited): Initiator auth server.

Simplified version of invariant *m1a-inv2i-serv*.

**definition**

$$m1\text{-inv}2i\text{-serv} :: 'x\ m1r\text{-pred}$$
**where**

$$\begin{aligned} m1\text{-inv}2i\text{-serv} &\equiv \{s. \forall A\ B\ Ra\ Kab\ Ts\ nla. \\ &A \notin \text{bad} \longrightarrow \\ &\text{runs } s\ Ra = \text{Some } (\text{Init}, [A, B], aKey\ Kab\ \# \ aNum\ Ts\ \# \ nla) \longrightarrow \\ &(\exists Rs. Kab = \text{ses}K\ (Rs\$sk) \wedge \\ &\text{runs } s\ Rs = \text{Some } (\text{Serv}, [A, B], [aNon\ (Ra\$na), aNum\ Ts])) \\ &\} \end{aligned}$$

**lemmas**  $m1\text{-inv}2i\text{-serv}I = m1\text{-inv}2i\text{-serv}\text{-def } [THEN\ \text{setc}\text{-def}\text{-to}\text{-intro},\ \text{rule}\text{-format}]$

**lemmas**  $m1\text{-inv}2i\text{-serv}E [elim] = m1\text{-inv}2i\text{-serv}\text{-def } [THEN\ \text{setc}\text{-def}\text{-to}\text{-elim},\ \text{rule}\text{-format}]$

**lemmas**  $m1\text{-inv}2i\text{-serv}D = m1\text{-inv}2i\text{-serv}\text{-def } [THEN\ \text{setc}\text{-def}\text{-to}\text{-dest},\ \text{rule}\text{-format},\ \text{rotated } -1]$

Proof of invariance.

**lemma**  $PO\text{-}m1\text{-inv}2i\text{-serv } [iff]:\ \text{reach } m1 \subseteq m1\text{-inv}2i\text{-serv}$   
 $\langle\text{proof}\rangle$

**declare**  $PO\text{-}m1\text{-inv}2i\text{-serv } [THEN\ \text{subset}D,\ \text{intro}]$

**inv (inherited): Initiator key freshness****definition**

$$m1\text{-inv}1\text{-ifresh} :: 'x\ m1\text{-pred}$$
**where**

$$\begin{aligned} m1\text{-inv}1\text{-ifresh} &\equiv \{s. \forall A\ A'\ B\ B'\ Ra\ Ra'\ Kab\ nl\ nl'. \\ &\text{runs } s\ Ra = \text{Some } (\text{Init}, [A, B], aKey\ Kab\ \# \ nl) \longrightarrow \\ &\text{runs } s\ Ra' = \text{Some } (\text{Init}, [A', B'], aKey\ Kab\ \# \ nl') \longrightarrow \\ &A \notin \text{bad} \longrightarrow B \notin \text{bad} \longrightarrow Kab \notin \text{Domain } (\text{leak } s) \longrightarrow \\ &Ra = Ra' \\ &\} \end{aligned}$$

**lemmas**  $m1\text{-inv}1\text{-ifresh}I = m1\text{-inv}1\text{-ifresh}\text{-def } [THEN\ \text{setc}\text{-def}\text{-to}\text{-intro},\ \text{rule}\text{-format}]$

**lemmas**  $m1\text{-inv}1\text{-ifresh}E [elim] = m1\text{-inv}1\text{-ifresh}\text{-def } [THEN\ \text{setc}\text{-def}\text{-to}\text{-elim},\ \text{rule}\text{-format}]$

**lemmas**  $m1\text{-inv}1\text{-ifresh}D = m1\text{-inv}1\text{-ifresh}\text{-def } [THEN\ \text{setc}\text{-def}\text{-to}\text{-dest},\ \text{rule}\text{-format},\ \text{rotated } 1]$

**lemma**  $PO\text{-}m1\text{-ifresh } [iff]:\ \text{reach } m1 \subseteq m1\text{-inv}1\text{-ifresh}$   
 $\langle\text{proof}\rangle$

**3.4.6 Refinement of  $a0i$  for responder/initiator**

The responder injectively agrees with the initiator on  $Kab$ ,  $Ts$ , and  $Ta$ .

**Simulation relation**

We define two auxiliary functions to reconstruct the signals of the initial model from completed initiator and responder runs.

**type-synonym**

$$\text{risig} = \text{key} \times \text{time} \times \text{time}$$
**abbreviation**

$ri\text{-}running :: [runs\text{-}t, agent, agent, key, time, time] \Rightarrow rid\text{-}t\ set$

**where**

$ri\text{-}running\ runz\ A\ B\ Kab\ Ts\ Ta \equiv \{Ra. \exists nl.$   
 $\quad runz\ Ra = Some\ (Init, [A, B], aKey\ Kab\ \# aNum\ Ts\ \# aNum\ Ta\ \# nl)$   
 $\}$

**abbreviation**

$ri\text{-}commit :: [runs\text{-}t, agent, agent, key, time, time] \Rightarrow rid\text{-}t\ set$

**where**

$ri\text{-}commit\ runz\ A\ B\ Kab\ Ts\ Ta \equiv \{Rb. \exists nl.$   
 $\quad runz\ Rb = Some\ (Resp, [A, B], aKey\ Kab\ \# aNum\ Ts\ \# aNum\ Ta\ \# nl)$   
 $\}$

**fun**

$ri\text{-}runs2sigs :: runs\text{-}t \Rightarrow risig\ signal \Rightarrow nat$

**where**

$ri\text{-}runs2sigs\ runz\ (Running\ [B, A]\ (Kab, Ts, Ta)) =$   
 $\quad card\ (ri\text{-}running\ runz\ A\ B\ Kab\ Ts\ Ta)$

$| ri\text{-}runs2sigs\ runz\ (Commit\ [B, A]\ (Kab, Ts, Ta)) =$   
 $\quad card\ (ri\text{-}commit\ runz\ A\ B\ Kab\ Ts\ Ta)$

$| ri\text{-}runs2sigs\ runz\ - = 0$

Simulation relation and mediator function. We map completed initiator and responder runs to commit and running signals, respectively.

**definition**

$med\text{-}a0iim1\text{-}ri :: m1\text{-}obs \Rightarrow risig\ a0i\text{-}obs$  **where**  
 $med\text{-}a0iim1\text{-}ri\ o1 \equiv (\mid signals = ri\text{-}runs2sigs\ (runs\ o1), corrupted = \{\} \mid)$

**definition**

$R\text{-}a0iim1\text{-}ri :: (risig\ a0i\text{-}state \times m1\text{-}state)\ set$  **where**  
 $R\text{-}a0iim1\text{-}ri \equiv \{(s, t). signals\ s = ri\text{-}runs2sigs\ (runs\ t) \wedge corrupted\ s = \{\} \}$

**lemmas**  $R\text{-}a0iim1\text{-}ri\text{-}defs = R\text{-}a0iim1\text{-}ri\text{-}def\ med\text{-}a0iim1\text{-}ri\text{-}def$

## Lemmas about the auxiliary functions

Other lemmas

**lemma**  $ri\text{-}runs2sigs\text{-}empty\ [simp]:$

$runz = Map.empty \Longrightarrow ri\text{-}runs2sigs\ runz = (\lambda s. 0)$

$\langle proof \rangle$

**lemma**  $finite\text{-}ri\text{-}running\ [simp, intro]:$

$finite\ (dom\ runz) \Longrightarrow finite\ (ri\text{-}running\ runz\ A\ B\ Kab\ Ts\ Ta)$

$\langle proof \rangle$

**lemma**  $finite\text{-}ri\text{-}commit\ [simp, intro]:$

$finite\ (dom\ runz) \Longrightarrow finite\ (ri\text{-}commit\ runz\ A\ B\ Kab\ Ts\ Ta)$

$\langle proof \rangle$

Update lemmas

**lemma** *ri-runs2sigs-upd-init-none* [simp]:

[[  $Na \notin \text{dom runz}$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Na \mapsto (\text{Init}, [A, B], []))) = \text{ri-runs2sigs runz}$   
 <proof>

**lemma** *ri-runs2sigs-upd-resp-none* [simp]:

[[  $Rb \notin \text{dom runz}$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], []))) = \text{ri-runs2sigs runz}$   
 <proof>

**lemma** *ri-runs2sigs-upd-serv* [simp]:

[[  $Rs \notin \text{dom runz}$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Rs \mapsto (\text{Serv}, [A, B], [a\text{Non } Na, a\text{Num } Ts])))$   
 $= \text{ri-runs2sigs runz}$   
 <proof>

**lemma** *ri-runs2sigs-upd-init-some* [simp]:

[[  $\text{runz } Ra = \text{Some} (\text{Init}, [A, B], []); \text{finite} (\text{dom runz})$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [a\text{Key } Kab, a\text{Num } Ts, a\text{Num } Ta]))) =$   
 $(\text{ri-runs2sigs runz})($   
 $\quad \text{Running } [B, A] (Kab, Ts, Ta) :=$   
 $\quad \text{Suc} (\text{card} (\text{ri-running runz } A B Kab Ts Ta)))$   
 <proof>

**lemma** *ri-runs2sigs-upd-resp-some* [simp]:

[[  $\text{runz } Rb = \text{Some} (\text{Resp}, [A, B], []); \text{finite} (\text{dom runz})$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], [a\text{Key } Kab, a\text{Num } Ts, a\text{Num } Ta]))) =$   
 $(\text{ri-runs2sigs runz})($   
 $\quad \text{Commit } [B, A] (Kab, Ts, Ta) :=$   
 $\quad \text{Suc} (\text{card} (\text{ri-commit runz } A B Kab Ts Ta)))$   
 <proof>

**lemma** *ri-runs2sigs-upd-init-some2* [simp]:

[[  $\text{runz } Ra = \text{Some} (\text{Init}, [A, B], [a\text{Key } Kab, a\text{Num } Ts, a\text{Num } Ta])$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [a\text{Key } Kab, a\text{Num } Ts, a\text{Num } Ta, \text{END}]))) =$   
 $\text{ri-runs2sigs runz}$   
 <proof>

## Refinement proof

**lemma** *PO-m1-step1-refines-a0-ri-skip*:

{  $R\text{-a0iim1-ri}$  }  
 $\text{Id}, (m1\text{-step1 } Ra A B Na)$   
 {  $> R\text{-a0iim1-ri}$  }  
 <proof>

**lemma** *PO-m1-step2-refines-a0-ri-skip*:

{  $R\text{-a0iim1-ri}$  }  
 $\text{Id}, (m1\text{-step2 } Rb A B)$   
 {  $> R\text{-a0iim1-ri}$  }  
 <proof>

**lemma** *PO-m1-step3-refines-a0-ri-skip*:

$\{R\text{-a0iim1-ri}\}$   
 $Id, (m1\text{-step3 } Rs \ A \ B \ Kab \ Na \ Ts)$   
 $\{> R\text{-a0iim1-ri}\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step4-refines-a0-ri-running*:  
 $\{R\text{-a0iim1-ri} \cap UNIV \times m1\text{-inv0-fin}\}$   
 $(a0i\text{-running } [B, A] (Kab, Ts, Ta)), (m1\text{-step4 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta)$   
 $\{> R\text{-a0iim1-ri}\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step5-refines-a0-ri-commit*:  
 $\{R\text{-a0iim1-ri} \cap UNIV \times (m1\text{-inv1r-cache} \cap m1\text{-inv0-fin})\}$   
 $(a0i\text{-commit } [B, A] (Kab, Ts, Ta)), (m1\text{-step5 } Rb \ A \ B \ Kab \ Ts \ Ta)$   
 $\{> R\text{-a0iim1-ri}\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step6-refines-a0-ri-skip*:  
 $\{R\text{-a0iim1-ri}\}$   
 $Id, (m1\text{-step6 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta)$   
 $\{> R\text{-a0iim1-ri}\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-leak-refines-a0-ri-skip*:  
 $\{R\text{-a0iim1-ri}\}$   
 $Id, (m1\text{-leak } Rs \ A \ B \ Na \ Ts)$   
 $\{> R\text{-a0iim1-ri}\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-tick-refines-a0-ri-skip*:  
 $\{R\text{-a0iim1-ri}\}$   
 $Id, (m1\text{-tick } T)$   
 $\{> R\text{-a0iim1-ri}\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-purge-refines-a0-ri-skip*:  
 $\{R\text{-a0iim1-ri}\}$   
 $Id, (m1\text{-purge } A)$   
 $\{> R\text{-a0iim1-ri}\}$   
 $\langle proof \rangle$

All together now...

**lemmas** *PO-m1-trans-refines-a0-ri-trans =*  
 $PO\text{-m1-step1-refines-a0-ri-skip } PO\text{-m1-step2-refines-a0-ri-skip}$   
 $PO\text{-m1-step3-refines-a0-ri-skip } PO\text{-m1-step4-refines-a0-ri-running}$   
 $PO\text{-m1-step5-refines-a0-ri-commit } PO\text{-m1-step6-refines-a0-ri-skip}$   
 $PO\text{-m1-leak-refines-a0-ri-skip } PO\text{-m1-tick-refines-a0-ri-skip}$   
 $PO\text{-m1-purge-refines-a0-ri-skip}$

**lemma** *PO-m1-refines-init-a0-ri [iff]*:  
 $init \ m1 \subseteq R\text{-a0iim1-ri}''(init \ a0i)$   
 $\langle proof \rangle$

**lemma** *PO-m1-refines-trans-a0-ri* [iff]:  
 $\{R\text{-a0iim1-ri} \cap a0i\text{-inv1-iagree} \times (m1\text{-inv1r-cache} \cap m1\text{-inv0-fin})\}$   
 $(\text{trans } a0i), (\text{trans } m1)$   
 $\{> R\text{-a0iim1-ri}\}$   
 ⟨proof⟩

**lemma** *obs-consistent-med-a0iim1-ri* [iff]:  
*obs-consistent*  
 $(R\text{-a0iim1-ri} \cap a0i\text{-inv1-iagree} \times (m1\text{-inv1r-cache} \cap m1\text{-inv0-fin}))$   
 $med\text{-a0iim1-ri } a0i \ m1$   
 ⟨proof⟩

Refinement result.

**lemma** *PO-m1-refines-a0ii-ri* [iff]:  
*refines*  
 $(R\text{-a0iim1-ri} \cap a0i\text{-inv1-iagree} \times (m1\text{-inv1r-cache} \cap m1\text{-inv0-fin}))$   
 $med\text{-a0iim1-ri } a0i \ m1$   
 ⟨proof⟩

**lemma** *m1-implements-a0ii-ri: implements med-a0iim1-ri a0i m1*  
 ⟨proof⟩

### inv3 (inherited): Responder and initiator

This is a translation of the agreement property to Level 1. It follows from the refinement and is needed to prove inv4 below.

#### definition

$m1\text{-inv3r-init} :: 'x \ m1\text{-pred}$

#### where

$m1\text{-inv3r-init} \equiv \{s. \forall A \ B \ Rb \ Kab \ Ts \ Ta \ nlb.$   
 $B \notin bad \longrightarrow A \notin bad \longrightarrow Kab \notin Domain \ (leak \ s) \longrightarrow$   
 $runs \ s \ Rb = Some \ (Resp, [A, B], aKey \ Kab \ \# \ aNum \ Ts \ \# \ aNum \ Ta \ \# \ nlb) \longrightarrow$   
 $(\exists Ra \ nla.$   
 $runs \ s \ Ra = Some \ (Init, [A, B], aKey \ Kab \ \# \ aNum \ Ts \ \# \ aNum \ Ta \ \# \ nla))$   
 $\}$

**lemmas**  $m1\text{-inv3r-init}I = m1\text{-inv3r-init-def} \ [THEN \ setc\text{-def-to-intro}, \ rule\text{-format}]$

**lemmas**  $m1\text{-inv3r-init}E \ [elim] = m1\text{-inv3r-init-def} \ [THEN \ setc\text{-def-to-elim}, \ rule\text{-format}]$

**lemmas**  $m1\text{-inv3r-init}D = m1\text{-inv3r-init-def} \ [THEN \ setc\text{-def-to-dest}, \ rule\text{-format}, \ rotated \ -1]$

Invariance proof.

**lemma** *PO-m1-inv3r-init* [iff]:  $reach \ m1 \subseteq m1\text{-inv3r-init}$   
 ⟨proof⟩

### inv4: Key freshness for responder

#### definition

$m1\text{-inv4-rfresh} :: 'x \ m1\text{-pred}$

#### where

$m1\text{-inv4-rfresh} \equiv \{s. \forall Rb1 \ Rb2 \ A1 \ A2 \ B1 \ B2 \ Kab \ Ts1 \ Ts2 \ Ta1 \ Ta2.$   
 $runs \ s \ Rb1 = Some \ (Resp, [A1, B1], [aKey \ Kab, aNum \ Ts1, aNum \ Ta1]) \longrightarrow$

$$\begin{aligned}
& \text{runs } s \text{ Rb2} = \text{Some} (\text{Resp}, [A2, B2], [\text{aKey } Kab, \text{aNum } Ts2, \text{aNum } Ta2]) \longrightarrow \\
& B1 \notin \text{bad} \longrightarrow A1 \notin \text{bad} \longrightarrow Kab \notin \text{Domain} (\text{leak } s) \longrightarrow \\
& \quad \text{Rb1} = \text{Rb2} \\
& \}
\end{aligned}$$

**lemmas**  $m1\text{-inv4}\text{-rfreshI} = m1\text{-inv4}\text{-rfresh}\text{-def} [\text{THEN } \text{setc}\text{-def}\text{-to}\text{-intro}, \text{rule}\text{-format}]$

**lemmas**  $m1\text{-inv4}\text{-rfreshE} [\text{elim}] = m1\text{-inv4}\text{-rfresh}\text{-def} [\text{THEN } \text{setc}\text{-def}\text{-to}\text{-elim}, \text{rule}\text{-format}]$

**lemmas**  $m1\text{-inv4}\text{-rfreshD} = m1\text{-inv4}\text{-rfresh}\text{-def} [\text{THEN } \text{setc}\text{-def}\text{-to}\text{-dest}, \text{rule}\text{-format}, \text{rotated } 1]$

Proof of key freshness for responder. All cases except step5 are straightforward.

**lemma**  $PO\text{-}m1\text{-inv4}\text{-rfresh}\text{-step5}$ :

$$\begin{aligned}
& \{m1\text{-inv4}\text{-rfresh} \cap m1\text{-inv3r}\text{-init} \cap m1\text{-inv2r}\text{-serv} \cap m1\text{-inv1r}\text{-cache} \cap \\
& \quad m1\text{-secrecy} \cap m1\text{-inv1}\text{-ifresh}\} \\
& \quad (m1\text{-step5 } Rb \ A \ B \ Kab \ Ts \ Ta) \\
& \{> m1\text{-inv4}\text{-rfresh}\} \\
& \langle \text{proof} \rangle
\end{aligned}$$

**lemmas**  $PO\text{-}m1\text{-inv4}\text{-rfresh}\text{-step5}\text{-lemmas} =$   
 $PO\text{-}m1\text{-inv4}\text{-rfresh}\text{-step5}$

**lemma**  $PO\text{-}m1\text{-inv4}\text{-rfresh}\text{-init} [\text{iff}]$ :

$$\begin{aligned}
& \text{init } m1 \subseteq m1\text{-inv4}\text{-rfresh} \\
& \langle \text{proof} \rangle
\end{aligned}$$

**lemma**  $PO\text{-}m1\text{-inv4}\text{-rfresh}\text{-trans} [\text{iff}]$ :

$$\begin{aligned}
& \{m1\text{-inv4}\text{-rfresh} \cap m1\text{-inv3r}\text{-init} \cap m1\text{-inv2r}\text{-serv} \cap m1\text{-inv1r}\text{-cache} \cap \\
& \quad m1\text{-secrecy} \cap m1\text{-inv1}\text{-ifresh}\} \\
& \quad \text{trans } m1 \\
& \{> m1\text{-inv4}\text{-rfresh}\} \\
& \langle \text{proof} \rangle
\end{aligned}$$

**lemma**  $PO\text{-}m1\text{-inv4}\text{-rfresh} [\text{iff}]$ :  $\text{reach } m1 \subseteq m1\text{-inv4}\text{-rfresh}$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m1\text{-obs}\text{-inv4}\text{-rfresh} [\text{iff}]$ :  $\text{oreach } m1 \subseteq m1\text{-inv4}\text{-rfresh}$

$\langle \text{proof} \rangle$

### 3.4.7 Refinement of $a0i$ for initiator/responder

The initiator injectively agrees with the responder on  $Kab$ ,  $Ts$ , and  $Ta$ .

#### Simulation relation

We define two auxiliary functions to reconstruct the signals of the initial model from completed initiator and responder runs.

**type-synonym**

$$\text{irsig} = \text{key} \times \text{time} \times \text{time}$$

**abbreviation**

$$\text{ir}\text{-running} :: [\text{runs}\text{-}t, \text{agent}, \text{agent}, \text{key}, \text{time}, \text{time}] \Rightarrow \text{rid}\text{-}t \text{ set}$$

**where**

$ir\text{-}running\ runz\ A\ B\ Kab\ Ts\ Ta \equiv \{Rb.\ \exists\ nl.$   
 $\quad runz\ Rb = Some\ (Resp,\ [A,\ B],\ aKey\ Kab\ \#\ aNum\ Ts\ \#\ aNum\ Ta\ \#\ nl)$   
 $\}$

**abbreviation**

$ir\text{-}commit :: [runs\text{-}t,\ agent,\ agent,\ key,\ time,\ time] \Rightarrow rid\text{-}t\ set$

**where**

$ir\text{-}commit\ runz\ A\ B\ Kab\ Ts\ Ta \equiv \{Ra.\ \exists\ nl.$   
 $\quad runz\ Ra = Some\ (Init,\ [A,\ B],\ aKey\ Kab\ \#\ aNum\ Ts\ \#\ aNum\ Ta\ \#\ END\ \#\ nl)$   
 $\}$

**fun**

$ir\text{-}runs2sigs :: runs\text{-}t \Rightarrow risig\ signal \Rightarrow nat$

**where**

$ir\text{-}runs2sigs\ runz\ (Running\ [A,\ B]\ (Kab,\ Ts,\ Ta)) =$   
 $\quad card\ (ir\text{-}running\ runz\ A\ B\ Kab\ Ts\ Ta)$   
  
 $| ir\text{-}runs2sigs\ runz\ (Commit\ [A,\ B]\ (Kab,\ Ts,\ Ta)) =$   
 $\quad card\ (ir\text{-}commit\ runz\ A\ B\ Kab\ Ts\ Ta)$   
  
 $| ir\text{-}runs2sigs\ runz\ - = 0$

Simulation relation and mediator function. We map completed initiator and responder runs to commit and running signals, respectively.

**definition**

$med\text{-}a0iim1\text{-}ir :: m1\text{-}obs \Rightarrow irsig\ a0i\text{-}obs$  **where**  
 $med\text{-}a0iim1\text{-}ir\ o1 \equiv (\ signals = ir\text{-}runs2sigs\ (runs\ o1),\ corrupted = \{\} )$

**definition**

$R\text{-}a0iim1\text{-}ir :: (irsig\ a0i\text{-}state \times m1\text{-}state)\ set$  **where**  
 $R\text{-}a0iim1\text{-}ir \equiv \{(s,\ t).\ signals\ s = ir\text{-}runs2sigs\ (runs\ t) \wedge corrupted\ s = \{\}\}$

**lemmas**  $R\text{-}a0iim1\text{-}ir\text{-}defs = R\text{-}a0iim1\text{-}ir\text{-}def\ med\text{-}a0iim1\text{-}ir\text{-}def$

**Lemmas about the auxiliary functions**

**lemma**  $ir\text{-}runs2sigs\text{-}empty$  [simp]:

$runz = Map.empty \Longrightarrow ir\text{-}runs2sigs\ runz = (\lambda s.\ 0)$   
<proof>

**lemma**  $ir\text{-}commit\text{-}finite$  [simp, intro]:

$finite\ (dom\ runz) \Longrightarrow finite\ (ir\text{-}commit\ runz\ A\ B\ Kab\ Ts\ Ta)$   
<proof>

Update lemmas

**lemma**  $ir\text{-}runs2sigs\text{-}upd\text{-}init\text{-}none$  [simp]:

$\llbracket Ra \notin dom\ runz \rrbracket$   
 $\Longrightarrow ir\text{-}runs2sigs\ (runz(Ra \mapsto (Init,\ [A,\ B],\ []))) = ir\text{-}runs2sigs\ runz$   
<proof>

**lemma** *ir-runs2sigs-upd-resp-none* [simp]:

[[  $Rb \notin \text{dom runz}$  ]]  
 $\implies \text{ir-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], []))) = \text{ir-runs2sigs runz}$   
 <proof>

**lemma** *ir-runs2sigs-upd-serv* [simp]:

[[  $Rs \notin \text{dom} (\text{runs } y)$  ]]  
 $\implies \text{ir-runs2sigs} ((\text{runs } y)(Rs \mapsto (\text{Serv}, [A, B], [\text{aNon } Na, \text{aNum } Ts])))$   
 $= \text{ir-runs2sigs} (\text{runs } y)$   
 <proof>

**lemma** *ir-runs2sigs-upd-init-some* [simp]:

[[  $\text{runz } Ra = \text{Some} (\text{Init}, [A, B], [])$  ]]  
 $\implies \text{ir-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [\text{aKey } Kab, \text{aNum } Ts, \text{aNum } Ta]))) =$   
 $\text{ir-runs2sigs runz}$   
 <proof>

**lemma** *ir-runs2sigs-upd-resp-some-raw*:

**assumes**

$\text{runz } Rb = \text{Some} (\text{Resp}, [A, B], [])$   
 $\text{finite} (\text{dom runz})$

**shows**

$\text{ir-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], [\text{aKey } Kab, \text{aNum } Ts, \text{aNum } Ta]))) s =$   
 $((\text{ir-runs2sigs runz})($   
 $\text{Running } [A, B] (Kab, Ts, Ta) :=$   
 $\text{Suc} (\text{card} (\text{ir-running runz } A B Kab Ts Ta)))) s$   
 <proof>

**lemma** *ir-runs2sigs-upd-resp-some* [simp]:

[[  $\text{runz } Rb = \text{Some} (\text{Resp}, [A, B], []); \text{finite} (\text{dom runz})$  ]]  
 $\implies \text{ir-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], [\text{aKey } Kab, \text{aNum } Ts, \text{aNum } Ta]))) =$   
 $(\text{ir-runs2sigs runz})($   
 $\text{Running } [A, B] (Kab, Ts, Ta) :=$   
 $\text{Suc} (\text{card} (\text{ir-running runz } A B Kab Ts Ta)))$   
 <proof>

**lemma** *ir-runs2sigs-upd-init-some2-raw*:

**assumes**

$\text{runz } Ra = \text{Some} (\text{Init}, [A, B], [\text{aKey } Kab, \text{aNum } Ts, \text{aNum } Ta])$   
 $\text{finite} (\text{dom runz})$

**shows**

$\text{ir-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [\text{aKey } Kab, \text{aNum } Ts, \text{aNum } Ta, \text{END}]))) s =$   
 $((\text{ir-runs2sigs runz})($   
 $\text{Commit } [A, B] (Kab, Ts, Ta) :=$   
 $\text{Suc} (\text{card} (\text{ir-commit runz } A B Kab Ts Ta)))) s$   
 <proof>

**lemma** *ir-runs2sigs-upd-init-some2* [simp]:

[[  $\text{runz } Na = \text{Some} (\text{Init}, [A, B], [\text{aKey } Kab, \text{aNum } Ts, \text{aNum } Ta]); \text{finite} (\text{dom runz})$  ]]  
 $\implies \text{ir-runs2sigs} (\text{runz}(Na \mapsto (\text{Init}, [A, B], [\text{aKey } Kab, \text{aNum } Ts, \text{aNum } Ta, \text{END}]))) =$   
 $(\text{ir-runs2sigs runz})($   
 $\text{Commit } [A, B] (Kab, Ts, Ta) :=$

$Suc (card (ir-commit\ runz\ A\ B\ Kab\ Ts\ Ta)))$

$\langle proof \rangle$

### Refinement proof

**lemma** *PO-m1-step1-refines-ir-a0ii-skip:*

$\{R-a0iim1-ir\}$

$Id, (m1-step1\ Ra\ A\ B\ Na)$

$\{>\ R-a0iim1-ir\}$

$\langle proof \rangle$

**lemma** *PO-m1-step2-refines-ir-a0ii-skip:*

$\{R-a0iim1-ir\}$

$Id, (m1-step2\ Rb\ A\ B)$

$\{>\ R-a0iim1-ir\}$

$\langle proof \rangle$

**lemma** *PO-m1-step3-refines-ir-a0ii-skip:*

$\{R-a0iim1-ir\}$

$Id, (m1-step3\ Rs\ A\ B\ Kab\ Na\ Ts)$

$\{>\ R-a0iim1-ir\}$

$\langle proof \rangle$

**lemma** *PO-m1-step4-refines-ir-a0ii-skip:*

$\{R-a0iim1-ir\}$

$Id, (m1-step4\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta)$

$\{>\ R-a0iim1-ir\}$

$\langle proof \rangle$

**lemma** *PO-m1-step5-refines-ir-a0ii-running:*

$\{R-a0iim1-ir \cap UNIV \times m1-inv0-fin\}$

$(a0i-running\ [A, B]\ (Kab, Ts, Ta)), (m1-step5\ Rb\ A\ B\ Kab\ Ts\ Ta)$

$\{>\ R-a0iim1-ir\}$

$\langle proof \rangle$

**lemma** *PO-m1-step6-refines-ir-a0ii-commit:*

$\{R-a0iim1-ir \cap UNIV \times m1-inv0-fin\}$

$(a0n-commit\ [A, B]\ (Kab, Ts, Ta)), (m1-step6\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta)$

$\{>\ R-a0iim1-ir\}$

$\langle proof \rangle$

**lemma** *PO-m1-leak-refines-ir-a0ii-skip:*

$\{R-a0iim1-ir\}$

$Id, (m1-leak\ Rs\ A\ B\ Na\ Ts)$

$\{>\ R-a0iim1-ir\}$

$\langle proof \rangle$

**lemma** *PO-m1-tick-refines-ir-a0ii-skip:*

$\{R-a0iim1-ir\}$

$Id, (m1-tick\ T)$

$\{>\ R-a0iim1-ir\}$

$\langle proof \rangle$

**lemma** *PO-m1-purge-refines-ir-a0ii-skip*:

{*R-a0iim1-ir*}  
  *Id*, (*m1-purge A*)  
  {> *R-a0iim1-ir*}  
<proof>

All together now...

**lemmas** *PO-m1-trans-refines-ir-a0ii-trans* =

*PO-m1-step1-refines-ir-a0ii-skip PO-m1-step2-refines-ir-a0ii-skip*  
*PO-m1-step3-refines-ir-a0ii-skip PO-m1-step4-refines-ir-a0ii-skip*  
*PO-m1-step5-refines-ir-a0ii-running PO-m1-step6-refines-ir-a0ii-commit*  
*PO-m1-leak-refines-ir-a0ii-skip PO-m1-tick-refines-ir-a0ii-skip*  
*PO-m1-purge-refines-ir-a0ii-skip*

**lemma** *PO-m1-refines-init-ir-a0ii* [*iff*]:

*init m1*  $\subseteq$  *R-a0iim1-ir*“(*init a0n*)  
<proof>

**lemma** *PO-m1-refines-trans-ir-a0ii* [*iff*]:

{*R-a0iim1-ir*  $\cap$  *UNIV*  $\times$  *m1-inv0-fin*}  
  (*trans a0n*), (*trans m1*)  
  {> *R-a0iim1-ir*}  
<proof>

Observation consistency.

**lemma** *obs-consistent-med-a0iim1-ir* [*iff*]:

*obs-consistent*  
  (*R-a0iim1-ir*  $\cap$  *UNIV*  $\times$  *m1-inv0-fin*)  
  *med-a0iim1-ir a0n m1*  
<proof>

Refinement result.

**lemma** *PO-m1-refines-a0ii-ir* [*iff*]:

*refines* (*R-a0iim1-ir*  $\cap$  *UNIV*  $\times$  *m1-inv0-fin*)  
  *med-a0iim1-ir a0n m1*  
<proof>

**lemma** *m1-implements-a0ii-ir: implements med-a0iim1-ir a0n m1*

<proof>

end

### 3.5 Abstract Kerberos core protocol (L2)

**theory** *m2-kerberos* **imports** *m1-kerberos* *../Refinement/Channels*  
**begin**

We model an abstract version of the core Kerberos protocol:

- M1.  $A \rightarrow S : A, B, Na$
- M2a.  $S \rightarrow A : \{Kab, Ts, B, Na\}_{Kas}$
- M2b.  $S \rightarrow B : \{Kab, Ts, A\}_{Kbs}$
- M3.  $A \rightarrow B : \{A, Ta\}_{Kab}$
- M4.  $B \rightarrow A : \{Ta\}_{Kab}$

Message 1 is sent over an insecure channel, the other four (cleartext) messages over secure channels.

**declare** *domIff* [*simp, iff del*]

### 3.5.1 State

State and observations

**record** *m2-state* = *m1-state* +  
*chan* :: *chmsg set* — channel messages

**type-synonym**

*m2-obs* = *m1-state*

**definition**

*m2-obs* :: *m2-state*  $\Rightarrow$  *m2-obs* **where**

*m2-obs* *s*  $\equiv$   $\langle$   
*runs* = *runs s*,  
*leak* = *leak s*,  
*clk* = *clk s*,  
*cache* = *cache s*  
 $\rangle$

**type-synonym**

*m2-pred* = *m2-state set*

**type-synonym**

*m2-trans* = (*m2-state*  $\times$  *m2-state*) *set*

### 3.5.2 Events

Protocol events.

**definition** — by *A*, refines *m1a-step1*

*m2-step1* :: [*rid-t, agent, agent, nonce*]  $\Rightarrow$  *m2-trans*

**where**

*m2-step1 Ra A B Na*  $\equiv$   $\{(s, s1)\}$ .

— guards:

*Ra*  $\notin$  *dom (runs s)*  $\wedge$  — *Ra* is fresh

*Na* = *Ra\$na*  $\wedge$  — generate nonce

— actions:

— create initiator thread and send message 1

*s1* = *s* $\langle$

*runs* := (*runs s*)(*Ra*  $\mapsto$  (*Init*, [*A*, *B*],  $\square$ )),

$chan := insert (Insec A B (Msg [aNon Na])) (chan s) \text{ --- send } M1$   
 $\Downarrow$   
 $\}$

**definition** — by  $B$ , refines  $m1e\text{-}step2$   
 $m2\text{-}step2 :: [rid\text{-}t, agent, agent] \Rightarrow m2\text{-}trans$   
**where**  
 $m2\text{-}step2 \equiv m1\text{-}step2$

**definition** — by  $Server$ , refines  $m1e\text{-}step3$   
 $m2\text{-}step3 ::$   
 $[rid\text{-}t, agent, agent, key, nonce, time] \Rightarrow m2\text{-}trans$   
**where**  
 $m2\text{-}step3 Rs A B Kab Na Ts \equiv \{(s, s1)\}.$

— guards:  
 $Rs \notin dom (runs s) \wedge$  — fresh server run  
 $Kab = sesK (Rs\$sk) \wedge$  — fresh session key  
 $Ts = clk s \wedge$  — fresh timestamp

$Insec A B (Msg [aNon Na]) \in chan s \wedge$  — recv  $M1$

— actions:  
— record key and send messages 2 and 3  
 $s1 = s[$   
 $runs := (runs s)(Rs \mapsto (Serv, [A, B], [aNon Na, aNum Ts])),$   
 $chan := \{Secure Sv A (Msg [aKey Kab, aAgt B, aNum Ts, aNon Na]), \text{ --- send } M2a/b$   
 $Secure Sv B (Msg [aKey Kab, aAgt A, aNum Ts])\} \cup chan s$   
 $\Downarrow$   
 $\}$

**definition** — by  $A$ , refines  $m1e\text{-}step4$   
 $m2\text{-}step4 :: [rid\text{-}t, agent, agent, nonce, key, time, time] \Rightarrow m2\text{-}trans$   
**where**  
 $m2\text{-}step4 Ra A B Na Kab Ts Ta \equiv \{(s, s1)\}.$

— guards:  
 $runs s Ra = Some (Init, [A, B], []) \wedge$  — session key not yet recv'd  
 $Na = Ra\$na \wedge$  — fix nonce  
 $Ta = clk s \wedge$  — fresh timestamp  
 $clk s < Ts + Ls \wedge$  — ensure key recentness

$Secure Sv A (Msg [aKey Kab, aAgt B, aNum Ts, aNon Na]) \in chan s \wedge$  — recv  $M2a$

— actions:  
— record session key  
 $s1 = s[$   
 $runs := (runs s)(Ra \mapsto (Init, [A, B], [aKey Kab, aNum Ts, aNum Ta])),$   
 $chan := insert (dAuth Kab (Msg [aAgt A, aNum Ta])) (chan s) \text{ --- send } M3$   
 $\Downarrow$   
 $\}$

**definition** — by  $B$ , refines  $m1e\text{-}step5$   
 $m2\text{-}step5 :: [rid\text{-}t, agent, agent, key, time, time] \Rightarrow m2\text{-}trans$

**where**

$m2\text{-step5 } Rb \ A \ B \ Kab \ Ts \ Ta \equiv \{(s, s1)\}.$

— guards:

$runs \ s \ Rb = Some \ (Resp, [A, B], []) \wedge$  —  $Kab$  not yet received  
 $Secure \ Sv \ B \ (Msg \ [aKey \ Kab, aAgt \ A, aNum \ Ts]) \in \ chan \ s \wedge$  —  $recv \ M2b$   
 $dAuth \ Kab \ (Msg \ [aAgt \ A, aNum \ Ta]) \in \ chan \ s \wedge$  —  $recv \ M3$

— ensure freshness of session key

$clk \ s < Ts + Ls \wedge$

— check authenticator's validity and replay; 'replays' with fresh authenticator ok!

$clk \ s < Ta + La \wedge$

$(B, Kab, Ta) \notin \ cache \ s \wedge$

— actions:

— record session key, send message  $M4$

$s1 = s\{$

$runs := (runs \ s)(Rb \mapsto (Resp, [A, B], [aKey \ Kab, aNum \ Ts, aNum \ Ta])),$

$cache := insert \ (B, Kab, Ta) \ (cache \ s),$

$chan := insert \ (dAuth \ Kab \ (Msg \ [aNum \ Ta])) \ (chan \ s)$  —  $send \ M4$

$\}$

}

**definition** — by  $A$ , refines  $m1e\text{-step6}$

$m2\text{-step6} :: [rid\text{-}t, agent, agent, nonce, key, time, time] \Rightarrow m2\text{-trans}$

**where**

$m2\text{-step6 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta \equiv \{(s, s')\}.$

$runs \ s \ Ra = Some \ (Init, [A, B], [aKey \ Kab, aNum \ Ts, aNum \ Ta]) \wedge$  — key  $recv$ 'd before

$Na = Ra\$na \wedge$  — generated nonce

$clk \ s < Ts + Ls \wedge$  — check session key's recentness

$dAuth \ Kab \ (Msg \ [aNum \ Ta]) \in \ chan \ s \wedge$  —  $recv \ M4$

— actions:

$s' = s\{$

$runs := (runs \ s)(Ra \mapsto (Init, [A, B], [aKey \ Kab, aNum \ Ts, aNum \ Ta, END]))$

$\}$

}

Clock tick event

**definition** — refines  $m1\text{-tick}$

$m2\text{-tick} :: time \Rightarrow m2\text{-trans}$

**where**

$m2\text{-tick} \equiv m1\text{-tick}$

Purge event: purge cache of expired timestamps

**definition** — refines  $m1\text{-purge}$

$m2\text{-purge} :: agent \Rightarrow m2\text{-trans}$

**where**

$m2\text{-purge} \equiv m1\text{-purge}$

Intruder events.

**definition** — refines *m1-leak*

*m2-leak* :: [*rid-t*, *agent*, *agent*, *nonce*, *time*] ⇒ *m2-trans*

**where**

*m2-leak* *Rs A B Na Ts* ≡ {(*s*, *s1*).

— guards:

*runs s Rs* = *Some* (*Serv*, [*A*, *B*], [*aNon Na*, *aNum Ts*]) ∧

(*clk s* ≥ *Ts* + *Ls*) ∧ — only compromise 'old' session keys

— actions:

— record session key as leaked;

— intruder sends himself an insecure channel message containing the key

*s1* = *s*(*leak* := *insert* (*sesK* (*Rs\$sk*), *A*, *B*, *Na*, *Ts*) (*leak s*),

*chan* := *insert* (*Insec undefined undefined* (*Msg* [*aKey* (*sesK* (*Rs\$sk*))])) (*chan s*) )

}

**definition** — refines *Id*

*m2-fake* :: *m2-trans*

**where**

*m2-fake* ≡ {(*s*, *s1*).

— actions:

*s1* = *s*(

— close under fakeable messages

*chan* := *fake ik0* (*dom* (*runs s*)) (*chan s*)

)

}

### 3.5.3 Transition system

**definition**

*m2-init* :: *m2-pred*

**where**

*m2-init* ≡ { (

*runs* = *Map.empty*,

*leak* = *corrKey* × {*undefined*},

*clk* = 0,

*cache* = {},

*chan* = {}

) }

**definition**

*m2-trans* :: *m2-trans* **where**

*m2-trans* ≡ (∪ *A B Ra Rb Rs Na Kab Ts Ta T*.

*m2-step1* *Ra A B Na* ∪

*m2-step2* *Rb A B* ∪

*m2-step3* *Rs A B Kab Na Ts* ∪

*m2-step4* *Ra A B Na Kab Ts Ta* ∪

*m2-step5* *Rb A B Kab Ts Ta* ∪

*m2-step6* *Ra A B Na Kab Ts Ta* ∪

*m2-tick* *T* ∪

*m2-purge* *A* ∪

*m2-leak* *Rs A B Na Ts* ∪

*m2-fake* ∪

*Id*  
)

**definition**

$m2 :: (m2\text{-state}, m2\text{-obs})$  spec **where**

$m2 \equiv \langle$   
 $init = m2\text{-init},$   
 $trans = m2\text{-trans},$   
 $obs = m2\text{-obs}$   
 $\rangle$

**lemmas**  $m2\text{-loc-defs} =$

$m2\text{-def } m2\text{-init-def } m2\text{-trans-def } m2\text{-obs-def}$   
 $m2\text{-step1-def } m2\text{-step2-def } m2\text{-step3-def } m2\text{-step4-def } m2\text{-step5-def}$   
 $m2\text{-step6-def } m2\text{-tick-def } m2\text{-purge-def } m2\text{-leak-def } m2\text{-fake-def}$

**lemmas**  $m2\text{-defs} = m2\text{-loc-defs } m1\text{-defs}$

### 3.5.4 Invariants and simulation relation

#### inv1: Key definedness

All session keys in channel messages stem from existing runs.

**definition**

$m2\text{-inv1-keys} :: m2\text{-state set}$

**where**

$m2\text{-inv1-keys} \equiv \{s. \forall R.$   
 $aKey (sesK (R\$sk)) \in atoms (chan s) \vee sesK (R\$sk) \in Domain (leak s) \longrightarrow$   
 $R \in dom (runs s)$   
 $\}$

**lemmas**  $m2\text{-inv1-keysI} = m2\text{-inv1-keys-def [THEN setc-def-to-intro, rule-format]$

**lemmas**  $m2\text{-inv1-keysE [elim]} = m2\text{-inv1-keys-def [THEN setc-def-to-elim, rule-format]$

**lemmas**  $m2\text{-inv1-keysD} = m2\text{-inv1-keys-def [THEN setc-def-to-dest, rule-format, rotated 1]$

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv1-keys-init [iff]:$

$init m2 \subseteq m2\text{-inv1-keys}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv1-keys-trans [iff]:$

$\{m2\text{-inv1-keys}\ trans m2 \{> m2\text{-inv1-keys}\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv1-keys [iff]: reach m2 \subseteq m2\text{-inv1-keys}$

$\langle proof \rangle$

#### inv2: Definedness of used keys

**definition**

$m2\text{-inv2-keys-for} :: m2\text{-state set}$

**where**

$$m2\text{-inv2-keys-for} \equiv \{s. \forall R. \\ \text{sesK } (R\$sk) \in \text{keys-for } (\text{chan } s) \longrightarrow R \in \text{dom } (\text{runs } s) \\ \}$$

**lemmas**  $m2\text{-inv2-keys-for}I = m2\text{-inv2-keys-for-def}$  [THEN setc-def-to-intro, rule-format]  
**lemmas**  $m2\text{-inv2-keys-for}E$  [elim] =  $m2\text{-inv2-keys-for-def}$  [THEN setc-def-to-elim, rule-format]  
**lemmas**  $m2\text{-inv2-keys-for}D = m2\text{-inv2-keys-for-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv2-keys-for-init}$  [iff]:  
 $\text{init } m2 \subseteq m2\text{-inv2-keys-for}$   
 ⟨proof⟩

**lemma**  $PO\text{-}m2\text{-inv2-keys-for-trans}$  [iff]:  
 $\{m2\text{-inv2-keys-for} \cap m2\text{-inv1-keys}\ \text{trans } m2 \ \{> \ m2\text{-inv2-keys-for}\}$   
 ⟨proof⟩

**lemma**  $PO\text{-}m2\text{-inv2-keys-for}$  [iff]:  $\text{reach } m2 \subseteq m2\text{-inv2-keys-for}$   
 ⟨proof⟩

### inv3a: Session key compromise

A L2 version of a session key compromise invariant. Roughly, it states that adding a set of keys  $KK$  to the parameter  $T$  of  $\text{extr}$  does not help the intruder to extract keys other than those in  $KK$  or extractable without adding  $KK$ .

#### definition

$m2\text{-inv3a-sesK-compr} :: m2\text{-state set}$

#### where

$m2\text{-inv3a-sesK-compr} \equiv \{s. \forall K KK. \\ \text{aKey } K \in \text{extr } (\text{aKey}'KK \cup ik0) (\text{chan } s) \longleftrightarrow (K \in KK \vee \text{aKey } K \in \text{extr } ik0 (\text{chan } s)) \\ \}$

**lemmas**  $m2\text{-inv3a-sesK-compr}I = m2\text{-inv3a-sesK-compr-def}$  [THEN setc-def-to-intro, rule-format]  
**lemmas**  $m2\text{-inv3a-sesK-compr}E$  [elim] =  $m2\text{-inv3a-sesK-compr-def}$  [THEN setc-def-to-elim, rule-format]  
**lemmas**  $m2\text{-inv3a-sesK-compr}D = m2\text{-inv3a-sesK-compr-def}$  [THEN setc-def-to-dest, rule-format]

Additional lemma to get the keys in front

**lemmas**  $\text{insert-commute-aKey} = \text{insert-commute}$  [where  $x=\text{aKey } K$  for  $K$ ]

**lemmas**  $m2\text{-inv3a-sesK-compr-simps} =$   
 $m2\text{-inv3a-sesK-compr}D$   
 $m2\text{-inv3a-sesK-compr}D$  [where  $KK=\text{insert } Kab \ KK$  for  $Kab \ KK$ , simplified]  
 $m2\text{-inv3a-sesK-compr}D$  [where  $KK=\{Kab\}$  for  $Kab$ , simplified]  
 $\text{insert-commute-aKey}$

**lemma**  $PO\text{-}m2\text{-inv3a-sesK-compr-init}$  [iff]:  
 $\text{init } m2 \subseteq m2\text{-inv3a-sesK-compr}$   
 ⟨proof⟩

**lemma**  $PO\text{-}m2\text{-inv3a-sesK-compr-trans}$  [iff]:

$\{m2\text{-inv3a-sesK-compr}\} \text{ trans } m2 \{> m2\text{-inv3a-sesK-compr}\}$   
 $\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m2\text{-inv3a-sesK-compr}$  [iff]:  $\text{reach } m2 \subseteq m2\text{-inv3a-sesK-compr}$   
 $\langle \text{proof} \rangle$

### inv3b: Leakage of old session keys

Only old session keys are leaked to the intruder.

#### definition

$m2\text{-inv3b-leak} :: m2\text{-state set}$

#### where

$m2\text{-inv3b-leak} \equiv \{s. \forall Rs A B Na Ts.$   
 $(\text{sesK } (Rs\$sk), A, B, Na, Ts) \in \text{leak } s \longrightarrow \text{clk } s \geq Ts + Ls$   
 $\}$

**lemmas**  $m2\text{-inv3b-leakI} = m2\text{-inv3b-leak-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv3b-leakE}$  [elim] =  $m2\text{-inv3b-leak-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv3b-leakD} = m2\text{-inv3b-leak-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv3b-leak-init}$  [iff]:

$\text{init } m2 \subseteq m2\text{-inv3b-leak}$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m2\text{-inv3b-leak-trans}$  [iff]:

$\{m2\text{-inv3b-leak} \cap m2\text{-inv1-keys}\} \text{ trans } m2 \{> m2\text{-inv3b-leak}\}$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m2\text{-inv3b-leak}$  [iff]:  $\text{reach } m2 \subseteq m2\text{-inv3b-leak}$

$\langle \text{proof} \rangle$

### inv3: Lost session keys

inv3: Lost but not leaked session keys generated by the server for at least one bad agent. This invariant is needed in the proof of the strengthening of the authorization guards in steps 4 and 5 (e.g.,  $Kab \notin \text{Domain } (\text{leaks } s) \longrightarrow (Kab, A) \in \text{azC } (\text{runs } s)$  for the initiator's step4).

#### definition

$m2\text{-inv3-extrKey} :: m2\text{-state set}$

#### where

$m2\text{-inv3-extrKey} \equiv \{s. \forall K.$   
 $aKey K \in \text{extr ik0 } (\text{chan } s) \longrightarrow$   
 $(K \in \text{corrKey} \wedge K \in \text{Domain } (\text{leak } s)) \vee$   
 $(\exists R A' B' Na' Ts'. K = \text{sesK } (R\$sk) \wedge$   
 $\text{runs } s R = \text{Some } (\text{Serv}, [A', B'], [\text{aNon } Na', \text{aNum } Ts']) \wedge$   
 $(A' \in \text{bad} \vee B' \in \text{bad} \vee (K, A', B', Na', Ts') \in \text{leak } s))$   
 $\}$

**lemmas**  $m2\text{-inv3-extrKeyI} = m2\text{-inv3-extrKey-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv3-extrKeyE}$  [elim] =  $m2\text{-inv3-extrKey-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv3-extrKeyD} = m2\text{-inv3-extrKey-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

**lemma** *PO-m2-inv3-extrKey-init* [iff]:

$init\ m2 \subseteq m2\text{-inv3-extrKey}$

*<proof>*

**lemma** *PO-m2-inv3-extrKey-trans* [iff]:

$\{m2\text{-inv3-extrKey} \cap m2\text{-inv3a-sesK-compr}\}$

$trans\ m2$

$\{>\ m2\text{-inv3-extrKey}\}$

*<proof>*

**lemma** *PO-m2-inv3-extrKey* [iff]:  $reach\ m2 \subseteq m2\text{-inv3-extrKey}$

*<proof>*

#### inv4: Messages M2a/M2b for good agents and server state

inv4: Secure messages to honest agents and server state; one variant for each of M2a and M2b. These invariants establish guard strengthening for server authentication by the initiator and the responder.

##### definition

$m2\text{-inv4-M2a} :: m2\text{-state set}$

##### where

$m2\text{-inv4-M2a} \equiv \{s. \forall A\ B\ Kab\ Ts\ Na.$

$Secure\ Sv\ A\ (Msg\ [aKey\ Kab,\ aAgt\ B,\ aNum\ Ts,\ aNon\ Na]) \in chan\ s \longrightarrow A \in good \longrightarrow$

$(\exists Rs.\ Kab = sesK\ (Rs\$sk) \wedge$

$runs\ s\ Rs = Some\ (Serv,\ [A,\ B],\ [aNon\ Na,\ aNum\ Ts]))$

$\}$

##### definition

$m2\text{-inv4-M2b} :: m2\text{-state set}$

##### where

$m2\text{-inv4-M2b} \equiv \{s. \forall A\ B\ Kab\ Ts.$

$Secure\ Sv\ B\ (Msg\ [aKey\ Kab,\ aAgt\ A,\ aNum\ Ts]) \in chan\ s \longrightarrow B \in good \longrightarrow$

$(\exists Rs\ Na.\ Kab = sesK\ (Rs\$sk) \wedge$

$runs\ s\ Rs = Some\ (Serv,\ [A,\ B],\ [aNon\ Na,\ aNum\ Ts]))$

$\}$

**lemmas**  $m2\text{-inv4-M2aI} = m2\text{-inv4-M2a-def}\ [THEN\ setc\text{-def-to-intro},\ rule\text{-format}]$

**lemmas**  $m2\text{-inv4-M2aE}\ [elim] = m2\text{-inv4-M2a-def}\ [THEN\ setc\text{-def-to-elim},\ rule\text{-format}]$

**lemmas**  $m2\text{-inv4-M2aD} = m2\text{-inv4-M2a-def}\ [THEN\ setc\text{-def-to-dest},\ rule\text{-format},\ rotated\ 1]$

**lemmas**  $m2\text{-inv4-M2bI} = m2\text{-inv4-M2b-def}\ [THEN\ setc\text{-def-to-intro},\ rule\text{-format}]$

**lemmas**  $m2\text{-inv4-M2bE}\ [elim] = m2\text{-inv4-M2b-def}\ [THEN\ setc\text{-def-to-elim},\ rule\text{-format}]$

**lemmas**  $m2\text{-inv4-M2bD} = m2\text{-inv4-M2b-def}\ [THEN\ setc\text{-def-to-dest},\ rule\text{-format},\ rotated\ 1]$

Invariance proofs.

**lemma** *PO-m2-inv4-M2a-init* [iff]:

$init\ m2 \subseteq m2\text{-inv4-M2a}$

*<proof>*

**lemma** *PO-m2-inv4-M2a-trans* [iff]:

$\{m2\text{-inv4-M2a}\}\ trans\ m2\ \{>\ m2\text{-inv4-M2a}\}$

*<proof>*

**lemma** *PO-m2-inv4-M2a* [iff]: *reach m2*  $\subseteq$  *m2-inv4-M2a*

*<proof>*

**lemma** *PO-m2-inv4-M2b-init* [iff]:

*init m2*  $\subseteq$  *m2-inv4-M2b*

*<proof>*

**lemma** *PO-m2-inv4-M2b-trans* [iff]:

$\{m2\text{-inv4-M2b}\}$  *trans m2*  $\{>$  *m2-inv4-M2b*

*<proof>*

**lemma** *PO-m2-inv4-M2b* [iff]: *reach m2*  $\subseteq$  *m2-inv4-M2b*

*<proof>*

Consequence needed in proof of inv8/step5 and inv9/step4: The session key uniquely identifies other fields in M2a and M2b, provided it is secret.

**lemma** *m2-inv4-M2a-M2b-match*:

$\llbracket$  *Secure Sv A'* (*Msg* [*aKey Kab*, *aAgt B'*, *aNum Ts'*, *aNon N*])  $\in$  *chan s*;

*Secure Sv B* (*Msg* [*aKey Kab*, *aAgt A*, *aNum Ts*])  $\in$  *chan s*;

*aKey Kab*  $\notin$  *extr ik0* (*chan s*); *s*  $\in$  *m2-inv4-M2a*; *s*  $\in$  *m2-inv4-M2b*  $\rrbracket$

$\implies A = A' \wedge B = B' \wedge Ts = Ts'$

*<proof>*

More consequences of invariants. Needed in ref/step4 and ref/step5 respectively to show the strengthening of the authorization guards.

**lemma** *m2-inv34-M2a-authorized*:

**assumes** *Secure Sv A* (*Msg* [*aKey K*, *aAgt B*, *aNum T*, *aNon N*])  $\in$  *chan s*

*s*  $\in$  *m2-inv4-M2a* *s*  $\in$  *m2-inv3-extrKey*

*K*  $\notin$  *Domain* (*leak s*)

**shows** (*K*, *A*)  $\in$  *azC* (*runs s*)

*<proof>*

**lemma** *m2-inv34-M2b-authorized*:

**assumes** *Secure Sv B* (*Msg* [*aKey K*, *aAgt A*, *aNum T*])  $\in$  *chan s*

*s*  $\in$  *m2-inv4-M2b* *s*  $\in$  *m2-inv3-extrKey*

*K*  $\notin$  *Domain* (*leak s*)

**shows** (*K*, *B*)  $\in$  *azC* (*runs s*)

*<proof>*

## inv5 (derived): Key secrecy for server

inv5: Key secrecy from server perspective. This invariant links the abstract notion of key secrecy to the intruder key knowledge.

**definition**

*m2-inv5-ikk-sv* :: *m2-state set*

**where**

*m2-inv5-ikk-sv*  $\equiv$   $\{s. \forall R A B Na Ts.$

*runs s R = Some* (*Serv*, [*A*, *B*], [*aNon Na*, *aNum Ts*])  $\longrightarrow A \in good \longrightarrow B \in good \longrightarrow$

$$\left. \begin{array}{l} aKey (sesK (R\$sk)) \in extr\ ik0\ (chan\ s) \longrightarrow \\ (sesK (R\$sk), A, B, Na, Ts) \in leak\ s \end{array} \right\}$$

**lemmas**  $m2\text{-inv5}\text{-ikk}\text{-sv}I = m2\text{-inv5}\text{-ikk}\text{-sv}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv5}\text{-ikk}\text{-sv}E$  [elim] =  $m2\text{-inv5}\text{-ikk}\text{-sv}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv5}\text{-ikk}\text{-sv}D = m2\text{-inv5}\text{-ikk}\text{-sv}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proof. This invariant follows from  $m2\text{-inv3}\text{-extrKey}$ .

**lemma**  $m2\text{-inv5}\text{-ikk}\text{-sv}\text{-derived}$ :

$s \in m2\text{-inv3}\text{-extrKey} \implies s \in m2\text{-inv5}\text{-ikk}\text{-sv}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv5}\text{-ikk}\text{-sv}$  [iff]:  $reach\ m2 \subseteq m2\text{-inv5}\text{-ikk}\text{-sv}$

$\langle proof \rangle$

### inv6 (derived): Key secrecy for initiator

This invariant is derivable (see below).

**definition**

$m2\text{-inv6}\text{-ikk}\text{-init} :: m2\text{-state}\ set$

**where**

$m2\text{-inv6}\text{-ikk}\text{-init} \equiv \{s. \forall A\ B\ Ra\ K\ Ts\ nl.$

$runs\ s\ Ra = Some\ (Init, [A, B], aKey\ K\ \# aNum\ Ts\ \# nl) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$

$aKey\ K \in extr\ ik0\ (chan\ s) \longrightarrow$

$(K, A, B, Ra\$na, Ts) \in leak\ s$

$\}$

**lemmas**  $m2\text{-inv6}\text{-ikk}\text{-init}I = m2\text{-inv6}\text{-ikk}\text{-init}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv6}\text{-ikk}\text{-init}E$  [elim] =  $m2\text{-inv6}\text{-ikk}\text{-init}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv6}\text{-ikk}\text{-init}D = m2\text{-inv6}\text{-ikk}\text{-init}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

### inv7 (derived): Key secrecy for responder

This invariant is derivable (see below).

**definition**

$m2\text{-inv7}\text{-ikk}\text{-resp} :: m2\text{-state}\ set$

**where**

$m2\text{-inv7}\text{-ikk}\text{-resp} \equiv \{s. \forall A\ B\ Rb\ K\ Ts\ nl.$

$runs\ s\ Rb = Some\ (Resp, [A, B], aKey\ K\ \# aNum\ Ts\ \# nl) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$

$aKey\ K \in extr\ ik0\ (chan\ s) \longrightarrow$

$(\exists Na. (K, A, B, Na, Ts) \in leak\ s)$

$\}$

**lemmas**  $m2\text{-inv7}\text{-ikk}\text{-resp}I = m2\text{-inv7}\text{-ikk}\text{-resp}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv7}\text{-ikk}\text{-resp}E$  [elim] =  $m2\text{-inv7}\text{-ikk}\text{-resp}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv7}\text{-ikk}\text{-resp}D = m2\text{-inv7}\text{-ikk}\text{-resp}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

### inv8: Relating M4 to the responder state

This invariant relates message M4 from the responder to the responder's state. It is required in the refinement of step 6 to prove that the initiator agrees with the responder on (A, B, Ta,

Kab).

**definition**

$m2\text{-inv8-}M_4 :: m2\text{-state set}$

**where**

$$\begin{aligned} m2\text{-inv8-}M_4 &\equiv \{s. \forall Kab A B Ts Ta N. \\ &\quad \text{Secure Sv } A \text{ (Msg [aKey Kab, aAgt B, aNum Ts, aNon N])} \in \text{chan } s \longrightarrow \\ &\quad \text{dAuth Kab (Msg [aNum Ta])} \in \text{chan } s \longrightarrow \\ &\quad \text{aKey Kab} \notin \text{extr ik0 (chan } s) \longrightarrow \\ &\quad (\exists Rb. \text{runs } s \text{ Rb} = \text{Some (Resp, [A, B], [aKey Kab, aNum Ts, aNum Ta])}) \\ &\} \end{aligned}$$

**lemmas**  $m2\text{-inv8-}M_4I = m2\text{-inv8-}M_4\text{-def [THEN setc-def-to-intro, rule-format]}$

**lemmas**  $m2\text{-inv8-}M_4E \text{ [elim]} = m2\text{-inv8-}M_4\text{-def [THEN setc-def-to-elim, rule-format]}$

**lemmas**  $m2\text{-inv8-}M_4D = m2\text{-inv8-}M_4\text{-def [THEN setc-def-to-dest, rule-format, rotated 1]}$

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv8-}M_4\text{-init [iff]}$ :

$\text{init } m2 \subseteq m2\text{-inv8-}M_4$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m2\text{-inv8-}M_4\text{-trans [iff]}$ :

$$\begin{aligned} &\{m2\text{-inv8-}M_4 \cap m2\text{-inv4-}M2a \cap m2\text{-inv4-}M2b \cap m2\text{-inv3a-sesK-compr} \cap m2\text{-inv2-keys-for}\} \\ &\quad \text{trans } m2 \\ &\{> m2\text{-inv8-}M_4\} \end{aligned}$$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m2\text{-inv8-}M_4 \text{ [iff]}: \text{reach } m2 \subseteq m2\text{-inv8-}M_4$

$\langle \text{proof} \rangle$

## inv9a: Relating the initiator state to M2a

**definition**

$m2\text{-inv9a-init-}M2a :: m2\text{-state set}$

**where**

$$\begin{aligned} m2\text{-inv9a-init-}M2a &\equiv \{s. \forall A B Ra Kab Ts z. \\ &\quad \text{runs } s \text{ Ra} = \text{Some (Init, [A, B], aKey Kab \# aNum Ts \# z)} \longrightarrow \\ &\quad \text{Secure Sv } A \text{ (Msg [aKey Kab, aAgt B, aNum Ts, aNon (Ra\$na)])} \in \text{chan } s \\ &\} \end{aligned}$$

**lemmas**  $m2\text{-inv9a-init-}M2aI = m2\text{-inv9a-init-}M2a\text{-def [THEN setc-def-to-intro, rule-format]}$

**lemmas**  $m2\text{-inv9a-init-}M2aE \text{ [elim]} = m2\text{-inv9a-init-}M2a\text{-def [THEN setc-def-to-elim, rule-format]}$

**lemmas**  $m2\text{-inv9a-init-}M2aD = m2\text{-inv9a-init-}M2a\text{-def [THEN setc-def-to-dest, rule-format, rotated 1]}$

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv9a-init-}M2a\text{-init [iff]}$ :

$\text{init } m2 \subseteq m2\text{-inv9a-init-}M2a$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m2\text{-inv9a-init-}M2a\text{-trans [iff]}$ :

$$\{m2\text{-inv9a-init-}M2a\} \text{ trans } m2 \{> m2\text{-inv9a-init-}M2a\}$$

$\langle \text{proof} \rangle$

**lemma** *PO-m2-inv9a-init-M2a* [iff]:  $reach\ m2 \subseteq m2\text{-inv9a-init-M2a}$   
 ⟨proof⟩

### inv9: Relating M3 to the initiator state

This invariant relates message M3 to the initiator's state. It is required in step 5 of the refinement to prove that the initiator agrees with the responder on (A, B, Ta, Kab).

#### definition

$m2\text{-inv9-M3} :: m2\text{-state set}$

#### where

$m2\text{-inv9-M3} \equiv \{s. \forall Kab\ A\ B\ Ts\ Ta.$

$Secure\ Sv\ B\ (Msg\ [aKey\ Kab,\ aAgt\ A,\ aNum\ Ts]) \in\ chan\ s \longrightarrow$

$dAuth\ Kab\ (Msg\ [aAgt\ A,\ aNum\ Ta]) \in\ chan\ s \longrightarrow$

$aKey\ Kab \notin\ extr\ ik0\ (chan\ s) \longrightarrow A/\#\/b6d/###/B/\#\/b6d/###/$

$(\exists Ra\ nl.\ runs\ s\ Ra = Some\ (Init,\ [A,\ B],\ aKey\ Kab\ \#\ aNum\ Ts\ \#\ aNum\ Ta\ \#\ nl))$

}  
 }  
 }

**lemmas**  $m2\text{-inv9-M3I} = m2\text{-inv9-M3-def}\ [THEN\ setc\text{-def-to-intro},\ rule\text{-format}]$

**lemmas**  $m2\text{-inv9-M3E}\ [elim] = m2\text{-inv9-M3-def}\ [THEN\ setc\text{-def-to-elim},\ rule\text{-format}]$

**lemmas**  $m2\text{-inv9-M3D} = m2\text{-inv9-M3-def}\ [THEN\ setc\text{-def-to-dest},\ rule\text{-format},\ rotated\ 1]$

Invariance proof.

**lemma** *PO-m2-inv9-M3-init* [iff]:

$init\ m2 \subseteq m2\text{-inv9-M3}$

⟨proof⟩

**lemma** *PO-m2-inv9-M3-trans* [iff]:

$\{m2\text{-inv9-M3} \cap m2\text{-inv4-M2a} \cap m2\text{-inv4-M2b} \cap m2\text{-inv3a-sesK-compr} \cap m2\text{-inv2-keys-for}\}$   
 $\quad trans\ m2$

$\{>\ m2\text{-inv9-M3}\}$

⟨proof⟩

**lemma** *PO-m2-inv9-M3* [iff]:  $reach\ m2 \subseteq m2\text{-inv9-M3}$

⟨proof⟩

### 3.5.5 Refinement

The simulation relation. This is a pure superposition refinement.

#### definition

$R12 :: (m1\text{-state} \times m2\text{-state})\ set$  **where**

$R12 \equiv \{(s, t). runs\ s = runs\ t \wedge leak\ s = leak\ t \wedge clk\ s = clk\ t \wedge cache\ s = cache\ t\}$

The mediator function is the identity.

#### definition

$med21 :: m2\text{-obs} \Rightarrow m1\text{-obs}$  **where**

$med21 = id$

Refinement proof.

**lemma** *PO-m2-step1-refines-m1-step1*:

$\{R12\}$

$(m1\text{-step1 } Ra \ A \ B \ Na), (m2\text{-step1 } Ra \ A \ B \ Na)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-step2-refines-m1-step2*:  
 $\{R12\}$   
 $(m1\text{-step2 } Rb \ A \ B), (m2\text{-step2 } Rb \ A \ B)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-step3-refines-m1-step3*:  
 $\{R12\}$   
 $(m1\text{-step3 } Rs \ A \ B \ Kab \ Na \ Ts), (m2\text{-step3 } Rs \ A \ B \ Kab \ Na \ Ts)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-step4-refines-m1-step4*:  
 $\{R12 \cap UNIV \times (m2\text{-inv4-}M2a \cap m2\text{-inv3-extrKey})\}$   
 $(m1\text{-step4 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta), (m2\text{-step4 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-step5-refines-m1-step5*:  
 $\{R12 \cap UNIV$   
 $\times (m2\text{-inv9-}M3 \cap m2\text{-inv5-ikk-sv} \cap m2\text{-inv4-}M2b \cap m2\text{-inv3-extrKey} \cap m2\text{-inv3b-leak})\}$   
 $(m1\text{-step5 } Rb \ A \ B \ Kab \ Ts \ Ta), (m2\text{-step5 } Rb \ A \ B \ Kab \ Ts \ Ta)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-step6-refines-m1-step6*:  
 $\{R12 \cap UNIV$   
 $\times (m2\text{-inv9a-init-}M2a \cap m2\text{-inv8-}M4 \cap m2\text{-inv5-ikk-sv} \cap m2\text{-inv4-}M2a \cap m2\text{-inv3b-leak})\}$   
 $(m1\text{-step6 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta), (m2\text{-step6 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-tick-refines-m1-tick*:  
 $\{R12\}$   
 $(m1\text{-tick } T), (m2\text{-tick } T)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-purge-refines-m1-purge*:  
 $\{R12\}$   
 $(m1\text{-purge } A), (m2\text{-purge } A)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-leak-refines-leak*:  
 $\{R12\}$   
 $m1\text{-leak } Rs \ A \ B \ Na \ Ts, m2\text{-leak } Rs \ A \ B \ Na \ Ts$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-fake-refines-skip*:

{*R12*}  
*Id, m2-fake*  
 {> *R12*}  
 ⟨*proof*⟩

All together now...

**lemmas** *PO-m2-trans-refines-m1-trans* =

*PO-m2-step1-refines-m1-step1 PO-m2-step2-refines-m1-step2*  
*PO-m2-step3-refines-m1-step3 PO-m2-step4-refines-m1-step4*  
*PO-m2-step5-refines-m1-step5 PO-m2-step6-refines-m1-step6*  
*PO-m2-tick-refines-m1-tick PO-m2-purge-refines-m1-purge*  
*PO-m2-leak-refines-leak PO-m2-fake-refines-skip*

**lemma** *PO-m2-refines-init-m1* [iff]:

*init m2*  $\subseteq$  *R12*“(*init m1*)  
 ⟨*proof*⟩

**lemma** *PO-m2-refines-trans-m1* [iff]:

{*R12*  $\cap$   
*UNIV*  $\times$  (*m2-inv9-M3*  $\cap$  *m2-inv9a-init-M2a*  $\cap$  *m2-inv8-M4*  $\cap$   
*m2-inv4-M2b*  $\cap$  *m2-inv4-M2a*  $\cap$  *m2-inv3-extrKey*  $\cap$  *m2-inv3b-leak*)}  
 (*trans m1*), (*trans m2*)  
 {> *R12*}  
 ⟨*proof*⟩

**lemma** *PO-obs-consistent-R12* [iff]:

*obs-consistent R12 med21 m1 m2*  
 ⟨*proof*⟩

Refinement result.

**lemma** *m2-refines-m1* [iff]:

*refines*  
 (*R12*  $\cap$   
 (*UNIV*  $\times$   
 (*m2-inv9-M3*  $\cap$  *m2-inv9a-init-M2a*  $\cap$  *m2-inv8-M4*  $\cap$   
*m2-inv4-M2b*  $\cap$  *m2-inv4-M2a*  $\cap$  *m2-inv3-extrKey*  $\cap$  *m2-inv3b-leak*  $\cap$   
*m2-inv3a-sesK-compr*  $\cap$  *m2-inv2-keys-for*  $\cap$  *m2-inv1-keys*)))  
*med21 m1 m2*)  
 ⟨*proof*⟩

**lemma** *m2-implements-m1* [iff]:

*implements med21 m1 m2*  
 ⟨*proof*⟩

### 3.5.6 Inherited and derived invariants

Show preservation of invariants *m1-inv2i-serv* and *m1-inv2r-serv* from *m1*.

**lemma** *PO-m2-sat-m1-inv2i-serv* [iff]: *reach m2*  $\subseteq$  *m1-inv2i-serv*

⟨*proof*⟩

**lemma** *PO-m2-sat-m1-inv2r-serv* [iff]:  $reach\ m2 \subseteq m1\text{-inv2r-serv}$   
 ⟨proof⟩

Now we derive the L2 key secrecy invariants for the initiator and the responder (see above for the definitions).

**lemma** *PO-m2-inv6-init-ikk* [iff]:  $reach\ m2 \subseteq m2\text{-inv6-ikk-init}$   
 ⟨proof⟩

**lemma** *PO-m2-inv6-resp-ikk* [iff]:  $reach\ m2 \subseteq m2\text{-inv7-ikk-resp}$   
 ⟨proof⟩

end

### 3.6 Core Kerberos, "parallel" variant (L3)

**theory** *m3-kerberos-par* **imports** *m2-kerberos* ../Refinement/Message  
**begin**

We model a direct implementation of the channel-based core Kerberos protocol at Level 2 without ticket forwarding:

- M1.  $A \rightarrow S : A, B, Na$
- M2a.  $S \rightarrow A : \{Kab, B, Ts, Na\}_{Kas}$
- M2b.  $S \rightarrow B : \{Kab, A, Ts\}_{Kbs}$
- M3.  $A \rightarrow B : \{A, Ta\}_{Kab}$
- M4.  $B \rightarrow A : \{Ta\}_{Kab}$

Proof tool configuration. Avoid annoying automatic unfolding of *dom*.

**declare** *domIff* [simp, iff del]

#### 3.6.1 Setup

Now we can define the initial key knowledge.

**overloading** *ltkkeySetup'*  $\equiv$  *ltkkeySetup* **begin**

**definition** *ltkkeySetup-def*: *ltkkeySetup'*  $\equiv$   $\{(shrK\ C, A) \mid C\ A.\ A = C \vee A = Sv\}$

**end**

**lemma** *corrKey-shrK-bad* [simp]:  $corrKey = shrK^{\text{bad}}$   
 ⟨proof⟩

#### 3.6.2 State

The secure channels are star-shaped to/from the server. Therefore, we have only one agent in the relation.

**record** *m3-state* = *m1-state* +

$IK :: msg\ set$  — intruder knowledge

Observable state:  $runs$ ,  $m1-state.leak$ ,  $clk$ , and  $cache$ .

**type-synonym**

$m3-obs = m2-obs$

**definition**

$m3-obs :: m3-state \Rightarrow m3-obs$  **where**  
 $m3-obs\ s \equiv \langle \langle runs = runs\ s, leak = leak\ s, clk = clk\ s, cache = cache\ s \rangle \rangle$

**type-synonym**

$m3-pred = m3-state\ set$

**type-synonym**

$m3-trans = (m3-state \times m3-state)\ set$

### 3.6.3 Events

Protocol events.

**definition** — by  $A$ , refines  $m2-step1$

$m3-step1 :: [rid-t, agent, agent, nonce] \Rightarrow m3-trans$

**where**

$m3-step1\ Ra\ A\ B\ Na \equiv \{(s, s1)\}$ .

— guards:

$Ra \notin dom\ (runs\ s) \wedge$  —  $Ra$  is fresh  
 $Na = Ra\$na \wedge$  — generated nonce

— actions:

$s1 = s \langle \langle \langle runs := (runs\ s)(Ra \mapsto (Init, [A, B], [])), \langle IK := insert\ \{Agent\ A, Agent\ B, Nonce\ Na\}\ (IK\ s) \rangle \rangle \rangle$  — send  $M1$   
 $\rangle$   
 $\}$

**definition** — by  $B$ , refines  $m2-step2$

$m3-step2 :: [rid-t, agent, agent] \Rightarrow m3-trans$

**where**

$m3-step2 \equiv m1-step2$

**definition** — by  $Server$ , refines  $m2-step3$

$m3-step3 :: [rid-t, agent, agent, key, nonce, time] \Rightarrow m3-trans$

**where**

$m3-step3\ Rs\ A\ B\ Kab\ Na\ Ts \equiv \{(s, s1)\}$ .

— guards:

$Rs \notin dom\ (runs\ s) \wedge$  — fresh server run  
 $Kab = sesK\ (Rs\$sk) \wedge$  — fresh session key

$\{Agent\ A, Agent\ B, Nonce\ Na\} \in IK\ s \wedge$  — recv  $M1$   
 $Ts = clk\ s \wedge$  — fresh timestamp

— actions:

— record session key and send  $M2$

$$\begin{array}{l}
s1 = s(| \\
\quad runs := (runs\ s)(Rs \mapsto (Serv, [A, B], [aNon\ Na, aNum\ Ts])), \\
\quad IK := insert\ (Crypt\ (shrK\ A)\ \{\{Key\ Kab, Agent\ B, Number\ Ts, Nonce\ Na\}\}) \\
\quad \quad (insert\ (Crypt\ (shrK\ B)\ \{\{Key\ Kab, Agent\ A, Number\ Ts\}\})\ (IK\ s)) \\
\quad |) \\
\}
\end{array}$$

**definition** — by  $A$ , refines  $m2\text{-}step4$

$m3\text{-}step4 :: [rid\text{-}t, agent, agent, nonce, key, time, time] \Rightarrow m3\text{-}trans$

**where**

$m3\text{-}step4\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta \equiv \{(s, s1)\}.$

— guards:

$runs\ s\ Ra = Some\ (Init, [A, B], []) \wedge$  — key not yet recv'd  
 $Na = Ra\$na \wedge$  — generated nonce

$Crypt\ (shrK\ A)$  — recv  $M2a$   
 $\{\{Key\ Kab, Agent\ B, Number\ Ts, Nonce\ Na\}\} \in IK\ s \wedge$

— read current time

$Ta = clk\ s \wedge$

— check freshness of session key

$clk\ s < Ts + Ls \wedge$

— actions:

— record session key and send  $M3$

$$\begin{array}{l}
s1 = s(| \\
\quad runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])), \\
\quad IK := insert\ (Crypt\ Kab\ \{\{Agent\ A, Number\ Ta\}\})\ (IK\ s) \text{ — } M3 \\
\quad |) \\
\}
\end{array}$$

**definition** — by  $B$ , refines  $m2\text{-}step5$

$m3\text{-}step5 :: [rid\text{-}t, agent, agent, key, time, time] \Rightarrow m3\text{-}trans$

**where**

$m3\text{-}step5\ Rb\ A\ B\ Kab\ Ts\ Ta \equiv \{(s, s1)\}.$

— guards:

$runs\ s\ Rb = Some\ (Resp, [A, B], []) \wedge$  — key not yet recv'd

$Crypt\ (shrK\ B)\ \{\{Key\ Kab, Agent\ A, Number\ Ts\}\} \in IK\ s \wedge$  — recv  $M2b$   
 $Crypt\ Kab\ \{\{Agent\ A, Number\ Ta\}\} \in IK\ s \wedge$  — recv  $M3$

— ensure freshness of session key

$clk\ s < Ts + Ls \wedge$

— check authenticator's validity and replay; 'replays' with fresh authenticator ok!

$clk\ s < Ta + La \wedge$

$(B, Kab, Ta) \notin cache\ s \wedge$

— actions:

— record session key

$$\begin{array}{l}
s1 = s(| \\
\quad runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])), \\
\}
\end{array}$$

```

    cache := insert (B, Kab, Ta) (cache s),
    IK := insert (Crypt Kab (Number Ta)) (IK s)          — send M4
  ⋈
}

```

**definition** — by  $A$ , refines  $m2\text{-step6}$

$m3\text{-step6} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}, \text{time}, \text{time}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step6 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta \equiv \{(s, s')\}.$

— guards:

$runs \ s \ Ra = \text{Some } (Init, [A, B], [aKey \ Kab, aNum \ Ts, aNum \ Ta]) \wedge$  — knows key

$Na = Ra\$na \wedge$  — generated nonce

$clk \ s < Ts + Ls \wedge$  — check session key's recentness

$Crypt \ Kab \ (Number \ Ta) \in IK \ s \wedge$  — recv  $M4$

— actions:

$s' = s \{$

$runs := (runs \ s)(Ra \mapsto (Init, [A, B], [aKey \ Kab, aNum \ Ts, aNum \ Ta, END]))$

$\}$

}

Clock tick event

**definition** — refines  $m2\text{-tick}$

$m3\text{-tick} :: \text{time} \Rightarrow m3\text{-trans}$

**where**

$m3\text{-tick} \equiv m1\text{-tick}$

Purge event: purge cache of expired timestamps

**definition** — refines  $m2\text{-purge}$

$m3\text{-purge} :: \text{agent} \Rightarrow m3\text{-trans}$

**where**

$m3\text{-purge} \equiv m1\text{-purge}$

Session key compromise.

**definition** — refines  $m2\text{-leak}$

$m3\text{-leak} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{time}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-leak } Rs \ A \ B \ Na \ Ts \equiv \{(s, s1)\}.$

— guards:

$runs \ s \ Rs = \text{Some } (Serv, [A, B], [aNon \ Na, aNum \ Ts]) \wedge$

$(clk \ s \geq Ts + Ls) \wedge$  — only compromise 'old' session keys!

— actions:

— record session key as leaked and add it to intruder knowledge

$s1 = s \{ \text{leak} := \text{insert } (sesK \ (Rs\$sk), A, B, Na, Ts) (\text{leak } s),$

$IK := \text{insert } (Key \ (sesK \ (Rs\$sk))) (IK \ s) \}$

}

Intruder fake event. The following "Dolev-Yao" event generates all intruder-derivable messages.

**definition** — refines  $m2\text{-fake}$

$m3-DY-fake :: m3-trans$

**where**

$m3-DY-fake \equiv \{(s, s1).\}$

— actions:

$s1 = s(| IK := synth (analz (IK s)) |) \quad \text{— take DY closure}$   
}

### 3.6.4 Transition system

**definition**

$m3-init :: m3-pred$

**where**

$m3-init \equiv \{ (\mid$   
   $runs = Map.empty,$   
   $leak = shrK'bad \times \{undefined\},$   
   $clk = 0,$   
   $cache = \{\},$   
   $IK = Key'shrK'bad$   
 $\mid \}$

**definition**

$m3-trans :: m3-trans$  **where**

$m3-trans \equiv (\bigcup A B Ra Rb Rs Na Kab Ts Ta T.$   
   $m3-step1 Ra A B Na \cup$   
   $m3-step2 Rb A B \cup$   
   $m3-step3 Rs A B Kab Na Ts \cup$   
   $m3-step4 Ra A B Na Kab Ts Ta \cup$   
   $m3-step5 Rb A B Kab Ts Ta \cup$   
   $m3-step6 Ra A B Na Kab Ts Ta \cup$   
   $m3-tick T \cup$   
   $m3-purge A \cup$   
   $m3-leak Rs A B Na Ts \cup$   
   $m3-DY-fake \cup$   
   $Id$   
)

**definition**

$m3 :: (m3-state, m3-obs) spec$  **where**

$m3 \equiv (\mid$   
   $init = m3-init,$   
   $trans = m3-trans,$   
   $obs = m3-obs$   
 $\mid \}$

**lemmas**  $m3-loc-defs =$

$m3-def m3-init-def m3-trans-def m3-obs-def$   
 $m3-step1-def m3-step2-def m3-step3-def m3-step4-def m3-step5-def$   
 $m3-step6-def m3-tick-def m3-purge-def m3-leak-def m3-DY-fake-def$

**lemmas**  $m3-defs = m3-loc-defs m2-defs$

### 3.6.5 Invariants

Specialized injection that we can apply more aggressively.

**lemmas**  $analz\text{-}Inj\text{-}IK = analz.Inj$  [where  $H=IK\ s$  for  $s$ ]  
**lemmas**  $parts\text{-}Inj\text{-}IK = parts.Inj$  [where  $H=IK\ s$  for  $s$ ]

**declare**  $parts\text{-}Inj\text{-}IK$  [dest!]

**declare**  $analz\text{-}into\text{-}parts$  [dest]

#### inv1: Secrecy of pre-distributed shared keys

inv1: Secrecy of long-term keys

##### definition

$m3\text{-}inv1\text{-}lkeysec :: m3\text{-}pred$

##### where

$m3\text{-}inv1\text{-}lkeysec \equiv \{s. \forall C.$   
 $(Key\ (shrK\ C) \in parts\ (IK\ s) \longrightarrow C \in bad) \wedge$   
 $(C \in bad \longrightarrow Key\ (shrK\ C) \in IK\ s)$   
 $\}$

**lemmas**  $m3\text{-}inv1\text{-}lkeysecI = m3\text{-}inv1\text{-}lkeysec\text{-}def$  [THEN  $setc\text{-}def\text{-}to\text{-}intro$ ,  $rule\text{-}format$ ]

**lemmas**  $m3\text{-}inv1\text{-}lkeysecE$  [elim] =  $m3\text{-}inv1\text{-}lkeysec\text{-}def$  [THEN  $setc\text{-}def\text{-}to\text{-}elim$ ,  $rule\text{-}format$ ]

**lemmas**  $m3\text{-}inv1\text{-}lkeysec\text{-}dest = m3\text{-}inv1\text{-}lkeysec\text{-}def$  [THEN  $setc\text{-}def\text{-}to\text{-}dest$ ,  $rule\text{-}format$ ]

Invariance proof.

**lemma**  $PO\text{-}m3\text{-}inv1\text{-}lkeysec\text{-}init$  [iff]:

$init\ m3 \subseteq m3\text{-}inv1\text{-}lkeysec$

$\langle proof \rangle$

**lemma**  $PO\text{-}m3\text{-}inv1\text{-}lkeysec\text{-}trans$  [iff]:

$\{m3\text{-}inv1\text{-}lkeysec\}\ trans\ m3 \{>\ m3\text{-}inv1\text{-}lkeysec\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m3\text{-}inv1\text{-}lkeysec$  [iff]:  $reach\ m3 \subseteq m3\text{-}inv1\text{-}lkeysec$

$\langle proof \rangle$

Useful simplifier lemmas

**lemma**  $m3\text{-}inv1\text{-}lkeysec\text{-}for\text{-}parts$  [simp]:

$\llbracket s \in m3\text{-}inv1\text{-}lkeysec \rrbracket \implies Key\ (shrK\ C) \in parts\ (IK\ s) \longleftrightarrow C \in bad$

$\langle proof \rangle$

**lemma**  $m3\text{-}inv1\text{-}lkeysec\text{-}for\text{-}analz$  [simp]:

$\llbracket s \in m3\text{-}inv1\text{-}lkeysec \rrbracket \implies Key\ (shrK\ C) \in analz\ (IK\ s) \longleftrightarrow C \in bad$

$\langle proof \rangle$

#### inv7a: Session keys not used to encrypt other session keys

Session keys are not used to encrypt other keys. Proof requires generalization to sets of session keys.

NOTE: This invariant will be derived from the corresponding L2 invariant using the simulation relation.

**definition**

$m3\text{-inv}\gamma a\text{-ses}K\text{-compr} :: m3\text{-pred}$

**where**

$m3\text{-inv}\gamma a\text{-ses}K\text{-compr} \equiv \{s. \forall K KK.$

$KK \subseteq \text{range ses}K \longrightarrow$

$(\text{Key } K \in \text{analz } (\text{Key } KK \cup (IK s))) = (K \in KK \vee \text{Key } K \in \text{analz } (IK s))$

$\}$

**lemmas**  $m3\text{-inv}\gamma a\text{-ses}K\text{-compr}I = m3\text{-inv}\gamma a\text{-ses}K\text{-compr}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m3\text{-inv}\gamma a\text{-ses}K\text{-compr}E = m3\text{-inv}\gamma a\text{-ses}K\text{-compr}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m3\text{-inv}\gamma a\text{-ses}K\text{-compr}D = m3\text{-inv}\gamma a\text{-ses}K\text{-compr}\text{-def}$  [THEN setc-def-to-dest, rule-format]

Additional lemma

**lemmas**  $\text{insert}\text{-commute}\text{-Key} = \text{insert}\text{-commute}$  [where  $x=\text{Key } K$  for  $K$ ]

**lemmas**  $m3\text{-inv}\gamma a\text{-ses}K\text{-compr}\text{-simps} =$

$m3\text{-inv}\gamma a\text{-ses}K\text{-compr}D$

$m3\text{-inv}\gamma a\text{-ses}K\text{-compr}D$  [where  $KK=\{Kab\}$  for  $Kab$ , simplified]

$m3\text{-inv}\gamma a\text{-ses}K\text{-compr}D$  [where  $KK=\text{insert } Kab KK$  for  $Kab KK$ , simplified]

$\text{insert}\text{-commute}\text{-Key}$

### 3.6.6 Refinement

#### Message abstraction and simulation relation

Abstraction function on sets of messages.

**inductive-set**

$\text{abs}\text{-msg} :: \text{msg set} \Rightarrow \text{chmsg set}$

for  $H :: \text{msg set}$

**where**

$\text{am}\text{-M1}:$

$\{\{\text{Agent } A, \text{Agent } B, \text{Nonce } N\}\} \in H$

$\Longrightarrow \text{Insec } A B (\text{Msg } [a\text{Non } N]) \in \text{abs}\text{-msg } H$

|  $\text{am}\text{-M2a}:$

$\text{Crypt } (\text{shr}K C) \{\{\text{Key } K, \text{Agent } B, \text{Number } T, \text{Nonce } N\}\} \in H$

$\Longrightarrow \text{Secure } Sv C (\text{Msg } [a\text{Key } K, a\text{Agt } B, a\text{Num } T, a\text{Non } N]) \in \text{abs}\text{-msg } H$

|  $\text{am}\text{-M2b}:$

$\text{Crypt } (\text{shr}K C) \{\{\text{Key } K, \text{Agent } A, \text{Number } T\}\} \in H$

$\Longrightarrow \text{Secure } Sv C (\text{Msg } [a\text{Key } K, a\text{Agt } A, a\text{Num } T]) \in \text{abs}\text{-msg } H$

|  $\text{am}\text{-M3}:$

$\text{Crypt } K \{\{\text{Agent } A, \text{Number } T\}\} \in H$

$\Longrightarrow \text{dAuth } K (\text{Msg } [a\text{Agt } A, a\text{Num } T]) \in \text{abs}\text{-msg } H$

|  $\text{am}\text{-M4}:$

$\text{Crypt } K (\text{Number } T) \in H$

$\Longrightarrow \text{dAuth } K (\text{Msg } [a\text{Num } T]) \in \text{abs}\text{-msg } H$

R23: The simulation relation. This is a data refinement of the insecure and secure channels of refinement 2.

**definition**

$R23\text{-msgs} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23\text{-msgs} \equiv \{(s, t). \text{abs-msg (parts (IK t))} \subseteq \text{chan } s \}$

**definition**

$R23\text{-keys} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23\text{-keys} \equiv \{(s, t). \forall KK K. KK \subseteq \text{range ses}K \longrightarrow$   
 $\text{Key } K \in \text{analz (Key'KK} \cup (IK t)) \longleftrightarrow \text{aKey } K \in \text{extr (aKey'KK} \cup \text{ik0)} (\text{chan } s)$   
 $\}$

**definition**

$R23\text{-non} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23\text{-non} \equiv \{(s, t). \forall KK N. KK \subseteq \text{range ses}K \longrightarrow$   
 $\text{Nonce } N \in \text{analz (Key'KK} \cup (IK t)) \longleftrightarrow \text{aNon } N \in \text{extr (aKey'KK} \cup \text{ik0)} (\text{chan } s)$   
 $\}$

**definition**

$R23\text{-pres} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23\text{-pres} \equiv \{(s, t). \text{runs } s = \text{runs } t \wedge \text{leak } s = \text{leak } t \wedge \text{clk } s = \text{clk } t \wedge \text{cache } s = \text{cache } t \}$

**definition**

$R23 :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23 \equiv R23\text{-msgs} \cap R23\text{-keys} \cap R23\text{-non} \cap R23\text{-pres}$

**lemmas**  $R23\text{-defs} =$

$R23\text{-def } R23\text{-msgs-def } R23\text{-keys-def } R23\text{-non-def } R23\text{-pres-def}$

The mediator function is the identity here.

**definition**

$\text{med}32 :: m3\text{-obs} \Rightarrow m2\text{-obs} \text{ where}$   
 $\text{med}32 \equiv \text{id}$

**lemmas**  $R23\text{-msgsI} = R23\text{-msgs-def [THEN rel-def-to-intro, simplified, rule-format]$

**lemmas**  $R23\text{-msgsE} [\text{elim}] = R23\text{-msgs-def [THEN rel-def-to-elim, simplified, rule-format]$

**lemmas**  $R23\text{-msgsE}' [\text{elim}] = R23\text{-msgs-def [THEN rel-def-to-dest, simplified, rule-format, THEN subsetD]$

**lemmas**  $R23\text{-keysI} = R23\text{-keys-def [THEN rel-def-to-intro, simplified, rule-format]$

**lemmas**  $R23\text{-keysE} [\text{elim}] = R23\text{-keys-def [THEN rel-def-to-elim, simplified, rule-format]$

**lemmas**  $R23\text{-nonI} = R23\text{-non-def [THEN rel-def-to-intro, simplified, rule-format]$

**lemmas**  $R23\text{-nonE} [\text{elim}] = R23\text{-non-def [THEN rel-def-to-elim, simplified, rule-format]$

**lemmas**  $R23\text{-presI} = R23\text{-pres-def [THEN rel-def-to-intro, simplified, rule-format]$

**lemmas**  $R23\text{-presE} [\text{elim}] = R23\text{-pres-def [THEN rel-def-to-elim, simplified, rule-format]$

**lemmas**  $R23\text{-intros} = R23\text{-msgsI } R23\text{-keysI } R23\text{-nonI } R23\text{-presI}$

Simplifier lemmas for various instantiations (keys and nonces).

**lemmas**  $R23\text{-keys-simp} = R23\text{-keys-def [THEN rel-def-to-dest, simplified, rule-format]$

**lemmas**  $R23\text{-keys-simps} =$

$R23\text{-keys-simp}$

*R23-keys-simp* [where  $KK = \{\}$ , simplified]  
*R23-keys-simp* [where  $KK = \{K\}$  for  $K$ , simplified]  
*R23-keys-simp* [where  $KK = \text{insert } K' \text{ } KK$  for  $K' \text{ } KK$ , simplified, OF - conjI]

**lemmas** *R23-non-simp* = *R23-non-def* [THEN rel-def-to-dest, simplified, rule-format]

**lemmas** *R23-non-simps* =

*R23-non-simp*  
*R23-non-simp* [where  $KK = \{\}$ , simplified]  
*R23-non-simp* [where  $KK = \{K\}$  for  $K$ , simplified]  
*R23-non-simp* [where  $KK = \text{insert } K \text{ } KK$  for  $K \text{ } KK$ , simplified, OF - conjI]

**lemmas** *R23-simps* = *R23-keys-simps* *R23-non-simps*

## General lemmas

General facts about *abs-msg*

**declare** *abs-msg.intros* [intro!]

**declare** *abs-msg.cases* [elim!]

**lemma** *abs-msg-empty*:  $\text{abs-msg } \{\} = \{\}$   
 <proof>

**lemma** *abs-msg-Un* [simp]:  
 $\text{abs-msg } (G \cup H) = \text{abs-msg } G \cup \text{abs-msg } H$   
 <proof>

**lemma** *abs-msg-mono* [elim]:  
 $\llbracket m \in \text{abs-msg } G; G \subseteq H \rrbracket \implies m \in \text{abs-msg } H$   
 <proof>

**lemma** *abs-msg-insert-mono* [intro]:  
 $\llbracket m \in \text{abs-msg } H \rrbracket \implies m \in \text{abs-msg } (\text{insert } m' \text{ } H)$   
 <proof>

Facts about *abs-msg* concerning abstraction of fakeable messages. This is crucial for proving the refinement of the intruder event.

**lemma** *abs-msg-DY-subset-fakeable*:  
 $\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; (s, t) \in R23\text{-non}; t \in m3\text{-inv1-lkeysec} \rrbracket$   
 $\implies \text{abs-msg } (\text{synth } (\text{analz } (IK \text{ } t))) \subseteq \text{fake } ik0 \text{ } (\text{dom } (\text{runs } s)) \text{ } (\text{chan } s)$   
 <proof>

## Refinement proof

Pair decomposition. These were set to **elim!**, which is too aggressive here.

**declare** *MPair-analz* [rule del, elim]

**declare** *MPair-parts* [rule del, elim]

Protocol events.

**lemma** *PO-m3-step1-refines-m2-step1*:  
 {*R23*}  
 $(m2\text{-step1 } Ra \text{ } A \text{ } B \text{ } Na), (m3\text{-step1 } Ra \text{ } A \text{ } B \text{ } Na)$

{> R23}  
<proof>

**lemma** *PO-m3-step2-refines-m2-step2*:

{R23}  
(m2-step2 Rb A B), (m3-step2 Rb A B)  
{> R23}  
<proof>

**lemma** *PO-m3-step3-refines-m2-step3*:

{R23  $\cap$  (m2-inv3a-sesK-compr)  $\times$  (m3-inv7a-sesK-compr  $\cap$  m3-inv1-lkeysec)}  
(m2-step3 Rs A B Kab Na Ts), (m3-step3 Rs A B Kab Na Ts)  
{> R23}  
<proof>

**lemma** *PO-m3-step4-refines-m2-step4*:

{R23  $\cap$  UNIV  $\times$  (m3-inv1-lkeysec) }  
(m2-step4 Ra A B Na Kab Ts Ta), (m3-step4 Ra A B Na Kab Ts Ta)  
{> R23}  
<proof>

**lemma** *PO-m3-step5-refines-m2-step5*:

{R23}  
(m2-step5 Rb A B Kab Ts Ta), (m3-step5 Rb A B Kab Ts Ta)  
{> R23}  
<proof>

**lemma** *PO-m3-step6-refines-m2-step6*:

{R23}  
(m2-step6 Ra A B Na Kab Ts Ta), (m3-step6 Ra A B Na Kab Ts Ta)  
{> R23}  
<proof>

**lemma** *PO-m3-tick-refines-m2-tick*:

{R23}  
(m2-tick T), (m3-tick T)  
{>R23}  
<proof>

**lemma** *PO-m3-purge-refines-m2-purge*:

{R23}  
(m2-purge A), (m3-purge A)  
{>R23}  
<proof>

Intruder events.

**lemma** *PO-m3-leak-refines-m2-leak*:

{R23}  
(m2-leak Rs A B Na Ts), (m3-leak Rs A B Na Ts)  
{>R23}  
<proof>

**lemma** *PO-m3-DY-fake-refines-m2-fake*:

$\{R23 \cap UNIV \times m3\text{-inv1-lkeysec}\}$

$m2\text{-fake}, m3\text{-DY-fake}$

$\{> R23\}$

$\langle\text{proof}\rangle$

All together now...

**lemmas** *PO-m3-trans-refines-m2-trans* =

*PO-m3-step1-refines-m2-step1 PO-m3-step2-refines-m2-step2*

*PO-m3-step3-refines-m2-step3 PO-m3-step4-refines-m2-step4*

*PO-m3-step5-refines-m2-step5 PO-m3-step6-refines-m2-step6*

*PO-m3-tick-refines-m2-tick PO-m3-purge-refines-m2-purge*

*PO-m3-leak-refines-m2-leak PO-m3-DY-fake-refines-m2-fake*

**lemma** *PO-m3-refines-init-m2* [iff]:

$\text{init } m3 \subseteq R23 \text{“}(\text{init } m2)$

$\langle\text{proof}\rangle$

**lemma** *PO-m3-refines-trans-m2* [iff]:

$\{R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv7a-sesK-compr} \cap m3\text{-inv1-lkeysec})\}$

$(\text{trans } m2), (\text{trans } m3)$

$\{> R23\}$

$\langle\text{proof}\rangle$

**lemma** *PO-m3-observation-consistent* [iff]:

$\text{obs-consistent } R23 \text{ med32 } m2 \ m3$

$\langle\text{proof}\rangle$

Refinement result.

**lemma** *m3-refines-m2* [iff]:

*refines*

$(R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv1-lkeysec}))$

$\text{med32 } m2 \ m3$

$\langle\text{proof}\rangle$

**lemma** *m3-implements-m2* [iff]:

*implements med32 m2 m3*

$\langle\text{proof}\rangle$

### 3.6.7 Inherited invariants

**inv3 (derived): Key secrecy for initiator**

**definition**

$m3\text{-inv3-ikk-init} :: m3\text{-state set}$

**where**

$m3\text{-inv3-ikk-init} \equiv \{s. \forall A \ B \ Ra \ K \ Ts \ nl.$

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], aKey \ K \ \# \ aNum \ Ts \ \# \ nl) \longrightarrow A \in \text{good} \longrightarrow B \in \text{good} \longrightarrow$

$\text{Key } K \in \text{analz } (IK \ s) \longrightarrow$

$(K, A, B, Ra\$na, Ts) \in \text{leak } s$

$\}$

**lemmas**  $m3\text{-inv3}\text{-ikk}\text{-init}I = m3\text{-inv3}\text{-ikk}\text{-init}\text{-def}$  [THEN setc-def-to-intro, rule-format]  
**lemmas**  $m3\text{-inv3}\text{-ikk}\text{-init}E$  [elim] =  $m3\text{-inv3}\text{-ikk}\text{-init}\text{-def}$  [THEN setc-def-to-elim, rule-format]  
**lemmas**  $m3\text{-inv3}\text{-ikk}\text{-init}D = m3\text{-inv3}\text{-ikk}\text{-init}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

**lemma**  $PO\text{-}m3\text{-inv3}\text{-ikk}\text{-init}$ : reach  $m3 \subseteq m3\text{-inv3}\text{-ikk}\text{-init}$   
 <proof>

#### inv4 (derived): Key secrecy for responder

##### definition

$m3\text{-inv4}\text{-ikk}\text{-resp} :: m3\text{-state set}$

##### where

$m3\text{-inv4}\text{-ikk}\text{-resp} \equiv \{s. \forall A B Rb K Ts nl.$

$runs\ s\ Rb = Some\ (Resp, [A, B], aKey\ K\ \# aNum\ Ts\ \# nl) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$

$Key\ K \in analz\ (IK\ s) \longrightarrow$

$(\exists Na. (K, A, B, Na, Ts) \in leak\ s)$

$\}$

**lemmas**  $m3\text{-inv4}\text{-ikk}\text{-resp}I = m3\text{-inv4}\text{-ikk}\text{-resp}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m3\text{-inv4}\text{-ikk}\text{-resp}E$  [elim] =  $m3\text{-inv4}\text{-ikk}\text{-resp}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m3\text{-inv4}\text{-ikk}\text{-resp}D = m3\text{-inv4}\text{-ikk}\text{-resp}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

**lemma**  $PO\text{-}m3\text{-inv4}\text{-ikk}\text{-resp}$ : reach  $m3 \subseteq m3\text{-inv4}\text{-ikk}\text{-resp}$   
 <proof>

end

## 3.7 Core Kerberos 5 (L3)

**theory**  $m3\text{-kerberos5}$  imports  $m2\text{-kerberos}$  ../Refinement/Message  
**begin**

We model the core Kerberos 5 protocol:

- M1.  $A \rightarrow S : A, B, Na$
- M2.  $S \rightarrow A : \{Kab, B, Ts, Na\}_{Kas}, \{Kab, A, Ts\}_{Kbs}$
- M3.  $A \rightarrow B : \{A, Ta\}_{Kab}, \{Kab, A, Ts\}_{Kbs}$
- M4.  $B \rightarrow A : \{Ta\}_{Kab}$

Proof tool configuration. Avoid annoying automatic unfolding of  $dom$ .

**declare**  $domIff$  [simp, iff del]

### 3.7.1 Setup

Now we can define the initial key knowledge.

**overloading**  $ltkSetup'$   $\equiv$   $ltkSetup$  **begin**

**definition**  $ltkSetup\text{-def}$ :  $ltkSetup' \equiv \{(sharK\ C, A) \mid C\ A. A = C \vee A = Sv\}$

**end**

**lemma** *corrKey-shrK-bad* [*simp*]:  $\text{corrKey} = \text{shrK}^{\text{bad}}$   
 ⟨*proof*⟩

### 3.7.2 State

The secure channels are star-shaped to/from the server. Therefore, we have only one agent in the relation.

**record**  $m3\text{-state} = m1\text{-state} +$   
 $IK :: \text{msg set}$  — intruder knowledge

Observable state:  $\text{runs}$ ,  $m1\text{-state.leak}$ ,  $\text{clk}$ , and  $\text{cache}$ .

**type-synonym**  
 $m3\text{-obs} = m2\text{-obs}$

**definition**  
 $m3\text{-obs} :: m3\text{-state} \Rightarrow m3\text{-obs}$  **where**  
 $m3\text{-obs } s \equiv \langle \text{runs} = \text{runs } s, \text{leak} = \text{leak } s, \text{clk} = \text{clk } s, \text{cache} = \text{cache } s \rangle$

**type-synonym**  
 $m3\text{-pred} = m3\text{-state set}$

**type-synonym**  
 $m3\text{-trans} = (m3\text{-state} \times m3\text{-state}) \text{ set}$

### 3.7.3 Events

Protocol events.

**definition** — by  $A$ , refines  $m2\text{-step1}$   
 $m3\text{-step1} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}] \Rightarrow m3\text{-trans}$   
**where**  
 $m3\text{-step1 } Ra \ A \ B \ Na \equiv \{(s, s1).$   
 — guards:  
 $Ra \notin \text{dom } (\text{runs } s) \wedge$  —  $Ra$  is fresh  
 $Na = Ra\$na \wedge$  — generated nonce  
 — actions:  
 $s1 = s \langle$   
 $\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [])),$   
 $IK := \text{insert } \{\text{Agent } A, \text{Agent } B, \text{Nonce } Na\} (IK \ s)$  — send M1  
 $\rangle$   
 $\}$

**definition** — by  $B$ , refines  $m2\text{-step2}$   
 $m3\text{-step2} :: [\text{rid-}t, \text{agent}, \text{agent}] \Rightarrow m3\text{-trans}$   
**where**  
 $m3\text{-step2} \equiv m1\text{-step2}$

**definition** — by  $\text{Server}$ , refines  $m2\text{-step3}$   
 $m3\text{-step3} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}, \text{nonce}, \text{time}] \Rightarrow m3\text{-trans}$   
**where**  
 $m3\text{-step3 } Rs \ A \ B \ Kab \ Na \ Ts \equiv \{(s, s1).$

— guards:  
 $R_s \notin \text{dom}(\text{runs } s) \wedge$  — fresh server run  
 $K_{ab} = \text{sesK}(R_s \$ sk) \wedge$  — fresh session key

$\{\{ \text{Agent } A, \text{Agent } B, \text{Nonce } Na \} \in IK \ s \wedge$  — recv  $M1$   
 $T_s = \text{clk } s \wedge$  — fresh timestamp

— actions:  
— record session key and send  $M2$   
 $s1 = s(\langle$   
 $\text{runs} := (\text{runs } s)(R_s \mapsto (\text{Serv}, [A, B], [aNon\ Na, aNum\ Ts])),$   
 $IK := \text{insert } \{\{ \text{Crypt}(\text{shrK } A) \{ \text{Key } K_{ab}, \text{Agent } B, \text{Number } Ts, \text{Nonce } Na \},$   
 $\text{Crypt}(\text{shrK } B) \{ \text{Key } K_{ab}, \text{Agent } A, \text{Number } Ts \} \}$   
 $(IK \ s)$   
 $\rangle$   
 $\}$

**definition** — by  $A$ , refines  $m2\text{-step}4$   
 $m3\text{-step}4 :: [\text{rid-t}, \text{agent}, \text{agent}, \text{nonce}, \text{key}, \text{time}, \text{time}, \text{msg}] \Rightarrow m3\text{-trans}$   
**where**  
 $m3\text{-step}4 \text{ Ra } A \ B \ Na \ K_{ab} \ Ts \ Ta \ X \equiv \{(s, s1)\}.$

— guards:  
 $\text{runs } s \text{ Ra} = \text{Some}(\text{Init}, [A, B], []) \wedge$  — key not yet recv'd  
 $Na = \text{Ra} \$ na \wedge$  — generated nonce

$\{\{ \text{Crypt}(\text{shrK } A)$  — recv  $M2$   
 $\{ \text{Key } K_{ab}, \text{Agent } B, \text{Number } Ts, \text{Nonce } Na \}, X \} \in IK \ s \wedge$

— read current time  
 $Ta = \text{clk } s \wedge$

— check freshness of session key  
 $\text{clk } s < Ts + Ls \wedge$

— actions:  
— record session key and send  $M3$   
 $s1 = s(\langle$   
 $\text{runs} := (\text{runs } s)(\text{Ra} \mapsto (\text{Init}, [A, B], [aKey\ K_{ab}, aNum\ Ts, aNum\ Ta])),$   
 $IK := \text{insert } \{\{ \text{Crypt } K_{ab} \{ \text{Agent } A, \text{Number } Ta \}, X \} (IK \ s) \text{ — } M3$   
 $\rangle$   
 $\}$

**definition** — by  $B$ , refines  $m2\text{-step}5$   
 $m3\text{-step}5 :: [\text{rid-t}, \text{agent}, \text{agent}, \text{key}, \text{time}, \text{time}] \Rightarrow m3\text{-trans}$   
**where**

$m3\text{-step}5 \text{ Rb } A \ B \ K_{ab} \ Ts \ Ta \equiv \{(s, s1)\}.$

— guards:  
 $\text{runs } s \text{ Rb} = \text{Some}(\text{Resp}, [A, B], []) \wedge$  — key not yet recv'd

$\{\{ \text{Crypt } K_{ab} \{ \text{Agent } A, \text{Number } Ta \},$  — recv  $M3$   
 $\text{Crypt}(\text{shrK } B) \{ \text{Key } K_{ab}, \text{Agent } A, \text{Number } Ts \} \} \in IK \ s \wedge$

— ensure freshness of session key  
 $clk\ s < Ts + Ls \wedge$

— check authenticator's validity and replay; 'replays' with fresh authenticator ok!  
 $clk\ s < Ta + La \wedge$   
 $(B, Kab, Ta) \notin cache\ s \wedge$

— actions:  
— record session key  
 $s1 = s(|$   
 $runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])),$   
 $cache := insert\ (B, Kab, Ta)\ (cache\ s),$   
 $IK := insert\ (Crypt\ Kab\ (Number\ Ta))\ (IK\ s) \quad \text{— send } M_4$   
 $|)$   
 $}$

**definition** — by  $A$ , refines  $m2\text{-}step6$

$m3\text{-}step6 :: [rid\text{-}t, agent, agent, nonce, key, time, time] \Rightarrow m3\text{-}trans$

**where**

$m3\text{-}step6\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta \equiv \{(s, s')\}.$

— guards:

$runs\ s\ Ra = Some\ (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta]) \wedge$  — knows key

$Na = Ra\$na \wedge$  — generated nonce

$clk\ s < Ts + Ls \wedge$  — check session key's recentness

$Crypt\ Kab\ (Number\ Ta) \in IK\ s \wedge$  — recv  $M_4$

— actions:

$s' = s(|$   
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta, END]))$   
 $|)$

}

Clock tick event

**definition** — refines  $m2\text{-}tick$

$m3\text{-}tick :: time \Rightarrow m3\text{-}trans$

**where**

$m3\text{-}tick \equiv m1\text{-}tick$

Purge event: purge cache of expired timestamps

**definition** — refines  $m2\text{-}purge$

$m3\text{-}purge :: agent \Rightarrow m3\text{-}trans$

**where**

$m3\text{-}purge \equiv m1\text{-}purge$

Session key compromise.

**definition** — refines  $m2\text{-}leak$

$m3\text{-}leak :: [rid\text{-}t, agent, agent, nonce, time] \Rightarrow m3\text{-}trans$

**where**

$m3\text{-}leak\ Rs\ A\ B\ Na\ Ts \equiv \{(s, s1)\}.$

— guards:

$runs\ s\ Rs = Some\ (Serv, [A, B], [aNon\ Na, aNum\ Ts]) \wedge$

$(clk\ s \geq Ts + Ls) \wedge$  — only compromise 'old' session keys

— actions:  
 — record session key as leaked and add it to intruder knowledge  
 $s1 = s(| leak := insert (sesK (Rs\$sk), A, B, Na, Ts) (leak\ s),$   
 $IK := insert (Key (sesK (Rs\$sk))) (IK\ s) |)$   
 $\}$

Intruder fake event. The following "Dolev-Yao" event generates all intruder-derivable messages.

**definition** — refines *m2-fake*

$m3-DY-fake :: m3-trans$

**where**

$m3-DY-fake \equiv \{(s, s1).\}$

— actions:  
 $s1 = s(| IK := synth (analz (IK\ s)) |)$  — take DY closure  
 $\}$

### 3.7.4 Transition system

**definition**

$m3-init :: m3-pred$

**where**

$m3-init \equiv \{(|$   
 $runs = Map.empty,$   
 $leak = shrK^bad \times \{undefined\},$   
 $clk = 0,$   
 $cache = \{\},$   
 $IK = Key^shrK^bad$   
 $|)\}$

**definition**

$m3-trans :: m3-trans$  **where**

$m3-trans \equiv (\bigcup A\ B\ Ra\ Rb\ Rs\ Na\ Kab\ Ts\ Ta\ T\ X.$

$m3-step1\ Ra\ A\ B\ Na \cup$

$m3-step2\ Rb\ A\ B \cup$

$m3-step3\ Rs\ A\ B\ Kab\ Na\ Ts \cup$

$m3-step4\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta\ X \cup$

$m3-step5\ Rb\ A\ B\ Kab\ Ts\ Ta \cup$

$m3-step6\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta \cup$

$m3-tick\ T \cup$

$m3-purge\ A \cup$

$m3-leak\ Rs\ A\ B\ Na\ Ts \cup$

$m3-DY-fake \cup$

$Id$

$)$

**definition**

$m3 :: (m3-state, m3-obs) spec$  **where**

$m3 \equiv (|$

$init = m3-init,$

$trans = m3-trans,$

```

    obs = m3-obs
  )

```

```

lemmas m3-loc-defs =
  m3-def m3-init-def m3-trans-def m3-obs-def
  m3-step1-def m3-step2-def m3-step3-def m3-step4-def m3-step5-def
  m3-step6-def m3-tick-def m3-purge-def m3-leak-def m3-DY-fake-def

```

```

lemmas m3-defs = m3-loc-defs m2-defs

```

### 3.7.5 Invariants

Specialized injection that we can apply more aggressively.

```

lemmas analz-Inj-IK = analz.Inj [where H=IK s for s]
lemmas parts-Inj-IK = parts.Inj [where H=IK s for s]

```

```

declare parts-Inj-IK [dest!]

```

```

declare analz-into-parts [dest]

```

#### inv1: Secrecy of pre-distributed shared keys

**definition**

```

  m3-inv1-lkeysec :: m3-pred

```

**where**

```

  m3-inv1-lkeysec ≡ {s. ∀ C.
    (Key (shrK C) ∈ parts (IK s) → C ∈ bad) ∧
    (C ∈ bad → Key (shrK C) ∈ IK s)
  }

```

```

lemmas m3-inv1-lkeysecI = m3-inv1-lkeysec-def [THEN setc-def-to-intro, rule-format]

```

```

lemmas m3-inv1-lkeysecE [elim] = m3-inv1-lkeysec-def [THEN setc-def-to-elim, rule-format]

```

```

lemmas m3-inv1-lkeysecD = m3-inv1-lkeysec-def [THEN setc-def-to-dest, rule-format]

```

Invariance proof.

```

lemma PO-m3-inv1-lkeysec-init [iff]:

```

```

  init m3 ⊆ m3-inv1-lkeysec

```

```

⟨proof⟩

```

```

lemma PO-m3-inv1-lkeysec-trans [iff]:

```

```

  {m3-inv1-lkeysec} trans m3 {> m3-inv1-lkeysec}

```

```

⟨proof⟩

```

```

lemma PO-m3-inv1-lkeysec [iff]: reach m3 ⊆ m3-inv1-lkeysec

```

```

⟨proof⟩

```

Useful simplifier lemmas

```

lemma m3-inv1-lkeysec-for-parts [simp]:

```

```

  ⟦ s ∈ m3-inv1-lkeysec ⟧ ⇒ Key (shrK C) ∈ parts (IK s) ↔ C ∈ bad

```

```

⟨proof⟩

```

```

lemma m3-inv1-lkeysec-for-analz [simp]:

```

$\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies \text{Key} (\text{shr}K C) \in \text{analz} (IK s) \longleftrightarrow C \in \text{bad}$   
 <proof>

## inv2: Session keys not used to encrypt other session keys

Session keys are not used to encrypt other keys. Proof requires generalization to sets of session keys.

NOTE: This invariant will be inherited from the corresponding L2 invariant using the simulation relation.

### definition

$m3\text{-inv2-sesK-compr} :: m3\text{-pred}$

### where

$m3\text{-inv2-sesK-compr} \equiv \{s. \forall K KK.$

$KK \subseteq \text{range ses}K \longrightarrow$

$(\text{Key } K \in \text{analz} (\text{Key } KK \cup (IK s))) = (K \in KK \vee \text{Key } K \in \text{analz} (IK s))$

$\}$

**lemmas**  $m3\text{-inv2-sesK-compr}I = m3\text{-inv2-sesK-compr-def} [THEN \text{setc-def-to-intro}, \text{rule-format}]$

**lemmas**  $m3\text{-inv2-sesK-compr}E = m3\text{-inv2-sesK-compr-def} [THEN \text{setc-def-to-elim}, \text{rule-format}]$

**lemmas**  $m3\text{-inv2-sesK-compr}D = m3\text{-inv2-sesK-compr-def} [THEN \text{setc-def-to-dest}, \text{rule-format}]$

Additional lemma

**lemmas**  $\text{insert-commute-}Key = \text{insert-commute} [\text{where } x=Key \ K \ \text{for } K]$

**lemmas**  $m3\text{-inv2-sesK-compr-simps} =$

$m3\text{-inv2-sesK-compr}D$

$m3\text{-inv2-sesK-compr}D [\text{where } KK=\text{insert } Kab \ KK \ \text{for } Kab \ KK, \text{ simplified}]$

$m3\text{-inv2-sesK-compr}D [\text{where } KK=\{Kab\} \ \text{for } Kab, \text{ simplified}]$

$\text{insert-commute-}Key$

## 3.7.6 Refinement

### Message abstraction and simulation relation

Abstraction function on sets of messages.

### inductive-set

$\text{abs-msg} :: \text{msg set} \Rightarrow \text{chmsg set}$

**for**  $H :: \text{msg set}$

### where

$\text{am-M1}:$

$\{\text{Agent } A, \text{Agent } B, \text{Nonce } N\} \in H$

$\implies \text{Insec } A \ B \ (\text{Msg } [a\text{Non } N]) \in \text{abs-msg } H$

|  $\text{am-M2a}:$

$\text{Crypt } (\text{shr}K C) \ \{\text{Key } K, \text{Agent } B, \text{Number } T, \text{Nonce } N\} \in H$

$\implies \text{Secure } Sv \ C \ (\text{Msg } [a\text{Key } K, a\text{Agt } B, a\text{Num } T, a\text{Non } N]) \in \text{abs-msg } H$

|  $\text{am-M2b}:$

$\text{Crypt } (\text{shr}K C) \ \{\text{Key } K, \text{Agent } A, \text{Number } T\} \in H$

$\implies \text{Secure } Sv \ C \ (\text{Msg } [a\text{Key } K, a\text{Agt } A, a\text{Num } T]) \in \text{abs-msg } H$

|  $\text{am-M3}:$

$\text{Crypt } K \ \{\text{Agent } A, \text{Number } T\} \in H$

$\implies \text{dAuth } K \ (\text{Msg } [a\text{Agt } A, a\text{Num } T]) \in \text{abs-msg } H$

| *am-M4*:  
 Crypt  $K$  (Number  $T$ )  $\in H$   
 $\implies$  *dAuth*  $K$  (Msg [*aNum*  $T$ ])  $\in$  *abs-msg*  $H$

R23: The simulation relation. This is a data refinement of the insecure and secure channels of refinement 2.

**definition**

*R23-msgs* :: (*m2-state*  $\times$  *m3-state*) set **where**  
*R23-msgs*  $\equiv$   $\{(s, t). \text{abs-msg (parts (IK t))} \subseteq \text{chan } s\}$

**definition**

*R23-keys* :: (*m2-state*  $\times$  *m3-state*) set **where**  
*R23-keys*  $\equiv$   $\{(s, t). \forall KK K. KK \subseteq \text{range sesK} \longrightarrow$   
 Key  $K \in \text{analz (Key'KK} \cup (IK t)) \longleftrightarrow \text{aKey } K \in \text{extr (aKey'KK} \cup ik0) (\text{chan } s)$   
 $\}$

**definition**

*R23-non* :: (*m2-state*  $\times$  *m3-state*) set **where**  
*R23-non*  $\equiv$   $\{(s, t). \forall KK N. KK \subseteq \text{range sesK} \longrightarrow$   
 Nonce  $N \in \text{analz (Key'KK} \cup (IK t)) \longleftrightarrow \text{aNon } N \in \text{extr (aKey'KK} \cup ik0) (\text{chan } s)$   
 $\}$

**definition**

*R23-pres* :: (*m2-state*  $\times$  *m3-state*) set **where**  
*R23-pres*  $\equiv$   $\{(s, t). \text{runs } s = \text{runs } t \wedge \text{leak } s = \text{leak } t \wedge \text{clk } s = \text{clk } t \wedge \text{cache } s = \text{cache } t\}$

**definition**

*R23* :: (*m2-state*  $\times$  *m3-state*) set **where**  
*R23*  $\equiv$  *R23-msgs*  $\cap$  *R23-keys*  $\cap$  *R23-non*  $\cap$  *R23-pres*

**lemmas** *R23-defs* =

*R23-def* *R23-msgs-def* *R23-keys-def* *R23-non-def* *R23-pres-def*

The mediator function is the identity here.

**definition**

*med32* :: *m3-obs*  $\Rightarrow$  *m2-obs* **where**  
*med32*  $\equiv$  *id*

**lemmas** *R23-msgsI* = *R23-msgs-def* [*THEN rel-def-to-intro, simplified, rule-format*]

**lemmas** *R23-msgsE* [*elim*] = *R23-msgs-def* [*THEN rel-def-to-elim, simplified, rule-format*]

**lemmas** *R23-msgsE'* [*elim*] = *R23-msgs-def* [*THEN rel-def-to-dest, simplified, rule-format, THEN subsetD*]

**lemmas** *R23-keysI* = *R23-keys-def* [*THEN rel-def-to-intro, simplified, rule-format*]

**lemmas** *R23-keysE* [*elim*] = *R23-keys-def* [*THEN rel-def-to-elim, simplified, rule-format*]

**lemmas** *R23-nonI* = *R23-non-def* [*THEN rel-def-to-intro, simplified, rule-format*]

**lemmas** *R23-nonE* [*elim*] = *R23-non-def* [*THEN rel-def-to-elim, simplified, rule-format*]

**lemmas** *R23-presI* = *R23-pres-def* [*THEN rel-def-to-intro, simplified, rule-format*]

**lemmas** *R23-presE* [*elim*] = *R23-pres-def* [*THEN rel-def-to-elim, simplified, rule-format*]

**lemmas** *R23-intros* = *R23-msgsI R23-keysI R23-nonI R23-presI*

Simplifier lemmas for various instantiations (keys and nonces).

**lemmas** *R23-keys-simp* = *R23-keys-def* [*THEN rel-def-to-dest, simplified, rule-format*]

**lemmas** *R23-keys-simps* =

*R23-keys-simp*

*R23-keys-simp* [**where** *KK*={}, *simplified*]

*R23-keys-simp* [**where** *KK*={*K*} **for** *K'*, *simplified*]

*R23-keys-simp* [**where** *KK*=*insert K' KK* **for** *K' KK*, *simplified, OF - conjI*]

**lemmas** *R23-non-simp* = *R23-non-def* [*THEN rel-def-to-dest, simplified, rule-format*]

**lemmas** *R23-non-simps* =

*R23-non-simp*

*R23-non-simp* [**where** *KK*={}, *simplified*]

*R23-non-simp* [**where** *KK*={*K*} **for** *K*, *simplified*]

*R23-non-simp* [**where** *KK*=*insert K KK* **for** *K KK*, *simplified, OF - conjI*]

**lemmas** *R23-simps* = *R23-keys-simps R23-non-simps*

## General lemmas

General facts about *abs-msg*

**declare** *abs-msg.intros* [*intro!*]

**declare** *abs-msg.cases* [*elim!*]

**lemma** *abs-msg-empty*: *abs-msg {} = {}*

*<proof>*

**lemma** *abs-msg-Un* [*simp*]:

*abs-msg (G ∪ H) = abs-msg G ∪ abs-msg H*

*<proof>*

**lemma** *abs-msg-mono* [*elim*]:

$\llbracket m \in \text{abs-msg } G; G \subseteq H \rrbracket \implies m \in \text{abs-msg } H$

*<proof>*

**lemma** *abs-msg-insert-mono* [*intro*]:

$\llbracket m \in \text{abs-msg } H \rrbracket \implies m \in \text{abs-msg } (\text{insert } m' H)$

*<proof>*

Facts about *abs-msg* concerning abstraction of fakeable messages. This is crucial for proving the refinement of the intruder event.

**lemma** *abs-msg-DY-subset-fakeable*:

$\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; (s, t) \in R23\text{-non}; t \in m3\text{-inv1-lkeysec} \rrbracket$

$\implies \text{abs-msg } (\text{synth } (\text{analz } (IK\ t))) \subseteq \text{fake ik0 } (\text{dom } (\text{runs } s)) (\text{chan } s)$

*<proof>*

## Refinement proof

Pair decomposition. These were set to **elim!**, which is too aggressive here.

**declare** *MPair-analz* [rule del, elim]  
**declare** *MPair-parts* [rule del, elim]

Protocol events.

**lemma** *PO-m3-step1-refines-m2-step1*:  
 $\{R23\}$   
 $(m2\text{-step1 } Ra \ A \ B \ Na), (m3\text{-step1 } Ra \ A \ B \ Na)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-step2-refines-m2-step2*:  
 $\{R23\}$   
 $(m2\text{-step2 } Rb \ A \ B), (m3\text{-step2 } Rb \ A \ B)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-step3-refines-m2-step3*:  
 $\{R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv2-sesK-compr} \cap m3\text{-inv1-lkeysec})\}$   
 $(m2\text{-step3 } Rs \ A \ B \ Kab \ Na \ Ts), (m3\text{-step3 } Rs \ A \ B \ Kab \ Na \ Ts)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-step4-refines-m2-step4*:  
 $\{R23 \cap UNIV \times m3\text{-inv1-lkeysec}\}$   
 $(m2\text{-step4 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta), (m3\text{-step4 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta \ X)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-step5-refines-m2-step5*:  
 $\{R23\}$   
 $(m2\text{-step5 } Rb \ A \ B \ Kab \ Ts \ Ta), (m3\text{-step5 } Rb \ A \ B \ Kab \ Ts \ Ta)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-step6-refines-m2-step6*:  
 $\{R23\}$   
 $(m2\text{-step6 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta), (m3\text{-step6 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-tick-refines-m2-tick*:  
 $\{R23\}$   
 $(m2\text{-tick } T), (m3\text{-tick } T)$   
 $\{>R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-purge-refines-m2-purge*:  
 $\{R23\}$   
 $(m2\text{-purge } A), (m3\text{-purge } A)$   
 $\{>R23\}$   
 $\langle proof \rangle$

Intruder events.

**lemma** *PO-m3-leak-refines-m2-leak*:

$\{R23\}$   
 $(m2\text{-leak } Rs \ A \ B \ Na \ Ts), (m3\text{-leak } Rs \ A \ B \ Na \ Ts)$   
 $\{>R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-DY-fake-refines-m2-fake*:

$\{R23 \cap m2\text{-inv3a-sesK-compr} \times (m3\text{-inv2-sesK-compr} \cap m3\text{-inv1-lkeysec})\}$   
 $m2\text{-fake}, m3\text{-DY-fake}$   
 $\{> R23\}$   
 $\langle proof \rangle$

All together now...

**lemmas** *PO-m3-trans-refines-m2-trans* =

*PO-m3-step1-refines-m2-step1 PO-m3-step2-refines-m2-step2*  
*PO-m3-step3-refines-m2-step3 PO-m3-step4-refines-m2-step4*  
*PO-m3-step5-refines-m2-step5 PO-m3-step6-refines-m2-step6*  
*PO-m3-tick-refines-m2-tick PO-m3-purge-refines-m2-purge*  
*PO-m3-leak-refines-m2-leak PO-m3-DY-fake-refines-m2-fake*

**lemma** *PO-m3-refines-init-m2* [iff]:

$init \ m3 \subseteq R23 \text{“}(init \ m2)$   
 $\langle proof \rangle$

**lemma** *PO-m3-refines-trans-m2* [iff]:

$\{R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv2-sesK-compr} \cap m3\text{-inv1-lkeysec})\}$   
 $(trans \ m2), (trans \ m3)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-observation-consistent* [iff]:

$obs\text{-consistent } R23 \ med32 \ m2 \ m3$   
 $\langle proof \rangle$

Refinement result.

**lemma** *m3-refines-m2* [iff]:

*refines*  
 $(R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv1-lkeysec}))$   
 $med32 \ m2 \ m3$   
 $\langle proof \rangle$

**lemma** *m3-implements-m2* [iff]:

*implements*  $med32 \ m2 \ m3$   
 $\langle proof \rangle$

### 3.7.7 Inherited invariants

**inv3 (derived): Key secrecy for initiator**

**definition**

$m3\text{-inv3-ikk-init} :: m3\text{-state set}$

**where**

```

m3-inv3-ikk-init ≡ {s. ∀ A B Ra K Ts nl.
  runs s Ra = Some (Init, [A, B], aKey K # aNum Ts # nl) → A ∈ good → B ∈ good →
  Key K ∈ analz (IK s) →
  (K, A, B, Ra$na, Ts) ∈ leak s
}

```

```

lemmas m3-inv3-ikk-initI = m3-inv3-ikk-init-def [THEN setc-def-to-intro, rule-format]
lemmas m3-inv3-ikk-initE [elim] = m3-inv3-ikk-init-def [THEN setc-def-to-elim, rule-format]
lemmas m3-inv3-ikk-initD = m3-inv3-ikk-init-def [THEN setc-def-to-dest, rule-format, rotated 1]

```

```

lemma PO-m3-inv3-ikk-init: reach m3 ⊆ m3-inv3-ikk-init
⟨proof⟩

```

#### inv4 (derived): Key secrecy for responder

##### definition

```

m3-inv4-ikk-resp :: m3-state set

```

##### where

```

m3-inv4-ikk-resp ≡ {s. ∀ A B Rb K Ts nl.
  runs s Rb = Some (Resp, [A, B], aKey K # aNum Ts # nl) → A ∈ good → B ∈ good →
  Key K ∈ analz (IK s) →
  (∃ Na. (K, A, B, Na, Ts) ∈ leak s)
}

```

```

lemmas m3-inv4-ikk-respI = m3-inv4-ikk-resp-def [THEN setc-def-to-intro, rule-format]
lemmas m3-inv4-ikk-respE [elim] = m3-inv4-ikk-resp-def [THEN setc-def-to-elim, rule-format]
lemmas m3-inv4-ikk-respD = m3-inv4-ikk-resp-def [THEN setc-def-to-dest, rule-format, rotated 1]

```

```

lemma PO-m3-inv4-ikk-resp: reach m3 ⊆ m3-inv4-ikk-resp
⟨proof⟩

```

end

## 3.8 Core Kerberos 4 (L3)

```

theory m3-kerberos4 imports m2-kerberos ../Refinement/Message
begin

```

We model the core Kerberos 4 protocol:

- M1.  $A \rightarrow S : A, B$
- M2.  $S \rightarrow A : \{Kab, B, Ts, Na, \{Kab, A, Ts\}_{Kbs}\}_{Kas}$
- M3.  $A \rightarrow B : \{A, Ta\}_{Kab}, \{Kab, A, Ts\}_{Kbs}$
- M4.  $B \rightarrow A : \{Ta\}_{Kab}$

Proof tool configuration. Avoid annoying automatic unfolding of *dom*.

```

declare domIff [simp, iff del]

```

### 3.8.1 Setup

Now we can define the initial key knowledge.

**overloading**  $ltkeySetup' \equiv ltkeySetup$  **begin**

**definition**  $ltkeySetup-def: ltkeySetup' \equiv \{(sharK\ C, A) \mid C\ A.\ A = C \vee A = Sv\}$   
**end**

**lemma**  $corrKey-shrK-bad$  [*simp*]:  $corrKey = shrK'bad$   
 ⟨*proof*⟩

### 3.8.2 State

The secure channels are star-shaped to/from the server. Therefore, we have only one agent in the relation.

**record**  $m3-state = m1-state +$   
 $IK :: msg\ set$  — intruder knowledge

Observable state:  $runs$ ,  $clk$ , and  $cache$ .

**type-synonym**  
 $m3-obs = m2-obs$

**definition**  
 $m3-obs :: m3-state \Rightarrow m3-obs$  **where**  
 $m3-obs\ s \equiv (\mid runs = runs\ s, leak = leak\ s, clk = clk\ s, cache = cache\ s \mid)$

**type-synonym**  
 $m3-pred = m3-state\ set$

**type-synonym**  
 $m3-trans = (m3-state \times m3-state)\ set$

### 3.8.3 Events

Protocol events.

**definition** — by  $A$ , refines  $m2-step1$   
 $m3-step1 :: [rid-t, agent, agent, nonce] \Rightarrow m3-trans$   
**where**  
 $m3-step1\ Ra\ A\ B\ Na \equiv \{(s, s1).$   
 — guards:  
 $Ra \notin dom\ (runs\ s) \wedge$  —  $Ra$  is fresh  
 $Na = Ra\$na \wedge$  — generated nonce  
 — actions:  
 $s1 = s(\mid$   
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [])),$   
 $IK := insert\ \{\ Agent\ A, Agent\ B, Nonce\ Na\} (IK\ s)$  — send  $M1$   
 $\mid)$   
 $\}$

**definition** — by  $B$ , refines  $m2-step2$   
 $m3-step2 :: [rid-t, agent, agent] \Rightarrow m3-trans$   
**where**  
 $m3-step2 \equiv m1-step2$

**definition** — by *Server*, refines *m2-step3*

$m3\text{-step3} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}, \text{nonce}, \text{time}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step3 } Rs \ A \ B \ Kab \ Na \ Ts \equiv \{(s, s1)\}.$

— guards:

$Rs \notin \text{dom } (\text{runs } s) \wedge$  — fresh server run  
 $Kab = \text{sesK } (Rs\$sk) \wedge$  — fresh session key

$\{\!\{ \text{Agent } A, \text{Agent } B, \text{Nonce } Na \}\!\} \in IK \ s \wedge$  — rcv *M1*  
 $Ts = \text{clk } s \wedge$  — fresh timestamp

— actions:

— record session key and send *M2*

$s1 = s \{$   
 $\text{runs} := (\text{runs } s)(Rs \mapsto (\text{Serv}, [A, B], [a\text{Non } Na, a\text{Num } Ts])),$   
 $IK := \text{insert } (\text{Crypt } (\text{shrK } A)$  — send *M2*  
 $\quad \{\!\{ \text{Key } Kab, \text{Agent } B, \text{Number } Ts, \text{Nonce } Na,$   
 $\quad \text{Crypt } (\text{shrK } B) \{\!\{ \text{Key } Kab, \text{Agent } A, \text{Number } Ts \}\!\} \})$   
 $\quad (IK \ s)$   
 $\}$

**definition** — by *A*, refines *m2-step4*

$m3\text{-step4} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}, \text{time}, \text{time}, \text{msg}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step4 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta \ X \equiv \{(s, s1)\}.$

— guards:

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], []) \wedge$  — key not yet rcv'd  
 $Na = Ra\$na \wedge$  — generated nonce

$\text{Crypt } (\text{shrK } A)$  — rcv *M2*  
 $\quad \{\!\{ \text{Key } Kab, \text{Agent } B, \text{Number } Ts, \text{Nonce } Na, X \}\!\} \in IK \ s \wedge$

— read current time

$Ta = \text{clk } s \wedge$

— check freshness of session key

$\text{clk } s < Ts + Ls \wedge$

— actions:

— record session key and send *M3*

$s1 = s \{$   
 $\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [a\text{Key } Kab, a\text{Num } Ts, a\text{Num } Ta])),$   
 $IK := \text{insert } \{\!\{ \text{Crypt } Kab \{\!\{ \text{Agent } A, \text{Number } Ta \}\!\}, X \}\!\} (IK \ s)$  — *M3*  
 $\}$

**definition** — by *B*, refines *m2-step5*

$m3\text{-step5} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}, \text{time}, \text{time}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step5 } Rb \ A \ B \ Kab \ Ts \ Ta \equiv \{(s, s1)\}.$

— guards:

$runs\ s\ Rb = Some\ (Resp, [A, B], []) \wedge$  — key not yet recv'd  
 $\{Crypt\ Kab\ \{Agent\ A, Number\ Ta\},$  — recv  $M3$   
 $\quad Crypt\ (shrK\ B)\ \{Key\ Kab, Agent\ A, Number\ Ts\}\} \in IK\ s \wedge$   
— ensure freshness of session key  
 $clk\ s < Ts + Ls \wedge$   
— check authenticator's validity and replay; 'replays' with fresh authenticator ok!  
 $clk\ s < Ta + La \wedge$   
 $(B, Kab, Ta) \notin cache\ s \wedge$   
— actions:  
— record session key  
 $s1 = s(|$   
 $runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])),$   
 $cache := insert\ (B, Kab, Ta)\ (cache\ s),$   
 $IK := insert\ (Crypt\ Kab\ (Number\ Ta))\ (IK\ s)$  — send  $M4$   
 $|)$   
 $\}$

**definition** — by  $A$ , refines  $m2\text{-}step6$   
 $m3\text{-}step6 :: [rid\text{-}t, agent, agent, nonce, key, time, time] \Rightarrow m3\text{-}trans$

**where**

$m3\text{-}step6\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta \equiv \{(s, s')\}.$

— guards:

$runs\ s\ Ra = Some\ (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta]) \wedge$  — knows key

$Na = Ra\$na \wedge$  — generated nonce

$clk\ s < Ts + Ls \wedge$  — check session key's recentness

$Crypt\ Kab\ (Number\ Ta) \in IK\ s \wedge$  — recv  $M4$

— actions:

$s' = s(|$   
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta, END]))$   
 $|)$

$\}$

Clock tick event

**definition** — refines  $m2\text{-}tick$

$m3\text{-}tick :: time \Rightarrow m3\text{-}trans$

**where**

$m3\text{-}tick \equiv m1\text{-}tick$

Purge event: purge cache of expired timestamps

**definition** — refines  $m2\text{-}purge$

$m3\text{-}purge :: agent \Rightarrow m3\text{-}trans$

**where**

$m3\text{-}purge \equiv m1\text{-}purge$

Session key compromise.

**definition** — refines  $m2\text{-}leak$

$m3\text{-leak} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{time}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-leak } Rs \ A \ B \ Na \ Ts \equiv \{(s, s1)\}.$

— guards:

$runs \ s \ Rs = \text{Some } (\text{Serv}, [A, B], [a\text{Non } Na, a\text{Num } Ts]) \wedge$   
 $(\text{clk } s \geq Ts + Ls) \wedge$  — only compromise 'old' session keys!

— actions:

— record session key as leaked and add it to intruder knowledge

$s1 = s \{ \text{leak} := \text{insert } (\text{sesK } (Rs\$sk), A, B, Na, Ts) (\text{leak } s),$   
 $IK := \text{insert } (\text{Key } (\text{sesK } (Rs\$sk))) (IK \ s) \}$

}

Intruder fake event. The following "Dolev-Yao" event generates all intruder-derivable messages.

**definition** — refines  $m2\text{-fake}$

$m3\text{-DY-fake} :: m3\text{-trans}$

**where**

$m3\text{-DY-fake} \equiv \{(s, s1)\}.$

— actions:

$s1 = s \{ IK := \text{synth } (\text{analz } (IK \ s)) \}$  — take DY closure

}

### 3.8.4 Transition system

**definition**

$m3\text{-init} :: m3\text{-pred}$

**where**

$m3\text{-init} \equiv \{ \{$   
 $runs = \text{Map.empty},$   
 $leak = \text{shrK}^{\text{bad}} \times \{\text{undefined}\},$   
 $clk = 0,$   
 $cache = \{\},$   
 $IK = \text{Key}^{\text{shrK}^{\text{bad}}}$   
 $\} \}$

**definition**

$m3\text{-trans} :: m3\text{-trans}$  **where**

$m3\text{-trans} \equiv (\bigcup A \ B \ Ra \ Rb \ Rs \ Na \ Kab \ Ts \ Ta \ T \ X.$

$m3\text{-step1 } Ra \ A \ B \ Na \cup$

$m3\text{-step2 } Rb \ A \ B \cup$

$m3\text{-step3 } Rs \ A \ B \ Kab \ Na \ Ts \cup$

$m3\text{-step4 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta \ X \cup$

$m3\text{-step5 } Rb \ A \ B \ Kab \ Ts \ Ta \cup$

$m3\text{-step6 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta \cup$

$m3\text{-tick } T \cup$

$m3\text{-purge } A \cup$

$m3\text{-leak } Rs \ A \ B \ Na \ Ts \cup$

$m3\text{-DY-fake} \cup$

$Id$

)

**definition**

$m3 :: (m3\text{-state}, m3\text{-obs}) \text{ spec}$  **where**  
 $m3 \equiv \langle$   
 $\text{init} = m3\text{-init},$   
 $\text{trans} = m3\text{-trans},$   
 $\text{obs} = m3\text{-obs}$   
 $\rangle$

**lemmas**  $m3\text{-loc-defs} =$ 

$m3\text{-def } m3\text{-init-def } m3\text{-trans-def } m3\text{-obs-def}$   
 $m3\text{-step1-def } m3\text{-step2-def } m3\text{-step3-def } m3\text{-step4-def } m3\text{-step5-def}$   
 $m3\text{-step6-def } m3\text{-tick-def } m3\text{-purge-def } m3\text{-leak-def } m3\text{-DY-fake-def}$

**lemmas**  $m3\text{-defs} = m3\text{-loc-defs } m2\text{-defs}$ **3.8.5 Invariants**

Specialized injection that we can apply more aggressively.

**lemmas**  $\text{analz-Inj-IK} = \text{analz.Inj}$  [**where**  $H=IK$   $s$  **for**  $s$ ]**lemmas**  $\text{parts-Inj-IK} = \text{parts.Inj}$  [**where**  $H=IK$   $s$  **for**  $s$ ]**declare**  $\text{parts-Inj-IK}$  [ $\text{dest!}$ ]**declare**  $\text{analz-into-parts}$  [ $\text{dest}$ ]**inv4: Secrecy of pre-distributed shared keys****definition** $m3\text{-inv4-lkeysec} :: m3\text{-pred}$ **where**

$m3\text{-inv4-lkeysec} \equiv \{s. \forall C.$   
 $(\text{Key } (\text{shrK } C) \in \text{parts } (IK \ s) \longrightarrow C \in \text{bad}) \wedge$   
 $(C \in \text{bad} \longrightarrow \text{Key } (\text{shrK } C) \in IK \ s)$   
 $\}$

**lemmas**  $m3\text{-inv4-lkeysecI} = m3\text{-inv4-lkeysec-def}$  [ $\text{THEN setc-def-to-intro, rule-format}$ ]**lemmas**  $m3\text{-inv4-lkeysecE}$  [ $\text{elim}$ ] =  $m3\text{-inv4-lkeysec-def}$  [ $\text{THEN setc-def-to-elim, rule-format}$ ]**lemmas**  $m3\text{-inv4-lkeysecD} = m3\text{-inv4-lkeysec-def}$  [ $\text{THEN setc-def-to-dest, rule-format}$ ]

Invariance proof.

**lemma**  $\text{PO-}m3\text{-inv4-lkeysec-init}$  [ $\text{iff}$ ]: $\text{init } m3 \subseteq m3\text{-inv4-lkeysec}$  $\langle \text{proof} \rangle$ **lemma**  $\text{PO-}m3\text{-inv4-lkeysec-trans}$  [ $\text{iff}$ ]: $\{m3\text{-inv4-lkeysec}\} \text{ trans } m3 \{> m3\text{-inv4-lkeysec}\}$  $\langle \text{proof} \rangle$ **lemma**  $\text{PO-}m3\text{-inv4-lkeysec}$  [ $\text{iff}$ ]:  $\text{reach } m3 \subseteq m3\text{-inv4-lkeysec}$  $\langle \text{proof} \rangle$ 

Useful simplifier lemmas

**lemma** *m3-inv4-lkeysec-for-parts* [simp]:

$\llbracket s \in m3\text{-inv4-lkeysec} \rrbracket \implies \text{Key} (\text{shr}K C) \in \text{parts} (IK s) \longleftrightarrow C \in \text{bad}$   
 <proof>

**lemma** *m3-inv4-lkeysec-for-analz* [simp]:

$\llbracket s \in m3\text{-inv4-lkeysec} \rrbracket \implies \text{Key} (\text{shr}K C) \in \text{analz} (IK s) \longleftrightarrow C \in \text{bad}$   
 <proof>

## inv6: Ticket shape for honestly encrypted M2

**definition**

*m3-inv6-ticket* :: *m3-pred*

**where**

$m3\text{-inv6-ticket} \equiv \{s. \forall A B T K N X.$

$A \notin \text{bad} \longrightarrow$

$\text{Crypt} (\text{shr}K A) \llbracket \text{Key } K, \text{Agent } B, \text{Number } T, \text{Nonce } N, X \rrbracket \in \text{parts} (IK s) \longrightarrow$

$X = \text{Crypt} (\text{shr}K B) \llbracket \text{Key } K, \text{Agent } A, \text{Number } T \rrbracket \wedge K \in \text{range ses}K$

$\}$

**lemmas** *m3-inv6-ticketI* = *m3-inv6-ticket-def* [THEN *setc-def-to-intro*, *rule-format*]

**lemmas** *m3-inv6-ticketE* [elim] = *m3-inv6-ticket-def* [THEN *setc-def-to-elim*, *rule-format*]

**lemmas** *m3-inv6-ticketD* = *m3-inv6-ticket-def* [THEN *setc-def-to-dest*, *rule-format*, *rotated -1*]

Invariance proof.

**lemma** *PO-m3-inv6-ticket-init* [iff]:

$\text{init } m3 \subseteq m3\text{-inv6-ticket}$

<proof>

**lemma** *PO-m3-inv6-ticket-trans* [iff]:

$\{m3\text{-inv6-ticket} \cap m3\text{-inv4-lkeysec}\} \text{trans } m3 \{> m3\text{-inv6-ticket}\}$

<proof>

**lemma** *PO-m3-inv6-ticket* [iff]:  $\text{reach } m3 \subseteq m3\text{-inv6-ticket}$

<proof>

## inv7: Session keys not used to encrypt other session keys

Session keys are not used to encrypt other keys. Proof requires generalization to sets of session keys.

NOTE: For Kerberos 4, this invariant cannot be inherited from the corresponding L2 invariant. The simulation relation is only an implication not an equivalence.

**definition**

*m3-inv7a-sesK-compr* :: *m3-pred*

**where**

$m3\text{-inv7a-sesK-compr} \equiv \{s. \forall K KK.$

$KK \subseteq \text{range ses}K \longrightarrow$

$(\text{Key } K \in \text{analz} (\text{Key}'KK \cup (IK s))) = (K \in KK \vee \text{Key } K \in \text{analz} (IK s))$

$\}$

**lemmas** *m3-inv7a-sesK-comprI* = *m3-inv7a-sesK-compr-def* [THEN *setc-def-to-intro*, *rule-format*]

**lemmas** *m3-inv7a-sesK-comprE* = *m3-inv7a-sesK-compr-def* [THEN *setc-def-to-elim*, *rule-format*]

**lemmas** *m3-inv7a-sesK-comprD* = *m3-inv7a-sesK-compr-def* [THEN *setc-def-to-dest*, *rule-format*]

Additional lemma

**lemmas** *insert-commute-Key = insert-commute* [where  $x = \text{Key } K$  for  $K$ ]

**lemmas** *m3-inv7a-sesK-compr-simps =*  
*m3-inv7a-sesK-comprD*  
*m3-inv7a-sesK-comprD* [where  $KK = \text{insert } Kab \text{ } KK$  for  $Kab \text{ } KK$ , simplified]  
*m3-inv7a-sesK-comprD* [where  $KK = \{Kab\}$  for  $Kab$ , simplified]  
*insert-commute-Key*

Invariance proof.

**lemma** *PO-m3-inv7a-sesK-compr-step4*:  
 $\{m3\text{-inv7a-sesK-compr} \cap m3\text{-inv6-ticket} \cap m3\text{-inv4-lkeysec}\}$   
 $m3\text{-step4 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta \ X$   
 $\{> m3\text{-inv7a-sesK-compr}\}$   
 ⟨proof⟩

All together now.

**lemmas** *PO-m3-inv7a-sesK-compr-trans-lemmas =*  
*PO-m3-inv7a-sesK-compr-step4*

**lemma** *PO-m3-inv7a-sesK-compr-init* [iff]:  
 $init \ m3 \subseteq m3\text{-inv7a-sesK-compr}$   
 ⟨proof⟩

**lemma** *PO-m3-inv7a-sesK-compr-trans* [iff]:  
 $\{m3\text{-inv7a-sesK-compr} \cap m3\text{-inv6-ticket} \cap m3\text{-inv4-lkeysec}\}$   
 $trans \ m3$   
 $\{> m3\text{-inv7a-sesK-compr}\}$   
 ⟨proof⟩

**lemma** *PO-m3-inv7a-sesK-compr* [iff]:  $reach \ m3 \subseteq m3\text{-inv7a-sesK-compr}$   
 ⟨proof⟩

### inv7b: Session keys not used to encrypt nonces

Session keys are not used to encrypt nonces. The proof requires a generalization to sets of session keys.

#### definition

$m3\text{-inv7b-sesK-compr-non} :: m3\text{-pred}$

where

$m3\text{-inv7b-sesK-compr-non} \equiv \{s. \forall N \ KK.$

$KK \subseteq range \ sesK \longrightarrow (Nonce \ N \in analz \ (Key'KK \cup (IK \ s))) = (Nonce \ N \in analz \ (IK \ s))$

$\}$

**lemmas** *m3-inv7b-sesK-compr-nonI = m3-inv7b-sesK-compr-non-def* [THEN *setc-def-to-intro*, *rule-format*]  
**lemmas** *m3-inv7b-sesK-compr-nonE = m3-inv7b-sesK-compr-non-def* [THEN *setc-def-to-elim*, *rule-format*]  
**lemmas** *m3-inv7b-sesK-compr-nonD = m3-inv7b-sesK-compr-non-def* [THEN *setc-def-to-dest*, *rule-format*]

**lemmas** *m3-inv7b-sesK-compr-non-simps =*  
*m3-inv7b-sesK-compr-nonD*  
*m3-inv7b-sesK-compr-nonD* [where  $KK = \text{insert } Kab \text{ } KK$  for  $Kab \text{ } KK$ , simplified]

*m3-inv7b-sesK-compr-nonD* [where  $KK=\{Kab\}$  for  $Kab$ , simplified]  
*insert-commute-Key*

Invariance proof.

**lemma** *PO-m3-inv7b-sesK-compr-non-step3*:  
 $\{m3-inv7b-sesK-compr-non\} m3-step3 Rs A B Kab Na Ts \{> m3-inv7b-sesK-compr-non\}$   
 ⟨proof⟩

**lemma** *PO-m3-inv7b-sesK-compr-non-step4*:  
 $\{m3-inv7b-sesK-compr-non \cap m3-inv6-ticket \cap m3-inv4-lkeysec\}$   
 $m3-step4 Ra A B Na Kab Ts Ta X$   
 $\{> m3-inv7b-sesK-compr-non\}$   
 ⟨proof⟩

All together now.

**lemmas** *PO-m3-inv7b-sesK-compr-non-trans-lemmas* =  
*PO-m3-inv7b-sesK-compr-non-step3 PO-m3-inv7b-sesK-compr-non-step4*

**lemma** *PO-m3-inv7b-sesK-compr-non-init* [iff]:  
 $init m3 \subseteq m3-inv7b-sesK-compr-non$   
 ⟨proof⟩

**lemma** *PO-m3-inv7b-sesK-compr-non-trans* [iff]:  
 $\{m3-inv7b-sesK-compr-non \cap m3-inv6-ticket \cap m3-inv4-lkeysec\}$   
 $trans m3$   
 $\{> m3-inv7b-sesK-compr-non\}$   
 ⟨proof⟩

**lemma** *PO-m3-inv7b-sesK-compr-non* [iff]:  $reach m3 \subseteq m3-inv7b-sesK-compr-non$   
 ⟨proof⟩

### 3.8.6 Refinement

#### Message abstraction and simulation relation

Abstraction function on sets of messages.

**inductive-set**

$abs-msg :: msg\ set \Rightarrow chmsg\ set$

**for**  $H :: msg\ set$

**where**

*am-M1*:

$\{Agent\ A, Agent\ B, Nonce\ N\} \in H$

$\Rightarrow Insec\ A\ B\ (Msg\ [aNon\ N]) \in abs-msg\ H$

| *am-M2a*:

$Crypt\ (shrK\ C)\ \{Key\ K, Agent\ B, Number\ T, Nonce\ N, X\} \in H$

$\Rightarrow Secure\ Sv\ C\ (Msg\ [aKey\ K, aAgt\ B, aNum\ T, aNon\ N]) \in abs-msg\ H$

| *am-M2b*:

$Crypt\ (shrK\ C)\ \{Key\ K, Agent\ A, Number\ T\} \in H$

$\Rightarrow Secure\ Sv\ C\ (Msg\ [aKey\ K, aAgt\ A, aNum\ T]) \in abs-msg\ H$

| *am-M3*:

$Crypt\ K\ \{Agent\ A, Number\ T\} \in H$

$\Rightarrow dAuth\ K\ (Msg\ [aAgt\ A, aNum\ T]) \in abs-msg\ H$

| *am-M4*:  
 $\text{Crypt } K \text{ (Number } T) \in H$   
 $\implies \text{dAuth } K \text{ (Msg [aNum } T]) \in \text{abs-msg } H$

R23: The simulation relation. This is a data refinement of the insecure and secure channels of refinement 2.

**definition**

$R23\text{-msgs} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23\text{-msgs} \equiv \{(s, t). \text{abs-msg (parts (IK } t)) \subseteq \text{chan } s \}$

**definition**

$R23\text{-keys} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23\text{-keys} \equiv \{(s, t). \forall KK K. KK \subseteq \text{range sesK} \longrightarrow$   
 $\text{Key } K \in \text{analz (Key'KK} \cup \text{(IK } t)) \longrightarrow \text{aKey } K \in \text{extr (aKey'KK} \cup \text{ik0) (chan } s)$   
 $\}$

**definition**

$R23\text{-non} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23\text{-non} \equiv \{(s, t). \forall KK N. KK \subseteq \text{range sesK} \longrightarrow$   
 $\text{Nonce } N \in \text{analz (Key'KK} \cup \text{(IK } t)) \longrightarrow \text{aNon } N \in \text{extr (aKey'KK} \cup \text{ik0) (chan } s)$   
 $\}$

**definition**

$R23\text{-pres} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23\text{-pres} \equiv \{(s, t). \text{runs } s = \text{runs } t \wedge \text{leak } s = \text{leak } t \wedge \text{clk } s = \text{clk } t \wedge \text{cache } s = \text{cache } t \}$

**definition**

$R23 :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$   
 $R23 \equiv R23\text{-msgs} \cap R23\text{-keys} \cap R23\text{-non} \cap R23\text{-pres}$

**lemmas**  $R23\text{-defs} =$

$R23\text{-def } R23\text{-msgs-def } R23\text{-keys-def } R23\text{-non-def } R23\text{-pres-def}$

The mediator function is the identity here.

**definition**

$\text{med32} :: m3\text{-obs} \Rightarrow m2\text{-obs where}$   
 $\text{med32} \equiv \text{id}$

**lemmas**  $R23\text{-msgsI} = R23\text{-msgs-def [THEN rel-def-to-intro, simplified, rule-format]$

**lemmas**  $R23\text{-msgsE} [\text{elim}] = R23\text{-msgs-def [THEN rel-def-to-elim, simplified, rule-format]$

**lemmas**  $R23\text{-msgsE}' [\text{elim}] =$

$R23\text{-msgs-def [THEN rel-def-to-dest, simplified, rule-format, THEN subsetD]$

**lemmas**  $R23\text{-keysI} = R23\text{-keys-def [THEN rel-def-to-intro, simplified, rule-format]$

**lemmas**  $R23\text{-keysE} [\text{elim}] = R23\text{-keys-def [THEN rel-def-to-elim, simplified, rule-format]$

**lemmas**  $R23\text{-keysD} = R23\text{-keys-def [THEN rel-def-to-dest, simplified, rule-format, rotated 2]$

**lemmas**  $R23\text{-nonI} = R23\text{-non-def [THEN rel-def-to-intro, simplified, rule-format]$

**lemmas**  $R23\text{-nonE} [\text{elim}] = R23\text{-non-def [THEN rel-def-to-elim, simplified, rule-format]$

**lemmas**  $R23\text{-nonD} = R23\text{-non-def [THEN rel-def-to-dest, simplified, rule-format, rotated 2]$

**lemmas** *R23-presI* = *R23-pres-def* [*THEN rel-def-to-intro, simplified, rule-format*]  
**lemmas** *R23-presE* [*elim*] = *R23-pres-def* [*THEN rel-def-to-elim, simplified, rule-format*]

**lemmas** *R23-intros* = *R23-msgsI R23-keysI R23-nonI R23-presI*

Lemmas for various instantiations (keys and nonces).

**lemmas** *R23-keys-dests* =  
*R23-keysD*  
*R23-keysD* [**where**  $KK=\{\}$ , *simplified*]  
*R23-keysD* [**where**  $KK=\{K\}$  **for**  $K$ , *simplified*]  
*R23-keysD* [**where**  $KK=insert\ K\ KK$  **for**  $K\ KK$ , *simplified, OF - - conjI*]

**lemmas** *R23-non-dests* =  
*R23-nonD*  
*R23-nonD* [**where**  $KK=\{\}$ , *simplified*]  
*R23-nonD* [**where**  $KK=\{K\}$  **for**  $K$ , *simplified*]  
*R23-nonD* [**where**  $KK=insert\ K\ KK$  **for**  $K\ KK$ , *simplified, OF - - conjI*]

**lemmas** *R23-dests* = *R23-keys-dests R23-non-dests*

## General lemmas

General facts about *abs-msg*

**declare** *abs-msg.intros* [*intro!*]  
**declare** *abs-msg.cases* [*elim!*]

**lemma** *abs-msg-empty*:  $abs\ msg\ \{\} = \{\}$   
 $\langle proof \rangle$

**lemma** *abs-msg-Un* [*simp*]:  
 $abs\ msg\ (G \cup H) = abs\ msg\ G \cup abs\ msg\ H$   
 $\langle proof \rangle$

**lemma** *abs-msg-mono* [*elim*]:  
 $\llbracket m \in abs\ msg\ G; G \subseteq H \rrbracket \implies m \in abs\ msg\ H$   
 $\langle proof \rangle$

**lemma** *abs-msg-insert-mono* [*intro*]:  
 $\llbracket m \in abs\ msg\ H \rrbracket \implies m \in abs\ msg\ (insert\ m'\ H)$   
 $\langle proof \rangle$

Facts about *abs-msg* concerning abstraction of fakeable messages. This is crucial for proving the refinement of the intruder event.

**lemma** *abs-msg-DY-subset-fakeable*:  
 $\llbracket (s, t) \in R23\ msgs; (s, t) \in R23\ keys; (s, t) \in R23\ non; t \in m3\ inv4\ lkeysec \rrbracket$   
 $\implies abs\ msg\ (synth\ (analz\ (IK\ t))) \subseteq fake\ ik0\ (dom\ (runs\ s))\ (chan\ s)$   
 $\langle proof \rangle$

## Refinement proof

Pair decomposition. These were set to **elim!**, which is too aggressive here.

**declare** *MPair-analz* [rule del, elim]  
**declare** *MPair-parts* [rule del, elim]

Protocol events.

**lemma** *PO-m3-step1-refines-m2-step1*:  
 {*R23*}  
 (*m2-step1 Ra A B Na*), (*m3-step1 Ra A B Na*)  
 {> *R23*}  
 ⟨proof⟩

**lemma** *PO-m3-step2-refines-m2-step2*:  
 {*R23*}  
 (*m2-step2 Rb A B*), (*m3-step2 Rb A B*)  
 {> *R23*}  
 ⟨proof⟩

**lemma** *PO-m3-step3-refines-m2-step3*:  
 {*R23*  $\cap$  (*m2-inv3a-sesK-compr*)  $\times$  (*m3-inv7a-sesK-compr*  $\cap$  *m3-inv4-lkeysec*)}  
 (*m2-step3 Rs A B Kab Na Ts*), (*m3-step3 Rs A B Kab Na Ts*)  
 {> *R23*}  
 ⟨proof⟩

**lemma** *PO-m3-step4-refines-m2-step4*:  
 {*R23*  $\cap$  (*UNIV*)  
 $\times$  (*m3-inv7a-sesK-compr*  $\cap$  *m3-inv7b-sesK-compr-non*  $\cap$  *m3-inv6-ticket*  $\cap$  *m3-inv4-lkeysec*)}  
 (*m2-step4 Ra A B Na Kab Ts Ta*), (*m3-step4 Ra A B Na Kab Ts Ta X*)  
 {> *R23*}  
 ⟨proof⟩

**lemma** *PO-m3-step5-refines-m2-step5*:  
 {*R23*}  
 (*m2-step5 Rb A B Kab Ts Ta*), (*m3-step5 Rb A B Kab Ts Ta*)  
 {> *R23*}  
 ⟨proof⟩

**lemma** *PO-m3-step6-refines-m2-step6*:  
 {*R23*}  
 (*m2-step6 Ra A B Na Kab Ts Ta*), (*m3-step6 Ra A B Na Kab Ts Ta*)  
 {> *R23*}  
 ⟨proof⟩

**lemma** *PO-m3-tick-refines-m2-tick*:  
 {*R23*}  
 (*m2-tick T*), (*m3-tick T*)  
 {> *R23*}  
 ⟨proof⟩

**lemma** *PO-m3-purge-refines-m2-purge*:  
 {*R23*}  
 (*m2-purge A*), (*m3-purge A*)  
 {> *R23*}  
 ⟨proof⟩

Intruder events.

**lemma** *PO-m3-leak-refines-m2-leak*:

$\{R23\}$   
 $(m2\text{-leak } Rs \ A \ B \ Na \ Ts), (m3\text{-leak } Rs \ A \ B \ Na \ Ts)$   
 $\{>R23\}$   
 $\langle\text{proof}\rangle$

**lemma** *PO-m3-DY-fake-refines-m2-fake*:

$\{R23 \cap UNIV \times (m3\text{-inv4-lkeysec})\}$   
 $m2\text{-fake}, m3\text{-DY-fake}$   
 $\{> R23\}$   
 $\langle\text{proof}\rangle$

All together now...

**lemmas** *PO-m3-trans-refines-m2-trans* =

*PO-m3-step1-refines-m2-step1 PO-m3-step2-refines-m2-step2*  
*PO-m3-step3-refines-m2-step3 PO-m3-step4-refines-m2-step4*  
*PO-m3-step5-refines-m2-step5 PO-m3-step6-refines-m2-step6*  
*PO-m3-tick-refines-m2-tick PO-m3-purge-refines-m2-purge*  
*PO-m3-leak-refines-m2-leak PO-m3-DY-fake-refines-m2-fake*

**lemma** *PO-m3-refines-init-m2* [iff]:

$init \ m3 \subseteq R23 \text{“}(init \ m2)$   
 $\langle\text{proof}\rangle$

**lemma** *PO-m3-refines-trans-m2* [iff]:

$\{R23 \cap (m2\text{-inv3a-sesK-compr})$   
 $\times (m3\text{-inv7a-sesK-compr} \cap m3\text{-inv7b-sesK-compr-non} \cap m3\text{-inv6-ticket} \cap m3\text{-inv4-lkeysec})\}$   
 $(trans \ m2), (trans \ m3)$   
 $\{> R23\}$   
 $\langle\text{proof}\rangle$

**lemma** *PO-m3-observation-consistent* [iff]:

$obs\text{-consistent } R23 \ med32 \ m2 \ m3$   
 $\langle\text{proof}\rangle$

Refinement result.

**lemma** *m3-refines-m2* [iff]:

*refines*  
 $(R23 \cap$   
 $(m2\text{-inv3a-sesK-compr}) \times$   
 $(m3\text{-inv7a-sesK-compr} \cap m3\text{-inv7b-sesK-compr-non} \cap m3\text{-inv6-ticket} \cap m3\text{-inv4-lkeysec}))$   
 $med32 \ m2 \ m3$   
 $\langle\text{proof}\rangle$

**lemma** *m3-implements-m2* [iff]:

$implements \ med32 \ m2 \ m3$   
 $\langle\text{proof}\rangle$

### 3.8.7 Inherited invariants

#### inv3 (derived): Key secrecy for initiator

##### definition

$m3\text{-inv3}\text{-ikk}\text{-init} :: m3\text{-state set}$

##### where

$m3\text{-inv3}\text{-ikk}\text{-init} \equiv \{s. \forall A B Ra K Ts nl.$

$runs\ s\ Ra = Some\ (Init, [A, B], aKey\ K \# aNum\ Ts \# nl) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$

$Key\ K \in analz\ (IK\ s) \longrightarrow$

$(K, A, B, Ra\$na, Ts) \in leak\ s$

$\}$

**lemmas**  $m3\text{-inv3}\text{-ikk}\text{-init}I = m3\text{-inv3}\text{-ikk}\text{-init}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

**lemmas**  $m3\text{-inv3}\text{-ikk}\text{-init}E\ [elim] = m3\text{-inv3}\text{-ikk}\text{-init}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

**lemmas**  $m3\text{-inv3}\text{-ikk}\text{-init}D = m3\text{-inv3}\text{-ikk}\text{-init}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

**lemma**  $PO\text{-}m3\text{-inv3}\text{-ikk}\text{-init}: reach\ m3 \subseteq m3\text{-inv3}\text{-ikk}\text{-init}$

$\langle proof \rangle$

#### inv4 (derived): Key secrecy for responder

##### definition

$m3\text{-inv4}\text{-ikk}\text{-resp} :: m3\text{-state set}$

##### where

$m3\text{-inv4}\text{-ikk}\text{-resp} \equiv \{s. \forall A B Rb K Ts nl.$

$runs\ s\ Rb = Some\ (Resp, [A, B], aKey\ K \# aNum\ Ts \# nl) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$

$Key\ K \in analz\ (IK\ s) \longrightarrow$

$(\exists Na. (K, A, B, Na, Ts) \in leak\ s)$

$\}$

**lemmas**  $m3\text{-inv4}\text{-ikk}\text{-resp}I = m3\text{-inv4}\text{-ikk}\text{-resp}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

**lemmas**  $m3\text{-inv4}\text{-ikk}\text{-resp}E\ [elim] = m3\text{-inv4}\text{-ikk}\text{-resp}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

**lemmas**  $m3\text{-inv4}\text{-ikk}\text{-resp}D = m3\text{-inv4}\text{-ikk}\text{-resp}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

**lemma**  $PO\text{-}m3\text{-inv4}\text{-ikk}\text{-resp}: reach\ m3 \subseteq m3\text{-inv4}\text{-ikk}\text{-resp}$

$\langle proof \rangle$

end

## 3.9 Abstract Needham-Schroeder Shared Key (L1)

**theory**  $m1\text{-nssk imports } m1\text{-keydist}\text{-iirn}$

**begin**

We add augment the basic abstract key distribution model such that the server reads and stores the initiator's nonce. We show three refinements, namely that this model refines

1. the basic key distribution model  $m1a$ , and
2. the injective agreement model  $a0i$ , instantiated such that the initiator agrees with the server on the session key and its nonce.

3. the non-injective agreement model  $a0n$ , instantiated such that the responder agrees with the server on the session key.

**consts**

$nb :: nat$  — responder nonce constant  
 $END :: atom$  — run end marker for responder

### 3.9.1 State

We extend the basic key distribution by adding nonces. The frames, the state, and the observations remain the same as in the previous model, but we will use the *nat list*'s to store nonces.

**record**  $m1\text{-state} = m1r\text{-state} +$   
 $leak :: (key \times fresh\text{-}t \times fresh\text{-}t) \text{ set}$  — keys leaked plus session context

**type-synonym**  $m1\text{-obs} = m1\text{-state}$

**type-synonym**  $'x \text{ m1-pred} = 'x \text{ m1-state-scheme set}$

**type-synonym**  $'x \text{ m1-trans} = ('x \text{ m1-state-scheme} \times 'x \text{ m1-state-scheme}) \text{ set}$

### 3.9.2 Events

**definition** — by  $A$ , refines  $m1a\text{-step1}$

$m1\text{-step1} :: [rid\text{-}t, agent, agent, nonce] \Rightarrow 'x \text{ m1r-trans}$

**where**

$m1\text{-step1 } Ra \ A \ B \ Na \equiv m1a\text{-step1 } Ra \ A \ B \ Na$

**definition** — by  $B$ , refines  $m1a\text{-step2}$

$m1\text{-step2} :: [rid\text{-}t, agent, agent] \Rightarrow 'x \text{ m1r-trans}$

**where**

$m1\text{-step2 } Rb \ A \ B \equiv m1a\text{-step2 } Rb \ A \ B$

**definition** — by  $Sv$ , refines  $m1a\text{-step3}$

$m1\text{-step3} :: [rid\text{-}t, agent, agent, nonce, key] \Rightarrow 'x \text{ m1r-trans}$

**where**

$m1\text{-step3 } Rs \ A \ B \ Na \ Kab \equiv m1a\text{-step3 } Rs \ A \ B \ Kab \ Na \ []$

**definition** — by  $A$ , refines  $m1a\text{-step4}$

$m1\text{-step4} :: [rid\text{-}t, agent, agent, nonce, key] \Rightarrow 'x \text{ m1-trans}$

**where**

$m1\text{-step4 } Ra \ A \ B \ Na \ Kab \equiv \{(s, s^\wedge).$

— guards:

$runs \ s \ Ra = Some \ (Init, [A, B], []) \wedge$

$Na = Ra\$na \wedge$

— fix parameter

$(Kab \notin Domain \ (leak \ s) \longrightarrow (Kab, A) \in azC \ (runs \ s)) \wedge$  — authorization guard

— new guard for agreement with server on  $(Kab, B, Na)$ ,

— injectiveness by including  $Na$

$(A \notin bad \longrightarrow (\exists Rs. Kab = sesK \ (Rs\$sk) \wedge$

$runs \ s \ Rs = Some \ (Serv, [A, B], [aNon \ Na]))) \wedge$

— actions:

$$s' = s \{ \text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [aKey Kab])) \}$$

**definition** — by  $B$ , refines  $m1a\text{-step}5$

$$m1\text{-step}5 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow 'x m1\text{-trans}$$

**where**

$$m1\text{-step}5 Rb A B Nb Kab \equiv \{(s, s')\}.$$

— new guards:

$$Nb = Rb\$nb \wedge$$

— generate  $Nb$

— prev guards:

$$\text{runs } s Rb = \text{Some } (\text{Resp}, [A, B], []) \wedge$$

$$(Kab \notin \text{Domain } (\text{leak } s) \longrightarrow (Kab, B) \in \text{azC } (\text{runs } s)) \wedge \quad \text{— authorization guard}$$

— guard for showing agreement with server on  $(Kab, A)$ ,

— this agreement is non-injective

$$(B \notin \text{bad} \longrightarrow (\exists Rs Na. Kab = \text{sesK } (Rs\$sk) \wedge$$

$$\text{runs } s Rs = \text{Some } (\text{Serv}, [A, B], [aNon Na]))) \wedge$$

— actions:

$$s' = s \{ \text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [aKey Kab])) \}$$

**definition** — by  $A$ , refines  $\text{skip}$

$$m1\text{-step}6 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{nonce}, \text{key}] \Rightarrow 'x m1\text{-trans}$$

**where**

$$m1\text{-step}6 Ra A B Na Nb Kab \equiv \{(s, s')\}.$$

$$\text{runs } s Ra = \text{Some } (\text{Init}, [A, B], [aKey Kab]) \wedge \quad \text{— key rcv'd before}$$

$$Na = Ra\$na \wedge$$

— guard for showing agreement with  $B$  on  $Kab$  and  $Nb$

$$(A \notin \text{bad} \longrightarrow B \notin \text{bad} \longrightarrow$$

$$(\forall Nb'. (Kab, Na, Nb') \notin \text{leak } s) \longrightarrow \quad \text{— NEW: weaker condition}$$

$$(\exists Rb nl. Nb = Rb\$nb \wedge \text{runs } s Rb = \text{Some } (\text{Resp}, [A, B], aKey Kab \# nl))) \wedge$$

— actions:

$$s' = s \{$$

$$\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [aKey Kab, aNon Nb]))$$

$$\}$$

**definition** — by  $B$ , refines  $\text{skip}$

$$m1\text{-step}7 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow 'x m1\text{-trans}$$

**where**

$$m1\text{-step}7 Rb A B Nb Kab \equiv \{(s, s')\}.$$

$$\text{runs } s Rb = \text{Some } (\text{Resp}, [A, B], [aKey Kab]) \wedge \quad \text{— key rcv'd before}$$

$$Nb = Rb\$nb \wedge$$

— guard for showing agreement with  $A$  on  $Kab$  and  $Nb$

$$(A \notin \text{bad} \longrightarrow B \notin \text{bad} \longrightarrow Kab \notin \text{Domain } (\text{leak } s) \longrightarrow$$

$$\text{— } (\forall Na'. (Kab, Na', Nb) \notin \text{leak } s) \longrightarrow \text{too strong, does not work}$$

$$(\exists Ra. \text{runs } s Ra = \text{Some } (\text{Init}, [A, B], [aKey Kab, aNon Nb]))) \wedge$$

— actions: (redundant) update local state marks successful termination

$$s' = s \langle \begin{array}{l} \text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [\text{aKey } Kab, \text{END}])) \\ \rangle \\ \} \end{array}$$

**definition** — by attacker, refines *s0g-leak*  
 $m1\text{-leak} :: [\text{rid-}t, \text{rid-}t, \text{rid-}t, \text{agent}, \text{agent}] \Rightarrow 'x \text{ m1-trans}$

**where**

$m1\text{-leak } Rs \ Ra \ Rb \ A \ B \equiv \{(s, s1)\}.$

— guards:

$\text{runs } s \ Rs = \text{Some } (\text{Serv}, [A, B], [\text{aNon } (Ra\$na)]) \wedge$

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], [\text{aKey } (\text{sesK } (Rs\$sk)), \text{aNon } (Rb\$nb)]) \wedge$

$\text{runs } s \ Rb = \text{Some } (\text{Resp}, [A, B], [\text{aKey } (\text{sesK } (Rs\$sk)), \text{END}]) \wedge$

— actions:

$s1 = s \langle \text{leak} := \text{insert } (\text{sesK } (Rs\$sk), Ra\$na, Rb\$nb) (\text{leak } s) \rangle$

$\}$

### 3.9.3 Specification

**abbreviation**

$m1\text{-init} :: m1\text{-state set}$

**where**

$m1\text{-init} \equiv \{ \langle$

$\text{runs} = \text{Map.empty},$

$\text{leak} = \text{corrKey} \times \{\text{undefined}\} \times \{\text{undefined}\} \quad \text{— initial leakage}$

$\rangle \}$

**definition**

$m1\text{-trans} :: 'x \text{ m1-trans}$  **where**

$m1\text{-trans} \equiv (\bigcup A \ B \ Ra \ Rb \ Rs \ Na \ Nb \ Kab.$

$m1\text{-step1 } Ra \ A \ B \ Na \cup$

$m1\text{-step2 } Rb \ A \ B \cup$

$m1\text{-step3 } Rs \ A \ B \ Na \ Kab \cup$

$m1\text{-step4 } Ra \ A \ B \ Na \ Kab \cup$

$m1\text{-step5 } Rb \ A \ B \ Nb \ Kab \cup$

$m1\text{-step6 } Ra \ A \ B \ Na \ Nb \ Kab \cup$

$m1\text{-step7 } Rb \ A \ B \ Nb \ Kab \cup$

$m1\text{-leak } Rs \ Ra \ Rb \ A \ B \cup$

$\text{Id}$

$\rangle$

**definition**

$m1 :: (m1\text{-state}, m1\text{-obs}) \text{ spec}$  **where**

$m1 \equiv \langle$

$\text{init} = m1\text{-init},$

$\text{trans} = m1\text{-trans},$

$\text{obs} = \text{id}$

$\rangle$

**lemmas**  $m1\text{-loc-defs} =$

$m1\text{-def } m1\text{-trans-def}$

*m1-step1-def m1-step2-def m1-step3-def m1-step4-def m1-step5-def  
m1-step6-def m1-step7-def m1-leak-def*

**lemmas** *m1-defs = m1-loc-defs m1a-defs*

**lemma** *m1-obs-id [simp]: obs m1 = id*  
 $\langle proof \rangle$

### 3.9.4 Invariants

#### inv0: Finite domain

There are only finitely many runs. This is needed to establish the responder/initiator agreements. This is already defined in the previous model, we just need to show that it still holds in the current model.

#### abbreviation

*m1-inv0-fin* :: 'x *m1-pred* **where**  
*m1-inv0-fin*  $\equiv$  *m1a-inv0-fin*

**lemmas** *m1-inv0-finI = m1a-inv0-finI*

**lemmas** *m1-inv0-finE = m1a-inv0-finE*

**lemmas** *m1-inv0-finD = m1a-inv0-finD*

Invariance proofs.

**lemma** *PO-m1-inv0-fin-init [iff]:*  
*init m1*  $\subseteq$  *m1-inv0-fin*  
 $\langle proof \rangle$

**lemma** *PO-m1-inv0-fin-trans [iff]:*  
 $\{m1-inv0-fin\}$  *trans m1*  $\{> m1-inv0-fin\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-inv0-fin [iff]: reach m1*  $\subseteq$  *m1-inv0-fin*  
 $\langle proof \rangle$

**declare** *PO-m1-inv0-fin [THEN subsetD, intro]*

### 3.9.5 Refinement of *m1a*

#### Simulation relation

med1a1: The mediator function maps a concrete observation (i.e., run) to an abstract one.

Instantiate parameters regarding list of freshness identifiers stored at server.

**overloading** *is-len'*  $\equiv$  *is-len* *rs-len'*  $\equiv$  *rs-len* **begin**

**definition** *is-len-def [simp]: is-len'*  $\equiv$  *0::nat*

**definition** *rs-len-def [simp]: rs-len'*  $\equiv$  *0::nat*

**end**

**fun**

*rm1a1* :: *role-t*  $\Rightarrow$  *atom list*  $\Rightarrow$  *atom list*

**where**

$rm1a1\ Init = take\ (Suc\ is-len) \quad \text{--- take } Kab$   
 $| rm1a1\ Resp = take\ (Suc\ rs-len) \quad \text{--- take } Kab$   
 $| rm1a1\ Serv = id \quad \text{--- take all}$

**abbreviation**

$runs1a1 :: runs-t \Rightarrow runs-t$  **where**  
 $runs1a1 \equiv map-runs\ rm1a1$

**lemmas**  $runs1a1-def = map-runs-def$

**lemma**  $knC-runs1a1$  [simp]:

$knC\ (runs1a1\ runz) = knC\ runz$

$\langle proof \rangle$

R1a1: The simulation relation is defined in terms of the mediator function.

**definition**

$med1a1 :: m1-obs \Rightarrow m1a-obs$  **where**

$med1a1\ s \equiv \langle runs = runs1a1\ (runs\ s), m1x-state.leak = Domain\ (leak\ s) \rangle$

**definition**

$R1a1 :: (m1a-state \times m1-state)$  *set* **where**

$R1a1 \equiv \{(s, t). s = med1a1\ t\}$

**lemmas**  $R1a1-defs = R1a1-def\ med1a1-def$

**Refinement proof**

**lemma**  $PO-m1-step1-refines-m1a-step1$ :

$\{R1a1\}$

$(m1a-step1\ Ra\ A\ B\ Na), (m1-step1\ Ra\ A\ B\ Na)$

$\{>\ R1a1\}$

$\langle proof \rangle$

**lemma**  $PO-m1-step2-refines-m1a-step2$ :

$\{R1a1\}$

$(m1a-step2\ Rb\ A\ B), (m1-step2\ Rb\ A\ B)$

$\{>\ R1a1\}$

$\langle proof \rangle$

**lemma**  $PO-m1-step3-refines-m1a-step3$ :

$\{R1a1\}$

$(m1a-step3\ Rs\ A\ B\ Kab\ Na\ []), (m1-step3\ Rs\ A\ B\ Na\ Kab)$

$\{>\ R1a1\}$

$\langle proof \rangle$

**lemma**  $PO-m1-step4-refines-m1a-step4$ :

$\{R1a1\}$

$(m1a-step4\ Ra\ A\ B\ Na\ Kab\ []), (m1-step4\ Ra\ A\ B\ Na\ Kab)$

$\{>\ R1a1\}$

$\langle proof \rangle$

**lemma**  $PO-m1-step5-refines-m1a-step5$ :

$\{R1a1\}$

$(m1a\text{-step5 } Rb \ A \ B \ Kab \ []), (m1\text{-step5 } Rb \ A \ B \ Nb \ Kab)$   
 $\{> R1a1\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step6-refines-m1a-skip*:  
 $\{R1a1\}$   
 $Id, (m1\text{-step6 } Ra \ A \ B \ Na \ Nb \ Kab)$   
 $\{> R1a1\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step7-refines-m1a-skip*:  
 $\{R1a1\}$   
 $Id, (m1\text{-step7 } Rb \ A \ B \ Nb \ Kab)$   
 $\{> R1a1\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-leak-refines-m1a-leak*:  
 $\{R1a1\}$   
 $(m1a\text{-leak } Rs), (m1\text{-leak } Rs \ Ra \ Rb \ A \ B)$   
 $\{> R1a1\}$   
 $\langle proof \rangle$

All together now...

**lemmas** *PO-m1-trans-refines-m1a-trans* =  
 $PO\text{-m1-step1-refines-m1a-step1 } PO\text{-m1-step2-refines-m1a-step2}$   
 $PO\text{-m1-step3-refines-m1a-step3 } PO\text{-m1-step4-refines-m1a-step4}$   
 $PO\text{-m1-step5-refines-m1a-step5 } PO\text{-m1-step6-refines-m1a-skip}$   
 $PO\text{-m1-step7-refines-m1a-skip } PO\text{-m1-leak-refines-m1a-leak}$

**lemma** *PO-m1-refines-init-m1a* [iff]:  
 $init \ m1 \subseteq R1a1 \text{“}(init \ m1a)$   
 $\langle proof \rangle$

**lemma** *PO-m1-refines-trans-m1a* [iff]:  
 $\{R1a1\}$   
 $(trans \ m1a), (trans \ m1)$   
 $\{> R1a1\}$   
 $\langle proof \rangle$

Observation consistency.

**lemma** *obs-consistent-med1a1* [iff]:  
 $obs\text{-consistent } R1a1 \ med1a1 \ m1a \ m1$   
 $\langle proof \rangle$

Refinement result.

**lemma** *PO-m1-refines-m1a* [iff]:  
 $refines \ R1a1 \ med1a1 \ m1a \ m1$   
 $\langle proof \rangle$

**lemma** *m1-implements-m1a* [iff]:  $implements \ med1a1 \ m1a \ m1$   
 $\langle proof \rangle$

### inv (inherited): Key secrecy

Secrecy, as external and internal invariant

#### definition

$m1\text{-secrecy} :: 'x\ m1\text{-pred\ where}$   
 $m1\text{-secrecy} \equiv \{s. knC\ (runs\ s) \subseteq azC\ (runs\ s) \cup Domain\ (leak\ s) \times UNIV\}$

**lemmas**  $m1\text{-secrecy}I = m1\text{-secrecy}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

**lemmas**  $m1\text{-secrecy}E\ [elim] = m1\text{-secrecy}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

**lemma**  $PO\text{-}m1\text{-obs}\text{-secrecy}\ [iff]:\ oreach\ m1 \subseteq m1\text{-secrecy}$   
 $\langle proof \rangle$

**lemma**  $PO\text{-}m1\text{-secrecy}\ [iff]:\ reach\ m1 \subseteq m1\text{-secrecy}$   
 $\langle proof \rangle$

### inv (inherited): Initiator auth server.

Simplified version of invariant  $m1a\text{-inv}2i\text{-serv}$ .

#### definition

$m1\text{-inv}2i\text{-serv} :: 'x\ m1r\text{-pred}$

#### where

$m1\text{-inv}2i\text{-serv} \equiv \{s. \forall A\ B\ Ra\ Na\ Kab\ nla.$   
 $A \notin bad \longrightarrow$   
 $runs\ s\ Ra = Some\ (Init,\ [A,\ B],\ aKey\ Kab\ \# \ nla) \longrightarrow$   
 $Na = Ra\$na \longrightarrow$   
 $(\exists Rs. Kab = sesK\ (Rs\$sk) \wedge runs\ s\ Rs = Some\ (Serv,\ [A,\ B],\ [aNon\ Na]))$   
 $\}$

**lemmas**  $m1\text{-inv}2i\text{-serv}I = m1\text{-inv}2i\text{-serv}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

**lemmas**  $m1\text{-inv}2i\text{-serv}E\ [elim] = m1\text{-inv}2i\text{-serv}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

**lemmas**  $m1\text{-inv}2i\text{-serv}D = m1\text{-inv}2i\text{-serv}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 2]$

Proof of invariance.

**lemma**  $PO\text{-}m1\text{-inv}2i\text{-serv}\ [iff]:\ reach\ m1 \subseteq m1\text{-inv}2i\text{-serv}$   
 $\langle proof \rangle$

**declare**  $PO\text{-}m1\text{-inv}2i\text{-serv}\ [THEN\ subsetD,\ intro]$

### inv (inherited): Responder auth server.

Simplified version of invariant  $m1a\text{-inv}2r\text{-serv}$ .

#### definition

$m1\text{-inv}2r\text{-serv} :: 'x\ m1r\text{-pred}$

#### where

$m1\text{-inv}2r\text{-serv} \equiv \{s. \forall A\ B\ Rb\ Kab\ nlb.$   
 $B \notin bad \longrightarrow$   
 $runs\ s\ Rb = Some\ (Resp,\ [A,\ B],\ aKey\ Kab\ \# \ nlb) \longrightarrow$   
 $(\exists Rs\ Na. Kab = sesK\ (Rs\$sk) \wedge runs\ s\ Rs = Some\ (Serv,\ [A,\ B],\ [aNon\ Na]))$   
 $\}$

**lemmas**  $m1\text{-inv}2r\text{-serv}I = m1\text{-inv}2r\text{-serv}\text{-def}$  [THEN setc-def-to-intro, rule-format]  
**lemmas**  $m1\text{-inv}2r\text{-serv}E$  [elim] =  $m1\text{-inv}2r\text{-serv}\text{-def}$  [THEN setc-def-to-elim, rule-format]  
**lemmas**  $m1\text{-inv}2r\text{-serv}D = m1\text{-inv}2r\text{-serv}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated -1]

Proof of invariance.

**lemma**  $PO\text{-}m1\text{-inv}2r\text{-serv}$  [iff]:  $reach\ m1 \subseteq m1\text{-inv}2r\text{-serv}$   
 <proof>

**declare**  $PO\text{-}m1\text{-inv}2r\text{-serv}$  [THEN subsetD, intro]

### inv (inherited): Initiator key freshness

#### definition

$m1\text{-inv}3\text{-ifresh} :: 'x\ m1\text{-pred}$

#### where

$m1\text{-inv}3\text{-ifresh} \equiv \{s. \forall A\ A'\ B\ B'\ Ra\ Ra'\ Kab\ nl\ nl'. \\
 runs\ s\ Ra = Some\ (Init,\ [A,\ B],\ aKey\ Kab\ \#\ nl) \longrightarrow \\
 runs\ s\ Ra' = Some\ (Init,\ [A',\ B'],\ aKey\ Kab\ \#\ nl') \longrightarrow \\
 A \notin bad \longrightarrow B \notin bad \longrightarrow Kab \notin Domain\ (leak\ s) \longrightarrow \\
 Ra = Ra' \\
 \}$

**lemmas**  $m1\text{-inv}3\text{-ifresh}I = m1\text{-inv}3\text{-ifresh}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m1\text{-inv}3\text{-ifresh}E$  [elim] =  $m1\text{-inv}3\text{-ifresh}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m1\text{-inv}3\text{-ifresh}D = m1\text{-inv}3\text{-ifresh}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

**lemma**  $PO\text{-}m1\text{-inv}3\text{-ifresh}$  [iff]:  $reach\ m1 \subseteq m1\text{-inv}3\text{-ifresh}$   
 <proof>

## 3.9.6 Refinement of $a0i$ for initiator/responder

### Simulation relation

We define two auxiliary functions to reconstruct the signals of the initial model from completed initiator and responder runs. For the initiator, we get an injective agreement with the responder on  $Kab$  and  $Nb$ .

#### type-synonym

$irsig = key \times nonce$

#### abbreviation

$ir\text{-commit} :: [runs\text{-}t,\ agent,\ agent,\ key,\ nonce] \Rightarrow rid\text{-}t\ set$

#### where

$ir\text{-commit}\ runz\ A\ B\ Kab\ Nb \equiv \{Ra. \\
 runz\ Ra = Some\ (Init,\ [A,\ B],\ [aKey\ Kab,\ aNon\ Nb]) \\
 \}$

#### fun

$ir\text{-runs}2sigs :: runs\text{-}t \Rightarrow irsig\ signal \Rightarrow nat$

#### where

$ir\text{-runs}2sigs\ runz\ (Commit\ [A,\ B]\ (Kab,\ Nb)) = \\
 card\ (ir\text{-commit}\ runz\ A\ B\ Kab\ Nb)$

|  $ir\text{-runs2sigs}\ runz\ (Running\ [A,\ B]\ (Kab,\ Nb)) =$   
    $(if\ \exists\ Rb\ nl.\ Nb = Rb\$nb \wedge runz\ Rb = Some\ (Resp,\ [A,\ B],\ aKey\ Kab\ \# \ nl)$   
    $then\ 1\ else\ 0)$

|  $ir\text{-runs2sigs}\ runz\ - = 0$

Simulation relation and mediator function. We map completed initiator and responder runs to commit and running signals, respectively.

**definition**

$med\text{-}a0im1\text{-}ir :: m1\text{-}obs \Rightarrow irsig\ a0i\text{-}obs$  **where**  
 $med\text{-}a0im1\text{-}ir\ o1 \equiv \llbracket signals = ir\text{-runs2sigs}\ (runs\ o1),\ corrupted = Domain\ (leak\ o1) \times UNIV \rrbracket$

**definition**

$R\text{-}a0im1\text{-}ir :: (irsig\ a0i\text{-}state \times m1\text{-}state)$  **set where**  
 $R\text{-}a0im1\text{-}ir \equiv \{(s,\ t).\ signals\ s = ir\text{-runs2sigs}\ (runs\ t) \wedge corrupted\ s = Domain\ (leak\ t) \times UNIV\}$

**lemmas**  $R\text{-}a0im1\text{-}ir\text{-}defs = R\text{-}a0im1\text{-}ir\text{-}def\ med\text{-}a0im1\text{-}ir\text{-}def$

**Lemmas about the abstraction function**

**lemma**  $ir\text{-runs2sigs}\text{-}empty$  [simp]:

$runz = Map.empty \implies ir\text{-runs2sigs}\ runz = (\lambda s.\ 0)$   
 $\langle proof \rangle$

**lemma**  $finite\text{-}ir\text{-}commit$  [simp, intro!]:

$finite\ (dom\ runz) \implies finite\ (ir\text{-}commit\ runz\ A\ B\ Kab\ Nb)$   
 $\langle proof \rangle$

Update lemmas

**lemma**  $ir\text{-runs2sigs}\text{-}upd\text{-}init\text{-}none$  [simp]:

$\llbracket Ra \notin dom\ runz \rrbracket$   
 $\implies ir\text{-runs2sigs}\ (runz(Ra \mapsto (Init,\ [A,\ B],\ []))) = ir\text{-runs2sigs}\ runz$   
 $\langle proof \rangle$

**lemma**  $ir\text{-runs2sigs}\text{-}upd\text{-}resp\text{-}none$  [simp]:

$\llbracket Rb \notin dom\ runz \rrbracket$   
 $\implies ir\text{-runs2sigs}\ (runz(Rb \mapsto (Resp,\ [A,\ B],\ []))) = ir\text{-runs2sigs}\ runz$   
 $\langle proof \rangle$

**lemma**  $ir\text{-runs2sigs}\text{-}upd\text{-}serv\text{-}none$  [simp]:

$\llbracket Rs \notin dom\ runz \rrbracket$   
 $\implies ir\text{-runs2sigs}\ (runz(Rs \mapsto (Serv,\ [A,\ B],\ nl))) = ir\text{-runs2sigs}\ runz$   
 $\langle proof \rangle$

**lemma**  $ir\text{-runs2sigs}\text{-}upd\text{-}init\text{-}some$  [simp]:

$\llbracket runz\ Ra = Some\ (Init,\ [A,\ B],\ []) \rrbracket$   
 $\implies ir\text{-runs2sigs}\ (runz(Ra \mapsto (Init,\ [A,\ B],\ [aKey\ Kab]))) = ir\text{-runs2sigs}\ runz$   
 $\langle proof \rangle$

**lemma**  $ir\text{-runs2sigs}\text{-}upd\text{-}resp$  [simp]:

$\llbracket runz\ Rb = Some\ (Resp,\ [A,\ B],\ []) \rrbracket$

$\implies ir\text{-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], [aKey Kab]))) =$   
 $(ir\text{-runs2sigs} \text{runz})(\text{Running } [A, B] (Kab, Rb\$nb) := 1)$   
 $\langle \text{proof} \rangle$

**lemma** *ir-runs2sigs-upd-init* [simp]:  
 $\llbracket \text{runz } Ra = \text{Some } (\text{Init}, [A, B], [aKey Kab]); \text{finite } (\text{dom } \text{runz}) \rrbracket$   
 $\implies ir\text{-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [aKey Kab, aNon Nb]))) =$   
 $(ir\text{-runs2sigs} \text{runz})$   
 $(\text{Commit } [A, B] (Kab, Nb) := \text{Suc } (\text{card } (ir\text{-commit } \text{runz } A B Kab Nb)))$   
 $\langle \text{proof} \rangle$

**lemma** *ir-runs2sigs-upd-resp-some* [simp]:  
 $\llbracket \text{runz } Rb = \text{Some } (\text{Resp}, [A, B], [aKey K]) \rrbracket$   
 $\implies ir\text{-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], [aKey K, END]))) = ir\text{-runs2sigs} \text{runz}$   
 $\langle \text{proof} \rangle$

Needed for injectiveness of agreement.

**lemma** *m1-inv2i-serv-lemma*:  
 $\llbracket \text{runs } t Ra = \text{Some } (\text{Init}, [A, B], [aKey Kab, aNon Nb]);$   
 $\text{runs } t Ra' = \text{Some } (\text{Init}, [A, B], [aKey Kab]);$   
 $A \notin \text{bad}; t \in m1\text{-inv2i-serv} \rrbracket$   
 $\implies P$   
 $\langle \text{proof} \rangle$

## Refinement proof

**lemma** *PO-m1-step1-refines-ir-a0i-skip*:  
 $\{R\text{-a0im1-ir}\}$   
 $\text{Id}, (m1\text{-step1 } Ra A B Na)$   
 $\{> R\text{-a0im1-ir}\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m1-step2-refines-ir-a0i-skip*:  
 $\{R\text{-a0im1-ir}\}$   
 $\text{Id}, (m1\text{-step2 } Rb A B)$   
 $\{> R\text{-a0im1-ir}\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m1-step3-refines-ir-a0i-skip*:  
 $\{R\text{-a0im1-ir}\}$   
 $\text{Id}, (m1\text{-step3 } Rs A B Na Kab)$   
 $\{> R\text{-a0im1-ir}\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m1-step4-refines-ir-a0i-skip*:  
 $\{R\text{-a0im1-ir}\}$   
 $\text{Id}, (m1\text{-step4 } Ra A B Na Kab)$   
 $\{> R\text{-a0im1-ir}\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m1-step5-refines-ir-a0i-running*:  
 $\{R\text{-a0im1-ir}\}$   
 $(a0i\text{-running } [A, B] (Kab, Nb)), (m1\text{-step5 } Rb A B Nb Kab)$

$\{> R\text{-}a0im1\text{-}ir\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step6-refines-ir-a0i-commit:*

$\{R\text{-}a0im1\text{-}ir \cap UNIV \times (m1\text{-}inv2i\text{-}serv \cap m1\text{-}inv0\text{-}fin)\}$   
 $(a0i\text{-}commit [A, B] (Kab, Nb)), (m1\text{-}step6 Ra A B Na Nb Kab)$   
 $\{> R\text{-}a0im1\text{-}ir\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step7-refines-ir-a0i-skip:*

$\{R\text{-}a0im1\text{-}ir\}$   
 $Id, (m1\text{-}step7 Rb A B Nb Kab)$   
 $\{> R\text{-}a0im1\text{-}ir\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-leak-refines-ir-a0i-corrupt:*

$\{R\text{-}a0im1\text{-}ir\}$   
 $(a0i\text{-}corrupt (\{sesK (Rs\$sk)\} \times UNIV)), (m1\text{-}leak Rs Ra Rb A B)$   
 $\{> R\text{-}a0im1\text{-}ir\}$   
 $\langle proof \rangle$

All together now...

**lemmas** *PO-m1-trans-refines-ir-a0i-trans =*

*PO-m1-step1-refines-ir-a0i-skip PO-m1-step2-refines-ir-a0i-skip*  
*PO-m1-step3-refines-ir-a0i-skip PO-m1-step4-refines-ir-a0i-skip*  
*PO-m1-step5-refines-ir-a0i-running PO-m1-step6-refines-ir-a0i-commit*  
*PO-m1-step7-refines-ir-a0i-skip PO-m1-leak-refines-ir-a0i-corrupt*

**lemma** *PO-m1-refines-ir-init-a0i [iff]:*

$init\ m1 \subseteq R\text{-}a0im1\text{-}ir''(init\ a0i)$   
 $\langle proof \rangle$

**lemma** *PO-m1-refines-ir-trans-a0i [iff]:*

$\{R\text{-}a0im1\text{-}ir \cap reach\ a0i \times reach\ m1\}$   
 $(trans\ a0i), (trans\ m1)$   
 $\{> R\text{-}a0im1\text{-}ir\}$   
 $\langle proof \rangle$

Observation consistency.

**lemma** *obs-consistent-med-a0im1-ir [iff]:*

$obs\text{-}consistent\ R\text{-}a0im1\text{-}ir\ med\text{-}a0im1\text{-}ir\ a0i\ m1$   
 $\langle proof \rangle$

Refinement result.

**lemma** *PO-m1-refines-ir-a0i [iff]:*

*refines*  
 $(R\text{-}a0im1\text{-}ir \cap reach\ a0i \times reach\ m1)$   
 $med\text{-}a0im1\text{-}ir\ a0i\ m1$   
 $\langle proof \rangle$

**lemma** *m1-implements-ir-a0i: implements med-a0im1-ir a0i m1*

$\langle proof \rangle$

### 3.9.7 Refinement of $a0i$ for responder/initiator

#### Simulation relation

We define two auxiliary functions to reconstruct the signals of the initial model from initiator and responder runs. For the responder, we get an injective agreement with the initiator on  $Kab$  and  $Nb$ .

#### type-synonym

$risig = key \times nonce$

#### abbreviation

$ri\text{-}running :: [runs\text{-}t, agent, agent, key, nonce] \Rightarrow rid\text{-}t\ set$

#### where

$ri\text{-}running\ runz\ A\ B\ Kab\ Nb \equiv \{Ra.$   
 $\quad runz\ Ra = Some\ (Init,\ [A,\ B],\ [aKey\ Kab,\ aNon\ Nb])$   
 $\}$

#### fun

$ri\text{-}runs2sigs :: runs\text{-}t \Rightarrow risig\ signal \Rightarrow nat$

#### where

$ri\text{-}runs2sigs\ runz\ (Commit\ [B,\ A]\ (Kab,\ Nb)) =$   
 $(if\ \exists\ Rb.\ Nb = Rb\$nb \wedge runz\ Rb = Some\ (Resp,\ [A,\ B],\ [aKey\ Kab,\ END])$   
 $\quad then\ 1\ else\ 0)$

$| ri\text{-}runs2sigs\ runz\ (Running\ [B,\ A]\ (Kab,\ Nb)) =$   
 $\quad card\ (ri\text{-}running\ runz\ A\ B\ Kab\ Nb)$

$| ri\text{-}runs2sigs\ runz\ - = 0$

Simulation relation and mediator function. We map completed initiator and responder runs to commit and running signals, respectively.

#### definition

$med\text{-}a0im1\text{-}ri :: m1\text{-}obs \Rightarrow risig\ a0i\text{-}obs$  **where**  
 $med\text{-}a0im1\text{-}ri\ o1 \equiv (\mid signals = ri\text{-}runs2sigs\ (runs\ o1),\ corrupted = Domain\ (leak\ o1) \times UNIV \mid)$

#### definition

$R\text{-}a0im1\text{-}ri :: (risig\ a0i\text{-}state \times m1\text{-}state)\ set$  **where**  
 $R\text{-}a0im1\text{-}ri \equiv \{(s,\ t).\ signals\ s = ri\text{-}runs2sigs\ (runs\ t) \wedge corrupted\ s = Domain\ (leak\ t) \times UNIV\}$

**lemmas**  $R\text{-}a0im1\text{-}ri\text{-}defs = R\text{-}a0im1\text{-}ri\text{-}def\ med\text{-}a0im1\text{-}ri\text{-}def$

#### Lemmas about the auxiliary functions

**lemma**  $ri\text{-}runs2sigs\text{-}empty$  [simp]:

$runz = Map.empty \Longrightarrow ri\text{-}runs2sigs\ runz = (\lambda s.\ 0)$   
 $\langle proof \rangle$

**lemma**  $finite\text{-}inv\text{-}ri\text{-}running$  [simp, intro!]:

$finite\ (dom\ runz) \Longrightarrow finite\ (ri\text{-}running\ runz\ A\ B\ Kab\ Nb)$   
 $\langle proof \rangle$

Update lemmas

**lemma** *ri-runs2sigs-upd-init-none* [*simp*]:

[[  $Na \notin \text{dom runz}$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Na \mapsto (\text{Init}, [A, B], []))) = \text{ri-runs2sigs runz}$   
 <proof>

**lemma** *ri-runs2sigs-upd-resp-none* [*simp*]:

[[  $Rb \notin \text{dom runz}$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], []))) = \text{ri-runs2sigs runz}$   
 <proof>

**lemma** *ri-runs2sigs-upd-serv-none* [*simp*]:

[[  $Rs \notin \text{dom runz}$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Rs \mapsto (\text{Serv}, [A, B], \text{nl}))) = \text{ri-runs2sigs runz}$   
 <proof>

**lemma** *ri-runs2sigs-upd-init* [*simp*]:

[[  $\text{runz Ra} = \text{Some} (\text{Init}, [A, B], [\text{aKey Kab}]); \text{finite} (\text{dom runz})$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [\text{aKey Kab}, \text{aNon Nb}]))) =$   
 ( $\text{ri-runs2sigs runz}$ )  
 ( $\text{Running } [B, A] (\text{Kab}, \text{Nb}) := \text{Suc} (\text{card} (\text{ri-running runz } A \ B \ \text{Kab} \ \text{Nb}))$ )  
 <proof>

**lemma** *ri-runs2sigs-upd-init-some* [*simp*]:

[[  $\text{runz Ra} = \text{Some} (\text{Init}, [A, B], [])$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [\text{aKey Kab}]))) = \text{ri-runs2sigs runz}$   
 <proof>

**lemma** *ri-runs2sigs-upd-resp-some* [*simp*]:

[[  $\text{runz Rb} = \text{Some} (\text{Resp}, [A, B], [])$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], [\text{aKey K}]))) = \text{ri-runs2sigs runz}$   
 <proof>

**lemma** *ri-runs2sigs-upd-resp-some2* [*simp*]:

[[  $\text{runz Rb} = \text{Some} (\text{Resp}, [A, B], [\text{aKey Kab}])$  ]]  
 $\implies \text{ri-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], [\text{aKey Kab}, \text{END}]))) =$   
 ( $\text{ri-runs2sigs runz}$ )( $\text{Commit } [B, A] (\text{Kab}, \text{Rb\$nb}) := 1$ )  
 <proof>

## Refinement proof

**lemma** *PO-m1-step1-refines-ri-a0i-skip*:

{ $R\text{-a0im1-ri}$ }  
 $\text{Id}, (\text{m1-step1 } Ra \ A \ B \ Na)$   
 { $> R\text{-a0im1-ri}$ }  
 <proof>

**lemma** *PO-m1-step2-refines-ri-a0i-skip*:

{ $R\text{-a0im1-ri}$ }  
 $\text{Id}, (\text{m1-step2 } Rb \ A \ B)$   
 { $> R\text{-a0im1-ri}$ }  
 <proof>

**lemma** *PO-m1-step3-refines-ri-a0i-skip*:

$\{R\text{-}a0im1\text{-}ri\}$   
 $Id, (m1\text{-}step3\ Rs\ A\ B\ Na\ Kab)$   
 $\{>\ R\text{-}a0im1\text{-}ri\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step4-refines-ri-a0i-skip*:  
 $\{R\text{-}a0im1\text{-}ri\}$   
 $Id, (m1\text{-}step4\ Ra\ A\ B\ Nb\ Kab)$   
 $\{>\ R\text{-}a0im1\text{-}ri\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step5-refines-ri-a0i-skip*:  
 $\{R\text{-}a0im1\text{-}ri\}$   
 $Id, (m1\text{-}step5\ Rb\ A\ B\ Nb\ Kab)$   
 $\{>\ R\text{-}a0im1\text{-}ri\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step6-refines-ri-a0i-running*:  
 $\{R\text{-}a0im1\text{-}ri \cap UNIV \times m1\text{-}inv0\text{-}fin\}$   
 $(a0i\text{-}running\ [B, A]\ (Kab, Nb)), (m1\text{-}step6\ Ra\ A\ B\ Na\ Nb\ Kab)$   
 $\{>\ R\text{-}a0im1\text{-}ri\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step7-refines-ri-a0i-commit*:  
 $\{R\text{-}a0im1\text{-}ri \cap UNIV \times m1\text{-}inv0\text{-}fin\}$   
 $(a0i\text{-}commit\ [B, A]\ (Kab, Nb)), (m1\text{-}step7\ Rb\ A\ B\ Nb\ Kab)$   
 $\{>\ R\text{-}a0im1\text{-}ri\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-leak-refines-ri-a0i-corrupt*:  
 $\{R\text{-}a0im1\text{-}ri\}$   
 $(a0i\text{-}corrupt\ (\{sesK\ (Rs\$sk)\} \times UNIV)), (m1\text{-}leak\ Rs\ Ra\ Rb\ A\ B)$   
 $\{>\ R\text{-}a0im1\text{-}ri\}$   
 $\langle proof \rangle$

All together now...

**lemmas** *PO-m1-trans-refines-ri-a0i-trans* =  
 $PO\text{-}m1\text{-}step1\text{-}refines\text{-}ri\text{-}a0i\text{-}skip\ PO\text{-}m1\text{-}step2\text{-}refines\text{-}ri\text{-}a0i\text{-}skip$   
 $PO\text{-}m1\text{-}step3\text{-}refines\text{-}ri\text{-}a0i\text{-}skip\ PO\text{-}m1\text{-}step4\text{-}refines\text{-}ri\text{-}a0i\text{-}skip$   
 $PO\text{-}m1\text{-}step5\text{-}refines\text{-}ri\text{-}a0i\text{-}skip\ PO\text{-}m1\text{-}step6\text{-}refines\text{-}ri\text{-}a0i\text{-}running$   
 $PO\text{-}m1\text{-}step7\text{-}refines\text{-}ri\text{-}a0i\text{-}commit\ PO\text{-}m1\text{-}leak\text{-}refines\text{-}ri\text{-}a0i\text{-}corrupt$

**lemma** *PO-m1-refines-ri-init-a0i* [iff]:  
 $init\ m1 \subseteq R\text{-}a0im1\text{-}ri''(init\ a0i)$   
 $\langle proof \rangle$

**lemma** *PO-m1-refines-ri-trans-a0i* [iff]:  
 $\{R\text{-}a0im1\text{-}ri \cap a0i\text{-}inv1\text{-}iagree \times m1\text{-}inv0\text{-}fin\}$   
 $(trans\ a0i), (trans\ m1)$   
 $\{>\ R\text{-}a0im1\text{-}ri\}$   
 $\langle proof \rangle$

Observation consistency.

**lemma** *obs-consistent-med-a0im1-ri* [iff]:  
*obs-consistent R-a0im1-ri med-a0im1-ri a0i m1*  
 ⟨proof⟩

Refinement result.

**lemma** *PO-m1-refines-ri-a0i* [iff]:  
*refines (R-a0im1-ri  $\cap$  a0i-inv1-iagree  $\times$  m1-inv0-fin) med-a0im1-ri a0i m1*  
 ⟨proof⟩

**lemma** *m1-implements-ri-a0i: implements med-a0im1-ri a0i m1*  
 ⟨proof⟩

### inv3 (inherited): Responder and initiator

This is a translation of the agreement property to Level 1. It follows from the refinement and is needed to prove inv4.

#### definition

*m1-inv3r-init* :: 'x m1-pred

#### where

*m1-inv3r-init*  $\equiv$  {s.  $\forall A B Rb Kab$ .  
 $B \notin bad \longrightarrow A \notin bad \longrightarrow Kab \notin Domain (leak s) \longrightarrow$   
 $runs\ s\ Rb = Some (Resp, [A, B], [aKey\ Kab, END]) \longrightarrow$   
 $(\exists Ra\ nla. runs\ s\ Ra = Some (Init, [A, B], aKey\ Kab \# aNon (Rb\$nb) \# nla))$   
 }

**lemmas** *m1-inv3r-initI* =  
*m1-inv3r-init-def [THEN setc-def-to-intro, rule-format]*

**lemmas** *m1-inv3r-initE* [elim] =  
*m1-inv3r-init-def [THEN setc-def-to-elim, rule-format]*

**lemmas** *m1-inv3r-initD* =  
*m1-inv3r-init-def [THEN setc-def-to-dest, rule-format, rotated -1]*

Invariance proof.

**lemma** *PO-m1-inv3r-init* [iff]: *reach m1  $\subseteq$  m1-inv3r-init*  
 ⟨proof⟩

### inv4: Key freshness for responder

#### definition

*m1-inv4-rfresh* :: 'x m1-pred

#### where

*m1-inv4-rfresh*  $\equiv$  {s.  $\forall Rb Rb' A A' B B' Kab$ .  
 $runs\ s\ Rb = Some (Resp, [A, B], [aKey\ Kab, END]) \longrightarrow$   
 $runs\ s\ Rb' = Some (Resp, [A', B'], [aKey\ Kab, END]) \longrightarrow$   
 $B \notin bad \longrightarrow A \notin bad \longrightarrow Kab \notin Domain (leak s) \longrightarrow$   
 $Rb = Rb'$   
 }

**lemmas** *m1-inv4-rfreshI* = *m1-inv4-rfresh-def [THEN setc-def-to-intro, rule-format]*

**lemmas** *m1-inv4-rfreshE* [elim] = *m1-inv4-rfresh-def [THEN setc-def-to-elim, rule-format]*

**lemmas** *m1-inv4-rfreshD* = *m1-inv4-rfresh-def [THEN setc-def-to-dest, rule-format, rotated 1]*

Proof of key freshness for responder

**lemma** *PO-m1-inv4-rfresh-init* [iff]:

$init\ m1 \subseteq m1\text{-inv4}\text{-rfresh}$

$\langle proof \rangle$

**lemma** *PO-m1-inv4-rfresh-trans* [iff]:

$\{m1\text{-inv4}\text{-rfresh} \cap m1\text{-inv3r}\text{-init} \cap m1\text{-inv2r}\text{-serv} \cap m1\text{-inv3}\text{-ifresh} \cap m1\text{-secrecy}\}$   
 $trans\ m1$

$\{>\ m1\text{-inv4}\text{-rfresh}\}$

$\langle proof \rangle$

**lemma** *PO-m1-inv4-rfresh* [iff]:  $reach\ m1 \subseteq m1\text{-inv4}\text{-rfresh}$

$\langle proof \rangle$

**lemma** *PO-m1-obs-inv4-rfresh* [iff]:  $oreach\ m1 \subseteq m1\text{-inv4}\text{-rfresh}$

$\langle proof \rangle$

end

### 3.10 Abstract Needham-Schroeder Shared Key (L2)

**theory** *m2-nssk* imports *m1-nssk* ../Refinement/Channels

**begin**

We model an abstract version of the Needham-Schroeder Shared Key protocol:

- M1.  $A \rightarrow S : A, B, Na$
- M2.  $S \rightarrow A : \{Na, B, Kab, \{Kab, A\}_{Kbs}\}_{Kas}$
- M3.  $A \rightarrow B : \{A, Kab\}_{Kbs}$
- M4.  $B \rightarrow A : \{Nb\}_{Kab}$
- M5.  $A \rightarrow B : \{Nb - 1\}_{Kab}$

The last two message are supposed to authenticate  $A$  to  $B$ , but this fails as shown by Dening and Sacco. Therefore and since we are mainly interested in secrecy at this point, we drop the last two message from this development.

This refinement introduces channels with security properties. We model a parallel/"channel-pure" version of the first three messages of the NSSK protocol:

- M1.  $A \rightarrow S : A, B, Na$
- M2.  $S \rightarrow A : \{Na, B, Kab\}_{Kas}$
- M3.  $S \rightarrow B : \{Kab, A\}_{Kbs}$

Message 1 is sent over an insecure channel, the other two message over secure channels to/from the server.

**declare** *domIff* [simp, iff del]

#### 3.10.1 State

**record** *m2-state* = *m1-state* +

$chan :: chmsg\ set$  — channel messages

**type-synonym**

$m2-obs = m1-state$

**definition**

$m2-obs :: m2-state \Rightarrow m2-obs$  **where**  
 $m2-obs\ s \equiv \langle \langle runs = runs\ s, leak = leak\ s \rangle \rangle$

**type-synonym**

$m2-pred = m2-state\ set$

**type-synonym**

$m2-trans = (m2-state \times m2-state)\ set$

### 3.10.2 Events

Protocol events.

**definition** — by  $A$ , refines  $m1a-step1$

$m2-step1 :: [rid-t, agent, agent, nonce] \Rightarrow m2-trans$

**where**

$m2-step1\ Ra\ A\ B\ Na \equiv \{(s, s1)\}.$

— guards:

$Ra \notin dom\ (runs\ s) \wedge$  — fresh run identifier  
 $Na = Ra\$na \wedge$  — generate nonce  $Na$

— actions:

— create initiator thread and send message 1  
 $s1 = s\langle$   
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [])),$   
 $chan := insert\ (Insec\ A\ B\ (Msg\ [aNon\ Na]))\ (chan\ s)$  — msg 1  
 $\rangle$   
 $\}$

**definition** — by  $B$ , refines  $m1a-step2$

$m2-step2 :: [rid-t, agent, agent] \Rightarrow m2-trans$

**where**

$m2-step2 \equiv m1-step2$

**definition** — by  $Server$ , refines  $m1a-step3$

$m2-step3 :: [rid-t, agent, agent, nonce, key] \Rightarrow m2-trans$

**where**

$m2-step3\ Rs\ A\ B\ Na\ Kab \equiv \{(s, s1)\}.$

— guards:

$Rs \notin dom\ (runs\ s) \wedge$  — new server run  
 $Kab = sesK\ (Rs\$sk) \wedge$  — fresh session key

$Insec\ A\ B\ (Msg\ [aNon\ Na]) \in chan\ s \wedge$  — recv msg 1

— actions:

— record key and send messages 2 and 3

— note that last field in server record is for responder nonce

$$s1 = s(\begin{array}{l} \text{runs} := (\text{runs } s)(Rs \mapsto (\text{Serv}, [A, B], [aNon Na])), \\ \text{chan} := \{\text{Secure Sv } A (\text{Msg } [aNon Na, aAgt B, aKey Kab]), \\ \quad \text{Secure Sv } B (\text{Msg } [aKey Kab, aAgt A])\} \cup \text{chan } s \end{array})$$

**definition** — by  $A$ , refines  $m1a\text{-step}4$   
 $m2\text{-step}4 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow m2\text{-trans}$

**where**  
 $m2\text{-step}4 \text{ Ra } A \text{ B } Na \text{ Kab} \equiv \{(s, s1)\}.$

— guards:  
 $\text{runs } s \text{ Ra} = \text{Some } (\text{Init}, [A, B], []) \wedge$   
 $Na = \text{Ra}\$na \wedge$

$\text{Secure Sv } A (\text{Msg } [aNon Na, aAgt B, aKey Kab]) \in \text{chan } s \wedge$  — recv msg 2

— actions:  
 — record session key  
 $s1 = s(\begin{array}{l} \text{runs} := (\text{runs } s)(\text{Ra} \mapsto (\text{Init}, [A, B], [aKey Kab])) \end{array})$

**definition** — by  $B$ , refines  $m1\text{-step}5$   
 $m2\text{-step}5 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow m2\text{-trans}$

**where**  
 $m2\text{-step}5 \text{ Rb } A \text{ B } Nb \text{ Kab} \equiv \{(s, s1)\}.$

— guards:  
 $\text{runs } s \text{ Rb} = \text{Some } (\text{Resp}, [A, B], []) \wedge$   
 $Nb = \text{Rb}\$nb \wedge$

$\text{Secure Sv } B (\text{Msg } [aKey Kab, aAgt A]) \in \text{chan } s \wedge$  — recv msg 3

— actions:  
 — record session key  
 $s1 = s(\begin{array}{l} \text{runs} := (\text{runs } s)(\text{Rb} \mapsto (\text{Resp}, [A, B], [aKey Kab])), \\ \text{chan} := \text{insert } (\text{dAuth Kab } (\text{Msg } [aNon Nb])) (\text{chan } s) \end{array})$

**definition** — by  $A$ , refines  $m1\text{-step}6$   
 $m2\text{-step}6 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{nonce}, \text{key}] \Rightarrow m2\text{-trans}$

**where**  
 $m2\text{-step}6 \text{ Ra } A \text{ B } Na \text{ Nb } Kab \equiv \{(s, s')\}.$   
 $\text{runs } s \text{ Ra} = \text{Some } (\text{Init}, [A, B], [aKey Kab]) \wedge$  — key recv'd before  
 $Na = \text{Ra}\$na \wedge$

$\text{dAuth Kab } (\text{Msg } [aNon Nb]) \in \text{chan } s \wedge$  — receive  $M4$

— actions:  
 $s' = s \langle$   
 $\quad runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNon\ Nb])),$   
 $\quad chan := insert\ (dAuth\ Kab\ (Msg\ [aNon\ Nb, aNon\ Nb]))\ (chan\ s)$   
 $\rangle$   
 $\}$

**definition** — by  $B$ , refines  $m1\text{-}step6$   
 $m2\text{-}step7 :: [rid\text{-}t, agent, agent, nonce, key] \Rightarrow m2\text{-}trans$

**where**

$m2\text{-}step7\ Rb\ A\ B\ Nb\ Kab \equiv \{(s, s')\}.$   
 $runs\ s\ Rb = Some\ (Resp, [A, B], [aKey\ Kab]) \wedge$  — key rcv'd before  
 $Nb = Rb\$nb \wedge$

$dAuth\ Kab\ (Msg\ [aNon\ Nb, aNon\ Nb]) \in chan\ s \wedge$  — receive  $M5$

— actions: (redundant) update local state marks successful termination

$s' = s \langle$   
 $\quad runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, END]))$   
 $\rangle$   
 $\}$

Intruder fake event.

**definition** — refines  $m1\text{-}leak$   
 $m2\text{-}leak :: [rid\text{-}t, rid\text{-}t, rid\text{-}t, agent, agent] \Rightarrow m2\text{-}trans$

**where**

$m2\text{-}leak\ Rs\ Ra\ Rb\ A\ B \equiv \{(s, s1)\}.$

— guards:

$runs\ s\ Rs = Some\ (Serv, [A, B], [aNon\ (Ra\$na)]) \wedge$   
 $runs\ s\ Ra = Some\ (Init, [A, B], [aKey\ (sesK\ (Rs\$sk)), aNon\ (Rb\$nb)]) \wedge$   
 $runs\ s\ Rb = Some\ (Resp, [A, B], [aKey\ (sesK\ (Rs\$sk)), END]) \wedge$

— actions:

$s1 = s \langle$   
 $\quad leak := insert\ (sesK\ (Rs\$sk), Ra\$na, Rb\$nb)\ (leak\ s),$   
 $\quad chan := insert\ (Insec\ undefined\ undefined\ (Msg\ [aKey\ (sesK\ (Rs\$sk))]))\ (chan\ s) \rangle$   
 $\}$

**definition** — refines  $Id$

$m2\text{-}fake :: m2\text{-}trans$

**where**

$m2\text{-}fake \equiv \{(s, s1)\}.$

— actions:

$s1 = s \langle$   
 $\quad chan := fake\ ik0\ (dom\ (runs\ s))\ (chan\ s)$   
 $\rangle$   
 $\}$

### 3.10.3 Transition system

**definition**

$m2\text{-}init :: m2\text{-}pred$

**where**

```
m2-init ≡ { (|
  runs = Map.empty,
  leak = corrKey × {undefined} × {undefined},
  chan = {} )
}
```

**definition**

```
m2-trans :: m2-trans where
m2-trans ≡ (| A B Ra Rb Rs Na Nb Kab.
  m2-step1 Ra A B Na ∪
  m2-step2 Rb A B ∪
  m2-step3 Rs A B Na Kab ∪
  m2-step4 Ra A B Na Kab ∪
  m2-step5 Rb A B Nb Kab ∪
  m2-step6 Ra A B Na Nb Kab ∪
  m2-step7 Rb A B Nb Kab ∪
  m2-leak Rs Ra Rb A B ∪
  m2-fake ∪
  Id
)
```

**definition**

```
m2 :: (m2-state, m2-obs) spec where
m2 ≡ (|
  init = m2-init,
  trans = m2-trans,
  obs = m2-obs
)
```

**lemmas** *m2-loc-defs* =

```
m2-def m2-init-def m2-trans-def m2-obs-def
m2-step1-def m2-step2-def m2-step3-def m2-step4-def m2-step5-def
m2-step6-def m2-step7-def m2-leak-def m2-fake-def
```

**lemmas** *m2-defs* = *m2-loc-defs* *m1-defs*

### 3.10.4 Invariants

#### inv1: Key definedness

All session keys in channel messages stem from existing runs.

**definition**

```
m2-inv1-keys :: m2-pred
```

**where**

```
m2-inv1-keys ≡ { s. ∀ R.
  aKey (sesK (R$sk)) ∈ atoms (chan s) ∨ sesK (R$sk) ∈ Domain (leak s) →
  R ∈ dom (runs s)
}
```

**lemmas** *m2-inv1-keysI* = *m2-inv1-keys-def* [*THEN setc-def-to-intro*, *rule-format*]

**lemmas** *m2-inv1-keysE* [*elim*] = *m2-inv1-keys-def* [*THEN setc-def-to-elim*, *rule-format*]

**lemmas**  $m2\text{-inv1-keys}D = m2\text{-inv1-keys-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv1-keys-init}$  [iff]:

$init\ m2 \subseteq m2\text{-inv1-keys}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv1-keys-trans}$  [iff]:

$\{m2\text{-inv1-keys}\ trans\ m2 \{>\ m2\text{-inv1-keys}\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv1-keys}$  [iff]:  $reach\ m2 \subseteq m2\text{-inv1-keys}$

$\langle proof \rangle$

## inv2: Definedness of used keys

**definition**

$m2\text{-inv2-keys-for} :: m2\text{-pred}$

**where**

$m2\text{-inv2-keys-for} \equiv \{s. \forall R.$

$sesK\ (R\$sk) \in keys\text{-for}\ (chan\ s) \longrightarrow R \in dom\ (runs\ s)$

$\}$

**lemmas**  $m2\text{-inv2-keys-for}I = m2\text{-inv2-keys-for-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv2-keys-for}E$  [elim] =  $m2\text{-inv2-keys-for-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv2-keys-for}D = m2\text{-inv2-keys-for-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv2-keys-for-init}$  [iff]:

$init\ m2 \subseteq m2\text{-inv2-keys-for}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv2-keys-for-trans}$  [iff]:

$\{m2\text{-inv2-keys-for} \cap m2\text{-inv1-keys}\ trans\ m2 \{>\ m2\text{-inv2-keys-for}\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv2-keys-for}$  [iff]:  $reach\ m2 \subseteq m2\text{-inv2-keys-for}$

$\langle proof \rangle$

Useful application of invariant.

**lemma**  $m2\text{-inv2-keys-for--extr-insert-key}$ :

$\llbracket R \notin dom\ (runs\ s); s \in m2\text{-inv2-keys-for} \rrbracket$

$\implies extr\ (insert\ (aKey\ (sesK\ (R\$sk)))\ T)\ (chan\ s) = insert\ (aKey\ (sesK\ (R\$sk)))\ (extr\ T\ (chan\ s))$

$\langle proof \rangle$

## inv2b: leaked keys include corrupted ones

**definition**

$m2\text{-inv2b-corrKey-leaked} :: m2\text{-pred}$

**where**

$m2\text{-inv2b-corrKey-leaked} \equiv \{s. \forall K.$

$K \in corrKey \longrightarrow K \in Domain\ (leak\ s)$

}

**lemmas**  $m2\text{-inv}2b\text{-corrKey-leaked}I = m2\text{-inv}2b\text{-corrKey-leaked-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv}2b\text{-corrKey-leaked}E$  [elim] =  $m2\text{-inv}2b\text{-corrKey-leaked-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv}2b\text{-corrKey-leaked}D = m2\text{-inv}2b\text{-corrKey-leaked-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv}2b\text{-corrKey-leaked-init}$  [iff]:

$init\ m2 \subseteq m2\text{-inv}2b\text{-corrKey-leaked}$

<proof>

**lemma**  $PO\text{-}m2\text{-inv}2b\text{-corrKey-leaked-trans}$  [iff]:

$\{m2\text{-inv}2b\text{-corrKey-leaked} \cap m2\text{-inv}1\text{-keys}\} trans\ m2 \{>\ m2\text{-inv}2b\text{-corrKey-leaked}\}$

<proof>

**lemma**  $PO\text{-}m2\text{-inv}2b\text{-corrKey-leaked}$  [iff]:  $reach\ m2 \subseteq m2\text{-inv}2b\text{-corrKey-leaked}$

<proof>

### inv3a: Session key compromise

A L2 version of a session key compromise invariant. Roughly, it states that adding a set of keys  $KK$  to the parameter  $T$  of  $extr$  does not help the intruder to extract keys other than those in  $KK$  or extractable without adding  $KK$ .

#### definition

$m2\text{-inv}3a\text{-sesK-compr} :: m2\text{-state set}$

where

$m2\text{-inv}3a\text{-sesK-compr} \equiv \{s. \forall K\ KK.$

~~$KK \neq \{ \} \wedge \text{set}K \neq \{ \}$~~

$aKey\ K \in extr\ (aKey\ KK \cup ik0)\ (chan\ s) \longleftrightarrow (K \in KK \vee aKey\ K \in extr\ ik0\ (chan\ s))$

}

**lemmas**  $m2\text{-inv}3a\text{-sesK-compr}I = m2\text{-inv}3a\text{-sesK-compr-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv}3a\text{-sesK-compr}E$  [elim] =  $m2\text{-inv}3a\text{-sesK-compr-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv}3a\text{-sesK-compr}D = m2\text{-inv}3a\text{-sesK-compr-def}$  [THEN setc-def-to-dest, rule-format]

Additional lemma to get the keys in front

**lemmas**  $insert\ commute\ aKey = insert\ commute$  [where  $x=aKey\ K$  for  $K$ ]

**lemmas**  $m2\text{-inv}3a\text{-sesK-compr-simps} =$

$m2\text{-inv}3a\text{-sesK-compr}D$

$m2\text{-inv}3a\text{-sesK-compr}D$  [where  $KK=\{Kab\}$  for  $Kab$ , simplified]

$m2\text{-inv}3a\text{-sesK-compr}D$  [where  $KK=insert\ Kab\ KK$  for  $Kab\ KK$ , simplified]

$insert\ commute\ aKey$

**lemma**  $PO\text{-}m2\text{-inv}3a\text{-sesK-compr-init}$  [iff]:

$init\ m2 \subseteq m2\text{-inv}3a\text{-sesK-compr}$

<proof>

**lemma**  $PO\text{-}m2\text{-inv}3a\text{-sesK-compr-trans}$  [iff]:



where

$$\begin{aligned}
& m2\text{-inv3}\text{-extrKey} \equiv \{s. \forall K. \\
& \quad aKey\ K \in \text{extr ik0} (\text{chan } s) \longrightarrow K \notin \text{corrKey} \longrightarrow \\
& \quad (\exists R\ A'\ B'\ Na'. K = \text{sesK} (R\$sk) \wedge \\
& \quad \quad \text{runs } s\ R = \text{Some} (\text{Serv}, [A', B'], [aNon\ Na']) \wedge \\
& \quad \quad (A' \in \text{bad} \vee B' \in \text{bad} \vee (\exists Nb'. (K, Na', Nb') \in \text{leak } s))) \\
& \}
\end{aligned}$$

**lemmas**  $m2\text{-inv3}\text{-extrKeyI} = m2\text{-inv3}\text{-extrKey}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv3}\text{-extrKeyE}$  [elim] =  $m2\text{-inv3}\text{-extrKey}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv3}\text{-extrKeyD} = m2\text{-inv3}\text{-extrKey}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

**lemma**  $PO\text{-}m2\text{-inv3}\text{-extrKey}\text{-init}$  [iff]:

$\text{init } m2 \subseteq m2\text{-inv3}\text{-extrKey}$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m2\text{-inv3}\text{-extrKey}\text{-trans}$  [iff]:

$\{m2\text{-inv3}\text{-extrKey} \cap m2\text{-inv3a}\text{-sesK}\text{-compr}\} \text{ trans } m2 \{> m2\text{-inv3}\text{-extrKey}\}$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m2\text{-inv3}\text{-extrKey}$  [iff]:  $\text{reach } m2 \subseteq m2\text{-inv3}\text{-extrKey}$

$\langle \text{proof} \rangle$

## inv4: Secure channel and message 2

inv4: Secure messages to honest agents and server state; one variant for each of M2 and M3. Note that the one for M2 is stronger than the one for M3.

**definition**

$m2\text{-inv4}\text{-M2} :: m2\text{-pred}$

where

$$\begin{aligned}
& m2\text{-inv4}\text{-M2} \equiv \{s. \forall A\ B\ Na\ Kab. \\
& \quad \text{Secure Sv } A (\text{Msg } [aNon\ Na, aAgt\ B, aKey\ Kab]) \in \text{chan } s \longrightarrow A \in \text{good} \longrightarrow \\
& \quad (\exists Rs. Kab = \text{sesK} (Rs\$sk) \wedge \text{runs } s\ Rs = \text{Some} (\text{Serv}, [A, B], [aNon\ Na])) \\
& \}
\end{aligned}$$

**lemmas**  $m2\text{-inv4}\text{-M2I} = m2\text{-inv4}\text{-M2}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv4}\text{-M2E}$  [elim] =  $m2\text{-inv4}\text{-M2}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv4}\text{-M2D} = m2\text{-inv4}\text{-M2}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv4}\text{-M2}\text{-init}$  [iff]:

$\text{init } m2 \subseteq m2\text{-inv4}\text{-M2}$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m2\text{-inv4}\text{-M2}\text{-trans}$  [iff]:

$\{m2\text{-inv4}\text{-M2}\} \text{ trans } m2 \{> m2\text{-inv4}\text{-M2}\}$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m2\text{-inv4}\text{-M2}$  [iff]:  $\text{reach } m2 \subseteq m2\text{-inv4}\text{-M2}$

$\langle \text{proof} \rangle$

## inv4b: Secure channel and message 3

### definition

$m2\text{-inv4}\text{-}M3 :: m2\text{-pred}$

### where

$m2\text{-inv4}\text{-}M3 \equiv \{s. \forall A B Kab.$

$Secure\ Sv\ B\ (Msg\ [aKey\ Kab,\ aAgt\ A]) \in\ chan\ s \longrightarrow B \in\ good \longrightarrow$

$(\exists\ Rs\ Na.\ Kab =\ sesK\ (Rs\$sk) \wedge\ runs\ s\ Rs =\ Some\ (Serv,\ [A,\ B],\ [aNon\ Na]))$

$\}$

**lemmas**  $m2\text{-inv4}\text{-}M3I = m2\text{-inv4}\text{-}M3\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

**lemmas**  $m2\text{-inv4}\text{-}M3E\ [elim] = m2\text{-inv4}\text{-}M3\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

**lemmas**  $m2\text{-inv4}\text{-}M3D = m2\text{-inv4}\text{-}M3\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv4}\text{-}M3\text{-init}\ [iff]:$

$init\ m2 \subseteq m2\text{-inv4}\text{-}M3$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv4}\text{-}M3\text{-trans}\ [iff]:$

$\{m2\text{-inv4}\text{-}M3\}\ trans\ m2\ \{>\ m2\text{-inv4}\text{-}M3\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv4}\text{-}M3\ [iff]:\ reach\ m2 \subseteq m2\text{-inv4}\text{-}M3$

$\langle proof \rangle$

Consequence needed in proof of inv8/step5

**lemma**  $m2\text{-inv4}\text{-}M2\text{-}M3\text{-unique}\text{-}names:$

### assumes

$Secure\ Sv\ A'\ (Msg\ [aNon\ Na,\ aAgt\ B',\ aKey\ Kab]) \in\ chan\ s$

$Secure\ Sv\ B\ (Msg\ [aKey\ Kab,\ aAgt\ A]) \in\ chan\ s\ aKey\ Kab \notin\ extr\ ik0\ (chan\ s)$

$s \in m2\text{-inv4}\text{-}M2\ s \in m2\text{-inv4}\text{-}M3$

### shows

$A = A' \wedge B = B'$

$\langle proof \rangle$

More consequences of invariants. Needed in ref/step4 and ref/step5 respectively to show the strengthening of the authorization guards.

**lemma**  $m2\text{-inv34}\text{-}M2\text{-authorized}:$

**assumes**  $Secure\ Sv\ A\ (Msg\ [aNon\ N,\ aAgt\ B,\ aKey\ K]) \in\ chan\ s$

$s \in m2\text{-inv4}\text{-}M2\ s \in m2\text{-inv3}\text{-}extrKey\ s \in m2\text{-inv2b}\text{-}corrKey\text{-}leaked$

$K \notin\ Domain\ (leak\ s)$

**shows**  $(K,\ A) \in\ azC\ (runs\ s)$

$\langle proof \rangle$

**lemma**  $m2\text{-inv34}\text{-}M3\text{-authorized}:$

**assumes**  $Secure\ Sv\ B\ (Msg\ [aKey\ K,\ aAgt\ A]) \in\ chan\ s$

$s \in m2\text{-inv4}\text{-}M3\ s \in m2\text{-inv3}\text{-}extrKey\ s \in m2\text{-inv2b}\text{-}corrKey\text{-}leaked$

$K \notin\ Domain\ (leak\ s)$

**shows**  $(K,\ B) \in\ azC\ (runs\ s)$

$\langle proof \rangle$

### inv5 (derived): Key secrecy for server

inv5: Key secrecy from server perspective. This invariant links the abstract notion of key secrecy to the intruder key knowledge.

#### definition

$m2\text{-inv5-ikk-sv} :: m2\text{-pred}$

#### where

$m2\text{-inv5-ikk-sv} \equiv \{s. \forall Rs A B Na al.$   
 $runs\ s\ Rs = Some\ (Serv, [A, B], aNon\ Na\ \# al) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$   
 $aKey\ (sesK\ (Rs\$sk)) \in extr\ ik0\ (chan\ s) \longrightarrow$   
 $(\exists Nb'. (sesK\ (Rs\$sk), Na, Nb') \in leak\ s)$   
 $\}$

**lemmas**  $m2\text{-inv5-ikk-svI} = m2\text{-inv5-ikk-sv-def}\ [THEN\ setc\text{-def-to-intro},\ rule\text{-format}]$

**lemmas**  $m2\text{-inv5-ikk-svE}\ [elim] = m2\text{-inv5-ikk-sv-def}\ [THEN\ setc\text{-def-to-elim},\ rule\text{-format}]$

**lemmas**  $m2\text{-inv5-ikk-svD} = m2\text{-inv5-ikk-sv-def}\ [THEN\ setc\text{-def-to-dest},\ rule\text{-format},\ rotated\ 1]$

Invariance proof. This invariant follows from  $m2\text{-inv3-extrKey}$ .

**lemma**  $m2\text{-inv5-ikk-sv-derived}$ :

$s \in m2\text{-inv3-extrKey} \implies s \in m2\text{-inv5-ikk-sv}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv5-ikk-sv}\ [iff]$ :  $reach\ m2 \subseteq m2\text{-inv5-ikk-sv}$

$\langle proof \rangle$

### inv6 (derived): Key secrecy for initiator

This invariant is derivable (see below).

#### definition

$m2\text{-inv6-ikk-init} :: m2\text{-pred}$

#### where

$m2\text{-inv6-ikk-init} \equiv \{s. \forall Ra K A B al.$   
 $runs\ s\ Ra = Some\ (Init, [A, B], aKey\ K\ \# al) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$   
 $aKey\ K \in extr\ ik0\ (chan\ s) \longrightarrow$   
 $(\exists Nb'. (K, Ra\ \$\ na, Nb') \in leak\ s)$   
 $\}$

**lemmas**  $m2\text{-inv6-ikk-initI} = m2\text{-inv6-ikk-init-def}\ [THEN\ setc\text{-def-to-intro},\ rule\text{-format}]$

**lemmas**  $m2\text{-inv6-ikk-initE}\ [elim] = m2\text{-inv6-ikk-init-def}\ [THEN\ setc\text{-def-to-elim},\ rule\text{-format}]$

**lemmas**  $m2\text{-inv6-ikk-initD} = m2\text{-inv6-ikk-init-def}\ [THEN\ setc\text{-def-to-dest},\ rule\text{-format},\ rotated\ 1]$

### inv7 (derived): Key secrecy for responder

This invariant is derivable (see below).

#### definition

$m2\text{-inv7-ikk-resp} :: m2\text{-pred}$

#### where

$m2\text{-inv7-ikk-resp} \equiv \{s. \forall Rb K A B al.$   
 $runs\ s\ Rb = Some\ (Resp, [A, B], aKey\ K\ \# al) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$   
 $aKey\ K \in extr\ ik0\ (chan\ s) \longrightarrow$   
 $K \in Domain\ (leak\ s)$   
 $\}$

}

**lemmas**  $m2\text{-inv}7\text{-ikk}\text{-resp}I = m2\text{-inv}7\text{-ikk}\text{-resp}\text{-def}$  [*THEN setc-def-to-intro, rule-format*]

**lemmas**  $m2\text{-inv}7\text{-ikk}\text{-resp}E$  [*elim*] =  $m2\text{-inv}7\text{-ikk}\text{-resp}\text{-def}$  [*THEN setc-def-to-elim, rule-format*]

**lemmas**  $m2\text{-inv}7\text{-ikk}\text{-resp}D = m2\text{-inv}7\text{-ikk}\text{-resp}\text{-def}$  [*THEN setc-def-to-dest, rule-format, rotated 1*]

### inv8: Relating M2 and M4 to the responder state

This invariant relates messages M2 and M4 to the responder's state. It is required in the refinement of step 6 to prove that the initiator agrees with the responder on (A, B, Nb, Kab).

#### definition

$m2\text{-inv}8\text{-M}4 :: m2\text{-pred}$

#### where

$m2\text{-inv}8\text{-M}4 \equiv \{s. \forall Kab\ A\ B\ Na\ Nb.$

$Secure\ Sv\ A\ (Msg\ [aNon\ Na,\ aAgt\ B,\ aKey\ Kab]) \in chan\ s \longrightarrow$

$dAuth\ Kab\ (Msg\ [aNon\ Nb]) \in chan\ s \longrightarrow$

$aKey\ Kab \notin extr\ ik0\ (chan\ s) \longrightarrow$

$(\exists Rb. Nb = Rb\$nb \wedge (\exists al. runs\ s\ Rb = Some\ (Resp,\ [A,\ B],\ aKey\ Kab\ \# al)))$

}

**lemmas**  $m2\text{-inv}8\text{-M}4I = m2\text{-inv}8\text{-M}4\text{-def}$  [*THEN setc-def-to-intro, rule-format*]

**lemmas**  $m2\text{-inv}8\text{-M}4E$  [*elim*] =  $m2\text{-inv}8\text{-M}4\text{-def}$  [*THEN setc-def-to-elim, rule-format*]

**lemmas**  $m2\text{-inv}8\text{-M}4D = m2\text{-inv}8\text{-M}4\text{-def}$  [*THEN setc-def-to-dest, rule-format, rotated 1*]

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv}8\text{-M}4\text{-step}1:$

$\{m2\text{-inv}8\text{-M}4\}\ m2\text{-step}1\ Ra\ A\ B\ Na\ \{>\ m2\text{-inv}8\text{-M}4\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv}8\text{-M}4\text{-step}2:$

$\{m2\text{-inv}8\text{-M}4\}\ m2\text{-step}2\ Rb\ A\ B\ \{>\ m2\text{-inv}8\text{-M}4\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv}8\text{-M}4\text{-step}3:$

$\{m2\text{-inv}8\text{-M}4 \cap m2\text{-inv}2\text{-keys}\text{-for}\}\ m2\text{-step}3\ Rs\ A\ B\ Na\ Kab\ \{>\ m2\text{-inv}8\text{-M}4\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv}8\text{-M}4\text{-step}4:$

$\{m2\text{-inv}8\text{-M}4\}\ m2\text{-step}4\ Ra\ A\ B\ Na\ Kab\ \{>\ m2\text{-inv}8\text{-M}4\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv}8\text{-M}4\text{-step}5:$

$\{m2\text{-inv}8\text{-M}4 \cap m2\text{-inv}4\text{-M}3 \cap m2\text{-inv}4\text{-M}2\}$

$m2\text{-step}5\ Rb\ A\ B\ Nb\ Kab$

$\{>\ m2\text{-inv}8\text{-M}4\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv}8\text{-M}4\text{-step}6:$

$\{m2\text{-inv}8\text{-M}4\}\ m2\text{-step}6\ Ra\ A\ B\ Na\ Nb\ Kab\ \{>\ m2\text{-inv}8\text{-M}4\}$

$\langle proof \rangle$

**lemma** *PO-m2-inv8-M4-step7*:  
 $\{m2\text{-inv8-M4}\} m2\text{-step7 } Rb \ A \ B \ Nb \ Kab \ \{> \ m2\text{-inv8-M4}\}$   
 $\langle\text{proof}\rangle$

**lemma** *PO-m2-inv8-M4-leak*:  
 $\{m2\text{-inv8-M4} \cap m2\text{-inv3a-sesK-compr}\} m2\text{-leak } Rs \ Ra \ Rb \ A \ B \ \{> \ m2\text{-inv8-M4}\}$   
 $\langle\text{proof}\rangle$

**lemma** *PO-m2-inv8-M4-fake*:  
 $\{m2\text{-inv8-M4}\} m2\text{-fake} \ \{> \ m2\text{-inv8-M4}\}$   
 $\langle\text{proof}\rangle$

All together now..

**lemmas** *PO-m2-inv8-M4-lemmas* =  
*PO-m2-inv8-M4-step1 PO-m2-inv8-M4-step2 PO-m2-inv8-M4-step3*  
*PO-m2-inv8-M4-step4 PO-m2-inv8-M4-step5 PO-m2-inv8-M4-step6*  
*PO-m2-inv8-M4-step7 PO-m2-inv8-M4-leak PO-m2-inv8-M4-fake*

**lemma** *PO-m2-inv8-M4-init* [iff]:  
 $init \ m2 \subseteq m2\text{-inv8-M4}$   
 $\langle\text{proof}\rangle$

**lemma** *PO-m2-inv8-M4-trans* [iff]:  
 $\{m2\text{-inv8-M4} \cap m2\text{-inv4-M3} \cap m2\text{-inv4-M2} \cap m2\text{-inv3a-sesK-compr} \cap m2\text{-inv2-keys-for}\}$   
 $trans \ m2$   
 $\{> \ m2\text{-inv8-M4}\}$   
 $\langle\text{proof}\rangle$

**lemma** *PO-m2-inv8-M4* [iff]:  $reach \ m2 \subseteq m2\text{-inv8-M4}$   
 $\langle\text{proof}\rangle$

## inv8a: Relating the initiator state to M2

### definition

$m2\text{-inv8a-init-M2} :: m2\text{-pred}$

### where

$m2\text{-inv8a-init-M2} \equiv \{s. \forall Ra \ A \ B \ Kab \ al.$   
 $runs \ s \ Ra = Some \ (Init, [A, B], aKey \ Kab \ \# \ al) \longrightarrow$   
 $Secure \ Sv \ A \ (Msg \ [aNon \ (Ra\$na), aAgt \ B, aKey \ Kab]) \in chan \ s$   
 $\}$

**lemmas**  $m2\text{-inv8a-init-M2I} = m2\text{-inv8a-init-M2-def} \ [THEN \ setc\text{-def-to-intro}, rule\text{-format}]$

**lemmas**  $m2\text{-inv8a-init-M2E} \ [elim] = m2\text{-inv8a-init-M2-def} \ [THEN \ setc\text{-def-to-elim}, rule\text{-format}]$

**lemmas**  $m2\text{-inv8a-init-M2D} = m2\text{-inv8a-init-M2-def} \ [THEN \ setc\text{-def-to-dest}, rule\text{-format}, rotated \ 1]$

Invariance proof.

**lemma** *PO-m2-inv8a-init-M2-init* [iff]:  
 $init \ m2 \subseteq m2\text{-inv8a-init-M2}$   
 $\langle\text{proof}\rangle$

**lemma** *PO-m2-inv8a-init-M2-trans* [iff]:  
 $\{m2\text{-inv8a-init-M2}\}$

$trans\ m2$   
 $\{> m2\text{-inv8a-init-M2}\}$   
 $\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv8a-init-M2}$  [iff]:  $reach\ m2 \subseteq m2\text{-inv8a-init-M2}$   
 $\langle proof \rangle$

### inv9a: Relating the responder state to M3

#### definition

$m2\text{-inv9a-resp-M3} :: m2\text{-pred}$

#### where

$m2\text{-inv9a-resp-M3} \equiv \{s. \forall Rb\ A\ B\ Kab\ al.$   
 $runs\ s\ Rb = Some\ (Resp,\ [A,\ B],\ aKey\ Kab\ \# \ al) \longrightarrow$   
 $Secure\ Sv\ B\ (Msg\ [aKey\ Kab,\ aAgt\ A]) \in\ chan\ s$   
 $\}$

**lemmas**  $m2\text{-inv9a-resp-M3I} = m2\text{-inv9a-resp-M3-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv9a-resp-M3E}$  [elim] =  $m2\text{-inv9a-resp-M3-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv9a-resp-M3D} = m2\text{-inv9a-resp-M3-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv9a-resp-M3-init}$  [iff]:

$init\ m2 \subseteq m2\text{-inv9a-resp-M3}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv9a-resp-M3-trans}$  [iff]:

$\{m2\text{-inv9a-resp-M3}\}$

$trans\ m2$

$\{> m2\text{-inv9a-resp-M3}\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m2\text{-inv9a-resp-M3}$  [iff]:  $reach\ m2 \subseteq m2\text{-inv9a-resp-M3}$

$\langle proof \rangle$

### inv9: Relating M3 and M5 to the initiator state

This invariant relates message M5 to the initiator's state. It is required in step 7 of the refinement to prove that the initiator agrees with the responder on (A, B, Nb, Kab).

#### definition

$m2\text{-inv9-M5} :: m2\text{-pred}$

#### where

$m2\text{-inv9-M5} \equiv \{s. \forall Kab\ A\ B\ Nb.$   
 $Secure\ Sv\ B\ (Msg\ [aKey\ Kab,\ aAgt\ A]) \in\ chan\ s \longrightarrow$   
 $dAuth\ Kab\ (Msg\ [aNon\ Nb,\ aNon\ Nb]) \in\ chan\ s \longrightarrow$   
 $aKey\ Kab \notin\ extr\ ik0\ (chan\ s) \longrightarrow$   
 $(\exists Ra. runs\ s\ Ra = Some\ (Init,\ [A,\ B],\ [aKey\ Kab,\ aNon\ Nb]))$   
 $\}$

**lemmas**  $m2\text{-inv9-M5I} = m2\text{-inv9-M5-def}$  [*THEN setc-def-to-intro, rule-format*]  
**lemmas**  $m2\text{-inv9-M5E}$  [*elim*] =  $m2\text{-inv9-M5-def}$  [*THEN setc-def-to-elim, rule-format*]  
**lemmas**  $m2\text{-inv9-M5D} = m2\text{-inv9-M5-def}$  [*THEN setc-def-to-dest, rule-format, rotated 1*]

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv9-M5-step1}$ :  
 $\{m2\text{-inv9-M5}\} m2\text{-step1 Ra A B Na \{> m2\text{-inv9-M5}\}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m2\text{-inv9-M5-step2}$ :  
 $\{m2\text{-inv9-M5}\} m2\text{-step2 Rb A B \{> m2\text{-inv9-M5}\}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m2\text{-inv9-M5-step3}$ :  
 $\{m2\text{-inv9-M5} \cap m2\text{-inv2-keys-for}\} m2\text{-step3 Rs A B Na Kab \{> m2\text{-inv9-M5}\}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m2\text{-inv9-M5-step4}$ :  
 $\{m2\text{-inv9-M5}\} m2\text{-step4 Ra A B Na Kab \{> m2\text{-inv9-M5}\}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m2\text{-inv9-M5-step5}$ :  
 $\{m2\text{-inv9-M5}\} m2\text{-step5 Rb A B Nb Kab \{> m2\text{-inv9-M5}\}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m2\text{-inv9-M5-step6}$ :  
 $\{m2\text{-inv9-M5} \cap m2\text{-inv8a-init-M2} \cap m2\text{-inv9a-resp-M3} \cap m2\text{-inv4-M2} \cap m2\text{-inv4-M3}\}$   
 $m2\text{-step6 Ra A B Na Nb Kab}$   
 $\{> m2\text{-inv9-M5}\}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m2\text{-inv9-M5-step7}$ :  
 $\{m2\text{-inv9-M5}\} m2\text{-step7 Rb A B Nb Kab \{> m2\text{-inv9-M5}\}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m2\text{-inv9-M5-leak}$ :  
 $\{m2\text{-inv9-M5} \cap m2\text{-inv3a-sesK-compr}\} m2\text{-leak Rs Ra Rb A B \{> m2\text{-inv9-M5}\}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m2\text{-inv9-M5-fake}$ :  
 $\{m2\text{-inv9-M5}\} m2\text{-fake \{> m2\text{-inv9-M5}\}$   
 $\langle\text{proof}\rangle$

All together now.

**lemmas**  $PO\text{-}m2\text{-inv9-M5-lemmas} =$   
 $PO\text{-}m2\text{-inv9-M5-step1 PO\text{-}m2\text{-inv9-M5-step2 PO\text{-}m2\text{-inv9-M5-step3}$   
 $PO\text{-}m2\text{-inv9-M5-step4 PO\text{-}m2\text{-inv9-M5-step5 PO\text{-}m2\text{-inv9-M5-step6}$   
 $PO\text{-}m2\text{-inv9-M5-step7 PO\text{-}m2\text{-inv9-M5-leak PO\text{-}m2\text{-inv9-M5-fake}$

**lemma**  $PO\text{-}m2\text{-inv9-M5-init}$  [*iff*]:  
 $init\ m2 \subseteq m2\text{-inv9-M5}$   
 $\langle\text{proof}\rangle$

**lemma** *PO-m2-inv9-M5-trans* [iff]:  
 $\{m2\text{-inv9-M5} \cap m2\text{-inv8a-init-M2} \cap m2\text{-inv9a-resp-M3} \cap$   
 $m2\text{-inv4-M2} \cap m2\text{-inv4-M3} \cap m2\text{-inv3a-sesK-compr} \cap m2\text{-inv2-keys-for}\}$   
 $\text{trans } m2$   
 $\{> m2\text{-inv9-M5}\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m2-inv9-M5* [iff]:  $\text{reach } m2 \subseteq m2\text{-inv9-M5}$   
 $\langle \text{proof} \rangle$

### 3.10.5 Refinement

The simulation relation. This is a pure superposition refinement.

**definition**

$R12 :: (m1\text{-state} \times m2\text{-state}) \text{ set where}$   
 $R12 \equiv \{(s, t). \text{runs } s = \text{runs } t \wedge \text{leak } s = \text{leak } t\}$

The mediator function projects on the local states.

**definition**

$\text{med21} :: m2\text{-obs} \Rightarrow m1\text{-obs where}$   
 $\text{med21 } o2 = (\text{runs} = \text{runs } o2, \text{leak} = \text{leak } o2)$

Refinement proof.

**lemma** *PO-m2-step1-refines-m1-step1*:  
 $\{R12\}$   
 $(m1\text{-step1 } Ra \ A \ B \ Na), (m2\text{-step1 } Ra \ A \ B \ Na)$   
 $\{> R12\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m2-step2-refines-m1-step2*:  
 $\{R12\}$   
 $(m1\text{-step2 } Rb \ A \ B), (m2\text{-step2 } Rb \ A \ B)$   
 $\{> R12\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m2-step3-refines-m1-step3*:  
 $\{R12\}$   
 $(m1\text{-step3 } Rs \ A \ B \ Na \ Kab), (m2\text{-step3 } Rs \ A \ B \ Na \ Kab)$   
 $\{> R12\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m2-step4-refines-m1-step4*:  
 $\{R12 \cap UNIV \times (m2\text{-inv4-M2} \cap m2\text{-inv3-extrKey} \cap m2\text{-inv2b-corrKey-leaked})\}$   
 $(m1\text{-step4 } Ra \ A \ B \ Na \ Kab), (m2\text{-step4 } Ra \ A \ B \ Na \ Kab)$   
 $\{> R12\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m2-step5-refines-m1-step5*:  
 $\{R12 \cap UNIV \times (m2\text{-inv4-M3} \cap m2\text{-inv3-extrKey} \cap m2\text{-inv2b-corrKey-leaked})\}$   
 $(m1\text{-step5 } Rb \ A \ B \ Nb \ Kab), (m2\text{-step5 } Rb \ A \ B \ Nb \ Kab)$   
 $\{> R12\}$

*<proof>*

**lemma** *PO-m2-step6-refines-m1-step6:*

$\{R12 \cap UNIV \times (m2-inv8a-init-M2 \cap m2-inv8-M4 \cap m2-inv6-ikk-init)\}$   
 $(m1-step6 Ra A B Na Nb Kab), (m2-step6 Ra A B Na Nb Kab)$   
 $\{> R12\}$

*<proof>*

**lemma** *PO-m2-step7-refines-m1-step7:*

$\{R12 \cap UNIV \times (m2-inv9-M5 \cap m2-inv9a-resp-M3 \cap m2-inv7-ikk-resp)\}$   
 $(m1-step7 Rb A B Nb Kab), (m2-step7 Rb A B Nb Kab)$   
 $\{> R12\}$

*<proof>*

**lemma** *PO-m2-leak-refines-leak:*

$\{R12\}$   
 $m1-leak Rs Ra Rb A B, m2-leak Rs Ra Rb A B$   
 $\{> R12\}$

*<proof>*

**lemma** *PO-m2-fake-refines-skip:*

$\{R12\}$   
 $Id, m2-fake$   
 $\{> R12\}$

*<proof>*

Consequences of simulation relation and invariants.

**lemma** *m2-inv6-ikk-init-derived:*

**assumes**  $(s, t) \in R12$   $s \in m1-inv2i-serv$   $t \in m2-inv5-ikk-sv$   
**shows**  $t \in m2-inv6-ikk-init$

*<proof>*

**lemma** *m2-inv7-ikk-resp-derived:*

**assumes**  $(s, t) \in R12$   $s \in m1-inv2r-serv$   $t \in m2-inv5-ikk-sv$   
**shows**  $t \in m2-inv7-ikk-resp$

*<proof>*

All together now...

**lemmas** *PO-m2-trans-refines-m1-trans =*

*PO-m2-step1-refines-m1-step1 PO-m2-step2-refines-m1-step2*  
*PO-m2-step3-refines-m1-step3 PO-m2-step4-refines-m1-step4*  
*PO-m2-step5-refines-m1-step5 PO-m2-step6-refines-m1-step6*  
*PO-m2-step7-refines-m1-step7 PO-m2-leak-refines-leak*  
*PO-m2-fake-refines-skip*

**lemma** *PO-m2-refines-init-m1 [iff]:*

$init\ m2 \subseteq R12 \text{“}(init\ m1)$

*<proof>*

**lemma** *PO-m2-refines-trans-m1 [iff]:*

$\{R12 \cap$   
 $(reach\ m1 \times$

$(m2\text{-inv9-M5} \cap m2\text{-inv8a-init-M2} \cap m2\text{-inv9a-resp-M3} \cap m2\text{-inv8-M4} \cap$   
 $m2\text{-inv4-M3} \cap m2\text{-inv4-M2} \cap m2\text{-inv3a-sesK-compr} \cap m2\text{-inv3-extrKey} \cap m2\text{-inv2b-corrKey-leaked}))\}$   
 $(\text{trans } m1), (\text{trans } m2)$   
 $\{> R12\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-obs-consistent-R12* [iff]:  
 $\text{obs-consistent } R12 \text{ med21 } m1 \text{ } m2$   
 $\langle \text{proof} \rangle$

Refinement result.

**lemma** *m2-refines-m1* [iff]:  
 $\text{refines}$   
 $(R12 \cap$   
 $(\text{reach } m1 \times$   
 $(m2\text{-inv9-M5} \cap m2\text{-inv8a-init-M2} \cap m2\text{-inv9a-resp-M3} \cap m2\text{-inv8-M4} \cap$   
 $m2\text{-inv4-M3} \cap m2\text{-inv4-M2} \cap m2\text{-inv3a-sesK-compr} \cap m2\text{-inv3-extrKey} \cap$   
 $m2\text{-inv2b-corrKey-leaked} \cap m2\text{-inv2-keys-for} \cap m2\text{-inv1-keys}))$   
 $\text{med21 } m1 \text{ } m2$   
 $\langle \text{proof} \rangle$

**lemma** *m2-implements-m1* [iff]:  
 $\text{implements } \text{med21 } m1 \text{ } m2$   
 $\langle \text{proof} \rangle$

### 3.10.6 Inherited and derived invariants

Show preservation of invariants *m1-inv2i-serv* and *m1-inv2r-serv* from *m1*.

**lemma** *PO-m2-sat-m1-inv2i-serv* [iff]:  $\text{reach } m2 \subseteq m1\text{-inv2i-serv}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m2-sat-m1-inv2r-serv* [iff]:  $\text{reach } m2 \subseteq m1\text{-inv2r-serv}$   
 $\langle \text{proof} \rangle$

Now we derive the additional invariants for the initiator and the responder (see above for the definitions).

**lemma** *PO-m2-inv6-init-ikk* [iff]:  $\text{reach } m2 \subseteq m2\text{-inv6-ikk-init}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m2-inv6-resp-ikk* [iff]:  $\text{reach } m2 \subseteq m2\text{-inv7-ikk-resp}$   
 $\langle \text{proof} \rangle$

end

## 3.11 Needham-Schroeder Shared Key, "parallel" variant (L3)

**theory** *m3-nssk-par* **imports** *m2-nssk* *../Refinement/Message*

**begin**

We model an abstract version of the Needham-Schroeder Shared Key protocol:

- M1.  $A \rightarrow S : A, B, Na$
- M2.  $S \rightarrow A : \{Na, B, Kab, \{Kab, A\}_{Kbs}\}_{Kas}$
- M3.  $A \rightarrow B : \{Kab, A\}_{Kbs}$
- M4.  $B \rightarrow A : \{Nb\}_{Kab}$
- M5.  $A \rightarrow B : \{Nb - 1\}_{Kab}$

We model a "parallel" version of the NSSK protocol:

- M1.  $A \rightarrow S : A, B, Na$
- M2.  $S \rightarrow A : \{Na, B, Kab\}_{Kas}$
- M3.  $S \rightarrow B : \{Kab, A\}_{Kbs}$
- M4.  $B \rightarrow A : \{Nb\}_{Kab}$
- M5.  $A \rightarrow B : \{Nb - 1\}_{Kab}$

Proof tool configuration. Avoid annoying automatic unfolding of *dom*.

**declare** *domIff* [*simp*, *iff del*]

### 3.11.1 Setup

Now we can define the initial key knowledge.

**overloading** *ltkeySetup'*  $\equiv$  *ltkeySetup* **begin**

**definition** *ltkeySetup-def*: *ltkeySetup'*  $\equiv$   $\{(sharK\ C, A) \mid C\ A.\ A = C \vee A = Sv\}$   
**end**

**lemma** *corrKey-shrK-bad* [*simp*]: *corrKey* = *shrK'bad*  
(*proof*)

### 3.11.2 State

The secure channels are star-shaped to/from the server. Therefore, we have only one agent in the relation.

**record** *m3-state* = *m1-state* +  
*IK* :: *msg set* — intruder knowledge

Observable state: agent's local state.

**type-synonym**  
*m3-obs* = *m2-obs*

**definition**  
*m3-obs* :: *m3-state*  $\Rightarrow$  *m3-obs* **where**  
*m3-obs* *s*  $\equiv$   $(\mid runs = runs\ s, leak = leak\ s \mid)$

**type-synonym**  
*m3-pred* = *m3-state set*

**type-synonym**  
*m3-trans* = (*m3-state*  $\times$  *m3-state*) *set*

### 3.11.3 Events

Protocol events.

**definition** — by  $A$ , refines  $m2\text{-step}1$

$m3\text{-step}1 :: [rid\text{-}t, agent, agent, nonce] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step}1\ Ra\ A\ B\ Na \equiv \{(s, s1)\}.$

— guards:

$Ra \notin dom\ (runs\ s) \wedge$  —  $Ra$  is fresh  
 $Na = Ra\$na \wedge$  — generate nonce  $Na$

— actions:

$s1 = s\{$   
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [])),$   
 $IK := insert\ \{\{Agent\ A, Agent\ B, Nonce\ Na\}\}\ (IK\ s)$  — send msg 1  
 $\}$

**definition** — by  $B$ , refines  $m2\text{-step}2$

$m3\text{-step}2 :: [rid\text{-}t, agent, agent] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step}2\ Rb\ A\ B \equiv \{(s, s1)\}.$

— guards:

$Rb \notin dom\ (runs\ s) \wedge$  —  $Rb$  is fresh

— actions:

— create responder thread

$s1 = s\{$   
 $runs := (runs\ s)(Rb \mapsto (Resp, [A, B], []))$   
 $\}$

**definition** — by  $Server$ , refines  $m2\text{-step}3$

$m3\text{-step}3 :: [rid\text{-}t, agent, agent, nonce, key] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step}3\ Rs\ A\ B\ Na\ Kab \equiv \{(s, s1)\}.$

— guards:

$Rs \notin dom\ (runs\ s) \wedge$  — fresh server run  
 $Kab = sesK\ (Rs\$sk) \wedge$  — fresh session key

$\{\{Agent\ A, Agent\ B, Nonce\ Na\}\} \in IK\ s \wedge$  — recv msg 1

— actions:

— record session key and send messages 2 and 3

— note that last field in server record is for responder nonce

$s1 = s\{$   
 $runs := (runs\ s)(Rs \mapsto (Serv, [A, B], [aNon\ Na])),$   
 $IK := \{Crypt\ (shrK\ A)\ \{\{Nonce\ Na, Agent\ B, Key\ Kab\}\},$   
 $Crypt\ (shrK\ B)\ \{\{Key\ Kab, Agent\ A\}\}\} \cup IK\ s$   
 $\}$

}

**definition** — by  $A$ , refines  $m2\text{-step}4$

$m3\text{-step}4 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step}4 \text{ Ra } A \text{ B } Na \text{ Kab} \equiv \{(s, s1)\}.$

— guards:

$\text{runs } s \text{ Ra} = \text{Some } (\text{Init}, [A, B], []) \wedge$

$Na = \text{Ra}\$na \wedge$

$\text{Crypt } (\text{shr}K \ A) \ \{\!\! \{ \text{Nonce } Na, \text{Agent } B, \text{Key } Kab \}\!\! \} \in IK \ s \wedge$  — recv msg 2

— actions:

— record session key

$s1 = s\{$

$\text{runs} := (\text{runs } s)(\text{Ra} \mapsto (\text{Init}, [A, B], [\text{aKey } Kab]))$

$\}$

}

**definition** — by  $B$ , refines  $m2\text{-step}5$

$m3\text{-step}5 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step}5 \text{ Rb } A \text{ B } Nb \text{ Kab} \equiv \{(s, s1)\}.$

— guards:

$\text{runs } s \text{ Rb} = \text{Some } (\text{Resp}, [A, B], []) \wedge$

$Nb = \text{Rb}\$nb \wedge$

$\text{Crypt } (\text{shr}K \ B) \ \{\!\! \{ \text{Key } Kab, \text{Agent } A \}\!\! \} \in IK \ s \wedge$  — recv msg 3

— actions:

— record session key

$s1 = s\{$

$\text{runs} := (\text{runs } s)(\text{Rb} \mapsto (\text{Resp}, [A, B], [\text{aKey } Kab])),$

$IK := \text{insert } (\text{Crypt } Kab \ (\text{Nonce } Nb)) \ (IK \ s)$

$\}$

}

**definition** — by  $A$ , refines  $m2\text{-step}6$

$m3\text{-step}6 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{nonce}, \text{key}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step}6 \text{ Ra } A \text{ B } Na \text{ Nb } Kab \equiv \{(s, s')\}.$

$\text{runs } s \text{ Ra} = \text{Some } (\text{Init}, [A, B], [\text{aKey } Kab]) \wedge$  — key recv'd before

$Na = \text{Ra}\$na \wedge$

$\text{Crypt } Kab \ (\text{Nonce } Nb) \in IK \ s \wedge$  — receive  $M4$

— actions:

$s' = s\{$

$\text{runs} := (\text{runs } s)(\text{Ra} \mapsto (\text{Init}, [A, B], [\text{aKey } Kab, \text{aNon } Nb])),$

$IK := \text{insert } (\text{Crypt } Kab \ \{\!\! \{ \text{Nonce } Nb, \text{Nonce } Nb \}\!\! \}) \ (IK \ s)$

$\}$

}

**definition** — by  $B$ , refines  $m2\text{-step6}$

$m3\text{-step7} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step7 } Rb \ A \ B \ Nb \ Kab \equiv \{(s, s')\}.$

$\text{runs } s \ Rb = \text{Some } (\text{Resp}, [A, B], [aKey \ Kab]) \wedge$  — key recv'd before  
 $Nb = Rb\$nb \wedge$

$\text{Crypt } Kab \ \{\{Nonce \ Nb, Nonce \ Nb\} \in IK \ s \wedge$  — receive  $M5$

— actions: (redundant) update local state marks successful termination

$s' = s(|$   
 $\text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [aKey \ Kab, END]))$

$|)$

}

Session key compromise.

**definition** — refines  $m2\text{-leak}$

$m3\text{-leak} :: [\text{rid-}t, \text{rid-}t, \text{rid-}t, \text{agent}, \text{agent}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-leak } Rs \ Ra \ Rb \ A \ B \equiv \{(s, s1)\}.$

— guards:

$\text{runs } s \ Rs = \text{Some } (\text{Serv}, [A, B], [aNon \ (Ra\$na)]) \wedge$

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], [aKey \ (\text{sesK } (Rs\$sk)), aNon \ (Rb\$nb)]) \wedge$

$\text{runs } s \ Rb = \text{Some } (\text{Resp}, [A, B], [aKey \ (\text{sesK } (Rs\$sk)), END]) \wedge$

— actions:

— record session key as leaked and add it to intruder knowledge

$s1 = s(| \text{leak} := \text{insert } (\text{sesK } (Rs\$sk), Ra\$na, Rb\$nb) (\text{leak } s),$   
 $IK := \text{insert } (\text{Key } (\text{sesK } (Rs\$sk))) (IK \ s) |)$

}

Intruder fake event.

**definition** — refines  $m2\text{-fake}$

$m3\text{-DY-fake} :: m3\text{-trans}$

**where**

$m3\text{-DY-fake} \equiv \{(s, s1)\}.$

— actions:

$s1 = s(|$   
 $IK := \text{synth } (\text{analz } (IK \ s))$

$|)$

}

### 3.11.4 Transition system

**definition**

$m3\text{-init} :: m3\text{-state set}$

**where**

$m3\text{-init} \equiv \{(|$

$\text{runs} = \text{Map.empty},$

$\text{leak} = \text{shrK'bad} \times \{\text{undefined}\} \times \{\text{undefined}\},$

```

    IK = Key'shrK'bad
  }

```

**definition**

```

m3-trans :: (m3-state × m3-state) set where
m3-trans ≡ (⋃ Ra Rb Rs A B Na Nb Kab.
  m3-step1 Ra A B Na ∪
  m3-step2 Rb A B ∪
  m3-step3 Rs A B Na Kab ∪
  m3-step4 Ra A B Na Kab ∪
  m3-step5 Rb A B Nb Kab ∪
  m3-step6 Ra A B Na Nb Kab ∪
  m3-step7 Rb A B Nb Kab ∪
  m3-leak Rs Ra Rb A B ∪
  m3-DY-fake ∪
  Id
)

```

**definition**

```

m3 :: (m3-state, m3-obs) spec where
m3 ≡ (
  init = m3-init,
  trans = m3-trans,
  obs = m3-obs
)

```

**lemmas** *m3-defs* =

```

m3-def m3-init-def m3-trans-def m3-obs-def
m3-step1-def m3-step2-def m3-step3-def m3-step4-def m3-step5-def
m3-step6-def m3-step7-def m3-leak-def m3-DY-fake-def

```

### 3.11.5 Invariants

Specialized injection that we can apply more aggressively.

**lemmas** *analz-Inj-IK* = *analz.Inj* [**where**  $H=IK$  *s* **for** *s*]

**lemmas** *parts-Inj-IK* = *parts.Inj* [**where**  $H=IK$  *s* **for** *s*]

**declare** *parts-Inj-IK* [*dest!*]

**declare** *analz-into-parts* [*dest*]

#### inv1: Secrecy of pre-distributed shared keys

inv1: Secrecy of long-term keys

**definition**

```

m3-inv1-lkeysec :: m3-state set
where
m3-inv1-lkeysec ≡ {s. ∀ C.
  (Key (shrK C) ∈ parts (IK s) → C ∈ bad) ∧
  (C ∈ bad → Key (shrK C) ∈ IK s)
}

```

**lemmas**  $m3\text{-inv1-lkeysecI} = m3\text{-inv1-lkeysec-def}$  [THEN setc-def-to-intro, rule-format]  
**lemmas**  $m3\text{-inv1-lkeysecE}$  [elim] =  $m3\text{-inv1-lkeysec-def}$  [THEN setc-def-to-elim, rule-format]  
**lemmas**  $m3\text{-inv1-lkeysecD} = m3\text{-inv1-lkeysec-def}$  [THEN setc-def-to-dest, rule-format]

Invariance proof.

**lemma**  $PO\text{-}m3\text{-inv1-lkeysec-init}$  [iff]:  
 $init\ m3 \subseteq m3\text{-inv1-lkeysec}$   
 ⟨proof⟩

**lemma**  $PO\text{-}m3\text{-inv1-lkeysec-trans}$  [iff]:  
 $\{m3\text{-inv1-lkeysec}\ trans\ m3 \{>\ m3\text{-inv1-lkeysec}\}$   
 ⟨proof⟩

**lemma**  $PO\text{-}m3\text{-inv1-lkeysec}$  [iff]:  $reach\ m3 \subseteq m3\text{-inv1-lkeysec}$   
 ⟨proof⟩

Useful simplifier lemmas

**lemma**  $m3\text{-inv1-lkeysec-for-parts}$  [simp]:  
 $\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies Key\ (shrK\ C) \in parts\ (IK\ s) \longleftrightarrow C \in bad$   
 ⟨proof⟩

**lemma**  $m3\text{-inv1-lkeysec-for-analz}$  [simp]:  
 $\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies Key\ (shrK\ C) \in analz\ (IK\ s) \longleftrightarrow C \in bad$   
 ⟨proof⟩

### inv7a: Session keys not used to encrypt other session keys

Session keys are not used to encrypt other keys. Proof requires generalization to sets of session keys.

NOTE: This invariant will be derived from the corresponding L2 invariant using the simulation relation.

#### definition

$m3\text{-inv7a-sesK-compr} :: m3\text{-pred}$

#### where

$m3\text{-inv7a-sesK-compr} \equiv \{s. \forall K\ KK.$   
 $KK \subseteq range\ sesK \longrightarrow$   
 $(Key\ K \in analz\ (Key\ KK \cup (IK\ s))) = (K \in KK \vee Key\ K \in analz\ (IK\ s))$   
 $\}$

**lemmas**  $m3\text{-inv7a-sesK-comprI} = m3\text{-inv7a-sesK-compr-def}$  [THEN setc-def-to-intro, rule-format]  
**lemmas**  $m3\text{-inv7a-sesK-comprE} = m3\text{-inv7a-sesK-compr-def}$  [THEN setc-def-to-elim, rule-format]  
**lemmas**  $m3\text{-inv7a-sesK-comprD} = m3\text{-inv7a-sesK-compr-def}$  [THEN setc-def-to-dest, rule-format]

Additional lemma

**lemmas**  $insert-commute-Key = insert-commute$  [where  $x=Key\ K$  for  $K$ ]

**lemmas**  $m3\text{-inv7a-sesK-compr-simps} =$   
 $m3\text{-inv7a-sesK-comprD}$   
 $m3\text{-inv7a-sesK-comprD}$  [where  $KK=\{Kab\}$  for  $Kab$ , simplified]  
 $m3\text{-inv7a-sesK-comprD}$  [where  $KK=insert\ Kab\ KK$  for  $Kab\ KK$ , simplified]  
 $insert-commute-Key$

### 3.11.6 Refinement

#### Message abstraction and simulation relation

Abstraction function on sets of messages.

**inductive-set**

$abs\text{-}msg :: msg\ set \Rightarrow chmsg\ set$

**for**  $H :: msg\ set$

**where**

*am-M1:*

$\{Agent\ A, Agent\ B, Nonce\ Na\} \in H$

$\Rightarrow Insec\ A\ B\ (Msg\ [aNon\ Na]) \in abs\text{-}msg\ H$

| *am-M2:*

$Crypt\ (shrK\ C)\ \{Nonce\ N, Agent\ B, Key\ K\} \in H$

$\Rightarrow Secure\ Sv\ C\ (Msg\ [aNon\ N, aAgt\ B, aKey\ K]) \in abs\text{-}msg\ H$

| *am-M3:*

$Crypt\ (shrK\ C)\ \{Key\ K, Agent\ A\} \in H$

$\Rightarrow Secure\ Sv\ C\ (Msg\ [aKey\ K, aAgt\ A]) \in abs\text{-}msg\ H$

| *am-M4:*

$Crypt\ K\ (Nonce\ N) \in H$

$\Rightarrow dAuth\ K\ (Msg\ [aNon\ N]) \in abs\text{-}msg\ H$

| *am-M5:*

$Crypt\ K\ \{Nonce\ N, Nonce\ N'\} \in H$

$\Rightarrow dAuth\ K\ (Msg\ [aNon\ N, aNon\ N']) \in abs\text{-}msg\ H$

R23: The simulation relation. This is a data refinement of the insecure and secure channels of refinement 2.

**definition**

$R23\text{-}msgs :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$

$R23\text{-}msgs \equiv \{(s, t). abs\text{-}msg\ (parts\ (IK\ t)) \subseteq chan\ s\}$

**definition**

$R23\text{-}keys :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$  — equivalence!

$R23\text{-}keys \equiv \{(s, t). \forall KK\ K. KK \subseteq range\ sesK \longrightarrow$

$Key\ K \in analz\ (Key'KK \cup IK\ t) \longleftrightarrow aKey\ K \in extr\ (aKey'KK \cup ik0)\ (chan\ s)$

$\}$

**definition**

$R23\text{-}non :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$  — only an implication!

$R23\text{-}non \equiv \{(s, t). \forall KK\ N. KK \subseteq range\ sesK \longrightarrow$

$Nonce\ N \in analz\ (Key'KK \cup IK\ t) \longrightarrow aNon\ N \in extr\ (aKey'KK \cup ik0)\ (chan\ s)$

$\}$

**definition**

$R23\text{-}pres :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$

$R23\text{-}pres \equiv \{(s, t). runs\ s = runs\ t \wedge leak\ s = leak\ t\}$

**definition**

$R23 :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$

$R23 \equiv R23\text{-}msgs \cap R23\text{-}keys \cap R23\text{-}non \cap R23\text{-}pres$

**lemmas**  $R23\text{-}defs =$

$R23\text{-}def\ R23\text{-}msgs\text{-}def\ R23\text{-}keys\text{-}def\ R23\text{-}non\text{-}def\ R23\text{-}pres\text{-}def$

The mediator function is the identity here.

**definition**

$med32 :: m3\text{-obs} \Rightarrow m2\text{-obs}$  **where**  
 $med32 \equiv id$

**lemmas**  $R23\text{-msgsI} = R23\text{-msgs-def}$  [THEN rel-def-to-intro, simplified, rule-format]  
**lemmas**  $R23\text{-msgsE}$  [elim] =  $R23\text{-msgs-def}$  [THEN rel-def-to-elim, simplified, rule-format]

**lemmas**  $R23\text{-keysI} = R23\text{-keys-def}$  [THEN rel-def-to-intro, simplified, rule-format]  
**lemmas**  $R23\text{-keysE}$  [elim] =  $R23\text{-keys-def}$  [THEN rel-def-to-elim, simplified, rule-format]  
**lemmas**  $R23\text{-keysD} = R23\text{-keys-def}$  [THEN rel-def-to-dest, simplified, rule-format]

**lemmas**  $R23\text{-nonI} = R23\text{-non-def}$  [THEN rel-def-to-intro, simplified, rule-format]  
**lemmas**  $R23\text{-nonE}$  [elim] =  $R23\text{-non-def}$  [THEN rel-def-to-elim, simplified, rule-format]  
**lemmas**  $R23\text{-nonD} = R23\text{-non-def}$  [THEN rel-def-to-dest, simplified, rule-format, rotated 2]

**lemmas**  $R23\text{-presI} = R23\text{-pres-def}$  [THEN rel-def-to-intro, simplified, rule-format]  
**lemmas**  $R23\text{-presE}$  [elim] =  $R23\text{-pres-def}$  [THEN rel-def-to-elim, simplified, rule-format]

**lemmas**  $R23\text{-intros} = R23\text{-msgsI}$   $R23\text{-keysI}$   $R23\text{-nonI}$   $R23\text{-presI}$

Further lemmas: general lemma for simplifier and different instantiations.

**lemmas**  $R23\text{-keys-simps} =$   
 $R23\text{-keysD}$   
 $R23\text{-keysD}$  [where  $KK = \{\}$ , simplified]  
 $R23\text{-keysD}$  [where  $KK = \{K'\}$  for  $K'$ , simplified]  
 $R23\text{-keysD}$  [where  $KK = \text{insert } K' \text{ } KK$  for  $K' \text{ } KK$ , simplified, OF - conjI]

**lemmas**  $R23\text{-non-dests} =$   
 $R23\text{-nonD}$   
 $R23\text{-nonD}$  [where  $KK = \{\}$ , simplified]  
 $R23\text{-nonD}$  [where  $KK = \{K\}$  for  $K$ , simplified]  
 $R23\text{-nonD}$  [where  $KK = \text{insert } K \text{ } KK$  for  $K \text{ } KK$ , simplified, OF - - conjI]

**General lemmas**

General facts about *abs-msg*

**declare**  $abs\text{-msg.intros}$  [intro!]  
**declare**  $abs\text{-msg.cases}$  [elim!]

**lemma**  $abs\text{-msg-empty}$ :  $abs\text{-msg } \{\} = \{\}$   
<proof>

**lemma**  $abs\text{-msg-Un}$  [simp]:  
 $abs\text{-msg } (G \cup H) = abs\text{-msg } G \cup abs\text{-msg } H$   
<proof>

**lemma**  $abs\text{-msg-mono}$  [elim]:  
 $\llbracket m \in abs\text{-msg } G; G \subseteq H \rrbracket \Longrightarrow m \in abs\text{-msg } H$   
<proof>

**lemma** *abs-msg-insert-mono* [intro]:  
 $\llbracket m \in \text{abs-msg } H \rrbracket \implies m \in \text{abs-msg } (\text{insert } m' H)$   
 <proof>

Facts about *abs-msg* concerning abstraction of fakeable messages. This is crucial for proving the refinement of the intruder event.

**lemma** *abs-msg-DY-subset-fakeable*:  
 $\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; (s, t) \in R23\text{-non}; t \in m3\text{-inv1-lkeysec} \rrbracket$   
 $\implies \text{abs-msg } (\text{synth } (\text{analz } (IK\ t))) \subseteq \text{fake ik0 } (\text{dom } (\text{runs } s)) (\text{chan } s)$   
 <proof>

## Refinement proof

Pair decomposition. These were set to `elim!`, which is too aggressive here.

**declare** *MPair-analz* [rule del, elim]  
**declare** *MPair-parts* [rule del, elim]

Protocol events.

**lemma** *PO-m3-step1-refines-m2-step1*:  
 $\{R23\}$   
 $(m2\text{-step1 } Ra\ A\ B\ Na), (m3\text{-step1 } Ra\ A\ B\ Na)$   
 $\{> R23\}$   
 <proof>

**lemma** *PO-m3-step2-refines-m2-step2*:  
 $\{R23\}$   
 $(m2\text{-step2 } Rb\ A\ B), (m3\text{-step2 } Rb\ A\ B)$   
 $\{> R23\}$   
 <proof>

**lemma** *PO-m3-step3-refines-m2-step3*:  
 $\{R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv7a-sesK-compr} \cap m3\text{-inv1-lkeysec})\}$   
 $(m2\text{-step3 } Rs\ A\ B\ Na\ Kab), (m3\text{-step3 } Rs\ A\ B\ Na\ Kab)$   
 $\{> R23\}$   
 <proof>

**lemma** *PO-m3-step4-refines-m2-step4*:  
 $\{R23\}$   
 $(m2\text{-step4 } Ra\ A\ B\ Na\ Kab), (m3\text{-step4 } Ra\ A\ B\ Na\ Kab)$   
 $\{> R23\}$   
 <proof>

**lemma** *PO-m3-step5-refines-m2-step5*:  
 $\{R23\}$   
 $(m2\text{-step5 } Rb\ A\ B\ Nb\ Kab), (m3\text{-step5 } Rb\ A\ B\ Nb\ Kab)$   
 $\{> R23\}$   
 <proof>

**lemma** *PO-m3-step6-refines-m2-step6*:

$\{R23\}$   
 $(m2\text{-step6 } Ra \ A \ B \ Na \ Nb \ Kab), (m3\text{-step6 } Ra \ A \ B \ Na \ Nb \ Kab)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-step7-refines-m2-step7*:

$\{R23\}$   
 $(m2\text{-step7 } Rb \ A \ B \ Nb \ Kab), (m3\text{-step7 } Rb \ A \ B \ Nb \ Kab)$   
 $\{> R23\}$   
 $\langle proof \rangle$

Intruder events.

**lemma** *PO-m3-leak-refines-m2-leak*:

$\{R23\}$   
 $(m2\text{-leak } Rs \ Ra \ Rb \ A \ B), (m3\text{-leak } Rs \ Ra \ Rb \ A \ B)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-DY-fake-refines-m2-fake*:

$\{R23 \cap UNIV \times m3\text{-inv1-lkeysec}\}$   
 $m2\text{-fake}, m3\text{-DY-fake}$   
 $\{> R23\}$   
 $\langle proof \rangle$

All together now...

**lemmas** *PO-m3-trans-refines-m2-trans* =

$PO\text{-}m3\text{-step1-refines-m2-step1}$   $PO\text{-}m3\text{-step2-refines-m2-step2}$   
 $PO\text{-}m3\text{-step3-refines-m2-step3}$   $PO\text{-}m3\text{-step4-refines-m2-step4}$   
 $PO\text{-}m3\text{-step5-refines-m2-step5}$   $PO\text{-}m3\text{-step6-refines-m2-step6}$   
 $PO\text{-}m3\text{-step7-refines-m2-step7}$   $PO\text{-}m3\text{-leak-refines-m2-leak}$   
 $PO\text{-}m3\text{-DY-fake-refines-m2-fake}$

**lemma** *PO-m3-refines-init-m2* [iff]:

$init \ m3 \subseteq R23''(init \ m2)$   
 $\langle proof \rangle$

**lemma** *PO-m3-refines-trans-m2* [iff]:

$\{R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv7a-sesK-compr} \cap m3\text{-inv1-lkeysec})\}$   
 $(trans \ m2), (trans \ m3)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-observation-consistent* [iff]:

$obs\text{-consistent } R23 \ med32 \ m2 \ m3$   
 $\langle proof \rangle$

Refinement result.

**lemma** *m3-refines-m2* [iff]:

$refines (R23 \cap m2\text{-inv3a-sesK-compr} \times m3\text{-inv1-lkeysec})$   
 $med32 \ m2 \ m3$   
 $\langle proof \rangle$

**lemma** *m3-implements-m2* [iff]:  
*implements med32 m2 m3*  
 ⟨proof⟩

### 3.11.7 Inherited invariants

#### inv4 (derived): Key secrecy for initiator

**definition**

*m3-inv4-ikk-init* :: *m3-state set*

**where**

*m3-inv4-ikk-init* ≡ {*s*. ∀ *Ra K A B al*.  
*runs s Ra = Some (Init, [A, B], aKey K # al) → A ∈ good → B ∈ good →*  
*Key K ∈ analz (IK s) →*  
*(∃ Nb'. (K, Ra \$ na, Nb') ∈ leak s)*  
 }

**lemmas** *m3-inv4-ikk-initI* = *m3-inv4-ikk-init-def* [THEN *setc-def-to-intro*, *rule-format*]

**lemmas** *m3-inv4-ikk-initE* [elim] = *m3-inv4-ikk-init-def* [THEN *setc-def-to-elim*, *rule-format*]

**lemmas** *m3-inv4-ikk-initD* = *m3-inv4-ikk-init-def* [THEN *setc-def-to-dest*, *rule-format*, *rotated 1*]

**lemma** *PO-m3-inv4-ikk-init*: *reach m3 ⊆ m3-inv4-ikk-init*  
 ⟨proof⟩

#### inv5 (derived): Key secrecy for responder

**definition**

*m3-inv5-ikk-resp* :: *m3-state set*

**where**

*m3-inv5-ikk-resp* ≡ {*s*. ∀ *Rb K A B al*.  
*runs s Rb = Some (Resp, [A, B], aKey K # al) → A ∈ good → B ∈ good →*  
*Key K ∈ analz (IK s) →*  
*K ∈ Domain (leak s)*  
 }

**lemmas** *m3-inv5-ikk-respI* = *m3-inv5-ikk-resp-def* [THEN *setc-def-to-intro*, *rule-format*]

**lemmas** *m3-inv5-ikk-respE* [elim] = *m3-inv5-ikk-resp-def* [THEN *setc-def-to-elim*, *rule-format*]

**lemmas** *m3-inv5-ikk-respD* = *m3-inv5-ikk-resp-def* [THEN *setc-def-to-dest*, *rule-format*, *rotated 1*]

**lemma** *PO-m3-inv4-ikk-resp*: *reach m3 ⊆ m3-inv5-ikk-resp*  
 ⟨proof⟩

end

## 3.12 Needham-Schroeder Shared Key (L3)

**theory** *m3-nssk* imports *m2-nssk* ../Refinement/Message  
 begin

We model an abstract version of the Needham-Schroeder Shared Key protocol:

- M1.  $A \rightarrow S : A, B, Na$
- M2.  $S \rightarrow A : \{Na, B, Kab, \{Kab, A\}_{Kbs}\}_{Kas}$
- M3.  $A \rightarrow B : \{Kab, A\}_{Kbs}$
- M4.  $B \rightarrow A : \{Nb\}_{Kab}$
- M5.  $A \rightarrow B : \{Nb - 1\}_{Kab}$

This refinement works with a single insecure channel and introduces the full Dolev-Yao intruder.

Proof tool configuration. Avoid annoying automatic unfolding of *dom*.

**declare** *domIff* [*simp*, *iff del*]

### 3.12.1 Setup

Now we can define the initial key knowledge.

**overloading** *ltkeySetup'*  $\equiv$  *ltkeySetup* **begin**

**definition** *ltkeySetup-def*: *ltkeySetup'*  $\equiv \{(shrK\ C, A) \mid C\ A.\ A = C \vee A = Sv\}$   
**end**

**lemma** *corrKey-shrK-bad* [*simp*]: *corrKey* = *shrK'bad*  
*<proof>*

### 3.12.2 State

The secure channels are star-shaped to/from the server. Therefore, we have only one agent in the relation.

**record** *m3-state* = *m1-state* +  
*IK* :: *msg set* — intruder knowledge

Observable state: agent's local state.

**type-synonym**  
*m3-obs* = *m2-obs*

**definition**  
*m3-obs* :: *m3-state*  $\Rightarrow$  *m3-obs* **where**  
*m3-obs* *s*  $\equiv$  (*runs* = *runs s*, *leak* = *leak s*)

**type-synonym**  
*m3-pred* = *m3-state set*

**type-synonym**  
*m3-trans* = (*m3-state*  $\times$  *m3-state*) *set*

### 3.12.3 Events

Protocol events.

**definition** — by *A*, refines *m2-step1*  
*m3-step1* :: [*rid-t*, *agent*, *agent*, *nonce*]  $\Rightarrow$  *m3-trans*

**where**

$m3\text{-step1 } Ra \ A \ B \ Na \equiv \{(s, s1)\}.$

— guards:

$Ra \notin \text{dom } (runs \ s) \wedge$

—  $Ra$  is fresh

$Na = Ra\$na \wedge$

— generate nonce  $Na$

— actions:

$s1 = s\{$

$runs := (runs \ s)(Ra \mapsto (Init, [A, B], [])),$

$IK := insert \ \{\!|Agent \ A, \ Agent \ B, \ Nonce \ Na|\!\} \ (IK \ s) \quad \text{— send msg 1}$

$\}$

**definition** — by  $B$ , refines  $m2\text{-step2}$

$m3\text{-step2} :: [rid\text{-}t, \ agent, \ agent] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step2 } Rb \ A \ B \equiv \{(s, s1)\}.$

— guards:

$Rb \notin \text{dom } (runs \ s) \wedge$

—  $Rb$  is fresh

— actions:

— create responder thread

$s1 = s\{$

$runs := (runs \ s)(Rb \mapsto (Resp, [A, B], []))$

$\}$

**definition** — by  $Server$ , refines  $m2\text{-step3}$

$m3\text{-step3} :: [rid\text{-}t, \ agent, \ agent, \ nonce, \ key] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step3 } Rs \ A \ B \ Na \ Kab \equiv \{(s, s1)\}.$

— guards:

$Rs \notin \text{dom } (runs \ s) \wedge$

— fresh server run

$Kab = sesK \ (Rs\$sk) \wedge$

— fresh session key

$\{\!|Agent \ A, \ Agent \ B, \ Nonce \ Na|\!\} \in IK \ s \wedge \quad \text{— recv msg 1}$

— actions:

— record session key and send messages 2 and 3

— note that last field in server record is for responder nonce

$s1 = s\{$

$runs := (runs \ s)(Rs \mapsto (Serv, [A, B], [aNon \ Na])),$

$IK := insert$

$(Crypt \ (shrK \ A)$

$\ \{\!|Nonce \ Na, \ Agent \ B, \ Key \ Kab,$

$\ \ Crypt \ (shrK \ B) \ \{\!|Key \ Kab, \ Agent \ A|\!\}\}$

$(IK \ s)$

$\}$

**definition** — by  $A$ , refines  $m2\text{-step4}$

$m3\text{-step4} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}, \text{msg}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step4 } Ra \ A \ B \ Na \ Kab \ X \equiv \{(s, s1)\}.$

— guards:

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], []) \wedge$   
 $Na = Ra\$na \wedge$

$\text{Crypt } (\text{shr}K \ A) \ \{\!\! \{ \text{Nonce } Na, \text{Agent } B, \text{Key } Kab, X \}\!\! \} \in IK \ s \wedge$  — recv msg 2

— actions:

— record session key, and forward  $X$

$s1 = s \{$   
 $\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [aKey \ Kab])),$   
 $IK := \text{insert } X \ (IK \ s)$

$\}$

**definition** — by  $B$ , refines  $m2\text{-step5}$

$m3\text{-step5} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step5 } Rb \ A \ B \ Nb \ Kab \equiv \{(s, s1)\}.$

— guards:

$\text{runs } s \ Rb = \text{Some } (\text{Resp}, [A, B], []) \wedge$   
 $Nb = Rb\$nb \wedge$

$\text{Crypt } (\text{shr}K \ B) \ \{\!\! \{ \text{Key } Kab, \text{Agent } A \}\!\! \} \in IK \ s \wedge$  — recv msg 3

— actions:

— record session key

$s1 = s \{$   
 $\text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [aKey \ Kab])),$   
 $IK := \text{insert } (\text{Crypt } Kab \ (\text{Nonce } Nb)) \ (IK \ s)$

$\}$

**definition** — by  $A$ , refines  $m2\text{-step6}$

$m3\text{-step6} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{nonce}, \text{key}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step6 } Ra \ A \ B \ Na \ Nb \ Kab \equiv \{(s, s')\}.$

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], [aKey \ Kab]) \wedge$  — key recv'd before  
 $Na = Ra\$na \wedge$

$\text{Crypt } Kab \ (\text{Nonce } Nb) \in IK \ s \wedge$  — receive  $M4$

— actions:

$s' = s \{$   
 $\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [aKey \ Kab, aNon \ Nb])),$   
 $IK := \text{insert } (\text{Crypt } Kab \ \{\!\! \{ \text{Nonce } Nb, \text{Nonce } Nb \}\!\! \}) \ (IK \ s)$

$\}$

**definition** — by  $B$ , refines  $m2\text{-step6}$

$m3\text{-step7} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step7 } Rb \ A \ B \ Nb \ Kab \equiv \{(s, s')\}.$

$\text{runs } s \ Rb = \text{Some } (\text{Resp}, [A, B], [aKey \ Kab]) \wedge$  — key recv'd before  
 $Nb = Rb\$nb \wedge$

$\text{Crypt } Kab \ \{\text{Nonce } Nb, \text{Nonce } Nb\} \in IK \ s \wedge$  — receive  $M5$

— actions: (redundant) update local state marks successful termination

$s' = s(|$   
 $\text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [aKey \ Kab, \text{END}])))$   
 $|)$   
 $\}$

Session key compromise.

**definition** — refines  $m2\text{-leak}$

$m3\text{-leak} :: [\text{rid-}t, \text{rid-}t, \text{rid-}t, \text{agent}, \text{agent}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-leak } Rs \ Ra \ Rb \ A \ B \equiv \{(s, s1)\}.$

— guards:

$\text{runs } s \ Rs = \text{Some } (\text{Serv}, [A, B], [aNon \ (Ra\$na)]) \wedge$   
 $\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], [aKey \ (\text{sesK } (Rs\$sk)), aNon \ (Rb\$nb)]) \wedge$   
 $\text{runs } s \ Rb = \text{Some } (\text{Resp}, [A, B], [aKey \ (\text{sesK } (Rs\$sk)), \text{END}]) \wedge$

— actions:

— record session key as leaked and add it to intruder knowledge

$s1 = s(|$   $\text{leak} := \text{insert } (\text{sesK } (Rs\$sk), Ra\$na, Rb\$nb) \ (\text{leak } s),$   
 $IK := \text{insert } (\text{Key } (\text{sesK } (Rs\$sk))) \ (IK \ s) \ |)$   
 $\}$

Intruder fake event.

**definition** — refines  $m2\text{-fake}$

$m3\text{-DY-fake} :: m3\text{-trans}$

**where**

$m3\text{-DY-fake} \equiv \{(s, s1)\}.$

— actions:

$s1 = s(|$   
 $IK := \text{synth } (\text{analz } (IK \ s))$   
 $|)$   
 $\}$

### 3.12.4 Transition system

**definition**

$m3\text{-init} :: m3\text{-state set}$

**where**

$m3\text{-init} \equiv \{(|$   
 $\text{runs} = \text{Map.empty},$   
 $\text{leak} = \text{shrK}'\text{bad} \times \{\text{undefined}\} \times \{\text{undefined}\},$   
 $IK = \text{Key}'\text{shrK}'\text{bad}$   
 $|)\}$

**definition**

```

m3-trans :: (m3-state × m3-state) set where
m3-trans ≡ (⋃ Ra Rb Rs A B Na Nb Kab X .
  m3-step1 Ra A B Na ∪
  m3-step2 Rb A B ∪
  m3-step3 Rs A B Na Kab ∪
  m3-step4 Ra A B Na Kab X ∪
  m3-step5 Rb A B Nb Kab ∪
  m3-step6 Ra A B Na Nb Kab ∪
  m3-step7 Rb A B Nb Kab ∪
  m3-leak Rs Ra Rb A B ∪
  m3-DY-fake ∪
  Id
)

```

**definition**

```

m3 :: (m3-state, m3-obs) spec where
m3 ≡ (
  init = m3-init,
  trans = m3-trans,
  obs = m3-obs
)

```

**lemmas** *m3-defs* =

```

m3-def m3-init-def m3-trans-def m3-obs-def
m3-step1-def m3-step2-def m3-step3-def m3-step4-def m3-step5-def
m3-step6-def m3-step7-def m3-leak-def m3-DY-fake-def

```

**3.12.5 Invariants**

Specialized injection that we can apply more aggressively.

```

lemmas analz-Inj-IK = analz.Inj [where H=IK s for s]
lemmas parts-Inj-IK = parts.Inj [where H=IK s for s]

```

```

declare parts-Inj-IK [dest!]

```

```

declare analz-into-parts [dest]

```

**inv1: Secrecy of pre-distributed shared keys**

inv1: Secrecy of long-term keys

**definition**

```

m3-inv1-lkeysec :: m3-state set
where
m3-inv1-lkeysec ≡ {s. ∀ C.
  (Key (shrK C) ∈ parts (IK s) → C ∈ bad) ∧
  (C ∈ bad → Key (shrK C) ∈ IK s)
}

```

```

lemmas m3-inv1-lkeysecI = m3-inv1-lkeysec-def [THEN setc-def-to-intro, rule-format]

```

```

lemmas m3-inv1-lkeysecE [elim] = m3-inv1-lkeysec-def [THEN setc-def-to-elim, rule-format]

```

**lemmas**  $m3\text{-inv1-lkeysec}D = m3\text{-inv1-lkeysec-def}$  [THEN setc-def-to-dest, rule-format]

Invariance proof.

**lemma**  $PO\text{-}m3\text{-inv1-lkeysec-init}$  [iff]:

$init\ m3 \subseteq m3\text{-inv1-lkeysec}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m3\text{-inv1-lkeysec-trans}$  [iff]:

$\{m3\text{-inv1-lkeysec}\ trans\ m3 \{>\ m3\text{-inv1-lkeysec}\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m3\text{-inv1-lkeysec}$  [iff]:  $reach\ m3 \subseteq m3\text{-inv1-lkeysec}$

$\langle proof \rangle$

Useful simplifier lemmas

**lemma**  $m3\text{-inv1-lkeysec-for-parts}$  [simp]:

$\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies Key\ (shrK\ C) \in parts\ (IK\ s) \longleftrightarrow C \in bad$

$\langle proof \rangle$

**lemma**  $m3\text{-inv1-lkeysec-for-analz}$  [simp]:

$\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies Key\ (shrK\ C) \in analz\ (IK\ s) \longleftrightarrow C \in bad$

$\langle proof \rangle$

## inv2: Ticket shape for honestly encrypted M2

**definition**

$m3\text{-inv2-ticket} :: m3\text{-state set}$

**where**

$m3\text{-inv2-ticket} \equiv \{s. \forall A\ B\ N\ K\ X.$

$A \notin bad \longrightarrow$

$Crypt\ (shrK\ A)\ \{\!\! \{ Nonce\ N, Agent\ B, Key\ K, X \}\!\!\} \in parts\ (IK\ s) \longrightarrow$

$X = Crypt\ (shrK\ B)\ \{\!\! \{ Key\ K, Agent\ A \}\!\!\} \wedge K \in range\ sesK$

$\}$

**lemmas**  $m3\text{-inv2-ticket}I =$

$m3\text{-inv2-ticket-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m3\text{-inv2-ticket}E$  [elim] =

$m3\text{-inv2-ticket-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m3\text{-inv2-ticket}D =$

$m3\text{-inv2-ticket-def}$  [THEN setc-def-to-dest, rule-format, rotated -1]

Invariance proof.

**lemma**  $PO\text{-}m3\text{-inv2-ticket-init}$  [iff]:

$init\ m3 \subseteq m3\text{-inv2-ticket}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m3\text{-inv2-ticket-trans}$  [iff]:

$\{m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec}\ trans\ m3 \{>\ m3\text{-inv2-ticket}\}$

$\langle proof \rangle$

**lemma**  $PO\text{-}m3\text{-inv2-ticket}$  [iff]:  $reach\ m3 \subseteq m3\text{-inv2-ticket}$

$\langle proof \rangle$

### inv3: Session keys not used to encrypt other session keys

Session keys are not used to encrypt other keys. Proof requires generalization to sets of session keys.

NOTE: For NSSK, this invariant cannot be inherited from the corresponding L2 invariant. The simulation relation is only an implication not an equivalence.

#### definition

$m3\text{-inv3-sesK-compr} :: m3\text{-state set}$

#### where

$m3\text{-inv3-sesK-compr} \equiv \{s. \forall K KK.$

$KK \subseteq \text{range sesK} \longrightarrow$

$(\text{Key } K \in \text{analz } (\text{Key } KK \cup (IK \ s))) = (K \in KK \vee \text{Key } K \in \text{analz } (IK \ s))$

$\}$

**lemmas**  $m3\text{-inv3-sesK-comprI} = m3\text{-inv3-sesK-compr-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m3\text{-inv3-sesK-comprE} = m3\text{-inv3-sesK-compr-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m3\text{-inv3-sesK-comprD} = m3\text{-inv3-sesK-compr-def}$  [THEN setc-def-to-dest, rule-format]

Additional lemma

**lemmas**  $\text{insert-commute-Key} = \text{insert-commute}$  [where  $x = \text{Key } K$  for  $K$ ]

**lemmas**  $m3\text{-inv3-sesK-compr-simps} =$

$m3\text{-inv3-sesK-comprD}$

$m3\text{-inv3-sesK-comprD}$  [where  $KK = \{Kab\}$  for  $Kab$ , simplified]

$m3\text{-inv3-sesK-comprD}$  [where  $KK = \text{insert } Kab \ KK$  for  $Kab \ KK$ , simplified]

$\text{insert-commute-Key}$

Invariance proof.

**lemma**  $PO\text{-}m3\text{-inv3-sesK-compr-step4}$ :

$\{m3\text{-inv3-sesK-compr} \cap m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec}\}$

$m3\text{-step4 } Ra \ A \ B \ Na \ Kab \ X$

$\{> m3\text{-inv3-sesK-compr}\}$

$\langle \text{proof} \rangle$

All together now.

**lemmas**  $PO\text{-}m3\text{-inv3-sesK-compr-trans-lemmas} =$

$PO\text{-}m3\text{-inv3-sesK-compr-step4}$

**lemma**  $PO\text{-}m3\text{-inv3-sesK-compr-init}$  [iff]:

$\text{init } m3 \subseteq m3\text{-inv3-sesK-compr}$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m3\text{-inv3-sesK-compr-trans}$  [iff]:

$\{m3\text{-inv3-sesK-compr} \cap m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec}\}$

$\text{trans } m3$

$\{> m3\text{-inv3-sesK-compr}\}$

$\langle \text{proof} \rangle$

**lemma**  $PO\text{-}m3\text{-inv3-sesK-compr}$  [iff]:  $\text{reach } m3 \subseteq m3\text{-inv3-sesK-compr}$

$\langle \text{proof} \rangle$

### 3.12.6 Refinement

#### Message abstraction and simulation relation

Abstraction function on sets of messages.

**inductive-set**

$abs\text{-}msg :: msg\ set \Rightarrow chmsg\ set$

**for**  $H :: msg\ set$

**where**

$am\text{-}M1:$

$\{Agent\ A, Agent\ B, Nonce\ Na\} \in H$

$\Rightarrow Insec\ A\ B\ (Msg\ [aNon\ Na]) \in abs\text{-}msg\ H$

|  $am\text{-}M2:$

$Crypt\ (shrK\ C)\ \{Nonce\ N, Agent\ B, Key\ K, X\} \in H$

$\Rightarrow Secure\ Sv\ C\ (Msg\ [aNon\ N, aAgt\ B, aKey\ K]) \in abs\text{-}msg\ H$

|  $am\text{-}M3:$

$Crypt\ (shrK\ C)\ \{Key\ K, Agent\ A\} \in H$

$\Rightarrow Secure\ Sv\ C\ (Msg\ [aKey\ K, aAgt\ A]) \in abs\text{-}msg\ H$

|  $am\text{-}M4:$

$Crypt\ K\ (Nonce\ N) \in H$

$\Rightarrow dAuth\ K\ (Msg\ [aNon\ N]) \in abs\text{-}msg\ H$

|  $am\text{-}M5:$

$Crypt\ K\ \{Nonce\ N, Nonce\ N'\} \in H$

$\Rightarrow dAuth\ K\ (Msg\ [aNon\ N, aNon\ N']) \in abs\text{-}msg\ H$

R23: The simulation relation. This is a data refinement of the insecure and secure channels of refinement 2.

**definition**

$R23\text{-}msgs :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$

$R23\text{-}msgs \equiv \{(s, t). abs\text{-}msg\ (parts\ (IK\ t)) \subseteq chan\ s\}$

**definition**

$R23\text{-}keys :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$  — only an implication!

$R23\text{-}keys \equiv \{(s, t). \forall KK\ K. KK \subseteq range\ sesK \longrightarrow$

$Key\ K \in analz\ (Key'KK \cup IK\ t) \longrightarrow aKey\ K \in extr\ (aKey'KK \cup ik0)\ (chan\ s)$

$\}$

**definition**

$R23\text{-}non :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$  — only an implication!

$R23\text{-}non \equiv \{(s, t). \forall KK\ N. KK \subseteq range\ sesK \longrightarrow$

$Nonce\ N \in analz\ (Key'KK \cup IK\ t) \longrightarrow aNon\ N \in extr\ (aKey'KK \cup ik0)\ (chan\ s)$

$\}$

**definition**

$R23\text{-}pres :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$

$R23\text{-}pres \equiv \{(s, t). runs\ s = runs\ t \wedge leak\ s = leak\ t\}$

**definition**

$R23 :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$

$R23 \equiv R23\text{-}msgs \cap R23\text{-}keys \cap R23\text{-}non \cap R23\text{-}pres$

**lemmas**  $R23\text{-}defs =$

$R23\text{-}def\ R23\text{-}msgs\text{-}def\ R23\text{-}keys\text{-}def\ R23\text{-}non\text{-}def\ R23\text{-}pres\text{-}def$

The mediator function is the identity here.

**definition**

$med32 :: m3\text{-obs} \Rightarrow m2\text{-obs}$  **where**  
 $med32 \equiv id$

**lemmas**  $R23\text{-msgsI} = R23\text{-msgs-def}$  [*THEN rel-def-to-intro, simplified, rule-format*]  
**lemmas**  $R23\text{-msgsE}$  [*elim*] =  $R23\text{-msgs-def}$  [*THEN rel-def-to-elim, simplified, rule-format*]  
**lemmas**  $R23\text{-msgsE}'$  [*elim*] =  
 $R23\text{-msgs-def}$  [*THEN rel-def-to-dest, simplified, rule-format, THEN subsetD*]

**lemmas**  $R23\text{-keysI} = R23\text{-keys-def}$  [*THEN rel-def-to-intro, simplified, rule-format*]  
**lemmas**  $R23\text{-keysE}$  [*elim*] =  $R23\text{-keys-def}$  [*THEN rel-def-to-elim, simplified, rule-format*]  
**lemmas**  $R23\text{-keysD} = R23\text{-keys-def}$  [*THEN rel-def-to-dest, simplified, rule-format, rotated 2*]

**lemmas**  $R23\text{-nonI} = R23\text{-non-def}$  [*THEN rel-def-to-intro, simplified, rule-format*]  
**lemmas**  $R23\text{-nonE}$  [*elim*] =  $R23\text{-non-def}$  [*THEN rel-def-to-elim, simplified, rule-format*]  
**lemmas**  $R23\text{-nonD} = R23\text{-non-def}$  [*THEN rel-def-to-dest, simplified, rule-format, rotated 2*]

**lemmas**  $R23\text{-presI} = R23\text{-pres-def}$  [*THEN rel-def-to-intro, simplified, rule-format*]  
**lemmas**  $R23\text{-presE}$  [*elim*] =  $R23\text{-pres-def}$  [*THEN rel-def-to-elim, simplified, rule-format*]

**lemmas**  $R23\text{-intros} = R23\text{-msgsI}$   $R23\text{-keysI}$   $R23\text{-nonI}$   $R23\text{-presI}$

Further lemmas: general lemma for simplifier and different instantiations.

**lemmas**  $R23\text{-keys-dests} =$   
 $R23\text{-keysD}$   
 $R23\text{-keysD}$  [**where**  $KK = \{\}$ , *simplified*]  
 $R23\text{-keysD}$  [**where**  $KK = \{K\}$  **for**  $K$ , *simplified*]  
 $R23\text{-keysD}$  [**where**  $KK = \text{insert } K \text{ } KK$  **for**  $K \text{ } KK$ , *simplified, OF - - conjI*]

**lemmas**  $R23\text{-non-dests} =$   
 $R23\text{-nonD}$   
 $R23\text{-nonD}$  [**where**  $KK = \{\}$ , *simplified*]  
 $R23\text{-nonD}$  [**where**  $KK = \{K\}$  **for**  $K$ , *simplified*]  
 $R23\text{-nonD}$  [**where**  $KK = \text{insert } K \text{ } KK$  **for**  $K \text{ } KK$ , *simplified, OF - - conjI*]

**lemmas**  $R23\text{-dests} = R23\text{-keys-dests}$   $R23\text{-non-dests}$

**General lemmas**

General facts about *abs-msg*

**declare**  $abs\text{-msg.intros}$  [*intro!*]  
**declare**  $abs\text{-msg.cases}$  [*elim!*]

**lemma**  $abs\text{-msg.empty}$ :  $abs\text{-msg} \ \{\} = \{\}$   
 $\langle proof \rangle$

**lemma**  $abs\text{-msg.Un}$  [*simp*]:  
 $abs\text{-msg} \ (G \cup H) = abs\text{-msg} \ G \cup abs\text{-msg} \ H$   
 $\langle proof \rangle$

**lemma** *abs-msg-mono* [*elim*]:  
 $\llbracket m \in \text{abs-msg } G; G \subseteq H \rrbracket \Longrightarrow m \in \text{abs-msg } H$   
 $\langle \text{proof} \rangle$

**lemma** *abs-msg-insert-mono* [*intro*]:  
 $\llbracket m \in \text{abs-msg } H \rrbracket \Longrightarrow m \in \text{abs-msg } (\text{insert } m' H)$   
 $\langle \text{proof} \rangle$

Facts about *abs-msg* concerning abstraction of fakeable messages. This is crucial for proving the refinement of the intruder event.

**lemma** *abs-msg-DY-subset-fakeable*:  
 $\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; (s, t) \in R23\text{-non}; t \in m3\text{-inv1-lkeysec} \rrbracket$   
 $\Longrightarrow \text{abs-msg } (\text{synth } (\text{analz } (IK t))) \subseteq \text{fake ik0 } (\text{dom } (\text{runs } s)) (\text{chan } s)$   
 $\langle \text{proof} \rangle$

## Refinement proof

Pair decomposition. These were set to **elim!**, which is too aggressive here.

**declare** *MPair-analz* [*rule del, elim*]  
**declare** *MPair-parts* [*rule del, elim*]

Protocol events.

**lemma** *PO-m3-step1-refines-m2-step1*:  
 $\{R23\}$   
 $(m2\text{-step1 } Ra A B Na), (m3\text{-step1 } Ra A B Na)$   
 $\{> R23\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m3-step2-refines-m2-step2*:  
 $\{R23\}$   
 $(m2\text{-step2 } Rb A B), (m3\text{-step2 } Rb A B)$   
 $\{> R23\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m3-step3-refines-m2-step3*:  
 $\{R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv3-sesK-compr} \cap m3\text{-inv1-lkeysec})\}$   
 $(m2\text{-step3 } Rs A B Na Kab), (m3\text{-step3 } Rs A B Na Kab)$   
 $\{> R23\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m3-step4-refines-m2-step4*:  
 $\{R23 \cap (m2\text{-inv3b-sesK-compr-non})$   
 $\times (m3\text{-inv3-sesK-compr} \cap m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec})\}$   
 $(m2\text{-step4 } Ra A B Na Kab), (m3\text{-step4 } Ra A B Na Kab X)$   
 $\{> R23\}$   
 $\langle \text{proof} \rangle$

**lemma** *PO-m3-step5-refines-m2-step5*:  
 $\{R23\}$

$(m2\text{-step5 } Rb \ A \ B \ Nb \ Kab), (m3\text{-step5 } Rb \ A \ B \ Nb \ Kab)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-step6-refines-m2-step6*:  
 $\{R23\}$   
 $(m2\text{-step6 } Ra \ A \ B \ Na \ Nb \ Kab), (m3\text{-step6 } Ra \ A \ B \ Na \ Nb \ Kab)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-step7-refines-m2-step7*:  
 $\{R23\}$   
 $(m2\text{-step7 } Rb \ A \ B \ Nb \ Kab), (m3\text{-step7 } Rb \ A \ B \ Nb \ Kab)$   
 $\{> R23\}$   
 $\langle proof \rangle$

Intruder events.

**lemma** *PO-m3-leak-refines-m2-leak*:  
 $\{R23\}$   
 $m2\text{-leak } Rs \ Ra \ Rb \ A \ B, m3\text{-leak } Rs \ Ra \ Rb \ A \ B$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-DY-fake-refines-m2-fake*:  
 $\{R23 \cap UNIV \times m3\text{-inv1-lkeysec}\}$   
 $m2\text{-fake}, m3\text{-DY-fake}$   
 $\{> R23\}$   
 $\langle proof \rangle$

All together now...

**lemmas** *PO-m3-trans-refines-m2-trans* =  
 $PO\text{-m3-step1-refines-m2-step1 } PO\text{-m3-step2-refines-m2-step2}$   
 $PO\text{-m3-step3-refines-m2-step3 } PO\text{-m3-step4-refines-m2-step4}$   
 $PO\text{-m3-step5-refines-m2-step5 } PO\text{-m3-step6-refines-m2-step6}$   
 $PO\text{-m3-step7-refines-m2-step7 } PO\text{-m3-leak-refines-m2-leak}$   
 $PO\text{-m3-DY-fake-refines-m2-fake}$

**lemma** *PO-m3-refines-init-m2* [iff]:  
 $init \ m3 \subseteq R23''(init \ m2)$   
 $\langle proof \rangle$

**lemma** *PO-m3-refines-trans-m2* [iff]:  
 $\{R23 \cap (m2\text{-inv3a-sesK-compr} \cap m2\text{-inv3b-sesK-compr-non})$   
 $\times (m3\text{-inv3-sesK-compr} \cap m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec})\}$   
 $(trans \ m2), (trans \ m3)$   
 $\{> R23\}$   
 $\langle proof \rangle$

**lemma** *PO-m3-observation-consistent* [iff]:  
 $obs\text{-consistent } R23 \ med32 \ m2 \ m3$   
 $\langle proof \rangle$

Refinement result.

**lemma** *m3-refines-m2* [iff]:

*refines*

$(R23 \cap (m2-inv3a-sesK-compr \cap m2-inv3b-sesK-compr-non)$   
 $\times (m3-inv3-sesK-compr \cap m3-inv2-ticket \cap m3-inv1-lkeysec))$   
*med32 m2 m3*

*<proof>*

**lemma** *m3-implements-m2* [iff]:

*implements med32 m2 m3*

*<proof>*

### 3.12.7 Inherited invariants

**inv4 (derived): Key secrecy for initiator**

**definition**

*m3-inv4-ikk-init* :: *m3-state set*

**where**

*m3-inv4-ikk-init*  $\equiv \{s. \forall Ra\ K\ A\ B\ al.$   
*runs*  $s\ Ra = Some\ (Init,\ [A,\ B],\ aKey\ K\ \# al) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$   
*Key*  $K \in analz\ (IK\ s) \longrightarrow$   
 $(\exists Nb'. (K,\ Ra\ \$\ na,\ Nb') \in leak\ s)$   
 $\}$

**lemmas** *m3-inv4-ikk-initI* = *m3-inv4-ikk-init-def* [THEN *setc-def-to-intro*, *rule-format*]

**lemmas** *m3-inv4-ikk-initE* [elim] = *m3-inv4-ikk-init-def* [THEN *setc-def-to-elim*, *rule-format*]

**lemmas** *m3-inv4-ikk-initD* = *m3-inv4-ikk-init-def* [THEN *setc-def-to-dest*, *rule-format*, *rotated 1*]

**lemma** *PO-m3-inv4-ikk-init*: *reach m3*  $\subseteq$  *m3-inv4-ikk-init*

*<proof>*

**inv5 (derived): Key secrecy for responder**

**definition**

*m3-inv5-ikk-resp* :: *m3-state set*

**where**

*m3-inv5-ikk-resp*  $\equiv \{s. \forall Rb\ K\ A\ B\ al.$   
*runs*  $s\ Rb = Some\ (Resp,\ [A,\ B],\ aKey\ K\ \# al) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$   
*Key*  $K \in analz\ (IK\ s) \longrightarrow$   
 $K \in Domain\ (leak\ s)$   
 $\}$

**lemmas** *m3-inv5-ikk-respI* = *m3-inv5-ikk-resp-def* [THEN *setc-def-to-intro*, *rule-format*]

**lemmas** *m3-inv5-ikk-respE* [elim] = *m3-inv5-ikk-resp-def* [THEN *setc-def-to-elim*, *rule-format*]

**lemmas** *m3-inv5-ikk-respD* = *m3-inv5-ikk-resp-def* [THEN *setc-def-to-dest*, *rule-format*, *rotated 1*]

**lemma** *PO-m3-inv4-ikk-resp*: *reach m3*  $\subseteq$  *m3-inv5-ikk-resp*

*<proof>*

**end**

### 3.13 Abstract Denning-Sacco protocol (L1)

**theory** *m1-ds* **imports** *m1-keydist-inrn ../Refinement/a0n-agree*  
**begin**

We augment the basic abstract key distribution model such that the server sends a timestamp along with the session key. We check the timestamp's validity to ensure recentness of the session key.

We establish one refinement for this model, namely that this model refines the basic authenticated key transport model *m1-keydist-inrn*, which guarantees non-injective agreement with the server on the session key and the server-generated timestamp.

#### 3.13.1 State

We extend the basic key distribution by adding timestamps. We add a clock variable modeling the current time. The frames, runs, and observations remain the same as in the previous model, but we will use the *nat list*'s to store timestamps.

**type-synonym**  
*time* = *nat* — for clock and timestamps

**consts**  
*Ls* :: *time* — life time for session keys

State and observations

**record**  
*m1-state* = *m1x-state* +  
*clk* :: *time*

**type-synonym**  
*m1-obs* = *m1-state*

**type-synonym**  
*'x m1-pred* = *'x m1-state-scheme set*

**type-synonym**  
*'x m1-trans* = (*'x m1-state-scheme* × *'x m1-state-scheme*) *set*

Instantiate parameters regarding list of freshness identifiers stored at server.

**overloading** *is-len'* ≡ *is-len* *rs-len'* ≡ *rs-len* **begin**

**definition** *is-len-def* [*simp*]: *is-len'* ≡ *1::nat*

**definition** *rs-len-def* [*simp*]: *rs-len'* ≡ *1::nat*

**end**

#### 3.13.2 Events

**definition** — by *A*, refines *m1x-step1*  
*m1-step1* :: [*rid-t*, *agent*, *agent*] ⇒ *'x m1-trans*

**where**

*m1-step1* ≡ *m1a-step1*

**definition** — by *B*, refines *m1x-step2*

$m1\text{-step2} :: [\text{rid-}t, \text{agent}, \text{agent}] \Rightarrow 'x \text{ m1-trans}$

**where**

$m1\text{-step2} \equiv m1a\text{-step2}$

**definition** — by  $Sv$ , refines  $m1x\text{-step3}$

$m1\text{-step3} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}, \text{time}] \Rightarrow 'x \text{ m1-trans}$

**where**

$m1\text{-step3 } Rs \ A \ B \ Kab \ Ts \equiv \{(s, s')\}.$

— new guards:

$Ts = \text{clk } s \wedge$  — fresh timestamp

— rest as before:

$(s, s') \in m1a\text{-step3 } Rs \ A \ B \ Kab \ [aNum \ Ts]$

}

**definition** — by  $A$ , refines  $m1x\text{-step5}$

$m1\text{-step4} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}, \text{time}] \Rightarrow 'x \text{ m1-trans}$

**where**

$m1\text{-step4 } Ra \ A \ B \ Kab \ Ts \equiv \{(s, s')\}.$

— new guards:

$\text{clk } s < Ts + Ls \wedge$  — ensure session key recentness

— rest as before

$(s, s') \in m1a\text{-step4 } Ra \ A \ B \ Kab \ [aNum \ Ts]$

}

**definition** — by  $B$ , refines  $m1x\text{-step4}$

$m1\text{-step5} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}, \text{time}] \Rightarrow 'x \text{ m1-trans}$

**where**

$m1\text{-step5 } Rb \ A \ B \ Kab \ Ts \equiv \{(s, s')\}.$

— new guards:

— ensure freshness of session key

$\text{clk } s < Ts + Ls \wedge$

— rest as before

$(s, s') \in m1a\text{-step5 } Rb \ A \ B \ Kab \ [aNum \ Ts]$

}

**definition** — refines  $skip$

$m1\text{-tick} :: \text{time} \Rightarrow 'x \text{ m1-trans}$

**where**

$m1\text{-tick } T \equiv \{(s, s')\}.$

$s' = s(\ \text{clk} := \text{clk } s + T \ )$

}

**definition** — by attacker, refines  $m1x\text{-leak}$

$m1\text{-leak} :: [\text{rid-}t] \Rightarrow 'x \text{ m1-trans}$

**where**

$m1\text{-leak} \equiv m1a\text{-leak}$

### 3.13.3 Specification

**definition**

$m1-init :: unit\ m1-pred$

**where**

$m1-init \equiv \{ \langle \text{runs} = \text{Map.empty}, \text{leak} = \text{corrKey}, \text{clk} = 0 \rangle \}$

**definition**

$m1-trans :: 'x\ m1-trans\ \mathbf{where}$

$m1-trans \equiv (\bigcup A\ B\ Ra\ Rb\ Rs\ Kab\ Ts\ T.$

$\ m1-step1\ Ra\ A\ B \cup$

$\ m1-step2\ Rb\ A\ B \cup$

$\ m1-step3\ Rs\ A\ B\ Kab\ Ts \cup$

$\ m1-step4\ Ra\ A\ B\ Kab\ Ts \cup$

$\ m1-step5\ Rb\ A\ B\ Kab\ Ts \cup$

$\ m1-tick\ T \cup$

$\ m1-leak\ Rs \cup$

$\ Id$

$\rangle$

**definition**

$m1 :: (m1-state, m1-obs)\ \text{spec}\ \mathbf{where}$

$m1 \equiv \langle$

$\ \text{init} = m1-init,$

$\ \text{trans} = m1-trans,$

$\ \text{obs} = id$

$\rangle$

**lemmas**  $m1-loc-defs =$

$m1-def\ m1-init-def\ m1-trans-def$

$m1-step1-def\ m1-step2-def\ m1-step3-def\ m1-step4-def\ m1-step5-def$

$m1-leak-def\ m1-tick-def$

**lemmas**  $m1-defs = m1-loc-defs\ m1a-defs$

**lemma**  $m1-obs-id$  [simp]:  $obs\ m1 = id$

$\langle proof \rangle$

### 3.13.4 Invariants

#### inv0: Finite domain

There are only finitely many runs. This is needed to establish the responder/initiator agreement.

**definition**

$m1-inv0-fin :: 'x\ m1-pred$

**where**

$m1-inv0-fin \equiv \{s.\ \text{finite}\ (\text{dom}\ (\text{runs}\ s))\}$

**lemmas**  $m1-inv0-finI = m1-inv0-fin-def$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m1-inv0-finE$  [elim] =  $m1-inv0-fin-def$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m1-inv0-finD = m1-inv0-fin-def$  [THEN setc-def-to-dest, rule-format]

Invariance proofs.

**lemma**  $PO-m1-inv0-fin-init$  [iff]:

$init\ m1 \subseteq m1\text{-inv0-fin}$   
 $\langle proof \rangle$

**lemma** *PO-m1-inv0-fin-trans* [iff]:  
 $\{m1\text{-inv0-fin}\} trans\ m1 \{>\ m1\text{-inv0-fin}\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-inv0-fin* [iff]:  $reach\ m1 \subseteq m1\text{-inv0-fin}$   
 $\langle proof \rangle$

### 3.13.5 Refinement of $m1a$

#### Simulation relation

R1a1: The simulation relation and mediator function are identities.

**definition**  
 $med1a1 :: m1\text{-obs} \Rightarrow m1a\text{-obs}$  **where**  
 $med1a1\ t \equiv (\mid runs = runs\ t, leak = leak\ t \mid)$

**definition**  
 $R1a1 :: (m1a\text{-state} \times m1\text{-state})\ set$  **where**  
 $R1a1 \equiv \{(s, t). s = med1a1\ t\}$

**lemmas**  $R1a1\text{-defs} = R1a1\text{-def}\ med1a1\text{-def}$

#### Refinement proof

**lemma** *PO-m1-step1-refines-m1a-step1*:  
 $\{R1a1\}$   
 $(m1a\text{-step1}\ Ra\ A\ B), (m1\text{-step1}\ Ra\ A\ B)$   
 $\{>\ R1a1\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step2-refines-m1a-step2*:  
 $\{R1a1\}$   
 $(m1a\text{-step2}\ Rb\ A\ B), (m1\text{-step2}\ Rb\ A\ B)$   
 $\{>\ R1a1\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step3-refines-m1a-step3*:  
 $\{R1a1\}$   
 $(m1a\text{-step3}\ Rs\ A\ B\ Kab\ [aNum\ Ts]), (m1\text{-step3}\ Rs\ A\ B\ Kab\ Ts)$   
 $\{>\ R1a1\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step4-refines-m1a-step4*:  
 $\{R1a1\}$   
 $(m1a\text{-step4}\ Ra\ A\ B\ Kab\ [aNum\ Ts]), (m1\text{-step4}\ Ra\ A\ B\ Kab\ Ts)$   
 $\{>\ R1a1\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-step5-refines-m1a-step5*:  
 $\{R1a1\}$

$(m1a\text{-step5 } Rb \ A \ B \ Kab \ [aNum \ Ts]), (m1\text{-step5 } Rb \ A \ B \ Kab \ Ts)$   
 $\{> R1a1\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-leak-refines-m1a-leak*:  
 $\{R1a1\}$   
 $(m1a\text{-leak } Rs), (m1\text{-leak } Rs)$   
 $\{> R1a1\}$   
 $\langle proof \rangle$

**lemma** *PO-m1-tick-refines-m1a-skip*:  
 $\{R1a1\}$   
 $Id, (m1\text{-tick } T)$   
 $\{> R1a1\}$   
 $\langle proof \rangle$

All together now...

**lemmas** *PO-m1-trans-refines-m1a-trans =*  
 $PO\text{-m1-step1-refines-m1a-step1 } PO\text{-m1-step2-refines-m1a-step2}$   
 $PO\text{-m1-step3-refines-m1a-step3 } PO\text{-m1-step4-refines-m1a-step4}$   
 $PO\text{-m1-step5-refines-m1a-step5 } PO\text{-m1-leak-refines-m1a-leak}$   
 $PO\text{-m1-tick-refines-m1a-skip}$

**lemma** *PO-m1-refines-init-m1a [iff]*:  
 $init \ m1 \subseteq R1a1 \text{“}(init \ m1a)$   
 $\langle proof \rangle$

**lemma** *PO-m1-refines-trans-m1a [iff]*:  
 $\{R1a1\}$   
 $(trans \ m1a), (trans \ m1)$   
 $\{> R1a1\}$   
 $\langle proof \rangle$

Observation consistency.

**lemma** *obs-consistent-med1a1 [iff]*:  
 $obs\text{-consistent } R1a1 \ med1a1 \ m1a \ m1$   
 $\langle proof \rangle$

Refinement result.

**lemma** *PO-m1-refines-m1a [iff]*:  
 $refines \ R1a1 \ med1a1 \ m1a \ m1$   
 $\langle proof \rangle$

**lemma** *m1-implements-m1: implements med1a1 m1a m1*  
 $\langle proof \rangle$

end

### 3.14 Abstract Denning-Sacco protocol (L2)

**theory** *m2-ds imports m1-ds ../Refinement/Channels*

**begin**

We model an abstract version of the Denning-Sacco protocol:

- M1.  $A \rightarrow S : A, B$
- M2.  $S \rightarrow A : \{B, Kab, T, \{Kab, A, T\}_{Kbs}\}_{Kas}$
- M3.  $A \rightarrow B : \{Kab, A, T\}_{Kbs}$

This refinement introduces channels with security properties. We model a parallel version of the DS protocol:

- M1.  $A \rightarrow S : A, B$
- M2a.  $S \rightarrow A : \{B, Kab, T\}_{Kas}$
- M2b.  $S \rightarrow B : \{Kab, A, T\}_{Kbs}$

Message 1 is sent over an insecure channel, the other two message over secure channels.

**declare** *domIff* [*simp, iff del*]

### 3.14.1 State

State and observations

**record** *m2-state* = *m1-state* +  
*chan* :: *chmsg set* — channel messages

**type-synonym**

*m2-obs* = *m1-state*

**definition**

*m2-obs* :: *m2-state*  $\Rightarrow$  *m2-obs* **where**

*m2-obs* *s*  $\equiv$  (  
  *runs* = *runs s*,  
  *leak* = *leak s*,  
  *clk* = *clk s*  
)

**type-synonym**

*m2-pred* = *m2-state set*

**type-synonym**

*m2-trans* = (*m2-state*  $\times$  *m2-state*) *set*

### 3.14.2 Events

Protocol events.

**definition** — by *A*, refines *m1a-step1*

*m2-step1* :: [*rid-t, agent, agent*]  $\Rightarrow$  *m2-trans*

**where**

*m2-step1 Ra A B*  $\equiv$   $\{(s, s1)\}$ .

— guards:

*Ra*  $\notin$  *dom (runs s)*  $\wedge$

— *Ra* is fresh



$m2\text{-step5} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}, \text{time}] \Rightarrow m2\text{-trans}$

**where**

$m2\text{-step5 } Rb \ A \ B \ Kab \ Ts \equiv \{(s, s1)\}.$

— guards:

$runs \ s \ Rb = Some \ (Resp, [A, B], []) \wedge$

$Secure \ Sv \ B \ (Msg \ [aKey \ Kab, aAgt \ A, aNum \ Ts]) \in \text{chan } s \wedge \text{— recv } M2b$

— ensure freshness of session key

$clk \ s < Ts + Ls \wedge$

— actions:

— record session key

$s1 = s\{$

$runs := (runs \ s)(Rb \mapsto (Resp, [A, B], [aKey \ Kab, aNum \ Ts]))$

$\}$

Clock tick event

**definition** — refines  $m1\text{-tick}$

$m2\text{-tick} :: \text{time} \Rightarrow m2\text{-trans}$

**where**

$m2\text{-tick} \equiv m1\text{-tick}$

Session key compromise.

**definition** — refines  $m1\text{-leak}$

$m2\text{-leak} :: \text{rid-}t \Rightarrow m2\text{-trans}$

**where**

$m2\text{-leak } Rs \equiv \{(s, s1)\}.$

— guards:

$Rs \in \text{dom} \ (runs \ s) \wedge$

$\text{fst} \ (the \ (runs \ s \ Rs)) = Serv \wedge \text{— compromise server run } Rs$

— actions:

— record session key as leaked;

— intruder sends himself an insecure channel message containing the key

$s1 = s\{ \text{leak} := \text{insert} \ (sesK \ (Rs\$sk)) \ (\text{leak } s),$

$\text{chan} := \text{insert} \ (\text{Insec} \ \text{undefined} \ \text{undefined} \ (Msg \ [aKey \ (sesK \ (Rs\$sk))])) \ (\text{chan } s) \}$

$\}$

Intruder fake event (new).

**definition** — refines  $Id$

$m2\text{-fake} :: m2\text{-trans}$

**where**

$m2\text{-fake} \equiv \{(s, s1)\}.$

— actions:

$s1 = s\{$

— close under fakeable messages

$\text{chan} := \text{fake} \ ik0 \ (\text{dom} \ (runs \ s)) \ (\text{chan } s)$

$\}$

$\}$

### 3.14.3 Transition system

#### definition

$m2-init :: m2-pred$

#### where

$m2-init \equiv \{ \langle \langle$   
 $runs = Map.empty,$   
 $leak = corrKey,$   
 $clk = 0,$   
 $chan = \{ \} \quad \text{--- Channels.ik0 contains aKey'corrKey}$   
 $\rangle \rangle \}$

#### definition

$m2-trans :: m2-trans$  **where**

$m2-trans \equiv (\bigcup A B Ra Rb Rs Kab Ts T.$   
 $m2-step1 Ra A B \cup$   
 $m2-step2 Rb A B \cup$   
 $m2-step3 Rs A B Kab Ts \cup$   
 $m2-step4 Ra A B Kab Ts \cup$   
 $m2-step5 Rb A B Kab Ts \cup$   
 $m2-tick T \cup$   
 $m2-leak Rs \cup$   
 $m2-fake \cup$   
 $Id$   
 $)$

#### definition

$m2 :: (m2-state, m2-obs) spec$  **where**

$m2 \equiv \langle \langle$   
 $init = m2-init,$   
 $trans = m2-trans,$   
 $obs = m2-obs$   
 $\rangle \rangle$

**lemmas**  $m2-loc-defs =$

$m2-def$   $m2-init-def$   $m2-trans-def$   $m2-obs-def$   
 $m2-step1-def$   $m2-step2-def$   $m2-step3-def$   $m2-step4-def$   $m2-step5-def$   
 $m2-tick-def$   $m2-leak-def$   $m2-fake-def$

**lemmas**  $m2-defs = m2-loc-defs m1-defs$

### 3.14.4 Invariants and simulation relation

#### inv3a: Session key compromise

A L2 version of a session key compromise invariant. Roughly, it states that adding a set of keys  $KK$  to the parameter  $T$  of  $extr$  does not help the intruder to extract keys other than those in  $KK$  or extractable without adding  $KK$ .

#### definition

$m2-inv3a-sesK-compr :: m2-state set$

#### where

$m2-inv3a-sesK-compr \equiv \{s. \forall K KK.$   
 $aKey K \in extr (aKey'KK \cup ik0) (chan s) \longleftrightarrow (K \in KK \vee aKey K \in extr ik0 (chan s))$

}

**lemmas**  $m2\text{-inv3a-sesK-comprI} =$   
 $m2\text{-inv3a-sesK-compr-def}$  [THEN setc-def-to-intro, rule-format]  
**lemmas**  $m2\text{-inv3a-sesK-comprE}$  [elim] =  
 $m2\text{-inv3a-sesK-compr-def}$  [THEN setc-def-to-elim, rule-format]  
**lemmas**  $m2\text{-inv3a-sesK-comprD} =$   
 $m2\text{-inv3a-sesK-compr-def}$  [THEN setc-def-to-dest, rule-format]

Additional lemma

**lemmas**  $insert\text{-commute-aKey} = insert\text{-commute}$  [where  $x=aKey$   $K$  for  $K$ ]

**lemmas**  $m2\text{-inv3a-sesK-compr-simps} =$   
 $m2\text{-inv3a-sesK-comprD}$   
 $m2\text{-inv3a-sesK-comprD}$  [where  $KK=insert$   $Kab$   $KK$  for  $Kab$   $KK$ , simplified]  
 $m2\text{-inv3a-sesK-comprD}$  [where  $KK=\{Kab\}$  for  $Kab$ , simplified]  
 $insert\text{-commute-aKey}$  — to get the keys to the front

**lemma**  $PO\text{-}m2\text{-inv3a-sesK-compr-init}$  [iff]:  
 $init\ m2 \subseteq m2\text{-inv3a-sesK-compr}$   
 ⟨proof⟩

**lemma**  $PO\text{-}m2\text{-inv3a-sesK-compr-trans}$  [iff]:  
 $\{m2\text{-inv3a-sesK-compr}\ trans\ m2 \{>\ m2\text{-inv3a-sesK-compr}\}$   
 ⟨proof⟩

**lemma**  $PO\text{-}m2\text{-inv3a-sesK-compr}$  [iff]:  $reach\ m2 \subseteq m2\text{-inv3a-sesK-compr}$   
 ⟨proof⟩

### inv3: Extracted session keys

inv3: Extracted non-leaked session keys were generated by the server for at least one bad agent. This invariant is needed in the proof of the strengthening of the authorization guards in steps 4 and 5 (e.g.,  $(Kab, A) \in azC$  ( $runs\ s$ ) for the initiator's step4).

#### definition

$m2\text{-inv3-extrKey} :: m2\text{-state set}$

where

$m2\text{-inv3-extrKey} \equiv \{s. \forall K.$   
 $aKey\ K \in extr\ ik0\ (chan\ s) \longrightarrow K \notin leak\ s \longrightarrow \text{was: } K \notin corrKey \longrightarrow$   
 $(\exists R\ A'\ B'\ Ts'. K = sesK\ (R\$sk) \wedge$   
 $runs\ s\ R = Some\ (Serv, [A', B'], [aNum\ Ts']) \wedge$   
 $(A' \in bad \vee B' \in bad))$   
 $\}$

**lemmas**  $m2\text{-inv3-extrKeyI} =$   
 $m2\text{-inv3-extrKey-def}$  [THEN setc-def-to-intro, rule-format]  
**lemmas**  $m2\text{-inv3-extrKeyE}$  [elim] =  
 $m2\text{-inv3-extrKey-def}$  [THEN setc-def-to-elim, rule-format]  
**lemmas**  $m2\text{-inv3-extrKeyD} =$   
 $m2\text{-inv3-extrKey-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

**lemma**  $PO\text{-}m2\text{-inv3-extrKey-init}$  [iff]:

$init\ m2 \subseteq m2\text{-inv3}\text{-extrKey}$   
 ⟨proof⟩

**lemma**  $PO\text{-}m2\text{-inv3}\text{-extrKey}\text{-trans}$  [iff]:  
 $\{m2\text{-inv3}\text{-extrKey} \cap m2\text{-inv3a}\text{-sesK}\text{-compr}\}$   
 $trans\ m2$   
 $\{>\ m2\text{-inv3}\text{-extrKey}\}$   
 ⟨proof⟩

**lemma**  $PO\text{-}m2\text{-inv3}\text{-extrKey}$  [iff]:  $reach\ m2 \subseteq m2\text{-inv3}\text{-extrKey}$   
 ⟨proof⟩

#### inv4: Messages M2a/M2b for good agents and server state

inv4: Secure messages to honest agents and server state; one variant for each of M2a and M2b. These invariants establish guard strengthening for server authentication by the initiator and the responder.

##### definition

$m2\text{-inv4}\text{-M2a} :: m2\text{-state set}$

##### where

$m2\text{-inv4}\text{-M2a} \equiv \{s. \forall A\ B\ Kab\ Ts.$   
 $Secure\ Sv\ A\ (Msg\ [aAgt\ B,\ aKey\ Kab,\ aNum\ Ts]) \in chan\ s \longrightarrow A \in good \longrightarrow$   
 $(\exists Rs. Kab = sesK\ (Rs\$sk) \wedge$   
 $runs\ s\ Rs = Some\ (Serv,\ [A,\ B],\ [aNum\ Ts]))$   
 $\}$

##### definition

$m2\text{-inv4}\text{-M2b} :: m2\text{-state set}$

##### where

$m2\text{-inv4}\text{-M2b} \equiv \{s. \forall A\ B\ Kab\ Ts.$   
 $Secure\ Sv\ B\ (Msg\ [aKey\ Kab,\ aAgt\ A,\ aNum\ Ts]) \in chan\ s \longrightarrow B \in good \longrightarrow$   
 $(\exists Rs. Kab = sesK\ (Rs\$sk) \wedge$   
 $runs\ s\ Rs = Some\ (Serv,\ [A,\ B],\ [aNum\ Ts]))$   
 $\}$

**lemmas**  $m2\text{-inv4}\text{-M2aI} =$   
 $m2\text{-inv4}\text{-M2a}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv4}\text{-M2aE}$  [elim] =  
 $m2\text{-inv4}\text{-M2a}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv4}\text{-M2aD} =$   
 $m2\text{-inv4}\text{-M2a}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

**lemmas**  $m2\text{-inv4}\text{-M2bI} = m2\text{-inv4}\text{-M2b}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv4}\text{-M2bE}$  [elim] =  
 $m2\text{-inv4}\text{-M2b}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv4}\text{-M2bD} =$   
 $m2\text{-inv4}\text{-M2b}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proofs.

**lemma**  $PO\text{-}m2\text{-inv4}\text{-M2a}\text{-init}$  [iff]:  
 $init\ m2 \subseteq m2\text{-inv4}\text{-M2a}$

*<proof>*

**lemma** *PO-m2-inv4-M2a-trans* [iff]:  
    {m2-inv4-M2a} trans m2 {> m2-inv4-M2a}  
*<proof>*

**lemma** *PO-m2-inv4-M2a* [iff]: reach m2  $\subseteq$  m2-inv4-M2a  
*<proof>*

**lemma** *PO-m2-inv4-M2b-init* [iff]:  
    init m2  $\subseteq$  m2-inv4-M2b  
*<proof>*

**lemma** *PO-m2-inv4-M2b-trans* [iff]:  
    {m2-inv4-M2b} trans m2 {> m2-inv4-M2b}  
*<proof>*

**lemma** *PO-m2-inv4-M2b* [iff]: reach m2  $\subseteq$  m2-inv4-M2b  
*<proof>*

Consequence needed in proof of inv8/step5 and inv9/step4: The session key uniquely identifies other fields in M2a and M2b, provided it is secret.

**lemma** *m2-inv4-M2a-M2b-match*:  
    [[ Secure Sv A' (Msg [aAgt B', aKey Kab, aNum Ts])  $\in$  chan s;  
      Secure Sv B (Msg [aKey Kab, aAgt A, aNum Ts])  $\in$  chan s;  
      aKey Kab  $\notin$  extr ik0 (chan s); s  $\in$  m2-inv4-M2a; s  $\in$  m2-inv4-M2b ]]  
     $\implies$  A = A'  $\wedge$  B = B'  $\wedge$  Ts = Ts'  
*<proof>*

More consequences of invariants. Needed in ref/step4 and ref/step5 respectively to show the strengthening of the authorization guards.

**lemma** *m2-inv34-M2a-authorized*:  
    **assumes** Secure Sv A (Msg [aAgt B, aKey K, aNum T])  $\in$  chan s  
            s  $\in$  m2-inv3-extrKey s  $\in$  m2-inv4-M2a K  $\notin$  leak s  
    **shows** (K, A)  $\in$  azC (runs s)  
*<proof>*

**lemma** *m2-inv34-M2b-authorized*:  
    **assumes** Secure Sv B (Msg [aKey K, aAgt A, aNum T])  $\in$  chan s  
            s  $\in$  m2-inv3-extrKey s  $\in$  m2-inv4-M2b K  $\notin$  leak s  
    **shows** (K, B)  $\in$  azC (runs s)  
*<proof>*

### inv5: Key secrecy for server

inv5: Key secrecy from server perspective. This invariant links the abstract notion of key secrecy to the intruder key knowledge.

**definition**  
    m2-inv5-ikk-sv :: m2-state set  
**where**

$$\begin{aligned}
& m2\text{-inv5}\text{-ikk}\text{-sv} \equiv \{s. \forall R A B al. \\
& \quad \text{runs } s R = \text{Some } (\text{Serv}, [A, B], al) \longrightarrow A \in \text{good} \longrightarrow B \in \text{good} \longrightarrow \\
& \quad \text{aKey } (\text{sesK } (R\$sk)) \in \text{extr ik0 } (\text{chan } s) \longrightarrow \\
& \quad \text{sesK } (R\$sk) \in \text{leak } s \\
& \}
\end{aligned}$$

**lemmas**  $m2\text{-inv5}\text{-ikk}\text{-svI} = m2\text{-inv5}\text{-ikk}\text{-sv}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv5}\text{-ikk}\text{-svE}$  [elim] =  $m2\text{-inv5}\text{-ikk}\text{-sv}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv5}\text{-ikk}\text{-svD} = m2\text{-inv5}\text{-ikk}\text{-sv}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proof.

**lemma**  $PO\text{-}m2\text{-inv5}\text{-ikk}\text{-sv}\text{-init}$  [iff]:

$\text{init } m2 \subseteq m2\text{-inv5}\text{-ikk}\text{-sv}$   
⟨proof⟩

**lemma**  $PO\text{-}m2\text{-inv5}\text{-ikk}\text{-sv}\text{-trans}$  [iff]:

$\{m2\text{-inv5}\text{-ikk}\text{-sv} \cap m2\text{-inv3a}\text{-sesK}\text{-compr} \cap m2\text{-inv3}\text{-extrKey}\}$   
 $\text{trans } m2$   
 $\{> m2\text{-inv5}\text{-ikk}\text{-sv}\}$   
⟨proof⟩

**lemma**  $PO\text{-}m2\text{-inv5}\text{-ikk}\text{-sv}$  [iff]:  $\text{reach } m2 \subseteq m2\text{-inv5}\text{-ikk}\text{-sv}$

⟨proof⟩

## inv6/7: Key secrecy for initiator and responder

These invariants are derivable.

### definition

$m2\text{-inv6}\text{-ikk}\text{-init} :: m2\text{-state set}$

**where**

$$\begin{aligned}
& m2\text{-inv6}\text{-ikk}\text{-init} \equiv \{s. \forall A B Ra K Ts nl. \\
& \quad \text{runs } s Ra = \text{Some } (\text{Init}, [A, B], \text{aKey } K \# \text{aNum } Ts \# nl) \longrightarrow \\
& \quad A \in \text{good} \longrightarrow B \in \text{good} \longrightarrow \text{aKey } K \in \text{extr ik0 } (\text{chan } s) \longrightarrow \\
& \quad K \in \text{leak } s \\
& \}
\end{aligned}$$

**lemmas**  $m2\text{-inv6}\text{-ikk}\text{-initI} = m2\text{-inv6}\text{-ikk}\text{-init}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m2\text{-inv6}\text{-ikk}\text{-initE}$  [elim] =  $m2\text{-inv6}\text{-ikk}\text{-init}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m2\text{-inv6}\text{-ikk}\text{-initD} = m2\text{-inv6}\text{-ikk}\text{-init}\text{-def}$  [THEN setc-def-to-dest, rule-format, rotated 1]

### definition

$m2\text{-inv7}\text{-ikk}\text{-resp} :: m2\text{-state set}$

**where**

$$\begin{aligned}
& m2\text{-inv7}\text{-ikk}\text{-resp} \equiv \{s. \forall A B Rb K Ts nl. \\
& \quad \text{runs } s Rb = \text{Some } (\text{Resp}, [A, B], \text{aKey } K \# \text{aNum } Ts \# nl) \longrightarrow \\
& \quad A \in \text{good} \longrightarrow B \in \text{good} \longrightarrow \text{aKey } K \in \text{extr ik0 } (\text{chan } s) \longrightarrow \\
& \quad K \in \text{leak } s \\
& \}
\end{aligned}$$

**lemmas**  $m2\text{-inv}\gamma\text{-ikk}\text{-resp}I = m2\text{-inv}\gamma\text{-ikk}\text{-resp}\text{-def}$  [*THEN setc-def-to-intro, rule-format*]  
**lemmas**  $m2\text{-inv}\gamma\text{-ikk}\text{-resp}E$  [*elim*] =  $m2\text{-inv}\gamma\text{-ikk}\text{-resp}\text{-def}$  [*THEN setc-def-to-elim, rule-format*]  
**lemmas**  $m2\text{-inv}\gamma\text{-ikk}\text{-resp}D = m2\text{-inv}\gamma\text{-ikk}\text{-resp}\text{-def}$  [*THEN setc-def-to-dest, rule-format, rotated 1*]

### 3.14.5 Refinement

The simulation relation. This is a pure superposition refinement.

**definition**

$R12 :: (m1\text{-state} \times m2\text{-state})$  set **where**  
 $R12 \equiv \{(s, t). \text{runs } s = \text{runs } t \wedge \text{leak } s = \text{leak } t \wedge \text{clk } s = \text{clk } t\}$

The mediator function is the identity.

**definition**

$med21 :: m2\text{-obs} \Rightarrow m1\text{-obs}$  **where**  
 $med21 = id$

Refinement proof.

**lemma** *PO-m2-step1-refines-m1-step1*:

$\{R12\}$   
 $(m1\text{-step1 } Ra \ A \ B), (m2\text{-step1 } Ra \ A \ B)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-step2-refines-m1-step2*:

$\{R12\}$   
 $(m1\text{-step2 } Rb \ A \ B), (m2\text{-step2 } Rb \ A \ B)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-step3-refines-m1-step3*:

$\{R12\}$   
 $(m1\text{-step3 } Rs \ A \ B \ Kab \ Ts), (m2\text{-step3 } Rs \ A \ B \ Kab \ Ts)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-step4-refines-m1-step4*:

$\{R12 \cap UNIV \times (m2\text{-inv}\gamma\text{-M2a} \cap m2\text{-inv}\gamma\text{-extrKey})\}$   
 $(m1\text{-step4 } Ra \ A \ B \ Kab \ Ts), (m2\text{-step4 } Ra \ A \ B \ Kab \ Ts)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-step5-refines-m1-step5*:

$\{R12 \cap UNIV \times (m2\text{-inv}\gamma\text{-M2b} \cap m2\text{-inv}\gamma\text{-extrKey})\}$  — REMOVED!:  $m2\text{-inv}\gamma\text{-ikk}\text{-sv}$   
 $(m1\text{-step5 } Rb \ A \ B \ Kab \ Ts), (m2\text{-step5 } Rb \ A \ B \ Kab \ Ts)$   
 $\{> R12\}$   
 $\langle proof \rangle$

**lemma** *PO-m2-tick-refines-m1-tick*:

$\{R12\}$   
 $(m1\text{-tick } T), (m2\text{-tick } T)$   
 $\{> R12\}$

$\langle proof \rangle$

**lemma** *PO-m2-leak-refines-m1-leak*:

$\{R12\}$   
 $(m1\text{-leak } Rs), (m2\text{-leak } Rs)$   
 $\{> R12\}$

$\langle proof \rangle$

**lemma** *PO-m2-fake-refines-skip*:

$\{R12\}$   
 $Id, m2\text{-fake}$   
 $\{> R12\}$

$\langle proof \rangle$

All together now...

**lemmas** *PO-m2-trans-refines-m1-trans* =

*PO-m2-step1-refines-m1-step1 PO-m2-step2-refines-m1-step2*  
*PO-m2-step3-refines-m1-step3 PO-m2-step4-refines-m1-step4*  
*PO-m2-step5-refines-m1-step5 PO-m2-tick-refines-m1-tick*  
*PO-m2-leak-refines-m1-leak PO-m2-fake-refines-skip*

**lemma** *PO-m2-refines-init-m1* [iff]:

$init\ m2 \subseteq R12 \text{“}(init\ m1)$

$\langle proof \rangle$

**lemma** *PO-m2-refines-trans-m1* [iff]:

$\{R12 \cap$   
 $UNIV \times (m2\text{-inv4-M2b} \cap m2\text{-inv4-M2a} \cap m2\text{-inv3-extrKey})\}$   
 $(trans\ m1), (trans\ m2)$   
 $\{> R12\}$

$\langle proof \rangle$

**lemma** *PO-obs-consistent-R12* [iff]:

$obs\text{-consistent } R12\ med21\ m1\ m2$

$\langle proof \rangle$

Refinement result.

**lemma** *m2-refines-m1* [iff]:

$refines$   
 $(R12 \cap$   
 $(UNIV \times (m2\text{-inv4-M2b} \cap m2\text{-inv4-M2a} \cap m2\text{-inv3-extrKey} \cap m2\text{-inv3a-sesK-compr})))$   
 $med21\ m1\ m2$

$\langle proof \rangle$

**lemma** *m2-implements-m1* [iff]:

$implements\ med21\ m1\ m2$

$\langle proof \rangle$

### 3.14.6 Inherited and derived invariants

end

### 3.15 Denning-Sacco, direct variant (L3)

**theory** *m3-ds-par* **imports** *m2-ds* *./Refinement/Message*  
**begin**

We model a direct implementation of the channel-based Denning-Sacco protocol at Level 2. In this version, there is no ticket forwarding.

- M1.  $A \rightarrow S : A, B$   
M2a.  $S \rightarrow A : \{Kab, B, Ts\}_{Kas}$   
M2b.  $S \rightarrow B : \{Kab, A, Ts\}_{Kbs}$

Proof tool configuration. Avoid annoying automatic unfolding of *dom*.

**declare** *domIff* [*simp*, *iff del*]

#### 3.15.1 Setup

Now we can define the initial key knowledge.

**overloading** *ltkeySetup'*  $\equiv$  *ltkeySetup* **begin**

**definition** *ltkeySetup-def*: *ltkeySetup'*  $\equiv$   $\{(shrK\ C, A) \mid C\ A.\ A = C \vee A = Sv\}$

**end**

**lemma** *corrKey-shrK-bad* [*simp*]: *corrKey* = *shrK'bad*  
 $\langle proof \rangle$

#### 3.15.2 State

The secure channels are star-shaped to/from the server. Therefore, we have only one agent in the relation.

**record** *m3-state* = *m1-state* +  
*IK* :: *msg set* — intruder knowledge

Observable state: *runs*, *leak*, *clk*, and *cache*.

**type-synonym**  
*m3-obs* = *m2-obs*

**definition**  
*m3-obs* :: *m3-state*  $\Rightarrow$  *m3-obs* **where**  
*m3-obs* *s*  $\equiv$   $(\mid runs = runs\ s, leak = leak\ s, clk = clk\ s \mid)$

**type-synonym**  
*m3-pred* = *m3-state set*

**type-synonym**  
*m3-trans* = (*m3-state*  $\times$  *m3-state*) *set*

#### 3.15.3 Events

Protocol events.

**definition** — by *A*, refines *m2-step1*

$m3\text{-step1} :: [\text{rid-t}, \text{agent}, \text{agent}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step1 } Ra \ A \ B \equiv \{(s, s1)\}.$

— guards:

$Ra \notin \text{dom} (\text{runs } s) \wedge$  —  $Ra$  is fresh

— actions:

$s1 = s\langle$

$\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [])),$

$IK := \text{insert } \{\text{Agent } A, \text{Agent } B\} (IK \ s)$  — send  $M1$

$\rangle$

$\}$

**definition** — by  $B$ , refines  $m2\text{-step2}$

$m3\text{-step2} :: [\text{rid-t}, \text{agent}, \text{agent}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step2} \equiv m1\text{-step2}$

**definition** — by  $Server$ , refines  $m2\text{-step3}$

$m3\text{-step3} :: [\text{rid-t}, \text{agent}, \text{agent}, \text{key}, \text{time}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step3 } Rs \ A \ B \ Kab \ Ts \equiv \{(s, s1)\}.$

— guards:

$Rs \notin \text{dom} (\text{runs } s) \wedge$  — fresh server run

$Kab = \text{sesK} (Rs\$sk) \wedge$  — fresh session key

$\{\text{Agent } A, \text{Agent } B\} \in IK \ s \wedge$  — recv  $M1$

$Ts = \text{clk } s \wedge$  — fresh timestamp

— actions:

— record session key and send  $M2$

$s1 = s\langle$

$\text{runs} := (\text{runs } s)(Rs \mapsto (\text{Serv}, [A, B], [\text{aNum } Ts])),$  — send  $M2a, M2b$

$IK := \text{insert } (\text{Crypt } (\text{shrK } A) \ \{\text{Agent } B, \text{Key } Kab, \text{Number } Ts\})$

$(\text{insert } (\text{Crypt } (\text{shrK } B) \ \{\text{Key } Kab, \text{Agent } A, \text{Number } Ts\}) (IK \ s))$

$\rangle$

$\}$

**definition** — by  $A$ , refines  $m2\text{-step4}$

$m3\text{-step4} :: [\text{rid-t}, \text{agent}, \text{agent}, \text{key}, \text{time}] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step4 } Ra \ A \ B \ Kab \ Ts \equiv \{(s, s1)\}.$

— guards:

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], []) \wedge$  — key not yet recv'd

$\text{Crypt } (\text{shrK } A)$  — recv  $M2$

$\{\text{Agent } B, \text{Key } Kab, \text{Number } Ts\} \in IK \ s \wedge$

— check freshness of session key

$\text{clk } s < Ts + Ls \wedge$

— actions:

— record session key  
 $s1 = s \{$   
 $\quad runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts]))$   
 $\}$   
 $\}$

**definition** — by  $B$ , refines  $m2\text{-}step5$   
 $m3\text{-}step5 :: [rid\text{-}t, agent, agent, key, time] \Rightarrow m3\text{-}trans$

**where**

$m3\text{-}step5\ Rb\ A\ B\ Kab\ Ts \equiv \{(s, s1)\}.$

— guards:

$runs\ s\ Rb = Some\ (Resp, [A, B], []) \wedge$  — key not yet recv'd

$Crypt\ (shrK\ B)\ \{Key\ Kab, Agent\ A, Number\ Ts\} \in IK\ s \wedge$  — recv  $M3$

— ensure freshness of session key; replays with fresh authenticator ok!  
 $clk\ s < Ts + Ls \wedge$

— actions:

— record session key

$s1 = s \{$   
 $\quad runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts]))$   
 $\}$

$\}$

Clock tick event

**definition** — refines  $m2\text{-}tick$   
 $m3\text{-}tick :: time \Rightarrow m3\text{-}trans$

**where**

$m3\text{-}tick \equiv m1\text{-}tick$

Session key compromise.

**definition** — refines  $m2\text{-}leak$   
 $m3\text{-}leak :: rid\text{-}t \Rightarrow m3\text{-}trans$

**where**

$m3\text{-}leak\ Rs \equiv \{(s, s1)\}.$

— guards:

$Rs \in dom\ (runs\ s) \wedge$

$fst\ (the\ (runs\ s\ Rs)) = Serv \wedge$  — compromise server run  $Rs$

— actions:

— record session key as leaked and add it to intruder knowledge

$s1 = s \{ leak := insert\ (sesK\ (Rs\$sk))\ (leak\ s),$   
 $\quad IK := insert\ (Key\ (sesK\ (Rs\$sk)))\ (IK\ s) \}$

$\}$

Intruder fake event. The following "Dolev-Yao" event generates all intruder-derivable messages.

**definition** — refines  $m2\text{-}fake$   
 $m3\text{-}DY\text{-}fake :: m3\text{-}trans$

**where**

$m3\text{-}DY\text{-}fake \equiv \{(s, s1)\}.$

```

— actions:
s1 = s(| IK := synth (anzl (IK s)) |) — take DY closure
}

```

### 3.15.4 Transition system

**definition**

```
m3-init :: m3-pred
```

**where**

```

m3-init ≡ { (|
  runs = Map.empty,
  leak = shrK'bad,
  clk = 0,
  IK = Key'shrK'bad
|) }

```

**definition**

```

m3-trans :: m3-trans where
m3-trans ≡ (| ∪ A B Ra Rb Rs Kab Ts T.
  m3-step1 Ra A B ∪
  m3-step2 Rb A B ∪
  m3-step3 Rs A B Kab Ts ∪
  m3-step4 Ra A B Kab Ts ∪
  m3-step5 Rb A B Kab Ts ∪
  m3-tick T ∪
  m3-leak Rs ∪
  m3-DY-fake ∪
  Id
|)

```

**definition**

```

m3 :: (m3-state, m3-obs) spec where
m3 ≡ (|
  init = m3-init,
  trans = m3-trans,
  obs = m3-obs
|)

```

**lemmas** *m3-loc-defs* =

```

m3-def m3-init-def m3-trans-def m3-obs-def
m3-step1-def m3-step2-def m3-step3-def m3-step4-def m3-step5-def
m3-tick-def m3-leak-def m3-DY-fake-def

```

**lemmas** *m3-defs* = *m3-loc-defs* *m2-defs*

### 3.15.5 Invariants

Specialized injection that we can apply more aggressively.

**lemmas** *anzl-Inj-IK* = *anzl.Inj* [**where** *H=IK s for s*]

**lemmas** *parts-Inj-IK* = *parts.Inj* [**where** *H=IK s for s*]

**declare** *parts-Inj-IK* [*dest!*]

**declare** *analz-into-parts* [*dest*]

### inv1: Secrecy of pre-distributed shared keys

#### definition

*m3-inv1-lkeysec* :: *m3-pred*

#### where

$m3\text{-inv1}\text{-lkeysec} \equiv \{s. \forall C. \\ (Key (shrK C) \in parts (IK s) \longrightarrow C \in bad) \wedge \\ (C \in bad \longrightarrow Key (shrK C) \in IK s) \\ \}$

**lemmas** *m3-inv1-lkeysecI* = *m3-inv1-lkeysec-def* [*THEN setc-def-to-intro, rule-format*]

**lemmas** *m3-inv1-lkeysecE* [*elim*] = *m3-inv1-lkeysec-def* [*THEN setc-def-to-elim, rule-format*]

**lemmas** *m3-inv1-lkeysecD* = *m3-inv1-lkeysec-def* [*THEN setc-def-to-dest, rule-format*]

Invariance proof.

**lemma** *PO-m3-inv1-lkeysec-init* [*iff*]:

$init\ m3 \subseteq m3\text{-inv1}\text{-lkeysec}$

*<proof>*

**lemma** *PO-m3-inv1-lkeysec-trans* [*iff*]:

$\{m3\text{-inv1}\text{-lkeysec}\ trans\ m3 \{>\ m3\text{-inv1}\text{-lkeysec}\}$

*<proof>*

**lemma** *PO-m3-inv1-lkeysec* [*iff*]:  $reach\ m3 \subseteq m3\text{-inv1}\text{-lkeysec}$

*<proof>*

Useful simplifier lemmas

**lemma** *m3-inv1-lkeysec-for-parts* [*simp*]:

$\llbracket s \in m3\text{-inv1}\text{-lkeysec} \rrbracket \implies Key (shrK C) \in parts (IK s) \longleftrightarrow C \in bad$

*<proof>*

**lemma** *m3-inv1-lkeysec-for-analz* [*simp*]:

$\llbracket s \in m3\text{-inv1}\text{-lkeysec} \rrbracket \implies Key (shrK C) \in analz (IK s) \longleftrightarrow C \in bad$

*<proof>*

### inv3: Session keys not used to encrypt other session keys

Session keys are not used to encrypt other keys. Proof requires generalization to sets of session keys.

NOTE: This invariant will be derived from the corresponding L2 invariant using the simulation relation.

#### definition

*m3-inv3-sesK-compr* :: *m3-pred*

#### where

$m3\text{-inv3}\text{-sesK}\text{-compr} \equiv \{s. \forall K KK. \\ KK \subseteq range\ sesK \longrightarrow$

$KK \subseteq range\ sesK \longrightarrow$

$(Key\ K \in analz (Key\ KK \cup (IK\ s))) = (K \in KK \vee Key\ K \in analz (IK\ s))$

}

**lemmas**  $m3\text{-inv3}\text{-sesK}\text{-comprI} = m3\text{-inv3}\text{-sesK}\text{-compr}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m3\text{-inv3}\text{-sesK}\text{-comprE} = m3\text{-inv3}\text{-sesK}\text{-compr}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m3\text{-inv3}\text{-sesK}\text{-comprD} = m3\text{-inv3}\text{-sesK}\text{-compr}\text{-def}$  [THEN setc-def-to-dest, rule-format]

Additional lemma

**lemmas**  $insert\text{-commute}\text{-Key} = insert\text{-commute}$  [where  $x=Key\ K$  for  $K$ ]

**lemmas**  $m3\text{-inv3}\text{-sesK}\text{-compr}\text{-simps} =$

$m3\text{-inv3}\text{-sesK}\text{-comprD}$

$m3\text{-inv3}\text{-sesK}\text{-comprD}$  [where  $KK=insert\ Kab\ KK$  for  $Kab\ KK$ , simplified]

$m3\text{-inv3}\text{-sesK}\text{-comprD}$  [where  $KK=\{Kab\}$  for  $Kab$ , simplified]

$insert\text{-commute}\text{-Key}$

### 3.15.6 Refinement

#### Message abstraction and simulation relation

Abstraction function on sets of messages.

**inductive-set**

$abs\text{-msg} :: msg\ set \Rightarrow chmsg\ set$

for  $H :: msg\ set$

**where**

$am\text{-M1}$ :

$\{\{Agent\ A, Agent\ B\}\} \in H$

$\implies Insec\ A\ B\ (Msg\ []) \in abs\text{-msg}\ H$

|  $am\text{-M2a}$ :

$Crypt\ (shrK\ C)\ \{\{Agent\ B, Key\ K, Number\ T\}\} \in H$

$\implies Secure\ Sv\ C\ (Msg\ [aAgt\ B, aKey\ K, aNum\ T]) \in abs\text{-msg}\ H$

|  $am\text{-M2b}$ :

$Crypt\ (shrK\ C)\ \{\{Key\ K, Agent\ A, Number\ T\}\} \in H$

$\implies Secure\ Sv\ C\ (Msg\ [aKey\ K, aAgt\ A, aNum\ T]) \in abs\text{-msg}\ H$

R23: The simulation relation. This is a data refinement of the insecure and secure channels of refinement 2.

**definition**

$R23\text{-msgs} :: (m2\text{-state} \times m3\text{-state})\ set\ \mathbf{where}$

$R23\text{-msgs} \equiv \{(s, t). abs\text{-msg}\ (parts\ (IK\ t)) \subseteq chan\ s\}$

**definition**

$R23\text{-keys} :: (m2\text{-state} \times m3\text{-state})\ set\ \mathbf{where}$

$R23\text{-keys} \equiv \{(s, t). \forall KK\ K. KK \subseteq range\ sesK \longrightarrow$

$Key\ K \in analz\ (Key'KK \cup (IK\ t)) \longleftrightarrow aKey\ K \in extr\ (aKey'KK \cup ik0)\ (chan\ s)$

}

**definition**

$R23\text{-pres} :: (m2\text{-state} \times m3\text{-state})\ set\ \mathbf{where}$

$R23\text{-pres} \equiv \{(s, t). runs\ s = runs\ t \wedge leak\ s = leak\ t \wedge clk\ s = clk\ t\}$

**definition**

$R23 :: (m2\text{-state} \times m3\text{-state})\ set\ \mathbf{where}$

$R23 \equiv R23\text{-msgs} \cap R23\text{-keys} \cap R23\text{-pres}$

**lemmas**  $R23\text{-defs} =$   
 $R23\text{-def } R23\text{-msgs-def } R23\text{-keys-def } R23\text{-pres-def}$

The mediator function is the identity here.

**definition**  
 $med32 :: m3\text{-obs} \Rightarrow m2\text{-obs}$  **where**  
 $med32 \equiv id$

**lemmas**  $R23\text{-msgsI} = R23\text{-msgs-def}$  [*THEN rel-def-to-intro, simplified, rule-format*]  
**lemmas**  $R23\text{-msgsE}$  [*elim*] =  $R23\text{-msgs-def}$  [*THEN rel-def-to-elim, simplified, rule-format*]

**lemmas**  $R23\text{-keysI} = R23\text{-keys-def}$  [*THEN rel-def-to-intro, simplified, rule-format*]  
**lemmas**  $R23\text{-keysE}$  [*elim*] =  $R23\text{-keys-def}$  [*THEN rel-def-to-elim, simplified, rule-format*]

**lemmas**  $R23\text{-presI} = R23\text{-pres-def}$  [*THEN rel-def-to-intro, simplified, rule-format*]  
**lemmas**  $R23\text{-presE}$  [*elim*] =  $R23\text{-pres-def}$  [*THEN rel-def-to-elim, simplified, rule-format*]

**lemmas**  $R23\text{-intros} = R23\text{-msgsI } R23\text{-keysI } R23\text{-presI}$

Simplifier lemmas for various instantiations (for keys).

**lemmas**  $R23\text{-keys-simp} = R23\text{-keys-def}$  [*THEN rel-def-to-dest, simplified, rule-format*]  
**lemmas**  $R23\text{-keys-simps} =$   
 $R23\text{-keys-simp}$   
 $R23\text{-keys-simp}$  [**where**  $KK = \{\}$ , *simplified*]  
 $R23\text{-keys-simp}$  [**where**  $KK = \{K'\}$  **for**  $K'$ , *simplified*]  
 $R23\text{-keys-simp}$  [**where**  $KK = \text{insert } K' \text{ } KK$  **for**  $K' \text{ } KK$ , *simplified, OF - conjI*]

## General lemmas

General facts about *abs-msg*

**declare**  $abs\text{-msg.intros}$  [*intro!*]  
**declare**  $abs\text{-msg.cases}$  [*elim!*]

**lemma**  $abs\text{-msg-empty}$ :  $abs\text{-msg } \{\} = \{\}$   
 $\langle proof \rangle$

**lemma**  $abs\text{-msg-Un}$  [*simp*]:  
 $abs\text{-msg } (G \cup H) = abs\text{-msg } G \cup abs\text{-msg } H$   
 $\langle proof \rangle$

**lemma**  $abs\text{-msg-mono}$  [*elim*]:  
 $\llbracket m \in abs\text{-msg } G; G \subseteq H \rrbracket \Longrightarrow m \in abs\text{-msg } H$   
 $\langle proof \rangle$

**lemma**  $abs\text{-msg-insert-mono}$  [*intro*]:  
 $\llbracket m \in abs\text{-msg } H \rrbracket \Longrightarrow m \in abs\text{-msg } (\text{insert } m' \text{ } H)$   
 $\langle proof \rangle$

Facts about *abs-msg* concerning abstraction of fakeable messages. This is crucial for proving the refinement of the intruder event.

**lemma** *abs-msg-DY-subset-fakeable*:

$$\begin{aligned} & \llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; t \in m3\text{-inv1-lkeysec} \rrbracket \\ & \implies \text{abs-msg } (\text{synth } (\text{analz } (IK\ t))) \subseteq \text{fake ik0 } (\text{dom } (\text{runs } s)) (\text{chan } s) \end{aligned}$$

*<proof>*

## Refinement proof

Pair decomposition. These were set to `elim!`, which is too aggressive here.

**declare** *MPair-analz* [rule del, elim]

**declare** *MPair-parts* [rule del, elim]

Protocol events.

**lemma** *PO-m3-step1-refines-m2-step1*:

$$\begin{aligned} & \{R23\} \\ & (m2\text{-step1 } Ra\ A\ B), (m3\text{-step1 } Ra\ A\ B) \\ & \{> R23\} \end{aligned}$$

*<proof>*

**lemma** *PO-m3-step2-refines-m2-step2*:

$$\begin{aligned} & \{R23\} \\ & (m2\text{-step2 } Rb\ A\ B), (m3\text{-step2 } Rb\ A\ B) \\ & \{> R23\} \end{aligned}$$

*<proof>*

**lemma** *PO-m3-step3-refines-m2-step3*:

$$\begin{aligned} & \{R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv3-sesK-compr} \cap m3\text{-inv1-lkeysec})\} \\ & (m2\text{-step3 } Rs\ A\ B\ Kab\ Ts), (m3\text{-step3 } Rs\ A\ B\ Kab\ Ts) \\ & \{> R23\} \end{aligned}$$

*<proof>*

**lemma** *PO-m3-step4-refines-m2-step4*:

$$\begin{aligned} & \{R23 \cap UNIV \times (m3\text{-inv1-lkeysec})\} \\ & (m2\text{-step4 } Ra\ A\ B\ Kab\ Ts), (m3\text{-step4 } Ra\ A\ B\ Kab\ Ts) \\ & \{> R23\} \end{aligned}$$

*<proof>*

**lemma** *PO-m3-step5-refines-m2-step5*:

$$\begin{aligned} & \{R23\} \\ & (m2\text{-step5 } Rb\ A\ B\ Kab\ Ts), (m3\text{-step5 } Rb\ A\ B\ Kab\ Ts) \\ & \{> R23\} \end{aligned}$$

*<proof>*

**lemma** *PO-m3-tick-refines-m2-tick*:

$$\begin{aligned} & \{R23\} \\ & (m2\text{-tick } T), (m3\text{-tick } T) \\ & \{> R23\} \end{aligned}$$

*<proof>*

Intruder events.

**lemma** *PO-m3-leak-refines-m2-leak*:

{*R23*}  
(*m2-leak Rs*), (*m3-leak Rs*)  
{>*R23*}  
<proof>

**lemma** *PO-m3-DY-fake-refines-m2-fake*:

{*R23*  $\cap$  *UNIV*  $\times$  (*m3-inv1-lkeysec*)}  
*m2-fake*, *m3-DY-fake*  
{> *R23*}  
<proof>

All together now...

**lemmas** *PO-m3-trans-refines-m2-trans* =

*PO-m3-step1-refines-m2-step1 PO-m3-step2-refines-m2-step2*  
*PO-m3-step3-refines-m2-step3 PO-m3-step4-refines-m2-step4*  
*PO-m3-step5-refines-m2-step5 PO-m3-tick-refines-m2-tick*  
*PO-m3-leak-refines-m2-leak PO-m3-DY-fake-refines-m2-fake*

**lemma** *PO-m3-refines-init-m2* [*iff*]:

*init m3*  $\subseteq$  *R23*“(*init m2*)  
<proof>

**lemma** *PO-m3-refines-trans-m2* [*iff*]:

{*R23*  $\cap$  (*m2-inv3a-sesK-compr*)  $\times$  (*m3-inv3-sesK-compr*  $\cap$  *m3-inv1-lkeysec*)}  
(*trans m2*), (*trans m3*)  
{> *R23*}  
<proof>

**lemma** *PO-m3-observation-consistent* [*iff*]:

*obs-consistent R23 med32 m2 m3*  
<proof>

Refinement result.

**lemma** *m3-refines-m2* [*iff*]:

*refines*  
(*R23*  $\cap$  (*m2-inv3a-sesK-compr*)  $\times$  (*m3-inv1-lkeysec*))  
*med32 m2 m3*  
<proof>

**lemma** *m3-implements-m2* [*iff*]:

*implements med32 m2 m3*  
<proof>

end

### 3.16 Denning-Sacco protocol (L3)

**theory** *m3-ds imports m2-ds ../Refinement/Message*  
**begin**

We model the Denning-Sacco protocol:

- M1.  $A \rightarrow S : A, B$
- M2.  $S \rightarrow A : \{Kab, B, Ts, Na, \{Kab, A, Ts\}_{Kbs}\}_{Kas}$
- M3.  $A \rightarrow B : \{Kab, A, Ts\}_{Kbs}$

Proof tool configuration. Avoid annoying automatic unfolding of *dom*.

**declare** *domIff* [*simp*, *iff del*]

### 3.16.1 Setup

Now we can define the initial key knowledge.

**overloading** *ltkkeySetup'*  $\equiv$  *ltkkeySetup* **begin**

**definition** *ltkkeySetup-def*: *ltkkeySetup'*  $\equiv$   $\{(sharK\ C, A) \mid C\ A.\ A = C \vee A = Sv\}$

**end**

**lemma** *corrKey-shrK-bad* [*simp*]: *corrKey* = *shrK'bad*

*<proof>*

### 3.16.2 State

The secure channels are star-shaped to/from the server. Therefore, we have only one agent in the relation.

**record** *m3-state* = *m1-state* +

*IK* :: *msg set*

— intruder knowledge

Observable state: *runs*, *leak*, *clk*, and *cache*.

**type-synonym**

*m3-obs* = *m2-obs*

**definition**

*m3-obs* :: *m3-state*  $\Rightarrow$  *m3-obs* **where**

*m3-obs* *s*  $\equiv$   $(\mid runs = runs\ s, leak = leak\ s, clk = clk\ s \mid)$

**type-synonym**

*m3-pred* = *m3-state set*

**type-synonym**

*m3-trans* = (*m3-state*  $\times$  *m3-state*) *set*

### 3.16.3 Events

Protocol events.

**definition** — by *A*, refines *m2-step1*

*m3-step1* :: [*rid-t*, *agent*, *agent*]  $\Rightarrow$  *m3-trans*

**where**

*m3-step1* *Ra* *A* *B*  $\equiv$   $\{(s, s1)\}$ .

— guards:

*Ra*  $\notin$  *dom* (*runs* *s*)  $\wedge$

— *Ra* is fresh

— actions:  
 $s1 = s\langle$   
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [])),$   
 $IK := insert \{Agent\ A, Agent\ B\} (IK\ s) \quad \text{— send } M1$   
 $\rangle$   
 $\}$

**definition** — by  $B$ , refines  $m2\text{-step}2$   
 $m3\text{-step}2 :: [rid\text{-}t, agent, agent] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step}2 \equiv m1\text{-step}2$

**definition** — by  $Server$ , refines  $m2\text{-step}3$

$m3\text{-step}3 :: [rid\text{-}t, agent, agent, key, time] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step}3\ Rs\ A\ B\ Kab\ Ts \equiv \{(s, s1).$

— guards:

$Rs \notin dom (runs\ s) \wedge \quad \text{— fresh server run}$   
 $Kab = sesK (Rs\$sk) \wedge \quad \text{— fresh session key}$

$\{Agent\ A, Agent\ B\} \in IK\ s \wedge \quad \text{— rcv } M1$   
 $Ts = clk\ s \wedge \quad \text{— fresh timestamp}$

— actions:

— record session key and send  $M2$

$s1 = s\langle$   
 $runs := (runs\ s)(Rs \mapsto (Serv, [A, B], [aNum\ Ts])),$   
 $IK := insert (Crypt (shrK\ A) \quad \text{— send } M2$   
 $\{Key\ Kab, Agent\ B, Number\ Ts,$   
 $Crypt (shrK\ B) \{Key\ Kab, Agent\ A, Number\ Ts\}\})$   
 $(IK\ s)$   
 $\rangle$   
 $\}$

**definition** — by  $A$ , refines  $m2\text{-step}4$

$m3\text{-step}4 :: [rid\text{-}t, agent, agent, key, time, msg] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step}4\ Ra\ A\ B\ Kab\ Ts\ X \equiv \{(s, s1).$

— guards:

$runs\ s\ Ra = Some (Init, [A, B], []) \wedge \quad \text{— key not yet rcv'd}$

$Crypt (shrK\ A) \quad \text{— rcv } M2$   
 $\{Key\ Kab, Agent\ B, Number\ Ts, X\} \in IK\ s \wedge$

— check freshness of session key

$clk\ s < Ts + Ls \wedge$

— actions:

— record session key and send  $M3$

$s1 = s\langle$   
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts])),$

$$IK := insert\ X\ (IK\ s) \quad \text{--- send } M3$$

$$\Downarrow$$

$$\}$$

**definition** — by  $B$ , refines  $m2\text{-step5}$   
 $m3\text{-step5} :: [rid\text{-}t, agent, agent, key, time] \Rightarrow m3\text{-trans}$

**where**

$m3\text{-step5}\ Rb\ A\ B\ Kab\ Ts \equiv \{(s, s1)\}.$

— guards:

$runs\ s\ Rb = Some\ (Resp, [A, B], []) \wedge$  — key not yet recv'd

$Crypt\ (shrK\ B)\ \{\!|Key\ Kab, Agent\ A, Number\ Ts\!\} \in IK\ s \wedge$  — recv  $M3$

— ensure freshness of session key; replays with fresh authenticator ok!

$clk\ s < Ts + Ls \wedge$

— actions:

— record session key

$s1 = s\{\!$

$runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts]))$

$\!\}$

}

Clock tick event

**definition** — refines  $m2\text{-tick}$

$m3\text{-tick} :: time \Rightarrow m3\text{-trans}$

**where**

$m3\text{-tick} \equiv m1\text{-tick}$

Session key compromise.

**definition** — refines  $m2\text{-leak}$

$m3\text{-leak} :: rid\text{-}t \Rightarrow m3\text{-trans}$

**where**

$m3\text{-leak}\ Rs \equiv \{(s, s1)\}.$

— guards:

$Rs \in dom\ (runs\ s) \wedge$

$fst\ (the\ (runs\ s\ Rs)) = Serv \wedge$  — compromise server run  $Rs$

— actions:

— record session key as leaked and add it to intruder knowledge

$s1 = s\{\! leak := insert\ (sesK\ (Rs\$sk))\ (leak\ s),$

$IK := insert\ (Key\ (sesK\ (Rs\$sk)))\ (IK\ s)\ \!\}$

}

Intruder fake event. The following "Dolev-Yao" event generates all intruder-derivable messages.

**definition** — refines  $m2\text{-fake}$

$m3\text{-DY-fake} :: m3\text{-trans}$

**where**

$m3\text{-DY-fake} \equiv \{(s, s1)\}.$

— actions:

```

    s1 = s(| IK := synth (anzl (IK s)) |) — take DY closure
  }

```

### 3.16.4 Transition system

#### definition

```

m3-init :: m3-pred

```

#### where

```

m3-init ≡ { (|
  runs = Map.empty,
  leak = shrK'bad,
  clk = 0,
  IK = Key'shrK'bad
|) }

```

#### definition

```

m3-trans :: m3-trans where
m3-trans ≡ (| A B Ra Rb Rs Kab Ts T X.
  m3-step1 Ra A B ∪
  m3-step2 Rb A B ∪
  m3-step3 Rs A B Kab Ts ∪
  m3-step4 Ra A B Kab Ts X ∪
  m3-step5 Rb A B Kab Ts ∪
  m3-tick T ∪
  m3-leak Rs ∪
  m3-DY-fake ∪
  Id
|)

```

#### definition

```

m3 :: (m3-state, m3-obs) spec where
m3 ≡ (|
  init = m3-init,
  trans = m3-trans,
  obs = m3-obs
|)

```

#### lemmas

```

m3-loc-defs =
  m3-def m3-init-def m3-trans-def m3-obs-def
  m3-step1-def m3-step2-def m3-step3-def m3-step4-def m3-step5-def
  m3-tick-def m3-leak-def m3-DY-fake-def

```

```

lemmas m3-defs = m3-loc-defs m2-defs

```

### 3.16.5 Invariants

Specialized injection that we can apply more aggressively.

```

lemmas anlz-Inj-IK = anlz.Inj [where H=IK s for s]

```

```

lemmas parts-Inj-IK = parts.Inj [where H=IK s for s]

```

```

declare parts-Inj-IK [dest!]

```

**declare** *analz-into-parts* [*dest*]

## inv1: Secrecy of pre-distributed shared keys

### definition

$m3\text{-inv1}\text{-lkeysec} :: m3\text{-pred}$

### where

$m3\text{-inv1}\text{-lkeysec} \equiv \{s. \forall C.$   
     $(\text{Key } (\text{shrK } C) \in \text{parts } (IK\ s) \longrightarrow C \in \text{bad}) \wedge$   
     $(C \in \text{bad} \longrightarrow \text{Key } (\text{shrK } C) \in IK\ s)$   
}

**lemmas**  $m3\text{-inv1}\text{-lkeysecI} = m3\text{-inv1}\text{-lkeysec}\text{-def}$  [*THEN setc-def-to-intro, rule-format*]

**lemmas**  $m3\text{-inv1}\text{-lkeysecE}$  [*elim*] =  $m3\text{-inv1}\text{-lkeysec}\text{-def}$  [*THEN setc-def-to-elim, rule-format*]

**lemmas**  $m3\text{-inv1}\text{-lkeysecD} = m3\text{-inv1}\text{-lkeysec}\text{-def}$  [*THEN setc-def-to-dest, rule-format*]

Invariance proof.

**lemma**  $PO\text{-}m3\text{-inv1}\text{-lkeysec}\text{-init}$  [*iff*]:

$\text{init } m3 \subseteq m3\text{-inv1}\text{-lkeysec}$

*<proof>*

**lemma**  $PO\text{-}m3\text{-inv1}\text{-lkeysec}\text{-trans}$  [*iff*]:

$\{m3\text{-inv1}\text{-lkeysec}\} \text{trans } m3 \{> m3\text{-inv1}\text{-lkeysec}\}$

*<proof>*

**lemma**  $PO\text{-}m3\text{-inv1}\text{-lkeysec}$  [*iff*]:  $\text{reach } m3 \subseteq m3\text{-inv1}\text{-lkeysec}$

*<proof>*

Useful simplifier lemmas

**lemma**  $m3\text{-inv1}\text{-lkeysec}\text{-for}\text{-parts}$  [*simp*]:

$\llbracket s \in m3\text{-inv1}\text{-lkeysec} \rrbracket \Longrightarrow \text{Key } (\text{shrK } C) \in \text{parts } (IK\ s) \longleftrightarrow C \in \text{bad}$

*<proof>*

**lemma**  $m3\text{-inv1}\text{-lkeysec}\text{-for}\text{-analz}$  [*simp*]:

$\llbracket s \in m3\text{-inv1}\text{-lkeysec} \rrbracket \Longrightarrow \text{Key } (\text{shrK } C) \in \text{analz } (IK\ s) \longleftrightarrow C \in \text{bad}$

*<proof>*

## inv2: Ticket shape for honestly encrypted M2

### definition

$m3\text{-inv2}\text{-ticket} :: m3\text{-pred}$

### where

$m3\text{-inv2}\text{-ticket} \equiv \{s. \forall A\ B\ T\ K\ X.$   
     $A \notin \text{bad} \longrightarrow$   
     $\text{Crypt } (\text{shrK } A) \ \{\!\!| \text{Key } K, \text{Agent } B, \text{Number } T, X \!\!\} \in \text{parts } (IK\ s) \longrightarrow$   
     $X = \text{Crypt } (\text{shrK } B) \ \{\!\!| \text{Key } K, \text{Agent } A, \text{Number } T \!\!\} \wedge K \in \text{range } \text{sesK}$   
}

**lemmas**  $m3\text{-inv2}\text{-ticketI} = m3\text{-inv2}\text{-ticket}\text{-def}$  [*THEN setc-def-to-intro, rule-format*]

**lemmas**  $m3\text{-inv2}\text{-ticketE}$  [*elim*] =  $m3\text{-inv2}\text{-ticket}\text{-def}$  [*THEN setc-def-to-elim, rule-format*]

**lemmas**  $m3\text{-inv2}\text{-ticketD} = m3\text{-inv2}\text{-ticket}\text{-def}$  [*THEN setc-def-to-dest, rule-format, rotated -1*]

Invariance proof.

**lemma** *PO-m3-inv2-ticket-init* [iff]:

$init\ m3 \subseteq m3\text{-inv2}\text{-ticket}$

$\langle proof \rangle$

**lemma** *PO-m3-inv2-ticket-trans* [iff]:

$\{m3\text{-inv2}\text{-ticket} \cap m3\text{-inv1}\text{-lkeysec}\} \text{ trans } m3 \{> m3\text{-inv2}\text{-ticket}\}$

$\langle proof \rangle$

**lemma** *PO-m3-inv2-ticket* [iff]:  $reach\ m3 \subseteq m3\text{-inv2}\text{-ticket}$

$\langle proof \rangle$

### inv3: Session keys not used to encrypt other session keys

Session keys are not used to encrypt other keys. Proof requires generalization to sets of session keys.

#### definition

$m3\text{-inv3}\text{-sesK}\text{-compr} :: m3\text{-pred}$

**where**

$m3\text{-inv3}\text{-sesK}\text{-compr} \equiv \{s. \forall K\ KK.$

$KK \subseteq range\ sesK \longrightarrow$

$(Key\ K \in analz\ (Key\ KK \cup (IK\ s))) = (K \in KK \vee Key\ K \in analz\ (IK\ s))$

$\}$

**lemmas**  $m3\text{-inv3}\text{-sesK}\text{-compr}I = m3\text{-inv3}\text{-sesK}\text{-compr}\text{-def}$  [THEN setc-def-to-intro, rule-format]

**lemmas**  $m3\text{-inv3}\text{-sesK}\text{-compr}E = m3\text{-inv3}\text{-sesK}\text{-compr}\text{-def}$  [THEN setc-def-to-elim, rule-format]

**lemmas**  $m3\text{-inv3}\text{-sesK}\text{-compr}D = m3\text{-inv3}\text{-sesK}\text{-compr}\text{-def}$  [THEN setc-def-to-dest, rule-format]

Additional lemma

**lemmas**  $insert\text{-commute}\text{-Key} = insert\text{-commute}$  [where  $x=Key\ K$  for  $K$ ]

**lemmas**  $m3\text{-inv3}\text{-sesK}\text{-compr}\text{-simps} =$

$m3\text{-inv3}\text{-sesK}\text{-compr}D$

$m3\text{-inv3}\text{-sesK}\text{-compr}D$  [where  $KK=\{Kab\}$  for  $Kab$ , simplified]

$m3\text{-inv3}\text{-sesK}\text{-compr}D$  [where  $KK=insert\ Kab\ KK$  for  $Kab\ KK$ , simplified]

$insert\text{-commute}\text{-Key}$  — to get the keys to the front

Invariance proof.

**lemma** *PO-m3-inv3-sesK-compr-step4*:

$\{m3\text{-inv3}\text{-sesK}\text{-compr} \cap m3\text{-inv2}\text{-ticket} \cap m3\text{-inv1}\text{-lkeysec}\}$

$m3\text{-step4}\ Ra\ A\ B\ Kab\ Ts\ X$

$\{> m3\text{-inv3}\text{-sesK}\text{-compr}\}$

$\langle proof \rangle$

All together now.

**lemmas**  $PO\text{-}m3\text{-inv3}\text{-sesK}\text{-compr}\text{-trans}\text{-lemmas} =$

$PO\text{-}m3\text{-inv3}\text{-sesK}\text{-compr}\text{-step4}$

**lemma** *PO-m3-inv3-sesK-compr-init* [iff]:

$init\ m3 \subseteq m3\text{-inv3}\text{-sesK}\text{-compr}$

$\langle proof \rangle$

**lemma** *PO-m3-inv3-sesK-compr-trans* [iff]:  
 $\{m3\text{-inv3-sesK-compr} \cap m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec}\}$   
*trans*  $m3$   
 $\{> m3\text{-inv3-sesK-compr}\}$   
 ⟨proof⟩

**lemma** *PO-m3-inv3-sesK-compr* [iff]: *reach*  $m3 \subseteq m3\text{-inv3-sesK-compr}$   
 ⟨proof⟩

### 3.16.6 Refinement

#### Message abstraction and simulation relation

Abstraction function on sets of messages.

##### inductive-set

*abs-msg* :: *msg set*  $\Rightarrow$  *chmsg set*

**for** *H* :: *msg set*

##### where

*am-M1*:

$\{\text{Agent } A, \text{Agent } B\} \in H$

$\Longrightarrow \text{Insec } A \ B \ (\text{Msg } []) \in \text{abs-msg } H$

| *am-M2a*:

$\text{Crypt } (\text{shrK } C) \ \{\text{Key } K, \text{Agent } B, \text{Number } T, X\} \in H$

$\Longrightarrow \text{Secure } \text{Sv } C \ (\text{Msg } [\text{aAgt } B, \text{aKey } K, \text{aNum } T]) \in \text{abs-msg } H$

| *am-M2b*:

$\text{Crypt } (\text{shrK } C) \ \{\text{Key } K, \text{Agent } A, \text{Number } T\} \in H$

$\Longrightarrow \text{Secure } \text{Sv } C \ (\text{Msg } [\text{aKey } K, \text{aAgt } A, \text{aNum } T]) \in \text{abs-msg } H$

R23: The simulation relation. This is a data refinement of the insecure and secure channels of refinement 2.

##### definition

*R23-msgs* :: (*m2-state*  $\times$  *m3-state*) *set* **where**

$R23\text{-msgs} \equiv \{(s, t). \text{abs-msg } (\text{parts } (IK \ t)) \subseteq \text{chan } s\}$

##### definition

*R23-keys* :: (*m2-state*  $\times$  *m3-state*) *set* **where**

$R23\text{-keys} \equiv \{(s, t). \forall KK \ K. KK \subseteq \text{range } \text{sesK} \longrightarrow$

$\text{Key } K \in \text{analz } (\text{Key}'KK \cup (IK \ t)) \longrightarrow \text{aKey } K \in \text{extr } (\text{aKey}'KK \cup ik0) (\text{chan } s)$

$\}$

##### definition

*R23-pres* :: (*m2-state*  $\times$  *m3-state*) *set* **where**

$R23\text{-pres} \equiv \{(s, t). \text{runs } s = \text{runs } t \wedge \text{clk } s = \text{clk } t \wedge \text{leak } s = \text{leak } t\}$

##### definition

*R23* :: (*m2-state*  $\times$  *m3-state*) *set* **where**

$R23 \equiv R23\text{-msgs} \cap R23\text{-keys} \cap R23\text{-pres}$

**lemmas** *R23-defs* =

*R23-def* *R23-msgs-def* *R23-keys-def* *R23-pres-def*

The mediator function is the identity here.

**definition**

$med32 :: m3\text{-obs} \Rightarrow m2\text{-obs}$  **where**  
 $med32 \equiv id$

**lemmas**  $R23\text{-msgsI} = R23\text{-msgs-def}$  [*THEN rel-def-to-intro, simplified, rule-format*]

**lemmas**  $R23\text{-msgsE}$  [*elim*] =  $R23\text{-msgs-def}$  [*THEN rel-def-to-elim, simplified, rule-format*]

**lemmas**  $R23\text{-keysI} = R23\text{-keys-def}$  [*THEN rel-def-to-intro, simplified, rule-format*]

**lemmas**  $R23\text{-keysE}$  [*elim*] =  $R23\text{-keys-def}$  [*THEN rel-def-to-elim, simplified, rule-format*]

**lemmas**  $R23\text{-presI} = R23\text{-pres-def}$  [*THEN rel-def-to-intro, simplified, rule-format*]

**lemmas**  $R23\text{-presE}$  [*elim*] =  $R23\text{-pres-def}$  [*THEN rel-def-to-elim, simplified, rule-format*]

**lemmas**  $R23\text{-intros} = R23\text{-msgsI } R23\text{-keysI } R23\text{-presI}$

Lemmas for various instantiations (for keys).

**lemmas**  $R23\text{-keys-dest} = R23\text{-keys-def}$  [*THEN rel-def-to-dest, simplified, rule-format, rotated 2*]

**lemmas**  $R23\text{-keys-dests} =$

$R23\text{-keys-dest}$

$R23\text{-keys-dest}$  [**where**  $KK = \{\}$ , *simplified*]

$R23\text{-keys-dest}$  [**where**  $KK = \{K'\}$  **for**  $K'$ , *simplified*]

$R23\text{-keys-dest}$  [**where**  $KK = \text{insert } K' \text{ } KK$  **for**  $K' \text{ } KK$ , *simplified, OF - - conjI*]

**General lemmas**

General facts about  $abs\text{-msg}$

**declare**  $abs\text{-msg.intros}$  [*intro!*]

**declare**  $abs\text{-msg.cases}$  [*elim!*]

**lemma**  $abs\text{-msg-empty}$ :  $abs\text{-msg } \{\} = \{\}$

*<proof>*

**lemma**  $abs\text{-msg-Un}$  [*simp*]:

$abs\text{-msg } (G \cup H) = abs\text{-msg } G \cup abs\text{-msg } H$

*<proof>*

**lemma**  $abs\text{-msg-mono}$  [*elim*]:

$\llbracket m \in abs\text{-msg } G; G \subseteq H \rrbracket \Longrightarrow m \in abs\text{-msg } H$

*<proof>*

**lemma**  $abs\text{-msg-insert-mono}$  [*intro*]:

$\llbracket m \in abs\text{-msg } H \rrbracket \Longrightarrow m \in abs\text{-msg } (\text{insert } m' \text{ } H)$

*<proof>*

Facts about  $abs\text{-msg}$  concerning abstraction of fakeable messages. This is crucial for proving the refinement of the intruder event.

**lemma**  $abs\text{-msg-DY-subset-fakeable}$ :

$\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; (s, t) \in R23\text{-non}; t \in m3\text{-inv1-lkeysec} \rrbracket$

$\Longrightarrow abs\text{-msg } (\text{synth } (\text{analz } (IK \ t))) \subseteq fake \ ik0 \ (dom \ (runs \ s)) \ (chan \ s)$

*<proof>*

## Refinement proof

Pair decomposition. These were set to `elim!`, which is too aggressive here.

```
declare MPair-analz [rule del, elim]
declare MPair-parts [rule del, elim]
```

Protocol events.

```
lemma PO-m3-step1-refines-m2-step1:
  {R23}
  (m2-step1 Ra A B), (m3-step1 Ra A B)
  {> R23}
  <proof>
```

```
lemma PO-m3-step2-refines-m2-step2:
  {R23}
  (m2-step2 Rb A B), (m3-step2 Rb A B)
  {> R23}
  <proof>
```

```
lemma PO-m3-step3-refines-m2-step3:
  {R23  $\cap$  (m2-inv3a-sesK-compr)  $\times$  (m3-inv3-sesK-compr  $\cap$  m3-inv1-lkeysec)}
  (m2-step3 Rs A B Kab Ts), (m3-step3 Rs A B Kab Ts)
  {> R23}
  <proof>
```

```
lemma PO-m3-step4-refines-m2-step4:
  {R23  $\cap$ 
  UNIV  $\times$  (m3-inv3-sesK-compr  $\cap$  m3-inv2-ticket  $\cap$  m3-inv1-lkeysec)}
  (m2-step4 Ra A B Kab Ts), (m3-step4 Ra A B Kab Ts X)
  {> R23}
  <proof>
```

```
lemma PO-m3-step5-refines-m2-step5:
  {R23}
  (m2-step5 Rb A B Kab Ts), (m3-step5 Rb A B Kab Ts)
  {> R23}
  <proof>
```

```
lemma PO-m3-tick-refines-m2-tick:
  {R23}
  (m2-tick T), (m3-tick T)
  {>R23}
  <proof>
```

Intruder events.

```
lemma PO-m3-leak-refines-m2-leak:
  {R23}
  (m2-leak Rs), (m3-leak Rs)
  {>R23}
  <proof>
```

```
lemma PO-m3-DY-fake-refines-m2-fake:
```

$\{R23 \cap UNIV \times (m3\text{-inv1-lkeysec})\}$   
 $m2\text{-fake}, m3\text{-DY-fake}$   
 $\{> R23\}$   
 $\langle\text{proof}\rangle$

All together now...

**lemmas**  $PO\text{-}m3\text{-trans-refines-}m2\text{-trans} =$   
 $PO\text{-}m3\text{-step1-refines-}m2\text{-step1}$   $PO\text{-}m3\text{-step2-refines-}m2\text{-step2}$   
 $PO\text{-}m3\text{-step3-refines-}m2\text{-step3}$   $PO\text{-}m3\text{-step4-refines-}m2\text{-step4}$   
 $PO\text{-}m3\text{-step5-refines-}m2\text{-step5}$   $PO\text{-}m3\text{-tick-refines-}m2\text{-tick}$   
 $PO\text{-}m3\text{-leak-refines-}m2\text{-leak}$   $PO\text{-}m3\text{-DY-fake-refines-}m2\text{-fake}$

**lemma**  $PO\text{-}m3\text{-refines-init-}m2$  [*iff*]:  
 $init\ m3 \subseteq R23''(init\ m2)$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m3\text{-refines-trans-}m2$  [*iff*]:  
 $\{R23 \cap$   
 $(m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv3-sesK-compr} \cap m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec})\}$   
 $(trans\ m2), (trans\ m3)$   
 $\{> R23\}$   
 $\langle\text{proof}\rangle$

**lemma**  $PO\text{-}m3\text{-observation-consistent}$  [*iff*]:  
 $obs\text{-consistent}\ R23\ med32\ m2\ m3$   
 $\langle\text{proof}\rangle$

Refinement result.

**lemma**  $m3\text{-refines-}m2$  [*iff*]:  
 $refines$   
 $(R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv3-sesK-compr} \cap m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec}))$   
 $med32\ m2\ m3$   
 $\langle\text{proof}\rangle$

**lemma**  $m3\text{-implements-}m2$  [*iff*]:  
 $implements\ med32\ m2\ m3$   
 $\langle\text{proof}\rangle$

**end**