

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

1	Protocol Modeling and Refinement Infrastructure	5
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	20
1.3.1	Agents	21
1.3.2	Nonces	21
1.4	Symmetric and Assymmetric Keys	21
1.4.1	Asymmetric Keys	23
1.4.2	Basic properties of <i>pubK</i> and <i>priK</i>	23
1.4.3	"Image" equations that hold for injective functions	23
1.4.4	Symmetric Keys	24
1.5	Atomic messages	24
1.5.1	Atoms datatype	24
1.5.2	Long-term key setup (abstractly)	25
1.6	Protocol runs	25
1.6.1	Runs	25
1.6.2	Run abstraction	26
1.7	Channel Messages	26
1.7.1	Channel messages	26
1.7.2	Keys used in dynamic channel messages	27
1.7.3	Atoms in a set of channel messages	28
1.7.4	Intruder knowledge (atoms)	28
1.7.5	Faking messages	31
1.8	Theory of Agents and Messages for Security Protocols	33
1.8.1	keysFor operator	34
1.8.2	Inductive relation "parts"	35
1.8.3	Inductive relation "analz"	38
1.8.4	Inductive relation "synth"	42
1.8.5	HPair: a combination of Hash and MPair	45
1.9	Secrecy with Leaking (global version)	48

1.9.1	State	48
1.9.2	Invariant definitions	49
1.9.3	Events	49
1.9.4	Specification	50
1.9.5	Invariant proofs	51
1.9.6	inv1: Secrecy	51
1.9.7	inv2: Authorized and leaked data is known to someone	51
1.10	Non-injective Agreement	51
1.10.1	State	52
1.10.2	Events	52
1.10.3	Invariants	54
1.10.4	inv1: non-injective agreement	54
1.11	Injective Agreement	54
1.11.1	State	55
1.11.2	Events	55
1.11.3	Invariants	56
1.11.4	Refinement	57
1.11.5	Derived invariants	58
2	Unidirectional Authentication Protocols	59
2.1	Refinement 1: Abstract Protocol	59
2.1.1	State	59
2.1.2	Events	59
2.1.3	Simulation relation	61
2.1.4	Refinement	62
2.2	Refinement 2a: Authentic Channel Protocol	63
2.2.1	State	63
2.2.2	Events	64
2.2.3	Invariants	65
2.2.4	Refinement	66
2.3	Refinement 2b: Confidential Channel Protocol	67
2.3.1	State and observations	67
2.3.2	Events	68
2.3.3	Invariants	70
2.3.4	Refinement	72
2.4	Refinement 3a: Signature-based Dolev-Yao Protocol (Variant A)	73
2.4.1	State	73
2.4.2	Events	73
2.4.3	Invariants	75
2.4.4	Refinement	77
2.5	Refinement 3b: Encryption-based Dolev-Yao Protocol (Variant A)	81
2.5.1	State and observations	81
2.5.2	Events	81
2.5.3	Invariants	83
2.5.4	Simulation relation	84
2.5.5	Misc lemmas	85
2.5.6	Refinement proof	86

3	Key Establishment Protocols	88
3.1	Basic abstract key distribution (L1)	88
3.1.1	State	88
3.1.2	Events	92
3.1.3	Specification	93
3.1.4	Invariants	94
3.1.5	Refinement of $s0g$	95
3.1.6	Derived invariants	96
3.2	Abstract (i/n)-authenticated key transport (L1)	97
3.2.1	State	97
3.2.2	Events	98
3.2.3	Specification	99
3.2.4	Invariants	100
3.2.5	Refinement of $m1x$	100
3.2.6	Refinement of $a0i$ for initiator/server	103
3.2.7	Refinement of $a0n$ for responder/server	108
3.3	Abstract (n/n)-authenticated key transport (L1)	112
3.3.1	State	112
3.3.2	Events	112
3.3.3	Specification	114
3.3.4	Invariants	114
3.3.5	Refinement of $m1x$	115
3.3.6	Refinement of $a0n$ for initiator/server	117
3.3.7	Refinement of $a0n$ for responder/server	120
3.4	Abstract Kerberos core protocol (L1)	123
3.4.1	State	124
3.4.2	Events	124
3.4.3	Specification	127
3.4.4	Invariants	128
3.4.5	Refinement of $m1a$	129
3.4.6	Refinement of $a0i$ for responder/initiator	133
3.4.7	Refinement of $a0i$ for initiator/responder	140
3.5	Abstract Kerberos core protocol (L2)	144
3.5.1	State	145
3.5.2	Events	145
3.5.3	Transition system	148
3.5.4	Invariants and simulation relation	149
3.5.5	Refinement	159
3.5.6	Inherited and derived invariants	161
3.6	Core Kerberos, "parallel" variant (L3)	162
3.6.1	Setup	162
3.6.2	State	163
3.6.3	Events	163
3.6.4	Transition system	166
3.6.5	Invariants	167
3.6.6	Refinement	168
3.6.7	Inherited invariants	174

3.7	Core Kerberos 5 (L3)	174
3.7.1	Setup	175
3.7.2	State	175
3.7.3	Events	175
3.7.4	Transition system	178
3.7.5	Invariants	179
3.7.6	Refinement	181
3.7.7	Inherited invariants	187
3.8	Core Kerberos 4 (L3)	187
3.8.1	Setup	188
3.8.2	State	188
3.8.3	Events	188
3.8.4	Transition system	191
3.8.5	Invariants	192
3.8.6	Refinement	197
3.8.7	Inherited invariants	204
3.9	Abstract Needham-Schroeder Shared Key (L1)	205
3.9.1	State	205
3.9.2	Events	205
3.9.3	Specification	207
3.9.4	Invariants	208
3.9.5	Refinement of $m1a$	209
3.9.6	Refinement of $a0i$ for initiator/responder	213
3.9.7	Refinement of $a0i$ for responder/initiator	217
3.10	Abstract Needham-Schroeder Shared Key (L2)	223
3.10.1	State	223
3.10.2	Events	224
3.10.3	Transition system	226
3.10.4	Invariants	227
3.10.5	Refinement	240
3.10.6	Inherited and derived invariants	243
3.11	Needham-Schroeder Shared Key, "parallel" variant (L3)	243
3.11.1	Setup	244
3.11.2	State	244
3.11.3	Events	245
3.11.4	Transition system	247
3.11.5	Invariants	248
3.11.6	Refinement	250
3.11.7	Inherited invariants	255
3.12	Needham-Schroeder Shared Key (L3)	256
3.12.1	Setup	256
3.12.2	State	256
3.12.3	Events	257
3.12.4	Transition system	260
3.12.5	Invariants	260
3.12.6	Refinement	264
3.12.7	Inherited invariants	270

3.13	Abstract Denning-Sacco protocol (L1)	271
3.13.1	State	272
3.13.2	Events	272
3.13.3	Specification	273
3.13.4	Invariants	274
3.13.5	Refinement of $m1a$	275
3.14	Abstract Denning-Sacco protocol (L2)	276
3.14.1	State	277
3.14.2	Events	277
3.14.3	Transition system	279
3.14.4	Invariants and simulation relation	280
3.14.5	Refinement	286
3.14.6	Inherited and derived invariants	287
3.15	Denning-Sacco, direct variant (L3)	288
3.15.1	Setup	288
3.15.2	State	288
3.15.3	Events	288
3.15.4	Transition system	291
3.15.5	Invariants	291
3.15.6	Refinement	293
3.16	Denning-Sacco protocol (L3)	298
3.16.1	Setup	298
3.16.2	State	298
3.16.3	Events	298
3.16.4	Transition system	301
3.16.5	Invariants	301
3.16.6	Refinement	305

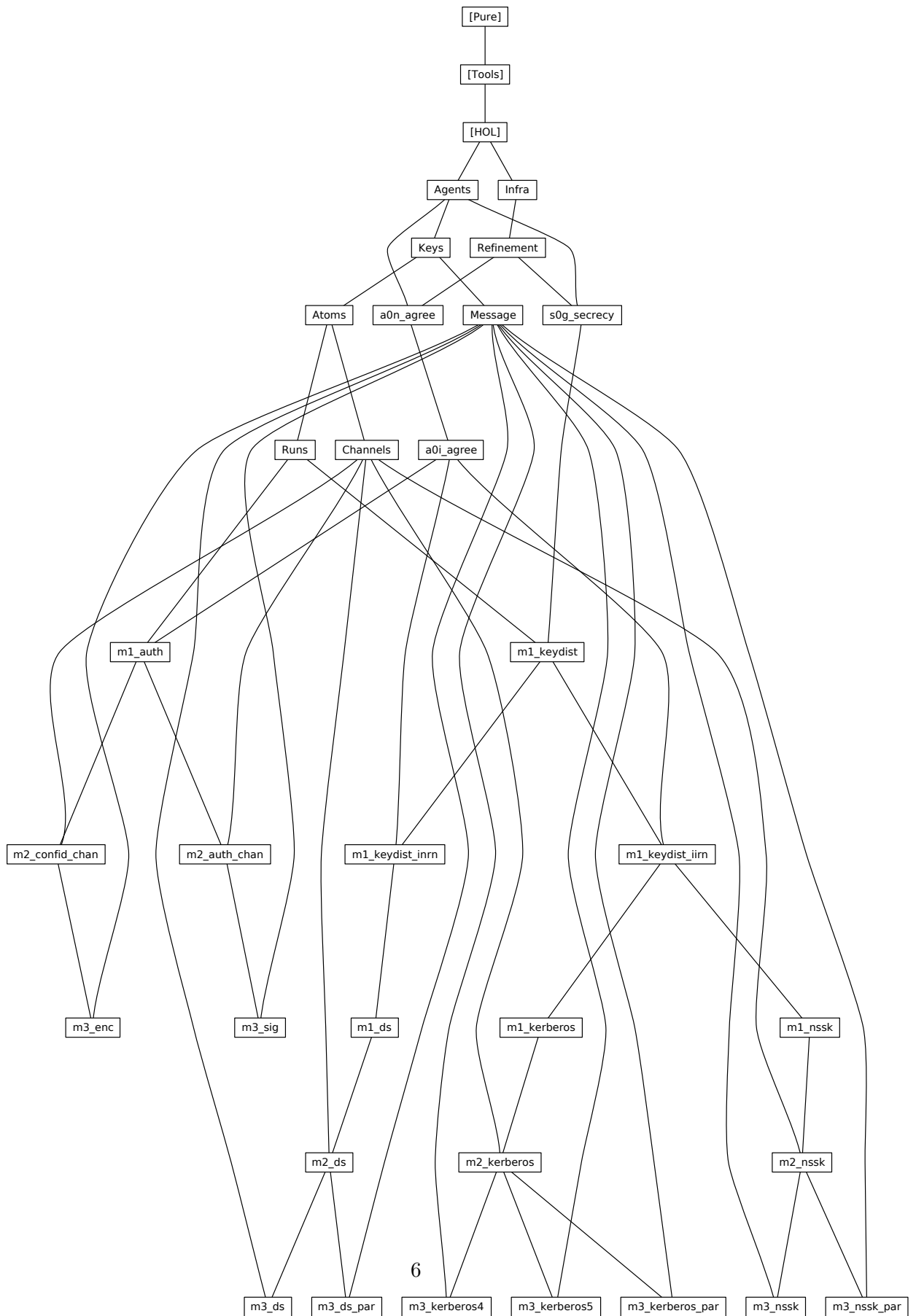


Figure 1: Theory dependencies

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*:

$$f\ x = \text{Some } y \implies x \in \text{dom } f$$

by (*simp add: domIff*)

lemma *map-definedness-contra*:

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

by (*auto simp add: domIff*)

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

Set

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

by (*auto simp add: image-def vimage-def*)

Relations

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

$$(R1\ O\ R2)^{\llcorner A} = R2^{\llcorner (R1^{\llcorner A})}$$

by (*auto*)

Lists

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

by (*simp*)

— Do NOT add the following equation to the simpset! (looping)

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

by (*simp*)

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

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

by (*induct l arbitrary: n xs, auto*)

(*case-tac n, auto*)

Finite sets

Cardinality.

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

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

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

by (*auto*)

lemma *finite-positive-cardD* [*dest!*]:
 $\llbracket 0 < \text{card } A; \text{finite } A \rrbracket \implies A \neq \{\}$
by (*auto*)

lemma *finite-zero-cardI* [*intro!*]:
 $\llbracket A = \{\}; \text{finite } A \rrbracket \implies \text{card } A = 0$
by (*auto*)

lemma *finite-zero-cardD* [*dest!*]:
 $\llbracket \text{card } A = 0; \text{finite } A \rrbracket \implies A = \{\}$
by (*auto*)

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*:

$s \# b \in \text{beh } T \implies b \in \text{beh } T$
by (*erule beh-non-empty, auto*)

lemma *beh-prefix-closed*:
 $c @ b \in \text{beh } T \implies b \in \text{beh } T$
by (*induct c, auto dest!: beh-immediate-prefix-closed*)

States in behaviours are exactly reachable.

lemma *beh-in-reach* [*rule-format*]:
 $b \in \text{beh } T \implies (\forall s \in \text{set } b. s \in \text{reach } T)$
by (*erule beh.induct*) (*auto*)

lemma *reach-in-beh*:
assumes $s \in \text{reach } T$ **shows** $\exists b \in \text{beh } T. s \in \text{set } b$
using *assms*
proof (*induction s rule: reach.induct*)
case (*r-init s*)
hence $s \in \text{set } [s]$ **and** $[s] \in \text{beh } T$ **by** *auto*
thus *?case by fastforce*
next
case (*r-trans s t*)
then obtain b **where** $b \in \text{beh } T$ **and** $s \in \text{set } b$ **by** *blast*
from $\langle s \in \text{set } b \rangle$ **obtain** $b1\ b2$ **where** $b = b2 @ s \# b1$ **by** (*blast dest: split-list*)
with $\langle b \in \text{beh } T \rangle$ **have** $s \# b1 \in \text{beh } T$ **by** (*blast intro: beh-prefix-closed*)
with $\langle (s, t) \in \text{trans } T \rangle$ **have** $t \# s \# b1 \in \text{beh } T$ **by** *blast*
thus *?case by force*
qed

lemma *reach-equiv-beh-states*: $\text{reach } T = \bigcup (\text{set}(\text{beh } T))$
by (*auto intro!: reach-in-beh beh-in-reach*)

Specifications, observability, and implementation

Specifications add an observer function to transition systems.

record ($'s, 'o$) *spec* = $'s\ TS +$
 $obs :: 's \Rightarrow 'o$

lemma *beh-obs-upd* [*simp*]: $\text{beh } (S(|\ \text{obs} := x\ |)) = \text{beh } S$
by (*safe*) (*erule beh.induct, auto*)+

lemma *reach-obs-upd* [*simp*]: $\text{reach } (S(|\ \text{obs} := x\ |)) = \text{reach } S$
by (*safe*) (*erule reach.induct, auto*)+

Observable behaviour and reachability.

definition
 $obeh :: ('s, 'o)\ \text{spec} \Rightarrow ('o\ \text{list})\ \text{set}$ **where**
 $obeh\ S \equiv (\text{map } (obs\ S))'(\text{beh } S)$

definition
 $oreach :: ('s, 'o)\ \text{spec} \Rightarrow 'o\ \text{set}$ **where**
 $oreach\ S \equiv (obs\ S)'(\text{reach } S)$

lemma *oreach-equiv-obeh-states*:

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

by (*auto simp add: reach-equiv-beh-states oreach-def obeh-def*)

lemma *obeh-pi-translation*:

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

by (*auto simp add: obeh-def image-comp*)

lemma *oreach-pi-translation*:

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

by (*auto simp add: oreach-def*)

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

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

where

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

definition

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

where

$$\text{observable2 } \text{ob } P \equiv \exists Q. P = \text{ob}'Q$$

definition

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

where

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

lemma *observableE [elim]*:

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

by (*unfold observable-def*) (*fast*)

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

by (*unfold observable-def observable2-def*) (*auto*)

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

by (*unfold observable3-def observable2-def*) (*auto*)

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

by (*simp add: observable-def*)

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*: $(\text{ob}'P \subseteq Q) = (P \subseteq \text{ob}'Q)$

by *auto*

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*: $\llbracket Q \subseteq \text{ob}'P; \text{observable } \text{ob } P \rrbracket \implies \text{ob}'Q \subseteq P$
by (*auto*)

lemma *vimage-l-image-r*: $\llbracket \text{ob}'Q \subseteq P; Q \subseteq \text{range } \text{ob} \rrbracket \implies Q \subseteq \text{ob}'P$
by (*drule image-mono [where f=ob], auto*)

Internal and external invariants

lemma *external-from-internal-invariant*:

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

by (*auto simp add: oreach-def*)

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

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

by (*erule external-from-internal-invariant*) (*auto*)

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

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

by (*auto simp add: oreach-def*)

lemma *external-to-internal-invariant*:

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

by (*erule external-to-internal-invariant-vimage*) (*auto*)

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

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

by (*fast intro: external-from-internal-invariant-vimage*
external-to-internal-invariant-vimage
del: subsetI)

lemma *external-equiv-internal-invariant*:

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

by (*rule external-equiv-internal-invariant-vimage*) (*auto*)

Our notion of implementation is inclusion of observable behaviours.

definition

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

Reflexivity and transitivity

lemma *implements-refl*: *implements* $\text{id } S S$

by (*auto simp add: implements-def*)

lemma *implements-trans*:

$\llbracket \text{implements } \pi_1 S_1 S_2; \text{implements } \pi_2 S_2 S_3 \rrbracket$

$\implies \text{implements } (pi1 \ o \ pi2) \ S1 \ S3$
by (*fastforce simp add: implements-def image-subset-iff*)

Preservation of external invariants

lemma *implements-oreach*:
 $\text{implements } pi \ Sa \ Sc \implies pi'(oreach \ Sc) \subseteq oreach \ Sa$
by (*auto simp add: implements-def oreach-equiv-obeh-states dest!: subsetD*)

lemma *external-invariant-preservation*:
 $\llbracket oreach \ Sa \subseteq Q; \text{implements } pi \ Sa \ Sc \rrbracket$
 $\implies pi'(oreach \ Sc) \subseteq Q$
by (*rule subset-trans [OF implements-oreach]*) (*auto*)

lemma *external-invariant-translation*:
 $\llbracket oreach \ Sa \subseteq Q; pi-'Q \subseteq P; \text{implements } pi \ Sa \ Sc \rrbracket$
 $\implies oreach \ Sc \subseteq P$
apply (*rule subset-trans [OF vimage-image-subset, of pi]*)
apply (*rule subset-trans [where B=pi-'Q]*)
apply (*intro vimage-mono external-invariant-preservation, auto*)
done

Preservation of internal invariants

lemma *internal-invariant-translation*:
 $\llbracket reach \ Sa \subseteq Pa; Pa \subseteq obs \ Sa \ -' \ Qa; pi \ -' \ Qa \subseteq Q; obs \ S \ -' \ Q \subseteq P; \text{implements } pi \ Sa \ S \rrbracket$
 $\implies reach \ S \subseteq P$
by (*rule external-from-internal-invariant-vimage [*
THEN external-invariant-translation,
THEN external-to-internal-invariant-vimage])

1.2.2 Invariants

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

Hoare triples

definition
 $PO\text{-hoare} :: ['s \ set, ('s \times 's) \ set, 's \ set] \Rightarrow \text{bool}$
 $(\langle (3\{-} \ - \ \{> \ -\}) \ [0, 0, 0] \ 90)$

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

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

lemma $\{P\} \ R \ \{> \ Q\} = (\forall s \ t. s \in P \longrightarrow (s, t) \in R \longrightarrow t \in Q)$
by (*auto simp add: PO-hoare-defs*)

Some essential facts about Hoare triples.

lemma *hoare-conseq-left* [*intro*]:
 $\llbracket \{P'\} \ R \ \{> \ Q\}; P \subseteq P' \rrbracket$

$\implies \{P\} R \{> Q\}$
by (*auto simp add: PO-hoare-defs*)

lemma *hoare-conseq-right*:
 $\llbracket \{P\} R \{> Q'\}; Q' \subseteq Q \rrbracket$
 $\implies \{P\} R \{> Q\}$
by (*auto simp add: PO-hoare-defs*)

lemma *hoare-false-left* [*simp*]:
 $\{\{\}\} R \{> Q\}$
by (*auto simp add: PO-hoare-defs*)

lemma *hoare-true-right* [*simp*]:
 $\{P\} R \{> UNIV\}$
by (*auto simp add: PO-hoare-defs*)

lemma *hoare-conj-right* [*intro!*]:
 $\llbracket \{P\} R \{> Q1\}; \{P\} R \{> Q2\} \rrbracket$
 $\implies \{P\} R \{> Q1 \cap Q2\}$
by (*auto simp add: PO-hoare-defs*)

Special transition relations.

lemma *hoare-stop* [*simp, intro!*]:
 $\{P\} \{\}\{> Q\}$
by (*auto simp add: PO-hoare-defs*)

lemma *hoare-skip* [*simp, intro!*]:
 $P \subseteq Q \implies \{P\} Id \{> Q\}$
by (*auto simp add: PO-hoare-defs*)

lemma *hoare-trans-Un* [*iff*]:
 $\{P\} R1 \cup R2 \{> Q\} = (\{P\} R1 \{> Q\} \wedge \{P\} R2 \{> Q\})$
by (*auto simp add: PO-hoare-defs*)

lemma *hoare-trans-UN* [*iff*]:
 $\{P\} \bigcup x. R x \{> Q\} = (\forall x. \{P\} R x \{> Q\})$
by (*auto simp add: PO-hoare-defs*)

Characterization of reachability

lemma *reach-init*: $reach\ T \subseteq I \implies init\ T \subseteq I$
by (*auto dest: subsetD*)

lemma *reach-trans*: $reach\ T \subseteq I \implies \{reach\ T\} trans\ T \{> I\}$
by (*auto simp add: PO-hoare-defs*)

Useful consequences.

corollary *init-reach* [*iff*]: $init\ T \subseteq reach\ T$
by (*rule reach-init, simp*)

corollary *trans-reach* [*iff*]: $\{reach\ T\} trans\ T \{> reach\ T\}$
by (*rule reach-trans, simp*)

Invariant proof rules

Basic proof rule for invariants.

lemma *inv-rule-basic*:

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

by (*safe, erule reach.induct, auto simp add: PO-hoare-def*)

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

lemma *inv-rule*:

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

apply (*rule subset-trans, auto*) — strengthen goal

apply (*erule reach.induct, auto simp add: PO-hoare-def*)

done

The following rule is equivalent to the previous one.

lemma *INV-rule*:

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

by (*safe, erule reach.induct, auto simp add: PO-hoare-defs*)

Proof of equivalence.

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

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

apply (*rule subset-trans, auto del: subsetI*)

apply (*rule INV-rule, auto*)

done

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

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

by (*rule-tac I=I \cap reach T in inv-rule, auto*)

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

lemma *inv-rule-incr*:

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

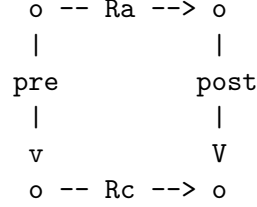
by (*rule INV-rule, auto*)

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:



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: $(O) \equiv \lambda r s. \{(xa, x). ((\lambda x xa. (x, xa) \in r) OO (\lambda x xa. (x, xa) \in s))\}$.)

definition

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

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*]:

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

by (*auto simp add: PO-rhoare-defs dest!: subsetD*)

lemma *relhoare-conseq-right*:

— do NOT declare [*intro*]

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

by (*auto simp add: PO-rhoare-defs*)

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

— do NOT declare [*intro*]

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

by (*auto simp add: PO-rhoare-defs*)

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

— not true in general

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

by (*auto simp add: PO-rhoare-defs*)

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

$\text{Domain } pre \subseteq R \implies \text{Domain } (pre \ O \ Rc) \subseteq R$

by (*auto simp add: Domain-def*)

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

by (*auto simp add: PO-rhoare-def*)

Reflexivity and transitivity.

lemma *relhoare-refl* [*simp*]: $\{Id\} R, R \{> Id\}$
by (*auto simp add: PO-rhoare-defs*)

lemma *rhoare-trans*:

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

apply (*auto simp add: PO-rhoare-def del: subsetI*)

apply (*drule subset-refl [THEN relcomp-mono, where r=R1]*)

apply (*drule subset-refl [THEN [2] relcomp-mono, where s=R2]*)

apply (*auto simp add: O-assoc del: subsetI*)

done

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!*
 $\implies \{pre\} Ra, Rc \{> post1 \ \cap \ post2\}$

by (*auto simp add: PO-rhoare-defs dest: single-valuedD dest!: subsetD*)

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

$\llbracket \{Domain \ pre\} Ra \{> I\}; \{Range \ pre\} Rc \{> J\};$
 $\{pre\} Ra, Rc \{> post\} \rrbracket$
 $\implies \{pre\} Ra, Rc \{> post \ \cap \ I \ \times \ J\}$

by (*force simp add: PO-rhoare-defs PO-hoare-defs Domain-def Range-def*)

Separate rule for cartesian products.

corollary *relhoare-cartesian*:

$\llbracket \{Domain \ pre\} Ra \{> I\}; \{Range \ pre\} Rc \{> J\};$
 $\{pre\} Ra, Rc \{> post\} \rrbracket$ — any post, including *UNIV!*
 $\implies \{pre\} Ra, Rc \{> I \ \times \ J\}$

by (*auto intro: relhoare-conseq-right*)

Unions of transition relations.

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

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

apply (*auto simp add: PO-rhoare-defs*)

apply (*auto dest!: subsetD*)

done

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

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

apply (*auto simp add: PO-rhoare-defs*)

apply (*auto dest!: subsetD*)

done

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

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

by (*auto simp add: PO-rhoare-defs*)

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

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

by (*auto simp add: PO-rhoare-defs*)

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

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

apply (*auto simp add: PO-rhoare-defs*)

apply (*auto dest!: subsetD*)

done

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-refines ::

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

where

$PO\ refines\ R\ Ta\ Tc \equiv (\$
 $\quad \mathit{init}\ Tc \subseteq R \mathit{``}(\mathit{init}\ Ta)$
 $\quad \wedge \{R\} (\mathit{trans}\ Ta), (\mathit{trans}\ Tc) \{> R\}$
 $\quad \left. \right)$

lemma *PO-refinesI*:

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

by (*simp add: PO-refines-def*)

lemma *PO-refinesE* [elim]:

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

by (*simp add: PO-refines-def*)

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

lemma *refine-basic*:

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

by (*simp add: PO-refines-def*)

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 *Domain* ($R \cap UNIV \times J$) and *Range* ($R \cap I \times UNIV$), 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*:

$\llbracket \text{init } Tc \subseteq R \text{ ``}(\text{init } Ta); \text{init } Ta \subseteq I; \text{init } Tc \subseteq J \rrbracket$
 $\implies \text{init } Tc \subseteq (R \cap I \times J) \text{ ``}(\text{init } Ta)$
by (*auto simp add: Image-def dest!: bspec subsetD*)

lemma *refine-trans-using-invariants*:
 $\llbracket \{R \cap I \times J\} (\text{trans } Ta), (\text{trans } Tc) \{> R\};$
 $\{I \cap \text{Domain } (R \cap \text{UNIV} \times J)\} (\text{trans } Ta) \{> I\};$
 $\{J \cap \text{Range } (R \cap I \times \text{UNIV})\} (\text{trans } Tc) \{> J\} \rrbracket$
 $\implies \{R \cap I \times J\} (\text{trans } Ta), (\text{trans } Tc) \{> R \cap I \times J\}$
by (*rule relhoare-conj-right-cartesian*) (*auto*)

This is our main rule for refinements.

lemma *refine-using-invariants*:
 $\llbracket \{R \cap I \times J\} (\text{trans } Ta), (\text{trans } Tc) \{> R\};$
 $\{I \cap \text{Domain } (R \cap \text{UNIV} \times J)\} (\text{trans } Ta) \{> I\};$
 $\{J \cap \text{Range } (R \cap I \times \text{UNIV})\} (\text{trans } Tc) \{> J\};$
 $\text{init } Tc \subseteq R \text{ ``}(\text{init } Ta);$
 $\text{init } Ta \subseteq I; \text{init } Tc \subseteq J \rrbracket$
 $\implies \text{PO-refines } (R \cap I \times J) \text{ } Ta \text{ } Tc$
by (*unfold PO-refines-def*)
(intro refine-init-using-invariants refine-trans-using-invariants conjI)

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*:
 $\text{PO-refines } R \text{ } Ta \text{ } Tc \implies \text{init } Tc \subseteq \text{Range } R$
by (*auto simp add: PO-refines-def*)

lemma *PO-refines-implies-Range-trans*:
 $\text{PO-refines } R \text{ } Ta \text{ } Tc \implies \{\text{Range } R\} \text{trans } Tc \{> \text{Range } R\}$
by (*auto simp add: PO-refines-def PO-rhoare-def PO-hoare-def*)

lemma *PO-refines-implies-Range-invariant*:
 $\text{PO-refines } R \text{ } Ta \text{ } Tc \implies \text{reach } Tc \subseteq \text{Range } R$
by (*rule INV-rule*)
(auto intro!: PO-refines-implies-Range-init
PO-refines-implies-Range-trans)

The following rules are more useful in proofs.

corollary *INV-init-from-refinement*:
 $\llbracket \text{PO-refines } R \text{ } Ta \text{ } Tc; \text{Range } R \subseteq I \rrbracket$
 $\implies \text{init } Tc \subseteq I$
by (*drule PO-refines-implies-Range-init, auto*)

corollary *INV-trans-from-refinement*:
 $\llbracket \text{PO-refines } R \text{ } Ta \text{ } Tc; K \subseteq \text{Range } R; \text{Range } R \subseteq I \rrbracket$
 $\implies \{K\} \text{trans } Tc \{> I\}$
apply (*drule PO-refines-implies-Range-trans*)

apply (*auto intro: hoare-conseq-right*)
done

corollary *INV-from-refinement*:

$\llbracket PO\text{-refines } R \text{ } Ta \text{ } Tc; \text{ Range } R \subseteq I \rrbracket$
 $\implies \text{reach } Tc \subseteq I$

by (*drule PO-refines-implies-Range-invariant, fast*)

Refinement of specifications

Lift relation membership to finite sequences

inductive-set

seq-lift :: ('s × 't) set ⇒ ('s list × 't list) set
for *R* :: ('s × 't) set

where

sl-nil [*iff*]: ([], []) ∈ *seq-lift R*

| *sl-cons* [*intro*]:

$\llbracket (xs, ys) \in \text{seq-lift } R; (x, y) \in R \rrbracket \implies (x\#xs, y\#ys) \in \text{seq-lift } R$

inductive-cases *sl-cons-right-invert*: (*ba'*, *t* # *bc*) ∈ *seq-lift R*

For each concrete behaviour there is a related abstract one.

lemma *behaviour-refinement*:

$\llbracket PO\text{-refines } R \text{ } Ta \text{ } Tc; bc \in \text{beh } Tc \rrbracket$
 $\implies \exists ba \in \text{beh } Ta. (ba, bc) \in \text{seq-lift } R$

apply (*erule beh.induct, auto*)

— case: singleton

apply (*clarsimp simp add: PO-refines-def Image-def*)

apply (*drule subsetD, auto*)

— case: cons; first construct related abstract state

apply (*erule sl-cons-right-invert, clarsimp*)

apply (*rename-tac s bc s' ba t*)

— now construct abstract transition

apply (*auto simp add: PO-refines-def PO-rhoare-def*)

apply (*thin-tac X ⊆ Y for X Y*)

apply (*drule subsetD, auto*)

done

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-consistent ::

$[('s \times 't) \text{ set}, 'p \Rightarrow 'o, ('s, 'o) \text{ spec}, ('t, 'p) \text{ spec}] \Rightarrow \text{bool}$

where

$\text{obs-consistent } R \text{ } pi \text{ } Sa \text{ } Sc \equiv (\forall s \ t. (s, t) \in R \longrightarrow pi (obs \ Sc \ t) = obs \ Sa \ s)$

lemma *obs-consistent-refl* [*iff*]: *obs-consistent Id id S S*

by (*simp add: obs-consistent-def*)

lemma *obs-consistent-trans* [intro]:
 $\llbracket \text{obs-consistent } R1 \text{ } \pi1 \text{ } S1 \text{ } S2; \text{obs-consistent } R2 \text{ } \pi2 \text{ } S2 \text{ } S3 \rrbracket$
 $\implies \text{obs-consistent } (R1 \text{ } O \text{ } R2) (\pi1 \text{ } o \text{ } \pi2) S1 \text{ } S3$
by (*auto simp add: obs-consistent-def*)

lemma *obs-consistent-empty*: *obs-consistent* {} *pi Sa Sc*
by (*auto simp add: obs-consistent-def*)

lemma *obs-consistent-conj1* [intro]:
 $\text{obs-consistent } R \text{ } \pi \text{ } Sa \text{ } Sc \implies \text{obs-consistent } (R \cap R') \text{ } \pi \text{ } Sa \text{ } Sc$
by (*auto simp add: obs-consistent-def*)

lemma *obs-consistent-conj2* [intro]:
 $\text{obs-consistent } R \text{ } \pi \text{ } Sa \text{ } Sc \implies \text{obs-consistent } (R' \cap R) \text{ } \pi \text{ } Sa \text{ } Sc$
by (*auto simp add: obs-consistent-def*)

lemma *obs-consistent-behaviours*:
 $\llbracket \text{obs-consistent } R \text{ } \pi \text{ } Sa \text{ } Sc; bc \in \text{beh } Sc; ba \in \text{beh } Sa; (ba, bc) \in \text{seq-lift } R \rrbracket$
 $\implies \text{map } \pi (\text{map } (\text{obs } Sc) bc) = \text{map } (\text{obs } Sa) ba$
by (*erule seq-lift.induct*) (*auto simp add: obs-consistent-def*)

Definition of refinement proof obligations.

definition

refines ::
 $[(\text{'s} \times \text{'t}) \text{ set}, \text{'p} \Rightarrow \text{'o}, (\text{'s}, \text{'o}) \text{ spec}, (\text{'t}, \text{'p}) \text{ spec}] \Rightarrow \text{bool}$

where

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

lemmas *refines-defs* =
refines-def PO-refines-def

lemma *refinesI*:
 $\llbracket \text{PO-refines } R \text{ } Sa \text{ } Sc; \text{obs-consistent } R \text{ } \pi \text{ } Sa \text{ } Sc \rrbracket$
 $\implies \text{refines } R \text{ } \pi \text{ } Sa \text{ } Sc$
by (*simp add: refines-def*)

lemma *refinesE* [elim]:
 $\llbracket \text{refines } R \text{ } \pi \text{ } Sa \text{ } Sc; \llbracket \text{PO-refines } R \text{ } Sa \text{ } Sc; \text{obs-consistent } R \text{ } \pi \text{ } Sa \text{ } Sc \rrbracket \implies P \rrbracket$
 $\implies P$
by (*simp add: refines-def*)

Reflexivity and transitivity of refinement.

lemma *refinement-reflexive*: *refines Id id S S*
by (*auto simp add: refines-defs*)

lemma *refinement-transitive*:
 $\llbracket \text{refines } R1 \text{ } \pi1 \text{ } S1 \text{ } S2; \text{refines } R2 \text{ } \pi2 \text{ } S2 \text{ } S3 \rrbracket$
 $\implies \text{refines } (R1 \text{ } O \text{ } R2) (\pi1 \text{ } o \text{ } \pi2) S1 \text{ } S3$
apply (*auto simp add: refines-defs del: subsetI*
intro: rhoare-trans)
apply (*fastforce dest: Image-mono*)
done

Soundness of refinement for proving implementation

lemma *observable-behaviour-refinement*:

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

by (*auto simp add: refines-def obeh-def image-def*
dest!: behaviour-refinement obs-consistent-behaviours)

theorem *refinement-soundness*:

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

by (*auto simp add: implements-def*
elim!: observable-behaviour-refinement)

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*:

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

by (*auto intro!: Refinement-using-invariants*)

Inheritance of internal invariants through refinements

lemma *INV-init-from-Refinement*:

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

by (*blast intro: INV-init-from-refinement*)

lemma *INV-trans-from-Refinement*:

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

by (*blast intro: INV-trans-from-refinement*)

lemma *INV-from-Refinement-basic*:

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

by (*rule INV-from-refinement*) *blast*

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 (*rule INV-from-Refinement-basic*)

show $\text{refines } (R \cap \text{reach } Sa \times \text{reach } Sc) \text{ pi } Sa \text{ Sc}$ **using** *assms(1)*

by (*rule refines-reachable-strengthening*)

next

show $\text{Range } (R \cap \text{reach } Sa \times \text{reach } Sc) \subseteq K$ **using** *assms(2-4)* **by** *blast*

qed

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.

agent = *Server* | *Agent nat*

consts

bad :: *agent set* — compromised agents

specification (*bad*)

Server-not-bad [iff]: *Server* \notin *bad*

by (*rule exI* [of - {*Agent 0*}], *simp*)

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 ⇒ key where
shrK A ≡ ltK (sharK A)
```

abbreviation

```
pubK :: agent ⇒ key where
pubK A ≡ ltK (publK A)
```

abbreviation

```
priK :: agent ⇒ key where
priK A ≡ 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 ⇒ key where
  invKey (ltK (publK A)) = priK A
| invKey (ltK (privK A)) = pubK A
| invKey K = K
```

definition

```
symKeys :: key set where
symKeys ≡ {K. invKey K = K}
```

```
lemma invKey-K: K ∈ symKeys ⇒ invKey K = K
by (simp add: symKeys-def)
```

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

```
lemma invKey-invKey-id [simp]: invKey (invKey K) = K
by (cases K, auto)
  (rename-tac ltk, case-tac ltk, auto)
```

```
lemma invKey-eq [simp]: (invKey K = invKey K') = (K=K')
apply (safe)
apply (drule-tac f=invKey in arg-cong, simp)
done
```

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*: $\text{priK } A \neq \text{pubK } A'$
by *auto*

lemma *publicKey-neq-privateKey*: $\text{pubK } A \neq \text{priK } A'$
by *auto*

1.4.2 Basic properties of *pubK* and *priK*

lemma *publicKey-inject* [iff]: $(\text{pubK } A = \text{pubK } A') = (A = A')$
by (*auto*)

lemma *not-symKeys-pubK* [iff]: $\text{pubK } A \notin \text{symKeys}$
by (*simp add: symKeys-def*)

lemma *not-symKeys-priK* [iff]: $\text{priK } A \notin \text{symKeys}$
by (*simp add: symKeys-def*)

lemma *symKey-neq-priK*: $K \in \text{symKeys} \implies K \neq \text{priK } A$
by (*auto simp add: symKeys-def*)

lemma *symKeys-neq-imp-neq*: $(K \in \text{symKeys}) \neq (K' \in \text{symKeys}) \implies K \neq K'$
by *blast*

lemma *symKeys-invKey-iff* [iff]: $(\text{invKey } K \in \text{symKeys}) = (K \in \text{symKeys})$
by (*unfold symKeys-def, auto*)

1.4.3 "Image" equations that hold for injective functions

lemma *invKey-image-eq* [simp]: $(\text{invKey } x \in \text{invKey } A) = (x \in A)$
by *auto*

lemma *invKey-pubK-image-priK-image* [simp]: $\text{invKey } A \text{ pubK } A' = \text{priK } A'$
by (*auto simp add: image-def*)

lemma *publicKey-notin-image-privateKey*: $\text{pubK } A \notin \text{priK } A'$
by *auto*

lemma *privateKey-notin-image-publicKey*: $\text{priK } x \notin \text{pubK } A'$
by *auto*

lemma *publicKey-image-eq* [simp]: $(\text{pubK } x \in \text{pubK } A') = (x \in A')$
by *auto*

lemma *privateKey-image-eq* [simp]: $(\text{priK } A \in \text{priK } A') = (A \in A')$
by *auto*

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
by (*simp add: symKeys-def*)

lemma *sym-shrK*: $shrK\ X \in symKeys$ — All shared keys are symmetric
by (*simp add: symKeys-def*)

Symmetric keys and inversion

lemma *symK-eq-invKey*: $\llbracket SK = invKey\ K; SK \in symKeys \rrbracket \implies K = SK$
by (*auto simp add: symKeys-def*)

Image-related lemmas.

lemma *publicKey-notin-image-shrK*: $pubK\ x \notin shrK\ 'AA$
by *auto*

lemma *privateKey-notin-image-shrK*: $priK\ x \notin shrK\ 'AA$
by *auto*

lemma *shrK-notin-image-publicKey*: $shrK\ x \notin pubK\ 'AA$
by *auto*

lemma *shrK-notin-image-privateKey*: $shrK\ x \notin priK\ 'AA$
by *auto*

lemma *sesK-notin-image-shrK* [*simp*]: $sesK\ K \notin shrK\ 'AA$
by (*auto*)

lemma *shrK-notin-image-sesK* [*simp*]: $shrK\ K \notin sesK\ 'AA$
by (*auto*)

lemma *sesK-image-eq* [*simp*]: $(sesK\ x \in sesK\ 'AA) = (x \in AA)$
by *auto*

lemma *shrK-image-eq* [*simp*]: $(shrK\ x \in shrK\ 'AA) = (x \in AA)$
by *auto*

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 :: (*ltkey* × *agent*) *set* — LT key setup, for now unspecified

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

definition

keySetup :: (*key* × *agent*) *set* **where**
keySetup ≡ {(*ltK* *K*, *A*) | *K* *A*. (*K*, *A*) ∈ *ltkeySetup*}

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

definition

corrKey :: *key set* **where**
corrKey ≡ *keySetup*⁻¹ “ *bad*

lemma *corrKey-Dom-keySetup* [*simp*, *intro*]: *K* ∈ *corrKey* ⇒ *K* ∈ *Domain keySetup*
by (*auto simp add: corrKey-def*)

lemma *keySetup-noSessionKeys* [*simp*]: (*sesK* *K*, *A*) ∉ *keySetup*
by (*simp add: keySetup-def*)

lemma *corrKey-noSessionKeys* [*simp*]: *sesK* *K* ∉ *corrKey*
by (*auto simp add: keySetup-def corrKey-def*)

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* ⇒ *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*]: $map-runs\ h\ Map.empty = Map.empty$

by (*rule ext*) (*auto simp add: map-runs-def*)

lemma $map-runs-dom$ [*simp*]: $dom\ (map-runs\ h\ runz) = dom\ runz$

by (*auto simp add: map-runs-def split: option.split-asm*)

lemma $map-runs-update$ [*simp*]:

$map-runs\ h\ (runz(R \mapsto (rol, agts, al)))$
 $= (map-runs\ h\ runz)(R \mapsto (rol, agts, h\ rol\ al))$

by (*auto simp add: map-runs-def*)

end

1.7 Channel Messages

theory $Channels$ **imports** $Atoms$

begin

1.7.1 Channel messages

datatype $secprop = auth \mid confid$

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*
by (*auto simp add: keys-for-def*)

lemma *keys-for-empty* [*simp*]: *keys-for* {} = {}
by (*simp add: keys-for-def*)

lemma *keys-for-monotone*: *G* \subseteq *H* \implies *keys-for* *G* \subseteq *keys-for* *H*
by (*auto simp add: keys-for-def*)

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*

by (auto simp add: keys-for-def)

lemma keys-for-insert-DynCh [simp]:

keys-for (insert (DynCh c K M) H) = insert K (keys-for H)

by (auto simp add: keys-for-def)

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 :: chmsg set

where

at-StatCh: $\llbracket \text{StatCh } c \ A \ B \ (\text{Msg } M) \in H; \text{At} \in \text{set } M \rrbracket \Longrightarrow \text{At} \in \text{atoms } H$

| at-DynCh: $\llbracket \text{DynCh } c \ K \ (\text{Msg } M) \in H; \text{At} \in \text{set } M \rrbracket \Longrightarrow \text{At} \in \text{atoms } H$

declare atoms.intros [intro]

lemma atoms-empty [simp]: atoms {} = {}

by (auto elim!: atoms.cases)

lemma atoms-monotone: $G \subseteq H \Longrightarrow \text{atoms } G \subseteq \text{atoms } H$

by (auto elim!: atoms.cases)

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

lemma atoms-insert-StatCh [simp]:

atoms (insert (StatCh c A B (Msg M)) H) = set M \cup atoms H

by (auto elim!: atoms.cases)

lemma atoms-insert-DynCh [simp]:

atoms (insert (DynCh c K (Msg M)) H) = set M \cup atoms H

by (auto elim!: atoms.cases)

1.7.4 Intruder knowledge (atoms)

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

inductive-set

extr :: atom set \Rightarrow chmsg set \Rightarrow atom set

for T :: atom set

and H :: chmsg set

where

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

| extr-StatCh:

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

| extr-DynCh:

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

declare *extr.intros* [*intro*]
declare *extr.cases* [*elim*]

Typical parameter describing initial intruder knowledge.

definition

ik0 :: atom set **where**
ik0 \equiv range *aAgt* \cup range *aNum* \cup *aKey'corrKey*

lemma *ik0-aAgt* [*iff*]: *aAgt* *A* \in *ik0*
by (*auto simp add: ik0-def*)

lemma *ik0-aNum* [*iff*]: *aNum* *T* \in *ik0*
by (*auto simp add: ik0-def*)

lemma *ik0-aNon* [*iff*]: *aNon* *N* \notin *ik0*
by (*auto simp add: ik0-def*)

lemma *ik0-aKey-corr* [*simp*]: (*aKey* *K* \in *ik0*) = (*K* \in *corrKey*)
by (*auto simp add: ik0-def*)

Basic lemmas

lemma *extr-empty* [*simp*]: *extr* *T* {} = *T*
by (*auto*)

lemma *extr-monotone* [*dest*]: *G* \subseteq *H* \implies *extr* *T* *G* \subseteq *extr* *T* *H*
by (*safe, erule extr.induct, auto*)

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*
by (*safe, erule extr.induct, auto*)

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*)
by (*erule extr-mono*) (*auto*)

lemma *extr-into-atoms* [*dest*]: *At* \in *extr* *T* *H* \implies *At* \in *T* \cup *atoms* *H*
by (*erule extr.induct, auto*)

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*)
proof –
{ **fix** *Bt*
assume *Bt* \in *extr* (*insert* *At* *T*) *H*
hence *Bt* \in *insert* *At* (*extr* *T* *H*)
using *assms* **by** *induct auto*
}

thus *?thesis* **by** *auto*
qed

lemma *extr-insert-unused-key-param* [*simp*]:
assumes $K \notin \text{keys-for } H$
shows $\text{extr} (\text{insert} (\text{aKey } K) T) H = \text{insert} (\text{aKey } K) (\text{extr } T H)$
proof –
{ **fix** *At*
assume $At \in \text{extr} (\text{insert} (\text{aKey } K) T) H$
hence $At \in \text{insert} (\text{aKey } K) (\text{extr } T H)$
using *assms* **by** *induct (auto simp add: keys-for-def)*
}
thus *?thesis* **by** *auto*
qed

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*]:
 $\text{extr } T (\text{insert} (\text{StatCh } c A B (\text{Msg } M)) H)$
 $= (\text{if } \text{confid} \notin c \vee A \in \text{bad} \vee B \in \text{bad} \text{ then } \text{extr} (\text{set } M \cup T) H \text{ else } \text{extr } T H)$
proof (*cases confid* $\notin c \vee A \in \text{bad} \vee B \in \text{bad}$)
case *True*
moreover
{
fix *At*
assume $At \in \text{extr } T (\text{insert} (\text{StatCh } c A B (\text{Msg } M)) H)$
hence $At \in \text{extr} (\text{set } M \cup T) H$ **by** *induct auto*
}
moreover
{
fix *At*
assume $At \in \text{extr} (\text{set } M \cup T) H$
and $\text{confid} \notin c \vee A \in \text{bad} \vee B \in \text{bad}$
hence $At \in \text{extr } T (\text{insert} (\text{StatCh } c A B (\text{Msg } M)) H)$ **by** *induct auto*
}
ultimately show *?thesis* **by** *auto*
next
case *False*
moreover
{
fix *At*
assume $At \in \text{extr } T (\text{insert} (\text{StatCh } c A B (\text{Msg } M)) H)$
and $\text{confid} \in c \wedge A \notin \text{bad} \wedge B \notin \text{bad}$
hence $At \in \text{extr } T H$ **by** *induct auto*
}
ultimately show *?thesis* **by** *auto*
qed

lemma *extr-insert-DynCh* [*simp*]:
 $\text{extr } T (\text{insert} (\text{DynCh } c K (\text{Msg } M)) H)$

```

= (if confid  $\notin$  c  $\vee$  aKey K  $\in$  extr T H then extr (set M  $\cup$  T) H else extr T H)
proof (cases confid  $\notin$  c  $\vee$  aKey K  $\in$  extr T H)
  case True
  moreover
  {
    fix At
    assume At  $\in$  extr T (insert (DynCh c K (Msg M)) H)
    hence At  $\in$  extr (set M  $\cup$  T) H by induct auto
  }
  moreover
  {
    fix At
    assume At  $\in$  extr (set M  $\cup$  T) H
    hence At  $\in$  extr T (insert (DynCh c K (Msg M)) H)
    using True by induct auto
  }
  ultimately show ?thesis by auto
next
  case False
  moreover
  hence extr T (insert (DynCh c K (Msg M)) H) = extr T H
    by (intro equalityI subsetI) (erule extr.induct, auto)+
  ultimately show ?thesis by auto
qed

```

```

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
by (auto simp add: image-def)

```

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:

```

```

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

```

declare *fake.cases* [elim]

declare *fake.intros* [intro]

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

lemma *fake-expanding* [intro]: $H \subseteq \text{fake } T \cup H$

by (*auto*)

lemma *fake-monotone* [intro]: $G \subseteq H \implies \text{fake } T \cup G \subseteq \text{fake } T \cup H$

by (*safe, erule fake.cases, auto intro!: fake-intros*)

lemma *fake-monotone-param1* [intro]:

$T \subseteq T' \implies \text{fake } T \cup H \subseteq \text{fake } T' \cup H$

by (*safe, erule fake.cases, auto intro!: fake-intros*)

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]: $\text{atoms } (\text{fake } T \cup H) = T \cup \text{atoms } H$

proof –

```

{
  fix At
  assume At ∈ T
  hence At ∈ atoms (fake T U H)
  proof –
    {
      fix A B
      have Insec A B (Msg [At]) ∈ fake T U H using ‹At ∈ T›
      by (intro fake-StatCh) (auto)
    }
  thus ?thesis by (intro at-StatCh) (auto)
}

```

qed

}

moreover

```

{
  fix At
  assume At ∈ atoms (fake T U H)
  hence At ∈ T ∪ atoms H by cases blast+
}

```

ultimately show ?thesis by auto

qed

```

lemma extr-fake [simp]:
  assumes  $T' \subseteq T$  shows  $\text{extr } T (\text{fake } T' U H) = \text{extr } T H$ 
proof (intro equalityI subsetI)
  fix At
  assume  $At \in \text{extr } T (\text{fake } T' U H)$ 
  with assms have  $At \in \text{extr } T (\text{fake } T U H)$  by auto
  thus  $At \in \text{extr } T H$  by induct auto
qed auto

```

end

1.8 Theory of Agents and Messages for Security Protocols

```

theory Message imports Keys begin

```

```

lemma Un-idem-collapse [simp]:  $A \cup (B \cup A) = B \cup A$ 
by blast

```

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

```

syntax-consts

```

  -MTuple    == MPair

```

translations

```

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

```

definition

```

  HPair :: [msg, msg] => msg                ( $\langle \langle \text{4Hash}[-] /- \rangle [0, 1000] \rangle$ )

```

where

```

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

```

definition

```

  keysFor :: msg set => key set

```

where

```

  — Keys useful to decrypt elements of a message set
   $\text{keysFor } H \equiv \text{invKey } \{K. \exists X. \text{Crypt } K X \in H\}$ 

```

Inductive Definition of All Parts" of a Message

inductive-set

```
parts :: msg set => msg set
for H :: msg set
where
  Inj [intro]:      X ∈ H ==> X ∈ parts H
| Fst:      {X, Y} ∈ parts H ==> X ∈ parts H
| Snd:      {X, Y} ∈ parts H ==> Y ∈ parts H
| Body:     Crypt K X ∈ parts H ==> X ∈ parts H
```

Monotonicity

```
lemma parts-mono: G ⊆ H ==> parts(G) ⊆ parts(H)
apply auto
apply (erule parts.induct)
apply (blast dest: parts.Fst parts.Snd parts.Body)+
done
```

Equations hold because constructors are injective.

```
lemma Other-image-eq [simp]: (Agent x ∈ Agent'A) = (x:A)
by auto
```

```
lemma Key-image-eq [simp]: (Key x ∈ Key'A) = (x∈A)
by auto
```

```
lemma Nonce-Key-image-eq [simp]: (Nonce x ∉ Key'A)
by auto
```

1.8.1 keysFor operator

```
lemma keysFor-empty [simp]: keysFor {} = {}
by (unfold keysFor-def, blast)
```

```
lemma keysFor-Un [simp]: keysFor (H ∪ H') = keysFor H ∪ keysFor H'
by (unfold keysFor-def, blast)
```

```
lemma keysFor-UN [simp]: keysFor (⋃ i∈A. H i) = (⋃ i∈A. keysFor (H i))
by (unfold keysFor-def, blast)
```

Monotonicity

```
lemma keysFor-mono: G ⊆ H ==> keysFor(G) ⊆ keysFor(H)
by (unfold keysFor-def, blast)
```

```
lemma keysFor-insert-Agent [simp]: keysFor (insert (Agent A) H) = keysFor H
by (unfold keysFor-def, auto)
```

```
lemma keysFor-insert-Nonce [simp]: keysFor (insert (Nonce N) H) = keysFor H
by (unfold keysFor-def, auto)
```

```
lemma keysFor-insert-Number [simp]: keysFor (insert (Number N) H) = keysFor H
by (unfold keysFor-def, auto)
```

```
lemma keysFor-insert-Key [simp]: keysFor (insert (Key K) H) = keysFor H
```

by (*unfold keysFor-def, auto*)

lemma *keysFor-insert-Hash* [*simp*]: $keysFor (insert (Hash X) H) = keysFor H$
by (*unfold keysFor-def, auto*)

lemma *keysFor-insert-MPair* [*simp*]: $keysFor (insert \{X, Y\} H) = keysFor H$
by (*unfold keysFor-def, auto*)

lemma *keysFor-insert-Crypt* [*simp*]:
 $keysFor (insert (Crypt K X) H) = insert (invKey K) (keysFor H)$
by (*unfold keysFor-def, auto*)

lemma *keysFor-image-Key* [*simp*]: $keysFor (Key E) = \{\}$
by (*unfold keysFor-def, auto*)

lemma *Crypt-imp-invKey-keysFor*: $Crypt K X \in H \implies invKey K \in keysFor H$
by (*unfold keysFor-def, blast*)

1.8.2 Inductive relation "parts"

lemma *MPair-parts*:
 $\llbracket \{X, Y\} \in parts H; \llbracket X \in parts H; Y \in parts H \rrbracket \implies P \rrbracket \implies P$
by (*blast dest: parts.Fst parts.Snd*)

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*: $H \subseteq parts(H)$
by *blast*

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

lemma *parts-empty* [*simp*]: $parts\{\} = \{\}$
apply *safe*
apply (*erule parts.induct, blast+*)
done

lemma *parts-emptyE* [*elim!*]: $X \in parts\{\} \implies P$
by *simp*

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

lemma *parts-singleton*: $X \in parts H \implies \exists Y \in H. X \in parts \{Y\}$
by (*erule parts.induct, fast+*)

Unions

lemma *parts-Un-subset1*: $parts(G) \cup parts(H) \subseteq parts(G \cup H)$
by (*intro Un-least parts-mono Un-upper1 Un-upper2*)

lemma *parts-Un-subset2*: $parts(G \cup H) \subseteq parts(G) \cup parts(H)$
apply (*rule subsetI*)
apply (*erule parts.induct, blast+*)
done

lemma *parts-Un [simp]*: $parts(G \cup H) = parts(G) \cup parts(H)$
by (*intro equalityI parts-Un-subset1 parts-Un-subset2*)

lemma *parts-insert*: $parts(insert\ X\ H) = parts\ \{X\} \cup parts\ H$
apply (*subst insert-is-Un [of - H]*)
apply (*simp only: parts-Un*)
done

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

lemma *parts-insert2*:
 $parts(insert\ X\ (insert\ Y\ H)) = parts\ \{X\} \cup parts\ \{Y\} \cup parts\ H$
apply (*simp add: Un-assoc*)
apply (*simp add: parts-insert [symmetric]*)
done

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*: $insert\ X\ (parts\ H) \subseteq parts(insert\ X\ H)$
by (*blast intro: parts-mono [THEN [2] rev-subsetD]*)

Idempotence and transitivity

lemma *parts-partsD [dest!]*: $X \in parts(parts\ H) \implies X \in parts\ H$
by (*erule parts.induct, blast+*)

lemma *parts-idem [simp]*: $parts(parts\ H) = parts\ H$
by *blast*

lemma *parts-subset-iff [simp]*: $(parts\ G \subseteq parts\ H) = (G \subseteq parts\ H)$
apply (*rule iffI*)
apply (*iprover intro: subset-trans parts-increasing*)
apply (*frule parts-mono, simp*)
done

lemma *parts-trans*: $[X \in parts\ G; G \subseteq parts\ H] \implies X \in parts\ H$
by (*drule parts-mono, blast*)

Cut

lemma *parts-cut*:
 $[Y \in parts(insert\ X\ G); X \in parts\ H] \implies Y \in parts(G \cup H)$
by (*blast intro: parts-trans*)

lemma *parts-cut-eq* [*simp*]: $X \in \text{parts } H \implies \text{parts } (\text{insert } X H) = \text{parts } H$
by (*force dest!*: *parts-cut intro: parts-insertI*)

Rewrite rules for pulling out atomic messages

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

lemma *parts-insert-Agent* [*simp*]:
 $\text{parts } (\text{insert } (\text{Agent } \text{agt}) H) = \text{insert } (\text{Agent } \text{agt}) (\text{parts } H)$
apply (*rule parts-insert-eq-I*)
apply (*erule parts.induct, auto*)
done

lemma *parts-insert-Nonce* [*simp*]:
 $\text{parts } (\text{insert } (\text{Nonce } N) H) = \text{insert } (\text{Nonce } N) (\text{parts } H)$
apply (*rule parts-insert-eq-I*)
apply (*erule parts.induct, auto*)
done

lemma *parts-insert-Number* [*simp*]:
 $\text{parts } (\text{insert } (\text{Number } N) H) = \text{insert } (\text{Number } N) (\text{parts } H)$
apply (*rule parts-insert-eq-I*)
apply (*erule parts.induct, auto*)
done

lemma *parts-insert-Key* [*simp*]:
 $\text{parts } (\text{insert } (\text{Key } K) H) = \text{insert } (\text{Key } K) (\text{parts } H)$
apply (*rule parts-insert-eq-I*)
apply (*erule parts.induct, auto*)
done

lemma *parts-insert-Hash* [*simp*]:
 $\text{parts } (\text{insert } (\text{Hash } X) H) = \text{insert } (\text{Hash } X) (\text{parts } H)$
apply (*rule parts-insert-eq-I*)
apply (*erule parts.induct, auto*)
done

lemma *parts-insert-Crypt* [*simp*]:
 $\text{parts } (\text{insert } (\text{Crypt } K X) H) = \text{insert } (\text{Crypt } K X) (\text{parts } (\text{insert } X H))$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*erule parts.induct, auto*)
apply (*blast intro: parts.Body*)
done

lemma *parts-insert-MPair* [*simp*]:
 $\text{parts } (\text{insert } \{X, Y\} H) =$
 $\text{insert } \{X, Y\} (\text{parts } (\text{insert } X (\text{insert } Y H)))$
apply (*rule equalityI*)
apply (*rule subsetI*)

```

apply (erule parts.induct, auto)
apply (blast intro: parts.Fst parts.Snd)+
done

```

```

lemma parts-image-Key [simp]: parts (Key'N) = Key'N
apply auto
apply (erule parts.induct, auto)
done

```

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 ∈ H ==> X ∈ analz H
| Fst:  {X,Y} ∈ analz H ==> X ∈ analz H
| Snd:  {X,Y} ∈ analz H ==> Y ∈ analz H
| Decrypt [dest]:
  [| Crypt K X ∈ analz H; Key(invKey K): analz H |] ==> X ∈ analz H

```

Monotonicity; Lemma 1 of Lowe's paper

```

lemma analz-mono: G ⊆ H ==> analz(G) ⊆ analz(H)
apply auto
apply (erule analz.induct)
apply (auto dest: analz.Fst analz.Snd)
done

```

```

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

```

Making it safe speeds up proofs

```

lemma MPair-analz [elim!]:
  [| {X,Y} ∈ analz H;
    | X ∈ analz H; Y ∈ analz H |] ==> P
  [|] ==> P
by (blast dest: analz.Fst analz.Snd)

```

```

lemma analz-increasing: H ⊆ analz(H)
by blast

```

```

lemma analz-subset-parts: analz H ⊆ parts H
apply (rule subsetI)
apply (erule analz.induct, blast+)
done

```

```

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

```

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

lemma *parts-analz* [*simp*]: $\text{parts} (\text{analz } H) = \text{parts } H$
apply (*rule equalityI*)
apply (*rule analz-subset-parts* [*THEN parts-mono, THEN subset-trans*], *simp*)
apply (*blast intro: analz-increasing* [*THEN parts-mono, THEN subsetD*])
done

lemma *analz-parts* [*simp*]: $\text{analz} (\text{parts } H) = \text{parts } H$
apply *auto*
apply (*erule analz.induct, auto*)
done

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

General equational properties

lemma *analz-empty* [*simp*]: $\text{analz}\{\} = \{\}$
apply *safe*
apply (*erule analz.induct, blast+*)
done

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*: $\text{analz}(G) \cup \text{analz}(H) \subseteq \text{analz}(G \cup H)$
by (*intro Un-least analz-mono Un-upper1 Un-upper2*)

lemma *analz-insert*: $\text{insert } X (\text{analz } H) \subseteq \text{analz}(\text{insert } X H)$
by (*blast intro: analz-mono* [*THEN [2] rev-subsetD*])

Rewrite rules for pulling out atomic messages

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

lemma *analz-insert-Agent* [*simp*]:
 $\text{analz} (\text{insert} (\text{Agent } \text{agt}) H) = \text{insert} (\text{Agent } \text{agt}) (\text{analz } H)$
apply (*rule analz-insert-eq-I*)
apply (*erule analz.induct, auto*)
done

lemma *analz-insert-Nonce* [*simp*]:
 $\text{analz} (\text{insert} (\text{Nonce } N) H) = \text{insert} (\text{Nonce } N) (\text{analz } H)$
apply (*rule analz-insert-eq-I*)
apply (*erule analz.induct, auto*)
done

lemma *analz-insert-Number* [*simp*]:
 $\text{analz} (\text{insert} (\text{Number } N) H) = \text{insert} (\text{Number } N) (\text{analz } H)$
apply (*rule analz-insert-eq-I*)
apply (*erule analz.induct, auto*)
done

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

$$\text{analz } (\text{insert } (\text{Hash } X) H) = \text{insert } (\text{Hash } X) (\text{analz } H)$$
apply (*rule analz-insert-eq-I*)
apply (*erule analz.induct, auto*)
done

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

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

$$K \notin \text{keysFor } (\text{analz } H) \implies \text{analz } (\text{insert } (\text{Key } K) H) = \text{insert } (\text{Key } K) (\text{analz } H)$$
apply (*unfold keysFor-def*)
apply (*rule analz-insert-eq-I*)
apply (*erule analz.induct, auto*)
done

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

$$\text{analz } (\text{insert } \{X, Y\} H) = \text{insert } \{X, Y\} (\text{analz } (\text{insert } X (\text{insert } Y H)))$$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*erule analz.induct, auto*)
apply (*erule analz.induct*)
apply (*blast intro: analz.Fst analz.Snd*)
done

Can pull out enCrypted message if the Key is not known

lemma *analz-insert-Crypt*:

$$\text{Key } (\text{invKey } K) \notin \text{analz } H \implies \text{analz } (\text{insert } (\text{Crypt } K X) H) = \text{insert } (\text{Crypt } K X) (\text{analz } H)$$
apply (*rule analz-insert-eq-I*)
apply (*erule analz.induct, auto*)
done

lemma *lemma1*: $\text{Key } (\text{invKey } K) \in \text{analz } H \implies$

$$\text{analz } (\text{insert } (\text{Crypt } K X) H) \subseteq \text{insert } (\text{Crypt } K X) (\text{analz } (\text{insert } X H))$$
apply (*rule subsetI*)
apply (*erule-tac x = x in analz.induct, auto*)
done

lemma *lemma2*: $\text{Key } (\text{invKey } K) \in \text{analz } H \implies$

$$\text{insert } (\text{Crypt } K X) (\text{analz } (\text{insert } X H)) \subseteq \text{analz } (\text{insert } (\text{Crypt } K X) H)$$
apply *auto*
apply (*erule-tac x = x in analz.induct, auto*)
apply (*blast intro: analz-insertI analz.Decrypt*)
done

lemma *analz-insert-Decrypt*:

$$\text{Key } (\text{invKey } K) \in \text{analz } H \implies$$

```

      analz (insert (Crypt K X) H) =
      insert (Crypt K X) (analz (insert X H))
by (intro equalityI lemma1 lemma2)

```

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) ∈ analz H)
      then insert (Crypt K X) (analz (insert X H))
      else insert (Crypt K X) (analz H))
by (simp add: analz-insert-Crypt analz-insert-Decrypt)

```

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

```

lemma analz-insert-Crypt-subset:
  analz (insert (Crypt K X) H) ⊆
  insert (Crypt K X) (analz (insert X H))
apply (rule subsetI)
apply (erule analz.induct, auto)
done

```

```

lemma analz-image-Key [simp]: analz (Key'N) = Key'N
apply auto
apply (erule analz.induct, auto)
done

```

Idempotence and transitivity

```

lemma analz-analzD [dest!]: X ∈ analz (analz H) ==> X ∈ analz H
by (erule analz.induct, blast+)

```

```

lemma analz-idem [simp]: analz (analz H) = analz H
by blast

```

```

lemma analz-subset-iff [simp]: (analz G ⊆ analz H) = (G ⊆ analz H)
apply (rule iffI)
apply (iprover intro: subset-trans analz-increasing)
apply (frule analz-mono, simp)
done

```

```

lemma analz-trans: [| X ∈ analz G; G ⊆ analz H |] ==> X ∈ analz H
by (drule analz-mono, blast)

```

Cut; Lemma 2 of Lowe

```

lemma analz-cut: [| Y ∈ analz (insert X H); X ∈ analz H |] ==> Y ∈ analz H
by (erule analz-trans, blast)

```

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 ∈ analz H ==> analz (insert X H) = analz H

```

by (blast intro: analz-cut analz-insertI)

A congruence rule for "analz"

lemma analz-subset-cong:

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

apply simp

apply (iprover intro: conjI subset-trans analz-mono Un-upper1 Un-upper2)

done

lemma analz-cong:

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

by (intro equalityI analz-subset-cong, simp-all)

lemma analz-insert-cong:

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

by (force simp only: insert-def intro!: analz-cong)

If there are no pairs or encryptions then analz does nothing

lemma analz-trivial:

$[[\forall X Y. \{X, Y\} \notin H; \forall X K. \text{Crypt } K X \notin H]] \implies \text{analz } H = H$

apply safe

apply (erule analz.induct, blast+)

done

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]: $X \in H \implies X \in \text{synth } H$

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

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

| *Hash* [intro]: $X \in \text{synth } H \implies \text{Hash } X \in \text{synth } H$

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

| *Crypt* [intro]: $[[X \in \text{synth } H; \text{Key}(K) \in H]] \implies \text{Crypt } K X \in \text{synth } H$

Monotonicity

lemma synth-mono: $G \subseteq H \implies \text{synth}(G) \subseteq \text{synth}(H)$

by (auto, erule synth.induct, auto)

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

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

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

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

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

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

lemma *synth-increasing*: $H \subseteq \text{synth}(H)$
by *blast*

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*: $\text{synth}(G) \cup \text{synth}(H) \subseteq \text{synth}(G \cup H)$
by (*intro Un-least synth-mono Un-upper1 Un-upper2*)

lemma *synth-insert*: $\text{insert } X \ (\text{synth } H) \subseteq \text{synth}(\text{insert } X \ H)$
by (*blast intro: synth-mono [THEN [2] rev-subsetD]*)

Idempotence and transitivity

lemma *synth-synthD* [dest!]: $X \in \text{synth} \ (\text{synth } H) \implies X \in \text{synth } H$
by (*erule synth.induct, blast+*)

lemma *synth-idem*: $\text{synth} \ (\text{synth } H) = \text{synth } H$
by *blast*

lemma *synth-subset-iff* [simp]: $(\text{synth } G \subseteq \text{synth } H) = (G \subseteq \text{synth } H)$
apply (*rule iffI*)
apply (*iprover intro: subset-trans synth-increasing*)
apply (*frule synth-mono, simp add: synth-idem*)
done

lemma *synth-trans*: $[| X \in \text{synth } G; G \subseteq \text{synth } H |] \implies X \in \text{synth } H$
by (*drule synth-mono, blast*)

Cut; Lemma 2 of Lowe

lemma *synth-cut*: $[| Y \in \text{synth} \ (\text{insert } X \ H); X \in \text{synth } H |] \implies Y \in \text{synth } H$
by (*erule synth-trans, blast*)

lemma *Agent-synth* [simp]: $\text{Agent } A \in \text{synth } H$
by *blast*

lemma *Number-synth* [simp]: $\text{Number } n \in \text{synth } H$
by *blast*

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

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

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

lemma *keysFor-synth* [*simp*]:
 $keysFor (synth H) = keysFor H \cup invKey\{K. Key K \in H\}$
by (*unfold keysFor-def*, *blast*)

Combinations of parts, analz and synth

lemma *parts-synth* [*simp*]: $parts (synth H) = parts H \cup synth H$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*erule parts.induct*)
apply (*blast intro: synth-increasing [THEN parts-mono, THEN subsetD]*
 $parts.Fst parts.Snd parts.Body$)
done

lemma *analz-analz-Un* [*simp*]: $analz (analz G \cup H) = analz (G \cup H)$
apply (*intro equalityI analz-subset-cong*)
apply *simp-all*
done

lemma *analz-synth-Un* [*simp*]: $analz (synth G \cup H) = analz (G \cup H) \cup synth G$
apply (*rule equalityI*)
apply (*rule subsetI*)
apply (*erule analz.induct*)
prefer 5 **apply** (*blast intro: analz-mono [THEN [2] rev-subsetD]*)
apply (*blast intro: analz.Fst analz.Snd analz.Decrypt*)
done

lemma *analz-synth* [*simp*]: $analz (synth H) = analz H \cup synth H$
apply (*cut-tac H = {} in analz-synth-Un*)
apply (*simp (no-asm-use)*)
done

chsp: added

lemma *analz-Un-analz* [*simp*]: $analz (G \cup analz H) = analz (G \cup H)$
by (*subst Un-commute, auto*)
done

lemma *analz-synth-Un2* [*simp*]: $analz (G \cup synth H) = analz (G \cup H) \cup synth H$
by (*subst Un-commute, auto*)
done

For reasoning about the Fake rule in traces

lemma *parts-insert-subset-Un*: $X \in G \implies parts(insert X H) \subseteq parts G \cup parts H$
by (*rule subset-trans [OF parts-mono parts-Un-subset2], blast*)

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

lemma *Fake-parts-insert*:
 $X \in synth (analz H) \implies$
 $parts (insert X H) \subseteq synth (analz H) \cup parts H$
apply (*erule parts-insert-subset-Un*)
apply (*simp (no-asm-use)*)

apply *blast*
done

lemma *Fake-parts-insert-in-Un*:

$[[Z \in \text{parts}(\text{insert } X \ H); X \in \text{synth}(\text{analz } H)]]$
 $\implies Z \in \text{synth}(\text{analz } H) \cup \text{parts } H$

by (*blast dest: Fake-parts-insert [THEN subsetD, dest]*)

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

lemma *Fake-analz-insert*:

$X \in \text{synth}(\text{analz } G) \implies$
 $\text{analz}(\text{insert } X \ H) \subseteq \text{synth}(\text{analz } G) \cup \text{analz}(G \cup H)$

apply (*rule subsetI*)

apply (*subgoal-tac* $x \in \text{analz}(\text{synth}(\text{analz } G) \cup H)$)

prefer 2

apply (*blast intro: analz-mono [THEN [2] rev-subsetD]*
 $\text{analz-mono [THEN synth-mono, THEN [2] rev-subsetD]}$)

apply (*simp (no-asm-use)*)

apply *blast*

done

lemma *analz-conj-parts [simp]*:

$(X \in \text{analz } H \ \& \ X \in \text{parts } H) = (X \in \text{analz } H)$

by (*blast intro: analz-subset-parts [THEN subsetD]*)

lemma *analz-disj-parts [simp]*:

$(X \in \text{analz } H \ | \ X \in \text{parts } H) = (X \in \text{parts } H)$

by (*blast intro: analz-subset-parts [THEN subsetD]*)

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

lemma *MPair-synth-analz [iff]*:

$(\{X, Y\} \in \text{synth}(\text{analz } H)) =$
 $(X \in \text{synth}(\text{analz } H) \ \& \ Y \in \text{synth}(\text{analz } H))$

by *blast*

lemma *Crypt-synth-analz*:

$[[\text{Key } K \in \text{analz } H; \ \text{Key}(\text{invKey } K) \in \text{analz } H \]]$
 $\implies (\text{Crypt } K \ X \in \text{synth}(\text{analz } H)) = (X \in \text{synth}(\text{analz } H))$

by *blast*

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

$X \notin \text{synth}(\text{analz } H)$
 $\implies (\text{Hash}\{X, Y\} \in \text{synth}(\text{analz } H)) = (\text{Hash}\{X, Y\} \in \text{analz } H)$

by *blast*

1.8.5 HPair: a combination of Hash and MPair

Freeness

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

by (*unfold HPair-def, simp*)

lemma *Nonce-neq-HPair*: $\text{Nonce } N \sim = \text{Hash}[X] Y$
by (*unfold HPair-def, simp*)

lemma *Number-neq-HPair*: $\text{Number } N \sim = \text{Hash}[X] Y$
by (*unfold HPair-def, simp*)

lemma *Key-neq-HPair*: $\text{Key } K \sim = \text{Hash}[X] Y$
by (*unfold HPair-def, simp*)

lemma *Hash-neq-HPair*: $\text{Hash } Z \sim = \text{Hash}[X] Y$
by (*unfold HPair-def, simp*)

lemma *Crypt-neq-HPair*: $\text{Crypt } K X' \sim = \text{Hash}[X] Y$
by (*unfold HPair-def, simp*)

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*]: $(\text{Hash}[X] Y' = \text{Hash}[X] Y) = (X' = X \ \& \ Y' = Y)$
by (*simp add: HPair-def*)

lemma *MPair-eq-HPair* [*iff*]:
 $(\{X', Y'\} = \text{Hash}[X] Y) = (X' = \text{Hash}\{X, Y\} \ \& \ Y' = Y)$
by (*simp add: HPair-def*)

lemma *HPair-eq-MPair* [*iff*]:
 $(\text{Hash}[X] Y = \{X', Y'\}) = (X' = \text{Hash}\{X, Y\} \ \& \ Y' = Y)$
by (*auto simp add: HPair-def*)

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

lemma *keysFor-insert-HPair* [*simp*]: $\text{keysFor } (\text{insert } (\text{Hash}[X] Y) H) = \text{keysFor } H$
by (*simp add: HPair-def*)

lemma *parts-insert-HPair* [*simp*]:
 $\text{parts } (\text{insert } (\text{Hash}[X] Y) H) =$
 $\text{insert } (\text{Hash}[X] Y) (\text{insert } (\text{Hash}\{X, Y\}) (\text{parts } (\text{insert } Y H)))$
by (*simp add: HPair-def*)

lemma *analz-insert-HPair* [*simp*]:
 $\text{analz } (\text{insert } (\text{Hash}[X] Y) H) =$
 $\text{insert } (\text{Hash}[X] Y) (\text{insert } (\text{Hash}\{X, Y\}) (\text{analz } (\text{insert } Y H)))$
by (*simp add: HPair-def*)

lemma *HPair-synth-analz* [*simp*]:
 $X \notin \text{synth } (\text{analz } H)$
 $\implies (\text{Hash}[X] Y \in \text{synth } (\text{analz } H)) =$
 $(\text{Hash}\{X, Y\} \in \text{analz } H \ \& \ Y \in \text{synth } (\text{analz } H))$
by (*simp add: HPair-def*)

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]: $Crypt\ K\ X \notin Key\ 'A$
by *auto*

lemma *Hash-notin-image-Key* [simp]: $Hash\ X \notin Key\ 'A$
by *auto*

lemma *synth-analz-mono*: $G \subseteq H \implies synth\ (analz\ (G)) \subseteq synth\ (analz\ (H))$
by (*iprover intro: synth-mono analz-mono*)

lemma *Fake-analz-eq* [simp]:
 $X \in synth\ (analz\ H) \implies synth\ (analz\ (insert\ X\ H)) = synth\ (analz\ H)$
apply (*drule Fake-analz-insert*[of - - H])
apply (*simp add: synth-increasing*[THEN *Un-absorb2*])
apply (*drule synth-mono*)
apply (*simp add: synth-idem*)
apply (*rule equalityI*)
apply *simp*
apply (*rule synth-analz-mono, blast*)
done

Two generalizations of *analz-insert-eq*

lemma *gen-analz-insert-eq* [rule-format]:

$X \in \text{analz } H \implies \text{ALL } G. H \subseteq G \dashrightarrow \text{analz } (\text{insert } X \ G) = \text{analz } G$
by (*blast intro: analz-cut analz-insertI analz-mono [THEN [2] rev-subsetD]*)

lemma *synth-analz-insert-eq* [*rule-format*]:

$X \in \text{synth } (\text{analz } H)$

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

apply (*erule synth.induct*)

apply (*simp-all add: gen-analz-insert-eq subset-trans [OF - subset-insertI]*)

done

lemma *Fake-parts-sing*:

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

apply (*rule subset-trans*)

apply (*erule-tac [2] Fake-parts-insert*)

apply (*rule parts-mono, blast*)

done

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

$$lkr\ s \equiv lk\ s \times UNIV$$
1.9.2 Invariant definitions

Global secrecy is stated as an invariant.

definition

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

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

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

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

lemmas $s0g\text{-}domE$ [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*]: $obs\ s0g = id$

by (*simp add: s0g-def*)

All state predicates are trivially observable.

lemma *s0g-anyP-observable* [iff]: *observable (obs s0g) P*
by (*auto*)

1.9.5 Invariant proofs

1.9.6 inv1: Secrecy

lemma *PO-s0g-secrecy-init* [iff]:
init s0g \subseteq s0g-secrecy
by (*auto simp add: s0g-defs intro!: s0g-secrecyI*)

lemma *PO-s0g-secrecy-trans* [iff]:
{s0g-secrecy} trans s0g {> s0g-secrecy}
apply (*auto simp add: s0g-defs PO-hoare-defs intro!: s0g-secrecyI*)
apply (*auto*)
done

lemma *PO-s0g-secrecy* [iff]: *reach s0g \subseteq s0g-secrecy*
by (*rule inv-rule-basic, auto*)

As an external invariant.

lemma *PO-s0g-obs-secrecy* [iff]: *oreach s0g \subseteq s0g-secrecy*
by (*rule external-from-internal-invariant*) (*auto del: subsetI*)

1.9.7 inv2: Authorized and leaked data is known to someone

lemma *PO-s0g-dom-init* [iff]:
init s0g \subseteq s0g-dom
by (*auto simp add: s0g-defs intro!: s0g-domI*)

lemma *PO-s0g-dom-trans* [iff]:
{s0g-dom} trans s0g {> s0g-dom}
apply (*auto simp add: s0g-defs PO-hoare-defs intro!: s0g-domI*)
apply (*blast*)
done

lemma *PO-s0g-dom* [iff]: *reach s0g \subseteq s0g-dom*
by (*rule inv-rule-basic, auto*)

As an external invariant.

lemma *PO-s0g-obs-dom* [iff]: *oreach s0g \subseteq s0g-dom*
by (*rule external-from-internal-invariant*) (*auto del: subsetI*)

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
 $\left. \vphantom{a0n-init} \right\}$

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)
 $\left. \vphantom{a0n-running} \right\}$

Commit signal, marking end of initiator run.

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

where

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

Data corruption.

definition

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

where

$a0n\text{-corrupt } ds \equiv \{(s, s')\}.$
— actions:
 $s' = s \{$
 $\text{corrupted} := \text{corrupted } s \cup ds$
 $\}$
 $\}$

Transition system.

definition

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

$a0n :: ('ds \ a0n\text{-state}, 'ds \ a0n\text{-obs}) \text{ spec}$ **where**
 $a0n \equiv \{$
 $\text{init} = a0n\text{-init},$
 $\text{trans} = a0n\text{-trans},$
 $\text{obs} = id$
 $\}$

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 $a0n\text{-obs } [simp]: \text{obs } a0n = id$
by ($simp \ add: a0n\text{-def}$)

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

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

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

where

$a0n\text{-}inv1\text{-}niagree \equiv \{s. \forall h\ d.$
 $set\ h \subseteq good \longrightarrow d \notin corrupted\ s \longrightarrow$
 $signals\ s\ (Commit\ h\ d) > 0 \longrightarrow signals\ s\ (Running\ h\ d) > 0$
 $\}$

lemmas $a0n\text{-}inv1\text{-}niagreeI =$

$a0n\text{-}inv1\text{-}niagree\text{-}def\ [THEN\ setc\text{-}def\text{-}to\text{-}intro,\ rule\text{-}format]$

lemmas $a0n\text{-}inv1\text{-}niagreeE\ [elim] =$

$a0n\text{-}inv1\text{-}niagree\text{-}def\ [THEN\ setc\text{-}def\text{-}to\text{-}elim,\ rule\text{-}format]$

lemmas $a0n\text{-}inv1\text{-}niagreeD =$

$a0n\text{-}inv1\text{-}niagree\text{-}def\ [THEN\ setc\text{-}def\text{-}to\text{-}dest,\ rule\text{-}format,\ rotated\ 2]$

Invariance proof.

lemma $PO\text{-}a0n\text{-}inv1\text{-}niagree\text{-}init\ [iff]:$

$init\ a0n \subseteq a0n\text{-}inv1\text{-}niagree$

by ($auto\ simp\ add: a0n\text{-}defs\ intro!: a0n\text{-}inv1\text{-}niagreeI$)

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

$\{a0n\text{-}inv1\text{-}niagree\}\ trans\ a0n\ \{>\ a0n\text{-}inv1\text{-}niagree\}$

apply ($auto\ simp\ add: PO\text{-}hoare\text{-}defs\ a0n\text{-}defs\ intro!: a0n\text{-}inv1\text{-}niagreeI$)

apply ($auto\ dest!: a0n\text{-}inv1\text{-}niagreeD\ dest: dom\text{-}lemmas$)

done

lemma $PO\text{-}a0n\text{-}inv1\text{-}niagree\ [iff]: reach\ a0n \subseteq a0n\text{-}inv1\text{-}niagree$

by ($rule\ inv\text{-}rule\text{-}basic$) ($auto$)

This is also an external invariant.

lemma $a0n\text{-}obs\text{-}inv1\text{-}niagree\ [iff]:$

$oreach\ a0n \subseteq a0n\text{-}inv1\text{-}niagree$

apply ($rule\ external\text{-}from\text{-}internal\text{-}invariant,\ fast$)

apply ($subst\ a0n\text{-}def,\ auto$)

done

end

1.11 Injective Agreement

theory $a0i\text{-}agree\ imports\ a0n\text{-}agree$

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

$'d \text{ a0i-state} = 'd \text{ a0n-state}$

type-synonym

$'d \text{ a0i-obs} = 'd \text{ a0n-obs}$

1.11.2 Events

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

abbreviation

$\text{a0i-init} :: 'ds \text{ a0n-state set}$

where

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

abbreviation

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

where

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

definition

$\text{a0i-commit} ::$

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

where

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

— 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

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

where

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

Transition system.

definition

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

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

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

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

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

Id

$)$

definition

$a0i :: ('ds\ a0i\text{-state}, 'ds\ a0i\text{-obs})\ \text{spec}\ \text{where}$
 $a0i \equiv \langle$
 $\ \ \ \ \ \text{init} = a0i\text{-init},$
 $\ \ \ \ \ \text{trans} = a0i\text{-trans},$
 $\ \ \ \ \ \text{obs} = id$
 $\ \ \ \ \ \rangle$

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

Any property is trivially observable.

lemma $a0i\text{-obs}\ [simp]:\ \text{obs}\ a0i = id$ **by** $(simp\ add:\ a0i\text{-def})$ **lemma** $a0i\text{-anyP-observable}\ [iff]:\ \text{observable}\ (\text{obs}\ a0i)\ P$ **by** $(auto)$

1.11.3 Invariants

Injective agreement.**definition** $a0i\text{-inv1-iagree} :: 'ds\ a0i\text{-state}\ \text{set}$ **where**

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

lemmas $a0i\text{-inv1-iagreeI} =$ $a0i\text{-inv1-iagree-def}\ [THEN\ \text{setc-def-to-intro},\ \text{rule-format}]$ **lemmas** $a0i\text{-inv1-iagreeE}\ [elim] =$ $a0i\text{-inv1-iagree-def}\ [THEN\ \text{setc-def-to-elim},\ \text{rule-format}]$ **lemmas** $a0i\text{-inv1-iagreeD} =$ $a0i\text{-inv1-iagree-def}\ [THEN\ \text{setc-def-to-dest},\ \text{rule-format},\ \text{rotated}\ 1]$ **lemma** $PO\text{-}a0i\text{-inv1-iagree-init}\ [iff]:$ $\ \ \ \ \ \text{init}\ a0i \subseteq a0i\text{-inv1-iagree}$ **by** $(auto\ simp\ add:\ a0i\text{-defs}\ \text{intro!}:\ a0i\text{-inv1-iagreeI})$ **lemma** $PO\text{-}a0i\text{-inv1-iagree-trans}\ [iff]:$ $\ \ \ \ \ \{a0i\text{-inv1-iagree}\}\ \text{trans}\ a0i\ \{>\ a0i\text{-inv1-iagree}\}$ **apply** $(auto\ simp\ add:\ PO\text{-hoare-defs}\ a0i\text{-defs}\ \text{intro!}:\ a0i\text{-inv1-iagreeI})$ **apply** $(auto\ dest:\ a0i\text{-inv1-iagreeD}\ \text{intro}:\ le\text{-SucI})$ **done****lemma** $PO\text{-}a0i\text{-inv1-iagree}\ [iff]:\ \text{reach}\ a0i \subseteq a0i\text{-inv1-iagree}$ **by** $(rule\ \text{inv-rule-basic})\ (auto)$

As an external invariant.

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

apply (*rule external-from-internal-invariant, fast*)
apply (*subst a0i-def, auto*)
done

1.11.4 Refinement

definition

$med0n0i :: 'd\ a0i\text{-obs} \Rightarrow 'd\ a0i\text{-obs}$

where

$med0n0i \equiv id$

definition

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

where

$R0n0i \equiv Id$

lemma *PO-a0i-running-refines-a0n-running*:

$\{R0n0i\}$
 $(a0n\text{-running}\ h\ d), (a0i\text{-running}\ h\ d)$
 $\{>\ R0n0i\}$

by (*unfold R0n0i-def*) (*rule relhoare-refl*)

lemma *PO-a0i-commit-refines-a0n-commit*:

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

by (*auto simp add: PO-rhoare-defs R0n0i-def a0i-defs*)

lemma *PO-a0i-corrupt-refines-a0n-corrupt*:

$\{R0n0i\}$
 $(a0n\text{-corrupt}\ d), (a0i\text{-corrupt}\ d)$
 $\{>\ R0n0i\}$

by (*unfold R0n0i-def*) (*rule relhoare-refl*)

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]*:

$init\ a0i \subseteq R0n0i\text{“}(init\ a0n)$

by (*auto simp add: R0n0i-def a0i-defs*)

lemma *PO-m1-refines-trans-a0n [iff]*:

$\{R0n0i\}$
 $(trans\ a0n), (trans\ a0i)$
 $\{>\ R0n0i\}$

by (*auto simp add: a0n-def a0n-trans-def a0i-def a0i-trans-def*
intro!: PO-a0i-trans-refines-a0n-trans)

lemma *PO-obs-consistent [iff]*:

obs-consistent R0n0i med0n0i a0n a0i
by (*auto simp add: obs-consistent-def R0n0i-def med0n0i-def a0i-def a0n-def*)

lemma *PO-a0i-refines-a0n*:
refines R0n0i med0n0i a0n a0i
by (*rule Refinement-basic*) (*auto*)

1.11.5 Derived invariants

lemma *iagree-implies-niagree [iff]: a0i-inv1-iagree \subseteq a0n-inv1-niagree*
apply (*auto intro!: a0n-inv1-niagreeI*)
apply (*drule-tac d=d in a0i-inv1-iagreeD, auto*)
done

Non-injective agreement as internal and external invariants.

lemma *PO-a0i-a0n-inv1-niagree [iff]: reach a0i \subseteq a0n-inv1-niagree*
by (*rule subset-trans, rule, rule*)

lemma *PO-a0i-obs-a0n-inv1-niagree [iff]: oreach a0i \subseteq a0n-inv1-niagree*
by (*rule subset-trans, rule, rule*)

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
 $runs \ s \ Ra = Some \ (Init, [A, B], []) \wedge$
 $Na = Ra\$0 \wedge$

— authentication guard:
 $(A \notin bad \wedge B \notin bad \longrightarrow (\exists Rb.$
 $Nb = Rb\$0 \wedge runs \ s \ Rb = Some \ (Resp, [A, B], [aNon \ Na]))) \wedge$

— 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*]:
 $runz = Map.empty \implies runs2sigs\ runz = (\lambda x. 0)$
by (*rule ext, erule rev-mp*)
(*rule runs2sigs.induct, auto*)

Update lemmas

lemma *runs2sigs-upd-init-none* [*simp*]:
 $\llbracket Ra \notin dom\ runz \rrbracket$
 $\implies runs2sigs\ (runz(Ra \mapsto (Init, [A, B], []))) = runs2sigs\ runz$
by (*rule ext, erule rev-mp*)
(*rule runs2sigs.induct, auto dest: dom-lemmas*)

lemma *runs2sigs-upd-init-some* [*simp*]:
 $\llbracket runz\ Ra = Some\ (Init, [A, B], []) \rrbracket$
 $\implies runs2sigs\ (runz(Ra \mapsto (Init, [A, B], [aNon\ Nb]))) =$
 $(runs2sigs\ runz)(Commit\ [A, B]\ (Ra\$0, Nb)\ :=\ 1)$
by (*rule ext, erule rev-mp*)
(*rule runs2sigs.induct, auto*)

lemma *runs2sigs-upd-resp* [*simp*]:
 $\llbracket Rb \notin dom\ runz \rrbracket$
 $\implies runs2sigs\ (runz(Rb \mapsto (Resp, [A, B], [aNon\ Na]))) =$
 $(runs2sigs\ runz)(Running\ [A, B]\ (Na, Rb\$0)\ :=\ 1)$
by (*rule ext, (erule rev-mp)+*)
(*rule runs2sigs.induct, auto dest: dom-lemmas*)

2.1.4 Refinement

lemma *PO-m1-step1-refines-skip*:
 $\{R01\}$
 $Id, (m1-step1\ Ra\ A\ B\ Na)$
 $\{>\ R01\}$
by (*auto simp add: PO-rhoare-def R01-defs a0i-defs m1-defs*)

lemma *PO-m1-step2-refines-a0i-running*:
 $\{R01\}$
 $(a0i-running\ [A, B]\ (Na, Nb), (m1-step2\ Rb\ A\ B\ Na\ Nb))$
 $\{>\ R01\}$
by (*auto simp add: PO-rhoare-defs R01-defs a0i-defs m1-defs dest: dom-lemmas*)

lemma *PO-m1-step3-refines-a0i-commit*:
 $\{R01\}$
 $(a0i-commit\ [A, B]\ (Na, Nb), (m1-step3\ Ra\ A\ B\ Na\ Nb))$
 $\{>\ R01\}$
by (*auto simp add: PO-rhoare-defs R01-defs a0i-defs m1-defs*)

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*]:
 $init\ m1 \subseteq R01 \iff (init\ a0i)$

by (*auto simp add: R01-defs a0i-defs m1-defs*)

lemma *PO-m1-refines-trans-a0i* [*iff*]:

{*R01*}
(*trans a0i*), (*trans m1*)
{> *R01*}

by (*auto simp add: m1-def m1-trans-def a0i-def a0i-trans-def*
intro!: PO-m1-trans-refines-a0i-trans)

lemma *PO-obs-consistent* [*iff*]:

obs-consistent R01 med10 a0i m1

by (*auto simp add: obs-consistent-def R01-defs a0i-def m1-def*)

lemma *PO-m1-refines-a0i*:

refines R01 med10 a0i m1

by (*rule Refinement-basic*) (*auto*)

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.

$$\begin{aligned} \text{M1. } A \rightarrow B & : A, B, N_A \\ \text{M2. } B \rightarrow A & : \{N_B, N_A, A\}_{K^{-1}(B)} \end{aligned}$$

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

$m2\text{-step1} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}] \Rightarrow (m2\text{-state} \times m2\text{-state}) \text{ set}$

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
 $s1 = s\{$
 $\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [])),$
 $\text{chan} := \text{insert } (\text{Insec } A \ B \ (\text{Msg } [a\text{Non } Na])) (\text{chan } s)$
 $\}$

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
 $Rb \notin \text{dom} (\text{runs } s) \wedge$
 $Nb = Rb\$0 \wedge$
 $\text{Insec } A \ B \ (\text{Msg } [a\text{Non } Na]) \in \text{chan } s \wedge$ — $\text{rcv } M1$
 — actions
 $s1 = s\{$
 $\text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [a\text{Non } Na])),$
 $\text{chan} := \text{insert } (\text{Auth } B \ A \ (\text{Msg } [a\text{Non } Nb, a\text{Non } Na])) (\text{chan } s)$ — $\text{snd } M2$
 $\}$

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
 $\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], []) \wedge$
 $Na = Ra\$0 \wedge$
 $\text{Auth } B \ A \ (\text{Msg } [a\text{Non } Nb, a\text{Non } Na]) \in \text{chan } s \wedge$ — $\text{recv } M2$
 — actions
 $s1 = s\{$
 $\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [a\text{Non } Nb]))$
 $\}$

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:

$s1 = s(\text{chan} := \text{fake ik}0 (\text{dom} (\text{runs } s)) (\text{chan } s) \mid)$
}

Transition system.

definition

$m2\text{-init} :: m2\text{-state set}$

where

$m2\text{-init} \equiv \{ \mid$
 $\text{runs} = \text{Map.empty},$
 $\text{chan} = \{\}$
 $\mid \}$

definition

$m2\text{-trans} :: (m2\text{-state} \times m2\text{-state}) \text{ set}$ **where**

$m2\text{-trans} \equiv (\bigcup A B Ra Rb Na Nb.$
 $(m2\text{-step1 } Ra A B Na) \cup$
 $(m2\text{-step2 } Rb A B Na Nb) \cup$
 $(m2\text{-step3 } Ra A B Na Nb) \cup$
 $m2\text{-fake} \cup$
 Id
 $)$

definition

$m2 :: (m2\text{-state}, m2\text{-obs}) \text{ spec}$ **where**

$m2 \equiv \{ \mid$
 $\text{init} = m2\text{-init},$
 $\text{trans} = m2\text{-trans},$
 $\text{obs} = m2\text{-obs}$
 \mid

lemmas $m2\text{-defs} =$

$m2\text{-def } m2\text{-init-def } m2\text{-trans-def } m2\text{-obs-def}$
 $m2\text{-step1-def } m2\text{-step2-def } m2\text{-step3-def } m2\text{-fake-def}$

2.2.3 Invariants

Authentic channel and responder

This property relates the messages in the authentic channel to the responder run frame.

definition

$m2\text{-inv1-auth} :: m2\text{-state set}$ **where**

$m2\text{-inv1-auth} \equiv \{s. \forall A B Na Nb.$
 $Auth B A (\text{Msg } [aNon Nb, aNon Na]) \in \text{chan } s \longrightarrow B \notin \text{bad} \longrightarrow A \notin \text{bad} \longrightarrow$
 $(\exists Rb. \text{runs } s Rb = \text{Some } (\text{Resp}, [A, B], [aNon Na]) \wedge Nb = Rb\$0)$
 $\}$

lemmas $m2\text{-inv1-authI} =$

$m2\text{-inv1-auth-def } [THEN \text{setc-def-to-intro}, \text{rule-format}]$

lemmas $m2\text{-inv1-authE}$ [elim] =
 $m2\text{-inv1-auth-def}$ [THEN setc-def-to-elim, rule-format]
lemmas $m2\text{-inv1-authD}$ [dest] =
 $m2\text{-inv1-auth-def}$ [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proof.

lemma $PO\text{-}m2\text{-inv2-init}$ [iff]:
 $init\ m2 \subseteq m2\text{-inv1-auth}$
by (auto simp add: $PO\text{-hoare-def}$ $m2\text{-defs}$ intro!: $m2\text{-inv1-authI}$)

lemma $PO\text{-}m2\text{-inv2-trans}$ [iff]:
 $\{m2\text{-inv1-auth}\ trans\ m2 \{>\ m2\text{-inv1-auth}\}$
apply (auto simp add: $PO\text{-hoare-def}$ $m2\text{-defs}$ intro!: $m2\text{-inv1-authI}$)
apply (auto dest: dom-lemmas)
— 1 subgoal
apply (force)
done

lemma $PO\text{-}m2\text{-inv2}$ [iff]: $reach\ m2 \subseteq m2\text{-inv1-auth}$
by (rule-tac inv-rule-incr) (auto)

2.2.4 Refinement

Simulation relation and mediator function. This is a pure superposition refinement.

definition
 $R12 :: (m1\text{-state} \times m2\text{-state})\ set$ **where**
 $R12 \equiv \{(s, t). runs\ s = runs\ t\}$ — That's it!

definition
 $med21 :: m2\text{-obs} \Rightarrow m1\text{-obs}$ **where**
 $med21 \equiv id$

Refinement proof

lemma $PO\text{-}m2\text{-step1-refines-}m1\text{-step1}$:
 $\{R12\}$
 $(m1\text{-step1}\ Ra\ A\ B\ Na), (m2\text{-step1}\ Ra\ A\ B\ Na)$
 $\{>\ R12\}$
by (auto simp add: $PO\text{-rhoare-defs}$ $R12\text{-def}$ $m1\text{-defs}$ $m2\text{-defs}$)

lemma $PO\text{-}m2\text{-step2-refines-}m1\text{-step2}$:
 $\{R12\}$
 $(m1\text{-step2}\ Ra\ A\ B\ Na\ Nb), (m2\text{-step2}\ Ra\ A\ B\ Na\ Nb)$
 $\{>\ R12\}$
by (auto simp add: $PO\text{-rhoare-defs}$ $R12\text{-def}$ $m1\text{-defs}$ $m2\text{-defs}$)

lemma $PO\text{-}m2\text{-step3-refines-}m1\text{-step3}$:
 $\{R12 \cap UNIV \times m2\text{-inv1-auth}\}$
 $(m1\text{-step3}\ Ra\ A\ B\ Na\ Nb), (m2\text{-step3}\ Ra\ A\ B\ Na\ Nb)$
 $\{>\ R12\}$
by (auto simp add: $PO\text{-rhoare-defs}$ $R12\text{-def}$ $m1\text{-defs}$ $m2\text{-defs}$)

New fake event refines skip.

lemma *PO-m2-fake-refines-m1-skip*:
 $\{R12\}$ *Id*, *m2-fake* $\{> R12\}$
by (*auto simp add: PO-rhoare-defs R12-def m1-defs m2-defs*)

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*]:
 $init\ m2 \subseteq R12''(init\ m1)$
by (*auto simp add: R12-def m1-defs m2-defs*)

lemma *PO-m2-refines-trans-m1* [*iff*]:
 $\{R12 \cap UNIV \times m2-inv1-auth\}$
 $(trans\ m1), (trans\ m2)$
 $\{> R12\}$
apply (*auto simp add: m2-def m2-trans-def m1-def m1-trans-def*)
apply (*blast intro!: PO-m2-trans-refines-m1-trans*)
done

lemma *PO-obs-consistent* [*iff*]:
 $obs-consistent\ R12\ med21\ m1\ m2$
by (*auto simp add: obs-consistent-def R12-def med21-def m1-defs m2-defs*)

lemma *m2-refines-m1*:
 $refines$
 $(R12 \cap UNIV \times m2-inv1-auth)$
 $med21\ m1\ m2$
by (*rule Refinement-using-invariants*) (*auto*)

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

$m2-obs = m1-state$

definition

$m2-obs :: m2-state \Rightarrow m2-obs$ **where**

$m2-obs\ s \equiv \langle$
 $runs = runs\ s$
 \rangle

2.3.2 Events

definition

$m2-init :: m2-state\ set$

where

$m2-init \equiv \{ \langle$
 $runs = Map.empty,$
 $chan = \{\}$
 $\rangle \}$

definition

$m2-step1 :: [rid-t, agent, agent, nonce] \Rightarrow (m2-state \times m2-state)\ set$

where

$m2-step1\ Ra\ A\ B\ Na \equiv \{(s, s1).$

— guards:

$Ra \notin dom\ (runs\ s) \wedge$
 $Na = Ra\$0 \wedge$

— actions:

$s1 = s \langle$
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [])),$
 — send Na on confidential channel 1
 $chan := insert\ (Confid\ A\ B\ (Msg\ [aNon\ Na]))\ (chan\ s)$
 \rangle

}

definition

$m2-step2 :: [rid-t, agent, agent, nonce, nonce] \Rightarrow (m2-state \times m2-state)\ set$

where

$m2-step2\ Rb\ A\ B\ Na\ Nb \equiv \{(s, s1).$

— guards

$Rb \notin dom\ (runs\ s) \wedge$
 $Nb = Rb\$0 \wedge$

$Confid\ A\ B\ (Msg\ [aNon\ Na]) \in chan\ s \wedge$ — receive M1

— actions

$s1 = s \langle$

```

    runs := (runs s)(Rb ↦ (Resp, [A, B], [aNon Na])),
    chan := insert (Confid B A (Msg [aNon Na, aNon Nb])) (chan s)
  ⌋
}

```

definition

$m2\text{-step3} :: [\text{rid-t}, \text{agent}, \text{agent}, \text{nonce}, \text{nonce}] \Rightarrow (\text{m2-state} \times \text{m2-state}) \text{ set}$

where

$m2\text{-step3 } Ra \ A \ B \ Na \ Nb \equiv \{(s, s1)\}.$

— guards

$runs \ s \ Ra = \text{Some } (Init, [A, B], []) \wedge$
 $Na = Ra\$0 \wedge$

$Confid \ B \ A \ (Msg \ [aNon \ Na, \ aNon \ Nb]) \in \text{chan } s \wedge$ — receive M2

— actions

$s1 = s \langle$
 $runs := (runs \ s)(Ra \mapsto (Init, [A, B], [aNon \ Nb]))$

\rangle
 $\}$

Intruder fake event.

definition — refines Id

$m2\text{-fake} :: (\text{m2-state} \times \text{m2-state}) \text{ set}$

where

$m2\text{-fake} \equiv \{(s, s1)\}.$

— actions:

$s1 = s \langle \text{chan} := \text{fake } ik0 \ (\text{dom } (runs \ s)) \ (\text{chan } s) \rangle$

$\}$

Transition system.

definition

$m2\text{-trans} :: (\text{m2-state} \times \text{m2-state}) \text{ set}$ **where**

$m2\text{-trans} \equiv (\bigcup \ A \ B \ Ra \ Rb \ Na \ Nb.$

$m2\text{-step1 } Ra \ A \ B \ Na \ \cup$

$m2\text{-step2 } Rb \ A \ B \ Na \ Nb \ \cup$

$m2\text{-step3 } Ra \ A \ B \ Na \ Nb \ \cup$

$m2\text{-fake} \ \cup$

Id

\rangle

definition

$m2 :: (\text{m2-state}, \text{m2-obs}) \text{ spec}$ **where**

$m2 \equiv \langle$

$init = m2\text{-init},$

$trans = m2\text{-trans},$

$obs = m2\text{-obs}$

\rangle

lemmas $m2\text{-defs} =$

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-inv1-noncesI* =

m2-inv1-nonces-def [*THEN setc-def-to-intro, rule-format*]

lemmas *m2-inv1-noncesE* [*elim*] =

m2-inv1-nonces-def [*THEN setc-def-to-elim, rule-format*]

lemmas *m2-inv1-noncesD* =

m2-inv1-nonces-def [*THEN setc-def-to-dest, rule-format, rotated 1*]

lemma *PO-m2-inv1-init* [*iff*]: *init m2* \subseteq *m2-inv1-nonces*

by (*auto simp add: PO-hoare-def m2-defs intro!: m2-inv1-noncesI*)

lemma *PO-m2-inv1-trans* [*iff*]:

$\{m2-inv1-nonces\}$ *trans m2* $\{> m2-inv1-nonces\}$

apply (*auto simp add: PO-hoare-def m2-defs intro!: m2-inv1-noncesI*)

apply (*auto dest: m2-inv1-noncesD*)

— 1 subgoal

apply (*subgoal-tac aNon (R\\$0) \in atoms (chan xa), auto*)

done

lemma *PO-m2-inv012* [*iff*]:

reach m2 \subseteq *m2-inv1-nonces*

by (*rule inv-rule-basic*) (*auto*)

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-inv3-msg2 :: *m2-state set* **where**

m2-inv3-msg2 $\equiv \{s. \forall A B Na Nb.$

$Confid B A (Msg [aNon Na, aNon Nb]) \in chan s \longrightarrow$

$aNon Na \notin extr ik0 (chan s) \longrightarrow$

$(\exists Rb. Nb = Rb\$0 \wedge runs s Rb = Some (Resp, [A, B], [aNon Na]))$

$\}$

lemmas *m2-inv3-msg2I* = *m2-inv3-msg2-def* [*THEN setc-def-to-intro, rule-format*]

lemmas *m2-inv3-msg2E* [*elim*] = *m2-inv3-msg2-def* [*THEN setc-def-to-elim, rule-format*]

lemmas *m2-inv3-msg2D* = *m2-inv3-msg2-def* [*THEN setc-def-to-dest, rule-format, rotated 1*]

lemma *PO-m2-inv4-init* [*iff*]:

$init\ m2 \subseteq m2\text{-inv3}\text{-msg2}$
by (*auto simp add: PO-hoare-def m2-defs intro!: m2-inv3-msg2I*)

lemma *PO-m2-inv4-trans* [*iff*]:
 $\{m2\text{-inv3}\text{-msg2}\} \text{ trans } m2 \{> m2\text{-inv3}\text{-msg2}\}$
apply (*auto simp add: PO-hoare-def m2-defs intro!: m2-inv3-msg2I*)
apply (*auto dest: m2-inv3-msg2D dom-lemmas*)
— 2 subgoals
apply (*drule m2-inv3-msg2D, auto dest: dom-lemmas*)
apply (*drule m2-inv3-msg2D, auto, force*)
done

lemma *PO-m2-inv4* [*iff*]: $reach\ m2 \subseteq m2\text{-inv3}\text{-msg2}$
by (*rule inv-rule-incr*) (*auto del: subsetI*)

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 $x = None$, but we have generalized it here.

definition

$m2\text{-inv4}\text{-inon}\text{-secret} :: m2\text{-state set}$ **where**
 $m2\text{-inv4}\text{-inon}\text{-secret} \equiv \{s. \forall A\ B\ Ra\ al.$
 $runs\ s\ Ra = Some\ (Init, [A, B], al) \longrightarrow$
 $A \notin bad \longrightarrow B \notin bad \longrightarrow$
 $aNon\ (Ra\$0) \notin extr\ ik0\ (chan\ s)$
 $\}$

lemmas $m2\text{-inv4}\text{-inon}\text{-secret}I =$
 $m2\text{-inv4}\text{-inon}\text{-secret}\text{-def}$ [*THEN setc-def-to-intro, rule-format*]
lemmas $m2\text{-inv4}\text{-inon}\text{-secret}E$ [*elim*] =
 $m2\text{-inv4}\text{-inon}\text{-secret}\text{-def}$ [*THEN setc-def-to-elim, rule-format*]
lemmas $m2\text{-inv4}\text{-inon}\text{-secret}D =$
 $m2\text{-inv4}\text{-inon}\text{-secret}\text{-def}$ [*THEN setc-def-to-dest, rule-format, rotated 1*]

lemma *PO-m2-inv3-init* [*iff*]:
 $init\ m2 \subseteq m2\text{-inv4}\text{-inon}\text{-secret}$
by (*auto simp add: PO-hoare-def m2-defs intro!: m2-inv4-inon-secretI*)

lemma *PO-m2-inv3-trans* [*iff*]:
 $\{m2\text{-inv4}\text{-inon}\text{-secret} \cap m2\text{-inv1}\text{-nonces}\}$
 $\text{ trans } m2$
 $\{> m2\text{-inv4}\text{-inon}\text{-secret}\}$
apply (*auto simp add: PO-hoare-def m2-defs intro!: m2-inv4-inon-secretI*)
apply (*auto dest: m2-inv4-inon-secretD*)
— 3 subgoals
apply (*fastforce*) — requires *m2-inv1-nonces*
apply (*fastforce*) — requires *ind hyp*
apply (*fastforce*) — requires *ind hyp*
done

lemma *PO-m2-inv3* [*iff*]: $reach\ m2 \subseteq m2\text{-inv4}\text{-inon}\text{-secret}$

by (rule inv-rule-incr [where $J=m2\text{-inv1}\text{-nonces}$]) (auto)

2.3.4 Refinement

definition

$R12 :: (m1\text{-state} \times m2\text{-state})$ set **where**
 $R12 \equiv \{(s, t). \text{runs } s = \text{runs } t\}$

abbreviation

$med21 :: m2\text{-obs} \Rightarrow m1\text{-obs}$ **where**
 $med21 \equiv id$

Proof obligations.

lemma *PO-m2-step1-refines-m1-step1*:

$\{R12\}$
 $(m1\text{-step1 } Ra \ A \ B \ Na), (m2\text{-step1 } Ra \ A \ B \ Na)$
 $\{> R12\}$

by (auto simp add: PO-rhoare-defs R12-def m1-defs m2-defs)

lemma *PO-m2-step2-refines-m1-step2*:

$\{R12\}$
 $(m1\text{-step2 } Rb \ A \ B \ Na \ Nb), (m2\text{-step2 } Rb \ A \ B \ Na \ Nb)$
 $\{> R12\}$

by (auto simp add: PO-rhoare-defs R12-def m1-defs m2-defs)

lemma *PO-m2-step3-refines-m1-step3*:

$\{R12 \cap UNIV \times (m2\text{-inv4}\text{-inon}\text{-secret} \cap m2\text{-inv3}\text{-msg2})\}$
 $(m1\text{-step3 } Ra \ A \ B \ Na \ Nb), (m2\text{-step3 } Ra \ A \ B \ Na \ Nb)$
 $\{> R12\}$

by (auto simp add: PO-rhoare-defs R12-def m1-defs m2-defs)
(blast)

New fake events refine skip.

lemma *PO-m2-fake-refines-skip*:

$\{R12\} Id, m2\text{-fake} \{> R12\}$

by (auto simp add: PO-rhoare-def R12-def m1-defs m2-defs)

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]:

$init \ m2 \subseteq R12 \text{“}(init \ m1)$

by (auto simp add: R12-def m1-defs m2-defs)

lemma *PO-m2-refines-trans-m1* [iff]:

$\{R12 \cap$
 $UNIV \times (m2\text{-inv4}\text{-inon}\text{-secret} \cap m2\text{-inv3}\text{-msg2})\}$
 $(trans \ m1), (trans \ m2)$
 $\{> R12\}$

apply (auto simp add: m2-def m2-trans-def m1-def m1-trans-def)

apply (*blast intro!*: *PO-m2-trans-refines-m1-trans*)
done

lemma *PO-R12-obs-consistent* [*iff*]:
obs-consistent R12 med21 m1 m2
by (*auto simp add: obs-consistent-def R12-def m1-defs m2-defs*)

lemma *PO-m3-refines-m2*:
refines
(R12 \cap
*UNIV \times (*m2-inv4-inon-secret* \cap *m2-inv3-msg2* \cap *m2-inv1-nonces*))*
med21 m1 m2
by (*rule Refinement-using-invariants*) (*auto*)

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

$m3\text{-step1 } Ra \ A \ B \ Na \equiv \{(s, s1)\}.$

— guards

$Ra \notin \text{dom}(\text{runs } s) \wedge$

$Na = Ra\$0 \wedge$

— actions

$s1 = s\{$

$\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [])),$

$IK := \text{insert } \{\text{Agent } A, \text{Agent } B, \text{Nonce } Na\} (IK \ s) \quad \text{— send msg 1}$

$\}$

$\}$

definition

$m3\text{-step2} :: [\text{rid-t}, \text{agent}, \text{agent}, \text{nonce}, \text{nonce}] \Rightarrow (m3\text{-state} \times m3\text{-state}) \text{ set}$

where

$m3\text{-step2 } Rb \ A \ B \ Na \ Nb \equiv \{(s, s1)\}.$

— guards

$Rb \notin \text{dom}(\text{runs } s) \wedge$

$Nb = Rb\$0 \wedge$

$\{\text{Agent } A, \text{Agent } B, \text{Nonce } Na\} \in IK \ s \wedge \quad \text{— receive msg 1}$

— actions

$s1 = s\{$

$\text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [aNon \ Na])),$

— send msg 2

$IK := \text{insert } (\text{Crypt } (\text{priK } B) \ \{\text{Nonce } Nb, \text{Nonce } Na, \text{Agent } A\}) (IK \ s)$

$\}$

$\}$

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

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], []) \wedge$

$Na = Ra\$0 \wedge$

$\text{Crypt } (\text{priK } B) \ \{\text{Nonce } Nb, \text{Nonce } Na, \text{Agent } A\} \in IK \ s \wedge \quad \text{— recv msg 2}$

— actions

$s1 = s\{$

$\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [aNon \ Nb]))$

$\}$

$\}$

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

$m3-DY-fake \equiv \{(s, s1)\}.$

— 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**

$m3-trans \equiv (\cup A B Ra Rb Na Nb.$

$m3-step1 Ra A B Na \cup$

$m3-step2 Rb A B Na Nb \cup$

$m3-step3 Ra A B Na Nb \cup$

$m3-DY-fake \cup$

Id

)

definition

$m3 :: (m3-state, m3-obs) \text{ spec}$ **where**

$m3 \equiv (|$

$init = m3-init,$

$trans = m3-trans,$

$obs = m3-obs$

|)

lemmas $m3-defs =$

$m3-def m3-init-def m3-trans-def m3-obs-def$

$m3-step1-def m3-step2-def m3-step3-def$

$m3-DY-fake-def$

2.4.3 Invariants

Specialize injectiveness of parts to enable aggressive application.

lemmas $parts-Inj-IK = parts.Inj$ [**where** $H=IK s$ **for** s]

lemmas $analz-Inj-IK = analz.Inj$ [**where** $H=IK s$ **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

$m3\text{-inv1}\text{-lkeysec} :: m3\text{-state set}$ **where**
 $m3\text{-inv1}\text{-lkeysec} \equiv \{s. \forall A.$
 $Key (priK A) \in \text{analz} (IK s) \longrightarrow A \in \text{bad}$
}

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, rotated 1]

lemma $PO\text{-}m3\text{-inv1}\text{-lkeysec}\text{-init}$ [iff]:

$\text{init } m3 \subseteq m3\text{-inv1}\text{-lkeysec}$

by (*auto simp add: PO-hoare-def m3-defs intro!: m3-inv1-lkeysecI*)

lemma $PO\text{-}m3\text{-inv1}\text{-lkeysec}\text{-trans}$ [iff]:

$\{m3\text{-inv1}\text{-lkeysec}\} \text{trans } m3 \{> m3\text{-inv1}\text{-lkeysec}\}$

by (*auto simp add: PO-hoare-def m3-defs intro!: m3-inv1-lkeysecI*)

lemma $PO\text{-}m3\text{-inv1}\text{-lkeysec}$ [iff]: $\text{reach } m3 \subseteq m3\text{-inv1}\text{-lkeysec}$

by (*rule inv-rule-basic*) (*auto*)

inv2: Intruder knows long-term keys of bad guys

definition

$m3\text{-inv2}\text{-badkeys} :: m3\text{-state set}$

where

$m3\text{-inv2}\text{-badkeys} \equiv \{s. \forall C.$

$C \in \text{bad} \longrightarrow Key (priK C) \in \text{analz} (IK s)$

}

lemmas $m3\text{-inv2}\text{-badkeysI} =$

$m3\text{-inv2}\text{-badkeys}\text{-def}$ [THEN setc-def-to-intro, rule-format]

lemmas $m3\text{-inv2}\text{-badkeysE}$ [elim] =

$m3\text{-inv2}\text{-badkeys}\text{-def}$ [THEN setc-def-to-elim, rule-format]

lemmas $m3\text{-inv2}\text{-badkeysD}$ [dest] =

$m3\text{-inv2}\text{-badkeys}\text{-def}$ [THEN setc-def-to-dest, rule-format, rotated 1]

Invariance proof.

lemma $PO\text{-}m3\text{-inv2}\text{-badkeys}\text{-init}$ [iff]:

$\text{init } m3 \subseteq m3\text{-inv2}\text{-badkeys}$

by (*auto simp add: m3-defs m3-inv2-badkeys-def*)

lemma *PO-m3-inv2-badkeys-trans* [iff]:
 $\{m3\text{-inv2-badkeys}\} \text{ trans } m3 \{> m3\text{-inv2-badkeys}\}$
by (*auto simp add: PO-hoare-defs m3-defs intro!: m3-inv2-badkeysI*)

lemma *PO-m3-inv2-badkeys* [iff]: *reach* $m3 \subseteq m3\text{-inv2-badkeys}$
by (*rule inv-rule-basic*) (*auto*)

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

$m3\text{-inv3-pubkeys} \equiv \{s. \forall C.$
 $\text{Key } (\text{pubK } C) \in \text{analz } (IK \ s)$
 $\}$

lemmas $m3\text{-inv3-pubkeysI} =$
 $m3\text{-inv3-pubkeys-def}$ [*THEN setc-def-to-intro, rule-format*]
lemmas $m3\text{-inv3-pubkeysE}$ [*elim*] =
 $m3\text{-inv3-pubkeys-def}$ [*THEN setc-def-to-elim, rule-format*]
lemmas $m3\text{-inv3-pubkeysD}$ [*dest*] =
 $m3\text{-inv3-pubkeys-def}$ [*THEN setc-def-to-dest, rule-format, rotated 1*]

Invariance proof.

lemma *PO-m3-inv3-pubkeys-init* [iff]:
 $\text{init } m3 \subseteq m3\text{-inv3-pubkeys}$
by (*auto simp add: m3-defs m3-inv3-pubkeys-def*)

lemma *PO-m3-inv3-pubkeys-trans* [iff]:
 $\{m3\text{-inv3-pubkeys}\} \text{ trans } m3 \{> m3\text{-inv3-pubkeys}\}$
by (*auto simp add: PO-hoare-defs m3-defs intro!: m3-inv3-pubkeysI*)

lemma *PO-m3-inv3-pubkeys* [iff]: *reach* $m3 \subseteq m3\text{-inv3-pubkeys}$
by (*rule inv-rule-basic*) (*auto*)

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

$\text{nonces} :: \text{msg set} \Rightarrow \text{nonce set}$

where

$\text{nonces } H \equiv \{N. \text{Nonce } N \in \text{analz } H\}$

abbreviation

$ink :: chmsg\ set \Rightarrow nonce\ set$

where

$ink\ H \equiv \{N. aNon\ N \in extr\ ik0\ H\}$

Abstraction function on sets of messages.

inductive-set

$abs-msg :: msg\ set \Rightarrow chmsg\ set$

for $H :: msg\ set$

where

$am-M1:$

$\{\!| Agent\ A, Agent\ B, Nonce\ Na |\!\} \in H$

$\Longrightarrow Insec\ A\ B\ (Msg\ [aNon\ Na]) \in abs-msg\ H$

| $am-M2:$

$Crypt\ (priK\ B)\ \{\!| Nonce\ Nb, Nonce\ Na, Agent\ A |\!\} \in H$

$\Longrightarrow Auth\ B\ A\ (Msg\ [aNon\ Nb, aNon\ Na]) \in abs-msg\ H$

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

$R23-msgs :: (m2-state \times m3-state)\ set\ \mathbf{where}$

$R23-msgs \equiv \{(s, t). abs-msg\ (parts\ (IK\ t)) \subseteq chan\ s\} \quad \text{— with parts!}$

definition

$R23-ink :: (m2-state \times m3-state)\ set\ \mathbf{where}$

$R23-ink \equiv \{(s, t). nonces\ (IK\ t) \subseteq ink\ (chan\ s)\}$

definition

$R23-preserved :: (m2-state \times m3-state)\ set\ \mathbf{where}$

$R23-preserved \equiv \{(s, t). runs\ s = runs\ t\}$

definition

$R23 :: (m2-state \times m3-state)\ set\ \mathbf{where}$

$R23 \equiv R23-msgs \cap R23-ink \cap R23-preserved$

lemmas $R23-defs = R23-def\ R23-msgs-def\ R23-ink-def\ R23-preserved-def$

Mediator function: nothing new.

definition

$med32 :: m3-obs \Rightarrow m2-obs\ \mathbf{where}$

$med32 \equiv id$

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 $R23-inkI =$

$R23-ink-def\ [THEN\ rel-def-to-intro, simplified, rule-format]$

lemmas *R23-inkE* [elim] =
R23-ink-def [THEN *rel-def-to-elim*, *simplified*, *rule-format*]

lemmas *R23-preservedI* =
R23-preserved-def [THEN *rel-def-to-intro*, *simplified*, *rule-format*]

lemmas *R23-preservedE* [elim] =
R23-preserved-def [THEN *rel-def-to-elim*, *simplified*, *rule-format*]

lemmas *R23-intros* = *R23-msgsI* *R23-inkI* *R23-preservedI*

Facts about the abstraction function

declare *abs-msg.intros* [intro!]

declare *abs-msg.cases* [elim!]

lemma *abs-msg-empty*: *abs-msg* {} = {}
by (*auto*)

lemma *abs-msg-Un* [*simp*]:
abs-msg ($G \cup H$) = *abs-msg* $G \cup$ *abs-msg* H
by (*auto*)

lemma *abs-msg-mono* [elim]:
 $\llbracket m \in \text{abs-msg } G; G \subseteq H \rrbracket \implies m \in \text{abs-msg } H$
by (*auto*)

lemma *abs-msg-insert-mono* [intro]:
 $\llbracket m \in \text{abs-msg } H \rrbracket \implies m \in \text{abs-msg } (\text{insert } m' H)$
by (*auto*)

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*:
 $\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-ink}; t \in m3\text{-inv1-lkeysec} \rrbracket$
 $\implies \text{abs-msg } (\text{synth } (\text{analz } (IK t))) \subseteq \text{fake ik0 } (\text{dom } (\text{runs } s)) (\text{chan } s)$
apply (*auto*)
apply (*rule fake-intros*, *auto*)
apply (*rule fake-Inj*, *auto*)
apply (*rule fake-intros*, *auto*)
done

lemma *absmsg-parts-subset-fakeable*:
 $\llbracket (s, t) \in R23\text{-msgs} \rrbracket$
 $\implies \text{abs-msg } (\text{parts } (IK t)) \subseteq \text{fake ik0 } (-\text{dom } (\text{runs } s)) (\text{chan } s)$
by (*rule-tac* $B=\text{chan } s$ **in** *subset-trans*) (*auto*)

declare *abs-msg-DY-subset-fakeable* [*simp*, *intro!*]

declare *absmsg-parts-subset-fakeable* [*simp*, *intro!*]

Refinement proof

lemma *PO-m3-step1-refines-m2-step1*:
{*R23*}

$(m2\text{-step1 } Ra \ A \ B \ Na), (m3\text{-step1 } Ra \ A \ B \ Na)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)
(auto)

lemma *PO-m3-step2-refines-m2-step2:*
 $\{R23 \cap UNIV \times (m3\text{-inv2-badkeys})\}$
 $(m2\text{-step2 } Rb \ A \ B \ Na \ Nb), (m3\text{-step2 } Rb \ A \ B \ Na \ Nb)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)
(auto)

lemma *PO-m3-step3-refines-m2-step3:*
 $\{R23\}$
 $(m2\text{-step3 } Ra \ A \ B \ Na \ Nb), (m3\text{-step3 } Ra \ A \ B \ Na \ Nb)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)
(auto)

The Dolev-Yao fake event refines the abstract fake event.

lemma *PO-m3-DY-fake-refines-m2-fake:*
 $\{R23 \cap UNIV \times (m3\text{-inv1-lkeysec} \cap m3\text{-inv2-badkeys})\}$
 $m2\text{-fake}, m3\text{-DY-fake}$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs*)
(rule R23-intros, auto)+

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-DY-fake-refines-m2-fake

lemma *PO-m3-refines-init-m2 [iff]:*
 $init \ m3 \subseteq R23''(init \ m2)$
by (*auto simp add: R23-defs m2-defs m3-defs*)

lemma *PO-m3-refines-trans-m2 [iff]:*
 $\{R23 \cap UNIV \times (m3\text{-inv2-badkeys} \cap m3\text{-inv1-lkeysec})\}$
 $(trans \ m2), (trans \ m3)$
 $\{> R23\}$
apply (*auto simp add: m3-def m3-trans-def m2-def m2-trans-def*)
apply (*blast intro!: PO-m3-trans-refines-m2-trans*)
done

lemma *PO-obs-consistent [iff]:*
 $obs\text{-consistent } R23 \ med32 \ m2 \ m3$
by (*auto simp add: obs-consistent-def R23-def med32-def m2-defs m3-defs*)

lemma *PO-m3-refines-m2:*
 $refines$
 $(R23 \cap UNIV \times (m3\text{-inv2-badkeys} \cap m3\text{-inv1-lkeysec}))$
 $med32 \ m2 \ m3$

by (rule *Refinement-using-invariants*) (auto)

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* \Rightarrow *m3-obs* **where**
m3-obs *s* \equiv (
 runs = *runs s*
)

2.5.2 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:
 $s1 = s\{$
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [])),$
 $IK := insert\ (Crypt\ (pubK\ B)\ \{\{Nonce\ Na, Agent\ A\}\})\ (IK\ s)$
 $\}$
 $\}$

definition

$m3\text{-step2} :: [rid\text{-}t, agent, agent, nonce, nonce] \Rightarrow (m3\text{-state} \times m3\text{-state})\ set$

where

$m3\text{-step2}\ Rb\ A\ B\ Na\ Nb \equiv \{(s, s1)\}.$

— guards
 $Rb \notin dom\ (runs\ s) \wedge$
 $Nb = Rb\$0 \wedge$
 $Crypt\ (pubK\ B)\ \{\{Nonce\ Na, Agent\ A\}\} \in IK\ s \wedge$ — receive msg 1

— actions
 $s1 = s\{$
 $runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aNon\ Na])),$
 $IK := insert\ (Crypt\ (pubK\ A)\ \{\{Nonce\ Na, Nonce\ Nb, Agent\ B\}\})\ (IK\ s)$
 $\}$
 $\}$

definition

$m3\text{-step3} :: [rid\text{-}t, agent, agent, nonce, nonce] \Rightarrow (m3\text{-state} \times m3\text{-state})\ set$

where

$m3\text{-step3}\ Ra\ A\ B\ Na\ Nb \equiv \{(s, s1)\}.$

— guards
 $runs\ s\ Ra = Some\ (Init, [A, B], []) \wedge$
 $Na = Ra\$0 \wedge$
 $Crypt\ (pubK\ A)\ \{\{Nonce\ Na, Nonce\ Nb, Agent\ B\}\} \in IK\ s \wedge$ — recv msg2

— actions
 $s1 = s\{$
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aNon\ Nb]))$
 $\}$
 $\}$

Standard Dolev-Yao intruder.

definition

$m3\text{-DY-fake} :: (m3\text{-state} \times m3\text{-state})\ set$

where

$m3\text{-DY-fake} \equiv \{(s, s1)\}.$

— actions:
 $s1 = s\{ IK := synth\ (analz\ (IK\ s))\ \}$
 $\}$

Transition system.

definition

$m3\text{-init} :: m3\text{-state set}$

where

$m3\text{-init} \equiv \{ \langle \rangle$
 $runs = Map.empty,$
 $IK = (Key'priK'bad) \cup (Key'range pubK) \cup (Key'shrK'bad)$
 $\rangle \}$

definition

$m3\text{-trans} :: (m3\text{-state} \times m3\text{-state}) set$ **where**

$m3\text{-trans} \equiv (\cup A B Ra Rb Na Nb.$
 $m3\text{-step1 } Ra A B Na \cup$
 $m3\text{-step2 } Rb A B Na Nb \cup$
 $m3\text{-step3 } Ra A B Na Nb \cup$
 $m3\text{-DY-fake} \cup$
 Id
 $)$

definition

$m3 :: (m3\text{-state}, m3\text{-obs}) spec$ **where**

$m3 \equiv \langle \rangle$
 $init = m3\text{-init},$
 $trans = m3\text{-trans},$
 $obs = m3\text{-obs}$
 \rangle

lemmas $m3\text{-defs} =$

$m3\text{-def } m3\text{-init-def } m3\text{-trans-def } m3\text{-obs-def}$
 $m3\text{-step1-def } m3\text{-step2-def } m3\text{-step3-def}$
 $m3\text{-DY-fake-def}$

2.5.3 Invariants

Automatic tool tuning. Tame too-aggressive pair decomposition, which is declared as a safe elim rule ([elim!]).

lemmas $M\text{Pair-parts}$ [rule del, elim]

lemmas $M\text{Pair-analz}$ [rule del, elim]

Specialize injectiveness of $parts$ and $analz$ to enable aggressive application.

lemmas $parts\text{-Inj-}IK = parts.Inj$ [where $H=IK$ s for s]

lemmas $analz\text{-Inj-}IK = analz.Inj$ [where $H=IK$ s for s]

declare $analz\text{-into-parts}$ [dest]

inv1: Key secrecy

Decryption keys are secret, that is, the intruder only knows private keys of corrupted agents.

definition

$m3\text{-inv1-keys} :: m3\text{-state set}$ **where**
 $m3\text{-inv1-keys} \equiv \{s. \forall A.$

$Key (priK A) \in parts (IK s) \longrightarrow A \in bad$
 $\}$

lemmas $m3\text{-inv1}\text{-keys}I = m3\text{-inv1}\text{-keys}\text{-def} [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m3\text{-inv1}\text{-keys}E [elim] =$
 $m3\text{-inv1}\text{-keys}\text{-def} [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m3\text{-inv1}\text{-keys}D [dest] =$
 $m3\text{-inv1}\text{-keys}\text{-def} [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

lemma $PO\text{-}m3\text{-inv1}\text{-keys}\text{-init} [iff]:$

$init\ m3 \subseteq m3\text{-inv1}\text{-keys}$

by (*auto simp add: PO-hoare-def m3-defs intro!: m3-inv1-keysI*)

lemma $PO\text{-}m3\text{-inv1}\text{-keys}\text{-trans} [iff]:$

$\{m3\text{-inv1}\text{-keys}\} trans\ m3 \{>\ m3\text{-inv1}\text{-keys}\}$

by (*auto simp add: PO-hoare-def m3-defs intro!: m3-inv1-keysI*)
(auto)

lemma $PO\text{-}m3\text{-inv1}\text{-keys} [iff]: reach\ m3 \subseteq m3\text{-inv1}\text{-keys}$

by (*rule inv-rule-basic, auto*)

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

$abs\text{-msg} :: msg\ set \Rightarrow chmsg\ set$

for $H :: msg\ set$

where

$am\text{-msg}1:$

$Crypt (pubK B) \{Nonce\ Na,\ Agent\ A\} \in H$

$\Longrightarrow Confid\ A\ B (Msg [aNon\ Na]) \in abs\text{-msg}\ H$

| $am\text{-msg}2:$

$Crypt (pubK A) \{Nonce\ Na,\ Nonce\ Nb,\ Agent\ B\} \in H$

$\Longrightarrow Confid\ B\ A (Msg [aNon\ Na,\ aNon\ Nb]) \in abs\text{-msg}\ H$

declare $abs\text{-msg.intros} [intro!]$

declare $abs\text{-msg.cases} [elim!]$

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

$R23\text{-msgs} :: (m2\text{-state} \times m3\text{-state})\ set\ \mathbf{where}$

$R23\text{-msgs} \equiv \{(s, t). abs\text{-msg} (parts (IK\ t)) \subseteq chan\ s\} \quad \text{— with } parts!$

definition

$R23\text{-non} :: (m2\text{-state} \times m3\text{-state})\ set\ \mathbf{where}$

$R23\text{-non} \equiv \{(s, t). \forall N. \text{Nonce } N \in \text{analz } (IK\ t) \longrightarrow a\text{Non } N \in \text{extr } ik0\ (chan\ s)\}$

definition

$R23\text{-pres} :: (m2\text{-state} \times m3\text{-state})\ \text{set}\ \mathbf{where}$

$R23\text{-pres} \equiv \{(s, t). \text{runs } s = \text{runs } t\}$

definition

$R23 :: (m2\text{-state} \times m3\text{-state})\ \text{set}\ \mathbf{where}$

$R23 \equiv R23\text{-msgs} \cap R23\text{-non} \cap R23\text{-pres}$

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

$med32 :: m3\text{-obs} \Rightarrow m2\text{-obs}\ \mathbf{where}$

$med32 \equiv id$

2.5.5 Misc lemmas

General facts about $abs\text{-msg}$

lemma $abs\text{-msg-empty}$: $abs\text{-msg } \{\} = \{\}$

by $(auto)$

lemma $abs\text{-msg-Un}$ $[simp]$:

$abs\text{-msg } (G \cup H) = abs\text{-msg } G \cup abs\text{-msg } H$

by $(auto)$

lemma $abs\text{-msg-mono}$ $[elim]$:

$\llbracket m \in abs\text{-msg } G; G \subseteq H \rrbracket \Longrightarrow m \in abs\text{-msg } H$

by $(auto)$

lemma $abs\text{-msg-insert-mono}$ $[intro]$:

$\llbracket m \in \text{abs-msg } H \rrbracket \implies m \in \text{abs-msg } (\text{insert } m' H)$
by (*auto*)

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*:
 $\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-non}; t \in m3\text{-inv1-keys} \rrbracket$
 $\implies \text{abs-msg } (\text{synth } (\text{analz } (IK t))) \subseteq \text{fake ik0 } (\text{dom } (\text{runs } s)) (\text{chan } s)$
apply (*auto*)
apply (*rule fake-Inj, fastforce*)
apply (*rule fake-intros, auto*)
apply (*rule fake-Inj, fastforce*)
apply (*rule fake-intros, auto*)
done

lemma *abs-msg-parts-subset-fake*:
 $\llbracket (s, t) \in R23\text{-msgs} \rrbracket$
 $\implies \text{abs-msg } (\text{parts } (IK t)) \subseteq \text{fake ik0 } (-\text{dom } (\text{runs } s)) (\text{chan } s)$
by (*rule-tac B=chan s in subset-trans*) (*auto*)

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*:
 $\{R23 \cap UNIV \times m3\text{-inv1-keys}\}$
 $(m2\text{-step1 } Ra A B Na), (m3\text{-step1 } Ra A B Na)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)
(*auto*)

lemma *PO-m3-step2-refines-m2-step2*:
 $\{R23 \cap UNIV \times m3\text{-inv1-keys}\}$
 $(m2\text{-step2 } Rb A B Na Nb), (m3\text{-step2 } Rb A B Na Nb)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)
(*auto*)

lemma *PO-m3-step3-refines-m2-step3*:
 $\{R23\}$
 $(m2\text{-step3 } Ra A B Na Nb), (m3\text{-step3 } Ra A B Na Nb)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)

Dolev-Yao fake event refines abstract fake event.

lemma *PO-m3-DY-fake-refines-m2-fake*:
 $\{R23 \cap UNIV \times m3\text{-inv1-keys}\}$
 $(m2\text{-fake}), (m3\text{-DY-fake})$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs*)

```

(rule R23-intros, auto)+

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-DY-fake-refines-m2-fake

lemma PO-m3-refines-init-m2 [iff]:
  init m3  $\subseteq$  R23“(init m2)
by (auto simp add: R23-defs m2-defs m3-defs)

lemma PO-m3-refines-trans-m2 [iff]:
  {R23  $\cap$  UNIV  $\times$  m3-inv1-keys}
  (trans m2), (trans m3)
  {> R23}
apply (auto simp add: m3-def m3-trans-def m2-def m2-trans-def)
apply (blast intro!: PO-m3-trans-refines-m2-trans)+
done

lemma PO-R23-obs-consistent [iff]:
  obs-consistent R23 med32 m2 m3
by (auto simp add: obs-consistent-def R23-def m2-defs m3-defs)

lemma PO-m3-refines-m2 [iff]:
  refines
  (R23  $\cap$  UNIV  $\times$  m3-inv1-keys)
  med32 m2 m3
by (rule Refinement-using-invariants) (auto)

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*
by (*auto elim: knC.cases*)

lemma *azC-empty* [*simp*]: *azC Map.empty* = *keySetup*
by (*auto elim: azC.cases*)

azC and run abstraction

lemma *azC-map-runs* [*simp*]: *azC (map-runs h runz)* = *azC runz*
by (*auto simp add: map-runs-def elim!: azC.cases*)

Update lemmas for *knC*

lemma *knC-upd-Init-Resp-None*:
[[*R* \notin *dom runz*; *rol* \in {*Init*, *Resp*}]]
 \implies *knC (runz(R* \mapsto (*rol*, [*A*, *B*], []))) = *knC runz*
by (*fastforce simp add: domIff elim!: knC.cases*)

lemma *knC-upd-Init-Some*:
[[*runz Ra* = *Some (Init, [A, B], [])*]]
 \implies *knC (runz(Ra* \mapsto (*Init*, [*A*, *B*], [*aKey Kab*]))) = *insert (Kab, A) (knC runz)*
apply (*auto elim!: knC.cases*)
— 3 subgoals
apply (*rename-tac Raa Aa Ba K al, rule-tac A=Aa and B=Ba and al=al in knC-init, auto*)
apply (*rename-tac Rb Aa Ba K al, rule-tac A=Aa and B=Ba and al=al in knC-resp, auto*)
apply (*rule-tac knC-serv, auto*)
done

lemma *knC-upd-Resp-Some*:
[[*runz Ra* = *Some (Resp, [A, B], [])*]]
 \implies *knC (runz(Ra* \mapsto (*Resp*, [*A*, *B*], [*aKey Kab*]))) = *insert (Kab, B) (knC runz)*
apply (*auto elim!: knC.cases*)
— 3 subgoals
apply (*rename-tac Raa Aa Ba K al, rule-tac A=Aa and B=Ba and al=al in knC-init, auto*)
apply (*rename-tac Raa Aa Ba K al, rule-tac A=Aa and B=Ba and al=al in knC-resp, auto*)
apply (*rule-tac knC-serv, auto*)
done

lemma *knC-upd-Server*:
[[*Rs* \notin *dom runz*]]
 \implies *knC (runz(Rs* \mapsto (*Serv*, [*A*, *B*], []))) = *insert (sesK (Rs\$sk), Sv) (knC runz)*
apply (*auto elim!: knC.cases*)
— 2 subgoals
apply (*rename-tac Raa Aa Ba K al, rule-tac A=Aa and B=Ba in knC-init, auto dest: dom-lemmas*)
apply (*rename-tac Raa Aa Ba K al, rule-tac A=Aa and B=Ba in knC-resp, auto dest: dom-lemmas*)
done

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*:

$\llbracket Ra \notin \text{dom runz} \rrbracket$

$\implies \text{azC} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], []))) = \text{azC runz}$

by (*auto simp add: azC.simps elim!: azC.cases dest: dom-lemmas*)

lemma *azC-upd-Resp-None*:

$\llbracket Rb \notin \text{dom runz} \rrbracket$

$\implies \text{azC} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], []))) = \text{azC runz}$

by (*auto simp add: azC.simps elim!: azC.cases dest: dom-lemmas*)

lemma *azC-upd-Init-Some*:

$\llbracket \text{runz } Ra = \text{Some} (\text{Init}, [A, B], []) \rrbracket$

$\implies \text{azC} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], al))) = \text{azC runz}$

apply (*auto elim!: azC.cases*)

— 5 subgoals

apply (*rule-tac azC-good, auto*)

apply (*rule-tac azC-good, auto*)

apply (*rule-tac azC-good, auto*)

apply (*rule-tac azC-bad, auto*)+

done

lemma *azC-upd-Resp-Some*:

$\llbracket \text{runz } Rb = \text{Some} (\text{Resp}, [A, B], []) \rrbracket$

$\implies \text{azC} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], al))) = \text{azC runz}$

apply (*auto elim!: azC.cases*)

— 5 subgoals

apply (*rule-tac azC-good, auto*)

apply (*rule-tac azC-good, auto*)

apply (*rule-tac azC-good, auto*)

apply (*rule-tac azC-bad, auto*)+

done

lemma *azC-upd-Serv-bad*:

$\llbracket Rs \notin \text{dom runz}; A \in \text{bad} \vee B \in \text{bad} \rrbracket$

$\implies \text{azC} (\text{runz}(Rs \mapsto (\text{Serv}, [A, B], al))) = \text{azC runz} \cup \{\text{sesK } (Rs\$sk)\} \times \text{UNIV}$

apply (*auto elim!: azC.cases*)

— 10 subgoals

apply (

rename-tac Rsa Aa Ba ala, rule-tac A=Aa and B=Ba and al=ala in azC-good, auto dest: dom-lemmas,

rename-tac Rsa Aa Ba ala, rule-tac A=Aa and B=Ba and al=ala in azC-good, auto dest: dom-lemmas,

rename-tac Rsa Aa Ba ala, rule-tac A=Aa and B=Ba and al=ala in azC-good, auto dest: dom-lemmas,

rename-tac Rsa Aa Ba ala C, rule-tac A=Aa and B=Ba and al=ala in azC-bad, auto dest:

dom-lemmas,

rename-tac Rsa Aa Ba ala C, rule-tac A=Aa and B=Ba and al=ala in azC-bad, auto dest:

dom-lemmas

)+

done

lemma *azC-upd-Serv-good*:

$\llbracket Rs \notin \text{dom runz}; K = \text{sesK } (Rs\$sk); A \notin \text{bad}; B \notin \text{bad} \rrbracket$

$\implies \text{azC} (\text{runz}(Rs \mapsto (\text{Serv}, [A, B], al)))$

$= \text{azC runz} \cup \{(K, A), (K, B), (K, Sv)\}$

apply (*auto elim!: azC.cases*)

— 5 subgoals

apply (
rename-tac Rsa Aa Ba ala, rule-tac A=Aa and B=Ba and al=ala in azC-good, auto dest: dom-lemmas,
rename-tac Rsa Aa Ba ala, rule-tac A=Aa and B=Ba and al=ala in azC-good, auto dest: dom-lemmas,
rename-tac Rsa Aa Ba ala, rule-tac A=Aa and B=Ba and al=ala in azC-good, auto dest: dom-lemmas,
rename-tac Rsa Aa Ba ala C, rule-tac A=Aa and B=Ba and al=ala in azC-bad, auto dest:
dom-lemmas,
rename-tac Rsa Aa Ba ala C, rule-tac A=Aa and B=Ba and al=ala in azC-bad, auto dest:
dom-lemmas
)+
done

lemma *azC-upd-Serv*:

$\llbracket Rs \notin \text{dom runs}; K = \text{ses}K (Rs\$sk) \rrbracket$
 $\implies \text{az}C (\text{runz}(Rs \mapsto (\text{Serv}, [A, B], al))) =$
 $\text{az}C \text{runz} \cup \{K\} \times (\text{if } A \notin \text{bad} \wedge B \notin \text{bad} \text{ then } \{A, B, Sv\} \text{ else UNIV})$

by (*simp add: azC-upd-Serv-bad azC-upd-Serv-good*)

lemmas *azC-upd-lemmas [simp] =*

azC-upd-Init-None azC-upd-Resp-None
azC-upd-Init-Some azC-upd-Resp-Some azC-upd-Serv

3.1.2 Events

definition — by *A*, refines skip

m1x-step1 :: [*rid-t, agent, agent*] \Rightarrow '*x m1r-trans*

where

m1x-step1 Ra A B $\equiv \{(s, s1)\}$.

— guards:

$Ra \notin \text{dom} (\text{runs } s) \wedge$ — *Ra* 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-step2 :: [*rid-t, agent, agent*] \Rightarrow '*x m1r-trans*

where

m1x-step2 Rb A B $\equiv \{(s, s1)\}$.

— guards:

$Rb \notin \text{dom} (\text{runs } s) \wedge$ — *Rb* is fresh

— actions:

— create responder thread

$s1 = s \langle \text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [])) \rangle$
 }

definition — by *Sv*, refines *s0g-gen*

m1x-step3 :: [*rid-t, agent, agent, key*] \Rightarrow '*x m1r-trans*

where

$m1x\text{-step}3 \text{ } Rs \ A \ B \ Kab \equiv \{(s, s1)\}.$

— 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 $s0g\text{-learn}$
 $m1x\text{-step}4 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}] \Rightarrow 'x \text{ } m1x\text{-trans}$

where

$m1x\text{-step}4 \text{ } Ra \ A \ B \ Kab \equiv \{(s, s1)\}.$
 — guards:
 $\text{runs } s \text{ } Ra = \text{Some}(\text{Init}, [A, B], []) \wedge$
 $(Kab \notin \text{leak } s \longrightarrow (Kab, A) \in \text{azC}(\text{runs } s)) \wedge$ — authorization guard

— actions:
 $s1 = s \{ \text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [\text{aKey } Kab])) \}$

definition — by B , refines $s0g\text{-learn}$
 $m1x\text{-step}5 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}] \Rightarrow 'x \text{ } m1x\text{-trans}$

where

$m1x\text{-step}5 \text{ } Rb \ A \ B \ Kab \equiv \{(s, s1)\}.$
 — guards:
 $\text{runs } s \text{ } Rb = \text{Some}(\text{Resp}, [A, B], []) \wedge$
 $(Kab \notin \text{leak } s \longrightarrow (Kab, B) \in \text{azC}(\text{runs } s)) \wedge$ — authorization guard

— actions:
 $s1 = s \{ \text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [\text{aKey } Kab])) \}$

definition — by attacker, refines $s0g\text{-leak}$

$m1x\text{-leak} :: \text{rid-}t \Rightarrow 'x \text{ } m1x\text{-trans}$

where

$m1x\text{-leak } Rs \equiv \{(s, s1)\}.$
 — guards:
 $Rs \in \text{dom}(\text{runs } s) \wedge$
 $\text{fst}(\text{the}(\text{runs } s \text{ } Rs)) = \text{Serv} \wedge$ — compromise server run Rs

— actions:
 $s1 = s \{ \text{leak} := \text{insert}(\text{sesK}(Rs\$sk))(\text{leak } s) \}$

3.1.3 Specification

definition

$m1x\text{-init} :: m1x\text{-state set}$

where

$m1x\text{-init} \equiv \{ \{ \}$
 $\text{runs} = \text{Map.empty},$

$leak = corrKey$ — statically corrupted keys initially leaked
 $\})$

definition

$m1x-trans :: 'x m1x-trans$ **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

$m1x :: (m1x-state, m1x-obs) spec$ **where**
 $m1x \equiv (\$
 $init = m1x-init,$
 $trans = m1x-trans,$
 $obs = id$
 $\)$

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 $m1x-obs-id [simp]: obs m1x = id$
by ($simp add: m1x-def$)

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 $m1x-inv1-keyI = m1x-inv1-key-def [THEN setc-def-to-intro, rule-format]$

lemmas $m1x-inv1-keyE [elim] =$
 $m1x-inv1-key-def [THEN setc-def-to-elim, rule-format]$

lemmas $m1x-inv1-keyD [dest] =$
 $m1x-inv1-key-def [THEN setc-def-to-dest, rule-format, rotated 1]$

Invariance proof.

lemma *PO-m1x-inv1-key-init* [iff]:
 $init\ m1x \subseteq m1x\text{-inv1}\text{-key}$
by (*auto simp add: m1x-defs m1x-inv1-key-def*)

lemma *PO-m1x-inv1-key-trans* [iff]:
 $\{m1x\text{-inv1}\text{-key}\ trans\ m1x\ \{>\ m1x\text{-inv1}\text{-key}\}$
by (*auto simp add: PO-hoare-defs m1x-defs intro!: m1x-inv1-keyI*)

lemma *PO-m1x-inv1-key* [iff]: $reach\ m1x \subseteq m1x\text{-inv1}\text{-key}$
by (*rule inv-rule-basic*) (*auto*)

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-m1x-step1-refines-skip*:
 $\{R01x\}$
 $Id, (m1x\text{-step1}\ Ra\ A\ B)$
 $\{>\ R01x\}$
by (*auto simp add: PO-rhoare-defs R01x-defs s0g-defs m1x-defs*)

lemma *PO-m1x-step2-refines-skip*:
 $\{R01x\}$
 $Id, (m1x\text{-step2}\ Rb\ A\ B)$
 $\{>\ R01x\}$
by (*auto simp add: PO-rhoare-defs R01x-defs s0g-defs m1x-defs*)

lemma *PO-m1x-step3-refines-s0g-gen*:
 $\{R01x \cap UNIV \times m1x\text{-inv1}\text{-key}\}$
 $(s0g\text{-gen}\ Kab\ Sv\ \{Sv, A, B\}), (m1x\text{-step3}\ Rs\ A\ B\ Kab)$
 $\{>\ R01x\}$
by (*auto simp add: PO-rhoare-defs R01x-defs s0g-defs m1x-defs*)

lemma *PO-m1x-step4-refines-s0g-learn*:
 $\{R01x\}$
 $(s0g\text{-learn}\ Kab\ A), (m1x\text{-step4}\ Ra\ A\ B\ Kab)$
 $\{>\ R01x\}$

by (*auto simp add: PO-rhoare-defs R01x-defs s0g-defs m1x-defs*)

lemma *PO-m1x-step5-refines-s0g-learn*:

$\{R01x\}$
(*s0g-learn Kab B*), (*m1x-step5 Rb A B Kab*)
 $\{> R01x\}$

by (*auto simp add: PO-rhoare-defs R01x-defs s0g-defs m1x-defs*)

lemma *PO-m1x-leak-refines-s0g-leak*:

$\{R01x\}$
(*s0g-leak (sesK (Rs\$sk))*), (*m1x-leak Rs*)
 $\{> R01x\}$

by (*fastforce simp add: PO-rhoare-defs R01x-defs s0g-defs m1x-defs*)

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*“(*init s0g*)

by (*auto simp add: R01x-defs s0g-defs m1x-defs intro!: s0g-secrecyI s0g-domI*)

lemma *PO-m1x-refines-trans-s0g [iff]*:

$\{R01x \cap UNIV \times m1x-inv1-key\}$
(*trans s0g*), (*trans m1x*)
 $\{> R01x\}$

by (*auto simp add: m1x-def m1x-trans-def s0g-def s0g-trans-def*
intro!: PO-m1x-trans-refines-s0g-trans)

Observation consistency.

lemma *obs-consistent-med01x [iff]*:

obs-consistent R01x med01x s0g m1x

by (*auto simp add: obs-consistent-def R01x-defs s0g-def m1x-def*)

Refinement result.

lemma *PO-m1x-refines-s0g [iff]*:

refines
(*R01x* \cap *UNIV* \times *m1x-inv1-key*)
med01x s0g m1x

by (*rule Refinement-using-invariants*) (*auto del: subsetI*)

lemma *m1x-implements-s0g [iff]*: *implements med01x s0g m1x*

by (*rule refinement-soundness*) (*fast*)

3.1.6 Derived invariants

inv2: Secrecy

Secrecy, expressed in terms of runs.

definition

```

    m1x-secrecy :: 'x m1x-pred
where
    m1x-secrecy ≡ {s. knC (runs s) ⊆ azC (runs s) ∪ leak s × UNIV}

lemmas m1x-secrecyI = m1x-secrecy-def [THEN setc-def-to-intro, rule-format]
lemmas m1x-secrecyE [elim] = m1x-secrecy-def [THEN setc-def-to-elim, rule-format]

Invariance proof.

lemma PO-m1x-obs-secrecy [iff]: oreach m1x ⊆ m1x-secrecy
apply (rule external-invariant-translation [OF PO-s0g-obs-secrecy - m1x-implements-s0g])
apply (auto simp add: med01x-def m1x-secrecy-def s0g-secrecy-def)
done

lemma PO-m1x-secrecy [iff]: reach m1x ⊆ m1x-secrecy
by (rule external-to-internal-invariant [OF PO-m1x-obs-secrecy], auto)

end

```

3.2 Abstract (i/n)-authenticated key transport (L1)

```

theory m1-keydist-irm 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]

```

```

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-state = m1x-state
type-synonym m1a-obs = m1x-obs

```

```

type-synonym 'x m1a-pred = 'x m1x-pred
type-synonym 'x m1a-trans = 'x m1x-trans

```

We need some parameters regarding the list of freshness values stored by the server. These should be defined in further refinements.

consts

$is-len :: nat$ — num of agreeing list elements for initiator-server
 $rs-len :: nat$ — num of agreeing list elements for responder-server

3.2.2 Events

definition — by A , refines $m1x-step1$

$m1a-step1 :: [rid-t, agent, agent, nonce] \Rightarrow 'x m1r-trans$

where

$m1a-step1 Ra A B Na \equiv \{(s, s1)\}.$

— guards:

$Ra \notin dom (runs s) \wedge$ — Ra is fresh
 $Na = Ra\$na \wedge$ — NEW: generate a nonce

— actions:

— create initiator thread

$s1 = s \langle runs := (runs s)(Ra \mapsto (Init, [A, B], [])) \rangle$
 $\}$

definition — by B , refines $m1x-step2$

$m1a-step2 :: [rid-t, agent, agent] \Rightarrow 'x m1r-trans$

where

$m1a-step2 \equiv m1x-step2$

definition — by Sv , refines $m1x-step3$

$m1a-step3 :: [rid-t, agent, agent, key, nonce, atom list] \Rightarrow 'x m1r-trans$

where

$m1a-step3 Rs A B Kab Na al \equiv \{(s, s1)\}.$

— guards:

$Rs \notin dom (runs s) \wedge$ — fresh run id
 $Kab = sesK (Rs\$sk) \wedge$ — generate session key

— actions:

$s1 = s \langle runs := (runs s)(Rs \mapsto (Serv, [A, B], aNon Na \# al)) \rangle$
 $\}$

definition — by A , refines $m1x-step4$

$m1a-step4 :: [rid-t, agent, agent, nonce, key, atom list] \Rightarrow 'x m1a-trans$

where

$m1a-step4 Ra A B Na Kab nla \equiv \{(s, s')\}.$

— 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 $isl = take is-len nla$; injectiveness by including Na

$(A \notin bad \longrightarrow (\exists Rs. Kab = sesK (Rs\$sk) \wedge$
 $runs s Rs = Some (Serv, [A, B], aNon Na \# take is-len nla))) \wedge$

— actions:

$s' = s \langle runs := (runs s)(Ra \mapsto (Init, [A, B], aKey Kab \# nla)) \rangle$
 $\}$

definition — by B , refines $m1x\text{-step5}$
 $m1a\text{-step5} :: [rid\text{-}t, agent, key, atom\ list] \Rightarrow 'x\ m1a\text{-trans}$

where

$m1a\text{-step5}\ Rb\ A\ B\ Kab\ nlb \equiv \{(s, s1).$
 — guards:
 $runs\ s\ Rb = Some\ (Resp, [A, B], []) \wedge$
 $(Kab \notin leak\ s \longrightarrow (Kab, B) \in azC\ (runs\ s)) \wedge$ — authorization guard
 — guard for showing agreement with server on (Kab, A, rsl) ,
 — where $rsl = take\ rs\text{-}len\ nlb$; this agreement is non-injective
 $(B \notin bad \longrightarrow (\exists\ Rs\ Na.\ Kab = sesK\ (Rs\$sk) \wedge$
 $runs\ s\ Rs = Some\ (Serv, [A, B], aNon\ Na\ \# take\ rs\text{-}len\ nlb))) \wedge$
 — actions:
 $s1 = s \lfloor runs := (runs\ s)(Rb \mapsto (Resp, [A, B], aKey\ Kab\ \# nlb)) \rfloor$
 $\}$

definition — by attacker, refines $m1x\text{-leak}$

$m1a\text{-leak} :: rid\text{-}t \Rightarrow 'x\ m1x\text{-trans}$

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

$m1a\text{-trans} :: 'x\ m1a\text{-trans}$ **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

$m1a :: (m1a\text{-state}, m1a\text{-obs})\ spec$ **where**

$m1a \equiv \langle$
 $init = m1a\text{-init},$
 $trans = m1a\text{-trans},$
 $obs = id$
 \rangle

lemma $init\text{-}m1a: init\ m1a = m1a\text{-init}$

by ($simp\ add: m1a\text{-def}$)

lemma *trans-m1a*: $\text{trans } m1a = m1a\text{-trans}$
by (*simp add: m1a-def*)

lemma *obs-m1a* [*simp*]: $\text{obs } m1a = \text{id}$
by (*simp add: m1a-def*)

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 $\equiv \{s. \text{finite } (\text{dom } (\text{runs } s))\}$

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*

by (*auto simp add: m1a-defs intro!: m1a-inv0-finI*)

lemma *PO-m1a-inv0-fin-trans* [*iff*]:

$\{m1a\text{-inv0-fin}\} \text{trans } m1a \{> m1a\text{-inv0-fin}\}$

by (*auto simp add: PO-hoare-defs m1a-defs intro!: m1a-inv0-finI*)

lemma *PO-m1a-inv0-fin* [*iff*]: *reach m1a* \subseteq *m1a-inv0-fin*

by (*rule inv-rule-incr, auto del: subsetI*)

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 $PO-m1a-step1-refines-m1x-step1:$

$\{R1x1a\}$
 $(m1x-step1\ Ra\ A\ B), (m1a-step1\ Ra\ A\ B\ Na)$
 $\{>\ R1x1a\}$

by $(auto\ simp\ add: PO-rhoare-defs\ R1x1a-defs\ m1a-defs)$

lemma $PO-m1a-step2-refines-m1x-step2:$

$\{R1x1a\}$
 $(m1x-step2\ Rb\ A\ B), (m1a-step2\ Rb\ A\ B)$
 $\{>\ R1x1a\}$

by $(auto\ simp\ add: PO-rhoare-defs\ R1x1a-defs\ m1a-defs)$

lemma $PO-m1a-step3-refines-m1x-step3:$

$\{R1x1a\}$
 $(m1x-step3\ Rs\ A\ B\ Kab), (m1a-step3\ Rs\ A\ B\ Kab\ Na\ nls)$
 $\{>\ R1x1a\}$

by $(auto\ simp\ add: PO-rhoare-defs\ R1x1a-defs\ m1a-defs)$

lemma $PO-m1a-step4-refines-m1x-step4:$

$\{R1x1a\}$
 $(m1x-step4\ Ra\ A\ B\ Kab), (m1a-step4\ Ra\ A\ B\ Na\ Kab\ nla)$
 $\{>\ R1x1a\}$

by $(auto\ simp\ add: PO-rhoare-defs\ R1x1a-defs\ m1a-defs\ map-runs-def)$

lemma $PO-m1a-step5-refines-m1x-step5:$

$\{R1x1a\}$
 $(m1x-step5\ A\ B\ Rb\ Kab), (m1a-step5\ A\ B\ Rb\ Kab\ nlb)$
 $\{>\ R1x1a\}$

by $(auto\ simp\ add: PO-rhoare-defs\ R1x1a-defs\ m1a-defs\ map-runs-def)$

lemma $PO-m1a-leak-refines-m1x-leak:$

$\{R1x1a\}$
 $(m1x-leak\ Rs), (m1a-leak\ Rs)$

$\{> R1x1a\}$
by (*auto simp add: PO-rhoare-defs R1x1a-defs m1a-defs map-runs-def*)

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)
by (*auto simp add: R1x1a-defs m1a-defs*)

lemma *PO-m1a-refines-trans-m1x [iff]:*
 $\{R1x1a\}$
(trans m1x), (trans m1a)
 $\{> R1x1a\}$
apply (*auto simp add: m1a-def m1a-trans-def m1x-def m1x-trans-def*
intro!: PO-m1a-trans-refines-m1x-trans)
apply (*force intro!: PO-m1a-trans-refines-m1x-trans*)
done

Observation consistency.

lemma *obs-consistent-med1x1a [iff]:*
obs-consistent R1x1a med1x1a m1x m1a
by (*auto simp add: obs-consistent-def R1x1a-def m1a-defs*)

Refinement result.

lemma *PO-m1a-refines-m1x [iff]:*
refines R1x1a med1x1a m1x m1a
by (*rule Refinement-basic*) (*auto del: subsetI*)

lemma *m1a-implements-m1x [iff]: implements med1x1a m1x m1a*
by (*rule refinement-soundness*) (*fast*)

By transitivity:

lemma *m1a-implements-s0g [iff]: implements (med01x o med1x1a) s0g m1a*
by (*rule implements-trans, auto*)

inv (inherited): Secrecy

Secrecy preserved from *m1x*.

lemma *knC-runs1x1a [simp]: knC (runs1x1a runz) = knC runz*
apply (*auto simp add: map-runs-def elim!: knC.cases, auto*)
— 5 subgoals
apply (*rename-tac b, case-tac b, auto*)
apply (*rename-tac b, case-tac b, auto*)
apply (*rule knC-init, auto simp add: map-runs-def*)
apply (*rule knC-resp, auto simp add: map-runs-def*)
apply (*rule knC-serv, auto simp add: map-runs-def*)

done

lemma *PO-m1a-obs-secrecy* [iff]: *oreach* $m1a \subseteq m1x\text{-secrecy}$
apply (*rule-tac* $Q=m1x\text{-secrecy}$ **in** *external-invariant-translation*)
apply (*auto del: subsetI*)
apply (*auto simp add: med1x1a-def m1x-secrecy-def*)
done

lemma *PO-m1a-secrecy* [iff]: *reach* $m1a \subseteq m1x\text{-secrecy}$
by (*rule external-to-internal-invariant*) (*auto del: subsetI*)

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

$is\text{-runs2sigs} :: runs\text{-}t \Rightarrow issig\ signal \Rightarrow nat$

where

$is\text{-runs2sigs}\ runz\ (Running\ [A, Sv]\ (Kab, B, Na, nl)) =$
 $(if\ \exists Rs. Kab = sesK\ (Rs\$sk) \wedge$
 $runz\ Rs = Some\ (Serv, [A, B], aNon\ Na \# nl)$
 $then\ 1\ else\ 0)$

$| is\text{-runs2sigs}\ runz\ (Commit\ [A, Sv]\ (Kab, B, Na, nl)) =$
 $(if\ \exists Ra\ nla. Na = Ra\$na \wedge$
 $runz\ Ra = Some\ (Init, [A, B], aKey\ Kab \# nla) \wedge$
 $take\ is\text{-}len\ nla = nl$
 $then\ 1\ else\ 0)$

$| is\text{-runs2sigs}\ runz\ - = 0$

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-runs2sigs-empty* [simp]:
 $runz = Map.empty \implies is-runs2sigs\ runz = (\lambda s. 0)$
by (rule *ext*, erule *rev-mp*)
 (rule *is-runs2sigs.induct*, *auto*)

Update lemmas

lemma *is-runs2sigs-upd-init-none* [simp]:
 $\llbracket Ra \notin dom\ runz \rrbracket$
 $\implies is-runs2sigs\ (runz(Ra \mapsto (Init, [A, B], []))) = is-runs2sigs\ runz$
by (rule *ext*, erule *rev-mp*)
 (rule *is-runs2sigs.induct*, *auto dest: dom-lemmas*)

lemma *is-runs2sigs-upd-resp-none* [simp]:
 $\llbracket Rb \notin dom\ runz \rrbracket$
 $\implies is-runs2sigs\ (runz(Rb \mapsto (Resp, [A, B], []))) = is-runs2sigs\ runz$
by (rule *ext*, erule *rev-mp*)
 (rule *is-runs2sigs.induct*, *auto dest: dom-lemmas*)

lemma *is-runs2sigs-upd-serv* [simp]:
 $\llbracket Rs \notin dom\ runz \rrbracket$
 $\implies is-runs2sigs\ (runz(Rs \mapsto (Serv, [A, B], aNon\ Na\ \# \ ils))) =$
 $(is-runs2sigs\ runz)(Running\ [A, Sv]\ (sesK\ (Rs\$sk), B, Na, ils) := 1)$
apply (rule *ext*, (erule *rev-mp*)+)
apply (rule *is-runs2sigs.induct*)
apply (*safe*, *simp-all*)+
apply (*fastforce simp add: domIff*)+
done

lemma *is-runs2sigs-upd-init-some* [simp]:
 $\llbracket runz\ Ra = Some\ (Init, [A, B], []);\ ils = take\ is-len\ nla \rrbracket$
 $\implies is-runs2sigs\ (runz(Ra \mapsto (Init, [A, B], aKey\ Kab\ \# \ nla))) =$
 $(is-runs2sigs\ runz)(Commit\ [A, Sv]\ (Kab, B, Ra\$na, ils) := 1)$
apply (rule *ext*, (erule *rev-mp*)+)
apply (rule *is-runs2sigs.induct*)
apply (*safe*, *simp-all*)+
apply (*fastforce*)+
done

lemma *is-runs2sigs-upd-resp-some* [simp]:
 $\llbracket runz\ Rb = Some\ (Resp, [A, B], []) \rrbracket$
 $\implies is-runs2sigs\ (runz(Rb \mapsto (Resp, [A, B], aKey\ Kab\ \# \ nlb))) =$
 $is-runs2sigs\ runz$
apply (rule *ext*, erule *rev-mp*)
apply (rule *is-runs2sigs.induct*, *auto*)
done

Refinement proof

lemma *PO-m1a-step1-refines-a0-is-skip*:
 $\{R-a0m1a-is\}$
 $Id, (m1a-step1\ Ra\ A\ B\ Na)$

$\{> R\text{-}a0m1a\text{-}is\}$
by (auto simp add: PO-rhoare-defs R-a0m1a-is-defs m1a-defs)

lemma PO-m1a-step2-refines-a0-is-skip:

$\{R\text{-}a0m1a\text{-}is\}$
 $Id, (m1a\text{-}step2\ Rb\ A\ B)$
 $\{> R\text{-}a0m1a\text{-}is\}$

by (auto simp add: PO-rhoare-defs R-a0m1a-is-defs m1a-defs)

lemma PO-m1a-step3-refines-a0-is-running:

$\{R\text{-}a0m1a\text{-}is\}$
 $(a0i\text{-}running\ [A, Sv]\ (Kab, B, Na, nls)),$
 $(m1a\text{-}step3\ Rs\ A\ B\ Kab\ Na\ nls)$
 $\{> R\text{-}a0m1a\text{-}is\}$

by (auto simp add: PO-rhoare-defs R-a0m1a-is-defs a0i-defs m1a-defs
dest: dom-lemmas)

lemma PO-m1a-step4-refines-a0-is-commit:

$\{R\text{-}a0m1a\text{-}is \cap UNIV \times m1a\text{-}inv0\text{-}fin\}$
 $(a0i\text{-}commit\ [A, Sv]\ (Kab, B, Na, take\ is\text{-}len\ nla)),$
 $(m1a\text{-}step4\ Ra\ A\ B\ Na\ Kab\ nla)$
 $\{> R\text{-}a0m1a\text{-}is\}$

by (auto simp add: PO-rhoare-defs R-a0m1a-is-defs a0i-defs m1a-defs)

lemma PO-m1a-step5-refines-a0-is-skip:

$\{R\text{-}a0m1a\text{-}is\}$
 $Id, (m1a\text{-}step5\ A\ B\ Rb\ Kab\ nlb)$
 $\{> R\text{-}a0m1a\text{-}is\}$

by (auto simp add: PO-rhoare-defs R-a0m1a-is-defs m1a-defs)

lemma PO-m1a-leak-refines-a0-is-skip:

$\{R\text{-}a0m1a\text{-}is\}$
 $Id, (m1a\text{-}leak\ Rs)$
 $\{> R\text{-}a0m1a\text{-}is\}$

by (auto simp add: PO-rhoare-defs R-a0m1a-is-defs m1a-defs)

All together now...

lemmas PO-m1a-trans-refines-a0-is-trans =

PO-m1a-step1-refines-a0-is-skip PO-m1a-step2-refines-a0-is-skip
PO-m1a-step3-refines-a0-is-running PO-m1a-step4-refines-a0-is-commit
PO-m1a-step5-refines-a0-is-skip PO-m1a-leak-refines-a0-is-skip

lemma PO-m1a-refines-init-a0-is [iff]:

$init\ m1a \subseteq R\text{-}a0m1a\text{-}is \text{“}(init\ a0i)$

by (auto simp add: R-a0m1a-is-defs a0i-defs m1a-defs)

lemma PO-m1a-refines-trans-a0-is [iff]:

$\{R\text{-}a0m1a\text{-}is \cap a0i\text{-}inv1\text{-}iagree \times m1a\text{-}inv0\text{-}fin\}$
 $(trans\ a0i), (trans\ m1a)$
 $\{> R\text{-}a0m1a\text{-}is\}$

by (force simp add: m1a-def m1a-trans-def a0i-def a0i-trans-def
intro!: PO-m1a-trans-refines-a0-is-trans)

lemma *obs-consistent-med-a0m1a-is* [iff]:
obs-consistent R-a0m1a-is med-a0m1a-is a0i m1a
by (*auto simp add: obs-consistent-def R-a0m1a-is-def med-a0m1a-is-def*
a0i-def m1a-def)

Refinement result.

lemma *PO-m1a-refines-a0-is* [iff]:
refines (R-a0m1a-is \cap a0i-inv1-iagree \times m1a-inv0-fin) med-a0m1a-is a0i m1a
by (*rule Refinement-using-invariants*)
(auto del: subsetI)

lemma *m1a-implements-a0-is: implements med-a0m1a-is a0i m1a*
by (*rule refinement-soundness*) (*fast*)

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-inv2i-serv :: 'x m1x-state-scheme set

where

m1a-inv2i-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-len nla))
 }

lemmas *m1a-inv2i-servI* =
m1a-inv2i-serv-def [THEN setc-def-to-intro, rule-format]
lemmas *m1a-inv2i-servE* =
m1a-inv2i-serv-def [THEN setc-def-to-elim, rule-format]
lemmas *m1a-inv2i-servD* =
m1a-inv2i-serv-def [THEN setc-def-to-dest, rule-format, rotated -1]

Invariance proof, see below after init/serv authentication proof.

lemma *PO-m1a-inv2i-serv* [iff]:
reach m1a \subseteq m1a-inv2i-serv
apply (*rule INV-from-Refinement-basic [OF PO-m1a-refines-a0-is]*)
apply (*auto simp add: R-a0m1a-is-def a0i-inv1-iagree-def*
intro!: m1a-inv2i-servI)
apply (*drule-tac x=[A, Sv] in spec, force*)
done

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-inv1-ifresh :: 'x m1a-pred

where

```

m1a-inv1-ifresh ≡ {s. ∀ A A' B B' Ra Ra' Kab nl nl'.
  runs s Ra = Some (Init, [A, B], aKey Kab # nl) →
  runs s Ra' = Some (Init, [A', B'], aKey Kab # nl) →
  A ∉ bad → B ∉ bad → Kab ∉ leak s →
  Ra = Ra'
}

```

lemmas *m1a-inv1-ifreshI* = *m1a-inv1-ifresh-def* [*THEN setc-def-to-intro*, *rule-format*]

lemmas *m1a-inv1-ifreshE* [*elim*] = *m1a-inv1-ifresh-def* [*THEN setc-def-to-elim*, *rule-format*]

lemmas *m1a-inv1-ifreshD* = *m1a-inv1-ifresh-def* [*THEN setc-def-to-dest*, *rule-format*, *rotated 1*]

Invariance proof

lemma *PO-m1a-inv1-ifresh-init* [*iff*]:

init m1a ⊆ *m1a-inv1-ifresh*

by (*auto simp add: m1a-defs intro!: m1a-inv1-ifreshI*)

lemma *PO-m1a-inv1-ifresh-step4*:

```

{m1a-inv1-ifresh ∩ m1a-inv2i-serv ∩ m1x-secrecy}
  m1a-step4 Ra A B Na Kab nla
  {> m1a-inv1-ifresh}

```

proof (*auto simp add: PO-hoare-defs m1a-defs intro!: m1a-inv1-ifreshI*,
auto dest: m1a-inv1-ifreshD, auto dest: m1a-inv2i-servD)

fix *Rs Ra' A' B' nl' s*

assume *H*:

(*sesK* (*Rs* \$ *sk*), *A*) ∈ *azC* (*runs s*) *sesK* (*Rs* \$ *sk*) ∉ *leak s*

A ∉ *bad* *B* ∉ *bad* *Ra' ≠ Ra*

runs s Ra = *Some* (*Init*, [*A*, *B*], [])

runs s Rs = *Some* (*Serv*, [*A*, *B*], *aNon* (*Ra* \$ *na*) # *take is-len nla*)

runs s Ra' = *Some* (*Init*, [*A'*, *B'*], *aKey* (*sesK* (*Rs* \$ *sk*)) # *nl'*)

s ∈ *m1x-secrecy* *s* ∈ *m1a-inv2i-serv*

thus *False*

proof (*cases A' ∈ bad*)

case *True*

from *H* **have** (*sesK* (*Rs*\$*sk*), *A'*) ∈ *azC* (*runs s*) **by** (*elim m1x-secrecyE, auto*)

with *H True* **show** *?thesis* **by** (*elim azC.cases*) (*auto dest: m1a-inv2i-servD*)

next

case *False* **thus** *?thesis* **using** *H* **by** (*auto dest: m1a-inv2i-servD*)

qed

next

fix *A' B' Ra' nl s*

assume

(*Kab*, *A*) ∈ *azC* (*runs s*) *Kab* ∉ *leak s*

A' ∉ bad *B' ∉ bad* *A ∈ bad* *Ra' ≠ Ra*

runs s Ra' = *Some* (*Init*, [*A'*, *B'*], *aKey* *Kab* # *nl*)

runs s Ra = *Some* (*Init*, [*A*, *B*], [])

s ∈ *m1a-inv2i-serv*

thus *False*

by (*elim azC.cases, auto dest: m1a-inv2i-servD*)

qed

lemma *PO-m1a-inv1-ifresh-trans* [*iff*]:

```

  {m1a-inv1-ifresh  $\cap$  m1a-inv2i-serv  $\cap$  m1x-secrecy} trans m1a {> m1a-inv1-ifresh}
proof (simp add: m1a-def m1a-trans-def, safe)
fix Ra A B Kab Ts nla
show
  {m1a-inv1-ifresh  $\cap$  m1a-inv2i-serv  $\cap$  m1x-secrecy}
    m1a-step4 Ra A B Kab Ts nla
  {> m1a-inv1-ifresh}
by (rule PO-m1a-inv1-ifresh-step4)
qed (auto simp add: PO-hoare-defs m1a-defs intro!: m1a-inv1-ifreshI)

lemma PO-m1a-inv1-ifresh [iff]: reach m1a  $\subseteq$  m1a-inv1-ifresh
by (rule-tac J= m1a-inv2i-serv  $\cap$  m1x-secrecy in inv-rule-incr)
  (auto simp add: Int-assoc del: subsetI)

```

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

$rs-commit\ runz\ A\ B\ Kab\ rsl \equiv \{Rb. \exists nlb.$
 $runz\ Rb = Some\ (Resp, [A, B], aKey\ Kab\ \# nlb) \wedge take\ rs-len\ nlb = rsl$
 $\}$

fun

$rs-runs2sigs :: runs-t \Rightarrow rssig\ signal \Rightarrow nat$

where

$rs-runs2sigs\ runz\ (Running\ [B, Sv]\ (Kab, A, rsl)) =$
 $(if\ (\exists Rs\ Na. Kab = sesK\ (Rs\$sk) \wedge$
 $runz\ Rs = Some\ (Serv, [A, B], aNon\ Na\ \# rsl))$
 $then\ 1\ else\ 0)$

$| rs-runs2sigs\ runz\ (Commit\ [B, Sv]\ (Kab, A, rsl)) =$
 $card\ (rs-commit\ runz\ A\ B\ Kab\ rsl)$

$| rs-runs2sigs\ runz\ - = 0$

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

$R\text{-a0m1a-rs} :: (\text{rssig a0n-state} \times \text{m1a-state}) \text{ set where}$
 $R\text{-a0m1a-rs} \equiv \{(s, t). \text{signals } s = \text{rs-runs2sigs (runs } t) \wedge \text{corrupted } s = \{\}\}$

lemmas $R\text{-a0m1a-rs-defs} = R\text{-a0m1a-rs-def med-a0m1a-rs-def}$

Lemmas about the auxiliary functions

Other lemmas

lemma $\text{rs-runs2sigs-empty [simp]}:$

$\text{runz} = \text{Map.empty} \implies \text{rs-runs2sigs runz} = (\lambda s. 0)$

by $(\text{rule ext, erule rev-mp})$

$(\text{rule rs-runs2sigs.induct, auto})$

lemma $\text{rs-commit-finite [simp, intro]}:$

$\text{finite (dom runz)} \implies \text{finite (rs-commit runz } A \ B \ Kab \ nls)$

by $(\text{auto intro: finite-subset dest: dom-lemmas})$

Update lemmas

lemma $\text{rs-runs2sigs-upd-init-none [simp]}:$

$\llbracket Ra \notin \text{dom runz} \rrbracket$

$\implies \text{rs-runs2sigs (runz}(Ra \mapsto (\text{Init}, [A, B], []))) = \text{rs-runs2sigs runz}$

by $(\text{rule ext, erule rev-mp})$

$(\text{rule rs-runs2sigs.induct, auto dest: dom-lemmas})$

lemma $\text{rs-runs2sigs-upd-resp-none [simp]}:$

$\llbracket Rb \notin \text{dom runz} \rrbracket$

$\implies \text{rs-runs2sigs (runz}(Rb \mapsto (\text{Resp}, [A, B], []))) = \text{rs-runs2sigs runz}$

by $(\text{rule ext, erule rev-mp})$

$(\text{rule rs-runs2sigs.induct, auto dest: dom-lemmas})$

lemma $\text{rs-runs2sigs-upd-serv [simp]}:$

$\llbracket Rs \notin \text{dom runz} \rrbracket$

$\implies \text{rs-runs2sigs (runz}(Rs \mapsto (\text{Serv}, [A, B], \text{aNon } Na \ \# \ nls))) =$

$(\text{rs-runs2sigs runz})(\text{Running } [B, Sv] (\text{sesK } (Rs\$sk), A, nls) := 1)$

by $(\text{rule ext, erule rev-mp})$

$(\text{rule rs-runs2sigs.induct, auto dest: dom-lemmas})$

lemma $\text{rs-runs2sigs-upd-init-some [simp]}:$

$\llbracket \text{runz } Ra = \text{Some (Init, [A, B], [])} \rrbracket$

$\implies \text{rs-runs2sigs (runz}(Ra \mapsto (\text{Init}, [A, B], \text{aKey } Kab \ \# \ nl))) =$

rs-runs2sigs runz

by $(\text{rule ext, erule rev-mp})$

$(\text{rule rs-runs2sigs.induct, auto dest: dom-lemmas})$

lemma $\text{rs-runs2sigs-upd-resp-some [simp]}:$

$\llbracket \text{runz } Rb = \text{Some (Resp, [A, B], []); finite (dom runz);}$

$\text{rsl} = \text{take rs-len nlb} \rrbracket$

$\implies \text{rs-runs2sigs (runz}(Rb \mapsto (\text{Resp}, [A, B], \text{aKey } Kab \ \# \ nlb))) =$

$(\text{rs-runs2sigs runz})($

$\text{Commit } [B, Sv] (Kab, A, rsl) := \text{Suc (card (rs-commit runz } A \ B \ Kab \ rsl)))$

```

apply (rule ext, (erule rev-mp)+)
apply (rule rs-runs2sigs.induct, auto dest: dom-lemmas)
— 1 subgoal
apply (rule-tac s=card (insert Rb (rs-commit runz A B Kab (take rs-len nlb))))
  in trans, fast, auto)
done

```

Refinement proof

lemma *PO-m1a-step1-refines-a0-rs-skip*:

```

{R-a0m1a-rs}
  Id, (m1a-step1 Ra A B Na)
{> R-a0m1a-rs}

```

by (auto simp add: PO-rhoare-defs R-a0m1a-rs-defs m1a-defs)

lemma *PO-m1a-step2-refines-a0-rs-skip*:

```

{R-a0m1a-rs}
  Id, (m1a-step2 Rb A B)
{> R-a0m1a-rs}

```

by (auto simp add: PO-rhoare-defs R-a0m1a-rs-defs m1a-defs)

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-a0m1a-rs}

```

by (auto simp add: PO-rhoare-defs R-a0m1a-rs-defs a0i-defs m1a-defs
dest: dom-lemmas)

lemma *PO-m1a-step4-refines-a0-rs-skip*:

```

{R-a0m1a-rs}
  Id, (m1a-step4 Ra A B Na Kab nla)
{> R-a0m1a-rs}

```

by (auto simp add: PO-rhoare-defs R-a0m1a-rs-defs a0i-defs m1a-defs)

lemma *PO-m1a-step5-refines-a0-rs-commit*:

```

{R-a0m1a-rs  $\cap$  UNIV  $\times$  m1a-inv0-fn}
  (a0n-commit [B, Sv] (Kab, A, take rs-len nlb)),
  (m1a-step5 Rb A B Kab nlb)
{> R-a0m1a-rs}

```

by (auto simp add: PO-rhoare-defs R-a0m1a-rs-defs a0i-defs m1a-defs)

lemma *PO-m1a-leak-refines-a0-rs-skip*:

```

{R-a0m1a-rs}
  Id, (m1a-leak Rs)
{> R-a0m1a-rs}

```

by (auto simp add: PO-rhoare-defs R-a0m1a-rs-defs a0i-defs m1a-defs)

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]:
 $init\ m1a \subseteq R\text{-}a0m1a\text{-}rs''(init\ a0n)$
by (*auto simp add: R-a0m1a-rs-defs a0n-defs m1a-defs*)

lemma *PO-m1a-refines-trans-ra0n* [iff]:
 $\{R\text{-}a0m1a\text{-}rs \cap a0n\text{-}inv1\text{-}niagree \times m1a\text{-}inv0\text{-}fin\}$
 $(trans\ a0n), (trans\ m1a)$
 $\{> R\text{-}a0m1a\text{-}rs\}$
by (*force simp add: m1a-def m1a-trans-def a0n-def a0n-trans-def*
intro!: PO-m1a-trans-refines-a0-rs-trans)

lemma *obs-consistent-med-a0m1a-rs* [iff]:
 $obs\text{-}consistent$
 $(R\text{-}a0m1a\text{-}rs \cap a0n\text{-}inv1\text{-}niagree \times m1a\text{-}inv0\text{-}fin)$
 $med\text{-}a0m1a\text{-}rs\ a0n\ m1a$
by (*auto simp add: obs-consistent-def R-a0m1a-rs-def med-a0m1a-rs-def*
a0n-def m1a-def)

Refinement result.

lemma *PO-m1a-refines-a0-rs* [iff]:
 $refines\ (R\text{-}a0m1a\text{-}rs \cap a0n\text{-}inv1\text{-}niagree \times m1a\text{-}inv0\text{-}fin)\ med\text{-}a0m1a\text{-}rs\ a0n\ m1a$
by (*rule Refinement-using-invariants*) (*auto*)

lemma *m1a-implements-ra0n: implements med-a0m1a-rs a0n m1a*
by (*rule refinement-soundness*) (*fast*)

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

$m1a\text{-}inv2r\text{-}serv :: 'x\ m1x\text{-}state\text{-}scheme\ set$

where

$m1a\text{-}inv2r\text{-}serv \equiv \{s. \forall A\ B\ Rb\ Kab\ nlb.$
 $B \notin bad \longrightarrow$
 $runs\ s\ Rb = Some\ (Resp, [A, B], aKey\ Kab\ \# \ nlb) \longrightarrow$
 $(\exists Rs\ Na. Kab = sesK\ (Rs\$sk) \wedge$
 $runs\ s\ Rs = Some\ (Serv, [A, B], aNon\ Na\ \# \ take\ rs\text{-}len\ nlb))$
 $\}$

lemmas $m1a\text{-}inv2r\text{-}servI =$

$m1a\text{-}inv2r\text{-}serv\text{-}def\ [THEN\ setc\text{-}def\text{-}to\text{-}intro, rule\text{-}format]$

lemmas $m1a\text{-}inv2r\text{-}servE\ [elim] =$

$m1a\text{-}inv2r\text{-}serv\text{-}def\ [THEN\ setc\text{-}def\text{-}to\text{-}elim, rule\text{-}format]$

lemmas $m1a\text{-}inv2r\text{-}servD =$

$m1a\text{-}inv2r\text{-}serv\text{-}def\ [THEN\ setc\text{-}def\text{-}to\text{-}dest, rule\text{-}format, rotated\ -1]$

Invariance proof

lemma *PO-m1a-inv2r-serv* [iff]:
 $reach\ m1a \subseteq m1a\text{-}inv2r\text{-}serv$

```

apply (rule INV-from-Refinement-basic [OF PO-m1a-refines-a0-rs])
apply (auto simp add: R-a0m1a-rs-def a0n-inv1-niagree-def intro!: m1a-inv2r-servI)
apply (rename-tac x A B Rb Kab nlb a, drule-tac x=[B, Sv] in spec, clarsimp)
apply (rename-tac x A B Rb Kab nlb a, drule-tac x=Kab in spec, force)
done

```

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-obs = m1x-obs

```

```

type-synonym 'x m1a-pred = 'x m1x-pred
type-synonym 'x m1a-trans = 'x m1x-trans

```

We need some parameters regarding the list of freshness values stored by the server. These should be defined in further refinements.

```

consts
  is-len :: nat — num of agreeing list elements for initiator-server
  rs-len :: nat — num of agreeing list elements for responder-server

```

3.3.2 Events

```

definition — by A, refines m1x-step1
  m1a-step1 :: [rid-t, agent, agent]  $\Rightarrow$  'x m1r-trans
where
  m1a-step1  $\equiv$  m1x-step1

```

```

definition — by B, refines m1x-step2
  m1a-step2 :: [rid-t, agent, agent]  $\Rightarrow$  'x m1r-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 $m1x\text{-step5}$

$m1a\text{-step5} :: [rid\text{-}t, agent, agent, key, atom\ list] \Rightarrow 'x\ m1a\text{-}trans$

where

$m1a\text{-step5}\ Rb\ A\ B\ Kab\ nlb \equiv \{(s, s1)\}.$

— guards:

$runs\ s\ Rb = Some\ (Resp, [A, B], []) \wedge$
 $(Kab \notin leak\ s \longrightarrow (Kab, B) \in azC\ (runs\ s)) \wedge$ — authorization guard

— guard for non-injective agreement with server on (Kab, A, rsl)

— where $rsl = take\ rs\text{-}len\ nlb$

$(B \notin bad \longrightarrow (\exists Rs. Kab = sesK\ (Rs\$sk) \wedge$
 $runs\ s\ Rs = Some\ (Serv, [A, B], take\ rs\text{-}len\ nlb))) \wedge$

— actions:

$s1 = s \langle runs := (runs\ s)(Rb \mapsto (Resp, [A, B], aKey\ Kab\ \# nlb)) \rangle$

}

definition — by attacker, refines $m1x\text{-leak}$

$m1a\text{-leak} :: rid\text{-}t \Rightarrow 'x\ m1x\text{-}trans$

where

$m1a\text{-leak} = m1x\text{-leak}$

3.3.3 Specification

definition

$m1a-init :: m1a-state\ set$

where

$m1a-init \equiv m1x-init$

definition

$m1a-trans :: 'x\ m1a-trans\ \mathbf{where}$

$m1a-trans \equiv (\bigcup A\ B\ Ra\ Rb\ Rs\ Kab\ nls\ nla\ nlb.$

$m1a-step1\ Ra\ A\ B \cup$

$m1a-step2\ Rb\ A\ B \cup$

$m1a-step3\ Rs\ A\ B\ Kab\ nls \cup$

$m1a-step4\ Ra\ A\ B\ Kab\ nla \cup$

$m1a-step5\ Rb\ A\ B\ Kab\ nlb \cup$

$m1a-leak\ Rs \cup$

Id

)

definition

$m1a :: (m1a-state, m1a-obs)\ spec\ \mathbf{where}$

$m1a \equiv (\langle$

$init = m1a-init,$

$trans = m1a-trans,$

$obs = id$

\rangle

lemma $init-m1a: init\ m1a = m1a-init$

by ($simp\ add: m1a-def$)

lemma $trans-m1a: trans\ m1a = m1a-trans$

by ($simp\ add: m1a-def$)

lemma $obs-m1a\ [simp]: obs\ m1a = id$

by ($simp\ add: m1a-def$)

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

$m1a-inv0-fin \equiv \{s.\ finite\ (dom\ (runs\ s))\}$

lemmas $m1a\text{-inv0}\text{-finI} = m1a\text{-inv0}\text{-fin}\text{-def}$ [THEN setc-def-to-intro, rule-format]
lemmas $m1a\text{-inv0}\text{-finE}$ [elim] = $m1a\text{-inv0}\text{-fin}\text{-def}$ [THEN setc-def-to-elim, rule-format]
lemmas $m1a\text{-inv0}\text{-finD} = m1a\text{-inv0}\text{-fin}\text{-def}$ [THEN setc-def-to-dest, rule-format]

Invariance proof.

lemma $PO\text{-}m1a\text{-inv0}\text{-fin}\text{-init}$ [iff]:
 $init\ m1a \subseteq m1a\text{-inv0}\text{-fin}$
by (auto simp add: $m1a\text{-defs}$ intro!: $m1a\text{-inv0}\text{-finI}$)

lemma $PO\text{-}m1a\text{-inv0}\text{-fin}\text{-trans}$ [iff]:
 $\{m1a\text{-inv0}\text{-fin}\}$ trans $m1a$ $\{> m1a\text{-inv0}\text{-fin}\}$
by (auto simp add: $PO\text{-hoare}\text{-defs}$ $m1a\text{-defs}$ intro!: $m1a\text{-inv0}\text{-finI}$)

lemma $PO\text{-}m1a\text{-inv0}\text{-fin}$ [iff]: $reach\ m1a \subseteq m1a\text{-inv0}\text{-fin}$
by (rule inv-rule-incr, auto del: subsetI)

3.3.5 Refinement of $m1x$

Simulation relation

Define run abstraction.

fun
 $rm1x1a :: role\text{-}t \Rightarrow atom\ list \Rightarrow atom\ list$
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 nls

abbreviation

$runs1x1a :: runs\text{-}t \Rightarrow runs\text{-}t$ **where**
 $runs1x1a \equiv map\text{-}runs\ rm1x1a$

med1x1: The mediator function maps a concrete observation to an abstract one.

definition

$med1x1a :: m1a\text{-obs} \Rightarrow m1x\text{-obs}$ **where**
 $med1x1a\ t \equiv (\ ()\ runs = runs1x1a\ (runs\ t),\ leak = leak\ t\)$

R1x1a: The simulation relation is defined in terms of the mediator function.

definition

$R1x1a :: (m1x\text{-state} \times m1a\text{-state})\ set$ **where**
 $R1x1a \equiv \{(s, t). s = med1x1a\ t\}$

lemmas $R1x1a\text{-defs} =$
 $R1x1a\text{-def}\ med1x1a\text{-def}$

Refinement proof

lemma $PO\text{-}m1a\text{-step1}\text{-refines}\text{-}m1x\text{-step1}$:
 $\{R1x1a\}$
 $(m1x\text{-step1}\ Ra\ A\ B), (m1a\text{-step1}\ Ra\ A\ B)$
 $\{> R1x1a\}$

by (*auto simp add: PO-rhoare-defs R1x1a-defs m1a-defs*)

lemma *PO-m1a-step2-refines-m1x-step2*:

{*R1x1a*}
(*m1x-step2 Rb A B*), (*m1a-step2 Rb A B*)
{> *R1x1a*}

by (*auto simp add: PO-rhoare-defs R1x1a-defs m1a-defs*)

lemma *PO-m1a-step3-refines-m1x-step3*:

{*R1x1a*}
(*m1x-step3 Rs A B Kab*), (*m1a-step3 Rs A B Kab nls*)
{> *R1x1a*}

by (*auto simp add: PO-rhoare-defs R1x1a-defs m1a-defs*)

lemma *PO-m1a-step4-refines-m1x-step4*:

{*R1x1a*}
(*m1x-step4 Ra A B Kab*), (*m1a-step4 Ra A B Kab nla*)
{> *R1x1a*}

by (*auto simp add: PO-rhoare-defs R1x1a-defs m1a-defs map-runs-def*)

lemma *PO-m1a-step5-refines-m1x-step5*:

{*R1x1a*}
(*m1x-step5 Rb A B Kab*), (*m1a-step5 Rb A B Kab nlb*)
{> *R1x1a*}

by (*auto simp add: PO-rhoare-defs R1x1a-defs m1a-defs map-runs-def*)

lemma *PO-m1a-leak-refines-m1x-leak*:

{*R1x1a*}
(*m1x-leak Rs*), (*m1a-leak Rs*)
{> *R1x1a*}

by (*auto simp add: PO-rhoare-defs R1x1a-defs m1a-defs map-runs-def*)

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*)

by (*auto simp add: R1x1a-defs m1a-defs*)

lemma *PO-m1a-refines-trans-m1x [iff]*:

{*R1x1a*}
(*trans m1x*), (*trans m1a*)
{> *R1x1a*}

apply (*auto simp add: m1a-def m1a-trans-def m1x-def m1x-trans-def*
intro!: PO-m1a-trans-refines-m1x-trans)

apply (*force intro!: PO-m1a-trans-refines-m1x-trans*)
done

Observation consistency.

lemma *obs-consistent-med1x1a* [iff]:
obs-consistent R1x1a med1x1a m1x m1a
by (*auto simp add: obs-consistent-def R1x1a-def m1a-defs*)

Refinement result.

lemma *PO-m1a-refines-m1x* [iff]:
refines R1x1a med1x1a m1x m1a
by (*rule Refinement-basic*) (*auto del: subsetI*)

lemma *m1a-implements-m1x* [iff]: *implements med1x1a m1x m1a*
by (*rule refinement-soundness*) (*fast*)

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* × *agent* × *atom list*

abbreviation

is-commit :: [*runs-t*, *agent*, *agent*, *key*, *atom list*] ⇒ *rid-t set*

where

is-commit runz A B Kab sl ≡ {*Ra*. ∃ *nla*.
runz Ra = *Some (Init, [A, B], aKey Kab # nla)* ∧ *take is-len nla* = *sl*
}

fun

is-runs2sigs :: *runs-t* ⇒ *issig signal* ⇒ *nat*

where

is-runs2sigs runz (Running [A, Sv] (Kab, B, sl)) =
(*if* ∃ *Rs nls*. *Kab* = *sesK (Rs\$sk)* ∧
runz Rs = *Some (Serv, [A, B], nls)* ∧ *take is-len nls* = *sl*
then 1 else 0)

| *is-runs2sigs runz (Commit [A, Sv] (Kab, B, sl))* =
card (is-commit runz A B Kab sl)

| *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* ⇒ *issig a0i-obs* **where**
med-a0m1a-is o1 ≡ (| *signals* = *is-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]:
 $runz = Map.empty \implies is\text{-}runs2sigs\ runz = (\lambda s. 0)$
by (rule ext, erule rev-mp)
(rule is-runs2sigs.induct, auto)

lemma $is\text{-}commit\text{-}finite$ [simp, intro]:
 $finite\ (dom\ runz) \implies finite\ (is\text{-}commit\ runz\ A\ B\ Kab\ nls)$
by (auto intro: finite-subset dest: dom-lemmas)

Update lemmas

lemma $is\text{-}runs2sigs\text{-}upd\text{-}init\text{-}none$ [simp]:
 $\llbracket Ra \notin dom\ runz \rrbracket$
 $\implies is\text{-}runs2sigs\ (runz(Ra \mapsto (Init, [A, B], []))) = is\text{-}runs2sigs\ runz$
by (rule ext, erule rev-mp)
(rule is-runs2sigs.induct, auto dest: dom-lemmas)

lemma $is\text{-}runs2sigs\text{-}upd\text{-}resp\text{-}none$ [simp]:
 $\llbracket Rb \notin dom\ runz \rrbracket$
 $\implies is\text{-}runs2sigs\ (runz(Rb \mapsto (Resp, [A, B], []))) = is\text{-}runs2sigs\ runz$
by (rule ext, erule rev-mp)
(rule is-runs2sigs.induct, auto dest: dom-lemmas)

lemma $is\text{-}runs2sigs\text{-}upd\text{-}serv$ [simp]:
 $\llbracket Rs \notin dom\ runz \rrbracket$
 $\implies is\text{-}runs2sigs\ (runz(Rs \mapsto (Serv, [A, B], nls))) =$
 $(is\text{-}runs2sigs\ runz)(Running\ [A, Sv]\ (sesK\ (Rs\$sk), B, take\ is\text{-}len\ nls) := 1)$
by (rule ext, erule rev-mp)
(rule is-runs2sigs.induct, auto dest: dom-lemmas)

lemma $is\text{-}runs2sigs\text{-}upd\text{-}init\text{-}some$ [simp]:
 $\llbracket runz\ Ra = Some\ (Init, [A, B], []); finite\ (dom\ runz);$
 $ils = take\ is\text{-}len\ nla \rrbracket$
 $\implies is\text{-}runs2sigs\ (runz(Ra \mapsto (Init, [A, B], aKey\ Kab\ \# \ nla))) =$
 $(is\text{-}runs2sigs\ runz)($
 $Commit\ [A, Sv]\ (Kab, B, ils) :=$
 $Suc\ (card\ (is\text{-}commit\ runz\ A\ B\ Kab\ ils)))$
apply (rule ext, erule rev-mp, erule rev-mp, erule rev-mp)
apply (rename-tac s)
apply (rule-tac ?a0.0=runz and ?a1.0=s in is-runs2sigs.induct, auto)
— 1 subgoal
apply (rename-tac runz)
apply (rule-tac s=card (insert Ra (is-commit runz A B Kab (take is-len nla)))
in trans, fast, auto)
done

lemma $is\text{-}runs2sigs\text{-}upd\text{-}resp\text{-}some$ [simp]:

$\llbracket \text{runz } Rb = \text{Some } (Resp, [A, B], []) \rrbracket$
 $\implies \text{is-runs2sigs } (\text{runz}(Rb \mapsto (Resp, [A, B], aKey Kab \# nlb))) =$
 is-runs2sigs runz
by (*rule ext, erule rev-mp*)
(rule is-runs2sigs.induct, auto dest: dom-lemmas)

Refinement proof

lemma *PO-m1a-step1-refines-a0-is-skip:*

$\{R\text{-a0m1a-is}\}$
 $Id, (m1a\text{-step1 } Ra \ A \ B)$
 $\{> R\text{-a0m1a-is}\}$

by (*auto simp add: PO-rhoare-defs R-a0m1a-is-defs m1a-defs*)

lemma *PO-m1a-step2-refines-a0-is-skip:*

$\{R\text{-a0m1a-is}\}$
 $Id, (m1a\text{-step2 } Rb \ A \ B)$
 $\{> R\text{-a0m1a-is}\}$

by (*auto simp add: PO-rhoare-defs R-a0m1a-is-defs m1a-defs*)

lemma *PO-m1a-step3-refines-a0-is-running:*

$\{R\text{-a0m1a-is}\}$
 $(a0n\text{-running } [A, Sv] (Kab, B, take \ is\text{-len } nls)),$
 $(m1a\text{-step3 } Rs \ A \ B \ Kab \ nls)$
 $\{> R\text{-a0m1a-is}\}$

by (*auto simp add: PO-rhoare-defs R-a0m1a-is-defs a0i-defs m1a-defs*
dest: dom-lemmas)

lemma *PO-m1a-step4-refines-a0-is-commit:*

$\{R\text{-a0m1a-is} \cap UNIV \times m1a\text{-inv0-fin}\}$
 $(a0n\text{-commit } [A, Sv] (Kab, B, take \ is\text{-len } nla)),$
 $(m1a\text{-step4 } Ra \ A \ B \ Kab \ nla)$
 $\{> R\text{-a0m1a-is}\}$

by (*simp add: PO-rhoare-defs R-a0m1a-is-defs a0i-defs m1a-defs, safe, auto*)

lemma *PO-m1a-step5-refines-a0-is-skip:*

$\{R\text{-a0m1a-is}\}$
 $Id, (m1a\text{-step5 } Rb \ A \ B \ Kab \ nlb)$
 $\{> R\text{-a0m1a-is}\}$

by (*simp add: PO-rhoare-defs R-a0m1a-is-defs m1a-defs, safe, auto*)

lemma *PO-m1a-leak-refines-a0-is-skip:*

$\{R\text{-a0m1a-is}\}$
 $Id, (m1a\text{-leak } Rs)$
 $\{> R\text{-a0m1a-is}\}$

by (*simp add: PO-rhoare-defs R-a0m1a-is-defs m1a-defs, safe, auto*)

All together now...

lemmas *PO-m1a-trans-refines-a0-is-trans =*

$PO\text{-m1a-step1-refines-a0-is-skip } PO\text{-m1a-step2-refines-a0-is-skip}$
 $PO\text{-m1a-step3-refines-a0-is-running } PO\text{-m1a-step4-refines-a0-is-commit}$
 $PO\text{-m1a-step5-refines-a0-is-skip } PO\text{-m1a-leak-refines-a0-is-skip}$

lemma *PO-m1a-refines-init-a0-is* [iff]:
init m1a \subseteq *R-a0m1a-is*“(init a0n)
by (auto simp add: *R-a0m1a-is-defs a0n-defs m1a-defs*)

lemma *PO-m1a-refines-trans-a0-is* [iff]:
 $\{R-a0m1a-is \cap UNIV \times m1a-inv0-fin\}$
 (trans a0n), (trans m1a)
 $\{> R-a0m1a-is\}$
by (auto simp add: *m1a-def m1a-trans-def a0n-def a0n-trans-def*
 intro!: *PO-m1a-trans-refines-a0-is-trans*)

lemma *obs-consistent-med-a0m1a-is* [iff]:
obs-consistent R-a0m1a-is med-a0m1a-is a0n m1a
by (auto simp add: *obs-consistent-def R-a0m1a-is-def med-a0m1a-is-def*
a0n-def m1a-def)

Refinement result.

lemma *PO-m1a-refines-a0-is* [iff]:
refines (R-a0m1a-is \cap UNIV \times m1a-inv0-fin) med-a0m1a-is a0n m1a
by (rule *Refinement-using-invariants*) (auto del: *subsetI*)

lemma *m1a-implements-a0-is: implements med-a0m1a-is a0n m1a*
by (rule *refinement-soundness*) (fast)

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-commit :: [*runs-t*, *agent*, *agent*, *key*, *atom list*] \Rightarrow *rid-t set*

where

rs-commit runz A B Kab rsl \equiv $\{Rb. \exists nlb.$
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)) =
(if $\exists Rs nls. Kab = sesK (Rs\$sk) \wedge$
runz Rs = Some (Serv, [A, B], nls) \wedge take rs-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*]:

$runz = Map.empty \implies rs\text{-runs2sigs runz = (\lambda s. 0)$

by (*rule ext, erule rev-mp*)

(*rule rs-runs2sigs.induct, auto*)

lemma $rs\text{-commit}\text{-finite}$ [*simp, intro*]:

$finite (dom runz) \implies finite (rs\text{-commit runz A B Kab nls)$

by (*auto intro: finite-subset dest: dom-lemmas*)

Update lemmas

lemma $rs\text{-runs2sigs}\text{-upd}\text{-init}\text{-none}$ [*simp*]:

$\llbracket Ra \notin dom runz \rrbracket$

$\implies rs\text{-runs2sigs (runz(Ra \mapsto (Init, [A, B], []))) = rs\text{-runs2sigs runz}$

by (*rule ext, erule rev-mp*)

(*rule rs-runs2sigs.induct, auto dest: dom-lemmas*)

lemma $rs\text{-runs2sigs}\text{-upd}\text{-resp}\text{-none}$ [*simp*]:

$\llbracket Rb \notin dom runz \rrbracket$

$\implies rs\text{-runs2sigs (runz(Rb \mapsto (Resp, [A, B], []))) = rs\text{-runs2sigs runz}$

by (*rule ext, erule rev-mp*)

(*rule rs-runs2sigs.induct, auto dest: dom-lemmas*)

lemma $rs\text{-runs2sigs}\text{-upd}\text{-serv}$ [*simp*]:

$\llbracket Rs \notin dom runz \rrbracket$

$\implies rs\text{-runs2sigs (runz(Rs \mapsto (Serv, [A, B], nls))) =$

$(rs\text{-runs2sigs runz)(Running [B, Sv] (sesK (Rs\$sk), A, take rs\text{-len nls) := 1)$

by (*rule ext, erule rev-mp*)

(*rule rs-runs2sigs.induct, auto dest: dom-lemmas*)

lemma $rs\text{-runs2sigs}\text{-upd}\text{-init}\text{-some}$ [*simp*]:

$\llbracket runz Ra = Some (Init, [A, B], []) \rrbracket$

\implies $rs\text{-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], \text{aKey } Kab \# nl))) =$
 $rs\text{-runs2sigs } \text{runz}$
by (*rule ext, erule rev-mp*)
(rule rs-runs2sigs.induct, auto dest: dom-lemmas)

lemma *rs-runs2sigs-upd-resp-some [simp]:*
 $\llbracket \text{runz } Rb = \text{Some } (\text{Resp}, [A, B], \llbracket \rrbracket); \text{finite } (\text{dom } \text{runz});$
 $\text{rsl} = \text{take } rs\text{-len } nlb \rrbracket$
 $\implies rs\text{-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], \text{aKey } Kab \# nlb))) =$
 $(rs\text{-runs2sigs } \text{runz})($
 $\text{Commit } [B, Sv] (Kab, A, rsl) := \text{Suc } (\text{card } (rs\text{-commit } \text{runz } A \ B \ Kab \ rsl)))$
apply (*rule ext, erule rev-mp, erule rev-mp, erule rev-mp*)
apply (*rule rs-runs2sigs.induct, auto dest: dom-lemmas*)
— 1 subgoal
apply (*rename-tac runz*)
apply (*rule-tac s=card (insert Rb (rs-commit runz A B Kab (take rs-len nlb)))*)
in *trans, fast, auto*)
done

Refinement proof

lemma *PO-m1a-step1-refines-a0-rs-skip:*
 $\{R\text{-a0m1a-rs}\}$
 $\text{Id}, (m1a\text{-step1 } Ra \ A \ B)$
 $\{> R\text{-a0m1a-rs}\}$
by (*auto simp add: PO-rhoare-defs R-a0m1a-rs-defs m1a-defs*)

lemma *PO-m1a-step2-refines-a0-rs-skip:*
 $\{R\text{-a0m1a-rs}\}$
 $\text{Id}, (m1a\text{-step2 } Rb \ A \ B)$
 $\{> R\text{-a0m1a-rs}\}$
by (*auto simp add: PO-rhoare-defs R-a0m1a-rs-defs m1a-defs*)

lemma *PO-m1a-step3-refines-a0-rs-running:*
 $\{R\text{-a0m1a-rs}\}$
 $(a0n\text{-running } [B, Sv] (Kab, A, \text{take } rs\text{-len } nls)),$
 $(m1a\text{-step3 } Rs \ A \ B \ Kab \ nls)$
 $\{> R\text{-a0m1a-rs}\}$
by (*auto simp add: PO-rhoare-defs R-a0m1a-rs-defs a0n-defs m1a-defs*
dest: dom-lemmas)

lemma *PO-m1a-step4-refines-a0-rs-skip:*
 $\{R\text{-a0m1a-rs}\}$
 $\text{Id}, (m1a\text{-step4 } Ra \ A \ B \ Kab \ nla)$
 $\{> R\text{-a0m1a-rs}\}$
by (*auto simp add: PO-rhoare-defs R-a0m1a-rs-defs m1a-defs*)

lemma *PO-m1a-step5-refines-a0-rs-commit:*
 $\{R\text{-a0m1a-rs} \cap UNIV \times m1a\text{-inv0-fin}\}$
 $(a0n\text{-commit } [B, Sv] (Kab, A, \text{take } rs\text{-len } nlb)),$
 $(m1a\text{-step5 } Rb \ A \ B \ Kab \ nlb)$
 $\{> R\text{-a0m1a-rs}\}$
by (*auto simp add: PO-rhoare-defs R-a0m1a-rs-defs a0n-defs m1a-defs*)

lemma *PO-m1a-leak-refines-a0-rs-skip*:
 $\{R\text{-}a0m1a\text{-}rs\}$
 $Id, (m1a\text{-}leak\ Rs)$
 $\{> R\text{-}a0m1a\text{-}rs\}$
by (*auto simp add: PO-rhoare-defs R-a0m1a-rs-defs a0i-defs m1a-defs*)

All together now...

lemmas *PO-m1a-trans-refines-a0-rs-trans* =
 $PO\text{-}m1a\text{-}step1\text{-}refines\text{-}a0\text{-}rs\text{-}skip$ $PO\text{-}m1a\text{-}step2\text{-}refines\text{-}a0\text{-}rs\text{-}skip$
 $PO\text{-}m1a\text{-}step3\text{-}refines\text{-}a0\text{-}rs\text{-}running$ $PO\text{-}m1a\text{-}step4\text{-}refines\text{-}a0\text{-}rs\text{-}skip$
 $PO\text{-}m1a\text{-}step5\text{-}refines\text{-}a0\text{-}rs\text{-}commit$ $PO\text{-}m1a\text{-}leak\text{-}refines\text{-}a0\text{-}rs\text{-}skip$

lemma *PO-m1a-refines-init-ra0n* [*iff*]:
 $init\ m1a \subseteq R\text{-}a0m1a\text{-}rs''(init\ a0n)$
by (*auto simp add: R-a0m1a-rs-defs a0n-defs m1a-defs*)

lemma *PO-m1a-refines-trans-ra0n* [*iff*]:
 $\{R\text{-}a0m1a\text{-}rs \cap UNIV \times m1a\text{-}inv0\text{-}fin\}$
 $(trans\ a0n), (trans\ m1a)$
 $\{> R\text{-}a0m1a\text{-}rs\}$
by (*auto simp add: m1a-def m1a-trans-def a0n-def a0n-trans-def*
intro!: PO-m1a-trans-refines-a0-rs-trans)

lemma *obs-consistent-med-a0m1a-rs* [*iff*]:
 $obs\text{-}consistent\ (R\text{-}a0m1a\text{-}rs \cap UNIV \times m1a\text{-}inv0\text{-}fin)\ med\text{-}a0m1a\text{-}rs\ a0n\ m1a$
by (*auto simp add: obs-consistent-def R-a0m1a-rs-def med-a0m1a-rs-def*
a0n-def m1a-def)

Refinement result.

lemma *PO-m1a-refines-a0-rs* [*iff*]:
 $refines\ (R\text{-}a0m1a\text{-}rs \cap UNIV \times m1a\text{-}inv0\text{-}fin)\ med\text{-}a0m1a\text{-}rs\ a0n\ m1a$
by (*rule Refinement-using-invariants*) (*auto*)

lemma *m1a-implements-ra0n: implements med-a0m1a-rs a0n m1a*
by (*rule refinement-soundness*) (*fast*)

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

$m1\text{-step3 } Rs A B Kab Na Ts \equiv \{(s, s')\}.$
 — new guards:
 $Ts = clk s \wedge$ — fresh timestamp
 — rest as before:
 $(s, s') \in m1a\text{-step3 } Rs A B Kab Na [aNum Ts]$
 $\}$

definition — by A , refines $m1x\text{-step5}$

$m1\text{-step4} :: [rid-t, agent, agent, nonce, key, time, time] \Rightarrow 'x m1\text{-trans}$

where

$m1\text{-step4 } Ra A B Na Kab Ts Ta \equiv \{(s, s')\}.$
 — previous guards:
 $runs s Ra = Some (Init, [A, B], []) \wedge$
 $(Kab \notin Domain (leak s) \longrightarrow (Kab, A) \in azC (runs s)) \wedge$ — authorization guard
 $Na = Ra\$na \wedge$ — fix parameter
 — guard for agreement with server on (Kab, B, Na, isl) ,
 — where $isl = take is-len nla$; injectiveness by including Na
 $(A \notin bad \longrightarrow (\exists Rs. Kab = sesK (Rs\$sk) \wedge$
 $runs s Rs = Some (Serv, [A, B], [aNon Na, aNum Ts]))) \wedge$
 — new guards:
 $Ta = clk s \wedge$ — fresh timestamp
 $clk s < Ts + Ls \wedge$ — 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 $m1x\text{-step4}$

$m1\text{-step5} :: [rid-t, agent, agent, key, time, time] \Rightarrow 'x m1\text{-trans}$

where

$m1\text{-step5 } Rb A B Kab Ts Ta \equiv \{(s, s')\}.$
 — previous guards:
 $runs s Rb = Some (Resp, [A, B], []) \wedge$
 $(Kab \notin Domain (leak s) \longrightarrow (Kab, B) \in azC (runs s)) \wedge$ — authorization guard
 — guard for showing agreement with server on (Kab, A, rsl) ,
 — where $rsl = take rs-len nlb$; this agreement is non-injective
 $(B \notin bad \longrightarrow (\exists Rs Na. Kab = sesK (Rs\$sk) \wedge$
 $runs s Rs = Some (Serv, [A, B], [aNon Na, aNum Ts]))) \wedge$
 — new guards:
 — guard for showing agreement with initiator A on (Kab, Ts, Ta)
 $(A \notin bad \longrightarrow B \notin bad \longrightarrow$
 $(\exists Ra nl. runs s Ra = Some (Init, [A, B], aKey Kab \# aNum Ts \# aNum Ta \# nl))) \wedge$
 — ensure recentness of session key
 $clk s < Ts + Ls \wedge$
 — check validity of authenticator and prevent its replay

— 'replays' with fresh authenticator ok!
 $clk\ s < Ta + La \wedge$
 $(B, Kab, Ta) \notin cache\ s \wedge$

— actions:
 $s' = s \langle$
 $runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])),$
 $cache := insert\ (B, Kab, Ta)\ (cache\ s)$
 \rangle
 $\}$

definition — by A , refines $skip$

$m1-step6 :: [rid-t, agent, agent, nonce, key, time, time] \Rightarrow 'x\ m1-trans$

where

$m1-step6\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta \equiv \{(s, s')\}.$

$runs\ s\ Ra = Some\ (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta]) \wedge$ — 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 $m1a-leak$

$m1-leak :: [rid-t, agent, agent, nonce, time] \Rightarrow 'x\ m1-trans$

where

$m1-leak\ Rs\ A\ B\ Na\ Ts \equiv \{(s, s1)\}.$

— 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 $skip$

$m1-tick :: time \Rightarrow 'x\ m1-trans$

where

$m1-tick\ T \equiv \{(s, s')\}.$

$s' = s \langle clk := clk\ s + T \rangle$

$\}$

Purge event: purge cache of expired timestamps

definition — refines *skip*
m1-purge :: *agent* \Rightarrow 'x *m1-trans*

where

m1-purge *A* \equiv $\{(s, s') \mid$
 $s' = s \setminus$
 $\text{cache} := \text{cache } s - \{(A, K, T) \mid A \ K \ T.$
 $(A, K, T) \in \text{cache } s \wedge T + La \leq \text{clk } s$
 $\}$
 $\}$

3.4.3 Specification

definition

m1-init :: *m1-state set*

where

m1-init \equiv $\{ \setminus \text{runs} = \text{Map.empty}, \text{leak} = \text{corrKey} \times \{\text{undefined}\}, \text{clk} = 0, \text{cache} = \{\} \setminus \}$

definition

m1-trans :: 'x *m1-trans* **where**
m1-trans \equiv $(\bigcup A \ B \ Ra \ Rb \ Rs \ Na \ Kab \ Ts \ Ta \ T.$
 $m1\text{-step1 } Ra \ A \ B \ Na \cup$
 $m1\text{-step2 } Rb \ A \ B \cup$
 $m1\text{-step3 } Rs \ A \ B \ Kab \ Na \ Ts \cup$
 $m1\text{-step4 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta \cup$
 $m1\text{-step5 } Rb \ A \ B \ Kab \ Ts \ Ta \cup$
 $m1\text{-step6 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta \cup$
 $m1\text{-leak } Rs \ A \ B \ Na \ Ts \cup$
 $m1\text{-tick } T \cup$
 $m1\text{-purge } A \cup$
 Id
 $)$

definition

m1 :: (*m1-state*, *m1-obs*) *spec* **where**

m1 \equiv $($
 $\text{init} = m1\text{-init},$
 $\text{trans} = m1\text{-trans},$
 $\text{obs} = id$
 $)$

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*

by (*simp add: m1-def*)

3.4.4 Invariants

inv0: Finite domain

There are only finitely many runs. This is needed to establish the responder/initiator agreement.

definition

$m1\text{-inv0}\text{-fin} :: 'x\ m1\text{-pred}$

where

$m1\text{-inv0}\text{-fin} \equiv \{s.\ finite\ (dom\ (runs\ s))\}$

lemmas $m1\text{-inv0}\text{-fin}I = m1\text{-inv0}\text{-fin}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m1\text{-inv0}\text{-fin}E\ [elim] = m1\text{-inv0}\text{-fin}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m1\text{-inv0}\text{-fin}D = m1\text{-inv0}\text{-fin}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format}]$

Invariance proofs.

lemma $PO\text{-}m1\text{-inv0}\text{-fin}\text{-init}\ [iff]:$

$init\ m1 \subseteq m1\text{-inv0}\text{-fin}$

by ($auto\ simp\ add: m1\text{-defs}\ intro!: m1\text{-inv0}\text{-fin}I$)

lemma $PO\text{-}m1\text{-inv0}\text{-fin}\text{-trans}\ [iff]:$

$\{m1\text{-inv0}\text{-fin}\}\ trans\ m1\ \{>\ m1\text{-inv0}\text{-fin}\}$

by ($auto\ simp\ add: PO\text{-hoare}\text{-defs}\ m1\text{-defs}\ intro!: m1\text{-inv0}\text{-fin}I$)

lemma $PO\text{-}m1\text{-inv0}\text{-fin}\ [iff]: reach\ m1 \subseteq m1\text{-inv0}\text{-fin}$

by ($rule\ inv\text{-rule}\text{-incr},\ auto\ del: subsetI$)

inv1: Caching invariant for responder

definition

$m1\text{-inv1r}\text{-cache} :: 'x\ m1\text{-pred}$

where

$m1\text{-inv1r}\text{-cache} \equiv \{s.\ \forall Rb\ A\ B\ Kab\ Ts\ Ta\ nl.$

$runs\ s\ Rb = Some\ (Resp,\ [A,\ B],\ aKey\ Kab\ \#\ aNum\ Ts\ \#\ aNum\ Ta\ \#\ nl) \longrightarrow$

$clk\ s < Ta + La \longrightarrow$

$(B,\ Kab,\ Ta) \in cache\ s$

$\}$

lemmas $m1\text{-inv1r}\text{-cache}I = m1\text{-inv1r}\text{-cache}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m1\text{-inv1r}\text{-cache}E\ [elim] = m1\text{-inv1r}\text{-cache}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m1\text{-inv1r}\text{-cache}D = m1\text{-inv1r}\text{-cache}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

Invariance proof

lemma $PO\text{-}m1\text{-inv1r}\text{-cache}\text{-init}\ [iff]:$

$init\ m1 \subseteq m1\text{-inv1r}\text{-cache}$

by ($auto\ simp\ add: m1\text{-defs}\ intro!: m1\text{-inv1r}\text{-cache}I$)

lemma $PO\text{-}m1\text{-inv1r}\text{-cache}\text{-trans}\ [iff]:$

$\{m1\text{-inv1r}\text{-cache}\}\ trans\ m1\ \{>\ m1\text{-inv1r}\text{-cache}\}$

apply ($auto\ simp\ add: PO\text{-hoare}\text{-defs}\ m1\text{-defs}\ intro!: m1\text{-inv1r}\text{-cache}I$

$dest: m1\text{-inv1r}\text{-cache}D$)

apply ($auto\ dest: m1\text{-inv1r}\text{-cache}D$)

done

lemma *PO-m1-inv1r-cache* [iff]: $reach\ m1 \subseteq m1\text{-inv1r-cache}$
by (rule *inv-rule-basic*) (auto del: *subsetI*)

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*

apply (auto simp add: *map-runs-def elim!*: *knC.cases*)

apply (*rename-tac b*, *case-tac b*, auto)

apply (*rename-tac b*, *case-tac b*, auto)

apply (rule *knC-init*, auto simp add: *map-runs-def*)

apply (rule *knC-resp*, auto simp add: *map-runs-def*)

apply (rule *tac knC-serv*, auto simp add: *map-runs-def*)

done

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* $\Rightarrow m1a\text{-obs}$ **where**

med1a1 *s* $\equiv \langle runs = runs1a1\ (runs\ s), m1x\text{-state.leak} = Domain\ (leak\ s) \rangle$

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*:

$\{R1a1\}$
 $(m1a\text{-step1 } Ra \ A \ B \ Na), (m1\text{-step1 } Ra \ A \ B \ Na)$
 $\{> R1a1\}$
by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs*)

lemma *PO-m1-step2-refines-m1a-step2*:
 $\{R1a1\}$
 $(m1a\text{-step2 } Rb \ A \ B), (m1\text{-step2 } Rb \ A \ B)$
 $\{> R1a1\}$
by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs*)

lemma *PO-m1-step3-refines-m1a-step3*:
 $\{R1a1\}$
 $(m1a\text{-step3 } Rs \ A \ B \ Kab \ Na \ [aNum \ Ts]), (m1\text{-step3 } Rs \ A \ B \ Kab \ Na \ Ts)$
 $\{> R1a1\}$
by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs*)

lemma *PO-m1-step4-refines-m1a-step4*:
 $\{R1a1\}$
 $(m1a\text{-step4 } Ra \ A \ B \ Na \ Kab \ [aNum \ Ts]), (m1\text{-step4 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta)$
 $\{> R1a1\}$
by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs map-runs-def*)

lemma *PO-m1-step5-refines-m1a-step5*:
 $\{R1a1\}$
 $(m1a\text{-step5 } Rb \ A \ B \ Kab \ [aNum \ Ts]), (m1\text{-step5 } Rb \ A \ B \ Kab \ Ts \ Ta)$
 $\{> R1a1\}$
by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs map-runs-def*)

lemma *PO-m1-step6-refines-m1a-skip*:
 $\{R1a1\}$
 $Id, (m1\text{-step6 } Ra \ A \ B \ Na \ Kab \ Ts \ Ta)$
 $\{> R1a1\}$
by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs map-runs-def*)

lemma *PO-m1-leak-refines-m1a-leak*:
 $\{R1a1\}$
 $(m1a\text{-leak } Rs), (m1\text{-leak } Rs \ A \ B \ Na \ Ts)$
 $\{> R1a1\}$
by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs map-runs-def dest: dom-lemmas*)

lemma *PO-m1-tick-refines-m1a-skip*:
 $\{R1a1\}$
 $Id, (m1\text{-tick } T)$
 $\{> R1a1\}$
by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs map-runs-def*)

lemma *PO-m1-purge-refines-m1a-skip*:
 $\{R1a1\}$
 $Id, (m1\text{-purge } A)$
 $\{> R1a1\}$
by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs map-runs-def*)

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)
by (*auto simp add: R1a1-defs m1-defs intro!: s0g-secrecyI*)

lemma *PO-m1-refines-trans-m1a* [iff]:
 {*R1a1*}
 (*trans m1a*), (*trans m1*)
 {> *R1a1*}
apply (*auto simp add: m1-def m1-trans-def m1a-def m1a-trans-def*
intro!: PO-m1-trans-refines-m1a-trans)
apply (*force intro!: PO-m1-trans-refines-m1a-trans*)+
done

Observation consistency.

lemma *obs-consistent-med1a1* [iff]:
obs-consistent R1a1 med1a1 m1a m1
by (*auto simp add: obs-consistent-def R1a1-def m1a-def m1-def*)

Refinement result.

lemma *PO-m1-refines-m1a* [iff]:
refines R1a1 med1a1 m1a m1
by (*rule Refinement-basic*) (*auto del: subsetI*)

lemma *m1-implements-m1a* [iff]: *implements med1a1 m1a m1*
by (*rule refinement-soundness*) (*fast*)

inv (inherited): Secrecy

Secrecy, as external and internal invariant

definition

m1-secrecy :: '*x m1-pred where*
m1-secrecy \equiv {*s. knC (runs s)* \subseteq *azC (runs s) \cup Domain (leak s) \times 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* \subseteq *m1-secrecy*
apply (*rule-tac Q=m1x-secrecy in external-invariant-translation*)
apply (*auto del: subsetI*)
apply (*fastforce simp add: med1a1-def intro!: m1-secrecyI*)
done

lemma *PO-m1-secrecy* [iff]: *reach m1* \subseteq *m1-secrecy*
by (*rule external-to-internal-invariant*) (*auto del: subsetI*)

inv (inherited): Responder auth server.

definition

$m1\text{-inv}2r\text{-serv} :: 'x\ m1r\text{-pred}$

where

$m1\text{-inv}2r\text{-serv} \equiv \{s. \forall A\ B\ Rb\ Kab\ Ts\ nlb.$
 $B \notin bad \longrightarrow$
 $runs\ s\ Rb = Some\ (Resp, [A, B], aKey\ Kab\ \#\ aNum\ Ts\ \#\ nlb) \longrightarrow$
 $(\exists\ Rs\ Na. Kab = sesK\ (Rs\$sk) \wedge$
 $runs\ s\ Rs = Some\ (Serv, [A, B], [aNon\ Na, aNum\ Ts]))$
 $\}$

lemmas $m1\text{-inv}2r\text{-serv}I = m1\text{-inv}2r\text{-serv}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m1\text{-inv}2r\text{-serv}E\ [elim] = m1\text{-inv}2r\text{-serv}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m1\text{-inv}2r\text{-serv}D = m1\text{-inv}2r\text{-serv}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ -1]$

Proof of invariance.

lemma $PO\text{-}m1\text{-inv}2r\text{-serv}\ [iff]:\ reach\ m1 \subseteq m1\text{-inv}2r\text{-serv}$

apply ($rule\text{-tac}\ Sa=m1a\ \mathbf{and}\ Pa=m1a\text{-inv}2r\text{-serv}$
 $\ \mathbf{and}\ Qa=m1a\text{-inv}2r\text{-serv}\ \mathbf{and}\ Q=m1\text{-inv}2r\text{-serv}$
 $\ \mathbf{in}\ internal\text{-invariant}\text{-translation}$)

apply ($auto\ del:\ subsetI$)

— 1 subgoal

apply ($auto\ simp\ add:\ vimage\text{-def}\ intro!:\ m1\text{-inv}2r\text{-serv}I$)

apply ($simp\ add:\ m1a\text{-inv}2r\text{-serv}\text{-def}\ med1a1\text{-def}$)

apply ($rename\text{-tac}\ x\ A\ B\ Rb\ Kab\ Ts\ nlb$)

apply ($drule\text{-tac}\ x=A\ \mathbf{in}\ spec$)

apply ($drule\text{-tac}\ x=B\ \mathbf{in}\ spec,\ clarsimp$)

apply ($drule\text{-tac}\ x=Rb\ \mathbf{in}\ spec$)

apply ($drule\text{-tac}\ x=Kab\ \mathbf{in}\ spec$)

apply ($drule\text{-tac}\ x=[aNum\ Ts]\ \mathbf{in}\ spec$)

apply ($auto\ simp\ add:\ map\text{-runs}\text{-def}$)

done

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

$m1\text{-inv}2i\text{-serv} \equiv \{s. \forall A\ B\ Ra\ Kab\ Ts\ nla.$
 $A \notin bad \longrightarrow$
 $runs\ s\ Ra = Some\ (Init, [A, B], aKey\ Kab\ \#\ aNum\ Ts\ \#\ nla) \longrightarrow$
 $(\exists\ Rs. Kab = sesK\ (Rs\$sk) \wedge$
 $runs\ s\ Rs = Some\ (Serv, [A, B], [aNon\ (Ra\$na), aNum\ Ts]))$
 $\}$

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 $m1\text{-inv}2i\text{-serv}D = m1\text{-inv}2i\text{-serv}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ -1]$

Proof of invariance.

```

lemma PO-m1-inv2i-serv [iff]: reach  $m1 \subseteq m1\text{-inv2i-serv}$ 
apply (rule-tac  $Pa=m1a\text{-inv2i-serv}$  and  $Qa=m1a\text{-inv2i-serv}$  and  $Q=m1\text{-inv2i-serv}$ 
  in internal-invariant-translation)
apply (auto del: subsetI)
— 1 subgoal
apply (auto simp add: m1a-inv2i-serv-def med1a1-def vimage-def intro!: m1-inv2i-servI)
apply (rename-tac  $x A B Ra Kab Ts nla$ )
apply (drule-tac  $x=A$  in spec, clarsimp)
apply (drule-tac  $x=B$  in spec)
apply (drule-tac  $x=Ra$  in spec)
apply (drule-tac  $x=Kab$  in spec)
apply (drule-tac  $x=[aNum Ts]$  in spec)
apply (auto simp add: map-runs-def)
done

```

```

declare PO-m1-inv2i-serv [THEN subsetD, intro]

```

inv (inherited): Initiator key freshness

definition

```

m1-inv1-ifresh :: 'x m1-pred

```

where

```

m1-inv1-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-inv1-ifreshI = m1-inv1-ifresh-def [THEN setc-def-to-intro, rule-format]

```

```

lemmas m1-inv1-ifreshE [elim] = m1-inv1-ifresh-def [THEN setc-def-to-elim, rule-format]

```

```

lemmas m1-inv1-ifreshD = m1-inv1-ifresh-def [THEN setc-def-to-dest, rule-format, rotated 1]

```

```

lemma PO-m1-ifresh [iff]: reach  $m1 \subseteq m1\text{-inv1-ifresh}$ 

```

```

apply (rule-tac  $Pa=m1a\text{-inv1-ifresh}$  and  $Qa=m1a\text{-inv1-ifresh}$  and  $Q=m1\text{-inv1-ifresh}$ 
  in internal-invariant-translation)

```

```

apply (auto del: subsetI)

```

```

apply (auto simp add: med1a1-def map-runs-def vimage-def m1-inv1-ifresh-def)

```

```

done

```

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

```

risig = key  $\times$  time  $\times$  time

```

abbreviation

$ri\text{-}running :: [runs\text{-}t, agent, agent, key, time, time] \Rightarrow rid\text{-}t\text{ set}$

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

$ri\text{-}commit :: [runs\text{-}t, agent, agent, key, time, time] \Rightarrow rid\text{-}t\text{ set}$

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]:

$runz = Map.empty \Longrightarrow ri\text{-}runs2sigs\ runz = (\lambda s. 0)$

by (rule ext, erule rev-mp)

(rule ri-runs2sigs.induct, auto)

lemma $finite\text{-}ri\text{-}running$ [simp, intro]:

$finite\ (dom\ runz) \Longrightarrow finite\ (ri\text{-}running\ runz\ A\ B\ Kab\ Ts\ Ta)$

by (auto intro: finite-subset dest: dom-lemmas)

lemma $finite\text{-}ri\text{-}commit$ [simp, intro]:

$finite\ (dom\ runz) \Longrightarrow finite\ (ri\text{-}commit\ runz\ A\ B\ Kab\ Ts\ Ta)$

by (auto intro: finite-subset dest: dom-lemmas)

Update lemmas

lemma *ri-runs2sigs-upd-init-none* [simp]:

[[$Na \notin \text{dom runz}$]]

$\implies \text{ri-runs2sigs} (\text{runz}(Na \mapsto (\text{Init}, [A, B], []))) = \text{ri-runs2sigs runz}$

by (rule ext, erule rev-mp, rule ri-runs2sigs.induct)

(auto dest: dom-lemmas)

lemma *ri-runs2sigs-upd-resp-none* [simp]:

[[$Rb \notin \text{dom runz}$]]

$\implies \text{ri-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], []))) = \text{ri-runs2sigs runz}$

by (rule ext, erule rev-mp, rule ri-runs2sigs.induct)

(auto dest: dom-lemmas)

lemma *ri-runs2sigs-upd-serv* [simp]:

[[$Rs \notin \text{dom runz}$]]

$\implies \text{ri-runs2sigs} (\text{runz}(Rs \mapsto (\text{Serv}, [A, B], [a\text{Non } Na, a\text{Num } Ts])))$

$= \text{ri-runs2sigs runz}$

by (rule ext, erule rev-mp, rule ri-runs2sigs.induct)

(auto dest: dom-lemmas)

lemma *ri-runs2sigs-upd-init-some* [simp]:

[[$\text{runz } Ra = \text{Some} (\text{Init}, [A, B], []); \text{finite} (\text{dom runz})$]]

$\implies \text{ri-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [a\text{Key } Kab, a\text{Num } Ts, a\text{Num } Ta]))) =$

$(\text{ri-runs2sigs runz})($

$\text{Running } [B, A] (Kab, Ts, Ta) :=$

$\text{Suc} (\text{card} (\text{ri-running runz } A B Kab Ts Ta)))$

apply (rule ext, (erule rev-mp)+)

apply (rule ri-runs2sigs.induct, auto)

— 1 subgoal

apply (rename-tac runz)

apply (rule-tac s=card (insert Ra (ri-running runz A B Kab Ts Ta)))

in trans, fast, auto)

done

lemma *ri-runs2sigs-upd-resp-some* [simp]:

[[$\text{runz } Rb = \text{Some} (\text{Resp}, [A, B], []); \text{finite} (\text{dom runz})$]]

$\implies \text{ri-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], [a\text{Key } Kab, a\text{Num } Ts, a\text{Num } Ta]))) =$

$(\text{ri-runs2sigs runz})($

$\text{Commit } [B, A] (Kab, Ts, Ta) :=$

$\text{Suc} (\text{card} (\text{ri-commit runz } A B Kab Ts Ta)))$

apply (rule ext, (erule rev-mp)+)

apply (rule ri-runs2sigs.induct, auto)

— 1 subgoal

apply (rename-tac runz)

apply (rule-tac s=card (insert Rb (ri-commit runz A B Kab Ts Ta)))

in trans, fast, auto)

done

lemma *ri-runs2sigs-upd-init-some2* [simp]:

[[$\text{runz } Ra = \text{Some} (\text{Init}, [A, B], [a\text{Key } Kab, a\text{Num } Ts, a\text{Num } Ta])$]]

$\implies \text{ri-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [a\text{Key } Kab, a\text{Num } Ts, a\text{Num } Ta, \text{END}]))) =$

ri-runs2sigs runz

by (rule ext, erule rev-mp, rule ri-runs2sigs.induct)
(auto dest: dom-lemmas)

Refinement proof

lemma *PO-m1-step1-refines-a0-ri-skip*:

{*R-a0iim1-ri*}
Id, (m1-step1 Ra A B Na)
{> *R-a0iim1-ri*}

by (auto simp add: PO-rhoare-defs *R-a0iim1-ri-defs* m1-defs)

lemma *PO-m1-step2-refines-a0-ri-skip*:

{*R-a0iim1-ri*}
Id, (m1-step2 Rb A B)
{> *R-a0iim1-ri*}

by (auto simp add: PO-rhoare-defs *R-a0iim1-ri-defs* m1-defs)

lemma *PO-m1-step3-refines-a0-ri-skip*:

{*R-a0iim1-ri*}
Id, (m1-step3 Rs A B Kab Na Ts)
{> *R-a0iim1-ri*}

by (auto simp add: PO-rhoare-defs *R-a0iim1-ri-defs* m1-defs)

lemma *PO-m1-step4-refines-a0-ri-running*:

{*R-a0iim1-ri* \cap UNIV \times m1-inv0-fin}
(a0i-running [B, A] (Kab, Ts, Ta)), (m1-step4 Ra A B Na Kab Ts Ta)
{> *R-a0iim1-ri*}

by (auto simp add: PO-rhoare-defs *R-a0iim1-ri-defs* a0i-defs m1-defs)

lemma *PO-m1-step5-refines-a0-ri-commit*:

{*R-a0iim1-ri* \cap UNIV \times (m1-inv1r-cache \cap m1-inv0-fin)}
(a0i-commit [B, A] (Kab, Ts, Ta)), (m1-step5 Rb A B Kab Ts Ta)
{> *R-a0iim1-ri*}

apply (auto simp add: PO-rhoare-defs *R-a0iim1-ri-defs* a0i-defs m1-defs)

— 2 subgoals

apply (rename-tac s t aa ab ac ba Rs Na Ra nl,

subgoal-tac

card (ri-commit (runs t) A B (sesK (Rs\$sk)) Ts Ta) = 0 \wedge
card (ri-running (runs t) A B (sesK (Rs\$sk)) Ts Ta) > 0, auto)

apply (rename-tac s t Rs Na Ra nl,

subgoal-tac

card (ri-commit (runs t) A B (sesK (Rs\$sk)) Ts Ta) = 0 \wedge
card (ri-running (runs t) A B (sesK (Rs\$sk)) Ts Ta) > 0, auto)

done

lemma *PO-m1-step6-refines-a0-ri-skip*:

{*R-a0iim1-ri*}
Id, (m1-step6 Ra A B Na Kab Ts Ta)
{> *R-a0iim1-ri*}

by (auto simp add: PO-rhoare-defs *R-a0iim1-ri-defs* m1-defs)

lemma *PO-m1-leak-refines-a0-ri-skip*:

{*R-a0iim1-ri*}

$Id, (m1-leak\ Rs\ A\ B\ Na\ Ts)$
 $\{> R-a0iim1-ri\}$
by (auto simp add: PO-rhoare-defs R-a0iim1-ri-defs a0i-defs m1-defs)

lemma PO-m1-tick-refines-a0-ri-skip:
 $\{R-a0iim1-ri\}$
 $Id, (m1-tick\ T)$
 $\{> R-a0iim1-ri\}$
by (auto simp add: PO-rhoare-defs R-a0iim1-ri-defs m1-defs)

lemma PO-m1-purge-refines-a0-ri-skip:
 $\{R-a0iim1-ri\}$
 $Id, (m1-purge\ A)$
 $\{> R-a0iim1-ri\}$
by (auto simp add: PO-rhoare-defs R-a0iim1-ri-defs m1-defs)

All together now...

lemmas PO-m1-trans-refines-a0-ri-trans =
PO-m1-step1-refines-a0-ri-skip PO-m1-step2-refines-a0-ri-skip
PO-m1-step3-refines-a0-ri-skip PO-m1-step4-refines-a0-ri-running
PO-m1-step5-refines-a0-ri-commit PO-m1-step6-refines-a0-ri-skip
PO-m1-leak-refines-a0-ri-skip PO-m1-tick-refines-a0-ri-skip
PO-m1-purge-refines-a0-ri-skip

lemma PO-m1-refines-init-a0-ri [iff]:
 $init\ m1 \subseteq R-a0iim1-ri \text{“}(init\ a0i)$
by (auto simp add: R-a0iim1-ri-defs a0i-defs m1-defs
intro!: exI [where $x=(signals = \lambda s. 0, corrupted = \{\})$])

lemma PO-m1-refines-trans-a0-ri [iff]:
 $\{R-a0iim1-ri \cap a0i-inv1-iagree \times (m1-inv1r-cache \cap m1-inv0-fin)\}$
 $(trans\ a0i), (trans\ m1)$
 $\{> R-a0iim1-ri\}$
by (force simp add: m1-def m1-trans-def a0i-def a0i-trans-def
intro!: PO-m1-trans-refines-a0-ri-trans)

lemma obs-consistent-med-a0iim1-ri [iff]:
obs-consistent
 $(R-a0iim1-ri \cap a0i-inv1-iagree \times (m1-inv1r-cache \cap m1-inv0-fin))$
 $med-a0iim1-ri\ a0i\ m1$
by (auto simp add: obs-consistent-def R-a0iim1-ri-def med-a0iim1-ri-def
a0i-def m1-def)

Refinement result.

lemma PO-m1-refines-a0ii-ri [iff]:
refines
 $(R-a0iim1-ri \cap a0i-inv1-iagree \times (m1-inv1r-cache \cap m1-inv0-fin))$
 $med-a0iim1-ri\ a0i\ m1$
by (rule Refinement-using-invariants) (auto)

lemma m1-implements-a0ii-ri: implements med-a0iim1-ri a0i m1
by (rule refinement-soundness) (fast)

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

$m1\text{-inv}3r\text{-init} :: 'x\ m1\text{-pred}$

where

$$\begin{aligned} m1\text{-inv}3r\text{-init} &\equiv \{s. \forall A\ B\ Rb\ Kab\ Ts\ Ta\ nlb. \\ &B \notin bad \longrightarrow A \notin bad \longrightarrow Kab \notin Domain\ (leak\ s) \longrightarrow \\ &runs\ s\ Rb = Some\ (Resp, [A, B], aKey\ Kab\ \# aNum\ Ts\ \# aNum\ Ta\ \# nlb) \longrightarrow \\ &(\exists Ra\ nla. \\ &runs\ s\ Ra = Some\ (Init, [A, B], aKey\ Kab\ \# aNum\ Ts\ \# aNum\ Ta\ \# nla)) \\ &\} \end{aligned}$$

lemmas $m1\text{-inv}3r\text{-init}I = m1\text{-inv}3r\text{-init}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m1\text{-inv}3r\text{-init}E\ [elim] = m1\text{-inv}3r\text{-init}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m1\text{-inv}3r\text{-init}D = m1\text{-inv}3r\text{-init}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ -1]$

Invariance proof.

lemma $PO\text{-}m1\text{-inv}3r\text{-init}\ [iff]:\ reach\ m1 \subseteq m1\text{-inv}3r\text{-init}$

apply ($rule\ INV\text{-from}\text{-Refinement}\text{-basic}\ [OF\ PO\text{-}m1\text{-refines}\text{-a}0ii\text{-ri}]$)

apply ($auto\ simp\ add: R\text{-a}0iim1\text{-ri}\text{-def}\ a0i\text{-inv}1\text{-iagree}\text{-def}\ intro!: m1\text{-inv}3r\text{-init}I$)

— 1 subgoal

apply ($rename\text{-tac}\ s\ A\ B\ Rb\ Kab\ Ts\ Ta\ nlb\ a$)

apply ($drule\text{-tac}\ x=[B, A]\ in\ spec, clarsimp$)

apply ($drule\text{-tac}\ x=Kab\ in\ spec$)

apply ($drule\text{-tac}\ x=Ts\ in\ spec$)

apply ($drule\text{-tac}\ x=Ta\ in\ spec$)

apply ($subgoal\text{-tac}\ card\ (ri\text{-commit}\ (runs\ s)\ A\ B\ Kab\ Ts\ Ta) > 0, auto$)

done

inv4: Key freshness for responder

definition

$m1\text{-inv}4\text{-rfresh} :: 'x\ m1\text{-pred}$

where

$$\begin{aligned} m1\text{-inv}4\text{-rfresh} &\equiv \{s. \forall Rb1\ Rb2\ A1\ A2\ B1\ B2\ Kab\ Ts1\ Ts2\ Ta1\ Ta2. \\ &runs\ s\ Rb1 = Some\ (Resp, [A1, B1], [aKey\ Kab, aNum\ Ts1, aNum\ Ta1]) \longrightarrow \\ &runs\ s\ Rb2 = Some\ (Resp, [A2, B2], [aKey\ Kab, aNum\ Ts2, aNum\ Ta2]) \longrightarrow \\ &B1 \notin bad \longrightarrow A1 \notin bad \longrightarrow Kab \notin Domain\ (leak\ s) \longrightarrow \\ &Rb1 = Rb2 \\ &\} \end{aligned}$$

lemmas $m1\text{-inv}4\text{-rfresh}I = m1\text{-inv}4\text{-rfresh}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m1\text{-inv}4\text{-rfresh}E\ [elim] = m1\text{-inv}4\text{-rfresh}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m1\text{-inv}4\text{-rfresh}D = m1\text{-inv}4\text{-rfresh}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

Proof of key freshness for responder. All cases except step5 are straightforward.

lemma $PO\text{-}m1\text{-inv}4\text{-rfresh}\text{-step}5:$

$$\{m1\text{-inv}4\text{-rfresh} \cap m1\text{-inv}3r\text{-init} \cap m1\text{-inv}2r\text{-serv} \cap m1\text{-inv}1r\text{-cache} \cap m1\text{-secrecy} \cap m1\text{-inv}1\text{-ifresh}\}$$

```

    (m1-step5 Rb A B Kab Ts Ta)
    {> m1-inv4-rfresh}
apply (auto simp add: PO-hoare-defs m1-defs intro!: m1-inv4-rfreshI)
apply (auto dest: m1-inv4-rfreshD)
apply (auto dest: m1-inv2r-servD)

— 5 subgoals
apply (drule m1-inv2r-servD, auto)
apply (elim azC.cases, auto)

apply (drule m1-inv2r-servD, auto)
apply (elim azC.cases, auto)

apply (drule m1-inv2r-servD, auto)
apply (elim azC.cases, auto)

apply (rename-tac Rb2 A2 B2 Ts2 Ta2 s Rs Na Ra nl)
apply (case-tac B2 ∈ bad)
  apply (thin-tac (sesK (Rs$sk), B) ∈ azC (runs s))
  apply (subgoal-tac (sesK (Rs$sk), B2) ∈ azC (runs s))
  apply (erule azC.cases, auto)
  apply (erule m1-secrecyE, auto)

  apply (case-tac A2 ∈ bad, auto dest: m1-inv2r-servD)
  apply (frule m1-inv3r-initD, auto)
  apply (rename-tac Raa nla, subgoal-tac Raa = Ra, auto) — uses cache invariant

apply (frule m1-inv3r-initD, auto)
apply (rename-tac Raa nla, subgoal-tac Raa = Ra, auto) — uses cache invariant
done

lemmas PO-m1-inv4-rfresh-step5-lemmas =
  PO-m1-inv4-rfresh-step5

lemma PO-m1-inv4-rfresh-init [iff]:
  init m1 ⊆ m1-inv4-rfresh
by (auto simp add: m1-defs intro!: m1-inv4-rfreshI)

lemma PO-m1-inv4-rfresh-trans [iff]:
  {m1-inv4-rfresh ∩ m1-inv3r-init ∩ m1-inv2r-serv ∩ m1-inv1r-cache ∩
   m1-secrecy ∩ m1-inv1-ifresh}
  trans m1
  {> m1-inv4-rfresh}
by (auto simp add: m1-def m1-trans-def intro!: PO-m1-inv4-rfresh-step5-lemmas)
  (auto simp add: PO-hoare-defs m1-defs intro!: m1-inv4-rfreshI dest: m1-inv4-rfreshD)

lemma PO-m1-inv4-rfresh [iff]: reach m1 ⊆ m1-inv4-rfresh
apply (rule-tac
  J=m1-inv3r-init ∩ m1-inv2r-serv ∩ m1-inv1r-cache ∩ m1-secrecy ∩ m1-inv1-ifresh
  in inv-rule-incr)
apply (auto simp add: Int-assoc del: subsetI)
done

```

lemma *PO-m1-obs-inv4-rfresh* [iff]: *oreach* $m1 \subseteq m1\text{-inv4-rfresh}$
by (*rule external-from-internal-invariant*)
(auto del: subsetI)

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

irsig = *key* × *time* × *time*

abbreviation

ir-running :: [*runs-t*, *agent*, *agent*, *key*, *time*, *time*] ⇒ *rid-t set*

where

ir-running *runz* *A B Kab Ts Ta* ≡ {*Rb*. ∃ *nl*.
runz *Rb* = *Some* (*Resp*, [*A*, *B*], *aKey* *Kab* # *aNum* *Ts* # *aNum* *Ta* # *nl*)
}

abbreviation

ir-commit :: [*runs-t*, *agent*, *agent*, *key*, *time*, *time*] ⇒ *rid-t set*

where

ir-commit *runz* *A B Kab Ts Ta* ≡ {*Ra*. ∃ *nl*.
runz *Ra* = *Some* (*Init*, [*A*, *B*], *aKey* *Kab* # *aNum* *Ts* # *aNum* *Ta* # *END* # *nl*)
}

fun

ir-runs2sigs :: *runs-t* ⇒ *risig signal* ⇒ *nat*

where

ir-runs2sigs *runz* (*Running* [*A*, *B*] (*Kab*, *Ts*, *Ta*)) =
card (*ir-running* *runz* *A B Kab Ts Ta*)

| *ir-runs2sigs* *runz* (*Commit* [*A*, *B*] (*Kab*, *Ts*, *Ta*)) =
card (*ir-commit* *runz* *A B Kab Ts Ta*)

| *ir-runs2sigs* *runz* - = 0

Simulation relation and mediator function. We map completed initiator and responder runs to commit and running signals, respectively.

definition

med-a0iim1-ir :: *m1-obs* ⇒ *irsig a0i-obs* **where**
med-a0iim1-ir *o1* ≡ (| *signals* = *ir-runs2sigs* (*runs* *o1*), *corrupted* = {} |)

definition

R-a0iim1-ir :: (*irsig a0i-state* × *m1-state*) *set* **where**
R-a0iim1-ir ≡ {(*s*, *t*). *signals* *s* = *ir-runs2sigs* (*runs* *t*) ∧ *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]:
 $runz = Map.empty \implies ir\text{-}runs2sigs\ runz = (\lambda s. 0)$
by (rule ext, erule rev-mp)
(rule $ir\text{-}runs2sigs.induct$, auto)

lemma $ir\text{-}commit\text{-}finite$ [simp, intro]:
 $finite\ (dom\ runz) \implies finite\ (ir\text{-}commit\ runz\ A\ B\ Kab\ Ts\ Ta)$
by (auto intro: finite-subset dest: dom-lemmas)

Update lemmas

lemma $ir\text{-}runs2sigs\text{-}upd\text{-}init\text{-}none$ [simp]:
 $\llbracket Ra \notin dom\ runz \rrbracket$
 $\implies ir\text{-}runs2sigs\ (runz(Ra \mapsto (Init, [A, B], []))) = ir\text{-}runs2sigs\ runz$
by (rule ext, erule rev-mp)
(rule $ir\text{-}runs2sigs.induct$, auto dest: dom-lemmas)

lemma $ir\text{-}runs2sigs\text{-}upd\text{-}resp\text{-}none$ [simp]:
 $\llbracket Rb \notin dom\ runz \rrbracket$
 $\implies ir\text{-}runs2sigs\ (runz(Rb \mapsto (Resp, [A, B], []))) = ir\text{-}runs2sigs\ runz$
by (rule ext, erule rev-mp)
(rule $ir\text{-}runs2sigs.induct$, auto dest: dom-lemmas)

lemma $ir\text{-}runs2sigs\text{-}upd\text{-}serv$ [simp]:
 $\llbracket Rs \notin dom\ (runs\ y) \rrbracket$
 $\implies ir\text{-}runs2sigs\ ((runs\ y)(Rs \mapsto (Serv, [A, B], [aNon\ Na, aNum\ Ts])))$
 $= ir\text{-}runs2sigs\ (runs\ y)$
by (rule ext, erule rev-mp)
(rule $ir\text{-}runs2sigs.induct$, auto dest: dom-lemmas)

lemma $ir\text{-}runs2sigs\text{-}upd\text{-}init\text{-}some$ [simp]:
 $\llbracket runz\ Ra = Some\ (Init, [A, B], []) \rrbracket$
 $\implies ir\text{-}runs2sigs\ (runz(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta]))) =$
 $ir\text{-}runs2sigs\ runz$
by (rule ext, erule rev-mp)
(rule $ir\text{-}runs2sigs.induct$, auto dest: dom-lemmas)

lemma $ir\text{-}runs2sigs\text{-}upd\text{-}resp\text{-}some\text{-}raw$:
assumes
 $runz\ Rb = Some\ (Resp, [A, B], [])$
 $finite\ (dom\ runz)$
shows
 $ir\text{-}runs2sigs\ (runz(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])))\ s =$
 $((ir\text{-}runs2sigs\ runz)($
 $Running\ [A, B]\ (Kab, Ts, Ta) :=$
 $Suc\ (card\ (ir\text{-}running\ runz\ A\ B\ Kab\ Ts\ Ta))))\ s$
using $assms$
proof (induct rule: $ir\text{-}runs2sigs.induct$)

case (1 runz A B Kab Ts Ta) **note** H = this
hence Rb \notin ir-running runz A B Kab Ts Ta **by** auto
moreover
with H **have**
card (insert Rb (ir-running runz A B Kab Ts Ta))
= Suc (card (ir-running runz A B Kab Ts Ta)) **by** auto
ultimately show ?case **by** (auto elim: subst)
qed (auto)

lemma ir-runs2sigs-upd-resp-some [simp]:
 \llbracket runz Rb = Some (Resp, [A, B], []); finite (dom runz) \rrbracket
 \implies ir-runs2sigs (runz(Rb \mapsto (Resp, [A, B], [aKey Kab, aNum Ts, aNum Ta]))) =
(ir-runs2sigs runz)(
Running [A, B] (Kab, Ts, Ta) :=
Suc (card (ir-running runz A B Kab Ts Ta)))
by (intro ext ir-runs2sigs-upd-resp-some-raw)

lemma ir-runs2sigs-upd-init-some2-raw:
assumes
runz Ra = Some (Init, [A, B], [aKey Kab, aNum Ts, aNum Ta])
finite (dom runz)
shows
ir-runs2sigs (runz(Ra \mapsto (Init, [A, B], [aKey Kab, aNum Ts, aNum Ta, END]))) s =
((ir-runs2sigs runz)(
Commit [A, B] (Kab, Ts, Ta) :=
Suc (card (ir-commit runz A B Kab Ts Ta)))) s
using assms

proof (induct runz s rule: ir-runs2sigs.induct)
case (2 runz A B Kab Ts Ta) **note** H = this
from H **have** Ra \notin ir-commit runz A B Kab Ts Ta **by** auto
moreover
with H **have**
card (insert Ra (ir-commit runz A B Kab Ts Ta))
= Suc (card (ir-commit runz A B Kab Ts Ta))
by (auto)
ultimately show ?case **by** (auto elim: subst)
qed (auto)

lemma ir-runs2sigs-upd-init-some2 [simp]:
 \llbracket runz Na = Some (Init, [A, B], [aKey Kab, aNum Ts, aNum Ta]); finite (dom runz) \rrbracket
 \implies ir-runs2sigs (runz(Na \mapsto (Init, [A, B], [aKey Kab, aNum Ts, aNum Ta, END]))) =
(ir-runs2sigs runz)(
Commit [A, B] (Kab, Ts, Ta) :=
Suc (card (ir-commit runz A B Kab Ts Ta)))
by (intro ir-runs2sigs-upd-init-some2-raw ext)

Refinement proof

lemma PO-m1-step1-refines-ir-a0ii-skip:
{R-a0iim1-ir}
Id, (m1-step1 Ra A B Na)
{> R-a0iim1-ir}
by (simp add: PO-rhoare-defs R-a0iim1-ir-defs m1-defs, safe, auto)

lemma *PO-m1-step2-refines-ir-a0ii-skip*:

$\{R\text{-}a0iim1\text{-}ir\}$

Id, (*m1-step2* *Rb A B*)

$\{> R\text{-}a0iim1\text{-}ir\}$

by (*simp add*: *PO-rhoare-defs R-a0iim1-ir-defs m1-defs, safe, auto*)

lemma *PO-m1-step3-refines-ir-a0ii-skip*:

$\{R\text{-}a0iim1\text{-}ir\}$

Id, (*m1-step3* *Rs A B Kab Na Ts*)

$\{> R\text{-}a0iim1\text{-}ir\}$

by (*simp add*: *PO-rhoare-defs R-a0iim1-ir-defs a0i-defs m1-defs, safe, auto*)

lemma *PO-m1-step4-refines-ir-a0ii-skip*:

$\{R\text{-}a0iim1\text{-}ir\}$

Id, (*m1-step4* *Ra A B Na Kab Ts Ta*)

$\{> R\text{-}a0iim1\text{-}ir\}$

by (*simp add*: *PO-rhoare-defs R-a0iim1-ir-defs m1-defs, safe, auto*)

lemma *PO-m1-step5-refines-ir-a0ii-running*:

$\{R\text{-}a0iim1\text{-}ir \cap UNIV \times m1\text{-}inv0\text{-}fin\}$

(*a0i-running* [*A, B*] (*Kab, Ts, Ta*)), (*m1-step5* *Rb A B Kab Ts Ta*)

$\{> R\text{-}a0iim1\text{-}ir\}$

by (*simp add*: *PO-rhoare-defs R-a0iim1-ir-defs a0i-defs m1-defs, safe, auto*)

lemma *PO-m1-step6-refines-ir-a0ii-commit*:

$\{R\text{-}a0iim1\text{-}ir \cap UNIV \times m1\text{-}inv0\text{-}fin\}$

(*a0n-commit* [*A, B*] (*Kab, Ts, Ta*)), (*m1-step6* *Ra A B Na Kab Ts Ta*)

$\{> R\text{-}a0iim1\text{-}ir\}$

by (*simp add*: *PO-rhoare-defs R-a0iim1-ir-defs a0n-defs m1-defs, safe, auto*)

lemma *PO-m1-leak-refines-ir-a0ii-skip*:

$\{R\text{-}a0iim1\text{-}ir\}$

Id, (*m1-leak* *Rs A B Na Ts*)

$\{> R\text{-}a0iim1\text{-}ir\}$

by (*simp add*: *PO-rhoare-defs R-a0iim1-ir-defs a0n-defs m1-defs, safe, auto*)

lemma *PO-m1-tick-refines-ir-a0ii-skip*:

$\{R\text{-}a0iim1\text{-}ir\}$

Id, (*m1-tick* *T*)

$\{> R\text{-}a0iim1\text{-}ir\}$

by (*simp add*: *PO-rhoare-defs R-a0iim1-ir-defs m1-defs, safe, auto*)

lemma *PO-m1-purge-refines-ir-a0ii-skip*:

$\{R\text{-}a0iim1\text{-}ir\}$

Id, (*m1-purge* *A*)

$\{> R\text{-}a0iim1\text{-}ir\}$

by (*simp add*: *PO-rhoare-defs R-a0iim1-ir-defs m1-defs, safe, auto*)

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]:
 $init\ m1 \subseteq R\text{-}a0iim1\text{-}ir\text{''}(init\ a0n)$
by (*auto simp add: R-a0iim1-ir-defs a0n-defs m1-defs*
intro!: exI [where x=(signals = $\lambda s. 0$, corrupted = {})])

lemma *PO-m1-refines-trans-ir-a0ii* [iff]:
 $\{R\text{-}a0iim1\text{-}ir \cap UNIV \times m1\text{-}inv0\text{-}fin\}$
 $(trans\ a0n), (trans\ m1)$
 $\{> R\text{-}a0iim1\text{-}ir\}$
by (*auto simp add: m1-def m1-trans-def a0n-def a0n-trans-def*
intro!: PO-m1-trans-refines-ir-a0ii-trans)

Observation consistency.

lemma *obs-consistent-med-a0iim1-ir* [iff]:
obs-consistent
 $(R\text{-}a0iim1\text{-}ir \cap UNIV \times m1\text{-}inv0\text{-}fin)$
 $med\text{-}a0iim1\text{-}ir\ a0n\ m1$
by (*auto simp add: obs-consistent-def R-a0iim1-ir-def med-a0iim1-ir-def*
a0n-def m1-def)

Refinement result.

lemma *PO-m1-refines-a0ii-ir* [iff]:
 $refines\ (R\text{-}a0iim1\text{-}ir \cap UNIV \times m1\text{-}inv0\text{-}fin)$
 $med\text{-}a0iim1\text{-}ir\ a0n\ m1$
by (*rule Refinement-using-invariants*) (*auto*)

lemma *m1-implements-a0ii-ir: implements med-a0iim1-ir a0n m1*
by (*rule refinement-soundness*) (*fast*)

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 $m2\text{-state} = m1\text{-state} +$
 $chan :: chmsg\ set$ — channel messages

type-synonym
 $m2\text{-obs} = m1\text{-state}$

definition
 $m2\text{-obs} :: m2\text{-state} \Rightarrow m2\text{-obs}$ **where**
 $m2\text{-obs } s \equiv ()$
 $runs = runs\ s,$
 $leak = leak\ s,$
 $clk = clk\ s,$
 $cache = cache\ s$
 $\})$

type-synonym
 $m2\text{-pred} = m2\text{-state } set$

type-synonym
 $m2\text{-trans} = (m2\text{-state} \times m2\text{-state})\ set$

3.5.2 Events

Protocol events.

definition — by A , refines $m1a\text{-step1}$
 $m2\text{-step1} :: [rid\text{-}t, agent, agent, nonce] \Rightarrow m2\text{-trans}$
where
 $m2\text{-step1 } Ra\ A\ B\ Na \equiv \{(s, s1).$
— guards:
 $Ra \notin dom\ (runs\ s) \wedge$ — Ra is fresh
 $Na = Ra\$na \wedge$ — generate nonce
— actions:
— create initiator thread and send message 1
 $s1 = s()$
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [])),$
 $chan := insert\ (Insec\ A\ B\ (Msg\ [aNon\ Na]))\ (chan\ s)$ — send $M1$
 $\})$
 $\}$

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:
 $Rs \notin \text{dom } (\text{runs } s) \wedge$ — fresh server run
 $Kab = \text{sesK } (Rs\$sk) \wedge$ — fresh session key
 $Ts = \text{clk } s \wedge$ — fresh timestamp

$\text{Insec } A B (\text{Msg } [aNon Na]) \in \text{chan } s \wedge$ — recv $M1$

— actions:
— record key and send messages 2 and 3
 $s1 = s[$
 $\text{runs} := (\text{runs } s)(Rs \mapsto (\text{Serv}, [A, B], [aNon Na, aNum Ts])),$
 $\text{chan} := \{\text{Secure Sv } A (\text{Msg } [aKey Kab, aAgt B, aNum Ts, aNon Na]),$ — send $M2a/b$
 $\text{Secure Sv } B (\text{Msg } [aKey Kab, aAgt A, aNum Ts])\} \cup \text{chan } s$
 $]$
 $\}$

definition — by A , refines $m1e\text{-step4}$

$m2\text{-step4} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}, \text{time}, \text{time}] \Rightarrow m2\text{-trans}$

where

$m2\text{-step4 } Ra A B Na Kab Ts Ta \equiv \{(s, s1)\}.$

— guards:
 $\text{runs } s Ra = \text{Some } (\text{Init}, [A, B], []) \wedge$ — session key not yet recv'd
 $Na = Ra\$na \wedge$ — fix nonce
 $Ta = \text{clk } s \wedge$ — fresh timestamp
 $\text{clk } s < Ts + Ls \wedge$ — ensure key recentness

$\text{Secure Sv } A (\text{Msg } [aKey Kab, aAgt B, aNum Ts, aNon Na]) \in \text{chan } s \wedge$ — recv $M2a$

— actions:
— record session key
 $s1 = s[$
 $\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [aKey Kab, aNum Ts, aNum Ta])),$
 $\text{chan} := \text{insert } (dAuth Kab (\text{Msg } [aAgt A, aNum Ta])) (\text{chan } s)$ — send $M3$
 $]$
 $\}$

definition — by B , refines $m1e\text{-step5}$

$m2\text{-step5} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}, \text{time}, \text{time}] \Rightarrow m2\text{-trans}$

where

$m2\text{-step5 } Rb A B Kab Ts Ta \equiv \{(s, s1)\}.$

— guards:
 $\text{runs } s Rb = \text{Some } (\text{Resp}, [A, B], []) \wedge$ — Kab not yet received
 $\text{Secure Sv } B (\text{Msg } [aKey Kab, aAgt A, aNum Ts]) \in \text{chan } s \wedge$ — recv $M2b$
 $dAuth Kab (\text{Msg } [aAgt A, aNum Ta]) \in \text{chan } s \wedge$ — recv $M3$

— ensure freshness of session key
 $\text{clk } s < Ts + Ls \wedge$

— check authenticator's validity and replay; 'replays' with fresh authenticator ok!

$\text{clk } s < Ta + La \wedge$
 $(B, Kab, Ta) \notin \text{cache } s \wedge$

— actions:
— record session key, send message M_4
 $s1 = s \{$
 $runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])),$
 $cache := insert\ (B, Kab, Ta)\ (cache\ s),$
 $chan := insert\ (dAuth\ Kab\ (Msg\ [aNum\ Ta]))\ (chan\ s) \quad \text{— send } M_4$
 $\}$
 $\}$

definition — by A , refines $m1e\text{-}step6$
 $m2\text{-}step6 :: [rid\text{-}t, agent, agent, nonce, key, time, time] \Rightarrow m2\text{-}trans$

where

$m2\text{-}step6\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta \equiv \{(s, s')\}.$
 $runs\ s\ Ra = Some\ (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta]) \wedge \quad \text{— key rcv'd before}$
 $Na = Ra\$na \wedge \quad \text{— generated nonce}$
 $clk\ s < Ts + Ls \wedge \quad \text{— check session key's recentness}$
 $dAuth\ Kab\ (Msg\ [aNum\ Ta]) \in chan\ s \wedge \quad \text{— rcv } M_4$

— actions:
 $s' = s \{$
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta, END]))$
 $\}$
 $\}$

Clock tick event

definition — refines $m1\text{-}tick$
 $m2\text{-}tick :: time \Rightarrow m2\text{-}trans$

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 $m1\text{-}leak$
 $m2\text{-}leak :: [rid\text{-}t, agent, agent, nonce, time] \Rightarrow m2\text{-}trans$

where

$m2\text{-}leak\ Rs\ A\ B\ Na\ Ts \equiv \{(s, s1)\}.$
— guards:
 $runs\ s\ Rs = Some\ (Serv, [A, B], [aNon\ Na, aNum\ Ts]) \wedge$
 $(clk\ s \geq Ts + Ls) \wedge \quad \text{— only compromise 'old' session keys}$

— 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-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-tick-def m2-purge-def m2-leak-def m2-fake-def

lemmas *m2-defs = m2-loc-defs m1-defs*

3.5.4 Invariants and simulation relation

inv1: Key definedness

All session keys in channel messages stem from existing runs.

definition

m2-inv1-keys :: m2-state set

where

m2-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-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-inv1-keysD = m2-inv1-keys-def [THEN setc-def-to-dest, rule-format, rotated 1]*

Invariance proof.

lemma *PO-m2-inv1-keys-init [iff]:*

init m2 \subseteq *m2-inv1-keys*

by (*auto simp add: m2-defs intro!: m2-inv1-keysI*)

lemma *PO-m2-inv1-keys-trans [iff]:*

$\{m2-inv1-keys\}$ *trans* *m2* $\{> m2-inv1-keys\}$

apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv1-keysI*)

apply (*auto simp add: dest: m2-inv1-keysD dom-lemmas*)

done

lemma *PO-m2-inv1-keys [iff]: reach m2* \subseteq *m2-inv1-keys*

by (*rule inv-rule-basic*) (*auto*)

inv2: Definedness of used keys

definition

m2-inv2-keys-for :: m2-state set

where

m2-inv2-keys-for $\equiv \{s. \forall R.$

$sesK (R\$sk) \in keys-for (chan s) \longrightarrow R \in dom (runs s)$

$\}$

lemmas *m2-inv2-keys-forI = m2-inv2-keys-for-def [THEN setc-def-to-intro, rule-format]*

lemmas *m2-inv2-keys-forE [elim] = m2-inv2-keys-for-def [THEN setc-def-to-elim, rule-format]*

lemmas *m2-inv2-keys-forD = m2-inv2-keys-for-def [THEN setc-def-to-dest, rule-format, rotated 1]*

Invariance proof.

lemma *PO-m2-inv2-keys-for-init [iff]:*

init $m2 \subseteq m2\text{-inv2-keys-for}$
by (*auto simp add: m2-defs intro!: m2-inv2-keys-forI*)

lemma *PO-m2-inv2-keys-for-trans* [*iff*]:
 $\{m2\text{-inv2-keys-for} \cap m2\text{-inv1-keys}\} \text{ trans } m2 \{> m2\text{-inv2-keys-for}\}$
apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv2-keys-forI*)
apply (*auto dest: m2-inv2-keys-forD m2-inv1-keysD dest: dom-lemmas*)
— 2 subgoals, from step3 and fake
apply (*rename-tac R s xb xc xd xi,*
subgoal-tac aKey (sesK (R\$sk)) \in atoms (chan s), auto)
apply (*auto simp add: keys-for-def, erule fake.cases, fastforce+*)
done

lemma *PO-m2-inv2-keys-for* [*iff*]: *reach* $m2 \subseteq m2\text{-inv2-keys-for}$
by (*rule inv-rule-incr*) (*auto del: subsetI*)

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 $\equiv \{s. \forall K KK.$

~~$KK \in \text{extr} (aKey \cdot KK \cup ik0) (chan s) \longleftrightarrow (K \in KK \vee aKey K \in \text{extr} ik0 (chan s))$~~

$aKey K \in \text{extr} (aKey \cdot KK \cup ik0) (chan s) \longleftrightarrow (K \in KK \vee aKey K \in \text{extr} ik0 (chan s))$

$\}$

lemmas *m2-inv3a-sesK-comprI* = *m2-inv3a-sesK-compr-def* [*THEN setc-def-to-intro, rule-format*]

lemmas *m2-inv3a-sesK-comprE* [*elim*] = *m2-inv3a-sesK-compr-def* [*THEN setc-def-to-elim, rule-format*]

lemmas *m2-inv3a-sesK-comprD* = *m2-inv3a-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-inv3a-sesK-compr-simps* =

m2-inv3a-sesK-comprD

m2-inv3a-sesK-comprD [**where** $KK = \text{insert } Kab \text{ } KK$ **for** $Kab \text{ } KK$, *simplified*]

m2-inv3a-sesK-comprD [**where** $KK = \{Kab\}$ **for** Kab , *simplified*]

insert-commute-aKey

lemma *PO-m2-inv3a-sesK-compr-init* [*iff*]:

init $m2 \subseteq m2\text{-inv3a-sesK-compr}$

by (*auto simp add: m2-defs intro!: m2-inv3a-sesK-comprI*)

lemma *PO-m2-inv3a-sesK-compr-trans* [*iff*]:

$\{m2\text{-inv3a-sesK-compr}\} \text{ trans } m2 \{> m2\text{-inv3a-sesK-compr}\}$

by (*auto simp add: PO-hoare-defs m2-defs m2-inv3a-sesK-compr-simps intro!: m2-inv3a-sesK-comprI*)

lemma *PO-m2-inv3a-sesK-compr* [*iff*]: *reach* $m2 \subseteq m2\text{-inv3a-sesK-compr}$

by (*rule inv-rule-basic*) (*auto*)

inv3b: Leakage of old session keys

Only old session keys are leaked to the intruder.

definition

$m2\text{-inv3b}\text{-leak} :: m2\text{-state set}$

where

$$m2\text{-inv3b}\text{-leak} \equiv \{s. \forall R s A B Na Ts. \\ (sesK (Rs\$sk), A, B, Na, Ts) \in leak\ s \longrightarrow clk\ s \geq Ts + Ls\}$$

lemmas $m2\text{-inv3b}\text{-leak}I = m2\text{-inv3b}\text{-leak}\text{-def} [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m2\text{-inv3b}\text{-leak}E [elim] = m2\text{-inv3b}\text{-leak}\text{-def} [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m2\text{-inv3b}\text{-leak}D = m2\text{-inv3b}\text{-leak}\text{-def} [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

Invariance proof.

lemma $PO\text{-}m2\text{-inv3b}\text{-leak}\text{-init} [iff]:$

$init\ m2 \subseteq m2\text{-inv3b}\text{-leak}$

by ($auto\ simp\ add: m2\text{-defs}\ intro!: m2\text{-inv3b}\text{-leak}I$)

lemma $PO\text{-}m2\text{-inv3b}\text{-leak}\text{-trans} [iff]:$

$\{m2\text{-inv3b}\text{-leak} \cap m2\text{-inv1}\text{-keys}\} trans\ m2 \{> m2\text{-inv3b}\text{-leak}\}$

by ($fastforce\ simp\ add: PO\text{-}hoare\text{-defs}\ m2\text{-defs}\ intro!: m2\text{-inv3b}\text{-leak}I\ dest: m2\text{-inv3b}\text{-leak}D$)

lemma $PO\text{-}m2\text{-inv3b}\text{-leak} [iff]: reach\ m2 \subseteq m2\text{-inv3b}\text{-leak}$

by ($rule\ inv\text{-rule}\text{-incr} (auto\ del: subsetI)$)

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., $Kab \notin Domain (leaks\ s) \longrightarrow (Kab, A) \in azC (runs\ s)$ for the initiator's step4).

definition

$m2\text{-inv3}\text{-extrKey} :: m2\text{-state set}$

where

$$m2\text{-inv3}\text{-extrKey} \equiv \{s. \forall K. \\ aKey\ K \in extr\ ik0 (chan\ s) \longrightarrow \\ (K \in corrKey \wedge K \in Domain (leak\ s)) \vee \\ (\exists R A' B' Na' Ts'. K = sesK (R\$sk) \wedge \\ runs\ s\ R = Some (Serv, [A', B'], [aNon\ Na', aNum\ Ts']) \wedge \\ (A' \in bad \vee B' \in bad \vee (K, A', B', Na', Ts') \in leak\ s))\}$$

lemmas $m2\text{-inv3}\text{-extrKey}I = m2\text{-inv3}\text{-extrKey}\text{-def} [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m2\text{-inv3}\text{-extrKey}E [elim] = m2\text{-inv3}\text{-extrKey}\text{-def} [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m2\text{-inv3}\text{-extrKey}D = m2\text{-inv3}\text{-extrKey}\text{-def} [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

lemma $PO\text{-}m2\text{-inv3}\text{-extrKey}\text{-init} [iff]:$

$init\ m2 \subseteq m2\text{-inv3}\text{-extrKey}$

by ($auto\ simp\ add: m2\text{-defs}\ intro!: m2\text{-inv3}\text{-extrKey}I$)

lemma $PO\text{-}m2\text{-inv3}\text{-extrKey}\text{-trans} [iff]:$

```

{m2-inv3-extrKey  $\cap$  m2-inv3a-sesK-compr}
  trans m2
  {> m2-inv3-extrKey}
apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv3-extrKeyI)
apply (auto simp add: m2-inv3a-sesK-compr-simps dest!: m2-inv3-extrKeyD dest: dom-lemmas)
done

```

lemma *PO-m2-inv3-extrKey [iff]: reach m2 \subseteq m2-inv3-extrKey*
by (rule-tac $J=m2-inv3a-sesK-compr$ **in** inv-rule-incr) (auto)

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-inv4-M2a :: m2-state set

where

```

m2-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-inv4-M2b :: m2-state set

where

```

m2-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-inv4-M2aI = m2-inv4-M2a-def [THEN setc-def-to-intro, rule-format]*
lemmas *m2-inv4-M2aE [elim] = m2-inv4-M2a-def [THEN setc-def-to-elim, rule-format]*
lemmas *m2-inv4-M2aD = m2-inv4-M2a-def [THEN setc-def-to-dest, rule-format, rotated 1]*

lemmas *m2-inv4-M2bI = m2-inv4-M2b-def [THEN setc-def-to-intro, rule-format]*
lemmas *m2-inv4-M2bE [elim] = m2-inv4-M2b-def [THEN setc-def-to-elim, rule-format]*
lemmas *m2-inv4-M2bD = m2-inv4-M2b-def [THEN setc-def-to-dest, rule-format, rotated 1]*

Invariance proofs.

lemma *PO-m2-inv4-M2a-init [iff]:*
init m2 \subseteq m2-inv4-M2a
by (auto simp add: m2-defs intro!: m2-inv4-M2aI)

lemma *PO-m2-inv4-M2a-trans [iff]:*
{m2-inv4-M2a} trans m2 {> m2-inv4-M2a}
apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv4-M2aI)
apply (auto dest!: m2-inv4-M2aD dest: dom-lemmas)
— 4 subgoals
apply (force dest!: spec)

```

apply (force dest!: spec)
apply (force dest!: spec)
apply (rule exI, auto)
done

```

```

lemma PO-m2-inv4-M2a [iff]: reach m2  $\subseteq$  m2-inv4-M2a
by (rule inv-rule-basic) (auto)

```

```

lemma PO-m2-inv4-M2b-init [iff]:
  init m2  $\subseteq$  m2-inv4-M2b
by (auto simp add: m2-defs intro!: m2-inv4-M2bI)

```

```

lemma PO-m2-inv4-M2b-trans [iff]:
  {m2-inv4-M2b} trans m2  $\{>$  m2-inv4-M2b}
apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv4-M2bI)
apply (auto dest!: m2-inv4-M2bD dest: dom-lemmas)
— 4 subgoals
apply (force dest!: spec)
apply (force dest!: spec)
apply (force dest!: spec)
apply (rule exI, auto)
done

```

```

lemma PO-m2-inv4-M2b [iff]: reach m2  $\subseteq$  m2-inv4-M2b
by (rule inv-rule-incr) (auto del: subsetI)

```

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'
apply (subgoal-tac A'  $\notin$  bad  $\wedge$  B  $\notin$  bad, auto)
apply (auto dest!: m2-inv4-M2aD m2-inv4-M2bD)
done

```

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 (cases A  $\in$  bad)
  case True
  hence aKey K  $\in$  extr ik0 (chan s) using assms(1) by auto
  thus ?thesis using assms (3-) by auto
next
  case False
  thus ?thesis using assms(1-2) by (auto dest: m2-inv4-M2aD)

```

qed

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*)

using *assms*

proof (*cases B* \in *bad*)

case *True*

from *assms*(1) $\langle B \in bad \rangle$ **have** *aKey K* \in *extr ik0* (*chan s*) **by** *auto*

thus *?thesis* **using** *assms* (3–) **by** *auto*

next

case *False*

thus *?thesis* **using** *assms* (1–2) **by** (*auto dest: m2-inv4-M2bD*)

qed

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

aKey (*sesK* (*R\$sk*)) \in *extr ik0* (*chan s*) \longrightarrow

(*sesK* (*R\$sk*), *A*, *B*, *Na*, *Ts*) \in *leak s*

}

lemmas *m2-inv5-ikk-svI* = *m2-inv5-ikk-sv-def* [*THEN setc-def-to-intro*, *rule-format*]

lemmas *m2-inv5-ikk-svE* [*elim*] = *m2-inv5-ikk-sv-def* [*THEN setc-def-to-elim*, *rule-format*]

lemmas *m2-inv5-ikk-svD* = *m2-inv5-ikk-sv-def* [*THEN setc-def-to-dest*, *rule-format*, *rotated 1*]

Invariance proof. This invariant follows from *m2-inv3-extrKey*.

lemma *m2-inv5-ikk-sv-derived*:

s \in *m2-inv3-extrKey* \implies *s* \in *m2-inv5-ikk-sv*

by (*auto simp add: m2-inv3-extrKey-def m2-inv5-ikk-sv-def*)

lemma *PO-m2-inv5-ikk-sv* [*iff*]: *reach m2* \subseteq *m2-inv5-ikk-sv*

proof –

have *reach m2* \subseteq *m2-inv3-extrKey* **by** *blast*

also have ... \subseteq *m2-inv5-ikk-sv* **by** (*blast intro: m2-inv5-ikk-sv-derived*)

finally show *?thesis* .

qed

inv6 (derived): Key secrecy for initiator

This invariant is derivable (see below).

definition

m2-inv6-ikk-init :: *m2-state set*

where

$$\begin{aligned}
& m2\text{-inv6}\text{-ikk}\text{-init} \equiv \{s. \forall A B Ra K Ts nl. \\
& \quad runs\ s\ Ra = Some\ (Init, [A, B], aKey\ K\ \# aNum\ Ts\ \# nl) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow \\
& \quad aKey\ K \in extr\ ik0\ (chan\ s) \longrightarrow \\
& \quad (K, A, B, Ra\$na, Ts) \in leak\ s \\
& \}
\end{aligned}$$

lemmas $m2\text{-inv6}\text{-ikk}\text{-init}I = m2\text{-inv6}\text{-ikk}\text{-init}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m2\text{-inv6}\text{-ikk}\text{-init}E\ [elim] = m2\text{-inv6}\text{-ikk}\text{-init}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m2\text{-inv6}\text{-ikk}\text{-init}D = m2\text{-inv6}\text{-ikk}\text{-init}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

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

$$\begin{aligned}
& m2\text{-inv7}\text{-ikk}\text{-resp} \equiv \{s. \forall A B Rb K Ts nl. \\
& \quad runs\ s\ Rb = Some\ (Resp, [A, B], aKey\ K\ \# aNum\ Ts\ \# nl) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow \\
& \quad aKey\ K \in extr\ ik0\ (chan\ s) \longrightarrow \\
& \quad (\exists Na. (K, A, B, Na, Ts) \in leak\ s) \\
& \}
\end{aligned}$$

lemmas $m2\text{-inv7}\text{-ikk}\text{-resp}I = m2\text{-inv7}\text{-ikk}\text{-resp}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m2\text{-inv7}\text{-ikk}\text{-resp}E\ [elim] = m2\text{-inv7}\text{-ikk}\text{-resp}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m2\text{-inv7}\text{-ikk}\text{-resp}D = m2\text{-inv7}\text{-ikk}\text{-resp}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

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

$m2\text{-inv8}\text{-M4} :: m2\text{-state}\ set$

where

$$\begin{aligned}
& m2\text{-inv8}\text{-M4} \equiv \{s. \forall Kab A B Ts Ta N. \\
& \quad Secure\ Sv\ A\ (Msg\ [aKey\ Kab, aAgt\ B, aNum\ Ts, aNon\ N]) \in chan\ s \longrightarrow \\
& \quad dAuth\ Kab\ (Msg\ [aNum\ Ta]) \in chan\ s \longrightarrow \\
& \quad aKey\ Kab \notin extr\ ik0\ (chan\ s) \longrightarrow \\
& \quad (\exists Rb. runs\ s\ Rb = Some\ (Resp, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])) \\
& \}
\end{aligned}$$

lemmas $m2\text{-inv8}\text{-M4}I = m2\text{-inv8}\text{-M4}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m2\text{-inv8}\text{-M4}E\ [elim] = m2\text{-inv8}\text{-M4}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m2\text{-inv8}\text{-M4}D = m2\text{-inv8}\text{-M4}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

Invariance proof.

lemma $PO\text{-}m2\text{-inv8}\text{-M4}\text{-init}\ [iff]:$

$init\ m2 \subseteq m2\text{-inv8}\text{-M4}$

by (*auto simp add: m2-defs intro!: m2-inv8-M4I*)

```

lemma PO-m2-inv8-M4-trans [iff]:
  {m2-inv8-M4  $\cap$  m2-inv4-M2a  $\cap$  m2-inv4-M2b  $\cap$  m2-inv3a-sesK-compr  $\cap$  m2-inv2-keys-for}
  trans m2
  {> m2-inv8-M4}
proof –
  {
    fix Rs A B Kab Na Ts
    have
      {m2-inv8-M4  $\cap$  m2-inv3a-sesK-compr  $\cap$  m2-inv2-keys-for}
      m2-step3 Rs A B Kab Na Ts
      {> m2-inv8-M4}
      apply (simp add: PO-hoare-defs m2-defs, safe intro!: m2-inv8-M4I)
      apply (auto simp add: m2-inv3a-sesK-compr-simps dest!: m2-inv8-M4D dest: dom-lemmas)
      done
  } moreover {
    fix Ra A B Na Kab Ts Ta
    have {m2-inv8-M4} m2-step4 Ra A B Na Kab Ts Ta {> m2-inv8-M4}
    apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv8-M4I)
    – 1 subgoal
    apply (drule m2-inv8-M4D, auto)
    apply (rename-tac Rb, rule-tac x=Rb in exI, auto)
    done
  } moreover {
    fix Rb A B Kab Ts Ta
    have {m2-inv8-M4  $\cap$  m2-inv4-M2a  $\cap$  m2-inv4-M2b} m2-step5 Rb A B Kab Ts Ta {> m2-inv8-M4}

    apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv8-M4I)
    – 2 subgoals
    apply (drule m2-inv4-M2a-M2b-match, auto)

    apply (auto dest!: m2-inv8-M4D)
    apply (rename-tac Rba, rule-tac x=Rba in exI, auto)
    done
  } moreover {
    fix Ra A B Na Kab Ts Ta
    have {m2-inv8-M4} m2-step6 Ra A B Na Kab Ts Ta {> m2-inv8-M4}
    apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv8-M4I)
    apply (auto dest!: m2-inv8-M4D)
    – 1 subgoal
    apply (rename-tac Rb, rule-tac x=Rb in exI, auto)
    done
  } moreover {
    have {m2-inv8-M4} m2-fake {> m2-inv8-M4}
    apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv8-M4I)
    – 1 subgoal
    apply (erule fake.cases, auto dest!: m2-inv8-M4D)
    done
  }
  }
ultimately show ?thesis
  apply (auto simp add: m2-def m2-trans-def dest!: spec)
  apply (simp-all (no-asm) add: PO-hoare-defs m2-defs, safe intro!: m2-inv8-M4I)
  apply (auto simp add: m2-inv3a-sesK-compr-simps dest!: m2-inv8-M4D dest: dom-lemmas)
  done

```

qed

lemma *PO-m2-inv8-M4* [iff]: $reach\ m2 \subseteq m2\text{-inv8-M4}$

by (rule-tac $J=m2\text{-inv4-M2a} \cap m2\text{-inv4-M2b} \cap m2\text{-inv3a-sesK-compr} \cap m2\text{-inv2-keys-for}$
in *inv-rule-incr*) (auto)

inv9a: Relating the initiator state to M2a

definition

m2-inv9a-init-M2a :: *m2-state set*

where

$m2\text{-inv9a-init-M2a} \equiv \{s. \forall A\ B\ Ra\ Kab\ Ts\ z.$

$runs\ s\ Ra = Some\ (Init, [A, B], aKey\ Kab\ \# \ aNum\ Ts\ \# \ z) \longrightarrow$

$Secure\ Sv\ A\ (Msg\ [aKey\ Kab, aAgt\ B, aNum\ Ts, aNon\ (Ra\$na)]) \in chan\ s$

$\}$

lemmas *m2-inv9a-init-M2aI* = *m2-inv9a-init-M2a-def* [THEN *setc-def-to-intro*, *rule-format*]

lemmas *m2-inv9a-init-M2aE* [elim] = *m2-inv9a-init-M2a-def* [THEN *setc-def-to-elim*, *rule-format*]

lemmas *m2-inv9a-init-M2aD* = *m2-inv9a-init-M2a-def* [THEN *setc-def-to-dest*, *rule-format*, *rotated* 1]

Invariance proof.

lemma *PO-m2-inv9a-init-M2a-init* [iff]:

$init\ m2 \subseteq m2\text{-inv9a-init-M2a}$

by (auto simp add: *m2-defs intro!*: *m2-inv9a-init-M2aI*)

lemma *PO-m2-inv9a-init-M2a-trans* [iff]:

$\{m2\text{-inv9a-init-M2a}\}\ trans\ m2\ \{>\ m2\text{-inv9a-init-M2a}\}$

by (fastforce simp add: *PO-hoare-defs m2-defs intro!*: *m2-inv9a-init-M2aI*
dest: *m2-inv9a-init-M2aD*)

lemma *PO-m2-inv9a-init-M2a* [iff]: $reach\ m2 \subseteq m2\text{-inv9a-init-M2a}$

by (rule *inv-rule-incr*) (auto del: *subsetI*)

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-inv9-M3 :: *m2-state set*

where

$m2\text{-inv9-M3} \equiv \{s. \forall Kab\ A\ B\ Ts\ Ta.$

$Secure\ Sv\ B\ (Msg\ [aKey\ Kab, aAgt\ A, aNum\ Ts]) \in chan\ s \longrightarrow$

$dAuth\ Kab\ (Msg\ [aAgt\ A, aNum\ Ta]) \in chan\ s \longrightarrow$

$aKey\ Kab \notin extr\ ik0\ (chan\ s) \longrightarrow$

$(\exists Ra\ nl. runs\ s\ Ra = Some\ (Init, [A, B], aKey\ Kab\ \# \ aNum\ Ts\ \# \ aNum\ Ta\ \# \ nl))$

$\}$

lemmas *m2-inv9-M3I* = *m2-inv9-M3-def* [THEN *setc-def-to-intro*, *rule-format*]

lemmas *m2-inv9-M3E* [elim] = *m2-inv9-M3-def* [THEN *setc-def-to-elim*, *rule-format*]

lemmas *m2-inv9-M3D* = *m2-inv9-M3-def* [THEN *setc-def-to-dest*, *rule-format*, *rotated* 1]

Invariance proof.

```

lemma PO-m2-inv9-M3-init [iff]:
  init m2  $\subseteq$  m2-inv9-M3
by (auto simp add: m2-defs intro!: m2-inv9-M3I)

lemma PO-m2-inv9-M3-trans [iff]:
  {m2-inv9-M3  $\cap$  m2-inv4-M2a  $\cap$  m2-inv4-M2b  $\cap$  m2-inv3a-sesK-compr  $\cap$  m2-inv2-keys-for}
  trans m2
  {> m2-inv9-M3}
proof –
{
  fix Rs A B Kab Na Ts
  have
    {m2-inv9-M3  $\cap$  m2-inv3a-sesK-compr  $\cap$  m2-inv2-keys-for}
    m2-step3 Rs A B Kab Na Ts
    {> m2-inv9-M3}
    apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M3I)
    apply (auto simp add: m2-inv3a-sesK-compr-simps dest!: m2-inv9-M3D dest: dom-lemmas)
    done
} moreover {
  fix Ra A B Na Kab Ts Ta
  have {m2-inv9-M3  $\cap$  m2-inv4-M2a  $\cap$  m2-inv4-M2b} m2-step4 Ra A B Na Kab Ts Ta {> m2-inv9-M3}
  apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M3I)
  apply (auto dest: m2-inv4-M2a-M2b-match)
  – 1 subgoal
  apply (frule m2-inv9-M3D, auto)
  apply (rule-tac x=Raa in exI, auto)
  done
} moreover {
  fix Rb A B Kab Ts Ta
  have {m2-inv9-M3} m2-step5 Rb A B Kab Ts Ta {> m2-inv9-M3}
  apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M3I)
  apply (auto dest!: m2-inv9-M3D dest: dom-lemmas)
  – 2 subgoals
  apply (auto dest!: spec intro!: exI) – witness Na in both cases
  done
} moreover {
  fix Ra A B Na Kab Ts Ta
  have {m2-inv9-M3} m2-step6 Ra A B Na Kab Ts Ta {> m2-inv9-M3}
  apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M3I)
  – 1 subgoals
  apply (auto dest!: m2-inv9-M3D dest: dom-lemmas)
  apply (rename-tac Raa nl, case-tac Raa = Ra, auto)
  done
} moreover {
  have {m2-inv9-M3} m2-fake {> m2-inv9-M3}
  apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M3I)
  – 1 subgoals
  apply (erule fake.cases, auto)+
  done
} ultimately
show ?thesis
  apply (auto simp add: m2-def m2-trans-def dest!: spec)
  apply (simp-all (no-asm) add: PO-hoare-defs m2-defs, safe intro!: m2-inv9-M3I)

```

apply (*auto simp add: m2-inv3a-sesK-compr-simps dest!: m2-inv9-M3D dest: dom-lemmas*)
done
qed

lemma *PO-m2-inv9-M3 [iff]: reach m2 \subseteq m2-inv9-M3*
by (*rule-tac J=m2-inv4-M2a \cap m2-inv4-M2b \cap m2-inv3a-sesK-compr \cap m2-inv2-keys-for*
in *inv-rule-incr*) (*auto*)

3.5.5 Refinement

The simulation relation. This is a pure superposition refinement.

definition

R12 :: (*m1-state* \times *m2-state*) set **where**
R12 \equiv $\{(s, t). \text{runs } s = \text{runs } t \wedge \text{leak } s = \text{leak } t \wedge \text{clk } s = \text{clk } t \wedge \text{cache } s = \text{cache } t\}$

The mediator function is the identity.

definition

med21 :: *m2-obs* \Rightarrow *m1-obs* **where**
med21 = *id*

Refinement proof.

lemma *PO-m2-step1-refines-m1-step1*:
 $\{R12\}$
(*m1-step1 Ra A B Na*), (*m2-step1 Ra A B Na*)
 $\{> R12\}$
by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

lemma *PO-m2-step2-refines-m1-step2*:
 $\{R12\}$
(*m1-step2 Rb A B*), (*m2-step2 Rb A B*)
 $\{> R12\}$
by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

lemma *PO-m2-step3-refines-m1-step3*:
 $\{R12\}$
(*m1-step3 Rs A B Kab Na Ts*), (*m2-step3 Rs A B Kab Na Ts*)
 $\{> R12\}$
by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

lemma *PO-m2-step4-refines-m1-step4*:
 $\{R12 \cap UNIV \times (m2-inv4-M2a \cap m2-inv3-extrKey)\}$
(*m1-step4 Ra A B Na Kab Ts Ta*), (*m2-step4 Ra A B Na Kab Ts Ta*)
 $\{> R12\}$

apply (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

apply (*auto dest: m2-inv34-M2a-authorized*)

done

lemma *PO-m2-step5-refines-m1-step5*:
 $\{R12 \cap UNIV$
 $\times (m2-inv9-M3 \cap m2-inv5-ikk-sv \cap m2-inv4-M2b \cap m2-inv3-extrKey \cap m2-inv3b-leak)\}$
(*m1-step5 Rb A B Kab Ts Ta*), (*m2-step5 Rb A B Kab Ts Ta*)
 $\{> R12\}$

apply (*simp add: PO-rhoare-defs R12-def m2-defs, safe, simp-all*)
apply (*auto dest: m2-inv34-M2b-authorized*)
— 1 subgoal
apply (*frule m2-inv4-M2bD, auto*)
apply (*auto dest: m2-inv9-M3D m2-inv5-ikk-svD [THEN m2-inv3b-leakD]*)
done

lemma *PO-m2-step6-refines-m1-step6:*

{*R12* \cap *UNIV*
 \times (*m2-inv9a-init-M2a* \cap *m2-inv8-M4* \cap *m2-inv5-ikk-sv* \cap *m2-inv4-M2a* \cap *m2-inv3b-leak*)}
(*m1-step6 Ra A B Na Kab Ts Ta*), (*m2-step6 Ra A B Na Kab Ts Ta*)
{> *R12*}

apply (*auto simp add: PO-rhoare-defs R12-def m2-defs*)

— 1 subgoal

apply (*frule m2-inv9a-init-M2aD [THEN m2-inv4-M2aD], auto*)

apply (*auto dest: m2-inv9a-init-M2aD [THEN m2-inv8-M4D] m2-inv5-ikk-svD [THEN m2-inv3b-leakD]*)
done

lemma *PO-m2-tick-refines-m1-tick:*

{*R12*}
(*m1-tick T*), (*m2-tick T*)
{> *R12*}

by (*auto simp add: PO-rhoare-defs R12-def m2-defs*)

lemma *PO-m2-purge-refines-m1-purge:*

{*R12*}
(*m1-purge A*), (*m2-purge A*)
{> *R12*}

by (*auto simp add: PO-rhoare-defs R12-def m2-defs*)

lemma *PO-m2-leak-refines-leak:*

{*R12*}
m1-leak Rs A B Na Ts, *m2-leak Rs A B Na Ts*
{> *R12*}

by (*auto simp add: PO-rhoare-defs R12-def m2-defs dest: dom-lemmas*)

lemma *PO-m2-fake-refines-skip:*

{*R12*}
Id, *m2-fake*
{> *R12*}

by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

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*)

by (*auto simp add: R12-def m1-defs m2-loc-defs*)

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 –
 — derive invariant *m2-inv5-ikk-sv* from *m2-inv3-extrKey*
let *?pre'* = *R12* \cap
UNIV \times (*m2-inv9-M3* \cap *m2-inv9a-init-M2a* \cap *m2-inv8-M4* \cap *m2-inv5-ikk-sv* \cap
m2-inv4-M2b \cap *m2-inv4-M2a* \cap *m2-inv3-extrKey* \cap *m2-inv3b-leak*)
show *?thesis* (**is** {*?pre*} *?t1*, *?t2* {> *?post*})
proof (*rule relhoare-conseq-left*)
show *?pre* \subseteq *?pre'*
by (*auto intro: m2-inv5-ikk-sv-derived*)
next
show {*?pre'*} *?t1*, *?t2* {> *?post*}
by (*auto simp add: m2-def m2-trans-def m1-def m1-trans-def*)
 (*blast intro!: PO-m2-trans-refines-m1-trans*)
qed
qed

lemma *PO-obs-consistent-R12* [iff]:
obs-consistent R12 med21 m1 m2
by (*auto simp add: obs-consistent-def R12-def med21-def m2-defs*)

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)
by (*rule Refinement-using-invariants*) (*auto*)

lemma *m2-implements-m1* [iff]:
implements med21 m1 m2
by (*rule refinement-soundness*) (*auto*)

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*
by (*rule-tac Pa=m1-inv2i-serv and Qa=m1-inv2i-serv and Q=m1-inv2i-serv*
in *m2-implements-m1* [*THEN* [5] *internal-invariant-translation*])
 (*auto simp add: m2-loc-defs med21-def intro!: m1-inv2i-servI*)

lemma *PO-m2-sat-m1-inv2r-serv* [iff]: $reach\ m2 \subseteq m1\text{-}inv2r\text{-}serv$
by (*rule-tac* $Pa=m1\text{-}inv2r\text{-}serv$ **and** $Qa=m1\text{-}inv2r\text{-}serv$ **and** $Q=m1\text{-}inv2r\text{-}serv$
in *m2-implements-m1* [THEN [5] *internal-invariant-translation*])
(*fastforce simp add: m2-loc-defs med21-def intro!: m1-inv2r-servI*)+

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]: $reach\ m2 \subseteq m2\text{-}inv6\text{-}ikk\text{-}init$

proof –

have $reach\ m2 \subseteq m1\text{-}inv2i\text{-}serv \cap m2\text{-}inv5\text{-}ikk\text{-}sv$ **by** *simp*

also have $\dots \subseteq m2\text{-}inv6\text{-}ikk\text{-}init$ **by** (*blast intro!: m2-inv6-ikk-initI dest: m2-inv5-ikk-svD*)

finally show *?thesis* .

qed

lemma *PO-m2-inv6-resp-ikk* [iff]: $reach\ m2 \subseteq m2\text{-}inv7\text{-}ikk\text{-}resp$

proof –

have $reach\ m2 \subseteq m1\text{-}inv2r\text{-}serv \cap m2\text{-}inv5\text{-}ikk\text{-}sv$ **by** *simp*

also have $\dots \subseteq m2\text{-}inv7\text{-}ikk\text{-}resp$ **by** (*blast intro!: m2-inv7-ikk-respI dest: m2-inv5-ikk-svD*)

finally show *?thesis* .

qed

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 *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*]: $corrKey = shrK\text{'}bad$

by (*auto simp add: keySetup-def ltkeySetup-def corrKey-def*)

3.6.2 State

The secure channels are star-shaped to/from the server. Therefore, we have only one agent in the relation.

record $m3\text{-state} = m1\text{-state} +$
 $IK :: \text{msg set}$ — intruder knowledge

Observable state: $runs$, $m1\text{-state}.leak$, clk , and $cache$.

type-synonym
 $m3\text{-obs} = m2\text{-obs}$

definition
 $m3\text{-obs} :: m3\text{-state} \Rightarrow m3\text{-obs}$ **where**
 $m3\text{-obs } s \equiv \langle runs = runs\ s, leak = leak\ s, clk = clk\ s, cache = cache\ s \rangle$

type-synonym
 $m3\text{-pred} = m3\text{-state set}$

type-synonym
 $m3\text{-trans} = (m3\text{-state} \times m3\text{-state})\ set$

3.6.3 Events

Protocol events.

definition — by A , refines $m2\text{-step1}$
 $m3\text{-step1} :: [rid\text{-}t, agent, agent, nonce] \Rightarrow m3\text{-trans}$
where
 $m3\text{-step1 } Ra\ A\ B\ Na \equiv \{(s, s1).\}$
— guards:
 $Ra \notin dom\ (runs\ s) \wedge$ — Ra is fresh
 $Na = Ra\$na \wedge$ — generated nonce
— actions:
 $s1 = s \langle$
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [])),$
 $IK := insert\ \{Agent\ A, Agent\ B, Nonce\ Na\}\ (IK\ s)$ — send $M1$
 \rangle
 $\}$

definition — by B , refines $m2\text{-step2}$
 $m3\text{-step2} :: [rid\text{-}t, agent, agent] \Rightarrow m3\text{-trans}$
where
 $m3\text{-step2} \equiv m1\text{-step2}$

definition — by $Server$, refines $m2\text{-step3}$
 $m3\text{-step3} :: [rid\text{-}t, agent, agent, key, nonce, time] \Rightarrow m3\text{-trans}$
where
 $m3\text{-step3 } Rs\ A\ B\ Kab\ Na\ Ts \equiv \{(s, s1).\}$
— guards:
 $Rs \notin dom\ (runs\ s) \wedge$ — fresh server run
 $Kab = sesK\ (Rs\$sk) \wedge$ — fresh session key

$\{Agent\ A, Agent\ B, Nonce\ Na\} \in IK\ s \wedge$ — recv *M1*
 $Ts = clk\ s \wedge$ — fresh timestamp
 — actions:
 — record session key and send *M2*
 $s1 = s(\{$
 $runs := (runs\ s)(Rs \mapsto (Serv, [A, B], [aNon\ Na, aNum\ Ts])),$
 $IK := insert\ (Crypt\ (shrK\ A)\ \{Key\ Kab, Agent\ B, Number\ Ts, Nonce\ Na\})$
 $(insert\ (Crypt\ (shrK\ B)\ \{Key\ Kab, Agent\ A, Number\ Ts\})\ (IK\ s))$
 $\})$
 $\}$

definition — by *A*, refines *m2-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*
 $s1 = s(\{$
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])),$
 $IK := insert\ (Crypt\ Kab\ \{Agent\ A, Number\ Ta\})\ (IK\ s)$ — *M3*
 $\})$
 $\}$

definition — by *B*, refines *m2-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!
 $clk\ s < Ta + La \wedge$

$(B, Kab, Ta) \notin \text{cache } s \wedge$
 — actions:
 — record session key
 $s1 = s \{$
 $\text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [aKey Kab, aNum Ts, aNum Ta])),$
 $\text{cache} := \text{insert } (B, Kab, Ta) (\text{cache } s),$
 $IK := \text{insert } (\text{Crypt } Kab (\text{Number } Ta)) (IK \ s) \quad \text{— send } M4$
 $\}$
 $\}$

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:

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], [aKey Kab, aNum Ts, aNum Ta]) \wedge$ — knows key

$Na = Ra\$na \wedge$ — generated nonce

$\text{clk } s < Ts + Ls \wedge$ — check session key's recentness

$\text{Crypt } Kab (\text{Number } Ta) \in IK \ s \wedge$ — recv $M4$

— actions:

$s' = s \{$

$\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [aKey Kab, aNum Ts, aNum Ta, \text{END}])))$

$\}$

$\}$

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:

$\text{runs } s \ Rs = \text{Some } (\text{Serv}, [A, B], [aNon Na, aNum Ts]) \wedge$

$(\text{clk } s \geq Ts + Ls) \wedge$ — 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 := 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 ≡ {(s, s1).

```

```

    — actions:

```

```

    s1 = s(| IK := synth (analz (IK s)) |) — take DY closure
  }

```

3.6.4 Transition system

definition

```

  m3-init :: m3-pred

```

where

```

  m3-init ≡ { (|
    runs = Map.empty,
    leak = shrK'bad × {undefined},
    clk = 0,
    cache = {},
    IK = Key'shrK'bad
  |) }

```

definition

```

  m3-trans :: m3-trans where

```

```

  m3-trans ≡ (| ∪ A B Ra Rb Rs Na Kab Ts Ta T.
    m3-step1 Ra A B Na ∪
    m3-step2 Rb A B ∪
    m3-step3 Rs A B Kab Na Ts ∪
    m3-step4 Ra A B Na Kab Ts Ta ∪
    m3-step5 Rb A B Kab Ts Ta ∪
    m3-step6 Ra A B Na Kab Ts Ta ∪
    m3-tick T ∪
    m3-purge A ∪
    m3-leak Rs A B Na Ts ∪
    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-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-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-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-lkeysec-dest = 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

by (*auto simp add: m3-defs intro!: m3-inv1-lkeysecI*)

lemma *PO-m3-inv1-lkeysec-trans [iff]:*

{m3-inv1-lkeysec} trans m3 {> m3-inv1-lkeysec}

by (*fastforce simp add: PO-hoare-defs m3-defs intro!: m3-inv1-lkeysecI*)

lemma *PO-m3-inv1-lkeysec [iff]: reach m3 ⊆ m3-inv1-lkeysec*

by (*rule inv-rule-basic*) (*fast+*)

Useful simplifier lemmas

lemma *m3-inv1-lkeysec-for-parts [simp]:*

[[s ∈ m3-inv1-lkeysec]] ⇒ Key (shrK C) ∈ parts (IK s) ↔ C ∈ bad

by *auto*

lemma *m3-inv1-lkeysec-for-analz [simp]:*

[[s ∈ m3-inv1-lkeysec]] ⇒ Key (shrK C) ∈ analz (IK s) ↔ C ∈ bad

by *auto*

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

$m3\text{-inv7a}\text{-sesK}\text{-compr} :: m3\text{-pred}$

where

$m3\text{-inv7a}\text{-sesK}\text{-compr} \equiv \{s. \forall K KK.$

$KK \subseteq \text{range sesK} \longrightarrow$

$(\text{Key } K \in \text{analz } (\text{Key } KK \cup (IK \ s))) = (K \in KK \vee \text{Key } K \in \text{analz } (IK \ s))$

$\}$

lemmas $m3\text{-inv7a}\text{-sesK}\text{-compr}I = m3\text{-inv7a}\text{-sesK}\text{-compr}\text{-def}$ [THEN setc-def-to-intro, rule-format]

lemmas $m3\text{-inv7a}\text{-sesK}\text{-compr}E = m3\text{-inv7a}\text{-sesK}\text{-compr}\text{-def}$ [THEN setc-def-to-elim, rule-format]

lemmas $m3\text{-inv7a}\text{-sesK}\text{-compr}D = m3\text{-inv7a}\text{-sesK}\text{-compr}\text{-def}$ [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 $m3\text{-inv7a}\text{-sesK}\text{-compr}\text{-simps} =$

$m3\text{-inv7a}\text{-sesK}\text{-compr}D$

$m3\text{-inv7a}\text{-sesK}\text{-compr}D$ [where $KK = \{Kab\}$ for Kab , simplified]

$m3\text{-inv7a}\text{-sesK}\text{-compr}D$ [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

$\text{am}\text{-M1}$:

$\{\text{Agent } A, \text{Agent } B, \text{Nonce } N\} \in H$

$\Longrightarrow \text{Insec } A \ B \ (\text{Msg } [\text{aNon } N]) \in \text{abs}\text{-msg } H$

| $\text{am}\text{-M2a}$:

$\text{Crypt } (\text{shrK } C) \ \{\text{Key } K, \text{Agent } B, \text{Number } T, \text{Nonce } N\} \in H$

$\Longrightarrow \text{Secure } Sv \ C \ (\text{Msg } [\text{aKey } K, \text{aAgt } B, \text{aNum } T, \text{aNon } N]) \in \text{abs}\text{-msg } H$

| $\text{am}\text{-M2b}$:

$\text{Crypt } (\text{shrK } C) \ \{\text{Key } K, \text{Agent } A, \text{Number } T\} \in H$

$\Longrightarrow \text{Secure } Sv \ C \ (\text{Msg } [\text{aKey } K, \text{aAgt } A, \text{aNum } T]) \in \text{abs}\text{-msg } H$

| $\text{am}\text{-M3}$:

$\text{Crypt } K \ \{\text{Agent } A, \text{Number } T\} \in H$

$\Longrightarrow \text{dAuth } K \ (\text{Msg } [\text{aAgt } A, \text{aNum } T]) \in \text{abs}\text{-msg } H$

| $\text{am}\text{-M4}$:

$\text{Crypt } K \ (\text{Number } T) \in H$

$\Longrightarrow \text{dAuth } K \ (\text{Msg } [\text{aNum } T]) \in \text{abs}\text{-msg } H$

R23: The simulation relation. This is a data refinement of the insecure and secure channels of refinement 2.

definition

$R23\text{-msgs} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$
 $R23\text{-msgs} \equiv \{(s, t). \text{abs-msg} (\text{parts} (IK\ t)) \subseteq \text{chan}\ s\}$

definition

$R23\text{-keys} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$
 $R23\text{-keys} \equiv \{(s, t). \forall KK\ K. KK \subseteq \text{range}\ \text{ses}K \longrightarrow$
 $\text{Key}\ K \in \text{analz}\ (\text{Key}'KK \cup (IK\ t)) \longleftrightarrow \text{aKey}\ K \in \text{extr}\ (\text{aKey}'KK \cup ik0)\ (\text{chan}\ s)$
 $\}$

definition

$R23\text{-non} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$
 $R23\text{-non} \equiv \{(s, t). \forall KK\ N. KK \subseteq \text{range}\ \text{ses}K \longrightarrow$
 $\text{Nonce}\ N \in \text{analz}\ (\text{Key}'KK \cup (IK\ t)) \longleftrightarrow \text{aNon}\ N \in \text{extr}\ (\text{aKey}'KK \cup ik0)\ (\text{chan}\ s)$
 $\}$

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

$med32 :: m3\text{-obs} \Rightarrow m2\text{-obs} \text{ where}$
 $med32 \equiv id$

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 $R23\text{-keysI} = R23\text{-keys-def}\ [THEN\ \text{rel-def-to-intro},\ \text{simplified},\ \text{rule-format}]$

lemmas $R23\text{-keysE}\ [elim] = R23\text{-keys-def}\ [THEN\ \text{rel-def-to-elim},\ \text{simplified},\ \text{rule-format}]$

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 $R23\text{-intros} = R23\text{-msgsI}\ R23\text{-keysI}\ R23\text{-nonI}\ R23\text{-presI}$

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** $KK = \text{insert } K' \text{ } KK$ **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*: $\text{abs-msg } \{\} = \{\}$

by (*auto*)

lemma *abs-msg-Un* [*simp*]:

$\text{abs-msg } (G \cup H) = \text{abs-msg } G \cup \text{abs-msg } H$

by (*auto*)

lemma *abs-msg-mono* [*elim*]:

$\llbracket m \in \text{abs-msg } G; G \subseteq H \rrbracket \implies m \in \text{abs-msg } H$

by (*auto*)

lemma *abs-msg-insert-mono* [*intro*]:

$\llbracket m \in \text{abs-msg } H \rrbracket \implies m \in \text{abs-msg } (\text{insert } m' \text{ } H)$

by (*auto*)

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*:

$\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; (s, t) \in R23\text{-non}; t \in m3\text{-inv1-lkeysec} \rrbracket$
 $\implies \text{abs-msg } (\text{synth } (\text{analz } (IK \ t))) \subseteq \text{fake ik0 } (\text{dom } (\text{runs } s)) (\text{chan } s)$

apply (*auto*)

— 9 subgoals, deal with replays first

prefer 2 **apply** (*blast*)

prefer 3 **apply** (*blast*)

prefer 4 **apply** (*blast*)

prefer 5 **apply** (*blast*)

— remaining 5 subgoals are real fakes

apply (*intro fake-StatCh fake-DynCh, auto simp add: R23-simps*)+

done

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}
```

by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros*) (*auto*)

lemma *PO-m3-step2-refines-m2-step2*:

```
{R23}
  (m2-step2 Rb A B), (m3-step2 Rb A B)
{> R23}
```

by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros*)

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 –

```
{ fix s t
```

```
  assume H:
```

```
    (s, t)  $\in$  R23-msgs (s, t)  $\in$  R23-keys (s, t)  $\in$  R23-non (s, t)  $\in$  R23-pres
    s  $\in$  m2-inv3a-sesK-compr
    t  $\in$  m3-inv7a-sesK-compr t  $\in$  m3-inv1-lkeysec
    Kab = sesK (Rs$sk) Rs  $\notin$  dom (runs t)
     $\{\!\! \{$  Agent A, Agent B, Nonce Na  $\}\!\! \} \in$  parts (IK t)
```

```
  let ?s' =
```

```
    s( $\ll$  runs := (runs s)(Rs  $\mapsto$  (Serv, [A, B], [aNon Na, aNum (clk t)])),
      chan := insert (Secure Sv A (Msg [aKey Kab, aAgt B, aNum (clk t), aNon Na]))
        (insert (Secure Sv B (Msg [aKey Kab, aAgt A, aNum (clk t)])) (chan s))  $\ll$ )
```

```
  let ?t' =
```

```
    t( $\ll$  runs := (runs t)(Rs  $\mapsto$  (Serv, [A, B], [aNon Na, aNum (clk t)])),
      IK := insert (Crypt (shrK A)  $\{\!\! \{$  Key Kab, Agent B, Number (clk t), Nonce Na  $\}\!\! \}$ )
        (insert (Crypt (shrK B)  $\{\!\! \{$  Key Kab, Agent A, Number (clk t)  $\}\!\! \}$ ) (IK t))  $\ll$ )
```

– here we go

```
  have (?s', ?t')  $\in$  R23-msgs using H
```

```
  by (–) (rule R23-intros, auto)
```

moreover

```
  have (?s', ?t')  $\in$  R23-keys using H
```

```
  by (–)
```

```
    (rule R23-intros,
```

```
      auto simp add: m2-inv3a-sesK-compr-simps m3-inv7a-sesK-compr-simps,
```

```
      auto simp add: R23-keys-simps)
```

moreover

```
  have (?s', ?t')  $\in$  R23-non using H
```

```
  by (–)
```

```
    (rule R23-intros,
```

$auto\ simp\ add:\ m2\ inv3a\ sesK\ compr\ simps\ m3\ inv7a\ sesK\ compr\ simps\ R23\ non\ simps)$
moreover
have $(?s', ?t') \in R23\ pres$ **using** H
by $(-)$ (rule $R23\ intros$, $auto$)
moreover
note $calculation$
}
thus $?thesis$
by $(auto\ simp\ add:\ PO\ rhoare\ defs\ R23\ def\ m3\ defs)$
qed

lemma $PO\ m3\ step4\ refines\ m2\ step4:$
 $\{R23 \cap UNIV \times (m3\ inv1\ lkeysec)\}$
 $(m2\ step4\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta), (m3\ step4\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta)$
 $\{> R23\}$
by $(auto\ simp\ add:\ PO\ rhoare\ defs\ R23\ def\ m3\ defs\ intro!:\ R23\ intros)$
 $(auto)$

lemma $PO\ m3\ step5\ refines\ m2\ step5:$
 $\{R23\}$
 $(m2\ step5\ Rb\ A\ B\ Kab\ Ts\ Ta), (m3\ step5\ Rb\ A\ B\ Kab\ Ts\ Ta)$
 $\{> R23\}$
by $(auto\ simp\ add:\ PO\ rhoare\ defs\ R23\ def\ m3\ defs\ intro!:\ R23\ intros)$
 $(auto)$

lemma $PO\ m3\ step6\ refines\ m2\ step6:$
 $\{R23\}$
 $(m2\ step6\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta), (m3\ step6\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta)$
 $\{> R23\}$
by $(auto\ simp\ add:\ PO\ rhoare\ defs\ R23\ def\ m3\ defs\ intro!:\ R23\ intros)$

lemma $PO\ m3\ tick\ refines\ m2\ tick:$
 $\{R23\}$
 $(m2\ tick\ T), (m3\ tick\ T)$
 $\{> R23\}$
by $(auto\ simp\ add:\ PO\ rhoare\ defs\ R23\ def\ m3\ defs\ intro!:\ R23\ intros)$

lemma $PO\ m3\ purge\ refines\ m2\ purge:$
 $\{R23\}$
 $(m2\ purge\ A), (m3\ purge\ A)$
 $\{> R23\}$
by $(auto\ simp\ add:\ PO\ rhoare\ defs\ R23\ def\ m3\ defs\ intro!:\ R23\ intros)$

Intruder events.

lemma $PO\ m3\ leak\ refines\ m2\ leak:$
 $\{R23\}$
 $(m2\ leak\ Rs\ A\ B\ Na\ Ts), (m3\ leak\ Rs\ A\ B\ Na\ Ts)$
 $\{> R23\}$
by $(auto\ simp\ add:\ PO\ rhoare\ defs\ R23\ def\ m3\ defs\ R23\ simps\ intro!:\ R23\ intros)$

lemma $PO\ m3\ DY\ fake\ refines\ m2\ fake:$

```

  {R23 ∩ UNIV × m3-inv1-lkeysec}
    m2-fake, m3-DY-fake
  {> R23}
apply (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros
        del: abs-msg.cases)
apply (auto intro: abs-msg-DY-subset-fakeable [THEN subsetD]
        del: abs-msg.cases)
apply (auto simp add: R23-simps)
done

```

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 ⊆ R23“(init m2)
by (auto simp add: R23-def m3-defs intro!: R23-intros)

```

```

lemma PO-m3-refines-trans-m2 [iff]:
  {R23 ∩ (m2-inv3a-sesK-compr) × (m3-inv7a-sesK-compr ∩ m3-inv1-lkeysec)}
    (trans m2), (trans m3)
  {> R23}
apply (auto simp add: m3-def m3-trans-def m2-def m2-trans-def)
apply (blast intro!: PO-m3-trans-refines-m2-trans)+
done

```

```

lemma PO-m3-observation-consistent [iff]:
  obs-consistent R23 med32 m2 m3
by (auto simp add: obs-consistent-def R23-def med32-def m3-defs)

```

Refinement result.

```

lemma m3-refines-m2 [iff]:
  refines
    (R23 ∩ (m2-inv3a-sesK-compr) × (m3-inv1-lkeysec))
    med32 m2 m3
proof –
  have R23 ∩ m2-inv3a-sesK-compr × UNIV ⊆ UNIV × m3-inv7a-sesK-compr
  by (auto simp add: R23-def R23-keys-simps intro!: m3-inv7a-sesK-comprI)
  thus ?thesis
  by (–) (rule Refinement-using-invariants, auto)
qed

```

```

lemma m3-implements-m2 [iff]:
  implements med32 m2 m3
by (rule refinement-soundness) (auto)

```

3.6.7 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 $PO\text{-}m3\text{-inv3}\text{-ikk}\text{-init}: reach\ m3 \subseteq m3\text{-inv3}\text{-ikk}\text{-init}$

proof (rule $INV\text{-from}\text{-Refinement}\text{-using}\text{-invariants}\ [OF\ m3\text{-refines}\text{-}m2]$)

show $Range\ (R23 \cap m2\text{-inv3a}\text{-ses}K\text{-compr} \times m3\text{-inv1}\text{-lkeysec} \cap m2\text{-inv6}\text{-ikk}\text{-init} \times UNIV)$
 $\subseteq m3\text{-inv3}\text{-ikk}\text{-init}$

by (fastforce simp add: $R23\text{-def}\ R23\text{-keys}\text{-simps}\ intro!$: $m3\text{-inv3}\text{-ikk}\text{-init}I$)

qed auto

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 $PO\text{-}m3\text{-inv4}\text{-ikk}\text{-resp}: reach\ m3 \subseteq m3\text{-inv4}\text{-ikk}\text{-resp}$

proof (rule $INV\text{-from}\text{-Refinement}\text{-using}\text{-invariants}\ [OF\ m3\text{-refines}\text{-}m2]$)

show $Range\ (R23 \cap m2\text{-inv3a}\text{-ses}K\text{-compr} \times m3\text{-inv1}\text{-lkeysec} \cap m2\text{-inv7}\text{-ikk}\text{-resp} \times UNIV)$
 $\subseteq m3\text{-inv4}\text{-ikk}\text{-resp}$

by (auto simp add: $R23\text{-def}\ R23\text{-keys}\text{-simps}\ intro!$: $m3\text{-inv4}\text{-ikk}\text{-resp}I$)

(elim $m2\text{-inv7}\text{-ikk}\text{-resp}E$, auto)

qed auto

end

3.7 Core Kerberos 5 (L3)

theory $m3\text{-kerberos5}$ **imports** $m2\text{-kerberos}\ \dots/Refinement/Message$

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 *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*

by (*auto simp add: keySetup-def ltkkeySetup-def corrKey-def*)

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-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 $\langle \text{runs} = \text{runs } s, \text{leak} = \text{leak } s, \text{clk} = \text{clk } s, \text{cache} = \text{cache } s \rangle$

type-synonym

m3-pred = *m3-state set*

type-synonym

m3-trans = (*m3-state* \times *m3-state*) *set*

3.7.3 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:
 $Ra \notin \text{dom } (\text{runs } s) \wedge$ — 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 $Server$, 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:
 $Rs \notin \text{dom } (\text{runs } s) \wedge$ — fresh server run
 $Kab = \text{sesK } (Rs\$sk) \wedge$ — fresh session key
 $\{\text{Agent } A, \text{Agent } B, \text{Nonce } Na\} \in IK \ s \wedge$ — recv M1
 $Ts = \text{clk } s \wedge$ — fresh timestamp

— actions:
— record session key and send M2
 $s1 = s\langle$
 $\text{runs} := (\text{runs } s)(Rs \mapsto (\text{Serv}, [A, B], [aNon \ Na, aNum \ Ts])),$
 $IK := \text{insert } \{\text{Crypt } (\text{shrK } A) \ \{\text{Key } Kab, \text{Agent } B, \text{Number } Ts, \text{Nonce } Na\},$
 $\text{Crypt } (\text{shrK } B) \ \{\text{Key } Kab, \text{Agent } A, \text{Number } Ts\}\}$
 $(IK \ s)$
 \rangle
 $\}$

definition — by A , refines $m2\text{-step4}$
 $m3\text{-step4} :: [\text{rid-t}, \text{agent}, \text{agent}, \text{nonce}, \text{key}, \text{time}, \text{time}, \text{msg}] \Rightarrow m3\text{-trans}$
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 recv'd
 $Na = Ra\$na \wedge$ — generated nonce
 $\{\text{Crypt } (\text{shrK } A)$ — recv M2
 $\ \{\text{Key } Kab, \text{Agent } B, \text{Number } Ts, \text{Nonce } Na\}, X\} \in IK \ s \wedge$
— read current time
 $Ta = \text{clk } s \wedge$

— check freshness of session key
 $clk\ s < Ts + Ls \wedge$

— actions:
 — record session key and send $M3$
 $s1 = s\{$
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])),$
 $IK := insert\ \{\{Crypt\ Kab\ \{Agent\ A, Number\ Ta\}\}, X\}\ (IK\ s) \text{ — } M3$
 $\}$
 $\}$

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
 $\{\{Crypt\ Kab\ \{Agent\ A, Number\ Ta\}\},$ — recv $M3$
 $Crypt\ (shrK\ B)\ \{\{Key\ Kab, Agent\ A, Number\ Ts\}\}\} \in IK\ s \wedge$
 — 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
 $s1 = s\{$
 $runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])),$
 $cache := insert\ (B, Kab, Ta)\ (cache\ s),$
 $IK := insert\ (Crypt\ Kab\ (Number\ Ta))\ (IK\ s) \text{ — send } M4$
 $\}$
 $\}$

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-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-purge*

$m3\text{-purge} :: \text{agent} \Rightarrow m3\text{-trans}$

where

$m3\text{-purge} \equiv m1\text{-purge}$

Session key compromise.

definition — refines *m2-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:

$\text{runs } s \ Rs = \text{Some } (\text{Serv}, [A, B], [a\text{Non } Na, a\text{Num } Ts]) \wedge$

$(\text{clk } s \geq Ts + Ls) \wedge$ — 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-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

}

3.7.4 Transition system

definition

$m3\text{-init} :: m3\text{-pred}$

where

$m3\text{-init} \equiv \{(|$

$\text{runs} = \text{Map.empty},$

$\text{leak} = \text{shrK}'\text{bad} \times \{\text{undefined}\},$

$\text{clk} = 0,$

$\text{cache} = \{\},$

$IK = \text{Key}'\text{shrK}'\text{bad}$

$|) \}$

definition

```

m3-trans :: m3-trans where
m3-trans ≡ (⋃ A B Ra Rb Rs Na Kab Ts Ta T X.
  m3-step1 Ra A B Na ∪
  m3-step2 Rb A B ∪
  m3-step3 Rs A B Kab Na Ts ∪
  m3-step4 Ra A B Na Kab Ts Ta X ∪
  m3-step5 Rb A B Kab Ts Ta ∪
  m3-step6 Ra A B Na Kab Ts Ta ∪
  m3-tick T ∪
  m3-purge A ∪
  m3-leak Rs A B Na Ts ∪
  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-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 $m3\text{-inv1-lkeysec}D = m3\text{-inv1-lkeysec-def}$ [THEN setc-def-to-dest, rule-format]

Invariance proof.

lemma $PO\text{-}m3\text{-inv1-lkeysec-init}$ [iff]:

$init\ m3 \subseteq m3\text{-inv1-lkeysec}$

by (auto simp add: m3-defs intro!: m3-inv1-lkeysecI)

lemma $PO\text{-}m3\text{-inv1-lkeysec-trans}$ [iff]:

$\{m3\text{-inv1-lkeysec}\ trans\ m3 \{>\ m3\text{-inv1-lkeysec}\}$

by (fastforce simp add: PO-hoare-defs m3-defs intro!: m3-inv1-lkeysecI)

lemma $PO\text{-}m3\text{-inv1-lkeysec}$ [iff]: $reach\ m3 \subseteq m3\text{-inv1-lkeysec}$

by (rule inv-rule-basic) (fast+)

Useful simplifier lemmas

lemma $m3\text{-inv1-lkeysec-for-parts}$ [simp]:

$\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies Key\ (shrK\ C) \in parts\ (IK\ s) \longleftrightarrow C \in bad$

by auto

lemma $m3\text{-inv1-lkeysec-for-analz}$ [simp]:

$\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies Key\ (shrK\ C) \in analz\ (IK\ s) \longleftrightarrow C \in bad$

by auto

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

$m3\text{-inv2-sesK-compr} \equiv \{s. \forall K\ KK.$

$KK \subseteq range\ sesK \longrightarrow$

$(Key\ K \in analz\ (Key\ KK \cup (IK\ s))) = (K \in KK \vee Key\ K \in analz\ (IK\ s))$

$\}$

lemmas $m3\text{-inv2-sesK-compr}I = m3\text{-inv2-sesK-compr-def}$ [THEN setc-def-to-intro, rule-format]

lemmas $m3\text{-inv2-sesK-compr}E = m3\text{-inv2-sesK-compr-def}$ [THEN setc-def-to-elim, rule-format]

lemmas $m3\text{-inv2-sesK-compr}D = m3\text{-inv2-sesK-compr-def}$ [THEN setc-def-to-dest, rule-format]

Additional lemma

lemmas $insert-commute-Key = insert-commute$ [where $x=Key\ K$ for K]

lemmas $m3\text{-inv2-sesK-compr-simps} =$

$m3\text{-inv2-sesK-compr}D$

$m3\text{-inv2-sesK-compr}D$ [where $KK=insert\ Kab\ KK$ for $Kab\ KK$, simplified]

$m3\text{-inv2-sesK-compr}D$ [where $KK=\{Kab\}$ for Kab , simplified]

$insert-commute-Key$

3.7.6 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

$am\text{-}M1:$

$\{Agent\ A, Agent\ B, Nonce\ N\} \in H$

$\Rightarrow Insec\ A\ B\ (Msg\ [aNon\ N]) \in abs\text{-}msg\ H$

| $am\text{-}M2a:$

$Crypt\ (shrK\ C)\ \{Key\ K, Agent\ B, Number\ T, Nonce\ N\} \in H$

$\Rightarrow Secure\ Sv\ C\ (Msg\ [aKey\ K, aAgt\ B, aNum\ T, aNon\ N]) \in abs\text{-}msg\ H$

| $am\text{-}M2b:$

$Crypt\ (shrK\ C)\ \{Key\ K, Agent\ A, Number\ T\} \in H$

$\Rightarrow Secure\ Sv\ C\ (Msg\ [aKey\ K, aAgt\ A, aNum\ T]) \in abs\text{-}msg\ H$

| $am\text{-}M3:$

$Crypt\ K\ \{Agent\ A, Number\ T\} \in H$

$\Rightarrow dAuth\ K\ (Msg\ [aAgt\ A, aNum\ T]) \in abs\text{-}msg\ H$

| $am\text{-}M4:$

$Crypt\ K\ (Number\ T) \in H$

$\Rightarrow dAuth\ K\ (Msg\ [aNum\ T]) \in abs\text{-}msg\ H$

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

$R23\text{-}non :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$

$R23\text{-}non \equiv \{(s, t). \forall KK\ N. KK \subseteq range\ sesK \longrightarrow$

$Nonce\ N \in analz\ (Key'KK \cup (IK\ t)) \longleftrightarrow aNon\ N \in extr\ (aKey'KK \cup ik0)\ (chan\ s)$

$\}$

definition

$R23\text{-}pres :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$

$R23\text{-}pres \equiv \{(s, t). runs\ s = runs\ t \wedge leak\ s = leak\ t \wedge clk\ s = clk\ t \wedge cache\ s = cache\ t\}$

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 $R23\text{-nonI} = R23\text{-non-def}$ [*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 $R23\text{-presI} = R23\text{-pres-def}$ [*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 $R23\text{-intros} = R23\text{-msgsI}$ $R23\text{-keysI}$ $R23\text{-nonI}$ $R23\text{-presI}$

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 $R23\text{-keys-simps} =$
 $R23\text{-keys-simp}$
 $R23\text{-keys-simp}$ [**where** $KK = \{\}$, *simplified*]
 $R23\text{-keys-simp}$ [**where** $KK = \{K\}$ **for** K' , *simplified*]
 $R23\text{-keys-simp}$ [**where** $KK = \text{insert } K' KK$ **for** $K' KK$, *simplified, OF - conjI*]

lemmas $R23\text{-non-simp} = R23\text{-non-def}$ [*THEN rel-def-to-dest, simplified, rule-format*]

lemmas $R23\text{-non-simps} =$
 $R23\text{-non-simp}$
 $R23\text{-non-simp}$ [**where** $KK = \{\}$, *simplified*]
 $R23\text{-non-simp}$ [**where** $KK = \{K\}$ **for** K , *simplified*]
 $R23\text{-non-simp}$ [**where** $KK = \text{insert } K KK$ **for** $K KK$, *simplified, OF - conjI*]

lemmas $R23\text{-simps} = R23\text{-keys-simps}$ $R23\text{-non-simps}$

General lemmas

General facts about *abs-msg*

declare $abs\text{-msg.intros}$ [*intro!*]
declare $abs\text{-msg.cases}$ [*elim!*]

lemma $abs\text{-msg.empty}$: $abs\text{-msg } \{\} = \{\}$
by (*auto*)

lemma $abs\text{-msg.Un}$ [*simp*]:
 $abs\text{-msg } (G \cup H) = abs\text{-msg } G \cup abs\text{-msg } H$
by (*auto*)

lemma *abs-msg-mono* [*elim*]:
 $\llbracket m \in \text{abs-msg } G; G \subseteq H \rrbracket \Longrightarrow m \in \text{abs-msg } H$
by (*auto*)

lemma *abs-msg-insert-mono* [*intro*]:
 $\llbracket m \in \text{abs-msg } H \rrbracket \Longrightarrow m \in \text{abs-msg } (\text{insert } m' H)$
by (*auto*)

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*:
 $\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; (s, t) \in R23\text{-non}; t \in m3\text{-inv1-lkeysec} \rrbracket$
 $\Longrightarrow \text{abs-msg } (\text{synth } (\text{analz } (IK\ t))) \subseteq \text{fake ik0 } (\text{dom } (\text{runs } s)) (\text{chan } s)$
apply (*auto*)
— 9 subgoals, deal with replays first
prefer 2 apply (*blast*)
prefer 3 apply (*blast*)
prefer 4 apply (*blast*)
prefer 5 apply (*blast*)
— remaining 5 subgoals are real fakes
apply (*intro fake-StatCh fake-DynCh, auto simp add: R23-simps*) +
done

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*:
 $\{R23\}$
 $(m2\text{-step1 } Ra\ A\ B\ Na), (m3\text{-step1 } Ra\ A\ B\ Na)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros*) (*auto*)

lemma *PO-m3-step2-refines-m2-step2*:
 $\{R23\}$
 $(m2\text{-step2 } Rb\ A\ B), (m3\text{-step2 } Rb\ A\ B)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros*)

lemma *PO-m3-step3-refines-m2-step3*:
 $\{R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv2-sesK-compr} \cap m3\text{-inv1-lkeysec})\}$
 $(m2\text{-step3 } Rs\ A\ B\ Kab\ Na\ Ts), (m3\text{-step3 } Rs\ A\ B\ Kab\ Na\ Ts)$
 $\{> R23\}$
proof –
 $\{ \text{fix } s\ t$
assume *H*:
 $(s, t) \in R23\text{-msgs } (s, t) \in R23\text{-keys } (s, t) \in R23\text{-non}$
 $(s, t) \in R23\text{-pres}$
 $s \in m2\text{-inv3a-sesK-compr}$

```

    t ∈ m3-inv2-sesK-compr t ∈ m3-inv1-lkeysec
    Kab = sesK (Rs$sk) Rs ∉ dom (runs t)
    ⌊ Agent A, Agent B, Nonce Na ⌋ ∈ parts (IK t)
  let ?s' =
    s( runs := (runs s)(Rs ↦ (Serv, [A, B], [aNon Na, aNum (clk t)])),
      chan := insert (Secure Sv A (Msg [aKey Kab, aAgt B, aNum (clk t), aNon Na]))
        (insert (Secure Sv B (Msg [aKey Kab, aAgt A, aNum (clk t)])) (chan s)) )
  let ?t' =
    t( runs := (runs t)(Rs ↦ (Serv, [A, B], [aNon Na, aNum (clk t)])),
      IK := insert
        ⌊ Crypt (shrK A) ⌋ Key Kab, Agent B, Number (clk t), Nonce Na ⌋,
        Crypt (shrK B) ⌋ Key Kab, Agent A, Number (clk t) ⌋ ⌋
        (IK t) )
  — here we go
  have (?s', ?t') ∈ R23-msgs using H
  by (−) (rule R23-intros, auto)
  moreover
  have (?s', ?t') ∈ R23-keys using H
  by (−) (rule R23-intros,
    auto simp add: m2-inv3a-sesK-compr-simps m3-inv2-sesK-compr-simps,
    auto simp add: R23-keys-simps)
  moreover
  have (?s', ?t') ∈ R23-non using H
  by (−)
  (rule R23-intros,
    auto simp add: m2-inv3a-sesK-compr-simps m3-inv2-sesK-compr-simps R23-non-simps)
  moreover
  have (?s', ?t') ∈ R23-pres using H
  by (−) (rule R23-intros, auto)
  moreover
  note calculation
}
thus ?thesis
by (auto simp add: PO-rhoare-defs R23-def m3-defs)
qed

```

lemma *PO-m3-step4-refines-m2-step4*:

```

{R23 ∩ UNIV × m3-inv1-lkeysec}
(m2-step4 Ra A B Na Kab Ts Ta), (m3-step4 Ra A B Na Kab Ts Ta X)
{> R23}

```

proof –

```

{ fix s t
  assume H:
    (s, t) ∈ R23-msgs (s, t) ∈ R23-keys (s, t) ∈ R23-non
    (s, t) ∈ R23-pres t ∈ m3-inv1-lkeysec
    runs t Ra = Some (Init, [A, B], []) Na = Ra$na
    ⌊ Crypt (shrK A) ⌋ Key Kab, Agent B, Number Ts, Nonce Na ⌋, X ⌋ ∈ analz (IK t)
  let ?s' = s( runs := (runs s)(Ra ↦ (Init, [A, B], [aKey Kab, aNum Ts, aNum (clk t)])),
    chan := insert (dAuth Kab (Msg [aAgt A, aNum (clk t)])) (chan s) )
  and ?t' = t( runs := (runs t)(Ra ↦ (Init, [A, B], [aKey Kab, aNum Ts, aNum (clk t)])),
    IK := insert ⌊ Crypt Kab ⌋ Agent A, Number (clk t) ⌋, X ⌋ (IK t) )
  from H have
    Secure Sv A (Msg [aKey Kab, aAgt B, aNum Ts, aNon Na]) ∈ chan s

```

by (auto dest!: analz-into-parts elim!: MPair-parts)
moreover
 from H have $X \in \text{parts } (IK\ t)$ by (auto)
 with H have $(?s', ?t') \in R23\text{-msgs}$ by (auto intro!: R23-intros) (auto)
moreover
 from H have $X \in \text{analz } (IK\ t)$ by (auto)
 with H have $(?s', ?t') \in R23\text{-keys}$
 by (auto intro!: R23-intros) (auto dest!: analz-cut intro: analz-monotonic)
moreover
 from H have $X \in \text{analz } (IK\ t)$ by (auto)
 with H have $(?s', ?t') \in R23\text{-non}$
 by (auto intro!: R23-intros) (auto dest!: analz-cut intro: analz-monotonic)
moreover
 have $(?s', ?t') \in R23\text{-pres}$ using H
 by (auto intro!: R23-intros)
moreover
 note calculation
 }
 thus ?thesis
 by (auto simp add: PO-rhoare-defs R23-def m3-defs dest!: analz-Inj-IK)
 qed

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}
 by (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros)
 (auto)

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}
 by (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros)

lemma *PO-m3-tick-refines-m2-tick*:
 {R23}
 ($m2\text{-tick } T$), ($m3\text{-tick } T$)
 {>R23}
 by (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros)

lemma *PO-m3-purge-refines-m2-purge*:
 {R23}
 ($m2\text{-purge } A$), ($m3\text{-purge } A$)
 {>R23}
 by (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros)

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}
 by (auto simp add: PO-rhoare-defs R23-def m3-defs R23-simps intro!: R23-intros)

lemma *PO-m3-DY-fake-refines-m2-fake*:
 $\{R23 \cap m2\text{-inv}3a\text{-sesK-compr} \times (m3\text{-inv}2\text{-sesK-compr} \cap m3\text{-inv}1\text{-lkeysec})\}$
 $m2\text{-fake}, m3\text{-DY-fake}$
 $\{> R23\}$
apply (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros*
del: abs-msg.cases)
apply (*auto intro: abs-msg-DY-subset-fakeable [THEN subsetD]*
del: abs-msg.cases)
apply (*auto simp add: R23-simps*)
done

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*]:
 $init\ m3 \subseteq R23 \text{“}(init\ m2)$
by (*auto simp add: R23-def m3-defs intro!: R23-intros*)

lemma *PO-m3-refines-trans-m2* [*iff*]:
 $\{R23 \cap (m2\text{-inv}3a\text{-sesK-compr}) \times (m3\text{-inv}2\text{-sesK-compr} \cap m3\text{-inv}1\text{-lkeysec})\}$
 $(trans\ m2), (trans\ m3)$
 $\{> R23\}$
by (*auto simp add: m3-def m3-trans-def m2-def m2-trans-def*)
(blast intro!: PO-m3-trans-refines-m2-trans)+

lemma *PO-m3-observation-consistent* [*iff*]:
 $obs\text{-consistent}\ R23\ med32\ m2\ m3$
by (*auto simp add: obs-consistent-def R23-def med32-def m3-defs*)

Refinement result.

lemma *m3-refines-m2* [*iff*]:
refines
 $(R23 \cap (m2\text{-inv}3a\text{-sesK-compr}) \times (m3\text{-inv}1\text{-lkeysec}))$
 $med32\ m2\ m3$

proof –
have $R23 \cap m2\text{-inv}3a\text{-sesK-compr} \times UNIV \subseteq UNIV \times m3\text{-inv}2\text{-sesK-compr}$
by (*auto simp add: R23-def R23-keys-simps intro!: m3-inv2-sesK-comprI*)
thus *?thesis*
by (–) (*rule Refinement-using-invariants, auto*)
qed

lemma *m3-implements-m2* [*iff*]:
 $implements\ med32\ m2\ m3$
by (*rule refinement-soundness*) (*auto*)

3.7.7 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 $PO\text{-}m3\text{-inv3}\text{-ikk}\text{-init}: reach\ m3 \subseteq m3\text{-inv3}\text{-ikk}\text{-init}$

proof (rule $INV\text{-from}\text{-Refinement}\text{-using}\text{-invariants}\ [OF\ m3\text{-refines}\text{-}m2]$)

show $Range\ (R23 \cap m2\text{-inv3a}\text{-ses}K\text{-compr} \times m3\text{-inv1}\text{-lkeysec} \cap m2\text{-inv6}\text{-ikk}\text{-init} \times UNIV)$
 $\subseteq m3\text{-inv3}\text{-ikk}\text{-init}$

by (fastforce simp add: $R23\text{-def}\ R23\text{-keys}\text{-simps}\ intro!:$ $m3\text{-inv3}\text{-ikk}\text{-init}I$)

qed auto

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 $PO\text{-}m3\text{-inv4}\text{-ikk}\text{-resp}: reach\ m3 \subseteq m3\text{-inv4}\text{-ikk}\text{-resp}$

proof (rule $INV\text{-from}\text{-Refinement}\text{-using}\text{-invariants}\ [OF\ m3\text{-refines}\text{-}m2]$)

show $Range\ (R23 \cap m2\text{-inv3a}\text{-ses}K\text{-compr} \times m3\text{-inv1}\text{-lkeysec} \cap m2\text{-inv7}\text{-ikk}\text{-resp} \times UNIV)$
 $\subseteq m3\text{-inv4}\text{-ikk}\text{-resp}$

by (auto simp add: $R23\text{-def}\ R23\text{-keys}\text{-simps}\ intro!:$ $m3\text{-inv4}\text{-ikk}\text{-resp}I$)

(elim $m2\text{-inv7}\text{-ikk}\text{-resp}E$, auto)

qed auto

end

3.8 Core Kerberos 4 (L3)

theory $m3\text{-kerberos4}$ **imports** $m2\text{-kerberos}\ \dots/Refinement/Message$

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 *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'bad*

by (*auto simp add: keySetup-def ltkkeySetup-def corrKey-def*)

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 $\langle \text{runs} = \text{runs } s, \text{leak} = \text{leak } s, \text{clk} = \text{clk } s, \text{cache} = \text{cache } s \rangle$

type-synonym

m3-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

$m3\text{-step1 } Ra \ A \ B \ Na \equiv \{(s, s1)\}.$
— guards:
 $Ra \notin \text{dom } (\text{runs } s) \wedge$ — Ra is fresh
 $Na = Ra\$na \wedge$ — generated nonce
— actions:
 $s1 = s\{$
 $\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [])),$
 $IK := \text{insert } \{\text{Agent } A, \text{Agent } B, \text{Nonce } Na\} (IK \ s)$ — send $M1$
 $\}$
 $\}$

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 $m2\text{-step3}$
 $m3\text{-step3} :: [\text{rid-t}, \text{agent}, \text{agent}, \text{key}, \text{nonce}, \text{time}] \Rightarrow m3\text{-trans}$
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
 $\{\text{Agent } A, \text{Agent } B, \text{Nonce } Na\} \in IK \ s \wedge$ — recv $M1$
 $Ts = \text{clk } s \wedge$ — fresh timestamp

— actions:
— record session key and send $M2$
 $s1 = s\{$
 $\text{runs} := (\text{runs } s)(Rs \mapsto (\text{Serv}, [A, B], [aNon \ Na, aNum \ Ts])),$
 $IK := \text{insert } (\text{Crypt } (\text{shrK } A)$ — send $M2$
 $\{\text{Key } Kab, \text{Agent } B, \text{Number } Ts, \text{Nonce } Na,$
 $\text{Crypt } (\text{shrK } B) \{\text{Key } Kab, \text{Agent } A, \text{Number } Ts\}\})$
 $(IK \ s)$
 $\}$
 $\}$

definition — by A , refines $m2\text{-step4}$
 $m3\text{-step4} :: [\text{rid-t}, \text{agent}, \text{agent}, \text{nonce}, \text{key}, \text{time}, \text{time}, \text{msg}] \Rightarrow m3\text{-trans}$
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 recv'd
 $Na = Ra\$na \wedge$ — generated nonce
 $\text{Crypt } (\text{shrK } A)$ — recv $M2$
 $\{\text{Key } Kab, \text{Agent } B, \text{Number } Ts, \text{Nonce } Na, X\} \in IK \ s \wedge$
— read current time
 $Ta = \text{clk } s \wedge$

— check freshness of session key
 $clk\ s < Ts + Ls \wedge$

— actions:
— record session key and send $M3$
 $s1 = s\{$
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])),$
 $IK := insert\ \{\{Crypt\ Kab\ \{Agent\ A, Number\ Ta\}\}, X\}\ (IK\ s) \text{ — } M3$
 $\}$
 $\}$

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 rcv'd

$\{\{Crypt\ Kab\ \{Agent\ A, Number\ Ta\}\},$ — rcv $M3$
 $Crypt\ (shrK\ B)\ \{Key\ Kab, Agent\ A, Number\ Ts\}\}\} \in IK\ s \wedge$

— 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
 $s1 = s\{$
 $runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta])),$
 $cache := insert\ (B, Kab, Ta)\ (cache\ s),$
 $IK := insert\ (Crypt\ Kab\ (Number\ Ta))\ (IK\ s) \text{ — send } M4$
 $\}$
 $\}$

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

$Crypt\ Kab\ (Number\ Ta) \in IK\ s \wedge$ — rcv $M4$

— actions:
 $s' = s\{$
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts, aNum\ Ta, END]))$
 $\}$

}

Clock tick event

definition — refines *m2-tick*

m3-tick :: *time* \Rightarrow *m3-trans*

where

m3-tick \equiv *m1-tick*

Purge event: purge cache of expired timestamps

definition — refines *m2-purge*

m3-purge :: *agent* \Rightarrow *m3-trans*

where

m3-purge \equiv *m1-purge*

Session key compromise.

definition — refines *m2-leak*

m3-leak :: [*rid-t*, *agent*, *agent*, *nonce*, *time*] \Rightarrow *m3-trans*

where

m3-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*) \downarrow

}

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*)) \downarrow) — take DY closure

}

3.8.4 Transition system

definition

m3-init :: *m3-pred*

where

m3-init \equiv { \downarrow

runs = *Map.empty*,

leak = *shrK*'*bad* \times {*undefined*},

clk = 0,

cache = {},

IK = *Key*'*shrK*'*bad*

\downarrow }

definition

```

m3-trans :: m3-trans where
m3-trans ≡ (⋃ A B Ra Rb Rs Na Kab Ts Ta T X.
  m3-step1 Ra A B Na ∪
  m3-step2 Rb A B ∪
  m3-step3 Rs A B Kab Na Ts ∪
  m3-step4 Ra A B Na Kab Ts Ta X ∪
  m3-step5 Rb A B Kab Ts Ta ∪
  m3-step6 Ra A B Na Kab Ts Ta ∪
  m3-tick T ∪
  m3-purge A ∪
  m3-leak Rs A B Na Ts ∪
  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-step6-def m3-tick-def m3-purge-def m3-leak-def m3-DY-fake-def

```

lemmas *m3-defs* = *m3-loc-defs m2-defs*

3.8.5 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]
```

inv4: Secrecy of pre-distributed shared keys**definition**

```
m3-inv4-lkeysec :: m3-pred
```

where

```

m3-inv4-lkeysec ≡ {s. ∀ C.
  (Key (shrK C) ∈ parts (IK s) → C ∈ bad) ∧
  (C ∈ bad → Key (shrK C) ∈ IK s)
}

```

lemmas $m3\text{-inv4}\text{-lkeysec}I = m3\text{-inv4}\text{-lkeysec}\text{-def}$ [*THEN setc-def-to-intro, rule-format*]
lemmas $m3\text{-inv4}\text{-lkeysec}E$ [*elim*] = $m3\text{-inv4}\text{-lkeysec}\text{-def}$ [*THEN setc-def-to-elim, rule-format*]
lemmas $m3\text{-inv4}\text{-lkeysec}D = m3\text{-inv4}\text{-lkeysec}\text{-def}$ [*THEN setc-def-to-dest, rule-format*]

Invariance proof.

lemma $PO\text{-}m3\text{-inv4}\text{-lkeysec}\text{-init}$ [*iff*]:
 $init\ m3 \subseteq m3\text{-inv4}\text{-lkeysec}$
by (*auto simp add: m3-defs intro!: m3-inv4-lkeysecI*)

lemma $PO\text{-}m3\text{-inv4}\text{-lkeysec}\text{-trans}$ [*iff*]:
 $\{m3\text{-inv4}\text{-lkeysec}\}$ *trans* $m3 \{> m3\text{-inv4}\text{-lkeysec}\}$
by (*auto simp add: PO-hoare-defs m3-defs intro!: m3-inv4-lkeysecI*)
(*auto dest!: Body*)

lemma $PO\text{-}m3\text{-inv4}\text{-lkeysec}$ [*iff*]: $reach\ m3 \subseteq m3\text{-inv4}\text{-lkeysec}$
by (*rule inv-rule-incr*) (*fast+*)

Useful simplifier lemmas

lemma $m3\text{-inv4}\text{-lkeysec}\text{-for}\text{-parts}$ [*simp*]:
 $\llbracket s \in m3\text{-inv4}\text{-lkeysec} \rrbracket \implies Key\ (shrK\ C) \in parts\ (IK\ s) \longleftrightarrow C \in bad$
by *auto*

lemma $m3\text{-inv4}\text{-lkeysec}\text{-for}\text{-analz}$ [*simp*]:
 $\llbracket s \in m3\text{-inv4}\text{-lkeysec} \rrbracket \implies Key\ (shrK\ C) \in analz\ (IK\ s) \longleftrightarrow C \in bad$
by *auto*

inv6: Ticket shape for honestly encrypted M2

definition

$m3\text{-inv6}\text{-ticket} :: m3\text{-pred}$

where

$m3\text{-inv6}\text{-ticket} \equiv \{s. \forall A\ B\ T\ K\ N\ X.$
 $A \notin bad \longrightarrow$
 $Crypt\ (shrK\ A)\ \{\!\!| Key\ K, Agent\ B, Number\ T, Nonce\ N, X \!\!\} \in parts\ (IK\ s) \longrightarrow$
 $X = Crypt\ (shrK\ B)\ \{\!\!| Key\ K, Agent\ A, Number\ T \!\!\} \wedge K \in range\ sesK$
 $\}$

lemmas $m3\text{-inv6}\text{-ticket}I = m3\text{-inv6}\text{-ticket}\text{-def}$ [*THEN setc-def-to-intro, rule-format*]
lemmas $m3\text{-inv6}\text{-ticket}E$ [*elim*] = $m3\text{-inv6}\text{-ticket}\text{-def}$ [*THEN setc-def-to-elim, rule-format*]
lemmas $m3\text{-inv6}\text{-ticket}D = m3\text{-inv6}\text{-ticket}\text{-def}$ [*THEN setc-def-to-dest, rule-format, rotated -1*]

Invariance proof.

lemma $PO\text{-}m3\text{-inv6}\text{-ticket}\text{-init}$ [*iff*]:
 $init\ m3 \subseteq m3\text{-inv6}\text{-ticket}$
by (*auto simp add: m3-defs intro!: m3-inv6-ticketI*)

lemma $PO\text{-}m3\text{-inv6}\text{-ticket}\text{-trans}$ [*iff*]:
 $\{m3\text{-inv6}\text{-ticket} \cap m3\text{-inv4}\text{-lkeysec}\}$ *trans* $m3 \{> m3\text{-inv6}\text{-ticket}\}$
apply (*auto simp add: PO-hoare-defs m3-defs intro!: m3-inv6-ticketI*)
apply (*auto dest: m3-inv6-ticketD*)
— 2 subgoals
apply (*drule parts-cut, drule Body, auto dest: m3-inv6-ticketD*)+

done

lemma *PO-m3-inv6-ticket* [iff]: $reach\ m3 \subseteq m3\text{-inv6}\text{-ticket}$
by (rule *inv-rule-incr*) (auto del: *subsetI*)

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

$m3\text{-inv7a}\text{-sesK}\text{-compr} \equiv \{s. \forall K\ KK.$

$KK \subseteq \text{range}\ \text{sesK} \longrightarrow$

$(\text{Key}\ K \in \text{analz}\ (\text{Key}\ 'KK \cup (IK\ s))) = (K \in KK \vee \text{Key}\ K \in \text{analz}\ (IK\ s))$

$\}$

lemmas *m3-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 $KK = \text{insert}\ Kab\ KK$ for $Kab\ KK$, *simplified*]

m3-inv7a-sesK-comprD [where $KK = \{Kab\}$ for Kab , *simplified*]

insert-commute-Key

Invariance proof.

lemma *PO-m3-inv7a-sesK-compr-step4*:

$\{m3\text{-inv7a}\text{-sesK}\text{-compr} \cap m3\text{-inv6}\text{-ticket} \cap m3\text{-inv4}\text{-lkeysec}\}$

$m3\text{-step4}\ Ra\ A\ B\ Na\ Kab\ Ts\ Ta\ X$

$\{>\ m3\text{-inv7a}\text{-sesK}\text{-compr}\}$

proof –

$\{ \text{fix}\ K\ KK\ s$

assume H :

$s \in m3\text{-inv4}\text{-lkeysec}\ s \in m3\text{-inv7a}\text{-sesK}\text{-compr}\ s \in m3\text{-inv6}\text{-ticket}$

$\text{runs}\ s\ Ra = \text{Some}\ (\text{Init}, [A, B], [])$

$Na = Ra\$na$

$KK \subseteq \text{range}\ \text{sesK}$

$\text{Crypt}\ (\text{shrK}\ A)\ \{\text{Key}\ Kab, \text{Agent}\ B, \text{Number}\ Ts, \text{Nonce}\ Na, X\} \in \text{analz}\ (IK\ s)$

have

$(\text{Key}\ K \in \text{analz}\ (\text{insert}\ X\ (\text{Key}\ 'KK \cup IK\ s))) =$

$(K \in KK \vee \text{Key}\ K \in \text{analz}\ (\text{insert}\ X\ (IK\ s)))$

proof (*cases* $A \in \text{bad}$)

case True

with H **have** $X \in \text{analz}\ (IK\ s)$ **by** (auto *dest!*: *Decrypt*)

```

moreover
  with  $H$  have  $X \in \text{analz} (\text{Key } 'KK \cup IK \ s)$ 
  by (auto intro: analz-monotonic)
ultimately show ?thesis using  $H$ 
  by (auto simp add: m3-inv7a-sesK-compr-simps analz-insert-eq)
next
  case False thus ?thesis using  $H$ 
  by (fastforce simp add: m3-inv7a-sesK-compr-simps
      dest!: m3-inv6-ticketD [OF analz-into-parts])
  qed
}
thus ?thesis
by (auto simp add: PO-hoare-defs m3-defs intro!: m3-inv7a-sesK-comprI dest!: analz-Inj-IK)
qed

```

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-inv7a-sesK-compr
by (auto simp add: m3-defs intro!: m3-inv7a-sesK-comprI)

```

```

lemma PO-m3-inv7a-sesK-compr-trans [iff]:
  {m3-inv7a-sesK-compr  $\cap$  m3-inv6-ticket  $\cap$  m3-inv4-lkeysec}
  trans m3
  { $>$  m3-inv7a-sesK-compr}
by (auto simp add: m3-def m3-trans-def intro!: PO-m3-inv7a-sesK-compr-trans-lemmas)
  (auto simp add: PO-hoare-defs m3-defs m3-inv7a-sesK-compr-simps intro!: m3-inv7a-sesK-comprI)

```

```

lemma PO-m3-inv7a-sesK-compr [iff]: reach m3  $\subseteq$  m3-inv7a-sesK-compr
by (rule-tac J=m3-inv6-ticket  $\cap$  m3-inv4-lkeysec in inv-rule-incr) (auto)

```

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-inv7b-sesK-compr-non :: m3-pred

```

where

```

m3-inv7b-sesK-compr-non  $\equiv$  {s.  $\forall N \ KK$ .

```

```

   $KK \subseteq \text{range } \text{sesK} \longrightarrow (\text{Nonce } N \in \text{analz} (\text{Key}'KK \cup (IK \ s))) = (\text{Nonce } N \in \text{analz} (IK \ s))$ 

```

```

}
```

```

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  $KK = \text{insert } Kab \ KK$  for  $Kab \ KK$ , 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*:

$\{m3\text{-inv7b-sesK-compr-non}\} m3\text{-step3 } Rs A B Kab Na Ts \{> m3\text{-inv7b-sesK-compr-non}\}$

by (*auto simp add: PO-hoare-defs m3-defs m3-inv7b-sesK-compr-non-simps*
intro!: m3-inv7b-sesK-compr-nonI dest!: analz-Inj-IK)

lemma *PO-m3-inv7b-sesK-compr-non-step4*:

$\{m3\text{-inv7b-sesK-compr-non} \cap m3\text{-inv6-ticket} \cap m3\text{-inv4-lkeysec}\}$

$m3\text{-step4 } Ra A B Na Kab Ts Ta X$

$\{> m3\text{-inv7b-sesK-compr-non}\}$

proof –

{ **fix** $N KK s$

assume H :

$s \in m3\text{-inv4-lkeysecs} \in m3\text{-inv7b-sesK-compr-non}$

$s \in m3\text{-inv6-ticket}$

$runs s Ra = Some (Init, [A, B], [])$

$Na = Ra\$na$

$KK \subseteq range sesK$

$Crypt (shrK A) \{Key Kab, Agent B, Number Ts, Nonce Na, X\} \in analz (IK s)$

have

$(Nonce N \in analz (insert X (Key ' KK \cup IK s))) =$

$(Nonce N \in analz (insert X (IK s)))$

proof (*cases*)

assume $A \in bad$ **show** $?thesis$

proof –

from $\langle A \in bad \rangle$ **have** $X \in analz (IK s)$ **using** H

by (*auto dest!: Decrypt*)

moreover

hence $X \in analz (Key ' KK \cup IK s)$

by (*auto intro: analz-monotonic*)

ultimately show $?thesis$ **using** H

by (*auto simp add: m3-inv7b-sesK-compr-non-simps analz-insert-eq*)

qed

next

assume $A \notin bad$ **thus** $?thesis$ **using** H

by (*fastforce simp add: m3-inv7b-sesK-compr-non-simps*

dest!: m3-inv6-ticketD [OF analz-into-parts])

qed

}

thus $?thesis$

by (*auto simp add: PO-hoare-defs m3-defs intro!: m3-inv7b-sesK-compr-nonI*

dest!: analz-Inj-IK)

qed

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]*:

$init m3 \subseteq m3\text{-inv7b-sesK-compr-non}$

by (*auto simp add: m3-defs intro!: m3-inv7b-sesK-compr-nonI*)

lemma *PO-m3-inv7b-sesK-compr-non-trans* [iff]:

$$\{m3\text{-inv7b-sesK-compr-non} \cap m3\text{-inv6-ticket} \cap m3\text{-inv4-lkeysec}\} \\ \text{trans } m3 \\ \{> m3\text{-inv7b-sesK-compr-non}\}$$

by (*auto simp add: m3-def m3-trans-def intro!: PO-m3-inv7b-sesK-compr-non-trans-lemmas*)
(auto simp add: PO-hoare-defs m3-defs m3-inv7b-sesK-compr-non-simps
intro!: m3-inv7b-sesK-compr-nonI)

lemma *PO-m3-inv7b-sesK-compr-non* [iff]: *reach m3* \subseteq *m3-inv7b-sesK-compr-non*

by (*rule-tac J=m3-inv6-ticket* \cap *m3-inv4-lkeysec* **in** *inv-rule-incr*)
(auto)

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:

$$\{\text{Agent } A, \text{Agent } B, \text{Nonce } N\} \in H \\ \Longrightarrow \text{Insec } A \ B \ (\text{Msg } [a\text{Non } N]) \in \text{abs-msg } H$$

| *am-M2a*:

$$\text{Crypt } (\text{shrK } C) \ \{\text{Key } K, \text{Agent } B, \text{Number } T, \text{Nonce } N, X\} \in H \\ \Longrightarrow \text{Secure } Sv \ C \ (\text{Msg } [a\text{Key } K, a\text{Agt } B, a\text{Num } T, a\text{Non } N]) \in \text{abs-msg } H$$

| *am-M2b*:

$$\text{Crypt } (\text{shrK } C) \ \{\text{Key } K, \text{Agent } A, \text{Number } T\} \in H \\ \Longrightarrow \text{Secure } Sv \ C \ (\text{Msg } [a\text{Key } K, a\text{Agt } A, a\text{Num } T]) \in \text{abs-msg } H$$

| *am-M3*:

$$\text{Crypt } K \ \{\text{Agent } A, \text{Number } T\} \in H \\ \Longrightarrow \text{dAuth } K \ (\text{Msg } [a\text{Agt } A, a\text{Num } T]) \in \text{abs-msg } H$$

| *am-M4*:

$$\text{Crypt } K \ (\text{Number } T) \in H \\ \Longrightarrow \text{dAuth } K \ (\text{Msg } [a\text{Num } T]) \in \text{abs-msg } H$$

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-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-keys \equiv $\{(s, t). \forall KK \ K. KK \subseteq \text{range } \text{sesK} \longrightarrow \\ \text{Key } K \in \text{analz } (\text{Key}'KK \cup (IK \ t)) \longrightarrow a\text{Key } K \in \text{extr } (a\text{Key}'KK \cup ik0) (\text{chan } s)\}$

definition

$R23\text{-non} :: (m2\text{-state} \times m3\text{-state}) \text{ set where}$
 $R23\text{-non} \equiv \{(s, t). \forall KK N. KK \subseteq \text{range ses}K \longrightarrow$
 $\quad \text{Nonce } N \in \text{analz } (Key'KK \cup (IK t)) \longrightarrow aNon N \in \text{extr } (aKey'KK \cup ik0) (\text{chan } s)$
 $\}$

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

$med32 :: m3\text{-obs} \Rightarrow m2\text{-obs where}$
 $med32 \equiv id$

lemmas $R23\text{-msgsI} = R23\text{-msgs-def } [THEN \text{rel-def-to-intro, simplified, rule-format}]$

lemmas $R23\text{-msgsE } [elim] = R23\text{-msgs-def } [THEN \text{rel-def-to-elim, simplified, rule-format}]$

lemmas $R23\text{-msgsE}' [elim] =$

$R23\text{-msgs-def } [THEN \text{rel-def-to-dest, simplified, rule-format, THEN subsetD}]$

lemmas $R23\text{-keysI} = R23\text{-keys-def } [THEN \text{rel-def-to-intro, simplified, rule-format}]$

lemmas $R23\text{-keysE } [elim] = R23\text{-keys-def } [THEN \text{rel-def-to-elim, simplified, rule-format}]$

lemmas $R23\text{-keysD} = R23\text{-keys-def } [THEN \text{rel-def-to-dest, simplified, rule-format, rotated 2}]$

lemmas $R23\text{-nonI} = R23\text{-non-def } [THEN \text{rel-def-to-intro, simplified, rule-format}]$

lemmas $R23\text{-nonE } [elim] = R23\text{-non-def } [THEN \text{rel-def-to-elim, simplified, rule-format}]$

lemmas $R23\text{-nonD} = R23\text{-non-def } [THEN \text{rel-def-to-dest, simplified, rule-format, rotated 2}]$

lemmas $R23\text{-presI} = R23\text{-pres-def } [THEN \text{rel-def-to-intro, simplified, rule-format}]$

lemmas $R23\text{-presE } [elim] = R23\text{-pres-def } [THEN \text{rel-def-to-elim, simplified, rule-format}]$

lemmas $R23\text{-intros} = R23\text{-msgsI } R23\text{-keysI } R23\text{-nonI } R23\text{-presI}$

Lemmas for various instantiations (keys and nonces).

lemmas $R23\text{-keys-dests} =$

$R23\text{-keysD}$

$R23\text{-keysD } [\text{where } KK = \{\}, \text{ simplified}]$

$R23\text{-keysD } [\text{where } KK = \{K\} \text{ for } K, \text{ simplified}]$

$R23\text{-keysD } [\text{where } KK = \text{insert } K \text{ KK for } K \text{ KK}, \text{ simplified, OF - - conjI}]$

lemmas $R23\text{-non-dests} =$

$R23\text{-nonD}$

$R23\text{-nonD } [\text{where } KK = \{\}, \text{ simplified}]$

$R23\text{-nonD } [\text{where } KK = \{K\} \text{ for } K, \text{ simplified}]$

$R23\text{-nonD } [\text{where } KK = \text{insert } K \text{ KK for } K \text{ KK}, \text{ simplified, OF - - conjI}]$

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 {} = {}*

by (*auto*)

lemma *abs-msg-Un* [*simp*]:

abs-msg (G ∪ H) = abs-msg G ∪ abs-msg H

by (*auto*)

lemma *abs-msg-mono* [*elim*]:

$\llbracket m \in \text{abs-msg } G; G \subseteq H \rrbracket \implies m \in \text{abs-msg } H$

by (*auto*)

lemma *abs-msg-insert-mono* [*intro*]:

$\llbracket m \in \text{abs-msg } H \rrbracket \implies m \in \text{abs-msg } (\text{insert } m' H)$

by (*auto*)

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*:

$\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; (s, t) \in R23\text{-non}; t \in m3\text{-inv4-lkeysec} \rrbracket$
 $\implies \text{abs-msg } (\text{synth } (\text{analz } (IK t))) \subseteq \text{fake } ik0 \text{ (dom (runs } s)) \text{ (chan } s)$

apply (*auto*)

— 9 subgoals, deal with replays first

prefer 2 **apply** (*blast*)

prefer 3 **apply** (*blast*)

prefer 4 **apply** (*blast*)

prefer 5 **apply** (*blast*)

— remaining 5 subgoals are real fakes

apply (*intro fake-StatCh fake-DynCh, auto dest: R23-dests*)+

done

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*:

$\{R23\}$
 $(m2\text{-step1 } Ra A B Na), (m3\text{-step1 } Ra A B Na)$
 $\{> R23\}$

by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros*)

(auto)

lemma *PO-m3-step2-refines-m2-step2*:

{*R23*}
 (*m2-step2 Rb A B*), (*m3-step2 Rb A B*)
 {> *R23*}

by (auto simp add: *PO-rhoare-defs R23-def m3-defs intro!*: *R23-intros*)

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 –

{ **fix** *s t*

assume *H*:

 (*s, t*) \in *R23-msgs* (*s, t*) \in *R23-keys* (*s, t*) \in *R23-non*

 (*s, t*) \in *R23-pres*

s \in *m2-inv3a-sesK-compr*

t \in *m3-inv7a-sesK-compr* *t* \in *m3-inv4-lkeysec*

Kab = *sesK (Rs\$sk) Rs* \notin *dom (runs t)*

 {*Agent A, Agent B, Nonce Na*} \in *parts (IK t)*

let *?s'* =

s (*runs* := (*runs s*) (*Rs* \mapsto (*Serv, [A, B], [aNon Na, aNum (clk t)]*)),
 chan := *insert (Secure Sv A (Msg [aKey Kab, aAgt B, aNum (clk t), aNon Na]))*
 (*insert (Secure Sv B (Msg [aKey Kab, aAgt A, aNum (clk t)])) (chan s)*) \Downarrow)

let *?t'* =

t (*runs* := (*runs t*) (*Rs* \mapsto (*Serv, [A, B], [aNon Na, aNum (clk t)]*)),
 IK := *insert*
 (*Crypt (shrK A)*
 {*Key Kab, Agent B, Number (clk t), Nonce Na,*
 Crypt (shrK B) {Key Kab, Agent A, Number (clk t)}})
 (*IK t*) \Downarrow)

— here we go

have (*?s', ?t'*) \in *R23-msgs* **using** *H*

by (–) (*rule R23-intros, auto*)

moreover

have (*?s', ?t'*) \in *R23-keys* **using** *H*

by (–) (*rule R23-intros,*

auto simp add: m2-inv3a-sesK-compr-simps m3-inv7a-sesK-compr-simps dest: R23-keys-dests)

moreover

have (*?s', ?t'*) \in *R23-non* **using** *H*

by (–) (*rule R23-intros,*

auto simp add: m2-inv3a-sesK-compr-simps m3-inv7a-sesK-compr-simps dest: R23-non-dests)

moreover

have (*?s', ?t'*) \in *R23-pres* **using** *H*

by (–) (*rule R23-intros, auto*)

moreover

note *calculation*

}

thus *?thesis*

by (*auto simp add: PO-rhoare-defs R23-def m3-defs*)

qed

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 –

{ **fix** *s t*

assume *H*:

(*s, t*) \in *R23-msgs* (*s, t*) \in *R23-keys* (*s, t*) \in *R23-non* (*s, t*) \in *R23-pres*

t \in *m3-inv7a-sesK-compr* *t* \in *m3-inv7b-sesK-compr-non*

t \in *m3-inv6-ticket* *t* \in *m3-inv4-lkeysec*

runs t Ra = *Some (Init, [A, B], [])*

Na = *Ra\$na*

Crypt (shrK A) {Key Kab, Agent B, Number Ts, Nonce Na, X} \in *analz (IK t)*

let *?s' = s* | *runs := (runs s)(Ra \mapsto (Init, [A, B], [aKey Kab, aNum Ts, aNum (clk t)]))*,

chan := insert (dAuth Kab (Msg [aAgt A, aNum (clk t)])) (chan s) |

and *?t' = t* | *runs := (runs t)(Ra \mapsto (Init, [A, B], [aKey Kab, aNum Ts, aNum (clk t)]))*,

IK := insert { Crypt Kab {Agent A, Number (clk t)}, X } (IK t) |

from *H* **have**

Secure Sv A (Msg [aKey Kab, aAgt B, aNum Ts, aNon Na]) \in *chan s*

by (*auto*)

moreover

have *X* \in *parts (IK t)* **using** *H* **by** (*auto dest!: Body MPair-parts*)

hence (*?s', ?t'*) \in *R23-msgs* **using** *H* **by** (*auto intro!: R23-intros*) (*auto*)

moreover

have (*?s', ?t'*) \in *R23-keys*

proof (*cases*)

assume *A* \in *bad*

with *H* **have** *X* \in *analz (IK t)* **by** ($-$) (*drule Decrypt, auto*)

with *H* **show** *?thesis*

by ($-$) (*rule R23-intros, auto dest!: analz-cut intro: analz-monotonic*)

next

assume *A* \notin *bad* **show** *?thesis*

proof –

note *H*

moreover

with $\langle A \notin \text{bad} \rangle$

have *X* = *Crypt (shrK B) {Key Kab, Agent A, Number Ts}* | \wedge *Kab* \in *range sesK*

by (*auto dest!: m3-inv6-ticketD*)

moreover

{ **assume** *H1*: *Key (shrK B)* \in *analz (IK t)*

have *aKey Kab* \in *extr ik0 (chan s)*

proof –

note *calculation*

moreover

hence *Secure Sv B (Msg [aKey Kab, aAgt A, aNum Ts])* \in *chan s*

by ($-$) (*drule analz-into-parts, drule Body, elim MPair-parts, auto*)

ultimately

show *?thesis* **using** *H1* **by** *auto*

qed

}

ultimately show *?thesis*

by ($-$) (*rule R23-intros, auto simp add: m3-inv7a-sesK-compr-simps*)

```

    qed
  qed
  moreover
  have (?s', ?t') ∈ R23-non
  proof (cases)
    assume A ∈ bad
    hence X ∈ analz (IK t) using H by (-) (drule Decrypt, auto)
    thus ?thesis using H
      by (-) (rule R23-intros, auto dest!: analz-cut intro: analz-monotonic)
  next
  assume A ∉ bad
  hence X = Crypt (shrK B) {Key Kab, Agent A, Number Ts} ∧ Kab ∈ range sesK using H
    by (auto dest!: m3-inv6-ticketD)
  thus ?thesis using H
    by (-) (rule R23-intros,
      auto simp add: m3-inv7a-sesK-compr-simps m3-inv7b-sesK-compr-non-simps)
  qed
  moreover
  have (?s', ?t') ∈ R23-pres using H
  by (auto intro!: R23-intros)
  moreover
  note calculation
}
thus ?thesis
by (auto simp add: PO-rhoare-defs R23-def m3-defs dest!: analz-Inj-IK)
qed

```

```

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}
by (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros)
(auto)

```

```

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}
by (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros)

```

```

lemma PO-m3-tick-refines-m2-tick:
  {R23}
  (m2-tick T), (m3-tick T)
  {>R23}
by (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros)

```

```

lemma PO-m3-purge-refines-m2-purge:
  {R23}
  (m2-purge A), (m3-purge A)
  {>R23}
by (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros)

```

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*}
by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros*)
 (*auto dest: R23-dests*)

lemma *PO-m3-DY-fake-refines-m2-fake*:
 {*R23* \cap *UNIV* \times (*m3-inv4-lkeysec*)}
m2-fake, *m3-DY-fake*
 {> *R23*}
apply (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros*)
 (*del: abs-msg.cases*)
apply (*auto intro: abs-msg-DY-subset-fakeable [THEN subsetD]*)
 (*del: abs-msg.cases*)
apply (*auto dest: R23-dests*)
done

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*“(*init m2*)
by (*auto simp add: R23-def m3-defs intro!: R23-intros*)

lemma *PO-m3-refines-trans-m2* [*iff*]:
 {*R23* \cap (*m2-inv3a-sesK-compr*)
 \times (*m3-inv7a-sesK-compr* \cap *m3-inv7b-sesK-compr-non* \cap *m3-inv6-ticket* \cap *m3-inv4-lkeysec*)}
 (*trans m2*), (*trans m3*)
 {> *R23*}
by (*auto simp add: m3-def m3-trans-def m2-def m2-trans-def*)
 (*blast intro!: PO-m3-trans-refines-m2-trans*)+

lemma *PO-m3-observation-consistent* [*iff*]:
obs-consistent R23 med32 m2 m3
by (*auto simp add: obs-consistent-def R23-def med32-def m3-defs*)

Refinement result.

lemma *m3-refines-m2* [*iff*]:
refines
 (*R23* \cap
 (*m2-inv3a-sesK-compr*) \times
 (*m3-inv7a-sesK-compr* \cap *m3-inv7b-sesK-compr-non* \cap *m3-inv6-ticket* \cap *m3-inv4-lkeysec*)
med32 m2 m3)
by (*rule Refinement-using-invariants*) (*auto*)

lemma *m3-implements-m2* [iff]:
implements med32 m2 m3
by (rule refinement-soundness) (auto)

3.8.7 Inherited invariants

inv3 (derived): Key secrecy for initiator

definition

m3-inv3-ikk-init :: *m3-state set*

where

m3-inv3-ikk-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-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 \subseteq m3-inv3-ikk-init*

proof (rule INV-from-Refinement-using-invariants [OF *m3-refines-m2*])

show *Range (R23 \cap*
m2-inv3a-sesK-compr
 \times (*m3-inv7a-sesK-compr \cap m3-inv7b-sesK-compr-non \cap m3-inv6-ticket \cap m3-inv4-lkeysec*)
 \cap *m2-inv6-ikk-init \times UNIV*)
 \subseteq *m3-inv3-ikk-init*
by (auto simp add: *R23-def R23-pres-def intro!*: *m3-inv3-ikk-initI*)
(elim *m2-inv6-ikk-initE*, auto dest: *R23-keys-dests*)

qed auto

inv4 (derived): Key secrecy for responder

definition

m3-inv4-ikk-resp :: *m3-state set*

where

m3-inv4-ikk-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-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 \subseteq m3-inv4-ikk-resp*

proof (rule INV-from-Refinement-using-invariants [OF *m3-refines-m2*])

show *Range (R23 \cap m2-inv3a-sesK-compr*
 \times (*m3-inv7a-sesK-compr \cap m3-inv7b-sesK-compr-non \cap m3-inv6-ticket \cap*
m3-inv4-lkeysec)
 \cap *m2-inv7-ikk-resp \times UNIV*)

```

    ⊆ m3-inv4-ikk-resp
  by (auto simp add: R23-def R23-pres-def intro!: m3-inv4-ikk-respI)
    (elim m2-inv7-ikk-respE, auto dest: R23-keys-dests)
qed auto

```

end

3.9 Abstract Needham-Schroeder Shared Key (L1)

```

theory m1-nssk imports m1-keydist-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-state = m1r-state +
  leak :: (key × fresh-t × fresh-t) set — keys leaked plus session context

```

```

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

```

3.9.2 Events

definition — by A , refines $m1a$ -step1

```

  m1-step1 :: [rid-t, agent, agent, nonce] ⇒ 'x m1r-trans

```

where

```

  m1-step1 Ra A B Na ≡ m1a-step1 Ra A B Na

```

definition — by B , refines $m1a$ -step2

```

  m1-step2 :: [rid-t, agent, agent] ⇒ 'x m1r-trans

```

where

$m1\text{-step2 } Rb \ A \ B \equiv m1a\text{-step2 } Rb \ A \ B$

definition — by Sv , refines $m1a\text{-step3}$

$m1\text{-step3} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow 'x \ m1r\text{-trans}$

where

$m1\text{-step3 } Rs \ A \ B \ Na \ Kab \equiv m1a\text{-step3 } Rs \ A \ B \ Kab \ Na \ []$

definition — by A , refines $m1a\text{-step4}$

$m1\text{-step4} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow 'x \ m1\text{-trans}$

where

$m1\text{-step4 } Ra \ A \ B \ Na \ Kab \equiv \{(s, s')\}.$

— guards:

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], []) \wedge$

$Na = Ra\$na \wedge$

— fix parameter

$(Kab \notin \text{Domain } (\text{leak } s) \longrightarrow (Kab, A) \in \text{azC } (\text{runs } s)) \wedge$ — authorization guard

— new guard for agreement with server on (Kab, B, Na) ,

— injectiveness by including Na

$(A \notin \text{bad} \longrightarrow (\exists Rs. Kab = \text{sesK } (Rs\$sk) \wedge$

$\text{runs } s \ Rs = \text{Some } (\text{Serv}, [A, B], [a\text{Non } Na]))) \wedge$

— actions:

$s' = s(\ \text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [a\text{Key } Kab])) \)$

}

definition — by B , refines $m1a\text{-step5}$

$m1\text{-step5} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow 'x \ m1\text{-trans}$

where

$m1\text{-step5 } Rb \ A \ B \ Nb \ Kab \equiv \{(s, s')\}.$

— new guards:

$Nb = Rb\$nb \wedge$

— generate Nb

— prev guards:

$\text{runs } s \ Rb = \text{Some } (\text{Resp}, [A, B], []) \wedge$

$(Kab \notin \text{Domain } (\text{leak } s) \longrightarrow (Kab, B) \in \text{azC } (\text{runs } s)) \wedge$ — authorization guard

— guard for showing agreement with server on (Kab, A) ,

— this agreement is non-injective

$(B \notin \text{bad} \longrightarrow (\exists Rs \ Na. Kab = \text{sesK } (Rs\$sk) \wedge$

$\text{runs } s \ Rs = \text{Some } (\text{Serv}, [A, B], [a\text{Non } Na]))) \wedge$

— actions:

$s' = s(\ \text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [a\text{Key } Kab])) \)$

}

definition — by A , refines skip

$m1\text{-step6} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{nonce}, \text{key}] \Rightarrow 'x \ m1\text{-trans}$

where

$m1\text{-step6 } Ra \ A \ B \ Na \ Nb \ Kab \equiv \{(s, s')\}.$

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], [a\text{Key } Kab]) \wedge$ — key recv'd before

$Na = Ra\$na \wedge$

— guard for showing agreement with B on Kab and Nb

$(A \notin \text{bad} \longrightarrow B \notin \text{bad} \longrightarrow$
 $(\forall Nb'. (Kab, Na, Nb') \notin \text{leak } s) \longrightarrow$ — NEW: weaker condition
 $(\exists Rb \text{ nl}. Nb = Rb\$nb \wedge \text{runs } s \text{ Rb} = \text{Some } (\text{Resp}, [A, B], \text{aKey } Kab \# \text{nl}))) \wedge$

— actions:

$s' = s \langle$
 $\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [\text{aKey } Kab, \text{aNon } Nb]))$
 \rangle
 $\}$

definition — by B , refines skip

$m1\text{-step}\gamma :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow 'x \text{ m1-trans}$

where

$m1\text{-step}\gamma \text{ Rb } A \text{ B } Nb \text{ Kab} \equiv \{(s, s')\}.$

$\text{runs } s \text{ Rb} = \text{Some } (\text{Resp}, [A, B], [\text{aKey } Kab]) \wedge$ — key recv'd before
 $Nb = Rb\$nb \wedge$

— guard for showing agreement with A on Kab and Nb

$(A \notin \text{bad} \longrightarrow B \notin \text{bad} \longrightarrow Kab \notin \text{Domain } (\text{leak } s) \longrightarrow$
 $\text{— } (\forall Na'. (Kab, Na', Nb) \notin \text{leak } s) \longrightarrow$ too strong, does not work
 $(\exists Ra. \text{runs } s \text{ Ra} = \text{Some } (\text{Init}, [A, B], [\text{aKey } Kab, \text{aNon } Nb]))) \wedge$

— actions: (redundant) update local state marks successful termination

$s' = s \langle$
 $\text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [\text{aKey } Kab, \text{END}]))$
 \rangle
 $\}$

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

$m1\text{-leak } Rs \text{ Ra } Rb \text{ A } B \equiv \{(s, s1)\}.$

— guards:

$\text{runs } s \text{ Rs} = \text{Some } (\text{Serv}, [A, B], [\text{aNon } (Ra\$na)]) \wedge$
 $\text{runs } s \text{ Ra} = \text{Some } (\text{Init}, [A, B], [\text{aKey } (\text{sesK } (Rs\$sk)), \text{aNon } (Rb\$nb)]) \wedge$
 $\text{runs } s \text{ Rb} = \text{Some } (\text{Resp}, [A, B], [\text{aKey } (\text{sesK } (Rs\$sk)), \text{END}]) \wedge$

— 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

$m1\text{-init} \equiv \{ \langle$
 $\text{runs} = \text{Map.empty},$
 $\text{leak} = \text{corrKey} \times \{\text{undefined}\} \times \{\text{undefined}\}$ — initial leakage
 $\rangle \}$

definition

$m1\text{-trans} :: 'x \text{ m1-trans}$ **where**

$$\begin{aligned}
m1-trans &\equiv (\bigcup A B Ra Rb Rs Na Nb Kab. \\
& m1-step1 Ra A B Na \cup \\
& m1-step2 Rb A B \cup \\
& m1-step3 Rs A B Na Kab \cup \\
& m1-step4 Ra A B Na Kab \cup \\
& m1-step5 Rb A B Nb Kab \cup \\
& m1-step6 Ra A B Na Nb Kab \cup \\
& m1-step7 Rb A B Nb Kab \cup \\
& m1-leak Rs Ra Rb A B \cup \\
& Id \\
&)
\end{aligned}$$

definition

$m1 :: (m1-state, m1-obs)$ spec **where**
 $m1 \equiv ()$
 $init = m1-init,$
 $trans = m1-trans,$
 $obs = id$
 $\})$

lemmas $m1-loc-defs =$

$m1-def m1-trans-def$
 $m1-step1-def m1-step2-def m1-step3-def m1-step4-def m1-step5-def$
 $m1-step6-def m1-step7-def m1-leak-def$

lemmas $m1-defs = m1-loc-defs m1a-defs$

lemma $m1-obs-id$ [simp]: $obs\ m1 = id$
by (simp add: m1-def)

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]:
 $init\ m1 \subseteq m1-inv0-fin$
by (auto simp add: m1-defs intro!: m1-inv0-finI)

lemma $PO-m1-inv0-fin-trans$ [iff]:
 $\{m1-inv0-fin\}\ trans\ m1 \{>\ m1-inv0-fin\}$

by (*auto simp add: PO-hoare-defs m1-defs intro!: m1-inv0-finI*)

lemma *PO-m1-inv0-fin [iff]: reach m1 \subseteq m1-inv0-fin*

by (*rule inv-rule-incr, auto del: subsetI*)

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) — take Kab

| *rm1a1 Resp = take (Suc rs-len) — take Kab*

| *rm1a1 Serv = id — 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

apply (*auto simp add: map-runs-def elim!: knC.cases*)

apply (*rename-tac b, case-tac b, auto*)

apply (*rename-tac b, case-tac b, auto*)

apply (*rule knC-init, auto simp add: runs1a1-def*)

apply (*rule knC-resp, auto simp add: runs1a1-def*)

apply (*rule-tac knC-serv, auto simp add: runs1a1-def*)

done

R1a1: The simulation relation is defined in terms of the mediator function.

definition

med1a1 :: m1-obs \Rightarrow m1a-obs **where**

med1a1 s \equiv (λ runs = runs1a1 (runs s), m1x-state.leak = Domain (leak s) \Downarrow)

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*}

by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs*)

lemma *PO-m1-step2-refines-m1a-step2*:

{*R1a1*}
(*m1a-step2 Rb A B*), (*m1-step2 Rb A B*)
{> *R1a1*}

by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs*)

lemma *PO-m1-step3-refines-m1a-step3*:

{*R1a1*}
(*m1a-step3 Rs A B Kab Na []*), (*m1-step3 Rs A B Na Kab*)
{> *R1a1*}

by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs*)

lemma *PO-m1-step4-refines-m1a-step4*:

{*R1a1*}
(*m1a-step4 Ra A B Na Kab []*), (*m1-step4 Ra A B Na Kab*)
{> *R1a1*}

by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs runs1a1-def*)

lemma *PO-m1-step5-refines-m1a-step5*:

{*R1a1*}
(*m1a-step5 Rb A B Kab []*), (*m1-step5 Rb A B Nb Kab*)
{> *R1a1*}

by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs runs1a1-def*)

lemma *PO-m1-step6-refines-m1a-skip*:

{*R1a1*}
Id, (*m1-step6 Ra A B Na Nb Kab*)
{> *R1a1*}

by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs runs1a1-def*)

lemma *PO-m1-step7-refines-m1a-skip*:

{*R1a1*}
Id, (*m1-step7 Rb A B Nb Kab*)
{> *R1a1*}

by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs runs1a1-def*)

lemma *PO-m1-leak-refines-m1a-leak*:

{*R1a1*}
(*m1a-leak Rs*), (*m1-leak Rs Ra Rb A B*)
{> *R1a1*}

by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs map-runs-def dest: dom-lemmas*)

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-step7-refines-m1a-skip PO-m1-leak-refines-m1a-leak

lemma *PO-m1-refines-init-m1a* [iff]:
init m1 \subseteq *R1a1*“(init m1a)
by (auto simp add: *R1a1-defs m1a-defs m1-loc-defs*)

lemma *PO-m1-refines-trans-m1a* [iff]:
{*R1a1*}
(*trans m1a*), (*trans m1*)
{> *R1a1*}
apply (auto simp add: *m1-def m1-trans-def m1a-def m1a-trans-def*
intro!: *PO-m1-trans-refines-m1a-trans*)
apply (*force intro!*: *PO-m1-trans-refines-m1a-trans*)
done

Observation consistency.

lemma *obs-consistent-med1a1* [iff]:
obs-consistent R1a1 med1a1 m1a m1
by (auto simp add: *obs-consistent-def R1a1-def med1a1-def m1a-def m1-def*)

Refinement result.

lemma *PO-m1-refines-m1a* [iff]:
refines R1a1 med1a1 m1a m1
by (rule *Refinement-basic*) (auto del: *subsetI*)

lemma *m1-implements-m1a* [iff]: *implements med1a1 m1a m1*
by (rule *refinement-soundness*) (*fast*)

inv (inherited): Key secrecy

Secrecy, as external and internal invariant

definition

m1-secrecy :: '*x m1-pred* **where**
m1-secrecy \equiv {*s. knC (runs s) \subseteq azC (runs s) \cup Domain (leak s) \times 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 \subseteq m1-secrecy*
apply (rule-tac *Q=m1x-secrecy* **in** *external-invariant-translation*)
apply (auto del: *subsetI*)
apply (*fastforce simp add: med1a1-def intro!*: *m1-secrecyI*)
done

lemma *PO-m1-secrecy* [iff]: *reach m1 \subseteq m1-secrecy*
by (rule *external-to-internal-invariant*) (auto del: *subsetI*)

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

$m1\text{-inv}2i\text{-serv} \equiv \{s. \forall A\ B\ Ra\ Na\ Kab\ nla.$

$A \notin \text{bad} \longrightarrow$

$\text{runs } s\ Ra = \text{Some } (\text{Init}, [A, B], a\text{Key } Kab \# nla) \longrightarrow$

$Na = Ra\$na \longrightarrow$

$(\exists Rs. Kab = \text{ses}K (Rs\$sk) \wedge \text{runs } s\ Rs = \text{Some } (\text{Serv}, [A, B], [a\text{Non } Na]))$

$\}$

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 $m1\text{-inv}2i\text{-serv}D = m1\text{-inv}2i\text{-serv}\text{-def } [THEN\ \text{setc}\text{-def}\text{-to}\text{-dest},\ \text{rule}\text{-format},\ \text{rotated } 2]$

Proof of invariance.

lemma $PO\text{-}m1\text{-inv}2i\text{-serv } [iff]:\ \text{reach } m1 \subseteq m1\text{-inv}2i\text{-serv}$

apply ($\text{rule}\text{-tac } Pa=m1a\text{-inv}2i\text{-serv}$ **and** $Qa=m1a\text{-inv}2i\text{-serv}$ **and** $Q=m1\text{-inv}2i\text{-serv}$
in $\text{internal}\text{-invariant}\text{-translation}$)

apply (auto del: subsetI)

apply ($\text{auto simp add: } m1a\text{-inv}2i\text{-serv}\text{-def } med1a1\text{-def } vimage\text{-def}$
 $\text{intro!: } m1\text{-inv}2i\text{-serv}I$)

apply ($\text{rename}\text{-tac } s\ A\ B\ Ra\ Kab\ nla$)

apply ($\text{drule}\text{-tac } x=A$ **in** $\text{spec},\ \text{clarsimp}$)

apply ($\text{drule}\text{-tac } x=B$ **in** spec)

apply ($\text{drule}\text{-tac } x=Ra$ **in** spec)

apply ($\text{drule}\text{-tac } x=Kab$ **in** spec)

apply ($\text{clarsimp simp add: runs}1a1\text{-def}$)

done

declare $PO\text{-}m1\text{-inv}2i\text{-serv } [THEN\ \text{subset}D,\ \text{intro}]$

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

$m1\text{-inv}2r\text{-serv} \equiv \{s. \forall A\ B\ Rb\ Kab\ nlb.$

$B \notin \text{bad} \longrightarrow$

$\text{runs } s\ Rb = \text{Some } (\text{Resp}, [A, B], a\text{Key } Kab \# nlb) \longrightarrow$

$(\exists Rs\ Na. Kab = \text{ses}K (Rs\$sk) \wedge \text{runs } s\ Rs = \text{Some } (\text{Serv}, [A, B], [a\text{Non } Na]))$

$\}$

lemmas $m1\text{-inv}2r\text{-serv}I = m1\text{-inv}2r\text{-serv}\text{-def } [THEN\ \text{setc}\text{-def}\text{-to}\text{-intro},\ \text{rule}\text{-format}]$

lemmas $m1\text{-inv}2r\text{-serv}E [elim] = m1\text{-inv}2r\text{-serv}\text{-def } [THEN\ \text{setc}\text{-def}\text{-to}\text{-elim},\ \text{rule}\text{-format}]$

lemmas $m1\text{-inv}2r\text{-serv}D = m1\text{-inv}2r\text{-serv}\text{-def } [THEN\ \text{setc}\text{-def}\text{-to}\text{-dest},\ \text{rule}\text{-format},\ \text{rotated } -1]$

Proof of invariance.

lemma $PO\text{-}m1\text{-inv}2r\text{-serv } [iff]:\ \text{reach } m1 \subseteq m1\text{-inv}2r\text{-serv}$

```

apply (rule-tac Pa=m1a-inv2r-serv and Qa=m1a-inv2r-serv and Q=m1-inv2r-serv
  in internal-invariant-translation)
apply (auto del: subsetI)
apply (auto simp add: simp add: m1a-inv2r-serv-def med1a1-def vimage-def
  intro!: m1-inv2r-servI)
apply (rename-tac s A B Rb Kab nlb)
apply (drule-tac x=A in spec)
apply (drule-tac x=B in spec, clarsimp)
apply (drule-tac x=Rb in spec)
apply (drule-tac x=Kab in spec)
apply (clarsimp simp add: runs1a1-def)
done

```

```

declare PO-m1-inv2r-serv [THEN subsetD, intro]

```

inv (inherited): Initiator key freshness

definition

$m1\text{-inv3-ifresh} :: 'x\ m1\text{-pred}$

where

$$\begin{aligned}
m1\text{-inv3-ifresh} &\equiv \{s. \forall A\ A'\ B\ B'\ Ra\ Ra'\ Kab\ nl\ nl'. \\
&\quad runs\ s\ Ra = Some\ (Init, [A, B], aKey\ Kab\ \#\ nl) \longrightarrow \\
&\quad runs\ s\ Ra' = Some\ (Init, [A', B'], aKey\ Kab\ \#\ nl') \longrightarrow \\
&\quad A \notin bad \longrightarrow B \notin bad \longrightarrow Kab \notin Domain\ (leak\ s) \longrightarrow \\
&\quad Ra = Ra' \\
&\}
\end{aligned}$$

lemmas $m1\text{-inv3-ifreshI} = m1\text{-inv3-ifresh-def}$ [THEN setc-def-to-intro, rule-format]

lemmas $m1\text{-inv3-ifreshE}$ [elim] = $m1\text{-inv3-ifresh-def}$ [THEN setc-def-to-elim, rule-format]

lemmas $m1\text{-inv3-ifreshD} = m1\text{-inv3-ifresh-def}$ [THEN setc-def-to-dest, rule-format, rotated 1]

lemma $PO\text{-}m1\text{-inv3-ifresh}$ [iff]: $reach\ m1 \subseteq m1\text{-inv3-ifresh}$

```

apply (rule-tac Pa=m1a-inv1-ifresh and Qa=m1a-inv1-ifresh and Q=m1-inv3-ifresh
  in internal-invariant-translation)

```

```

apply (auto del: subsetI)

```

```

apply (auto simp add: med1a1-def runs1a1-def vimage-def m1-inv3-ifresh-def)

```

```

done

```

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-runs2sigs :: runs-t ⇒ irsig signal ⇒ nat

```

where

```

  ir-runs2sigs runz (Commit [A, B] (Kab, Nb)) =
    card (ir-commit runz A B Kab Nb)

```

```

| ir-runs2sigs runz (Running [A, B] (Kab, Nb)) =
  (if ∃ Rb nl. Nb = Rb$nb ∧ runz Rb = Some (Resp, [A, B], aKey Kab # nl)
   then 1 else 0)

```

```

| ir-runs2sigs runz - = 0

```

Simulation relation and mediator function. We map completed initiator and responder runs to commit and running signals, respectively.

definition

```

  med-a0im1-ir :: m1-obs ⇒ irsig a0i-obs where
  med-a0im1-ir o1 ≡ (| signals = ir-runs2sigs (runs o1), corrupted = Domain (leak o1) × UNIV |)

```

definition

```

  R-a0im1-ir :: (irsig a0i-state × m1-state) set where
  R-a0im1-ir ≡ {(s, t). signals s = ir-runs2sigs (runs t) ∧ corrupted s = Domain (leak t) × 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-runs2sigs-empty* [simp]:

```

  runz = Map.empty ⇒ ir-runs2sigs runz = (λs. 0)

```

by (rule ext, erule rev-mp)

```

  (rule ir-runs2sigs.induct, auto)

```

lemma *finite-ir-commit* [simp, intro!]:

```

  finite (dom runz) ⇒ finite (ir-commit runz A B Kab Nb)

```

by (auto intro: finite-subset dest: dom-lemmas)

Update lemmas

lemma *ir-runs2sigs-upd-init-none* [simp]:

```

  [| Ra ∉ dom runz |]
  ⇒ ir-runs2sigs (runz(Ra ↦ (Init, [A, B], []))) = ir-runs2sigs runz

```

by (rule ext, erule rev-mp)

```

  (rule ir-runs2sigs.induct, auto dest: dom-lemmas)

```

lemma *ir-runs2sigs-upd-resp-none* [simp]:

```

  [| Rb ∉ dom runz |]
  ⇒ ir-runs2sigs (runz(Rb ↦ (Resp, [A, B], []))) = ir-runs2sigs runz

```

by (rule ext, erule rev-mp)

```

  (rule ir-runs2sigs.induct, auto dest: dom-lemmas)

```

lemma *ir-runs2sigs-upd-serv-none* [simp]:

$\llbracket Rs \notin \text{dom runz} \rrbracket$
 $\implies \text{ir-runs2sigs} (\text{runz}(Rs \mapsto (\text{Serv}, [A, B], nl))) = \text{ir-runs2sigs runz}$
by (rule ext, erule rev-mp)
(rule ir-runs2sigs.induct, auto dest: dom-lemmas)

lemma *ir-runs2sigs-upd-init-some* [simp]:
 $\llbracket \text{runz Ra} = \text{Some} (\text{Init}, [A, B], []) \rrbracket$
 $\implies \text{ir-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [aKey Kab]))) = \text{ir-runs2sigs runz}$
by (rule ext, erule rev-mp)
(rule ir-runs2sigs.induct, auto)

lemma *ir-runs2sigs-upd-resp* [simp]:
 $\llbracket \text{runz Rb} = \text{Some} (\text{Resp}, [A, B], []) \rrbracket$
 $\implies \text{ir-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], [aKey Kab]))) =$
 $(\text{ir-runs2sigs runz})(\text{Running } [A, B] (Kab, Rb\$nb) := 1)$
apply (rule ext, erule rev-mp)
apply (rule ir-runs2sigs.induct, fastforce+)
done

lemma *ir-runs2sigs-upd-init* [simp]:
 $\llbracket \text{runz Ra} = \text{Some} (\text{Init}, [A, B], [aKey Kab]); \text{finite} (\text{dom runz}) \rrbracket$
 $\implies \text{ir-runs2sigs} (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [aKey Kab, aNon Nb]))) =$
 $(\text{ir-runs2sigs runz})$
 $(\text{Commit } [A, B] (Kab, Nb) := \text{Suc} (\text{card} (\text{ir-commit runz } A \ B \ Kab \ Nb)))$
apply (rule ext, erule rev-mp, erule rev-mp)
apply (rule-tac ?a0.0=runz in ir-runs2sigs.induct, auto)
— 1 subgoal, solved using $\llbracket \text{finite } ?A; ?x \notin ?A \rrbracket \implies \text{card} (\text{insert } ?x \ ?A) = \text{Suc} (\text{card } ?A)$
apply (rename-tac runz)
apply (rule-tac
 $s = \text{card} (\text{insert } Ra \ (\text{ir-commit runz } A \ B \ Kab \ Nb))$
in trans, fast, auto)
done

lemma *ir-runs2sigs-upd-resp-some* [simp]:
 $\llbracket \text{runz Rb} = \text{Some} (\text{Resp}, [A, B], [aKey K]) \rrbracket$
 $\implies \text{ir-runs2sigs} (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], [aKey K, END]))) = \text{ir-runs2sigs runz}$
by (rule ext, erule rev-mp)
(rule ir-runs2sigs.induct, fastforce+)

Needed for injectiveness of agreement.

lemma *m1-inv2i-serv-lemma*:
 $\llbracket \text{runs } t \ Ra = \text{Some} (\text{Init}, [A, B], [aKey Kab, aNon Nb]);$
 $\text{runs } t \ Ra' = \text{Some} (\text{Init}, [A, B], [aKey Kab]);$
 $A \notin \text{bad}; t \in \text{m1-inv2i-serv} \rrbracket$
 $\implies P$
apply (frule m1-inv2i-servD, auto)
apply (rotate-tac 1)
apply (frule m1-inv2i-servD, auto)
done

Refinement proof

lemma *PO-m1-step1-refines-ir-a0i-skip*:

$\{R\text{-a0im1-ir}\}$
 $Id, (m1\text{-step1 } Ra \ A \ B \ Na)$
 $\{> R\text{-a0im1-ir}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ir-defs m1-defs, safe, auto*)

lemma *PO-m1-step2-refines-ir-a0i-skip:*
 $\{R\text{-a0im1-ir}\}$
 $Id, (m1\text{-step2 } Rb \ A \ B)$
 $\{> R\text{-a0im1-ir}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ir-defs m1-defs, safe, auto*)

lemma *PO-m1-step3-refines-ir-a0i-skip:*
 $\{R\text{-a0im1-ir}\}$
 $Id, (m1\text{-step3 } Rs \ A \ B \ Na \ Kab)$
 $\{> R\text{-a0im1-ir}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ir-defs m1-defs, safe, auto*)

lemma *PO-m1-step4-refines-ir-a0i-skip:*
 $\{R\text{-a0im1-ir}\}$
 $Id, (m1\text{-step4 } Ra \ A \ B \ Na \ Kab)$
 $\{> R\text{-a0im1-ir}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ir-defs a0i-defs m1-defs, safe, auto*)

lemma *PO-m1-step5-refines-ir-a0i-running:*
 $\{R\text{-a0im1-ir}\}$
 $(a0i\text{-running } [A, B] \ (Kab, Nb)), (m1\text{-step5 } Rb \ A \ B \ Nb \ Kab)$
 $\{> R\text{-a0im1-ir}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ir-defs a0i-defs m1-defs, safe, auto*)

lemma *PO-m1-step6-refines-ir-a0i-commit:*
 $\{R\text{-a0im1-ir} \cap UNIV \times (m1\text{-inv2i-serv} \cap m1\text{-inv0-fin})\}$
 $(a0i\text{-commit } [A, B] \ (Kab, Nb)), (m1\text{-step6 } Ra \ A \ B \ Na \ Nb \ Kab)$
 $\{> R\text{-a0im1-ir}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ir-defs a0i-defs m1-defs, safe, auto*)
(auto dest: m1-inv2i-serv-lemma)

lemma *PO-m1-step7-refines-ir-a0i-skip:*
 $\{R\text{-a0im1-ir}\}$
 $Id, (m1\text{-step7 } Rb \ A \ B \ Nb \ Kab)$
 $\{> R\text{-a0im1-ir}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ir-defs a0i-defs m1-defs, safe, auto*)

lemma *PO-m1-leak-refines-ir-a0i-corrupt:*
 $\{R\text{-a0im1-ir}\}$
 $(a0i\text{-corrupt } (\{sesK \ (Rs\$sk)\} \times UNIV)), (m1\text{-leak } Rs \ Ra \ Rb \ A \ B)$
 $\{> R\text{-a0im1-ir}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ir-defs a0i-defs m1-defs, safe, auto*)

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]:
 $init\ m1 \subseteq R\text{-}a0im1\text{-}ir\text{''}(init\ a0i)$
by (*auto simp add: R-a0im1-ir-defs a0i-defs m1-defs*
intro!: exI [where x=(signals = $\lambda s. 0$, corrupted = corrKey \times UNIV)])

lemma *PO-m1-refines-ir-trans-a0i* [iff]:
 $\{R\text{-}a0im1\text{-}ir \cap reach\ a0i \times reach\ m1\}$
 $(trans\ a0i), (trans\ m1)$
 $\{> R\text{-}a0im1\text{-}ir\}$
apply (*rule-tac pre'=R-a0im1-ir \cap UNIV \times (m1-inv2i-serv \cap m1-inv0-fin)*
in relhoare-conseq-left, auto)
apply (*auto simp add: m1-def m1-trans-def a0i-def a0i-trans-def*
intro!: PO-m1-trans-refines-ir-a0i-trans)
done

Observation consistency.

lemma *obs-consistent-med-a0im1-ir* [iff]:
 $obs\ consistent\ R\text{-}a0im1\text{-}ir\ med\text{-}a0im1\text{-}ir\ a0i\ m1$
by (*auto simp add: obs-consistent-def R-a0im1-ir-def med-a0im1-ir-def*
a0i-def m1-def)

Refinement result.

lemma *PO-m1-refines-ir-a0i* [iff]:
refines
 $(R\text{-}a0im1\text{-}ir \cap reach\ a0i \times reach\ m1)$
 $med\text{-}a0im1\text{-}ir\ a0i\ m1$
by (*rule Refinement-using-invariants*) (*auto*)

lemma *m1-implements-ir-a0i: implements med-a0im1-ir a0i m1*
by (*rule refinement-soundness*) (*fast*)

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
 $ri\text{-}running\ runz\ A\ B\ Kab\ Nb \equiv \{Ra.$
 $runz\ Ra = Some\ (Init, [A, B], [aKey\ Kab, aNon\ Nb])$
 $\}$

fun
 $ri\text{-}runs2sigs :: runs\text{-}t \Rightarrow risig\ signal \Rightarrow nat$

where

$ri\text{-runs2sigs}\ runz\ (Commit\ [B,\ A]\ (Kab,\ Nb)) =$
 $(if\ \exists\ Rb.\ Nb = Rb\$nb \wedge\ runz\ Rb = Some\ (Resp,\ [A,\ B],\ [aKey\ Kab,\ END])$
 $then\ 1\ else\ 0)$

$| ri\text{-runs2sigs}\ runz\ (Running\ [B,\ A]\ (Kab,\ Nb)) =$
 $card\ (ri\text{-running}\ runz\ A\ B\ Kab\ Nb)$

$| ri\text{-runs2sigs}\ runz\ - = 0$

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 $ri\text{-runs2sigs}\text{-}empty$ $[simp]$:

$runz = Map.empty \Longrightarrow ri\text{-runs2sigs}\ runz = (\lambda s.\ 0)$

by $(rule\ ext,\ erule\ rev\text{-}mp)$

$(rule\ ri\text{-runs2sigs}.induct,\ auto)$

lemma $finite\text{-}inv\text{-}ri\text{-}running$ $[simp,\ intro!]$:

$finite\ (dom\ runz) \Longrightarrow finite\ (ri\text{-running}\ runz\ A\ B\ Kab\ Nb)$

by $(auto\ intro:\ finite\text{-}subset\ dest:\ dom\text{-}lemmas)$

Update lemmas

lemma $ri\text{-runs2sigs}\text{-}upd\text{-}init\text{-}none$ $[simp]$:

$\llbracket Na \notin dom\ runz \rrbracket$

$\Longrightarrow ri\text{-runs2sigs}\ (runz(Na \mapsto (Init,\ [A,\ B],\ []))) = ri\text{-runs2sigs}\ runz$

by $(rule\ ext,\ erule\ rev\text{-}mp)$

$(rule\ ri\text{-runs2sigs}.induct,\ auto\ dest:\ dom\text{-}lemmas)$

lemma $ri\text{-runs2sigs}\text{-}upd\text{-}resp\text{-}none$ $[simp]$:

$\llbracket Rb \notin dom\ runz \rrbracket$

$\Longrightarrow ri\text{-runs2sigs}\ (runz(Rb \mapsto (Resp,\ [A,\ B],\ []))) = ri\text{-runs2sigs}\ runz$

by $(rule\ ext,\ erule\ rev\text{-}mp)$

$(rule\ ri\text{-runs2sigs}.induct,\ auto\ dest:\ dom\text{-}lemmas)$

lemma $ri\text{-runs2sigs}\text{-}upd\text{-}serv\text{-}none$ $[simp]$:

$\llbracket Rs \notin dom\ runz \rrbracket$

$\Longrightarrow ri\text{-runs2sigs}\ (runz(Rs \mapsto (Serv,\ [A,\ B],\ nl))) = ri\text{-runs2sigs}\ runz$

by $(rule\ ext,\ erule\ rev\text{-}mp)$

$(rule\ ri\text{-runs2sigs}.induct,\ auto\ dest:\ dom\text{-}lemmas)$

lemma *ri-runs2sigs-upd-init* [simp]:
 $\llbracket \text{runz } Ra = \text{Some } (\text{Init}, [A, B], [aKey \text{ Kab}]); \text{finite } (\text{dom } \text{runz}) \rrbracket$
 $\implies \text{ri-runs2sigs } (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [aKey \text{ Kab}, aNon \text{ Nb}]))) =$
 $(\text{ri-runs2sigs } \text{runz})$
 $(\text{Running } [B, A] (\text{Kab}, \text{Nb}) := \text{Suc } (\text{card } (\text{ri-running } \text{runz } A \text{ B } \text{Kab } \text{Nb})))$
apply (*rule ext*, *erule rev-mp*, *erule rev-mp*)
apply (*rule-tac* *?a0.0=runz* **in** *ri-runs2sigs.induct*, *auto*)
— 1 subgoal, solved using $\llbracket \text{finite } ?A; ?x \notin ?A \rrbracket \implies \text{card } (\text{insert } ?x ?A) = \text{Suc } (\text{card } ?A)$
apply (*rename-tac* *runz*)
apply (*rule-tac*
 $s = \text{card } (\text{insert } Ra (\text{ri-running } \text{runz } A \text{ B } \text{Kab } \text{Nb}))$
in *trans*, *fast*, *auto*)
done

lemma *ri-runs2sigs-upd-init-some* [simp]:
 $\llbracket \text{runz } Ra = \text{Some } (\text{Init}, [A, B], []) \rrbracket$
 $\implies \text{ri-runs2sigs } (\text{runz}(Ra \mapsto (\text{Init}, [A, B], [aKey \text{ Kab}]))) = \text{ri-runs2sigs } \text{runz}$
by (*rule ext*, *erule rev-mp*)
(rule ri-runs2sigs.induct, auto)

lemma *ri-runs2sigs-upd-resp-some* [simp]:
 $\llbracket \text{runz } Rb = \text{Some } (\text{Resp}, [A, B], []) \rrbracket$
 $\implies \text{ri-runs2sigs } (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], [aKey \text{ K}]))) = \text{ri-runs2sigs } \text{runz}$
by (*rule ext*, *erule rev-mp*)
(rule ri-runs2sigs.induct, auto)

lemma *ri-runs2sigs-upd-resp-some2* [simp]:
 $\llbracket \text{runz } Rb = \text{Some } (\text{Resp}, [A, B], [aKey \text{ Kab}]) \rrbracket$
 $\implies \text{ri-runs2sigs } (\text{runz}(Rb \mapsto (\text{Resp}, [A, B], [aKey \text{ Kab}, \text{END}]))) =$
 $(\text{ri-runs2sigs } \text{runz})(\text{Commit } [B, A] (\text{Kab}, \text{Rb\$nb}) := 1)$
apply (*rule ext*, *erule rev-mp*)
apply (*rule ri-runs2sigs.induct*, *fastforce+*)
done

Refinement proof

lemma *PO-m1-step1-refines-ri-a0i-skip*:
 $\{R\text{-a0im1-ri}\}$
 $\text{Id}, (\text{m1-step1 } Ra \text{ A } B \text{ Na})$
 $\{> R\text{-a0im1-ri}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ri-defs m1-defs, safe, auto*)

lemma *PO-m1-step2-refines-ri-a0i-skip*:
 $\{R\text{-a0im1-ri}\}$
 $\text{Id}, (\text{m1-step2 } Rb \text{ A } B)$
 $\{> R\text{-a0im1-ri}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ri-defs m1-defs, safe, auto*)

lemma *PO-m1-step3-refines-ri-a0i-skip*:
 $\{R\text{-a0im1-ri}\}$
 $\text{Id}, (\text{m1-step3 } Rs \text{ A } B \text{ Na } \text{Kab})$
 $\{> R\text{-a0im1-ri}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ri-defs a0i-defs m1-defs, safe, auto*)

lemma *PO-m1-step4-refines-ri-a0i-skip*:
 $\{R\text{-a0im1-ri}\}$
 $Id, (m1\text{-step4 } Ra \ A \ B \ Nb \ Kab)$
 $\{> R\text{-a0im1-ri}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ri-defs a0i-defs m1-defs, safe, auto*)

lemma *PO-m1-step5-refines-ri-a0i-skip*:
 $\{R\text{-a0im1-ri}\}$
 $Id, (m1\text{-step5 } Rb \ A \ B \ Nb \ Kab)$
 $\{> R\text{-a0im1-ri}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ri-defs a0i-defs m1-defs, safe, auto*)

lemma *PO-m1-step6-refines-ri-a0i-running*:
 $\{R\text{-a0im1-ri} \cap UNIV \times m1\text{-inv0-fin}\}$
 $(a0i\text{-running } [B, A] (Kab, Nb)), (m1\text{-step6 } Ra \ A \ B \ Na \ Nb \ Kab)$
 $\{> R\text{-a0im1-ri}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ri-defs a0i-defs m1-defs, safe, auto*)

lemma *PO-m1-step7-refines-ri-a0i-commit*:
 $\{R\text{-a0im1-ri} \cap UNIV \times m1\text{-inv0-fin}\}$
 $(a0i\text{-commit } [B, A] (Kab, Nb)), (m1\text{-step7 } Rb \ A \ B \ Nb \ Kab)$
 $\{> R\text{-a0im1-ri}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ri-defs a0i-defs m1-defs, safe, auto*)

lemma *PO-m1-leak-refines-ri-a0i-corrupt*:
 $\{R\text{-a0im1-ri}\}$
 $(a0i\text{-corrupt } (\{sesK (Rs\$sk)\} \times UNIV)), (m1\text{-leak } Rs \ Ra \ Rb \ A \ B)$
 $\{> R\text{-a0im1-ri}\}$
by (*simp add: PO-rhoare-defs R-a0im1-ri-defs a0i-defs m1-defs, safe, auto*)

All together now...

lemmas *PO-m1-trans-refines-ri-a0i-trans =*
PO-m1-step1-refines-ri-a0i-skip PO-m1-step2-refines-ri-a0i-skip
PO-m1-step3-refines-ri-a0i-skip PO-m1-step4-refines-ri-a0i-skip
PO-m1-step5-refines-ri-a0i-skip PO-m1-step6-refines-ri-a0i-running
PO-m1-step7-refines-ri-a0i-commit PO-m1-leak-refines-ri-a0i-corrupt

lemma *PO-m1-refines-ri-init-a0i [iff]*:
 $init \ m1 \subseteq R\text{-a0im1-ri} \text{''}(init \ a0i)$
by (*auto simp add: R-a0im1-ri-defs a0i-defs m1-defs*
intro!: exI [where x=(signals = $\lambda s. 0$, corrupted = corrKey \times UNIV)])

lemma *PO-m1-refines-ri-trans-a0i [iff]*:
 $\{R\text{-a0im1-ri} \cap a0i\text{-inv1-iagree} \times m1\text{-inv0-fin}\}$
 $(trans \ a0i), (trans \ m1)$
 $\{> R\text{-a0im1-ri}\}$
by (*auto simp add: m1-def m1-trans-def a0i-def a0i-trans-def*
(blast intro!: PO-m1-trans-refines-ri-a0i-trans)+)

Observation consistency.

lemma *obs-consistent-med-a0im1-ri [iff]*:

obs-consistent R-a0im1-ri med-a0im1-ri a0i m1
by (*auto simp add: obs-consistent-def R-a0im1-ri-def med-a0im1-ri-def a0i-def m1-def*)

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
by (*rule Refinement-using-invariants*) (*auto*)

lemma *m1-implements-ri-a0i: implements med-a0im1-ri a0i m1*
by (*rule refinement-soundness*) (*fast*)

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*
apply (*rule INV-from-Refinement-basic* [*OF PO-m1-refines-ri-a0i*])
apply (*auto simp add: R-a0im1-ri-def a0i-inv1-iagree-def*
intro!: m1-inv3r-initI)
apply (*rename-tac s A B Rb Kab a*)
apply (*drule-tac x=[B, A] in spec, clarsimp*)
apply (*drule-tac x=Kab in spec*)
apply (*subgoal-tac card (ri-running (runs s) A B Kab (Rb\\$nb)) > 0, auto*)
done

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 $m1\text{-inv4}\text{-rfresh}I = m1\text{-inv4}\text{-rfresh}\text{-def}$ [THEN setc-def-to-intro, rule-format]
lemmas $m1\text{-inv4}\text{-rfresh}E$ [elim] = $m1\text{-inv4}\text{-rfresh}\text{-def}$ [THEN setc-def-to-elim, rule-format]
lemmas $m1\text{-inv4}\text{-rfresh}D = m1\text{-inv4}\text{-rfresh}\text{-def}$ [THEN setc-def-to-dest, rule-format, rotated 1]

Proof of key freshness for responder

lemma $PO\text{-}m1\text{-inv4}\text{-rfresh}\text{-init}$ [iff]:

$init\ m1 \subseteq m1\text{-inv4}\text{-rfresh}$

by (auto simp add: $m1\text{-defs}$ intro!: $m1\text{-inv4}\text{-rfresh}I$)

lemma $PO\text{-}m1\text{-inv4}\text{-rfresh}\text{-trans}$ [iff]:

$\{m1\text{-inv4}\text{-rfresh} \cap m1\text{-inv3}r\text{-init} \cap m1\text{-inv2}r\text{-serv} \cap m1\text{-inv3}\text{-ifresh} \cap m1\text{-secrecy}\}$
 $trans\ m1$

$\{> m1\text{-inv4}\text{-rfresh}\}$

apply (simp add: $PO\text{-hoare}\text{-defs}$ $m1\text{-defs}$, safe intro!: $m1\text{-inv4}\text{-rfresh}I$, simp-all)

apply (auto dest: $m1\text{-inv4}\text{-rfresh}D$)

— 4 subgoals, from responder's final step 7

apply (rename-tac $Rb\ A\ A'\ B\ B'\ Kab\ xa\ xe$)

apply (frule-tac $B=B$ in $m1\text{-inv2}r\text{-serv}D$, fast, fast, clarsimp)

apply (case-tac $B' \notin bad$, auto dest: $m1\text{-inv2}r\text{-serv}D$)

apply (subgoal-tac ($sesK\ (Rs\$sk), B'$) $\in azC$ (runs xa))

prefer 2 **apply** (erule $m1\text{-secrecy}E$, auto)

apply (erule $azC.cases$, auto)

apply (rename-tac $Rb\ A\ A'\ B\ B'\ Kab\ xa\ xe$)

apply (frule-tac $B=B$ in $m1\text{-inv2}r\text{-serv}D$, fast, fast, clarify)

apply (subgoal-tac ($sesK\ (Rs\$sk), B'$) $\in azC$ (runs xa))

prefer 2 **apply** (erule $m1\text{-secrecy}E$, auto)

apply (erule $azC.cases$, auto)

apply (rename-tac $Rb'\ A\ A'\ B\ B'\ Kab\ xa\ xe\ Ra$)

apply (case-tac $A' \notin bad \wedge B' \notin bad$, auto)

apply (frule $m1\text{-inv3}r\text{-init}D$, auto)

apply (rename-tac $Raa\ nla$)

apply (frule-tac $Ra=Ra$ in $m1\text{-inv3}\text{-ifresh}D$, auto)

apply (subgoal-tac $Ra = Raa$, auto)

— $A' \in bad$

apply (frule-tac $B=B$ in $m1\text{-inv2}r\text{-serv}D$, fast, fast, clarify)

apply (rename-tac $Rs\ Na$)

apply (case-tac $B' \notin bad$, auto dest: $m1\text{-inv2}r\text{-serv}D$)

apply (subgoal-tac ($sesK\ (Rs\$sk), B'$) $\in azC$ (runs xa))

prefer 2 **apply** (erule $m1\text{-secrecy}E$, auto)

apply (erule $azC.cases$, auto)

— $B' \in bad$

apply (frule-tac $B=B$ in $m1\text{-inv2}r\text{-serv}D$, fast, fast, clarify)

apply (rename-tac $Rs\ Na$)

apply (subgoal-tac ($sesK\ (Rs\$sk), B'$) $\in azC$ (runs xa))

prefer 2 **apply** (erule $m1\text{-secrecy}E$, auto)

```

apply (erule azC.cases, auto)

apply (frule m1-inv3r-initD, auto)
apply (rename-tac Raa nla)
apply (subgoal-tac Raa = Ra, auto)
done

lemma PO-m1-inv4-rfresh [iff]: reach m1  $\subseteq$  m1-inv4-rfresh
apply (rule-tac
  J=m1-inv3r-init  $\cap$  m1-inv2r-serv  $\cap$  m1-inv3-ifresh  $\cap$  m1-secrecy
  in inv-rule-incr)
apply (auto simp add: Int-assoc del: subsetI)
done

lemma PO-m1-obs-inv4-rfresh [iff]: oreach m1  $\subseteq$  m1-inv4-rfresh
by (rule external-from-internal-invariant)
  (auto del: subsetI)

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

$$s1 = s\{$$

$$runs := (runs\ s)(Rs \mapsto (Serv, [A, B], [aNon\ Na])),$$

$$chan := \{Secure\ Sv\ A\ (Msg\ [aNon\ Na, aAgt\ B, aKey\ Kab]),$$

$$Secure\ Sv\ B\ (Msg\ [aKey\ Kab, aAgt\ A])\} \cup chan\ s$$

$$\}$$

definition — by A , refines $m1a-step4$

$$m2-step4 :: [rid-t, agent, agent, nonce, key] \Rightarrow m2-trans$$

where

$$m2-step4\ Ra\ A\ B\ Na\ Kab \equiv \{(s, s1).\}$$

— guards:

$$runs\ s\ Ra = Some\ (Init, [A, B], []) \wedge$$

$$Na = Ra\$na \wedge$$

$$Secure\ Sv\ A\ (Msg\ [aNon\ Na, aAgt\ B, aKey\ Kab]) \in chan\ s \wedge \text{— recv msg 2}$$

— actions:

— record session key

$$s1 = s\{$$

$$runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab]))$$

$$\}$$

definition — by B , refines $m1-step5$

$$m2-step5 :: [rid-t, agent, agent, nonce, key] \Rightarrow m2-trans$$

where

$$m2-step5\ Rb\ A\ B\ Nb\ Kab \equiv \{(s, s1).\}$$

— guards:

$$runs\ s\ Rb = Some\ (Resp, [A, B], []) \wedge$$

$$Nb = Rb\$nb \wedge$$

$$Secure\ Sv\ B\ (Msg\ [aKey\ Kab, aAgt\ A]) \in chan\ s \wedge \text{— recv msg 3}$$

— actions:

— record session key

$$s1 = s\{$$

$$runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab])),$$

$$chan := insert\ (dAuth\ Kab\ (Msg\ [aNon\ Nb]))\ (chan\ s)$$

$$\}$$

definition — by A , refines $m1-step6$

$$m2-step6 :: [rid-t, agent, agent, nonce, nonce, key] \Rightarrow m2-trans$$

where

$$m2-step6\ Ra\ A\ B\ Na\ Nb\ Kab \equiv \{(s, s').\}$$

$$runs\ s\ Ra = Some\ (Init, [A, B], [aKey\ Kab]) \wedge \text{— key recv'd before}$$

$$Na = Ra\$na \wedge$$

$$dAuth\ Kab\ (Msg\ [aNon\ Nb]) \in chan\ s \wedge \text{— receive } M4$$

— 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]:

$init\ m2 \subseteq m2\text{-inv1-keys}$

by (auto simp add: m2-defs intro!: m2-inv1-keysI)

lemma $PO\text{-}m2\text{-inv1-keys-trans}$ [iff]:

$\{m2\text{-inv1-keys}\ trans\ m2 \{>\ m2\text{-inv1-keys}\}$

apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv1-keysI)

apply (auto dest!: m2-inv1-keysD dest: dom-lemmas)

done

lemma $PO\text{-}m2\text{-inv1-keys}$ [iff]: $reach\ m2 \subseteq m2\text{-inv1-keys}$

by (rule inv-rule-basic) (auto)

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]:

$init\ m2 \subseteq m2\text{-inv2-keys-for}$

by (auto simp add: m2-defs intro!: m2-inv2-keys-forI)

lemma $PO\text{-}m2\text{-inv2-keys-for-trans}$ [iff]:

$\{m2\text{-inv2-keys-for} \cap m2\text{-inv1-keys}\ trans\ m2 \{>\ m2\text{-inv2-keys-for}\}$

apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv2-keys-forI)

apply (auto dest!: m2-inv2-keys-forD dest: m2-inv1-keysD dest: dom-lemmas)

— 2 subgoals, from step 4 and fake

apply (rename-tac $R\ xa\ xb\ xc\ xe$)

apply (subgoal-tac $aKey\ (sesK\ (R\$sk)) \in atoms\ (chan\ xa),$

$auto\ dest!: m2\text{-inv1-keys}D\ dest: dom\text{-lemmas}$)

apply (auto simp add: keys-for-def, erule fake.cases, fastforce+)

done

lemma $PO\text{-}m2\text{-inv2-keys-for}$ [iff]: $reach\ m2 \subseteq m2\text{-inv2-keys-for}$

by (rule inv-rule-incr) (auto del: subsetI)

Useful application of invariant.

lemma $m2\text{-inv2-keys-for--extr-insert-key}$:

$\llbracket R \notin dom\ (runs\ s); s \in m2\text{-inv2-keys-for} \rrbracket$

$\implies extr\ (insert\ (aKey\ (sesK\ (R\$sk)))\ T)\ (chan\ s) = insert\ (aKey\ (sesK\ (R\$sk)))\ (extr\ T\ (chan\ s))$

by (subgoal-tac $sesK\ (R\$sk) \notin keys\text{-for}\ (chan\ s)$) (auto)

inv2b: leaked keys include corrupted ones

definition

$m2\text{-inv2b-corrKey-leaked} :: m2\text{-pred}$

where

$m2\text{-inv2b-corrKey-leaked} \equiv \{s. \forall K.$
 $K \in \text{corrKey} \longrightarrow K \in \text{Domain} (\text{leak } s)$
 $\}$

lemmas $m2\text{-inv2b-corrKey-leakedI} = m2\text{-inv2b-corrKey-leaked-def} [THEN \text{setc-def-to-intro}, \text{rule-format}]$

lemmas $m2\text{-inv2b-corrKey-leakedE} [\text{elim}] = m2\text{-inv2b-corrKey-leaked-def} [THEN \text{setc-def-to-elim}, \text{rule-format}]$

lemmas $m2\text{-inv2b-corrKey-leakedD} = m2\text{-inv2b-corrKey-leaked-def} [THEN \text{setc-def-to-dest}, \text{rule-format}, \text{rotated } 1]$

Invariance proof.

lemma $PO\text{-}m2\text{-inv2b-corrKey-leaked-init} [\text{iff}]$:

$\text{init } m2 \subseteq m2\text{-inv2b-corrKey-leaked}$

by ($\text{auto simp add: } m2\text{-defs intro!} : m2\text{-inv2b-corrKey-leakedI}$)

lemma $PO\text{-}m2\text{-inv2b-corrKey-leaked-trans} [\text{iff}]$:

$\{m2\text{-inv2b-corrKey-leaked} \cap m2\text{-inv1-keys}\} \text{trans } m2 \{> m2\text{-inv2b-corrKey-leaked}\}$

by ($\text{auto simp add: } PO\text{-hoare-defs } m2\text{-defs intro!} : m2\text{-inv2b-corrKey-leakedI}$)

lemma $PO\text{-}m2\text{-inv2b-corrKey-leaked} [\text{iff}]$: $\text{reach } m2 \subseteq m2\text{-inv2b-corrKey-leaked}$

by ($\text{rule inv-rule-incr}$) (auto del: subsetI)

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

$m2\text{-inv3a-sesK-compr} :: m2\text{-state set}$

where

$m2\text{-inv3a-sesK-compr} \equiv \{s. \forall K KK.$

~~$KK \neq \{\}/\text{not/true}/\text{set } K/\text{###}$~~

$aKey K \in \text{extr} (aKey KK \cup ik0) (\text{chan } s) \longleftrightarrow (K \in KK \vee aKey K \in \text{extr } ik0 (\text{chan } s))$

$\}$

lemmas $m2\text{-inv3a-sesK-comprI} = m2\text{-inv3a-sesK-compr-def} [THEN \text{setc-def-to-intro}, \text{rule-format}]$

lemmas $m2\text{-inv3a-sesK-comprE} [\text{elim}] = m2\text{-inv3a-sesK-compr-def} [THEN \text{setc-def-to-elim}, \text{rule-format}]$

lemmas $m2\text{-inv3a-sesK-comprD} = m2\text{-inv3a-sesK-compr-def} [THEN \text{setc-def-to-dest}, \text{rule-format}]$

Additional lemma to get the keys in front

lemmas $\text{insert-commute-aKey} = \text{insert-commute} [\text{where } x = aKey K \text{ for } K]$

lemmas $m2\text{-inv3a-sesK-compr-simps} =$

$m2\text{-inv3a-sesK-comprD}$

$m2\text{-inv3a-sesK-comprD} [\text{where } KK = \{Kab\} \text{ for } Kab, \text{simplified}]$

$m2\text{-inv3a-sesK-comprD} [\text{where } KK = \text{insert } Kab KK \text{ for } Kab KK, \text{simplified}]$

$\text{insert-commute-aKey}$

inv3: Lost session keys

inv3: Lost 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

$m2\text{-inv3}\text{-extrKey} :: m2\text{-state set}$

where

$m2\text{-inv3}\text{-extrKey} \equiv \{s. \forall K.$
 $aKey K \in extr\ ik0\ (chan\ s) \longrightarrow K \notin corrKey \longrightarrow$
 $(\exists R\ A'\ B'\ Na'. K = sesK\ (R\$sk) \wedge$
 $runs\ s\ R = Some\ (Serv, [A', B'], [aNon\ Na']) \wedge$
 $(A' \in bad \vee B' \in bad \vee (\exists Nb'. (K, Na', Nb') \in leak\ s)))$
}

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*]:

$init\ m2 \subseteq m2\text{-inv3}\text{-extrKey}$

by (*auto simp add: m2-defs intro!: m2-inv3-extrKeyI*)

lemma $PO\text{-}m2\text{-inv3}\text{-extrKey}\text{-trans}$ [*iff*]:

$\{m2\text{-inv3}\text{-extrKey} \cap m2\text{-inv3a}\text{-sesK}\text{-compr}\} trans\ m2 \{>\ m2\text{-inv3}\text{-extrKey}\}$

apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv3-extrKeyI*)

apply (*auto simp add: m2-inv3a-sesK-compr-simps dest: dom-lemmas*)

— 11 subgoals, sledgehammer

apply (*metis m2-inv3-extrKeyD map-definedness-contr*)

apply (*metis m2-inv3-extrKeyD map-definedness-contr*)

apply (*metis m2-inv3-extrKeyD*)

apply (*metis m2-inv3-extrKeyD*)

apply (*metis m2-inv3-extrKeyD*)

apply (*metis m2-inv3-extrKeyD map-definedness-contr*)

apply (*metis m2-inv3-extrKeyD not-Cons-self2 prod.inject option.inject*)

apply (*metis m2-inv3-extrKeyD not-Cons-self2 prod.inject option.inject*)

apply (*metis m2-inv3-extrKeyD atom.distinct(7) list.inject option.inject prod.inject*)

apply (*metis m2-inv3-extrKeyD atom.distinct(7) list.inject option.inject prod.inject*)

apply (*metis m2-inv3-extrKeyD*)

done

lemma $PO\text{-}m2\text{-inv3}\text{-extrKey}$ [*iff*]: $reach\ m2 \subseteq m2\text{-inv3}\text{-extrKey}$

by (*rule-tac J=m2-inv3a-sesK-compr in inv-rule-incr*) (*auto*)

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

$m2\text{-inv4}\text{-}M2 \equiv \{s. \forall A B Na Kab.$
 $\text{Secure Sv } A (\text{Msg } [a\text{Non } Na, a\text{Agt } B, a\text{Key } Kab]) \in \text{chan } s \longrightarrow A \in \text{good} \longrightarrow$
 $(\exists Rs. Kab = \text{sesK } (Rs\$sk) \wedge \text{runs } s Rs = \text{Some } (\text{Serv}, [A, B], [a\text{Non } Na]))$
 $\}$

lemmas $m2\text{-inv4}\text{-}M2I = m2\text{-inv4}\text{-}M2\text{-def } [THEN \text{setc}\text{-def}\text{-to}\text{-intro}, \text{rule}\text{-format}]$
lemmas $m2\text{-inv4}\text{-}M2E [\text{elim}] = m2\text{-inv4}\text{-}M2\text{-def } [THEN \text{setc}\text{-def}\text{-to}\text{-elim}, \text{rule}\text{-format}]$
lemmas $m2\text{-inv4}\text{-}M2D = m2\text{-inv4}\text{-}M2\text{-def } [THEN \text{setc}\text{-def}\text{-to}\text{-dest}, \text{rule}\text{-format}, \text{rotated } 1]$

Invariance proof.

lemma $PO\text{-}m2\text{-inv4}\text{-}M2\text{-init } [iff]:$

$\text{init } m2 \subseteq m2\text{-inv4}\text{-}M2$

by ($\text{auto simp add: } m2\text{-defs intro!: } m2\text{-inv4}\text{-}M2I$)

lemma $PO\text{-}m2\text{-inv4}\text{-}M2\text{-trans } [iff]:$

$\{m2\text{-inv4}\text{-}M2\} \text{ trans } m2 \{> m2\text{-inv4}\text{-}M2\}$

apply ($\text{auto simp add: } PO\text{-hoare}\text{-defs } m2\text{-defs intro!: } m2\text{-inv4}\text{-}M2I$)

apply ($\text{auto dest!: } m2\text{-inv4}\text{-}M2D \text{ dest: dom}\text{-lemmas}$)

— 5 subgoals

apply (force dest!: spec)

apply (force dest!: spec)

apply (force dest!: spec)

apply (rule exI, auto)⁺

done

lemma $PO\text{-}m2\text{-inv4}\text{-}M2 [iff]: \text{reach } m2 \subseteq m2\text{-inv4}\text{-}M2$

by ($\text{rule inv}\text{-rule}\text{-basic}$) (auto)

inv4b: Secure channel and message 3

definition

$m2\text{-inv4}\text{-}M3 :: m2\text{-pred}$

where

$m2\text{-inv4}\text{-}M3 \equiv \{s. \forall A B Kab.$

$\text{Secure Sv } B (\text{Msg } [a\text{Key } Kab, a\text{Agt } A]) \in \text{chan } s \longrightarrow B \in \text{good} \longrightarrow$

$(\exists Rs Na. Kab = \text{sesK } (Rs\$sk) \wedge \text{runs } s Rs = \text{Some } (\text{Serv}, [A, B], [a\text{Non } Na]))$

$\}$

lemmas $m2\text{-inv4}\text{-}M3I = m2\text{-inv4}\text{-}M3\text{-def } [THEN \text{setc}\text{-def}\text{-to}\text{-intro}, \text{rule}\text{-format}]$

lemmas $m2\text{-inv4}\text{-}M3E [\text{elim}] = m2\text{-inv4}\text{-}M3\text{-def } [THEN \text{setc}\text{-def}\text{-to}\text{-elim}, \text{rule}\text{-format}]$

lemmas $m2\text{-inv4}\text{-}M3D = m2\text{-inv4}\text{-}M3\text{-def } [THEN \text{setc}\text{-def}\text{-to}\text{-dest}, \text{rule}\text{-format}, \text{rotated } 1]$

Invariance proof.

lemma $PO\text{-}m2\text{-inv4}\text{-}M3\text{-init } [iff]:$

$\text{init } m2 \subseteq m2\text{-inv4}\text{-}M3$

by ($\text{auto simp add: } m2\text{-defs intro!: } m2\text{-inv4}\text{-}M3I$)

lemma $PO\text{-}m2\text{-inv4}\text{-}M3\text{-trans } [iff]:$

$\{m2\text{-inv4}\text{-}M3\} \text{ trans } m2 \{> m2\text{-inv4}\text{-}M3\}$

apply ($\text{auto simp add: } PO\text{-hoare}\text{-defs } m2\text{-defs intro!: } m2\text{-inv4}\text{-}M3I$)

apply ($\text{auto dest!: } m2\text{-inv4}\text{-}M3D \text{ dest: dom}\text{-lemmas}$)

— 5 subgoals

apply (force)⁺

done

lemma *PO-m2-inv4-M3 [iff]: reach m2 \subseteq m2-inv4-M3*
by (rule *inv-rule-incr*) (auto del: *subsetI*)

Consequence needed in proof of *inv8/step5*

lemma *m2-inv4-M2-M3-unique-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-inv4-M2 s \in m2-inv4-M3

shows

A = A' \wedge B = B'

proof (cases *A' \in bad \vee B \in bad*)

case *True* **thus** ?thesis **using** *assms(1-3)* **by** auto

next

case *False* **thus** ?thesis **using** *assms(1,2,4,5)* **by** (auto dest!: *m2-inv4-M2D m2-inv4-M3D*)

qed

More consequences of invariants. Needed in *ref/step4* and *ref/step5* respectively to show the strengthening of the authorization guards.

lemma *m2-inv34-M2-authorized:*

assumes *Secure Sv A (Msg [aNon N, aAgt B, aKey K]) \in chan s*
s \in m2-inv4-M2 s \in m2-inv3-extrKey s \in m2-inv2b-corrKey-leaked
K \notin Domain (leak s)

shows *(K, A) \in azC (runs s)*

proof (cases)

assume *A \in bad*

hence *aKey K \in extr ik0 (chan s)* **using** *assms(1)* **by** auto

thus ?thesis **using** *assms(3-)* **by** auto

next

assume *A \notin bad*

thus ?thesis **using** *assms(1-2)* **by** (auto dest: *m2-inv4-M2D*)

qed

lemma *m2-inv34-M3-authorized:*

assumes *Secure Sv B (Msg [aKey K, aAgt A]) \in chan s*
s \in m2-inv4-M3 s \in m2-inv3-extrKey s \in m2-inv2b-corrKey-leaked
K \notin Domain (leak s)

shows *(K, B) \in azC (runs s)*

proof (cases)

assume *B \in bad*

hence *aKey K \in extr ik0 (chan s)* **using** *assms(1)* **by** auto

thus ?thesis **using** *assms(3-)* **by** auto

next

assume *B \notin bad*

thus ?thesis **using** *assms(1-2)* **by** (auto dest: *m2-inv4-M3D*)

qed

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

$m2\text{-inv5}\text{-ikk}\text{-sv} :: m2\text{-pred}$

where

$m2\text{-inv5}\text{-ikk}\text{-sv} \equiv \{s. \forall Rs A B Na al.$
 $runs\ s\ Rs = Some\ (Serv, [A, B], aNon\ Na\ \# al) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$
 $aKey\ (sesK\ (Rs\$sk)) \in extr\ ik0\ (chan\ s) \longrightarrow$
 $(\exists Nb'. (sesK\ (Rs\$sk), Na, Nb') \in leak\ s)$
 $\}$

lemmas $m2\text{-inv5}\text{-ikk}\text{-sv}I = m2\text{-inv5}\text{-ikk}\text{-sv}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m2\text{-inv5}\text{-ikk}\text{-sv}E\ [elim] = m2\text{-inv5}\text{-ikk}\text{-sv}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m2\text{-inv5}\text{-ikk}\text{-sv}D = m2\text{-inv5}\text{-ikk}\text{-sv}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

Invariance proof. This invariant follows from $m2\text{-inv3}\text{-extrKey}$.

lemma $m2\text{-inv5}\text{-ikk}\text{-sv}\text{-derived}$:

$s \in m2\text{-inv3}\text{-extrKey} \implies s \in m2\text{-inv5}\text{-ikk}\text{-sv}$

by (*auto simp add: m2-inv3-extrKey-def m2-inv5-ikk-sv-def*) (*force*)

lemma $PO\text{-}m2\text{-inv5}\text{-ikk}\text{-sv}\ [iff]: reach\ m2 \subseteq m2\text{-inv5}\text{-ikk}\text{-sv}$

proof –

have $reach\ m2 \subseteq m2\text{-inv3}\text{-extrKey}$ **by** *blast*

also have $\dots \subseteq m2\text{-inv5}\text{-ikk}\text{-sv}$ **by** (*blast intro: m2-inv5-ikk-sv-derived*)

finally show *?thesis* .

qed

inv6 (derived): Key secrecy for initiator

This invariant is derivable (see below).

definition

$m2\text{-inv6}\text{-ikk}\text{-init} :: m2\text{-pred}$

where

$m2\text{-inv6}\text{-ikk}\text{-init} \equiv \{s. \forall Ra K A B al.$
 $runs\ s\ Ra = Some\ (Init, [A, B], aKey\ K\ \# al) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$
 $aKey\ K \in extr\ ik0\ (chan\ s) \longrightarrow$
 $(\exists Nb'. (K, Ra\ \$\ na, Nb') \in leak\ s)$
 $\}$

lemmas $m2\text{-inv6}\text{-ikk}\text{-init}I = m2\text{-inv6}\text{-ikk}\text{-init}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-intro},\ rule\text{-format}]$

lemmas $m2\text{-inv6}\text{-ikk}\text{-init}E\ [elim] = m2\text{-inv6}\text{-ikk}\text{-init}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-elim},\ rule\text{-format}]$

lemmas $m2\text{-inv6}\text{-ikk}\text{-init}D = m2\text{-inv6}\text{-ikk}\text{-init}\text{-def}\ [THEN\ setc\text{-def}\text{-to}\text{-dest},\ rule\text{-format},\ rotated\ 1]$

inv7 (derived): Key secrecy for responder

This invariant is derivable (see below).

definition

$m2\text{-inv7}\text{-ikk}\text{-resp} :: m2\text{-pred}$

where

```

m2-inv7-ikk-resp ≡ {s. ∀ Rb K A B al.
  runs s Rb = Some (Resp, [A, B], aKey K # al) → A ∈ good → B ∈ good →
  aKey K ∈ extr ik0 (chan s) →
  K ∈ Domain (leak s)
}

```

lemmas *m2-inv7-ikk-respI* = *m2-inv7-ikk-resp-def* [*THEN setc-def-to-intro, rule-format*]
lemmas *m2-inv7-ikk-respE* [*elim*] = *m2-inv7-ikk-resp-def* [*THEN setc-def-to-elim, rule-format*]
lemmas *m2-inv7-ikk-respD* = *m2-inv7-ikk-resp-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-inv8-M4 :: *m2-pred*

where

```

m2-inv8-M4 ≡ {s. ∀ Kab A B Na Nb.
  Secure Sv A (Msg [aNon Na, aAgt B, aKey Kab]) ∈ chan s →
  dAuth Kab (Msg [aNon Nb]) ∈ chan s →
  aKey Kab ∉ extr ik0 (chan s) →
  (∃ Rb. Nb = Rb$nb ∧ (∃ al. runs s Rb = Some (Resp, [A, B], aKey Kab # al)))
}

```

lemmas *m2-inv8-M4I* = *m2-inv8-M4-def* [*THEN setc-def-to-intro, rule-format*]
lemmas *m2-inv8-M4E* [*elim*] = *m2-inv8-M4-def* [*THEN setc-def-to-elim, rule-format*]
lemmas *m2-inv8-M4D* = *m2-inv8-M4-def* [*THEN setc-def-to-dest, rule-format, rotated 1*]

Invariance proof.

lemma *PO-m2-inv8-M4-step1*:

{*m2-inv8-M4*} *m2-step1 Ra A B Na* {> *m2-inv8-M4*}

apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv8-M4I*)

apply (*auto dest!: m2-inv8-M4D dest: dom-lemmas*)

done

lemma *PO-m2-inv8-M4-step2*:

{*m2-inv8-M4*} *m2-step2 Rb A B* {> *m2-inv8-M4*}

apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv8-M4I*)

apply (*auto dest!: m2-inv8-M4D dest: dom-lemmas*)

done

lemma *PO-m2-inv8-M4-step3*:

{*m2-inv8-M4* ∩ *m2-inv2-keys-for*} *m2-step3 Rs A B Na Kab* {> *m2-inv8-M4*}

apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv8-M4I*)

apply (*auto simp add: m2-inv2-keys-for--extr-insert-key dest!: m2-inv8-M4D dest: dom-lemmas*)

done

lemma *PO-m2-inv8-M4-step4*:

{*m2-inv8-M4*} *m2-step4 Ra A B Na Kab* {> *m2-inv8-M4*}

apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv8-M4I*)

— 1 subgoal

apply (*drule* *m2-inv8-M4D*, *auto*)

apply (*rule* *exI*, *auto*)

done

lemma *PO-m2-inv8-M4-step5*:

$\{m2\text{-inv8-}M_4 \cap m2\text{-inv4-}M_3 \cap m2\text{-inv4-}M_2\}$

m2-step5 *Rb A B Nb Kab*

$\{> m2\text{-inv8-}M_4\}$

apply (*auto simp add*: *PO-hoare-defs m2-defs intro!*: *m2-inv8-M4I*)

apply (*auto dest*: *m2-inv4-M2-M3-unique-names*)

apply (*auto dest!*: *m2-inv8-M4D*)

done

lemma *PO-m2-inv8-M4-step6*:

$\{m2\text{-inv8-}M_4\}$ *m2-step6* *Ra A B Na Nb Kab* $\{> m2\text{-inv8-}M_4\}$

apply (*auto simp add*: *PO-hoare-defs m2-defs intro!*: *m2-inv8-M4I*)

apply (*auto dest!*: *m2-inv8-M4D*)

— 1 subgoal

apply (*rule* *exI*, *auto*)

done

lemma *PO-m2-inv8-M4-step7*:

$\{m2\text{-inv8-}M_4\}$ *m2-step7* *Rb A B Nb Kab* $\{> m2\text{-inv8-}M_4\}$

apply (*auto simp add*: *PO-hoare-defs m2-defs intro!*: *m2-inv8-M4I*)

apply (*auto dest!*: *m2-inv8-M4D*)

done

lemma *PO-m2-inv8-M4-leak*:

$\{m2\text{-inv8-}M_4 \cap m2\text{-inv3a-sesK-compr}\}$ *m2-leak* *Rs Ra Rb A B* $\{> m2\text{-inv8-}M_4\}$

apply (*auto simp add*: *PO-hoare-defs m2-defs intro!*: *m2-inv8-M4I*)

apply (*auto simp add*: *m2-inv3a-sesK-compr-simps dest!*: *m2-inv8-M4D*)

done

lemma *PO-m2-inv8-M4-fake*:

$\{m2\text{-inv8-}M_4\}$ *m2-fake* $\{> m2\text{-inv8-}M_4\}$

apply (*auto simp add*: *PO-hoare-defs m2-defs intro!*: *m2-inv8-M4I*)

— 1 subgoal

apply (*erule* *fake.cases*, *auto dest!*: *m2-inv8-M4D*)

done

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-inv8-M4*

by (*auto simp add*: *m2-defs intro!*: *m2-inv8-M4I*)

lemma *PO-m2-inv8-M4-trans* [*iff*]:

$\{m2\text{-inv8-}M_4 \cap m2\text{-inv4-}M_3 \cap m2\text{-inv4-}M_2 \cap m2\text{-inv3a-sesK-compr} \cap m2\text{-inv2-keys-for}\}$

$trans\ m2$
 $\{> m2-inv8-M4\}$
by (*auto simp add: m2-def m2-trans-def intro!: PO-m2-inv8-M4-lemmas*)

lemma *PO-m2-inv8-M4 [iff]: reach m2 \subseteq m2-inv8-M4*
by (*rule-tac J=m2-inv4-M3 \cap m2-inv4-M2 \cap m2-inv3a-sesK-compr \cap m2-inv2-keys-for in inv-rule-incr*)

(auto)

inv8a: Relating the initiator state to M2

definition

m2-inv8a-init-M2 :: m2-pred

where

$m2-inv8a-init-M2 \equiv \{s. \forall Ra\ A\ B\ Kab\ al.$
 $runs\ s\ Ra = Some\ (Init,\ [A,\ B],\ aKey\ Kab\ \# \ al) \longrightarrow$
 $Secure\ Sv\ A\ (Msg\ [aNon\ (Ra\$na),\ aAgt\ B,\ aKey\ Kab]) \in\ chan\ s$
 $\}$

lemmas *m2-inv8a-init-M2I = m2-inv8a-init-M2-def [THEN setc-def-to-intro, rule-format]*
lemmas *m2-inv8a-init-M2E [elim] = m2-inv8a-init-M2-def [THEN setc-def-to-elim, rule-format]*
lemmas *m2-inv8a-init-M2D = m2-inv8a-init-M2-def [THEN setc-def-to-dest, rule-format, rotated 1]*

Invariance proof.

lemma *PO-m2-inv8a-init-M2-init [iff]:*
 $init\ m2 \subseteq m2-inv8a-init-M2$
by (*auto simp add: m2-defs intro!: m2-inv8a-init-M2I*)

lemma *PO-m2-inv8a-init-M2-trans [iff]:*
 $\{m2-inv8a-init-M2\}$
 $trans\ m2$
 $\{> m2-inv8a-init-M2\}$
apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv8a-init-M2I*)
apply (*blast*)
done

lemma *PO-m2-inv8a-init-M2 [iff]: reach m2 \subseteq m2-inv8a-init-M2*
by (*rule inv-rule-incr (auto del: subsetI)*)

inv9a: Relating the responder state to M3

definition

m2-inv9a-resp-M3 :: m2-pred

where

$m2-inv9a-resp-M3 \equiv \{s. \forall Rb\ A\ B\ Kab\ al.$
 $runs\ s\ Rb = Some\ (Resp,\ [A,\ B],\ aKey\ Kab\ \# \ al) \longrightarrow$
 $Secure\ Sv\ B\ (Msg\ [aKey\ Kab,\ aAgt\ A]) \in\ chan\ s$
 $\}$

lemmas *m2-inv9a-resp-M3I = m2-inv9a-resp-M3-def [THEN setc-def-to-intro, rule-format]*
lemmas *m2-inv9a-resp-M3E [elim] = m2-inv9a-resp-M3-def [THEN setc-def-to-elim, rule-format]*
lemmas *m2-inv9a-resp-M3D = m2-inv9a-resp-M3-def [THEN setc-def-to-dest, rule-format, rotated 1]*

Invariance proof.

lemma *PO-m2-inv9a-resp-M3-init* [iff]:
 $init\ m2 \subseteq m2\text{-inv9a-resp-M3}$
by (*auto simp add: m2-defs intro!: m2-inv9a-resp-M3I*)

lemma *PO-m2-inv9a-resp-M3-trans* [iff]:
 $\{m2\text{-inv9a-resp-M3}\}$
 $trans\ m2$
 $\{>\ m2\text{-inv9a-resp-M3}\}$
by (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9a-resp-M3I dest: m2-inv9a-resp-M3D*)
(blast)

lemma *PO-m2-inv9a-resp-M3* [iff]: $reach\ m2 \subseteq m2\text{-inv9a-resp-M3}$
by (*rule inv-rule-incr*) (*auto del: subsetI*)

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 $m2\text{-inv9-M5I} = m2\text{-inv9-M5-def}\ [THEN\ setc\text{-def-to-intro},\ rule\text{-format}]$
lemmas $m2\text{-inv9-M5E}\ [elim] = m2\text{-inv9-M5-def}\ [THEN\ setc\text{-def-to-elim},\ rule\text{-format}]$
lemmas $m2\text{-inv9-M5D} = m2\text{-inv9-M5-def}\ [THEN\ setc\text{-def-to-dest},\ rule\text{-format},\ rotated\ 1]$

Invariance proof.

lemma *PO-m2-inv9-M5-step1*:
 $\{m2\text{-inv9-M5}\}\ m2\text{-step1}\ Ra\ A\ B\ Na\ \{>\ m2\text{-inv9-M5}\}$
apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M5I*)
apply (*auto dest!: m2-inv9-M5D dest: dom-lemmas*)
done

lemma *PO-m2-inv9-M5-step2*:
 $\{m2\text{-inv9-M5}\}\ m2\text{-step2}\ Rb\ A\ B\ \{>\ m2\text{-inv9-M5}\}$
apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M5I*)
apply (*auto dest!: m2-inv9-M5D dest: dom-lemmas*)
done

lemma *PO-m2-inv9-M5-step3*:
 $\{m2\text{-inv9-M5} \cap m2\text{-inv2-keys-for}\}\ m2\text{-step3}\ Rs\ A\ B\ Na\ Kab\ \{>\ m2\text{-inv9-M5}\}$
apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M5I*)

apply (*auto simp add: m2-inv2-keys-for--extr-insert-key dest!: m2-inv9-M5D dest: dom-lemmas*)
done

lemma *PO-m2-inv9-M5-step4:*

$\{m2\text{-inv9-M5}\} m2\text{-step4 } Ra A B Na Kab \{> m2\text{-inv9-M5}\}$

apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M5I*)

apply (*auto dest!: m2-inv9-M5D dest: dom-lemmas*)

— 1 subgoal

apply (*rule exI, auto*)

done

lemma *PO-m2-inv9-M5-step5:*

$\{m2\text{-inv9-M5}\} m2\text{-step5 } Rb A B Nb Kab \{> m2\text{-inv9-M5}\}$

apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M5I*)

— 1 subgoal

apply (*drule m2-inv9-M5D, fast, fast, fast, clarsimp*)

apply (*rule exI, auto*)

done

lemma *PO-m2-inv9-M5-step6:*

$\{m2\text{-inv9-M5} \cap m2\text{-inv8a-init-M2} \cap m2\text{-inv9a-resp-M3} \cap m2\text{-inv4-M2} \cap m2\text{-inv4-M3}\}$

$m2\text{-step6 } Ra A B Na Nb Kab$

$\{> m2\text{-inv9-M5}\}$

apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M5I*)

— 2 subgoals

defer 1

apply (*drule m2-inv9-M5D, fast, fast, fast, clarsimp*)

apply (*rename-tac Raa, rule-tac x=Raa in exI, auto*)

apply (*auto dest!: m2-inv8a-init-M2D m2-inv9a-resp-M3D m2-inv4-M2-M3-unique-names*)

done

lemma *PO-m2-inv9-M5-step7:*

$\{m2\text{-inv9-M5}\} m2\text{-step7 } Rb A B Nb Kab \{> m2\text{-inv9-M5}\}$

apply (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M5I*)

— 1 subgoal

apply (*drule m2-inv9-M5D, fast, fast, fast, clarsimp*)

apply (*rule exI, auto*)

done

lemma *PO-m2-inv9-M5-leak:*

$\{m2\text{-inv9-M5} \cap m2\text{-inv3a-sesK-compr}\} m2\text{-leak } Rs Ra Rb A B \{> m2\text{-inv9-M5}\}$

by (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M5I*)

(*auto simp add: m2-inv3a-sesK-compr-simps dest!: m2-inv9-M5D*)

lemma *PO-m2-inv9-M5-fake:*

$\{m2\text{-inv9-M5}\} m2\text{-fake} \{> m2\text{-inv9-M5}\}$

by (*auto simp add: PO-hoare-defs m2-defs intro!: m2-inv9-M5I*)

(*auto dest!: m2-inv9-M5D*)

All together now.

lemmas *PO-m2-inv9-M5-lemmas =*

PO-m2-inv9-M5-step1 PO-m2-inv9-M5-step2 PO-m2-inv9-M5-step3

*PO-m2-inv9-M5-step4 PO-m2-inv9-M5-step5 PO-m2-inv9-M5-step6
 PO-m2-inv9-M5-step7 PO-m2-inv9-M5-leak PO-m2-inv9-M5-fake*

lemma *PO-m2-inv9-M5-init* [iff]:

init m2 ⊆ m2-inv9-M5

by (*auto simp add: m2-defs intro!: m2-inv9-M5I*)

lemma *PO-m2-inv9-M5-trans* [iff]:

*{m2-inv9-M5 ∩ m2-inv8a-init-M2 ∩ m2-inv9a-resp-M3 ∩
 m2-inv4-M2 ∩ m2-inv4-M3 ∩ m2-inv3a-sesK-compr ∩ m2-inv2-keys-for}
 trans m2*

{> m2-inv9-M5}

by (*auto simp add: m2-def m2-trans-def intro!: PO-m2-inv9-M5-lemmas*)

lemma *PO-m2-inv9-M5* [iff]: *reach m2 ⊆ m2-inv9-M5*

by (*rule-tac J=m2-inv8a-init-M2 ∩ m2-inv9a-resp-M3 ∩*

m2-inv4-M2 ∩ m2-inv4-M3 ∩ m2-inv3a-sesK-compr ∩ m2-inv2-keys-for

in *inv-rule-incr*)

(*auto simp add: Int-assoc del: subsetI*)

3.10.5 Refinement

The simulation relation. This is a pure superposition refinement.

definition

R12 :: (*m1-state* × *m2-state*) set **where**

R12 ≡ {(*s*, *t*). *runs s = runs t* ∧ *leak s = leak t*}

The mediator function projects on the local states.

definition

med21 :: *m2-obs* ⇒ *m1-obs* **where**

med21 o2 = (| *runs = runs o2*, *leak = leak o2* |)

Refinement proof.

lemma *PO-m2-step1-refines-m1-step1*:

{*R12*}

(*m1-step1 Ra A B Na*), (*m2-step1 Ra A B Na*)

{> *R12*}

by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

lemma *PO-m2-step2-refines-m1-step2*:

{*R12*}

(*m1-step2 Rb A B*), (*m2-step2 Rb A B*)

{> *R12*}

by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

lemma *PO-m2-step3-refines-m1-step3*:

{*R12*}

(*m1-step3 Rs A B Na Kab*), (*m2-step3 Rs A B Na Kab*)

{> *R12*}

by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

lemma *PO-m2-step4-refines-m1-step4*:

$\{R12 \cap UNIV \times (m2-inv4-M2 \cap m2-inv3-extrKey \cap m2-inv2b-corrKey-leaked)\}$
 $(m1-step4 Ra A B Na Kab), (m2-step4 Ra A B Na Kab)$
 $\{> R12\}$
by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, simp-all*)
(auto dest: m2-inv34-M2-authorized)

lemma *PO-m2-step5-refines-m1-step5:*
 $\{R12 \cap UNIV \times (m2-inv4-M3 \cap m2-inv3-extrKey \cap m2-inv2b-corrKey-leaked)\}$
 $(m1-step5 Rb A B Nb Kab), (m2-step5 Rb A B Nb Kab)$
 $\{> R12\}$
by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, simp-all*)
(auto dest: m2-inv34-M3-authorized)

lemma *PO-m2-step6-refines-m1-step6:*
 $\{R12 \cap UNIV \times (m2-inv8a-init-M2 \cap m2-inv8-M4 \cap m2-inv6-ikk-init)\}$
 $(m1-step6 Ra A B Na Nb Kab), (m2-step6 Ra A B Na Nb Kab)$
 $\{> R12\}$
by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)
(auto intro!: m2-inv8-M4D [OF m2-inv8a-init-M2D] dest: m2-inv6-ikk-initD)

lemma *PO-m2-step7-refines-m1-step7:*
 $\{R12 \cap UNIV \times (m2-inv9-M5 \cap m2-inv9a-resp-M3 \cap m2-inv7-ikk-resp)\}$
 $(m1-step7 Rb A B Nb Kab), (m2-step7 Rb A B Nb Kab)$
 $\{> R12\}$
by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)
(auto intro!: m2-inv9-M5D [OF m2-inv9a-resp-M3D] dest: m2-inv7-ikk-respD)

lemma *PO-m2-leak-refines-leak:*
 $\{R12\}$
 $m1-leak Rs Ra Rb A B, m2-leak Rs Ra Rb A B$
 $\{> R12\}$
by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

lemma *PO-m2-fake-refines-skip:*
 $\{R12\}$
 $Id, m2-fake$
 $\{> R12\}$
by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

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 –
have $t \in m1-inv2i-serv$ **using** *assms(1,2)* **by** (*simp add: R12-def m1-inv2i-serv-def*)
thus *?thesis* **using** *assms(3)*
by (*auto simp add: m2-inv6-ikk-init-def dest: m1-inv2i-servD m2-inv5-ikk-svD*)
qed

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 –

have $t \in m1\text{-inv}2r\text{-serv}$ **using** $assms(1,2)$ **by** (*simp add: R12-def m1-inv2r-serv-def*)
thus $?thesis$ **using** $assms(3)$
by (*auto simp add: m2-inv7-ikk-resp-def dest!: m1-inv2r-servD m2-inv5-ikk-svD*)
qed

All together now...

lemmas $PO\text{-}m2\text{-trans-refines-}m1\text{-trans} =$
 $PO\text{-}m2\text{-step1-refines-}m1\text{-step1}$ $PO\text{-}m2\text{-step2-refines-}m1\text{-step2}$
 $PO\text{-}m2\text{-step3-refines-}m1\text{-step3}$ $PO\text{-}m2\text{-step4-refines-}m1\text{-step4}$
 $PO\text{-}m2\text{-step5-refines-}m1\text{-step5}$ $PO\text{-}m2\text{-step6-refines-}m1\text{-step6}$
 $PO\text{-}m2\text{-step7-refines-}m1\text{-step7}$ $PO\text{-}m2\text{-leak-refines-leak}$
 $PO\text{-}m2\text{-fake-refines-skip}$

lemma $PO\text{-}m2\text{-refines-init-}m1$ [*iff*]:
 $init\ m2 \subseteq R12''(init\ m1)$
by (*auto simp add: R12-def m2-defs*)

lemma $PO\text{-}m2\text{-refines-trans-}m1$ [*iff*]:
 $\{R12 \cap$
 $(reach\ m1 \times$
 $(m2\text{-inv}9\text{-}M5 \cap m2\text{-inv}8a\text{-init-}M2 \cap m2\text{-inv}9a\text{-resp-}M3 \cap m2\text{-inv}8\text{-}M4 \cap$
 $m2\text{-inv}4\text{-}M3 \cap m2\text{-inv}4\text{-}M2 \cap m2\text{-inv}3a\text{-sesK-compr} \cap m2\text{-inv}3\text{-extrKey} \cap m2\text{-inv}2b\text{-corrKey-leaked}))\}$
 $(trans\ m1), (trans\ m2)$
 $\{> R12\}$

proof –

– derive the key secrecy invariants from simulation relation and the other invariants

let $?pre' = R12 \cap (UNIV \times (m2\text{-inv}9\text{-}M5 \cap m2\text{-inv}8a\text{-init-}M2 \cap m2\text{-inv}9a\text{-resp-}M3 \cap$
 $m2\text{-inv}8\text{-}M4 \cap m2\text{-inv}7\text{-ikk-resp} \cap m2\text{-inv}6\text{-ikk-init} \cap m2\text{-inv}5\text{-ikk-sv} \cap$
 $m2\text{-inv}4\text{-}M3 \cap m2\text{-inv}4\text{-}M2 \cap m2\text{-inv}3a\text{-sesK-compr} \cap m2\text{-inv}3\text{-extrKey} \cap$
 $m2\text{-inv}2b\text{-corrKey-leaked}))$

show $?thesis$ (**is** $\{?pre\}$ $?t1, ?t2$ $\{> ?post\}$)

proof (*rule relhoare-conseq-left*)

show $?pre \subseteq ?pre'$

by (*auto intro: m2-inv6-ikk-init-derived m2-inv7-ikk-resp-derived m2-inv5-ikk-sv-derived*)

next

show $\{?pre'\}$ $?t1, ?t2$ $\{> ?post\}$

by (*auto simp add: m2-def m2-trans-def m1-def m1-trans-def*)

(*blast intro!: PO-m2-trans-refines-m1-trans*)**+**

qed

qed

lemma $PO\text{-obs-consistent-}R12$ [*iff*]:
 $obs\text{-consistent}\ R12\ med21\ m1\ m2$
by (*auto simp add: obs-consistent-def R12-def med21-def m2-defs*)

Refinement result.

lemma $m2\text{-refines-}m1$ [*iff*]:
 $refines$
 $(R12 \cap$
 $(reach\ m1 \times$
 $(m2\text{-inv}9\text{-}M5 \cap m2\text{-inv}8a\text{-init-}M2 \cap m2\text{-inv}9a\text{-resp-}M3 \cap m2\text{-inv}8\text{-}M4 \cap$

```

      m2-inv4-M3  $\cap$  m2-inv4-M2  $\cap$  m2-inv3a-sesK-compr  $\cap$  m2-inv3-extrKey  $\cap$ 
      m2-inv2b-corrKey-leaked  $\cap$  m2-inv2-keys-for  $\cap$  m2-inv1-keys)))
    med21 m1 m2
  by (rule Refinement-using-invariants) (auto)

```

```

lemma m2-implements-m1 [iff]:
  implements med21 m1 m2
by (rule refinement-soundness) (auto)

```

3.10.6 Inherited and derived invariants

Show preservation of invariants $m1\text{-inv}2i\text{-serv}$ and $m1\text{-inv}2r\text{-serv}$ from $m1$.

```

lemma PO-m2-sat-m1-inv2i-serv [iff]: reach m2  $\subseteq$  m1-inv2i-serv
apply (rule-tac Pa=m1-inv2i-serv and Qa=m1-inv2i-serv and Q=m1-inv2i-serv
  in m2-implements-m1 [THEN [5] internal-invariant-translation])
apply (auto simp add: m2-loc-defs med21-def intro!: m1-inv2i-servI)
done

```

```

lemma PO-m2-sat-m1-inv2r-serv [iff]: reach m2  $\subseteq$  m1-inv2r-serv
by (rule-tac Pa=m1-inv2r-serv and Qa=m1-inv2r-serv and Q=m1-inv2r-serv
  in m2-implements-m1 [THEN [5] internal-invariant-translation])
  (fastforce simp add: m2-defs med21-def intro!: m1-inv2r-servI)+

```

Now we derive the additional invariants for the initiator and the responder (see above for the definitions).

```

lemma PO-m2-inv6-init-ikk [iff]: reach m2  $\subseteq$  m2-inv6-ikk-init
proof –
  have reach m2  $\subseteq$  m1-inv2i-serv  $\cap$  m2-inv5-ikk-sv by simp
  also have ...  $\subseteq$  m2-inv6-ikk-init by (blast intro!: m2-inv6-ikk-initI dest: m2-inv5-ikk-svD)
  finally show ?thesis .
qed

```

```

lemma PO-m2-inv6-resp-ikk [iff]: reach m2  $\subseteq$  m2-inv7-ikk-resp
proof –
  have reach m2  $\subseteq$  m1-inv2r-serv  $\cap$  m2-inv5-ikk-sv by simp
  also have ...  $\subseteq$  m2-inv7-ikk-resp by (blast intro!: m2-inv7-ikk-respI dest: m2-inv5-ikk-svD)
  finally show ?thesis .
qed

```

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 $\{(shrK\ C, A) \mid C\ A.\ A = C \vee A = Sv\}$

end

lemma *corrKey-shrK-bad* [*simp*]: *corrKey* = *shrK'bad*

by (*auto simp add: keySetup-def ltkeySetup-def corrKey-def*)

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 $m2\text{-step}1$

$m3\text{-step}1 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}] \Rightarrow m3\text{-trans}$

where

$m3\text{-step}1 \text{ Ra } A \text{ B } Na \equiv \{(s, s1)\}.$

— guards:

$Ra \notin \text{dom } (\text{runs } s) \wedge$ — Ra is fresh
 $Na = Ra\$na \wedge$ — generate nonce Na

— actions:

$s1 = s\{$
 $\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [])),$
 $IK := \text{insert } \{\{\text{Agent } A, \text{Agent } B, \text{Nonce } Na\}\} (IK \ s)$ — send msg 1
 $\}$

definition — by B , refines $m2\text{-step}2$

$m3\text{-step}2 :: [\text{rid-}t, \text{agent}, \text{agent}] \Rightarrow m3\text{-trans}$

where

$m3\text{-step}2 \text{ Rb } A \text{ B} \equiv \{(s, s1)\}.$

— guards:

$Rb \notin \text{dom } (\text{runs } s) \wedge$ — Rb is fresh

— actions:

— create responder thread

$s1 = s\{$
 $\text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], []))$
 $\}$

definition — by Server , refines $m2\text{-step}3$

$m3\text{-step}3 :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow m3\text{-trans}$

where

$m3\text{-step}3 \text{ Rs } A \text{ B } Na \text{ Kab} \equiv \{(s, s1)\}.$

— guards:

$Rs \notin \text{dom } (\text{runs } s) \wedge$ — fresh server run
 $Kab = \text{sesK } (Rs\$sk) \wedge$ — fresh session key

$\{\{\text{Agent } A, \text{Agent } B, \text{Nonce } Na\}\} \in IK \ s \wedge$ — recv msg 1

— actions:

— record session key and send messages 2 and 3

— note that last field in server record is for responder nonce

$s1 = s\{$
 $\text{runs} := (\text{runs } s)(Rs \mapsto (\text{Serv}, [A, B], [a\text{Non } Na])),$
 $IK := \{\{\text{Crypt } (\text{shrK } A) \{\{\text{Nonce } Na, \text{Agent } B, \text{Key } Kab\}\},$
 $\text{Crypt } (\text{shrK } B) \{\{\text{Key } Kab, \text{Agent } A\}\}\} \cup IK \ s$
 $\}$

}

definition — by A , refines $m2\text{-step4}$

$m3\text{-step4} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow m3\text{-trans}$

where

$m3\text{-step4 } Ra \ A \ B \ Na \ Kab \equiv \{(s, s1)\}.$

— guards:

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], []) \wedge$

$Na = Ra\$na \wedge$

$\text{Crypt } (\text{shr}K \ A) \ \{\!\! \{ \text{Nonce } Na, \text{Agent } B, \text{Key } Kab \} \in IK \ s \wedge$ — recv msg 2

— actions:

— record session key

$s1 = s\{$

$\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [aKey \ Kab]))$

$\}$

}

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
 $Nb = Rb\$nb \wedge$

$\text{Crypt } Kab \ \{\{Nonce \ Nb, Nonce \ Nb\} \in IK \ s \wedge$ — 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]:
 $init\ m3 \subseteq m3\text{-inv1-lkeysec}$
by (auto simp add: m3-defs m3-inv1-lkeysec-def)

lemma $PO\text{-}m3\text{-inv1-lkeysec-trans}$ [iff]:
 $\{m3\text{-inv1-lkeysec}\ trans\ m3\ \{>\ m3\text{-inv1-lkeysec}\}$
by (fastforce simp add: PO-hoare-defs m3-defs intro!: m3-inv1-lkeysecI)

lemma $PO\text{-}m3\text{-inv1-lkeysec}$ [iff]: $reach\ m3 \subseteq m3\text{-inv1-lkeysec}$
by (rule inv-rule-incr) (fast+)

Useful simplifier lemmas

lemma $m3\text{-inv1-lkeysec-for-parts}$ [simp]:
 $\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies Key\ (shrK\ C) \in parts\ (IK\ s) \longleftrightarrow C \in bad$
by auto

lemma $m3\text{-inv1-lkeysec-for-analz}$ [simp]:
 $\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies Key\ (shrK\ C) \in analz\ (IK\ s) \longleftrightarrow C \in bad$
by auto

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

$am\text{-}M1:$

$\{Agent\ A, Agent\ B, Nonce\ Na\} \in H$

$\Rightarrow Insec\ A\ B\ (Msg\ [aNon\ Na]) \in abs\text{-}msg\ H$

| $am\text{-}M2:$

$Crypt\ (shrK\ C)\ \{Nonce\ N, Agent\ B, Key\ K\} \in H$

$\Rightarrow Secure\ Sv\ C\ (Msg\ [aNon\ N, aAgt\ B, aKey\ K]) \in abs\text{-}msg\ H$

| $am\text{-}M3:$

$Crypt\ (shrK\ C)\ \{Key\ K, Agent\ A\} \in H$

$\Rightarrow Secure\ Sv\ C\ (Msg\ [aKey\ K, aAgt\ A]) \in abs\text{-}msg\ H$

| $am\text{-}M4:$

$Crypt\ K\ (Nonce\ N) \in H$

$\Rightarrow dAuth\ K\ (Msg\ [aNon\ N]) \in abs\text{-}msg\ H$

| $am\text{-}M5:$

$Crypt\ K\ \{Nonce\ N, Nonce\ N'\} \in H$

$\Rightarrow dAuth\ K\ (Msg\ [aNon\ N, aNon\ N']) \in abs\text{-}msg\ H$

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

$R23\text{-}keys :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$ — 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 $abs\text{-msg}$

declare $abs\text{-msg.intros}$ [intro!]
declare $abs\text{-msg.cases}$ [elim!]

lemma $abs\text{-msg-empty}$: $abs\text{-msg} \{\} = \{\}$
by (auto)

lemma $abs\text{-msg-Un}$ [simp]:
 $abs\text{-msg} (G \cup H) = abs\text{-msg} G \cup abs\text{-msg} H$
by (auto)

lemma $abs\text{-msg-mono}$ [elim]:
 $\llbracket m \in abs\text{-msg} G; G \subseteq H \rrbracket \Longrightarrow m \in abs\text{-msg} H$
by (auto)

lemma *abs-msg-insert-mono* [intro]:
 $\llbracket m \in \text{abs-msg } H \rrbracket \implies m \in \text{abs-msg } (\text{insert } m' H)$
by (*auto*)

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*:
 $\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; (s, t) \in R23\text{-non}; t \in m3\text{-inv1-lkeysec} \rrbracket$
 $\implies \text{abs-msg } (\text{synth } (\text{analz } (IK t))) \subseteq \text{fake ik0 } (\text{dom } (\text{runs } s)) (\text{chan } s)$

apply (*auto*)
— 9 subgoals, deal with replays first
prefer 2 apply (*blast*)
prefer 3 apply (*blast*)
prefer 4 apply (*blast*)
prefer 5 apply (*blast*)
— remaining 5 subgoals are real fakes
apply (*intro fake-StatCh fake-DynCh, auto simp add: R23-keys-simps dest: R23-non-dests*)
done

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*:
 $\{R23\}$
 $(m2\text{-step1 } Ra A B Na), (m3\text{-step1 } Ra A B Na)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)
(*auto*)

lemma *PO-m3-step2-refines-m2-step2*:
 $\{R23\}$
 $(m2\text{-step2 } Rb A B), (m3\text{-step2 } Rb A B)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)
(*auto*)

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 —
{ fix *s t*
assume *H*:
 $(s, t) \in R23\text{-msgs } (s, t) \in R23\text{-keys } (s, t) \in R23\text{-non } (s, t) \in R23\text{-pres}$
 $s \in m2\text{-inv3a-sesK-compr } t \in m3\text{-inv7a-sesK-compr } t \in m3\text{-inv1-lkeysec}$
 $Kab = \text{sesK } (Rs \$sk) \ Rs \notin \text{dom } (\text{runs } t)$
 $\{ \text{Agent } A, \text{Agent } B, \text{Nonce } Na \} \in \text{parts } (IK t)$
let *?s' =*

```

s(| runs := (runs s)(Rs ↦ (Serv, [A, B], [aNon Na])),
  chan := insert (Secure Sv A (Msg [aNon Na, aAgt B, aKey Kab]))
    (insert (Secure Sv B (Msg [aKey Kab, aAgt A])) (chan s)) |)
let ?t' =
  t(| runs := (runs t)(Rs ↦ (Serv, [A, B], [aNon Na])),
    IK := insert (Crypt (shrK A) { Nonce Na, Agent B, Key Kab })
      (insert (Crypt (shrK B) { Key Kab, Agent A }) (IK t)) |)
have (?s', ?t') ∈ R23-msgs using H
by (−) (rule R23-intros, auto)
moreover
have (?s', ?t') ∈ R23-keys using H
by (−) (rule R23-intros,
  auto simp add: m2-inv3a-sesK-compr-simps m3-inv7a-sesK-compr-simps,
  auto simp add: R23-keys-simps)
moreover
have (?s', ?t') ∈ R23-non using H
by (−) (rule R23-intros,
  auto simp add: m2-inv3a-sesK-compr-simps m3-inv7a-sesK-compr-simps
  dest: R23-non-dests)
moreover
have (?s', ?t') ∈ R23-pres using H
by (−) (rule R23-intros, auto)
moreover
note calculation
}
thus ?thesis
by (auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs)
qed

```

```

lemma PO-m3-step4-refines-m2-step4:
  {R23}
  (m2-step4 Ra A B Na Kab), (m3-step4 Ra A B Na Kab)
  {> R23}
by (auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros)
  (auto)

```

```

lemma PO-m3-step5-refines-m2-step5:
  {R23}
  (m2-step5 Rb A B Nb Kab), (m3-step5 Rb A B Nb Kab)
  {> R23}
by (auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros)
  (auto)

```

```

lemma PO-m3-step6-refines-m2-step6:
  {R23}
  (m2-step6 Ra A B Na Nb Kab), (m3-step6 Ra A B Na Nb Kab)
  {> R23}
by (auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros)
  (auto)

```

lemma *PO-m3-step7-refines-m2-step7*:
 $\{R23\}$
 $(m2\text{-step7 } Rb \ A \ B \ Nb \ Kab), (m3\text{-step7 } Rb \ A \ B \ Nb \ Kab)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)
(auto)

Intruder events.

lemma *PO-m3-leak-refines-m2-leak*:
 $\{R23\}$
 $(m2\text{-leak } Rs \ Ra \ Rb \ A \ B), (m3\text{-leak } Rs \ Ra \ Rb \ A \ B)$
 $\{>R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)
(auto simp add: R23-keys-simps dest: R23-non-dests)

lemma *PO-m3-DY-fake-refines-m2-fake*:
 $\{R23 \cap UNIV \times m3\text{-inv1-lkeysec}\}$
 $m2\text{-fake}, m3\text{-DY-fake}$
 $\{> R23\}$
apply (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)
 $del: abs\text{-msg.cases}$
apply (*auto intro: abs-msg-DY-subset-fakeable [THEN subsetD]*)
 $del: abs\text{-msg.cases}$
apply (*auto simp add: R23-keys-simps dest: R23-non-dests*)
done

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-step7-refines-m2-step7 PO-m3-leak-refines-m2-leak
PO-m3-DY-fake-refines-m2-fake

lemma *PO-m3-refines-init-m2 [iff]*:
 $init \ m3 \subseteq R23''(init \ m2)$
by (*auto simp add: R23-def m2-defs m3-defs intro!: R23-intros*)

lemma *PO-m3-refines-trans-m2 [iff]*:
 $\{R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv7a-sesK-compr} \cap m3\text{-inv1-lkeysec})\}$
 $(trans \ m2), (trans \ m3)$
 $\{> R23\}$
apply (*auto simp add: m3-def m3-trans-def m2-def m2-trans-def*)
apply (*blast intro!: PO-m3-trans-refines-m2-trans*)
done

lemma *PO-m3-observation-consistent [iff]*:
 $obs\text{-consistent } R23 \ med32 \ m2 \ m3$
by (*auto simp add: obs-consistent-def R23-def med32-def m2-defs m3-defs*)

Refinement result.

lemma *m3-refines-m2* [iff]:
refines ($R23 \cap m2\text{-inv}3a\text{-ses}K\text{-compr} \times m3\text{-inv}1\text{-lkeysec}$)
med32 m2 m3

proof –
have $R23 \cap m2\text{-inv}3a\text{-ses}K\text{-compr} \times UNIV \subseteq UNIV \times m3\text{-inv}7a\text{-ses}K\text{-compr}$
by (*auto simp add: R23-def R23-keys-simps intro!: m3-inv7a-sesK-comprI*)
thus *?thesis*
by (–) (*rule Refinement-using-invariants, auto*)
qed

lemma *m3-implements-m2* [iff]:
implements med32 m2 m3
by (*rule refinement-soundness*) (*auto*)

3.11.7 Inherited invariants

inv4 (derived): Key secrecy for initiator

definition

m3-inv4-ikk-init :: *m3-state set*

where

$m3\text{-inv}4\text{-ikk-init} \equiv \{s. \forall Ra\ K\ A\ B\ al.$
runs $s\ Ra = \text{Some}\ (Init, [A, B], aKey\ K\ \# al) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$
Key $K \in \text{analz}\ (IK\ s) \longrightarrow$
 $(\exists Nb'. (K, Ra\ \$\ na, Nb') \in leak\ s)$
 $\}$

lemmas $m3\text{-inv}4\text{-ikk-init}I = m3\text{-inv}4\text{-ikk-init-def}\ [THEN\ \text{setc-def-to-intro},\ \text{rule-format}]$
lemmas $m3\text{-inv}4\text{-ikk-init}E\ [elim] = m3\text{-inv}4\text{-ikk-init-def}\ [THEN\ \text{setc-def-to-elim},\ \text{rule-format}]$
lemmas $m3\text{-inv}4\text{-ikk-init}D = m3\text{-inv}4\text{-ikk-init-def}\ [THEN\ \text{setc-def-to-dest},\ \text{rule-format},\ \text{rotated}\ 1]$

lemma *PO-m3-inv4-ikk-init: reach m3* $\subseteq m3\text{-inv}4\text{-ikk-init}$

proof (*rule INV-from-Refinement-using-invariants [OF m3-refines-m2]*)

show $\text{Range}\ (R23 \cap m2\text{-inv}3a\text{-ses}K\text{-compr} \times m3\text{-inv}1\text{-lkeysec}$
 $\cap m2\text{-inv}6\text{-ikk-init} \times UNIV)$

$\subseteq m3\text{-inv}4\text{-ikk-init}$

by (*auto simp add: R23-def R23-pres-def R23-keys-simps intro!: m3-inv4-ikk-initI*)

qed *auto*

inv5 (derived): Key secrecy for responder

definition

m3-inv5-ikk-resp :: *m3-state set*

where

$m3\text{-inv}5\text{-ikk-resp} \equiv \{s. \forall Rb\ K\ A\ B\ al.$
runs $s\ Rb = \text{Some}\ (Resp, [A, B], aKey\ K\ \# al) \longrightarrow A \in good \longrightarrow B \in good \longrightarrow$
Key $K \in \text{analz}\ (IK\ s) \longrightarrow$
 $K \in \text{Domain}\ (leak\ s)$
 $\}$

lemmas $m3\text{-inv}5\text{-ikk-resp}I = m3\text{-inv}5\text{-ikk-resp-def}\ [THEN\ \text{setc-def-to-intro},\ \text{rule-format}]$

lemmas $m3\text{-inv}5\text{-ikk-resp}E\ [elim] = m3\text{-inv}5\text{-ikk-resp-def}\ [THEN\ \text{setc-def-to-elim},\ \text{rule-format}]$

lemmas $m3\text{-inv5}\text{-ikk}\text{-resp}D = m3\text{-inv5}\text{-ikk}\text{-resp}\text{-def}$ [THEN setc-def-to-dest, rule-format, rotated 1]

lemma $PO\text{-}m3\text{-inv4}\text{-ikk}\text{-resp}$: reach $m3 \subseteq m3\text{-inv5}\text{-ikk}\text{-resp}$

proof (rule INV-from-Refinement-using-invariants [OF $m3\text{-refines}\text{-}m2$])

show Range ($R23 \cap m2\text{-inv3a}\text{-ses}K\text{-compr} \times m3\text{-inv1}\text{-lkeysec}$
 $\cap m2\text{-inv7}\text{-ikk}\text{-resp} \times UNIV$)
 $\subseteq m3\text{-inv5}\text{-ikk}\text{-resp}$

by (auto simp add: R23-def R23-pres-def R23-keys-simps intro!: $m3\text{-inv5}\text{-ikk}\text{-resp}I$)
 (elim $m2\text{-inv7}\text{-ikk}\text{-resp}E$, auto)

qed auto

end

3.12 Needham-Schroeder Shared Key (L3)

theory $m3\text{-nssk}$ imports $m2\text{-nssk}$../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 $ltkeySetup\text{-}def$: $ltkeySetup' \equiv \{(shrK\ C, A) \mid C\ A.\ A = C \vee A = Sv\}$

end

lemma $corrKey\text{-}shrK\text{-}bad$ [simp]: $corrKey = shrK'bad$

by (auto simp add: $keySetup\text{-}def$ $ltkeySetup\text{-}def$ $corrKey\text{-}def$)

3.12.2 State

The secure channels are star-shaped to/from the server. Therefore, we have only one agent in the relation.

record $m3\text{-state} = m1\text{-state} +$

$IK :: msg\ set$

— intruder knowledge

Observable state: agent's local state.

type-synonym

$$m3\text{-obs} = m2\text{-obs}$$
definition

$$m3\text{-obs} :: m3\text{-state} \Rightarrow m3\text{-obs} \text{ where}$$

$$m3\text{-obs } s \equiv \langle \langle \text{runs} = \text{runs } s, \text{leak} = \text{leak } s \rangle \rangle$$
type-synonym

$$m3\text{-pred} = m3\text{-state set}$$
type-synonym

$$m3\text{-trans} = (m3\text{-state} \times m3\text{-state}) \text{ set}$$

3.12.3 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:

$$Ra \notin \text{dom } (\text{runs } s) \wedge \quad \text{— } Ra \text{ is fresh}$$

$$Na = Ra\$na \wedge \quad \text{— generate nonce } Na$$

— actions:

$$s1 = s \langle$$

$$\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [])),$$

$$IK := \text{insert } \{\text{Agent } A, \text{Agent } B, \text{Nonce } Na\} (IK \ s) \quad \text{— send msg 1}$$

$$\rangle$$

$$\}$$
definition — by B , refines $m2\text{-step2}$

$$m3\text{-step2} :: [\text{rid-}t, \text{agent}, \text{agent}] \Rightarrow m3\text{-trans}$$
where

$$m3\text{-step2 } Rb \ A \ B \equiv \{(s, s1).\}$$

— guards:

$$Rb \notin \text{dom } (\text{runs } s) \wedge \quad \text{— } Rb \text{ is fresh}$$

— actions:

— create responder thread

$$s1 = s \langle$$

$$\text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], []))$$

$$\rangle$$

$$\}$$
definition — by $Server$, refines $m2\text{-step3}$

$$m3\text{-step3} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{nonce}, \text{key}] \Rightarrow m3\text{-trans}$$
where

$$m3\text{-step3 } Rs \ A \ B \ Na \ Kab \equiv \{(s, s1).\}$$

— guards:

$R_s \notin \text{dom}(\text{runs } s) \wedge$ — fresh server run
 $K_{ab} = \text{sesK}(R_s \$ sk) \wedge$ — fresh session key

$\{\text{Agent } A, \text{Agent } B, \text{Nonce } Na\} \in IK\ s \wedge$ — rcv msg 1

— actions:
 — record session key and send messages 2 and 3
 — note that last field in server record is for responder nonce
 $s1 = s\langle$
 $\text{runs} := (\text{runs } s)(R_s \mapsto (\text{Serv}, [A, B], [aNon\ Na])),$
 $IK := \text{insert}$
 $(\text{Crypt}(\text{shrK } A)$
 $\{\text{Nonce } Na, \text{Agent } B, \text{Key } K_{ab},$
 $\text{Crypt}(\text{shrK } B) \{\text{Key } K_{ab}, \text{Agent } A\}\})$
 $(IK\ s)$
 \rangle
 $\}$

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

$m3\text{-step4 } Ra\ A\ B\ Na\ K_{ab}\ X \equiv \{(s, s1)\}.$

— guards:
 $\text{runs } s\ Ra = \text{Some}(\text{Init}, [A, B], []) \wedge$
 $Na = Ra \$ na \wedge$

$\text{Crypt}(\text{shrK } A) \{\text{Nonce } Na, \text{Agent } B, \text{Key } K_{ab}, X\} \in IK\ s \wedge$ — rcv msg 2

— actions:
 — record session key, and forward X
 $s1 = s\langle$
 $\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [aKey\ K_{ab}])),$
 $IK := \text{insert } X\ (IK\ s)$
 \rangle
 $\}$

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

$m3\text{-step5 } Rb\ A\ B\ Nb\ K_{ab} \equiv \{(s, s1)\}.$

— guards:
 $\text{runs } s\ Rb = \text{Some}(\text{Resp}, [A, B], []) \wedge$
 $Nb = Rb \$ nb \wedge$

$\text{Crypt}(\text{shrK } B) \{\text{Key } K_{ab}, \text{Agent } A\} \in IK\ s \wedge$ — rcv msg 3

— actions:
 — record session key
 $s1 = s\langle$
 $\text{runs} := (\text{runs } s)(Rb \mapsto (\text{Resp}, [A, B], [aKey\ K_{ab}])),$
 $IK := \text{insert}(\text{Crypt } K_{ab}(\text{Nonce } Nb))(IK\ s)$
 \rangle

$m3-DY-fake :: m3-trans$

where

$m3-DY-fake \equiv \{(s, s1)\}.$

— actions:
 $s1 = s(|$
 $IK := synth (analz (IK s))$
 $|)$
 $\}$

3.12.4 Transition system

definition

$m3-init :: m3-state\ set$

where

$m3-init \equiv \{ (|$
 $runs = Map.empty,$
 $leak = shrK'bad \times \{undefined\} \times \{undefined\},$
 $IK = Key'shrK'bad$
 $|) \}$

definition

$m3-trans :: (m3-state \times m3-state)\ set$ **where**

$m3-trans \equiv (\bigcup Ra\ Rb\ Rs\ A\ B\ Na\ Nb\ Kab\ X.$

$m3-step1\ Ra\ A\ B\ Na \cup$

$m3-step2\ Rb\ A\ B \cup$

$m3-step3\ Rs\ A\ B\ Na\ Kab \cup$

$m3-step4\ Ra\ A\ B\ Na\ Kab\ X \cup$

$m3-step5\ Rb\ A\ B\ Nb\ Kab \cup$

$m3-step6\ Ra\ A\ B\ Na\ Nb\ Kab \cup$

$m3-step7\ Rb\ A\ B\ Nb\ Kab \cup$

$m3-leak\ Rs\ Ra\ Rb\ A\ B \cup$

$m3-DY-fake \cup$

Id

)

definition

$m3 :: (m3-state, m3-obs)\ spec$ **where**

$m3 \equiv (|$

$init = m3-init,$

$trans = m3-trans,$

$obs = m3-obs$

|)

lemmas $m3-defs =$

$m3-def\ m3-init-def\ m3-trans-def\ m3-obs-def$

$m3-step1-def\ m3-step2-def\ m3-step3-def\ m3-step4-def\ m3-step5-def$

$m3-step6-def\ m3-step7-def\ m3-leak-def\ m3-DY-fake-def$

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

$m3\text{-inv1-lkeysec} \equiv \{s. \forall C.$
 $(Key (shrK C) \in parts (IK s) \longrightarrow C \in bad) \wedge$
 $(C \in bad \longrightarrow Key (shrK C) \in IK s)$
 $\}$

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*]:

$init\ m3 \subseteq m3\text{-inv1-lkeysec}$

by (*auto simp add: m3-defs m3-inv1-lkeysec-def*)

lemma *PO-m3-inv1-lkeysec-trans* [*iff*]:

$\{m3\text{-inv1-lkeysec}\} trans\ m3 \{>\ m3\text{-inv1-lkeysec}\}$

by (*fastforce simp add: PO-hoare-defs m3-defs intro!: m3-inv1-lkeysecI dest: Body*)

lemma *PO-m3-inv1-lkeysec* [*iff*]: $reach\ m3 \subseteq m3\text{-inv1-lkeysec}$

by (*rule inv-rule-incr*) (*fast+*)

Useful simplifier lemmas

lemma *m3-inv1-lkeysec-for-parts* [*simp*]:

$\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies Key (shrK C) \in parts (IK s) \longleftrightarrow C \in bad$

by *auto*

lemma *m3-inv1-lkeysec-for-analz* [*simp*]:

$\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies Key (shrK C) \in analz (IK s) \longleftrightarrow C \in bad$

by *auto*

inv2: Ticket shape for honestly encrypted M2

definition

m3-inv2-ticket :: *m3-state set*

where

$m3\text{-inv2-ticket} \equiv \{s. \forall A\ B\ N\ K\ X.$

$A \notin bad \longrightarrow$

$Crypt (shrK A) \llbracket Nonce\ N, Agent\ B, Key\ K, X \rrbracket \in parts (IK s) \longrightarrow$

$X = Crypt (shrK B) \llbracket Key\ K, Agent\ A \rrbracket \wedge K \in range\ sesK$

}

lemmas *m3-inv2-ticketI* =
m3-inv2-ticket-def [THEN setc-def-to-intro, rule-format]
lemmas *m3-inv2-ticketE* [elim] =
m3-inv2-ticket-def [THEN setc-def-to-elim, rule-format]
lemmas *m3-inv2-ticketD* =
m3-inv2-ticket-def [THEN setc-def-to-dest, rule-format, rotated -1]

Invariance proof.

lemma *PO-m3-inv2-ticket-init* [iff]:
init m3 \subseteq *m3-inv2-ticket*
by (*auto simp add: m3-defs intro!: m3-inv2-ticketI*)

lemma *PO-m3-inv2-ticket-trans* [iff]:
 $\{m3-inv2-ticket \cap m3-inv1-lkeysec\}$ *trans m3* $\{> m3-inv2-ticket\}$
apply (*auto simp add: PO-hoare-defs m3-defs intro!: m3-inv2-ticketI*)
apply (*auto dest: m3-inv2-ticketD*)
— 2 subgoals, from step4 [?]
apply (*drule Body [where H=IK s for s], drule parts-cut,*
auto dest: m3-inv2-ticketD)
done

lemma *PO-m3-inv2-ticket* [iff]: *reach m3* \subseteq *m3-inv2-ticket*
by (*rule inv-rule-incr*) (*auto del: subsetI*)

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-inv3-sesK-compr :: *m3-state set*

where

m3-inv3-sesK-compr \equiv $\{s. \forall K KK.$

$KK \subseteq \text{range sesK} \longrightarrow$

$(\text{Key } K \in \text{analz } (\text{Key } KK \cup (\text{IK } s))) = (K \in KK \vee \text{Key } K \in \text{analz } (\text{IK } s))$

$\}$

lemmas *m3-inv3-sesK-comprI* = *m3-inv3-sesK-compr-def* [THEN setc-def-to-intro, rule-format]
lemmas *m3-inv3-sesK-comprE* = *m3-inv3-sesK-compr-def* [THEN setc-def-to-elim, rule-format]
lemmas *m3-inv3-sesK-comprD* = *m3-inv3-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-inv3-sesK-compr-simps* =
m3-inv3-sesK-comprD
m3-inv3-sesK-comprD [where $KK = \{Kab\}$ for Kab , simplified]
m3-inv3-sesK-comprD [where $KK = \text{insert } Kab \text{ } KK$ for $Kab \text{ } KK$, simplified]

insert-commute-Key

Invariance proof.

lemma *PO-m3-inv3-sesK-compr-step4*:

$\{m3\text{-inv3-sesK-compr} \cap m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec}\}$
 $m3\text{-step4 } Ra \ A \ B \ Na \ Kab \ X$
 $\{> m3\text{-inv3-sesK-compr}\}$

proof –

{ **fix** $K \ KK \ s$

assume H :

$s \in m3\text{-inv1-lkeysec} \ s \in m3\text{-inv3-sesK-compr} \ s \in m3\text{-inv2-ticket}$

$runs \ s \ Ra = Some \ (Init, [A, B], [])$

$Na = Ra\$na$

$KK \subseteq range \ sesK$

$Crypt \ (shrK \ A) \ \{\!| \ Nonce \ Na, \ Agent \ B, \ Key \ Kab, \ X \!\} \in \ analz \ (IK \ s)$

have

$(Key \ K \in \ analz \ (insert \ X \ (Key \ ' \ KK \cup \ IK \ s))) =$

$(K \in \ KK \ \vee \ Key \ K \in \ analz \ (insert \ X \ (IK \ s)))$

proof (*cases* $A \in \ bad$)

case *True*

with H **have** $X \in \ analz \ (IK \ s)$ **by** (*auto* *dest!*: *Decrypt*)

moreover

with H **have** $X \in \ analz \ (Key \ ' \ KK \cup \ IK \ s)$

by (*auto* *intro*: *analz-monotonic*)

ultimately show *?thesis* **using** H

by (*auto* *simp* *add*: *m3-inv3-sesK-compr-simps* *analz-insert-eq*)

next

case *False* **thus** *?thesis* **using** H

by (*fastforce* *simp* *add*: *m3-inv3-sesK-compr-simps*

dest!: *m3-inv2-ticketD* [*OF* *analz-into-parts*])

qed

}

thus *?thesis*

by (*auto* *simp* *add*: *PO-hoare-defs* *m3-defs* *intro!*: *m3-inv3-sesK-comprI* *dest!*: *analz-Inj-IK*)

qed

All together now.

lemmas *PO-m3-inv3-sesK-compr-trans-lemmas* =

PO-m3-inv3-sesK-compr-step4

lemma *PO-m3-inv3-sesK-compr-init* [*iff*]:

$init \ m3 \subseteq m3\text{-inv3-sesK-compr}$

by (*auto* *simp* *add*: *m3-defs* *intro!*: *m3-inv3-sesK-comprI*)

lemma *PO-m3-inv3-sesK-compr-trans* [*iff*]:

$\{m3\text{-inv3-sesK-compr} \cap m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec}\}$

$trans \ m3$

$\{> m3\text{-inv3-sesK-compr}\}$

by (*auto* *simp* *add*: *m3-def* *m3-trans-def* *intro!*: *PO-m3-inv3-sesK-compr-trans-lemmas*)

(*auto* *simp* *add*: *PO-hoare-defs* *m3-defs* *m3-inv3-sesK-compr-simps* *intro!*: *m3-inv3-sesK-comprI*)

lemma *PO-m3-inv3-sesK-compr* [*iff*]: $reach \ m3 \subseteq m3\text{-inv3-sesK-compr}$

by (*rule-tac* $J=m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec}$ **in** *inv-rule-incr*) (*auto*)

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-obs* ⇒ *m2-obs* **where**
med32 ≡ *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-keysD* = *R23-keys-def* [*THEN rel-def-to-dest, simplified, rule-format, rotated 2*]

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-nonD* = *R23-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*

Further lemmas: general lemma for simplifier and different instantiations.

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* {} = {}
by (*auto*)

lemma *abs-msg-Un* [*simp*]:
abs-msg (*G* ∪ *H*) = *abs-msg* *G* ∪ *abs-msg* *H*
by (*auto*)

lemma *abs-msg-mono* [*elim*]:
 $\llbracket m \in \text{abs-msg } G; G \subseteq H \rrbracket \Longrightarrow m \in \text{abs-msg } H$
by (*auto*)

lemma *abs-msg-insert-mono* [*intro*]:
 $\llbracket m \in \text{abs-msg } H \rrbracket \Longrightarrow m \in \text{abs-msg } (\text{insert } m' H)$
by (*auto*)

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*:
 $\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; (s, t) \in R23\text{-non}; t \in m3\text{-inv1-lkeysec} \rrbracket$
 $\Longrightarrow \text{abs-msg } (\text{synth } (\text{analz } (IK t))) \subseteq \text{fake ik0 } (\text{dom } (\text{runs } s)) (\text{chan } s)$
apply (*auto*)
— 9 subgoals, deal with replays first
prefer 2 apply (*blast*)
prefer 3 apply (*blast*)
prefer 4 apply (*blast*)
prefer 5 apply (*blast*)
— remaining 5 subgoals are real fakes
apply (*intro fake-StatCh fake-DynCh, auto dest: R23-dests*)
done

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*:
 $\{R23\}$
 $(m2\text{-step1 } Ra A B Na), (m3\text{-step1 } Ra A B Na)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)
(*auto*)

lemma *PO-m3-step2-refines-m2-step2*:
 $\{R23\}$
 $(m2\text{-step2 } Rb A B), (m3\text{-step2 } Rb A B)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros*)

lemma *PO-m3-step3-refines-m2-step3*:
 $\{R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv3-sesK-compr} \cap m3\text{-inv1-lkeysec})\}$
 $(m2\text{-step3 } Rs A B Na Kab), (m3\text{-step3 } Rs A B Na Kab)$
 $\{> R23\}$
proof —
 $\{ \text{fix } s t$
assume *H*:
 $(s, t) \in R23\text{-msgs } (s, t) \in R23\text{-keys } (s, t) \in R23\text{-non } (s, t) \in R23\text{-pres}$
 $s \in m2\text{-inv3a-sesK-compr } t \in m3\text{-inv3-sesK-compr } t \in m3\text{-inv1-lkeysec}$

```

Kab = sesK (Rs$sk) Rs ∉ dom (runs t)
⊥ Agent A, Agent B, Nonce Na ⊥ ∈ parts (IK t)
let ?s' =
  s( runs := (runs s)(Rs ↦ (Serv, [A, B], [aNon Na])),
    chan := insert (Secure Sv A (Msg [aNon Na, aAgt B, aKey Kab]))
      (insert (Secure Sv B (Msg [aKey Kab, aAgt A])) (chan s)) )
let ?t' =
  t( runs := (runs t)(Rs ↦ (Serv, [A, B], [aNon Na])),
    IK := insert
      (Crypt (shrK A)
        ⊥ Nonce Na, Agent B, Key Kab,
        Crypt (shrK B) ⊥ Key Kab, Agent A ⊥) )
have (?s', ?t') ∈ R23-msgs using H
by (−) (rule R23-intros, auto)
moreover
have (?s', ?t') ∈ R23-keys using H
by (−) (rule R23-intros,
  auto simp add: m2-inv3a-sesK-compr-simps m3-inv3-sesK-compr-simps dest: R23-keys-dests)
moreover
have (?s', ?t') ∈ R23-non using H
by (−) (rule R23-intros,
  auto simp add: m2-inv3a-sesK-compr-simps m3-inv3-sesK-compr-simps dest: R23-non-dests)
moreover
have (?s', ?t') ∈ R23-pres using H
by (−) (rule R23-intros, auto)
moreover
note calculation
}
thus ?thesis
by (auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs)
qed

```

lemma *PO-m3-step4-refines-m2-step4*:

```

{R23 ∩ (m2-inv3b-sesK-compr-non)
  × (m3-inv3-sesK-compr ∩ m3-inv2-ticket ∩ m3-inv1-lkeysec)}
(m2-step4 Ra A B Na Kab), (m3-step4 Ra A B Na Kab X)
{> R23}

```

proof –

```

{
  fix s t
  assume H:
    (s, t) ∈ R23-msgs (s, t) ∈ R23-keys (s, t) ∈ R23-non (s, t) ∈ R23-pres
    s ∈ m2-inv3b-sesK-compr-non
    t ∈ m3-inv3-sesK-compr t ∈ m3-inv2-ticket t ∈ m3-inv1-lkeysec
    runs t Ra = Some (Init, [A, B], [])
    Na = Ra$na
    Crypt (shrK A) ⊥ Nonce Na, Agent B, Key Kab, X ⊥ ∈ analz (IK t)
  let ?s' = s( runs := (runs s)(Ra ↦ (Init, [A, B], [aKey Kab])) )
  and ?t' = t( runs := (runs t)(Ra ↦ (Init, [A, B], [aKey Kab])),

```

$IK := \text{insert } X (IK t) \Downarrow$

from H
have $\text{Secure Sv } A (Msg [aNon Na, aAgt B, aKey Kab]) \in \text{chan } s$ **by** auto
moreover
have $X \in \text{parts } (IK t)$ **using** H **by** $(\text{auto dest!}: \text{Body MPair-parts})$
hence $(?s', ?t') \in R23\text{-msgs}$ **using** H **by** $(\text{auto intro!}: R23\text{-intros})$
moreover
have $(?s', ?t') \in R23\text{-keys}$
proof (cases)
 assume $A \in \text{bad}$
 with H **have** $X \in \text{analz } (IK t)$ **by** $(-)$ $(\text{drule Decrypt, auto})$
 with H **show** $?thesis$
 by $(-)$ $(\text{rule } R23\text{-intros, auto dest!}: \text{analz-cut intro}: \text{analz-monotonic})$
next
assume $A \notin \text{bad}$ **show** $?thesis$
proof $-$
 note H
 moreover
 with $\langle A \notin \text{bad} \rangle$
 have $X = \text{Crypt } (shrK B) \{Key Kab, Agent A\} \wedge Kab \in \text{range sesK}$
 by $(\text{auto dest!}: m3\text{-inv2-ticketD})$
 moreover
 { assume $H1: Key (shrK B) \in \text{analz } (IK t)$
 have $aKey Kab \in \text{extr ik0 } (\text{chan } s)$
 proof $-$
 note calculation
 moreover
 hence $\text{Secure Sv } B (Msg [aKey Kab, aAgt A]) \in \text{chan } s$
 by $(-)$ $(\text{drule analz-into-parts, drule Body, elim MPair-parts, auto})$
 ultimately
 show $?thesis$ **using** $H1$ **by** auto
 qed
 }
 ultimately show $?thesis$
 by $(-)$ $(\text{rule } R23\text{-intros, auto simp add}: m3\text{-inv3-sesK-compr-simps})$
 qed
qed
moreover
have $(?s', ?t') \in R23\text{-non}$
proof (cases)
 assume $A \in \text{bad}$
 hence $X \in \text{analz } (IK t)$ **using** H **by** $(-)$ $(\text{drule Decrypt, auto})$
 thus $?thesis$ **using** H
 by $(-)$ $(\text{rule } R23\text{-intros, auto dest!}: \text{analz-cut intro}: \text{analz-monotonic})$
next
 assume $A \notin \text{bad}$
 hence $X = \text{Crypt } (shrK B) \{Key Kab, Agent A\} \wedge Kab \in \text{range sesK}$ **using** H
 by $(\text{auto dest!}: m3\text{-inv2-ticketD})$
 thus $?thesis$ **using** H
 by $(-)$ $(\text{rule } R23\text{-intros, auto simp add}: m2\text{-inv3b-sesK-compr-non-simps } m3\text{-inv3-sesK-compr-simps } \text{dest}: R23\text{-non-dests})$
qed

```

moreover
  have (?s', ?t') ∈ R23-pres using H by (auto intro!: R23-intros)
moreover
  note calculation
}
thus ?thesis
by (auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs dest!: analz-Inj-IK)
qed

```

```

lemma PO-m3-step5-refines-m2-step5:
  {R23}
  (m2-step5 Rb A B Nb Kab), (m3-step5 Rb A B Nb Kab)
  {> R23}
by (auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros)
  (auto)

```

```

lemma PO-m3-step6-refines-m2-step6:
  {R23}
  (m2-step6 Ra A B Na Nb Kab), (m3-step6 Ra A B Na Nb Kab)
  {> R23}
by (auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros)
  (auto)

```

```

lemma PO-m3-step7-refines-m2-step7:
  {R23}
  (m2-step7 Rb A B Nb Kab), (m3-step7 Rb A B Nb Kab)
  {> R23}
by (auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros)

```

Intruder events.

```

lemma PO-m3-leak-refines-m2-leak:
  {R23}
  m2-leak Rs Ra Rb A B, m3-leak Rs Ra Rb A B
  {> R23}
by (auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros)
  (auto dest: R23-dests)

```

```

lemma PO-m3-DY-fake-refines-m2-fake:
  {R23 ∩ UNIV × m3-inv1-lkeysec}
  m2-fake, m3-DY-fake
  {> R23}
apply (auto simp add: PO-rhoare-defs R23-def m2-defs m3-defs intro!: R23-intros
  del: abs-msg.cases)
apply (auto intro: abs-msg-DY-subset-fakeable [THEN subsetD]
  del: abs-msg.cases)
apply (auto dest: R23-dests)
done

```

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-step7-refines-m2-step7 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)
by (auto simp add: *R23-def m2-defs m3-defs intro!*: *R23-intros*)

lemma *PO-m3-refines-trans-m2* [iff]:
 $\{R23 \cap (m2-inv3a-sesK-compr \cap m2-inv3b-sesK-compr-non)$
 $\times (m3-inv3-sesK-compr \cap m3-inv2-ticket \cap m3-inv1-lkeysec)\}$
 $(trans\ m2), (trans\ m3)$
 $\{> R23\}$
apply (auto simp add: *m3-def m3-trans-def m2-def m2-trans-def*)
apply (blast intro!: *PO-m3-trans-refines-m2-trans*)
done

lemma *PO-m3-observation-consistent* [iff]:
obs-consistent R23 med32 m2 m3
by (auto simp add: *obs-consistent-def R23-def med32-def m2-defs m3-defs*)

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
by (rule *Refinement-using-invariants*) (auto)

lemma *m3-implements-m2* [iff]:
implements med32 m2 m3
by (rule *refinement-soundness*) (auto)

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 (rule *INV-from-Refinement-using-invariants* [*OF m3-refines-m2*])

```

show Range (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)
   $\cap$  m2-inv6-ikk-init  $\times$  UNIV)
   $\subseteq$  m3-inv4-ikk-init
by (auto simp add: R23-def R23-pres-def intro!: m3-inv4-ikk-initI)
  (elim m2-inv6-ikk-initE, auto dest: R23-keys-dests)
qed auto

```

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 $s Rb = \text{Some } (Resp, [A, B], aKey K \# al) \longrightarrow A \in \text{good} \longrightarrow B \in \text{good} \longrightarrow$
 Key $K \in \text{analz } (IK s) \longrightarrow$
 $K \in \text{Domain } (\text{leak } s)$
 $\}$

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 (rule INV-from-Refinement-using-invariants [OF *m3-refines-m2*])

show Range (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)
 \cap m2-inv7-ikk-resp \times UNIV)
 \subseteq *m3-inv5-ikk-resp*

by (auto simp add: R23-def R23-pres-def intro!: *m3-inv5-ikk-respI*)
 (elim m2-inv7-ikk-respE, auto dest: R23-keys-dests)

qed auto

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

$'x m1-trans = ('x m1-state-scheme \times 'x m1-state-scheme) set$

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*]: $is-len' \equiv 1::nat$

definition $rs-len-def$ [*simp*]: $rs-len' \equiv 1::nat$

end

3.13.2 Events

definition — by A , refines $m1x-step1$

$m1-step1 :: [rid-t, agent, agent] \Rightarrow 'x m1-trans$

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 $m1x-step3$

$m1-step3 :: [rid-t, agent, agent, key, time] \Rightarrow 'x m1-trans$

where

$m1-step3 Rs A B Kab Ts \equiv \{(s, s')\}.$

— new guards:

$Ts = clk s \wedge$ — fresh timestamp

— rest as before:

$(s, s') \in m1a-step3 Rs A B Kab [aNum Ts]$

}

definition — by A , refines $m1x\text{-step5}$
 $m1\text{-step4} :: [rid\text{-}t, agent, agent, key, time] \Rightarrow 'x\ m1\text{-trans}$

where

$m1\text{-step4}\ Ra\ A\ B\ Kab\ Ts \equiv \{(s, s')\}.$
 — new guards:
 $clk\ s < Ts + Ls \wedge$ — ensure session key recentness
 — rest as before
 $(s, s') \in m1a\text{-step4}\ Ra\ A\ B\ Kab\ [aNum\ Ts]$
 }

definition — by B , refines $m1x\text{-step4}$
 $m1\text{-step5} :: [rid\text{-}t, agent, agent, key, time] \Rightarrow 'x\ m1\text{-trans}$

where

$m1\text{-step5}\ Rb\ A\ B\ Kab\ Ts \equiv \{(s, s')\}.$
 — new guards:
 — ensure freshness of session key
 $clk\ s < Ts + Ls \wedge$
 — rest as before
 $(s, s') \in m1a\text{-step5}\ Rb\ A\ B\ Kab\ [aNum\ Ts]$
 }

definition — refines $skip$
 $m1\text{-tick} :: time \Rightarrow 'x\ m1\text{-trans}$

where

$m1\text{-tick}\ T \equiv \{(s, s')\}.$
 $s' = s(\ clk := clk\ s + T)$
 }

definition — by attacker, refines $m1x\text{-leak}$
 $m1\text{-leak} :: [rid\text{-}t] \Rightarrow 'x\ m1\text{-trans}$

where

$m1\text{-leak} \equiv m1a\text{-leak}$

3.13.3 Specification

definition

$m1\text{-init} :: unit\ m1\text{-pred}$

where

$m1\text{-init} \equiv \{ (\ runs = Map.empty, leak = corrKey, clk = 0) \}$

definition

$m1\text{-trans} :: 'x\ m1\text{-trans}$ **where**
 $m1\text{-trans} \equiv (\bigcup A\ B\ Ra\ Rb\ Rs\ Kab\ Ts\ T.$
 $m1\text{-step1}\ Ra\ A\ B \cup$
 $m1\text{-step2}\ Rb\ A\ B \cup$
 $m1\text{-step3}\ Rs\ A\ B\ Kab\ Ts \cup$
 $m1\text{-step4}\ Ra\ A\ B\ Kab\ Ts \cup$
 $m1\text{-step5}\ Rb\ A\ B\ Kab\ Ts \cup$
 $m1\text{-tick}\ T \cup$

$m1\text{-leak } Rs \cup$
 Id
 $)$

definition

$m1 :: (m1\text{-state}, m1\text{-obs}) \text{ spec where}$
 $m1 \equiv \langle$
 $init = m1\text{-init},$
 $trans = m1\text{-trans},$
 $obs = id$
 \rangle

lemmas $m1\text{-loc-defs} =$

$m1\text{-def } m1\text{-init-def } m1\text{-trans-def}$
 $m1\text{-step1-def } m1\text{-step2-def } m1\text{-step3-def } m1\text{-step4-def } m1\text{-step5-def}$
 $m1\text{-leak-def } m1\text{-tick-def}$

lemmas $m1\text{-defs} = m1\text{-loc-defs } m1a\text{-defs}$

lemma $m1\text{-obs-id [simp]: obs } m1 = id$
by $(simp \text{ add: } m1\text{-def})$

3.13.4 Invariants

inv0: Finite domain

There are only finitely many runs. This is needed to establish the responder/initiator agreement.

definition

$m1\text{-inv0-fin} :: 'x \text{ } m1\text{-pred}$

where

$m1\text{-inv0-fin} \equiv \{s. \text{finite } (\text{dom } (\text{runs } s))\}$

lemmas $m1\text{-inv0-finI} = m1\text{-inv0-fin-def [THEN setc-def-to-intro, rule-format]$

lemmas $m1\text{-inv0-finE [elim] = m1\text{-inv0-fin-def [THEN setc-def-to-elim, rule-format]$

lemmas $m1\text{-inv0-finD} = m1\text{-inv0-fin-def [THEN setc-def-to-dest, rule-format]$

Invariance proofs.

lemma $PO\text{-}m1\text{-inv0-fin-init [iff]:$

$init \ m1 \subseteq m1\text{-inv0-fin}$

by $(auto \ simp \ \text{add: } m1\text{-defs intro!: } m1\text{-inv0-finI})$

lemma $PO\text{-}m1\text{-inv0-fin-trans [iff]:$

$\{m1\text{-inv0-fin}\} \text{ trans } m1 \ \{> \ m1\text{-inv0-fin}\}$

by $(auto \ simp \ \text{add: } PO\text{-hoare-defs } m1\text{-defs intro!: } m1\text{-inv0-finI})$

lemma $PO\text{-}m1\text{-inv0-fin [iff]: reach \ m1 \subseteq m1\text{-inv0-fin}$

by $(rule \ \text{inv-rule-incr, auto del: subsetI})$

3.13.5 Refinement of $m1a$

Simulation relation

R1a1: The simulation relation and mediator function are identities.

definition

$med1a1 :: m1-obs \Rightarrow m1a-obs$ **where**
 $med1a1\ t \equiv (\ \!| \ runs = runs\ t, leak = leak\ t \)$

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 $PO-m1-step1-refines-m1a-step1$:

$\{R1a1\}$
 $(m1a-step1\ Ra\ A\ B), (m1-step1\ Ra\ A\ B)$
 $\{>\ R1a1\}$

by $(auto\ simp\ add: PO-rhoare-defs\ R1a1-defs\ m1-defs)$

lemma $PO-m1-step2-refines-m1a-step2$:

$\{R1a1\}$
 $(m1a-step2\ Rb\ A\ B), (m1-step2\ Rb\ A\ B)$
 $\{>\ R1a1\}$

by $(auto\ simp\ add: PO-rhoare-defs\ R1a1-defs\ m1-defs)$

lemma $PO-m1-step3-refines-m1a-step3$:

$\{R1a1\}$
 $(m1a-step3\ Rs\ A\ B\ Kab\ [aNum\ Ts]), (m1-step3\ Rs\ A\ B\ Kab\ Ts)$
 $\{>\ R1a1\}$

by $(auto\ simp\ add: PO-rhoare-defs\ R1a1-defs\ m1-defs)$

lemma $PO-m1-step4-refines-m1a-step4$:

$\{R1a1\}$
 $(m1a-step4\ Ra\ A\ B\ Kab\ [aNum\ Ts]), (m1-step4\ Ra\ A\ B\ Kab\ Ts)$
 $\{>\ R1a1\}$

by $(auto\ simp\ add: PO-rhoare-defs\ R1a1-defs\ m1-defs)$

lemma $PO-m1-step5-refines-m1a-step5$:

$\{R1a1\}$
 $(m1a-step5\ Rb\ A\ B\ Kab\ [aNum\ Ts]), (m1-step5\ Rb\ A\ B\ Kab\ Ts)$
 $\{>\ R1a1\}$

by $(auto\ simp\ add: PO-rhoare-defs\ R1a1-defs\ m1-defs)$

lemma $PO-m1-leak-refines-m1a-leak$:

$\{R1a1\}$
 $(m1a-leak\ Rs), (m1-leak\ Rs)$
 $\{>\ R1a1\}$

by $(auto\ simp\ add: PO-rhoare-defs\ R1a1-defs\ m1-defs)$

lemma *PO-m1-tick-refines-m1a-skip*:
 {*R1a1*}
Id, (*m1-tick T*)
 {> *R1a1*}
by (*auto simp add: PO-rhoare-defs R1a1-defs m1-defs*)

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-leak-refines-m1a-leak
PO-m1-tick-refines-m1a-skip

lemma *PO-m1-refines-init-m1a [iff]*:
init m1 \subseteq *R1a1*“(*init m1a*)
by (*auto simp add: R1a1-defs m1-defs intro!: s0g-secrecyI*)

lemma *PO-m1-refines-trans-m1a [iff]*:
 {*R1a1*}
 (*trans m1a*), (*trans m1*)
 {> *R1a1*}
apply (*auto simp add: m1-def m1-trans-def m1a-def m1a-trans-def*
intro!: PO-m1-trans-refines-m1a-trans)
apply (*force intro!: PO-m1-trans-refines-m1a-trans*)
done

Observation consistency.

lemma *obs-consistent-med1a1 [iff]*:
obs-consistent R1a1 med1a1 m1a m1
by (*auto simp add: obs-consistent-def R1a1-def m1a-def m1-def*)

Refinement result.

lemma *PO-m1-refines-m1a [iff]*:
refines R1a1 med1a1 m1a m1
by (*rule Refinement-basic*) (*auto del: subsetI*)

lemma *m1-implements-m1: implements med1a1 m1a m1*
by (*rule refinement-soundness*) (*fast*)

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:

$$\begin{aligned} \text{M1. } & A \rightarrow S : A, B \\ \text{M2a. } & S \rightarrow A : \{B, Kab, T\}_{Kas} \\ \text{M2b. } & S \rightarrow B : \{Kab, A, T\}_{Kbs} \end{aligned}$$

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

— actions:

— create initiator thread and send message 1

s1 = *s* (
runs := (*runs s*)(*Ra* \mapsto (*Init*, [*A*, *B*], [])),
chan := *insert (Insec A B (Msg [])) (chan s)* — send M1
>)

}

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 $m1e\text{-}step3$
 $m2\text{-}step3 :: [rid\text{-}t, agent, agent, key, time] \Rightarrow m2\text{-}trans$

where

$m2\text{-}step3\ Rs\ A\ B\ Kab\ Ts \equiv \{(s, s1)\}.$

— guards:

$Rs \notin dom\ (runs\ s) \wedge$ — fresh server run
 $Kab = sesK\ (Rs\$sk) \wedge$ — fresh session key
 $Ts = clk\ s \wedge$ — fresh timestamp

$Insec\ A\ B\ (Msg\ []) \in chan\ s \wedge$ — recv M1

— actions:

— record key and send messages 2 and 3

$s1 = s[$
 $runs := (runs\ s)(Rs \mapsto (Serv, [A, B], [aNum\ Ts])),$
 $chan := \{Secure\ Sv\ A\ (Msg\ [aAgt\ B, aKey\ Kab, aNum\ Ts]),$ — send $M2a/b$
 $Secure\ Sv\ B\ (Msg\ [aKey\ Kab, aAgt\ A, aNum\ Ts])\} \cup chan\ s$

$]$

}

definition — by A , refines $m1e\text{-}step4$
 $m2\text{-}step4 :: [rid\text{-}t, agent, agent, key, time] \Rightarrow m2\text{-}trans$

where

$m2\text{-}step4\ Ra\ A\ B\ Kab\ Ts \equiv \{(s, s1)\}.$

— guards:

$runs\ s\ Ra = Some\ (Init, [A, B], []) \wedge$
 $Secure\ Sv\ A\ (Msg\ [aAgt\ B, aKey\ Kab, aNum\ Ts]) \in chan\ s \wedge$ — recv $M2a$

$clk\ s < Ts + Ls \wedge$ — ensure key freshness

— actions:

— record session key

$s1 = s[$
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts]))$

$]$

}

definition — by B , refines $m1e\text{-}step5$
 $m2\text{-}step5 :: [rid\text{-}t, agent, agent, key, time] \Rightarrow m2\text{-}trans$

where

$m2\text{-}step5\ Rb\ A\ B\ Kab\ Ts \equiv \{(s, s1)\}.$

— guards:

$runs\ s\ Rb = Some\ (Resp, [A, B], []) \wedge$
 $Secure\ Sv\ B\ (Msg\ [aKey\ Kab, aAgt\ A, aNum\ Ts]) \in chan\ s \wedge$ — recv $M2b$

— ensure freshness of session key
 $clk\ s < Ts + Ls \wedge$

— actions:
— record session key
 $s1 = s\langle$
 $runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts]))$
 \rangle
 $\}$

Clock tick event

definition — refines *m1-tick*

$m2\text{-tick} :: time \Rightarrow m2\text{-trans}$

where

$m2\text{-tick} \equiv m1\text{-tick}$

Session key compromise.

definition — refines *m1-leak*

$m2\text{-leak} :: rid\text{-}t \Rightarrow m2\text{-trans}$

where

$m2\text{-leak}\ Rs \equiv \{(s, s1).\}$

— guards:

$Rs \in dom\ (runs\ s) \wedge$

$fst\ (the\ (runs\ s\ Rs)) = Serv \wedge$ — compromise server run *Rs*

— actions:

— record session key as leaked;

— intruder sends himself an insecure channel message containing the key

$s1 = s\langle leak := insert\ (sesK\ (Rs\$sk))\ (leak\ s),$

$chan := insert\ (Insec\ undefined\ undefined\ (Msg\ [aKey\ (sesK\ (Rs\$sk))]))\ (chan\ s) \rangle$

$\}$

Intruder fake event (new).

definition — refines *Id*

$m2\text{-fake} :: m2\text{-trans}$

where

$m2\text{-fake} \equiv \{(s, s1).\}$

— actions:

$s1 = s\langle$

— close under fakeable messages

$chan := fake\ ik0\ (dom\ (runs\ s))\ (chan\ s)$

\rangle

$\}$

3.14.3 Transition system

definition

$m2\text{-init} :: m2\text{-pred}$

where

$m2\text{-init} \equiv \{ \langle$

$runs = Map.empty,$

```

    leak = corrKey,
    clk = 0,
    chan = {}
  }
}

```

— Channels.ik0 contains aKey'corrKey

definition

```

m2-trans :: m2-trans where
m2-trans ≡ (⋃ A B Ra Rb Rs Kab Ts T.
  m2-step1 Ra A B ∪
  m2-step2 Rb A B ∪
  m2-step3 Rs A B Kab Ts ∪
  m2-step4 Ra A B Kab Ts ∪
  m2-step5 Rb A B Kab Ts ∪
  m2-tick T ∪
  m2-leak Rs ∪
  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-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 ≡ {s. ∀ K KK.
  aKey K ∈ extr (aKey'KK ∪ ik0) (chan s) ↔ (K ∈ KK ∨ aKey K ∈ extr ik0 (chan s))
}

```

lemmas m2-inv3a-sesK-comprI =

```

m2-inv3a-sesK-compr-def [THEN setc-def-to-intro, rule-format]

```

lemmas m2-inv3a-sesK-comprE [elim] =

```

m2-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]:
 $init\ m2 \subseteq m2\text{-inv3a-sesK-compr}$
by (auto simp add: $m2\text{-defs intro!$: $m2\text{-inv3a-sesK-comprI}$)

lemma $PO\text{-}m2\text{-inv3a-sesK-compr-trans}$ [iff]:
 $\{m2\text{-inv3a-sesK-compr}\ trans\ m2 \{>\ m2\text{-inv3a-sesK-compr}\}$
by (auto simp add: $PO\text{-hoare-defs}$ $m2\text{-defs}$ $m2\text{-inv3a-sesK-compr-simps intro!$: $m2\text{-inv3a-sesK-comprI}$)

lemma $PO\text{-}m2\text{-inv3a-sesK-compr}$ [iff]: $reach\ m2 \subseteq m2\text{-inv3a-sesK-compr}$
by (rule $inv\text{-rule-basic}$) (auto)

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]:
 $init\ m2 \subseteq m2\text{-inv3-extrKey}$
by (auto simp add: $m2\text{-defs ik0-def intro!$: $m2\text{-inv3-extrKeyI}$)

lemma $PO\text{-}m2\text{-inv3-extrKey-trans}$ [iff]:
 $\{m2\text{-inv3-extrKey} \cap m2\text{-inv3a-sesK-compr}\}$
 $trans\ m2$

```

    {> m2-inv3-extrKey}
proof (simp add: m2-def m2-trans-def, safe)
  fix Rs A B Kab Ts
  show
    {m2-inv3-extrKey  $\cap$  m2-inv3a-sesK-compr} m2-step3 Rs A B Kab Ts {> m2-inv3-extrKey}
  apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv3-extrKeyI)
  apply (auto simp add: m2-inv3a-sesK-compr-simps
    dest!: m2-inv3-extrKeyD dest: dom-lemmas)
  done
next
  fix Ra A B Kab Ts
  show
    {m2-inv3-extrKey  $\cap$  m2-inv3a-sesK-compr} m2-step4 Ra A B Kab Ts {> m2-inv3-extrKey}
  apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv3-extrKeyI)
  apply (auto simp add: dest!: m2-inv3-extrKeyD dest: dom-lemmas)
  apply (auto intro!: exI)
  done
next
  fix Rb A B Kab Ts
  show
    {m2-inv3-extrKey  $\cap$  m2-inv3a-sesK-compr} m2-step5 Rb A B Kab Ts {> m2-inv3-extrKey}
  apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv3-extrKeyI)
  apply (auto dest!: m2-inv3-extrKeyD dest: dom-lemmas)
  apply (auto intro!: exI)
  done
next
  fix Rs
  show
    {m2-inv3-extrKey  $\cap$  m2-inv3a-sesK-compr} m2-leak Rs {> m2-inv3-extrKey}
  apply (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv3-extrKeyI)
  apply (auto simp add: m2-inv3a-sesK-compr-simps)
  done
qed (auto simp add: PO-hoare-defs m2-defs intro!: m2-inv3-extrKeyI,
  auto dest!: m2-inv3-extrKeyD dest: dom-lemmas)

```

lemma *PO-m2-inv3-extrKey [iff]: reach m2 \subseteq m2-inv3-extrKey*
by (rule-tac $J=m2-inv3a-sesK-compr$ **in** inv-rule-incr) (auto)

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-inv4-M2a :: *m2-state set*

where

$m2-inv4-M2a \equiv \{s. \forall A B Kab Ts.$

$Secure\ Sv\ A\ (Msg\ [aAgt\ B,\ aKey\ Kab,\ aNum\ Ts]) \in\ chan\ s \longrightarrow A \in\ good \longrightarrow$

$(\exists\ Rs.\ Kab = sesK\ (Rs\$sk) \wedge$

$runs\ s\ Rs = Some\ (Serv,\ [A,\ B],\ [aNum\ Ts])$

$\}$

definition

$m2\text{-inv4-M2b} :: m2\text{-state set}$

where

$m2\text{-inv4-M2b} \equiv \{s. \forall A B Kab Ts.$
 $Secure\ Sv\ B\ (Msg\ [aKey\ Kab,\ aAgt\ A,\ aNum\ Ts]) \in\ chan\ s \longrightarrow B \in\ good \longrightarrow$
 $(\exists Rs. Kab = sesK\ (Rs\$sk) \wedge$
 $runs\ s\ Rs = Some\ (Serv,\ [A,\ B],\ [aNum\ Ts]))$
 $\}$

lemmas $m2\text{-inv4-M2aI} =$
 $m2\text{-inv4-M2a-def}\ [THEN\ setc\ def\ to\ intro,\ rule\ format]$

lemmas $m2\text{-inv4-M2aE}\ [elim] =$
 $m2\text{-inv4-M2a-def}\ [THEN\ setc\ def\ to\ elim,\ rule\ format]$

lemmas $m2\text{-inv4-M2aD} =$
 $m2\text{-inv4-M2a-def}\ [THEN\ setc\ def\ to\ dest,\ rule\ format,\ rotated\ 1]$

lemmas $m2\text{-inv4-M2bI} = m2\text{-inv4-M2b-def}\ [THEN\ setc\ def\ to\ intro,\ rule\ format]$

lemmas $m2\text{-inv4-M2bE}\ [elim] =$
 $m2\text{-inv4-M2b-def}\ [THEN\ setc\ def\ to\ elim,\ rule\ format]$

lemmas $m2\text{-inv4-M2bD} =$
 $m2\text{-inv4-M2b-def}\ [THEN\ setc\ def\ to\ dest,\ rule\ format,\ rotated\ 1]$

Invariance proofs.

lemma $PO\text{-}m2\text{-inv4-M2a-init}\ [iff]:$

$init\ m2 \subseteq m2\text{-inv4-M2a}$

by $(auto\ simp\ add:\ m2\text{-defs}\ intro!:\ m2\text{-inv4-M2aI})$

lemma $PO\text{-}m2\text{-inv4-M2a-trans}\ [iff]:$

$\{m2\text{-inv4-M2a}\}\ trans\ m2\ \{>\ m2\text{-inv4-M2a}\}$

apply $(auto\ simp\ add:\ PO\text{-}hoare\ defs\ m2\text{-defs}\ intro!:\ m2\text{-inv4-M2aI})$

apply $(auto\ dest!:\ m2\text{-inv4-M2aD}\ dest:\ dom\ lemmas)$

— 3 subgoals

apply $(force\ dest!:\ spec)+$

done

lemma $PO\text{-}m2\text{-inv4-M2a}\ [iff]:\ reach\ m2 \subseteq m2\text{-inv4-M2a}$

by $(rule\ inv\ rule\ basic)\ (auto)$

lemma $PO\text{-}m2\text{-inv4-M2b-init}\ [iff]:$

$init\ m2 \subseteq m2\text{-inv4-M2b}$

by $(auto\ simp\ add:\ m2\text{-defs}\ intro!:\ m2\text{-inv4-M2bI})$

lemma $PO\text{-}m2\text{-inv4-M2b-trans}\ [iff]:$

$\{m2\text{-inv4-M2b}\}\ trans\ m2\ \{>\ m2\text{-inv4-M2b}\}$

apply $(auto\ simp\ add:\ PO\text{-}hoare\ defs\ m2\text{-defs}\ intro!:\ m2\text{-inv4-M2bI})$

apply $(auto\ dest!:\ m2\text{-inv4-M2bD}\ dest:\ dom\ lemmas)$

— 3 subgoals

apply $(force\ dest!:\ spec)+$

done

lemma $PO\text{-}m2\text{-inv4-M2b}\ [iff]:\ reach\ m2 \subseteq m2\text{-inv4-M2b}$

by (rule inv-rule-incr) (auto del: subsetI)

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 [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 \rrbracket
 $\implies A = A' \wedge B = B' \wedge Ts = Ts'$

apply (subgoal-tac $A' \notin$ bad $\wedge B \notin$ bad, auto)

apply (auto dest!: m2-inv4-M2aD m2-inv4-M2bD)

done

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 (cases $A \in$ bad)

case True

from assms(1) $\langle A \in$ bad \rangle **have** aKey $K \in$ extr ik0 (chan s) **by** auto

with $\langle s \in$ m2-inv3-extrKey $\rangle \langle K \notin$ leak $s \rangle$ **show** ?thesis **by** auto

next

case False

with assms **show** ?thesis **by** (auto dest: m2-inv4-M2aD)

qed

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 (cases $B \in$ bad)

case True

from assms(1) $\langle B \in$ bad \rangle **have** aKey $K \in$ extr ik0 (chan s) **by** auto

with $\langle s \in$ m2-inv3-extrKey $\rangle \langle K \notin$ leak $s \rangle$ **show** ?thesis **by** auto

next

case False

with assms **show** ?thesis **by** (auto dest: m2-inv4-M2bD)

qed

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

$m2-inv5-ikk-sv \equiv \{s. \forall R A B al.$

runs $s R =$ Some (Serv, [A, B], al) $\longrightarrow A \in$ good $\longrightarrow B \in$ good \longrightarrow

aKey (sesK (R\$sk)) \in extr ik0 (chan s) \longrightarrow

$sesK (R\$sk) \in leak\ s$
 $\}$

lemmas $m2\text{-inv5-ikk-svI} = m2\text{-inv5-ikk-sv-def [THEN setc-def-to-intro, rule-format]$

lemmas $m2\text{-inv5-ikk-svE [elim]} = m2\text{-inv5-ikk-sv-def [THEN setc-def-to-elim, rule-format]$

lemmas $m2\text{-inv5-ikk-svD} = m2\text{-inv5-ikk-sv-def [THEN setc-def-to-dest, rule-format, rotated 1]$

Invariance proof.

lemma $PO\text{-}m2\text{-inv5-ikk-sv-init [iff]:$

$init\ m2 \subseteq m2\text{-inv5-ikk-sv}$

by $(auto\ simp\ add: m2\text{-defs}\ intro!: m2\text{-inv5-ikk-svI})$

lemma $PO\text{-}m2\text{-inv5-ikk-sv-trans [iff]:$

$\{m2\text{-inv5-ikk-sv} \cap m2\text{-inv3a-sesK-compr} \cap m2\text{-inv3-extrKey}\}$
 $trans\ m2$

$\{> m2\text{-inv5-ikk-sv}\}$

by $(simp\ add: PO\text{-hoare-defs}\ m2\text{-defs, safe}\ intro!: m2\text{-inv5-ikk-svI})$

$(auto\ simp\ add: m2\text{-inv3a-sesK-compr-simps}\ dest: dom\text{-lemmas})$

lemma $PO\text{-}m2\text{-inv5-ikk-sv [iff]: reach\ m2 \subseteq m2\text{-inv5-ikk-sv}$

by $(rule\text{-tac}\ J=m2\text{-inv3-extrKey} \cap m2\text{-inv3a-sesK-compr\ in\ inv\text{-rule-incr}) (auto)$

inv6/7: Key secrecy for initiator and responder

These invariants are derivable.

definition

$m2\text{-inv6-ikk-init} :: m2\text{-state}\ set$

where

$m2\text{-inv6-ikk-init} \equiv \{s. \forall A\ B\ Ra\ K\ Ts\ nl.$

$runs\ s\ Ra = Some\ (Init, [A, B], aKey\ K \# aNum\ Ts \# nl) \longrightarrow$

$A \in good \longrightarrow B \in good \longrightarrow aKey\ K \in extr\ ik0\ (chan\ s) \longrightarrow$

$K \in leak\ s$

$\}$

lemmas $m2\text{-inv6-ikk-initI} = m2\text{-inv6-ikk-init-def [THEN setc-def-to-intro, rule-format]$

lemmas $m2\text{-inv6-ikk-initE [elim]} = m2\text{-inv6-ikk-init-def [THEN setc-def-to-elim, rule-format]$

lemmas $m2\text{-inv6-ikk-initD} = m2\text{-inv6-ikk-init-def [THEN setc-def-to-dest, rule-format, rotated 1]$

definition

$m2\text{-inv7-ikk-resp} :: m2\text{-state}\ set$

where

$m2\text{-inv7-ikk-resp} \equiv \{s. \forall A\ B\ Rb\ K\ Ts\ nl.$

$runs\ s\ Rb = Some\ (Resp, [A, B], aKey\ K \# aNum\ Ts \# nl) \longrightarrow$

$A \in good \longrightarrow B \in good \longrightarrow aKey\ K \in extr\ ik0\ (chan\ s) \longrightarrow$

$K \in leak\ s$

$\}$

lemmas $m2\text{-inv7-ikk-respI} = m2\text{-inv7-ikk-resp-def [THEN setc-def-to-intro, rule-format]$

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]

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 $PO\text{-}m2\text{-step1}\text{-refines}\text{-}m1\text{-step1}$:

{ $R12$ }
 $(m1\text{-step1 } Ra \ A \ B), (m2\text{-step1 } Ra \ A \ B)$
{> $R12$ }

by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

lemma $PO\text{-}m2\text{-step2}\text{-refines}\text{-}m1\text{-step2}$:

{ $R12$ }
 $(m1\text{-step2 } Rb \ A \ B), (m2\text{-step2 } Rb \ A \ B)$
{> $R12$ }

by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

lemma $PO\text{-}m2\text{-step3}\text{-refines}\text{-}m1\text{-step3}$:

{ $R12$ }
 $(m1\text{-step3 } Rs \ A \ B \ Kab \ Ts), (m2\text{-step3 } Rs \ A \ B \ Kab \ Ts)$
{> $R12$ }

by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

lemma $PO\text{-}m2\text{-step4}\text{-refines}\text{-}m1\text{-step4}$:

{ $R12 \cap UNIV \times (m2\text{-inv4}\text{-}M2a \cap m2\text{-inv3}\text{-}extrKey)$ }
 $(m1\text{-step4 } Ra \ A \ B \ Kab \ Ts), (m2\text{-step4 } Ra \ A \ B \ Kab \ Ts)$
{> $R12$ }

by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, simp-all*)
(auto dest: m2-inv34-M2a-authorized)

lemma $PO\text{-}m2\text{-step5}\text{-refines}\text{-}m1\text{-step5}$:

{ $R12 \cap UNIV \times (m2\text{-inv4}\text{-}M2b \cap m2\text{-inv3}\text{-}extrKey)$ }
 $(m1\text{-step5 } Rb \ A \ B \ Kab \ Ts), (m2\text{-step5 } Rb \ A \ B \ Kab \ Ts)$
{> $R12$ }

by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, simp-all*)
(auto dest: m2-inv34-M2b-authorized)

— REMOVED!: $m2\text{-inv5}\text{-ikk}\text{-sv}$

lemma $PO\text{-}m2\text{-tick}\text{-refines}\text{-}m1\text{-tick}$:

{ $R12$ }
 $(m1\text{-tick } T), (m2\text{-tick } T)$
{> $R12$ }

by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, simp-all*)

lemma *PO-m2-leak-refines-m1-leak*:

{*R12*}
(*m1-leak Rs*), (*m2-leak Rs*)
{> *R12*}

by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

lemma *PO-m2-fake-refines-skip*:

{*R12*}
Id, *m2-fake*
{> *R12*}

by (*simp add: PO-rhoare-defs R12-def m2-defs, safe, auto*)

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*“(*init m1*)

by (*auto simp add: R12-def m1-defs m2-loc-defs*)

lemma *PO-m2-refines-trans-m1 [iff]*:

{*R12* \cap
UNIV \times (*m2-inv4-M2b* \cap *m2-inv4-M2a* \cap *m2-inv3-extrKey*)}
(*trans m1*), (*trans m2*)
{> *R12*}

by (*auto simp add: m2-def m2-trans-def m1-def m1-trans-def*)
(*blast intro!: PO-m2-trans-refines-m1-trans*)+

lemma *PO-obs-consistent-R12 [iff]*:

obs-consistent R12 med21 m1 m2

by (*auto simp add: obs-consistent-def R12-def med21-def m2-defs*)

Refinement result.

lemma *m2-refines-m1 [iff]*:

refines
(*R12* \cap
(*UNIV* \times (*m2-inv4-M2b* \cap *m2-inv4-M2a* \cap *m2-inv3-extrKey* \cap *m2-inv3a-sesK-compr*)))
med21 m1 m2

by (*rule Refinement-using-invariants*) (*auto*)

lemma *m2-implements-m1 [iff]*:

implements med21 m1 m2

by (*rule refinement-soundness*) (*auto*)

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 *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'bad*

by (*auto simp add: keySetup-def ltkkeySetup-def corrKey-def*)

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 $(\langle runs = runs\ s, leak = leak\ s, clk = clk\ s \rangle)$

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 $M2$

$s1 = s\langle$

$\text{runs} := (\text{runs } s)(Rs \mapsto (\text{Serv}, [A, B], [aNum \ Ts])),$ — 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
 $s1 = s \{$
 $\quad runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts]))$
 $\quad \}$
 $\}$

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

$s1 = s \{$
 $\quad runs := (runs\ s)(Rb \mapsto (Resp, [A, B], [aKey\ Kab, aNum\ Ts]))$
 $\quad \}$

$\}$

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 (analz (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 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

$m3\text{-inv1-lkeysec} \equiv \{s. \forall C. \\ (Key (shrK C) \in parts (IK s) \longrightarrow C \in bad) \wedge \\ (C \in bad \longrightarrow Key (shrK C) \in IK s)\}$

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*]:

$init\ m3 \subseteq m3\text{-inv1-lkeysec}$

by (*auto simp add: m3-defs intro!: m3-inv1-lkeysecI*)

lemma *PO-m3-inv1-lkeysec-trans* [*iff*]:

$\{m3\text{-inv1-lkeysec}\} trans\ m3 \{> m3\text{-inv1-lkeysec}\}$

by (*fastforce simp add: PO-hoare-defs m3-defs intro!: m3-inv1-lkeysecI*)

lemma *PO-m3-inv1-lkeysec* [*iff*]: $reach\ m3 \subseteq m3\text{-inv1-lkeysec}$

by (*rule inv-rule-incr*) (*fast+*)

Useful simplifier lemmas

lemma *m3-inv1-lkeysec-for-parts* [*simp*]:

$\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies Key (shrK C) \in parts (IK s) \longleftrightarrow C \in bad$

by *auto*

lemma *m3-inv1-lkeysec-for-analz* [*simp*]:

$\llbracket s \in m3\text{-inv1-lkeysec} \rrbracket \implies Key (shrK C) \in analz (IK s) \longleftrightarrow C \in bad$

by *auto*

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

$m3\text{-inv3-sesK-compr} \equiv \{s. \forall K KK. \\ KK \subseteq range\ sesK \longrightarrow$

$KK \subseteq range\ sesK \longrightarrow$

$(Key\ K \in analz (Key\ KK \cup (IK\ s))) = (K \in KK \vee Key\ K \in analz (IK\ s))\}$

}

lemmas $m3\text{-inv3-sesK-comprI} = m3\text{-inv3-sesK-compr-def}$ [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 $insert\text{-commute-Key} = insert\text{-commute}$ [where $x=Key\ K$ for K]

lemmas $m3\text{-inv3-sesK-compr-simps} =$

$m3\text{-inv3-sesK-comprD}$

$m3\text{-inv3-sesK-comprD}$ [where $KK=insert\ Kab\ KK$ for $Kab\ KK$, simplified]

$m3\text{-inv3-sesK-comprD}$ [where $KK=\{Kab\}$ for Kab , simplified]

$insert\text{-commute-Key}$

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 $abs\text{-msg-empty}$: $abs\text{-msg } \{\} = \{\}$
by (*auto*)

lemma $abs\text{-msg-Un}$ [*simp*]:
 $abs\text{-msg } (G \cup H) = abs\text{-msg } G \cup abs\text{-msg } H$
by (*auto*)

lemma $abs\text{-msg-mono}$ [*elim*]:
 $\llbracket m \in abs\text{-msg } G; G \subseteq H \rrbracket \Longrightarrow m \in abs\text{-msg } H$
by (*auto*)

lemma $abs\text{-msg-insert-mono}$ [*intro*]:
 $\llbracket m \in abs\text{-msg } H \rrbracket \Longrightarrow m \in abs\text{-msg } (\text{insert } m' \text{ } H)$
by (*auto*)

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*:
 $\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; t \in m3\text{-inv1-lkeysec} \rrbracket$
 $\implies \text{abs-msg } (\text{synth } (\text{analz } (IK\ t))) \subseteq \text{fake ik0 } (\text{dom } (\text{runs } s)) (\text{chan } s)$
apply (*auto*)
— 4 subgoals, deal with replays first
apply (*blast*)
defer 1 apply (*blast*)
— remaining 2 subgoals are real fakes
apply (*rule fake-StatCh, auto simp add: R23-keys-simps*)
done

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*:
 $\{R23\}$
 $(m2\text{-step1 } Ra\ A\ B), (m3\text{-step1 } Ra\ A\ B)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros*)
(*auto*)

lemma *PO-m3-step2-refines-m2-step2*:
 $\{R23\}$
 $(m2\text{-step2 } Rb\ A\ B), (m3\text{-step2 } Rb\ A\ B)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros*)
(*auto*)

lemma *PO-m3-step3-refines-m2-step3*:
 $\{R23 \cap (m2\text{-inv3a-sesK-compr}) \times (m3\text{-inv3-sesK-compr} \cap m3\text{-inv1-lkeysec})\}$
 $(m2\text{-step3 } Rs\ A\ B\ Kab\ Ts), (m3\text{-step3 } Rs\ A\ B\ Kab\ Ts)$
 $\{> R23\}$
proof –
{ fix *s t*
assume *H*:
 $(s, t) \in R23\text{-msgs } (s, t) \in R23\text{-keys } (s, t) \in R23\text{-pres}$
 $s \in m2\text{-inv3a-sesK-compr } t \in m3\text{-inv3-sesK-compr } t \in m3\text{-inv1-lkeysec}$
 $Kab = \text{sesK } (Rs\$sk) \ Rs \notin \text{dom } (\text{runs } t)$
 $\{\text{Agent } A, \text{Agent } B\} \in \text{parts } (IK\ t)$
let *?s'*=
 $s(\text{runs} := (\text{runs } s)(Rs \mapsto (\text{Serv}, [A, B], [aNum\ (clk\ t)])),$
 $\text{chan} := \text{insert } (\text{Secure } Sv\ A\ (Msg\ [aAgt\ B, aKey\ Kab, aNum\ (clk\ t)]))$
 $(\text{insert } (\text{Secure } Sv\ B\ (Msg\ [aKey\ Kab, aAgt\ A, aNum\ (clk\ t)])) (\text{chan } s)) \)$
let *?t'*=
 $t(\text{runs} := (\text{runs } t)(Rs \mapsto (\text{Serv}, [A, B], [aNum\ (clk\ t)])),$

```

    IK := insert
      (Crypt (shrK A) { Agent B, Key Kab, Number (clk t) })
    (insert
      (Crypt (shrK B) { Key Kab, Agent A, Number (clk t) })
      (IK t) )
  have (?s', ?t') ∈ R23-msgs using H
  by (-) (rule R23-intros, auto)
  moreover
  have (?s', ?t') ∈ R23-keys using H
  by (-) (rule R23-intros,
    auto simp add: m2-inv3a-sesK-compr-simps m3-inv3-sesK-compr-simps,
    auto simp add: R23-keys-simps)
  moreover
  have (?s', ?t') ∈ R23-pres using H
  by (-) (rule R23-intros, auto)
  moreover
  note calculation
}
thus ?thesis
by (auto simp add: PO-rhoare-defs R23-def m3-defs)
qed

```

lemma *PO-m3-step4-refines-m2-step4*:
 $\{R23 \cap UNIV \times (m3-inv1-lkeysec)\}$
 $(m2-step4\ Ra\ A\ B\ Kab\ Ts), (m3-step4\ Ra\ A\ B\ Kab\ Ts)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!*: R23-intros)
(auto)

lemma *PO-m3-step5-refines-m2-step5*:
 $\{R23\}$
 $(m2-step5\ Rb\ A\ B\ Kab\ Ts), (m3-step5\ Rb\ A\ B\ Kab\ Ts)$
 $\{> R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!*: R23-intros)
(auto)

lemma *PO-m3-tick-refines-m2-tick*:
 $\{R23\}$
 $(m2-tick\ T), (m3-tick\ T)$
 $\{>R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!*: R23-intros)
(auto)

Intruder events.

lemma *PO-m3-leak-refines-m2-leak*:
 $\{R23\}$
 $(m2-leak\ Rs), (m3-leak\ Rs)$
 $\{>R23\}$
by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!*: R23-intros)
(auto simp add: R23-keys-simps)

lemma *PO-m3-DY-fake-refines-m2-fake*:
 $\{R23 \cap UNIV \times (m3-inv1-lkeysec)\}$

```

    m2-fake, m3-DY-fake
  {> R23}
apply (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros
        del: abs-msg.cases)
apply (auto intro: abs-msg-DY-subset-fakeable [THEN subsetD]
        del: abs-msg.cases)
apply (auto simp add: R23-keys-simps)
done

```

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)
by (auto simp add: R23-def m3-defs intro!: R23-intros)

```

```

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}
by (auto simp add: m3-def m3-trans-def m2-def m2-trans-def
  (blast intro!: PO-m3-trans-refines-m2-trans)+)

```

```

lemma PO-m3-observation-consistent [iff]:
  obs-consistent R23 med32 m2 m3
by (auto simp add: obs-consistent-def R23-def med32-def m3-defs)

```

Refinement result.

```

lemma m3-refines-m2 [iff]:
  refines
  (R23  $\cap$  (m2-inv3a-sesK-compr)  $\times$  (m3-inv1-lkeysec))
  med32 m2 m3

```

```

proof –
  have R23  $\cap$  m2-inv3a-sesK-compr  $\times$  UNIV  $\subseteq$  UNIV  $\times$  m3-inv3-sesK-compr
  by (auto simp add: R23-def R23-keys-simps intro!: m3-inv3-sesK-comprI)
  thus ?thesis
  by (–) (rule Refinement-using-invariants, auto)
qed

```

```

lemma m3-implements-m2 [iff]:
  implements med32 m2 m3
by (rule refinement-soundness) (auto)

```

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 *ltkSetup'* \equiv *ltkSetup* **begin**

definition *ltkSetup-def*: *ltkSetup'* \equiv $\{(shrK\ C, A) \mid C\ A.\ A = C \vee A = Sv\}$

end

lemma *corrKey-shrK-bad* [*simp*]: *corrKey* = *shrK'bad*

by (*auto simp add: keySetup-def ltkSetup-def corrKey-def*)

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*

$m3\text{-step1} :: [\text{rid-}t, \text{agent}, \text{agent}] \Rightarrow m3\text{-trans}$

where

$m3\text{-step1 } Ra \ A \ B \equiv \{(s, s1)\}.$

— guards:

$Ra \notin \text{dom} (\text{runs } s) \wedge$ — Ra is fresh

— actions:

$s1 = s\{$

$\text{runs} := (\text{runs } s)(Ra \mapsto (\text{Init}, [A, B], [])),$

$IK := \text{insert } \{\text{Agent } A, \text{Agent } B\} (IK \ s)$ — send $M1$

$\}$

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 $M2$

$s1 = s\{$

$\text{runs} := (\text{runs } s)(Rs \mapsto (\text{Serv}, [A, B], [aNum \ Ts])),$

$IK := \text{insert } (\text{Crypt } (\text{shrK } A)$ — send $M2$

$\{\text{Key } Kab, \text{Agent } B, \text{Number } Ts,$

$\text{Crypt } (\text{shrK } B) \{\text{Key } Kab, \text{Agent } A, \text{Number } Ts\}\}$

$(IK \ s)$

$\}$

definition — by A , refines $m2\text{-step4}$

$m3\text{-step4} :: [\text{rid-}t, \text{agent}, \text{agent}, \text{key}, \text{time}, \text{msg}] \Rightarrow m3\text{-trans}$

where

$m3\text{-step4 } Ra \ A \ B \ Kab \ Ts \ X \equiv \{(s, s1)\}.$

— guards:

$\text{runs } s \ Ra = \text{Some } (\text{Init}, [A, B], []) \wedge$ — key not yet recv'd

$\text{Crypt } (\text{shrK } A)$ — recv $M2$

$\{\text{Key } Kab, \text{Agent } B, \text{Number } Ts, X\} \in IK \ s \wedge$

— check freshness of session key

$\text{clk } s < Ts + Ls \wedge$

— actions:
— record session key and send $M3$
 $s1 = s\{$
 $runs := (runs\ s)(Ra \mapsto (Init, [A, B], [aKey\ Kab, aNum\ Ts])),$
 $IK := insert\ X\ (IK\ s)$ — send $M3$
 $\}$
 $\}$

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 $m2\text{-}fake$

$m3-DY-fake :: m3-trans$

where

$m3-DY-fake \equiv \{(s, s1).\}$

— actions:

$s1 = s(| IK := synth (anz (IK s)) |) \quad \text{— take DY closure}$
 $\}$

3.16.4 Transition system

definition

$m3-init :: m3-pred$

where

$m3-init \equiv \{ (|$
 $runs = Map.empty,$
 $leak = shrK'bad,$
 $clk = 0,$
 $IK = Key'shrK'bad$
 $|) \}$

definition

$m3-trans :: m3-trans$ **where**

$m3-trans \equiv (\bigcup A B Ra Rb Rs Kab Ts T X.$

$m3-step1 Ra A B \cup$

$m3-step2 Rb A B \cup$

$m3-step3 Rs A B Kab Ts \cup$

$m3-step4 Ra A B Kab Ts X \cup$

$m3-step5 Rb A B Kab Ts \cup$

$m3-tick T \cup$

$m3-leak Rs \cup$

$m3-DY-fake \cup$

Id

)

definition

$m3 :: (m3-state, m3-obs) spec$ **where**

$m3 \equiv (|$

$init = m3-init,$

$trans = m3-trans,$

$obs = m3-obs$

|)

lemmas $m3-loc-defs =$

$m3-def m3-init-def m3-trans-def m3-obs-def$

$m3-step1-def m3-step2-def m3-step3-def m3-step4-def m3-step5-def$

$m3-tick-def m3-leak-def m3-DY-fake-def$

lemmas $m3-defs = m3-loc-defs m2-defs$

3.16.5 Invariants

Specialized injection that we can apply more aggressively.

lemmas *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

$m3-inv1-lkeysec \equiv \{s. \forall C.$
 $(Key (shrK C) \in parts (IK s) \longrightarrow C \in bad) \wedge$
 $(C \in bad \longrightarrow Key (shrK C) \in IK s)$
 $\}$

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]:

$init\ m3 \subseteq m3-inv1-lkeysec$

by (*auto simp add: m3-defs intro!: m3-inv1-lkeysecI*)

lemma *PO-m3-inv1-lkeysec-trans* [iff]:

$\{m3-inv1-lkeysec\}\ trans\ m3 \{>\ m3-inv1-lkeysec\}$

by (*auto simp add: PO-hoare-defs m3-defs intro!: m3-inv1-lkeysecI*)

(*auto dest!: Body*)

lemma *PO-m3-inv1-lkeysec* [iff]: $reach\ m3 \subseteq m3-inv1-lkeysec$

by (*rule inv-rule-incr*) (*fast+*)

Useful simplifier lemmas

lemma *m3-inv1-lkeysec-for-parts* [simp]:

$\llbracket s \in m3-inv1-lkeysec \rrbracket \implies Key (shrK C) \in parts (IK s) \longleftrightarrow C \in bad$

by *auto*

lemma *m3-inv1-lkeysec-for-analz* [simp]:

$\llbracket s \in m3-inv1-lkeysec \rrbracket \implies Key (shrK C) \in analz (IK s) \longleftrightarrow C \in bad$

by *auto*

inv2: Ticket shape for honestly encrypted M2

definition

m3-inv2-ticket :: *m3-pred*

where

$m3-inv2-ticket \equiv \{s. \forall A\ B\ T\ K\ X.$

$A \notin bad \longrightarrow$

$Crypt (shrK A) \llbracket Key\ K, Agent\ B, Number\ T, X \rrbracket \in parts (IK s) \longrightarrow$

$X = Crypt (shrK B) \llbracket Key\ K, Agent\ A, Number\ T \rrbracket \wedge K \in range\ sesK$

$\}$

lemmas $m3\text{-inv2}\text{-ticket}I = m3\text{-inv2}\text{-ticket}\text{-def}$ [*THEN setc-def-to-intro, rule-format*]
lemmas $m3\text{-inv2}\text{-ticket}E$ [*elim*] = $m3\text{-inv2}\text{-ticket}\text{-def}$ [*THEN setc-def-to-elim, rule-format*]
lemmas $m3\text{-inv2}\text{-ticket}D = m3\text{-inv2}\text{-ticket}\text{-def}$ [*THEN setc-def-to-dest, rule-format, rotated -1*]

Invariance proof.

lemma $PO\text{-}m3\text{-inv2}\text{-ticket}\text{-init}$ [*iff*]:
 $init\ m3 \subseteq m3\text{-inv2}\text{-ticket}$
by (*auto simp add: m3-defs intro!: m3-inv2-ticketI*)

lemma $PO\text{-}m3\text{-inv2}\text{-ticket}\text{-trans}$ [*iff*]:
 $\{m3\text{-inv2}\text{-ticket} \cap m3\text{-inv1}\text{-lkeysec}\ trans\ m3 \{>\ m3\text{-inv2}\text{-ticket}\}$
apply (*auto simp add: PO-hoare-defs m3-defs intro!: m3-inv2-ticketI*)
apply (*auto dest: m3-inv2-ticketD*)
— 2 subgoals, from step4
apply (*drule parts-cut, drule Body, auto dest: m3-inv2-ticketD*)
done

lemma $PO\text{-}m3\text{-inv2}\text{-ticket}$ [*iff*]: $reach\ m3 \subseteq m3\text{-inv2}\text{-ticket}$
by (*rule inv-rule-incr*) (*auto del: subsetI*)

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 $PO\text{-}m3\text{-inv3}\text{-sesK}\text{-compr}\text{-step4}$:
 $\{m3\text{-inv3}\text{-sesK}\text{-compr} \cap m3\text{-inv2}\text{-ticket} \cap m3\text{-inv1}\text{-lkeysec}\}$
 $m3\text{-step4}\ Ra\ A\ B\ Kab\ Ts\ X$
 $\{>\ m3\text{-inv3}\text{-sesK}\text{-compr}\}$

```

proof –
  { fix  $K$   $KK$   $s$ 
    assume  $H$ :
       $s \in m3\text{-inv1-lkeysec}$   $s \in m3\text{-inv3-sesK-compr}$   $s \in m3\text{-inv2-ticket}$ 
       $runs\ s\ Ra = Some\ (Init,\ [A,\ B],\ [])$ 
       $KK \subseteq range\ sesK$ 
       $Crypt\ (shrK\ A)\ \{\!|Key\ Kab,\ Agent\ B,\ Number\ Ts,\ X\!\} \in analz\ (IK\ s)$ 
    have
       $(Key\ K \in analz\ (insert\ X\ (Key\ 'KK \cup IK\ s))) =$ 
         $(K \in KK \vee Key\ K \in analz\ (insert\ X\ (IK\ s)))$ 
    proof (cases  $A \in bad$ )
      case True show ?thesis
        proof –
          note  $H$ 
          moreover
            with  $\langle A \in bad \rangle$  have  $X \in analz\ (IK\ s)$ 
            by (auto dest!: Decrypt)
          moreover
            hence  $X \in analz\ (Key\ 'KK \cup IK\ s)$ 
            by (auto intro: analz-mono [THEN [?] rev-subsetD])
          ultimately
            show ?thesis
            by (auto simp add: m3-inv3-sesK-compr-simps analz-insert-eq)
          qed
        next
          case False thus ?thesis using  $H$ 
          by (fastforce simp add: m3-inv3-sesK-compr-simps dest!: m3-inv2-ticketD [OF analz-into-parts])
          qed
        }
      thus ?thesis
      by (auto simp add: PO-hoare-defs m3-defs intro!: m3-inv3-sesK-comprI dest!: analz-Inj-IK)
    qed

```

All together now.

lemmas *PO-m3-inv3-sesK-compr-trans-lemmas* =
PO-m3-inv3-sesK-compr-step4

lemma *PO-m3-inv3-sesK-compr-init* [*iff*]:
init $m3 \subseteq m3\text{-inv3-sesK-compr}$
by (*auto simp add*: *m3-defs intro!*: *m3-inv3-sesK-comprI*)

lemma *PO-m3-inv3-sesK-compr-trans* [*iff*]:
 $\{m3\text{-inv3-sesK-compr} \cap m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec}\}$
trans $m3$
 $\{> m3\text{-inv3-sesK-compr}\}$
by (*auto simp add*: *m3-def m3-trans-def intro!*: *PO-m3-inv3-sesK-compr-trans-lemmas*)
(*auto simp add*: *PO-hoare-defs m3-defs m3-inv3-sesK-compr-simps intro!*: *m3-inv3-sesK-comprI*)

lemma *PO-m3-inv3-sesK-compr* [*iff*]: *reach* $m3 \subseteq m3\text{-inv3-sesK-compr}$
by (*rule-tac* $J=m3\text{-inv2-ticket} \cap m3\text{-inv1-lkeysec}$ **in** *inv-rule-incr*) (*auto*)

3.16.6 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:

$\{\{Agent\ A, Agent\ B\}\} \in H$

$\Longrightarrow Insec\ A\ B\ (Msg\ []) \in abs\text{-}msg\ H$

| *am-M2a:*

$Crypt\ (shrK\ C)\ \{\{Key\ K, Agent\ B, Number\ T, X\}\} \in H$

$\Longrightarrow Secure\ Sv\ C\ (Msg\ [aAgt\ B, aKey\ K, aNum\ T]) \in abs\text{-}msg\ H$

| *am-M2b:*

$Crypt\ (shrK\ C)\ \{\{Key\ K, Agent\ A, Number\ T\}\} \in H$

$\Longrightarrow Secure\ Sv\ C\ (Msg\ [aKey\ K, aAgt\ A, aNum\ T]) \in abs\text{-}msg\ H$

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

$R23\text{-}keys :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$

$R23\text{-}keys \equiv \{(s, t). \forall KK\ K. KK \subseteq range\ sesK \longrightarrow$

$Key\ K \in analz\ (Key\ KK \cup (IK\ t)) \longrightarrow aKey\ K \in extr\ (aKey\ KK \cup ik0)\ (chan\ s)$

$\}$

definition

$R23\text{-}pres :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$

$R23\text{-}pres \equiv \{(s, t). runs\ s = runs\ t \wedge clk\ s = clk\ t \wedge leak\ s = leak\ t\}$

definition

$R23 :: (m2\text{-}state \times m3\text{-}state)\ set\ \mathbf{where}$

$R23 \equiv R23\text{-}msgs \cap R23\text{-}keys \cap R23\text{-}pres$

lemmas $R23\text{-}defs =$

$R23\text{-}def\ R23\text{-}msgs\text{-}def\ R23\text{-}keys\text{-}def\ R23\text{-}pres\text{-}def$

The mediator function is the identity here.

definition

$med32 :: m3\text{-}obs \Rightarrow m2\text{-}obs\ \mathbf{where}$

$med32 \equiv id$

lemmas $R23\text{-}msgsI = R23\text{-}msgs\text{-}def\ [THEN\ rel\text{-}def\text{-}to\text{-}intro, simplified, rule\text{-}format]$

lemmas $R23\text{-}msgsE [elim] = R23\text{-}msgs\text{-}def\ [THEN\ rel\text{-}def\text{-}to\text{-}elim, simplified, rule\text{-}format]$

lemmas $R23\text{-}keysI = R23\text{-}keys\text{-}def\ [THEN\ rel\text{-}def\text{-}to\text{-}intro, simplified, rule\text{-}format]$

lemmas *R23-keysE* [elim] = *R23-keys-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-presI*

Lemmas for various instantiations (for keys).

lemmas *R23-keys-dest* = *R23-keys-def* [THEN *rel-def-to-dest*, *simplified*, *rule-format*, *rotated 2*]

lemmas *R23-keys-dests* =

R23-keys-dest

R23-keys-dest [where $KK = \{\}$, *simplified*]

R23-keys-dest [where $KK = \{K'\}$ for K' , *simplified*]

R23-keys-dest [where $KK = \text{insert } K' \text{ } KK$ for $K' \text{ } KK$, *simplified*, *OF - - conjI*]

General lemmas

General facts about *abs-msg*

declare *abs-msg.intros* [intro!]

declare *abs-msg.cases* [elim!]

lemma *abs-msg-empty*: *abs-msg* $\{\}$ = $\{\}$

by (*auto*)

lemma *abs-msg-Un* [*simp*]:

abs-msg ($G \cup H$) = *abs-msg* $G \cup$ *abs-msg* H

by (*auto*)

lemma *abs-msg-mono* [elim]:

$\llbracket m \in \text{abs-msg } G; G \subseteq H \rrbracket \implies m \in \text{abs-msg } H$

by (*auto*)

lemma *abs-msg-insert-mono* [intro]:

$\llbracket m \in \text{abs-msg } H \rrbracket \implies m \in \text{abs-msg } (\text{insert } m' \text{ } H)$

by (*auto*)

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*:

$\llbracket (s, t) \in R23\text{-msgs}; (s, t) \in R23\text{-keys}; (s, t) \in R23\text{-non}; t \in m3\text{-inv1-lkeysec} \rrbracket$

$\implies \text{abs-msg } (\text{synth } (\text{analz } (IK \ t))) \subseteq \text{fake ik0 } (\text{dom } (\text{runs } s)) \text{ } (\text{chan } s)$

apply (*auto*)

— 4 subgoals, deal with replays first

apply (*blast*)

defer 1 apply (*blast*)

— remaining 2 subgoals are real fakes

apply (*rule fake-StatCh*, *auto dest: R23-keys-dests*)+

done

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*}

by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros*)
 (*auto*)

lemma *PO-m3-step2-refines-m2-step2*:

{*R23*}
 (*m2-step2 Rb A B*), (*m3-step2 Rb A B*)
 {> *R23*}

by (*auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros*)
 (*auto*)

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 –

{ **fix** *s t*

assume *H*:

(*s, t*) \in *R23-msgs* (*s, t*) \in *R23-keys* (*s, t*) \in *R23-pres*
s \in *m2-inv3a-sesK-compr* *t* \in *m3-inv3-sesK-compr* *t* \in *m3-inv1-lkeysec*
Kab = *sesK (Rs\$sk)* *Rs* \notin *dom (runs t)*
 $\{\{Agent A, Agent B\}\} \in parts (IK t)$

let *?s'* =

s(*runs* := (*runs s*)(*Rs* \mapsto (*Serv*, [*A*, *B*], [*aNum (clk t)*])),
chan := *insert (Secure Sv A (Msg [aAgt B, aKey Kab, aNum (clk t)]))*
 (*insert (Secure Sv B (Msg [aKey Kab, aAgt A, aNum (clk t)])) (chan s)*))

let *?t'* =

t(*runs* := (*runs t*)(*Rs* \mapsto (*Serv*, [*A*, *B*], [*aNum (clk t)*])),
IK := *insert*
 (*Crypt (shrK A)*
 $\{\{Key Kab, Agent B, Number (clk t),$
*Crypt (shrK B) \{Key Kab, Agent A, Number (clk t)\}\}
 (*IK t*))*

have (*?s', ?t'*) \in *R23-msgs* **using** *H*

by (–) (*rule R23-intros, auto*)

moreover

have (*?s', ?t'*) \in *R23-keys* **using** *H*

by (–) (*rule R23-intros,*

auto simp add: m2-inv3a-sesK-compr-simps m3-inv3-sesK-compr-simps dest: R23-keys-dests)

moreover

have (*?s', ?t'*) \in *R23-pres* **using** *H*

by (–) (*rule R23-intros, auto*)

moreover

note *calculation*

}

thus *?thesis* **by** (*auto simp add: PO-rhoare-defs R23-def m3-defs*)

qed

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 –

{ **fix** s t

assume H:

 (s, t) \in *R23-msgs* (s, t) \in *R23-keys* (s, t) \in *R23-pres*
 t \in *m3-inv3-sesK-compr* t \in *m3-inv2-ticket* t \in *m3-inv1-lkeysec*
 runs t Ra = Some (Init, [A, B], [])

 Crypt (shrK A) {Key Kab, Agent B, Number Ts, X} \in *analz* (IK t)

let ?s' = s | runs := (runs s)(Ra \mapsto (Init, [A, B], [aKey Kab, aNum Ts])) |

and ?t' = t | runs := (runs t)(Ra \mapsto (Init, [A, B], [aKey Kab, aNum Ts])),
 IK := insert X (IK t) |

from H **have**

 Secure Sv A (Msg [aAgt B, aKey Kab, aNum Ts]) \in *chan* s

by (auto)

moreover

have X \in *parts* (IK t) **using** H

by (auto dest!: *Body MPair-parts*)

hence (?s', ?t') \in *R23-msgs* **using** H

by (auto intro!: *R23-intros*) (auto)

moreover

have (?s', ?t') \in *R23-keys*

proof (*cases*)

assume A \in *bad* **show** ?thesis

proof –

note H

moreover

hence X \in *analz* (IK t) **using** \langle A \in *bad* \rangle

by (–) (*drule Decrypt, auto*)

ultimately

show ?thesis

by (–) (*rule R23-intros, auto dest!: analz-cut intro: analz-monotonic*)

qed

next

assume A \notin *bad* **show** ?thesis

proof –

note H

moreover

with \langle A \notin *bad* \rangle **have**

 X = Crypt (shrK B) {Key Kab, Agent A, Number Ts} \wedge Kab \in *range sesK*

by (auto dest!: *m3-inv2-ticketD*)

moreover

 { **assume** H1: Key (shrK B) \in *analz* (IK t)

have aKey Kab \in *extr ik0* (*chan* s)

proof –

note *calculation*

moreover

hence Secure Sv B (Msg [aKey Kab, aAgt A, aNum Ts]) \in *chan* s

```

      by (−) (drule analz-into-parts, drule Body, elim MPair-parts, auto)
    ultimately
      show ?thesis using H1 by auto
    qed
  }
  ultimately show ?thesis
    by (−) (rule R23-intros, auto simp add: m3-inv3-sesK-compr-simps)
  qed
  qed
  moreover
    have (?s', ?t') ∈ R23-pres using H
      by (auto intro!: R23-intros)
  moreover
    note calculation
  }
  thus ?thesis
    by (auto simp add: PO-rhoare-defs R23-def m3-defs dest!: analz-Inj-IK)
  qed

```

lemma *PO-m3-step5-refines-m2-step5:*
 {R23}
 (m2-step5 Rb A B Kab Ts), (m3-step5 Rb A B Kab Ts)
 {> R23}
 by (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros)
 (auto)

lemma *PO-m3-tick-refines-m2-tick:*
 {R23}
 (m2-tick T), (m3-tick T)
 {>R23}
 by (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros)
 (auto)

Intruder events.

lemma *PO-m3-leak-refines-m2-leak:*
 {R23}
 (m2-leak Rs), (m3-leak Rs)
 {>R23}
 by (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros)
 (auto dest: R23-keys-dests)

lemma *PO-m3-DY-fake-refines-m2-fake:*
 {R23 ∩ UNIV × (m3-inv1-lkeysec)}
 m2-fake, m3-DY-fake
 {> R23}
apply (auto simp add: PO-rhoare-defs R23-def m3-defs intro!: R23-intros del: abs-msg.cases)
apply (auto intro: abs-msg-DY-subset-fakeable [THEN subsetD] del: abs-msg.cases
 dest: R23-keys-dests)

done

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)

by (*auto simp add: R23-def m3-defs ik0-def intro!: R23-intros*)

lemma *PO-m3-refines-trans-m2* [iff]:

{*R23* \cap

(*m2-inv3a-sesK-compr*) \times (*m3-inv3-sesK-compr* \cap *m3-inv2-ticket* \cap *m3-inv1-lkeysec*)}

(*trans m2*), (*trans m3*)

{ $>$ *R23*}

by (*auto simp add: m3-def m3-trans-def m2-def m2-trans-def*)

(*blast intro!: PO-m3-trans-refines-m2-trans*)+

lemma *PO-m3-observation-consistent* [iff]:

obs-consistent R23 med32 m2 m3

by (*auto simp add: obs-consistent-def R23-def med32-def m3-defs*)

Refinement result.

lemma *m3-refines-m2* [iff]:

refines

(*R23* \cap (*m2-inv3a-sesK-compr*) \times (*m3-inv3-sesK-compr* \cap *m3-inv2-ticket* \cap *m3-inv1-lkeysec*))

med32 m2 m3

by (*rule Refinement-using-invariants*) (*auto*)

lemma *m3-implements-m2* [iff]:

implements med32 m2 m3

by (*rule refinement-soundness*) (*auto*)

end