

# Security Protocols

Giampaolo Bella, Frederic Blanqui, Lawrence C. Paulson et al.

December 17, 2025

## Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Theory of Agents and Messages for Security Protocols</b>               | <b>2</b>  |
| 1.1      | Inductive Definition of All Parts of a Message . . . . .                  | 3         |
| 1.2      | Inverse of keys . . . . .   | 3         |
| 1.3      | The <i>keysFor</i> operator . . . . .                                     | 3         |
| 1.4      | Inductive relation "parts" . . . . .                                      | 4         |
| 1.4.1    | Unions . . . . .  | 5         |
| 1.4.2    | Idempotence and transitivity . . . . .                                    | 5         |
| 1.4.3    | Rewrite rules for pulling out atomic messages . . . . .                   | 6         |
| 1.5      | Inductive relation "analz" . . . . .                                      | 6         |
| 1.5.1    | General equational properties . . . . .                                   | 7         |
| 1.5.2    | Rewrite rules for pulling out atomic messages . . . . .                   | 8         |
| 1.5.3    | Idempotence and transitivity . . . . .                                    | 9         |
| 1.6      | Inductive relation "synth" . . . . .                                      | 10        |
| 1.6.1    | Unions . . . . .  | 10        |
| 1.6.2    | Idempotence and transitivity . . . . .                                    | 10        |
| 1.6.3    | Combinations of parts, analz and synth . . . . .                          | 11        |
| 1.6.4    | For reasoning about the Fake rule in traces . . . . .                     | 11        |
| 1.7      | HPair: a combination of Hash and MPair . . . . .                          | 12        |
| 1.7.1    | Freeness . . . . .  | 12        |
| 1.7.2    | Specialized laws, proved in terms of those for Hash and MPair . . . . .   | 13        |
| 1.8      | The set of key-free messages . . . . .                                    | 14        |
| 1.9      | Tactics useful for many protocol proofs . . . . .                         | 14        |
| <b>2</b> | <b>Theory of Events for Security Protocols</b>                            | <b>15</b> |
| 2.1      | Function <i>knows</i> . . . . .   | 16        |
| 2.2      | Knowledge of Agents . . . . .   | 17        |
| 2.3      | Asymmetric Keys . . . . .   | 19        |
| 2.4      | Basic properties of <i>pubEK</i> and <i>priEK</i> . . . . .               | 20        |
| 2.5      | "Image" equations that hold for injective functions . . . . .             | 21        |
| 2.6      | Symmetric Keys . . . . .  | 21        |
| 2.7      | Initial States of Agents . . . . .  | 22        |
| 2.8      | Function <i>knows Spy</i> . . . . .                                       | 24        |
| 2.9      | Fresh Nonces . . . . .  | 25        |
| 2.10     | Supply fresh nonces for possibility theorems . . . . .                    | 25        |
| 2.11     | Specialized Rewriting for Theorems About <i>analz</i> and Image . . . . . | 25        |
| 2.12     | Specialized Methods for Possibility Theorems . . . . .                    | 26        |

|          |   |           |
|----------|---|-----------|
| <b>3</b> | <b>Needham-Schroeder Shared-Key Protocol</b>  | <b>26</b> |
| 3.1      | Inductive proofs about <i>ns_shared</i> . . . . .   | 27        |
| 3.1.1    | Forwarding lemmas, to aid simplification . . . . .  | 27        |
| 3.1.2    | Lemmas concerning the form of items passed in messages . . . . .  | 28        |
| 3.1.3    | Session keys are not used to encrypt other session keys . . . . .   | 29        |
| 3.1.4    | The session key K uniquely identifies the message . . . . .   | 29        |
| 3.1.5    | Crucial secrecy property: Spy doesn't see the keys sent in NS2 . . . . .  | 29        |
| 3.2      | Guarantees available at various stages of protocol . . . . .  | 29        |
| 3.3      | Lemmas for reasoning about predicate "Issues" . . . . .   | 31        |
| 3.4      | Guarantees of non-injective agreement on the session key, and of key distribution. They also express forms of freshness of certain messages, namely that agents were alive after something happened. . . . .  | 31        |
| <b>4</b> | <b>The Kerberos Protocol, BAN Version</b>   | <b>32</b> |
| 4.1      | Lemmas for reasoning about predicate "Issues" . . . . .   | 34        |
| 4.2      | Lemmas concerning the form of items passed in messages . . . . .  | 36        |
| 4.3      | Non-temporal guarantees, explicitly relying on non-occurrence of oops events - refined below by temporal guarantees . . . . .   | 37        |
| 4.4      | Temporal guarantees, relying on a temporal check that insures that no oops event occurred. These are available in the sense of goal availability . . . . .  | 40        |
| 4.5      | Treatment of the key distribution goal using trace inspection. All guarantees are in non-temporal form, hence non available, though their temporal form is trivial to derive. These guarantees also convey a stronger form of authentication - non-injective agreement on the session key . . . . . | 41        |
| <b>5</b> | <b>The Kerberos Protocol, BAN Version, with Gets event</b>  | <b>42</b> |
| 5.1      | Lemmas concerning the form of items passed in messages . . . . .  | 46        |
| 5.2      | Non-temporal guarantees, explicitly relying on non-occurrence of oops events - refined below by temporal guarantees . . . . .   | 48        |
| 5.3      | Temporal guarantees, relying on a temporal check that insures that no oops event occurred. These are available in the sense of goal availability . . . . .  | 50        |
| 5.4      | Combined guarantees of key distribution and non-injective agreement on the session keys . . . . .   | 51        |
| <b>6</b> | <b>The Kerberos Protocol, Version IV</b>  | <b>52</b> |
| 6.1      | Lemmas about lists, for reasoning about Issues . . . . .  | 56        |
| 6.2      | Lemmas about <i>authKeys</i> . . . . .  | 56        |
| 6.3      | Forwarding Lemmas . . . . .   | 57        |
| 6.4      | Lemmas for reasoning about predicate "before" . . . . .   | 58        |
| 6.5      | Regularity Lemmas . . . . .   | 59        |
| 6.6      | Authenticity theorems: confirm origin of sensitive messages . . . . .   | 61        |
| 6.7      | Reliability: friendly agents send something if something else happened . . . . .  | 63        |
| 6.8      | Unicity Theorems . . . . .  | 64        |
| 6.9      | Lemmas About the Predicate <i>AKcryptSK</i> . . . . .   | 65        |
| 6.10     | Secrecy Theorems . . . . .  | 67        |

|           |   |            |
|-----------|---|------------|
| 6.11      | Parties authentication: each party verifies "the identity of another party who generated some data" (quoted from Neuman and Ts'o).  | 70         |
| 6.12      | Key distribution guarantees An agent knows a session key if he used it to issue a cipher. These guarantees also convey a stronger form of authentication - non-injective agreement on the session key | 72         |
| <b>7</b>  | <b>The Kerberos Protocol, Version IV</b>  | <b>75</b>  |
| 7.1       | Lemmas about reception event  | 79         |
| 7.2       | Lemmas about <i>authKeys</i>  | 80         |
| 7.3       | Forwarding Lemmas   | 80         |
| 7.4       | Regularity Lemmas   | 81         |
| 7.5       | Authenticity theorems: confirm origin of sensitive messages   | 83         |
| 7.6       | Reliability: friendly agents send something if something else happened  | 85         |
| 7.7       | Unicity Theorems  | 86         |
| 7.8       | Lemmas About the Predicate <i>AKcryptSK</i>   | 87         |
| 7.9       | Secrecy Theorems  | 89         |
| 7.10      | 2. Parties' strong authentication: non-injective agreement on the session key. The same guarantees also express key distribution, hence their names   | 92         |
| <b>8</b>  | <b>The Kerberos Protocol, Version V</b>   | <b>95</b>  |
| 8.1       | Lemmas about lists, for reasoning about Issues  | 98         |
| 8.2       | Lemmas about <i>authKeys</i>  | 99         |
| 8.3       | Forwarding Lemmas   | 99         |
| 8.4       | Regularity Lemmas   | 100        |
| 8.5       | Authenticity theorems: confirm origin of sensitive messages   | 101        |
| 8.6       | Reliability: friendly agents send something if something else happened  | 104        |
| 8.7       | Unicity Theorems  | 105        |
| 8.8       | Lemmas About the Predicate <i>AKcryptSK</i>   | 105        |
| 8.9       | Secrecy Theorems  | 107        |
| 8.10      | Authentication  | 110        |
| 8.11      | Parties' knowledge of session keys  | 112        |
| 8.12      | Novel guarantees, never studied before  | 113        |
| <b>9</b>  | <b>The Original Otway-Rees Protocol</b>   | <b>115</b> |
| 9.1       | Towards Secrecy: Proofs Involving <i>analz</i>  | 117        |
| 9.2       | Authenticity properties relating to NA  | 118        |
| 9.3       | Authenticity properties relating to NB  | 120        |
| <b>10</b> | <b>The Otway-Rees Protocol as Modified by Abadi and Needham</b>   | <b>121</b> |
| 10.1      | Proofs involving <i>analz</i>   | 123        |
| 10.2      | Authenticity properties relating to NA  | 124        |
| 10.3      | Authenticity properties relating to NB  | 125        |
| <b>11</b> | <b>The Otway-Rees Protocol: The Faulty BAN Version</b>  | <b>126</b> |
| 11.1      | For reasoning about the encrypted portion of messages   | 128        |
| 11.2      | Proofs involving <i>analz</i>   | 128        |
| 11.3      | Attempting to prove stronger properties   | 129        |

|   |            |
|---|------------|
| <b>12 Bella's version of the Otway-Rees protocol</b>  | <b>130</b> |
| 12.1 Proofs involving <i>analz</i> . . . . .  | 132        |
| <b>13 The Woo-Lam Protocol</b>  | <b>135</b> |
| <b>14 The Otway-Bull Recursive Authentication Protocol</b>  | <b>138</b> |
| <b>15 The Yahalom Protocol</b>  | <b>143</b> |
| 15.1 Regularity Lemmas for Yahalom . . . . .  | 145        |
| 15.2 Secrecy Theorems . . . . .   | 146        |
| 15.2.1 Security Guarantee for A upon receiving YM3 . . . . .  | 147        |
| 15.2.2 Security Guarantees for B upon receiving YM4 . . . . .   | 147        |
| 15.2.3 Towards proving secrecy of Nonce NB . . . . .  | 148        |
| 15.2.4 The Nonce NB uniquely identifies B's message. . . . .  | 149        |
| 15.2.5 A nonce value is never used both as NA and as NB . . . . .                                       | 149        |
| 15.3 Authenticating B to A . . . . .  | 150        |
| 15.4 Authenticating A to B using the certificate <i>Crypt K (Nonce NB)</i> . . . . .                    | 151        |
| <b>16 The Yahalom Protocol, Variant 2</b>   | <b>152</b> |
| 16.1 Inductive Proofs . . . . .   | 153        |
| 16.2 Crucial Secrecy Property: Spy Does Not See Key <i>KAB</i> . . . . .                                | 154        |
| 16.3 Security Guarantee for A upon receiving YM3 . . . . .  | 155        |
| 16.4 Security Guarantee for B upon receiving YM4 . . . . .  | 156        |
| 16.5 Authenticating B to A . . . . .  | 156        |
| 16.6 Authenticating A to B . . . . .  | 157        |
| <b>17 The Yahalom Protocol: A Flawed Version</b>  | <b>158</b> |
| 17.1 Regularity Lemmas for Yahalom . . . . .  | 159        |
| 17.2 For reasoning about the encrypted portion of messages . . . . .                                    | 159        |
| 17.3 Secrecy Theorems . . . . .   | 160        |
| 17.4 Session keys are not used to encrypt other session keys . . . . .                                  | 160        |
| 17.5 Security Guarantee for A upon receiving YM3 . . . . .  | 161        |
| 17.6 Security Guarantees for B upon receiving YM4 . . . . .   | 161        |
| 17.7 The Flaw in the Model . . . . .  | 161        |
| 17.8 Basic Lemmas . . . . .   | 164        |
| 17.9 About NRO: Validity for <i>B</i> . . . . .   | 165        |
| 17.10 About NRR: Validity for <i>A</i> . . . . .  | 165        |
| 17.11 Proofs About <i>sub_K</i> . . . . .   | 166        |
| 17.12 Proofs About <i>con_K</i> . . . . .   | 166        |
| 17.13 Proving fairness . . . . .  | 167        |
| <b>18 Conventional protocols: rely on conventional Message, Event and Public – Shared-key protocols</b> | <b>168</b> |
| <b>19 The Needham-Schroeder Public-Key Protocol (Flawed)</b>  | <b>169</b> |
| 19.1 Inductive proofs about <i>ns_public</i> . . . . .  | 169        |
| 19.2 Authenticity properties obtained from term NS1 . . . . .   | 170        |
| 19.3 Authenticity properties obtained from term NS2 . . . . .   | 171        |

|   |            |
|---|------------|
| <b>20 The Needham-Schroeder Public-Key Protocol</b>   | <b>172</b> |
| 20.1 Inductive proofs about <i>ns_public</i> . . . . .  | 172        |
| 20.2 Authenticity properties obtained from term NS1 . . . . .   | 173        |
| 20.3 Authenticity properties obtained from term NS2 . . . . .   | 174        |
| 20.4 Overall guarantee for term B . . . . .   | 175        |
| <b>21 The TLS Protocol: Transport Layer Security</b>  | <b>175</b> |
| 21.1 Protocol Proofs . . . . .  | 179        |
| 21.2 Inductive proofs about <i>tls</i> . . . . .  | 180        |
| 21.2.1 Properties of items found in Notes . . . . .   | 181        |
| 21.2.2 Protocol goal: if B receives CertVerify, then A sent it . . . . .  | 181        |
| 21.2.3 Unicity results for PMS, the pre-master-secret . . . . .   | 182        |
| 21.3 Secrecy Theorems . . . . .   | 182        |
| 21.3.1 Protocol goal: <i>serverK(Na,Nb,M)</i> and <i>clientK(Na,Nb,M)</i><br>remain secure . . . . .  | 183        |
| 21.3.2 Weakening the Oops conditions for leakage of <i>clientK</i> . . . . .  | 184        |
| 21.3.3 Weakening the Oops conditions for leakage of <i>serverK</i> . . . . .  | 184        |
| 21.3.4 Protocol goals: if A receives <i>ServerFinished</i> , then B is<br>present and has used the quoted values PA, PB, etc. Note<br>that it is up to A to compare PA with what she originally<br>sent. . . . .                | 185        |
| 21.3.5 Protocol goal: if B receives any message encrypted with<br><i>clientK</i> then A has sent it . . . . .   | 185        |
| 21.3.6 Protocol goal: if B receives <i>ClientFinished</i> , and if B is<br>able to check a <i>CertVerify</i> from A, then A has used the<br>quoted values PA, PB, etc. Even this one requires A to<br>be uncompromised. . . . . | 186        |
| <b>22 The Certified Electronic Mail Protocol by Abadi et al.</b>  | <b>186</b> |
| 22.1 Proving Confidentiality Results . . . . .  | 189        |
| 22.2 The Guarantees for Sender and Recipient . . . . .  | 191        |
| <b>23 Conventional protocols: rely on conventional Message, Event<br/>and Public – Public-key protocols</b>   | <b>192</b> |
| <b>24 Theory of Events for Security Protocols that use smartcards</b>   | <b>193</b> |
| 24.1 Function <i>knows</i> . . . . .  | 195        |
| 24.2 Knowledge of Agents . . . . .  | 197        |
| <b>25 Theory of smartcards</b>  | <b>199</b> |
| 25.1 Basic properties of <i>shrK</i> . . . . .  | 201        |
| 25.2 Function "knows" . . . . .   | 201        |
| 25.3 Fresh nonces . . . . .   | 203        |
| 25.4 Supply fresh nonces for possibility theorems. . . . .  | 203        |
| 25.5 Specialized Rewriting for Theorems About <i>analz</i> and <i>Image</i> . . . . .   | 204        |
| 25.6 Tactics for possibility theorems . . . . .   | 204        |
| <b>26 Original Shoup-Rubin protocol</b>   | <b>205</b> |
| <b>27 Bella's modification of the Shoup-Rubin protocol</b>  | <b>223</b> |

|  |            |
|--|------------|
| <b>28 Smartcard protocols: rely on conventional Message and on new EventSC and Smartcard</b> | <b>242</b> |
| <b>29 Extensions to Standard Theories</b>  | <b>242</b> |
| 29.1 Extensions to Theory <i>Set</i> . . . . .   | 242        |
| 29.2 Extensions to Theory <i>List</i> . . . . .  | 243        |
| 29.2.1 "remove l x" erase the first element of "l" equal to "x" . . .                        | 243        |
| 29.3 Extensions to Theory <i>Message</i> . . . . .   | 243        |
| 29.3.1 declarations for tactics . . . . .  | 243        |
| 29.3.2 extract the agent number of an Agent message . . . . .                                | 243        |
| 29.3.3 messages that are pairs . . . . .   | 243        |
| 29.3.4 well-foundedness of messages . . . . .  | 244        |
| 29.3.5 lemmas on keysFor . . . . .   | 244        |
| 29.3.6 lemmas on parts . . . . .   | 244        |
| 29.3.7 lemmas on synth . . . . .   | 245        |
| 29.3.8 lemmas on analz . . . . .   | 245        |
| 29.3.9 lemmas on parts, synth and analz . . . . .  | 246        |
| 29.3.10 greatest nonce used in a message . . . . .   | 246        |
| 29.3.11 sets of keys . . . . .   | 246        |
| 29.3.12 keys a priori necessary for decrypting the messages of G .                           | 246        |
| 29.3.13 only the keys necessary for G are useful in analz . . . . .                          | 247        |
| 29.4 Extensions to Theory <i>Event</i> . . . . .   | 247        |
| 29.4.1 general protocol properties . . . . .   | 247        |
| 29.4.2 lemma on knows . . . . .  | 248        |
| 29.4.3 knows without initState . . . . .   | 248        |
| 29.4.4 decomposition of knows into knows' and initState . . . . .                            | 248        |
| 29.4.5 knows' is finite . . . . .  | 249        |
| 29.4.6 monotonicity of knows . . . . .   | 249        |
| 29.4.7 maximum knowledge an agent can have includes messages<br>sent to the agent . . . . .  | 249        |
| 29.4.8 basic facts about <i>knows_max</i> . . . . .  | 250        |
| 29.4.9 used without initState . . . . .  | 250        |
| 29.4.10 monotonicity of used . . . . .   | 251        |
| 29.4.11 lemmas on used and knows . . . . .   | 252        |
| 29.4.12 a nonce or key in a message cannot equal a fresh nonce or<br>key . . . . .           | 252        |
| 29.4.13 message of an event . . . . .  | 252        |
| <b>30 Decomposition of Analz into two parts</b>  | <b>253</b> |
| 30.1 messages that do not contribute to analz . . . . .                                      | 253        |
| 30.2 basic facts about <i>pparts</i> . . . . .   | 253        |
| 30.3 facts about <i>pparts</i> and <i>parts</i> . . . . .                                    | 254        |
| 30.4 facts about <i>pparts</i> and <i>analz</i> . . . . .                                    | 255        |
| 30.5 messages that contribute to analz . . . . .   | 255        |
| 30.6 basic facts about <i>kparts</i> . . . . .   | 255        |
| 30.7 facts about <i>kparts</i> and <i>parts</i> . . . . .                                    | 256        |
| 30.8 facts about <i>kparts</i> and <i>analz</i> . . . . .                                    | 257        |
| 30.9 analz is pparts + analz of kparts . . . . .   | 257        |

|  |            |
|--|------------|
| <b>31 Protocol-Independent Confidentiality Theorem on Nonces</b>   | <b>258</b> |
| 31.1 basic facts about <i>guard</i>  | 258        |
| 31.2 guarded sets  | 259        |
| 31.3 basic facts about <i>Guard</i>  | 259        |
| 31.4 set obtained by decrypting a message  | 260        |
| 31.5 number of Crypt's in a message  | 261        |
| 31.6 basic facts about <i>crypt_nb</i>   | 261        |
| 31.7 number of Crypt's in a message list   | 261        |
| 31.8 basic facts about <i>cnb</i>  | 261        |
| 31.9 list of kparts  | 261        |
| 31.10 list corresponding to "decrypt"  | 261        |
| 31.11 basic facts about <i>decrypt'</i>  | 262        |
| 31.12 if the analyse of a finite guarded set gives n then it must also<br>gives one of the keys of Ks                    | 262        |
| 31.13 if the analyse of a finite guarded set and a (possibly infinite) set<br>of keys gives n then it must also gives Ks | 262        |
| <b>32 protocol-independent confidentiality theorem on keys</b>   | <b>262</b> |
| 32.1 basic facts about <i>guardK</i>   | 263        |
| 32.2 guarded sets  | 263        |
| 32.3 basic facts about <i>GuardK</i>   | 264        |
| 32.4 set obtained by decrypting a message  | 265        |
| 32.5 number of Crypt's in a message  | 265        |
| 32.6 basic facts about <i>crypt_nb</i>   | 265        |
| 32.7 number of Crypt's in a message list   | 265        |
| 32.8 basic facts about <i>cnb</i>  | 265        |
| 32.9 list of kparts  | 266        |
| 32.10 list corresponding to "decrypt"  | 266        |
| 32.11 basic facts about <i>decrypt'</i>  | 266        |
| 32.12 Basic properties of shrK   | 267        |
| 32.13 Function "knows"   | 268        |
| 32.14 Fresh nonces   | 268        |
| 32.15 Supply fresh nonces for possibility theorems.  | 268        |
| 32.16 Specialized Rewriting for Theorems About <i>analz</i> and Image  | 269        |
| 32.17 Tactics for possibility theorems   | 269        |
| <b>33 lemmas on guarded messages for protocols with symmetric keys</b>   | <b>270</b> |
| 33.1 Extensions to Theory <i>Shared</i>  | 270        |
| 33.1.1 a little abbreviation   | 270        |
| 33.1.2 agent associated to a key   | 270        |
| 33.1.3 basic facts about <i>initState</i>  | 270        |
| 33.1.4 sets of symmetric keys  | 271        |
| 33.1.5 sets of good keys   | 271        |
| 33.2 Proofs About Guarded Messages   | 271        |
| 33.2.1 small hack  | 271        |
| 33.2.2 guardedness results on nonces   | 271        |
| 33.2.3 guardedness results on keys   | 272        |
| 33.2.4 regular protocols   | 272        |

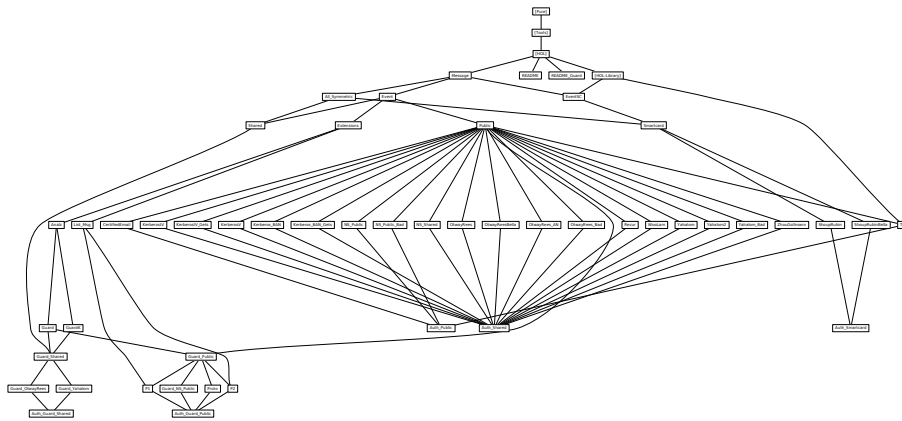
|   |            |
|---|------------|
| <b>34 Otway-Rees Protocol</b>   | <b>273</b> |
| 34.1 messages used in the protocol  | 273        |
| 34.2 definition of the protocol   | 274        |
| 34.3 declarations for tactics   | 274        |
| 34.4 general properties of or   | 274        |
| 34.5 or is regular  | 275        |
| 34.6 guardedness of KAB   | 275        |
| 34.7 guardedness of NB  | 275        |
| <b>35 Yahalom Protocol</b>  | <b>275</b> |
| 35.1 messages used in the protocol  | 275        |
| 35.2 definition of the protocol   | 276        |
| 35.3 declarations for tactics   | 276        |
| 35.4 general properties of ya   | 277        |
| 35.5 guardedness of KAB   | 277        |
| 35.6 session keys are not symmetric keys                                      | 277        |
| 35.7 ya2' implies ya1'  | 277        |
| 35.8 uniqueness of NB   | 277        |
| 35.9 ya3' implies ya2'  | 278        |
| 35.10 ya3' implies ya3  | 278        |
| 35.11 guardedness of NB   | 278        |
| <b>36 Blanqui's "guard" concept: protocol-independent secrecy</b>             | <b>278</b> |
| 36.1 Extensions to Theory <i>Public</i>                                       | 279        |
| 36.1.1 signature  | 279        |
| 36.1.2 agent associated to a key  | 279        |
| 36.1.3 basic facts about <i>initState</i>                                     | 279        |
| 36.1.4 sets of private keys   | 279        |
| 36.1.5 sets of good keys  | 280        |
| 36.1.6 greatest nonce used in a trace, 0 if there is no nonce                 | 280        |
| 36.1.7 function giving a new nonce  | 280        |
| 36.2 Proofs About Guarded Messages  | 280        |
| 36.2.1 small hack necessary because priK is defined as the inverse<br>of pubK | 280        |
| 36.2.2 guardedness results  | 281        |
| 36.2.3 regular protocols  | 281        |
| <b>37 Lists of Messages and Lists of Agents</b>                               | <b>281</b> |
| 37.1 Implementation of Lists by Messages                                      | 282        |
| 37.1.1 nil is represented by any message which is not a pair                  | 282        |
| 37.1.2 induction principle  | 282        |
| 37.1.3 head   | 282        |
| 37.1.4 tail   | 282        |
| 37.1.5 length   | 282        |
| 37.1.6 membership   | 282        |
| 37.1.7 delete an element  | 282        |
| 37.1.8 concatenation  | 282        |
| 37.1.9 replacement  | 283        |
| 37.1.10 ith element   | 283        |
| 37.1.11 insertion   | 283        |



|           |  |            |
|-----------|--|------------|
| 37.1.12   | truncation . . . . .   | 283        |
| 37.2      | Agent Lists . . . . .  | 283        |
| 37.2.1    | set of well-formed agent-list messages . . . . .   | 283        |
| 37.2.2    | basic facts about agent lists . . . . .  | 283        |
| <b>38</b> | <b>Protocol P1</b>   | <b>284</b> |
| 38.1      | Protocol Definition . . . . .  | 284        |
| 38.1.1    | offer chaining: B chains his offer for A with the head offer<br>of L for sending it to C . . . . . | 284        |
| 38.1.2    | agent whose key is used to sign an offer . . . . .   | 284        |
| 38.1.3    | nonce used in an offer . . . . .   | 285        |
| 38.1.4    | next shop . . . . .  | 285        |
| 38.1.5    | anchor of the offer list . . . . .   | 285        |
| 38.1.6    | request event . . . . .  | 285        |
| 38.1.7    | propose event . . . . .  | 286        |
| 38.1.8    | protocol . . . . .   | 286        |
| 38.1.9    | Composition of Traces . . . . .  | 286        |
| 38.1.10   | Valid Offer Lists . . . . .  | 287        |
| 38.1.11   | basic properties of valid . . . . .  | 287        |
| 38.1.12   | offers of an offer list . . . . .  | 287        |
| 38.1.13   | the originator can get the offers . . . . .  | 287        |
| 38.1.14   | list of offers . . . . .   | 287        |
| 38.1.15   | list of agents whose keys are used to sign a list of offers . . . . .                              | 287        |
| 38.1.16   | builds a trace from an itinerary . . . . .   | 287        |
| 38.1.17   | there is a trace in which the originator receives a valid<br>answer . . . . .                      | 288        |
| 38.2      | properties of protocol P1 . . . . .  | 288        |
| 38.2.1    | strong forward integrity: except the last one, no offer can<br>be modified . . . . .               | 288        |
| 38.2.2    | insertion resilience: except at the beginning, no offer can<br>be inserted . . . . .               | 288        |
| 38.2.3    | truncation resilience: only shop i can truncate at offer i . . . . .                               | 288        |
| 38.2.4    | declarations for tactics . . . . .   | 289        |
| 38.2.5    | get components of a message . . . . .  | 289        |
| 38.2.6    | general properties of p1 . . . . .   | 289        |
| 38.2.7    | private keys are safe . . . . .  | 289        |
| 38.2.8    | general guardedness properties . . . . .   | 290        |
| 38.2.9    | guardedness of messages . . . . .  | 290        |
| 38.2.10   | Nonce uniqueness . . . . .   | 291        |
| 38.2.11   | requests are guarded . . . . .   | 291        |
| 38.2.12   | propositions are guarded . . . . .   | 291        |
| 38.2.13   | data confidentiality: no one other than the originator can<br>decrypt the offers . . . . .         | 291        |
| 38.2.14   | non repudiability: an offer signed by B has been sent by B . . . . .                               | 292        |
| <b>39</b> | <b>Protocol P2</b>   | <b>292</b> |
| 39.1      | Protocol Definition . . . . .  | 292        |
| 39.1.1    | offer chaining: B chains his offer for A with the head offer<br>of L for sending it to C . . . . . | 292        |
| 39.1.2    | agent whose key is used to sign an offer . . . . .   | 293        |

|           |  |            |
|-----------|--|------------|
| 39.1.3    | nonce used in an offer . . . . .   | 293        |
| 39.1.4    | next shop . . . . .  | 293        |
| 39.1.5    | anchor of the offer list . . . . .   | 293        |
| 39.1.6    | request event . . . . .  | 293        |
| 39.1.7    | propose event . . . . .  | 294        |
| 39.1.8    | protocol . . . . .   | 294        |
| 39.1.9    | valid offer lists . . . . .  | 295        |
| 39.1.10   | basic properties of valid . . . . .  | 295        |
| 39.1.11   | list of offers . . . . .   | 295        |
| 39.2      | Properties of Protocol P2 . . . . .  | 295        |
| 39.3      | strong forward integrity: except the last one, no offer can be<br>modified . . . . .       | 295        |
| 39.4      | insertion resilience: except at the beginning, no offer can be inserted                    | 295        |
| 39.5      | truncation resilience: only shop i can truncate at offer i . . . . .                       | 295        |
| 39.6      | declarations for tactics . . . . .   | 296        |
| 39.7      | get components of a message . . . . .  | 296        |
| 39.8      | general properties of p2 . . . . .   | 296        |
| 39.9      | private keys are safe . . . . .  | 297        |
| 39.10     | general guardedness properties . . . . .   | 297        |
| 39.11     | guardedness of messages . . . . .  | 297        |
| 39.12     | Nonce uniqueness . . . . .   | 298        |
| 39.13     | requests are guarded . . . . .   | 298        |
| 39.14     | propositions are guarded . . . . .   | 298        |
| 39.15     | data confidentiality: no one other than the originator can decrypt<br>the offers . . . . . | 298        |
| 39.16     | forward privacy: only the originator can know the identity of the<br>shops . . . . .       | 299        |
| 39.17     | non repudiability: an offer signed by B has been sent by B . . . .                         | 299        |
| <b>40</b> | <b>Needham-Schroeder-Lowe Public-Key Protocol</b>  | <b>300</b> |
| 40.1      | messages used in the protocol . . . . .  | 300        |
| 40.2      | definition of the protocol . . . . .   | 300        |
| 40.3      | declarations for tactics . . . . .   | 300        |
| 40.4      | general properties of nsp . . . . .  | 301        |
| 40.5      | nonce are used only once . . . . .   | 301        |
| 40.6      | guardedness of NA . . . . .  | 301        |
| 40.7      | guardedness of NB . . . . .  | 302        |
| 40.8      | Agents' Authentication . . . . .   | 302        |
| <b>41</b> | <b>Other Protocol-Independent Results</b>  | <b>302</b> |
| 41.1      | protocols . . . . .  | 302        |
| 41.2      | substitutions . . . . .  | 302        |
| 41.3      | nonces generated by a rule . . . . .   | 303        |
| 41.4      | traces generated by a protocol . . . . .   | 304        |
| 41.5      | general properties . . . . .   | 304        |
| 41.6      | types . . . . .  | 304        |
| 41.7      | introduction of a fresh guarded nonce . . . . .  | 304        |
| 41.8      | safe keys . . . . .  | 305        |
| 41.9      | guardedness preservation . . . . .   | 306        |
| 41.10     | monotonic keyfun . . . . .   | 306        |

|   |            |
|---|------------|
| 41.11 guardedness theorem . . . . .                               | 306        |
| 41.12 useful properties for guardedness . . . . .                 | 306        |
| 41.13 unicity . . . . .   | 307        |
| 41.14 Needham-Schroeder-Lowe . . . . .                            | 308        |
| 41.15 general properties . . . . .                                | 309        |
| 41.16 guardedness for NSL . . . . .                               | 309        |
| 41.17 unicity for NSL . . . . .                                   | 309        |
| <b>42 Blanqui's "guard" concept: protocol-independent secrecy</b> | <b>309</b> |



# 1 Theory of Agents and Messages for Security Protocols

```
theory Message
imports Main
begin
```

```
lemma [simp] : "A  $\cup$  (B  $\cup$  A) = B  $\cup$  A"
  <proof>
```

```
type_synonym
  key = nat
```

```
consts
  all_symmetric :: bool          — true if all keys are symmetric
  invKey         :: "key  $\Rightarrow$  key" — inverse of a symmetric key
```

```
specification (invKey)
  invKey [simp]: "invKey (invKey K) = K"
  invKey_symmetric: "all_symmetric  $\longrightarrow$  invKey = id"
  <proof>
```

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

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

```
datatype — We allow any number of friendly agents
  agent = Server | Friend nat | Spy
```

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

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

```
syntax
  "_MTuple" :: "'a, args]  $\Rightarrow$  'a * 'b" (<(<indent=2 notation=<mixfix message
tuple>>{_,/ _}>>)
syntax_consts
  "_MTuple"  $\equiv$  MPair
translations
  "{x, y, z}"  $\equiv$  "{x, {y, z}}"
  "{x, y}"  $\equiv$  "CONST MPair x y"
```

```
definition HPair :: "[msg,msg]  $\Rightarrow$  msg" (<(4Hash[_] /_)> [0, 1000]) where
  — Message Y paired with a MAC computed with the help of X
```

"Hash[X] Y == Hash{X, Y}, Y"

**definition** keysFor :: "msg set  $\Rightarrow$  key set" **where**  
 — Keys useful to decrypt elements of a message set  
 "keysFor H == invKey ' {K.  $\exists X$ . Crypt K X  $\in$  H}"

## 1.1 Inductive Definition of All Parts of a Message

**inductive\_set**

parts :: "msg set  $\Rightarrow$  msg set"  
 for H :: "msg set"  
**where**  
 Inj [intro]: "X  $\in$  H  $\implies$  X  $\in$  parts H"  
 | Fst: " {X, Y}  $\in$  parts H  $\implies$  X  $\in$  parts H"  
 | Snd: " {X, Y}  $\in$  parts H  $\implies$  Y  $\in$  parts H"  
 | Body: "Crypt K X  $\in$  parts H  $\implies$  X  $\in$  parts H"

Monotonicity

**lemma** parts\_mono\_aux: "G  $\subseteq$  H; X  $\in$  parts G  $\implies$  X  $\in$  parts H"  
 <proof>

**lemma** parts\_mono: "G  $\subseteq$  H  $\implies$  parts(G)  $\subseteq$  parts(H)"  
 <proof>

Equations hold because constructors are injective.

**lemma** Friend\_image\_eq [simp]: "(Friend x  $\in$  Friend'A) = (x  $\in$  A)"  
 <proof>

**lemma** Key\_image\_eq [simp]: "(Key x  $\in$  Key'A) = (x  $\in$  A)"  
 <proof>

**lemma** Nonce\_Key\_image\_eq [simp]: "(Nonce x  $\notin$  Key'A)"  
 <proof>

## 1.2 Inverse of keys

**lemma** invKey\_eq [simp]: "(invKey K = invKey K') = (K=K')"  
 <proof>

## 1.3 The keysFor operator

**lemma** keysFor\_empty [simp]: "keysFor {} = {}"  
 <proof>

**lemma** keysFor\_Un [simp]: "keysFor (H  $\cup$  H') = keysFor H  $\cup$  keysFor H'"  
 <proof>

**lemma** keysFor\_UN [simp]: "keysFor ( $\bigcup i \in A$ . H i) = ( $\bigcup i \in A$ . keysFor (H i))"  
 <proof>

Monotonicity

**lemma** keysFor\_mono: "G  $\subseteq$  H  $\implies$  keysFor(G)  $\subseteq$  keysFor(H)"  
 <proof>

**lemma** `keysFor_insert_Agent [simp]: "keysFor (insert (Agent A) H) = keysFor H"`

*<proof>*

**lemma** `keysFor_insert_Nonce [simp]: "keysFor (insert (Nonce N) H) = keysFor H"`

*<proof>*

**lemma** `keysFor_insert_Number [simp]: "keysFor (insert (Number N) H) = keysFor H"`

*<proof>*

**lemma** `keysFor_insert_Key [simp]: "keysFor (insert (Key K) H) = keysFor H"`

*<proof>*

**lemma** `keysFor_insert_Hash [simp]: "keysFor (insert (Hash X) H) = keysFor H"`

*<proof>*

**lemma** `keysFor_insert_MPair [simp]: "keysFor (insert {X,Y} H) = keysFor H"`

*<proof>*

**lemma** `keysFor_insert_Crypt [simp]:`  
`"keysFor (insert (Crypt K X) H) = insert (invKey K) (keysFor H)"`

*<proof>*

**lemma** `keysFor_image_Key [simp]: "keysFor (Key'E) = {}"`

*<proof>*

**lemma** `Crypt_imp_invKey_keysFor: "Crypt K X ∈ H ⇒ invKey K ∈ keysFor H"`

*<proof>*

## 1.4 Inductive relation "parts"

**lemma** `MPair_parts:`  
`"[{X,Y}] ∈ parts H;`  
`[X ∈ parts H; Y ∈ parts H] ⇒ P] ⇒ P"`

*<proof>*

**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 ⊆ parts(H)"`

*<proof>*

**lemmas** `parts_insertI = subset_insertI [THEN parts_mono, THEN subsetD]`

**lemma** `parts_empty_aux: "X ∈ parts{} ⇒ False"`

*<proof>*

**lemma** *parts\_empty [simp]*: " $\text{parts}\{\} = \{\}$ "  
 <proof>

**lemma** *parts\_emptyE [elim!]*: " $X \in \text{parts}\{\} \implies P$ "  
 <proof>

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

**lemma** *parts\_singleton*: " $X \in \text{parts } H \implies \exists Y \in H. X \in \text{parts } \{Y\}$ "  
 <proof>

#### 1.4.1 Unions

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

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

TWO inserts to avoid looping. This rewrite is better than nothing. But its behaviour can be strange.

**lemma** *parts\_insert2*:  
 " $\text{parts } (\text{insert } X \ (\text{insert } Y \ H)) = \text{parts } \{X\} \cup \text{parts } \{Y\} \cup \text{parts } H$ "  
 <proof>

**lemma** *parts\_image [simp]*:  
 " $\text{parts } (f \ ` \ A) = (\bigcup x \in A. \text{parts } \{f \ x\})$ "  
 <proof>

Added to simplify arguments to *parts*, *analz* and *synth*.

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

**lemmas** *in\_parts\_UnE* = *parts\_Un [THEN equalityD1, THEN subsetD, THEN UnE]*

**declare** *in\_parts\_UnE [elim!]*

**lemma** *parts\_insert\_subset*: " $\text{insert } X \ (\text{parts } H) \subseteq \text{parts}(\text{insert } X \ H)$ "  
 <proof>

#### 1.4.2 Idempotence and transitivity

**lemma** *parts\_partsD [dest!]*: " $X \in \text{parts } (\text{parts } H) \implies X \in \text{parts } H$ "  
 <proof>

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

**lemma** *parts\_subset\_iff [simp]*: " $(\text{parts } G \subseteq \text{parts } H) = (G \subseteq \text{parts } H)$ "  
 <proof>

**lemma** *parts\_trans*: " $\llbracket X \in \text{parts } G; \ G \subseteq \text{parts } H \rrbracket \implies X \in \text{parts } H$ "  
 <proof>

Cut

**lemma** *parts\_cut*:

" $\llbracket Y \in \text{parts } (\text{insert } X \ G); \ X \in \text{parts } H \rrbracket \implies Y \in \text{parts } (G \cup H)$ "  
 ⟨*proof*⟩

**lemma** *parts\_cut\_eq [simp]*: " $X \in \text{parts } H \implies \text{parts } (\text{insert } X \ H) = \text{parts } H$ "  
 ⟨*proof*⟩

### 1.4.3 Rewrite rules for pulling out atomic messages

**lemmas** *parts\_insert\_eq\_I = equalityI [OF subsetI parts\_insert\_subset]*

**lemma** *parts\_insert\_Agent [simp]*:

" $\text{parts } (\text{insert } (\text{Agent } \text{agt}) \ H) = \text{insert } (\text{Agent } \text{agt}) \ (\text{parts } H)$ "  
 ⟨*proof*⟩

**lemma** *parts\_insert\_Nonce [simp]*:

" $\text{parts } (\text{insert } (\text{Nonce } N) \ H) = \text{insert } (\text{Nonce } N) \ (\text{parts } H)$ "  
 ⟨*proof*⟩

**lemma** *parts\_insert\_Number [simp]*:

" $\text{parts } (\text{insert } (\text{Number } N) \ H) = \text{insert } (\text{Number } N) \ (\text{parts } H)$ "  
 ⟨*proof*⟩

**lemma** *parts\_insert\_Key [simp]*:

" $\text{parts } (\text{insert } (\text{Key } K) \ H) = \text{insert } (\text{Key } K) \ (\text{parts } H)$ "  
 ⟨*proof*⟩

**lemma** *parts\_insert\_Hash [simp]*:

" $\text{parts } (\text{insert } (\text{Hash } X) \ H) = \text{insert } (\text{Hash } X) \ (\text{parts } H)$ "  
 ⟨*proof*⟩

**lemma** *parts\_insert\_Crypt [simp]*:

" $\text{parts } (\text{insert } (\text{Crypt } K \ X) \ H) = \text{insert } (\text{Crypt } K \ X) \ (\text{parts } (\text{insert } X \ H))$ "  
 ⟨*proof*⟩

**lemma** *parts\_insert\_MPair [simp]*:

" $\text{parts } (\text{insert } \llbracket X, Y \rrbracket \ H) = \text{insert } \llbracket X, Y \rrbracket \ (\text{parts } (\text{insert } X \ (\text{insert } Y \ H)))$ "  
 ⟨*proof*⟩

**lemma** *parts\_image\_Key [simp]*: " $\text{parts } (\text{Key}^{\prime} N) = \text{Key}^{\prime} N$ "

⟨*proof*⟩

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

**lemma** *msg\_Nonce\_supply*: " $\exists N. \forall n. N \leq n \longrightarrow \text{Nonce } n \notin \text{parts } \{\text{msg}\}$ "

⟨*proof*⟩

## 1.5 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  $\Rightarrow$  msg set"
for H :: "msg set"
where
  Inj [intro,simp]: "X  $\in$  H  $\Longrightarrow$  X  $\in$  analz H"
  | Fst:      "⟦X,Y⟧  $\in$  analz H  $\Longrightarrow$  X  $\in$  analz H"
  | Snd:      "⟦X,Y⟧  $\in$  analz H  $\Longrightarrow$  Y  $\in$  analz H"
  | Decrypt [dest]:
    "⟦Crypt K X  $\in$  analz H; Key(invKey K)  $\in$  analz H⟧  $\Longrightarrow$  X  $\in$  analz H"

```

Monotonicity; Lemma 1 of Lowe's paper

```

lemma analz_mono_aux: "⟦G  $\subseteq$  H; X  $\in$  analz G⟧  $\Longrightarrow$  X  $\in$  analz H"
  <proof>

```

```

lemma analz_mono: "G  $\subseteq$  H  $\Longrightarrow$  analz(G)  $\subseteq$  analz(H)"
  <proof>

```

Making it safe speeds up proofs

```

lemma MPair_analz [elim!]:
  "⟦⟦X,Y⟧  $\in$  analz H;
   ⟦X  $\in$  analz H; Y  $\in$  analz H⟧  $\Longrightarrow$  P⟧  $\Longrightarrow$  P"
  <proof>

```

```

lemma analz_increasing: "H  $\subseteq$  analz(H)"
  <proof>

```

```

lemma analz_into_parts: "X  $\in$  analz H  $\Longrightarrow$  X  $\in$  parts H"
  <proof>

```

```

lemma analz_subset_parts: "analz H  $\subseteq$  parts H"
  <proof>

```

```

lemma analz_parts [simp]: "analz (parts H) = parts H"
  <proof>

```

```

lemmas not_parts_not_analz = analz_subset_parts [THEN contra_subsetD]

```

```

lemma parts_analz [simp]: "parts (analz H) = parts H"
  <proof>

```

```

lemmas analz_insertI = subset_insertI [THEN analz_mono, THEN [2] rev_subsetD]

```

### 1.5.1 General equational properties

```

lemma analz_empty [simp]: "analz{} = {}"
  <proof>

```

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

```

lemma analz_Un: "analz(G)  $\cup$  analz(H)  $\subseteq$  analz(G  $\cup$  H)"
  <proof>

```

```

lemma analz_insert: "insert X (analz H)  $\subseteq$  analz(insert X H)"
  <proof>

```

## 1.5.2 Rewrite rules for pulling out atomic messages

lemmas *analz\_insert\_eq\_I* = *equalityI* [OF *subsetI* *analz\_insert*]

**lemma** *analz\_insert\_Agent* [simp]:  
 "analz (insert (Agent agt) H) = insert (Agent agt) (analz H)"  
 <proof>

**lemma** *analz\_insert\_Nonce* [simp]:  
 "analz (insert (Nonce N) H) = insert (Nonce N) (analz H)"  
 <proof>

**lemma** *analz\_insert\_Number* [simp]:  
 "analz (insert (Number N) H) = insert (Number N) (analz H)"  
 <proof>

**lemma** *analz\_insert\_Hash* [simp]:  
 "analz (insert (Hash X) H) = insert (Hash X) (analz H)"  
 <proof>

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

**lemma** *analz\_insert\_Key* [simp]:  
 "K  $\notin$  keysFor (analz H)  $\implies$   
     analz (insert (Key K) H) = insert (Key K) (analz H)"  
 <proof>

**lemma** *analz\_insert\_MPair* [simp]:  
 "analz (insert {X,Y} H) = insert {X,Y} (analz (insert X (insert Y H)))"  
 <proof>

Can pull out encrypted message if the Key is not known

**lemma** *analz\_insert\_Crypt*:  
 "Key (invKey K)  $\notin$  analz H  
      $\implies$  analz (insert (Crypt K X) H) = insert (Crypt K X) (analz H)"  
 <proof>

**lemma** *analz\_insert\_Decrypt*:  
 assumes "Key (invKey K)  $\in$  analz H"  
 shows "analz (insert (Crypt K X) H) = insert (Crypt K X) (analz (insert X H))"  
 <proof>

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

**lemma** *analz\_Crypt\_if* [simp]:  
 "analz (insert (Crypt K X) H) =  
     (if (Key (invKey K)  $\in$  analz H)  
       then insert (Crypt K X) (analz (insert X H))  
       else insert (Crypt K X) (analz H))"  
 <proof>

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))"
⟨proof⟩
```

```
lemma analz_image_Key [simp]: "analz (Key'N) = Key'N"
⟨proof⟩
```

### 1.5.3 Idempotence and transitivity

```
lemma analz_analzD [dest!]: "X ∈ analz (analz H) ⟹ X ∈ analz H"
⟨proof⟩
```

```
lemma analz_idem [simp]: "analz (analz H) = analz H"
⟨proof⟩
```

```
lemma analz_subset_iff [simp]: "(analz G ⊆ analz H) = (G ⊆ analz H)"
⟨proof⟩
```

```
lemma analz_trans: "[X ∈ analz G; G ⊆ analz H] ⟹ X ∈ analz H"
⟨proof⟩
```

Cut; Lemma 2 of Lowe

```
lemma analz_cut: "[Y ∈ analz (insert X H); X ∈ analz H] ⟹ Y ∈ analz H"
⟨proof⟩
```

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"
⟨proof⟩
```

A congruence rule for "analz"

```
lemma analz_subset_cong:
  "[analz G ⊆ analz G'; analz H ⊆ analz H']
   ⟹ analz (G ∪ H) ⊆ analz (G' ∪ H')"
⟨proof⟩
```

```
lemma analz_cong:
  "[analz G = analz G'; analz H = analz H']
   ⟹ analz (G ∪ H) = analz (G' ∪ H')"
⟨proof⟩
```

```
lemma analz_insert_cong:
  "analz H = analz H' ⟹ analz (insert X H) = analz (insert X H')"
⟨proof⟩
```

If there are no pairs or encryptions then analz does nothing

```
lemma analz_trivial:
  "[∀ X Y. {X, Y} ∉ H; ∀ X K. Crypt K X ∉ H] ⟹ analz H = H"
⟨proof⟩
```

## 1.6 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 ∈ H ⟹ X ∈ synth H"
  | Agent  [intro]:  "Agent agt ∈ synth H"
  | Number [intro]:  "Number n ∈ synth H"
  | Hash   [intro]:  "X ∈ synth H ⟹ Hash X ∈ synth H"
  | MPair  [intro]:  "[X ∈ synth H; Y ∈ synth H] ⟹ {X,Y} ∈ synth H"
  | Crypt  [intro]:  "[X ∈ synth H; Key(K) ∈ H] ⟹ Crypt K X ∈ synth H"

```

Monotonicity

```

lemma synth_mono: "G ⊆ H ⟹ synth(G) ⊆ synth(H)"
  <proof>

```

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

```

inductive_simps synth_simps [iff]:
  "Nonce n ∈ synth H"
  "Key K ∈ synth H"
  "Hash X ∈ synth H"
  "{X,Y} ∈ synth H"
  "Crypt K X ∈ synth H"

```

```

lemma synth_increasing: "H ⊆ synth(H)"
  <proof>

```

### 1.6.1 Unions

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

```

lemma synth_Un: "synth(G) ∪ synth(H) ⊆ synth(G ∪ H)"
  <proof>

```

```

lemma synth_insert: "insert X (synth H) ⊆ synth(insert X H)"
  <proof>

```

### 1.6.2 Idempotence and transitivity

```

lemma synth_synthD [dest!]: "X ∈ synth (synth H) ⟹ X ∈ synth H"
  <proof>

```

```

lemma synth_idem: "synth (synth H) = synth H"
  <proof>

```

```

lemma synth_subset_iff [simp]: "(synth G ⊆ synth H) = (G ⊆ synth H)"

```

$\langle proof \rangle$

**lemma** *synth\_trans*: " $\llbracket X \in \text{synth } G; \quad G \subseteq \text{synth } H \rrbracket \implies X \in \text{synth } H$ "  
 $\langle proof \rangle$

Cut; Lemma 2 of Lowe

**lemma** *synth\_cut*: " $\llbracket Y \in \text{synth } (\text{insert } X \ H); \quad X \in \text{synth } H \rrbracket \implies Y \in \text{synth } H$ "  
 $\langle proof \rangle$

**lemma** *Crypt\_synth\_eq [simp]*:  
 " $\text{Key } K \notin H \implies (\text{Crypt } K \ X \in \text{synth } H) = (\text{Crypt } K \ X \in H)$ "  
 $\langle proof \rangle$

**lemma** *keysFor\_synth [simp]*:  
 " $\text{keysFor } (\text{synth } H) = \text{keysFor } H \cup \text{invKey} \{K. \text{Key } K \in H\}$ "  
 $\langle proof \rangle$

### 1.6.3 Combinations of parts, analz and synth

**lemma** *parts\_synth [simp]*: " $\text{parts } (\text{synth } H) = \text{parts } H \cup \text{synth } H$ "  
 $\langle proof \rangle$

**lemma** *analz\_analz\_Un [simp]*: " $\text{analz } (\text{analz } G \cup H) = \text{analz } (G \cup H)$ "  
 $\langle proof \rangle$

**lemma** *analz\_synth\_Un [simp]*: " $\text{analz } (\text{synth } G \cup H) = \text{analz } (G \cup H) \cup \text{synth } G$ "  
 $\langle proof \rangle$

**lemma** *analz\_synth [simp]*: " $\text{analz } (\text{synth } H) = \text{analz } H \cup \text{synth } H$ "  
 $\langle proof \rangle$

### 1.6.4 For reasoning about the Fake rule in traces

**lemma** *parts\_insert\_subset\_Un*: " $X \in G \implies \text{parts}(\text{insert } X \ H) \subseteq \text{parts } G \cup \text{parts } H$ "  
 $\langle proof \rangle$

More specifically for Fake. See also *Fake\_parts\_sing* below

**lemma** *Fake\_parts\_insert*:  
 " $X \in \text{synth } (\text{analz } H) \implies$   
 $\text{parts } (\text{insert } X \ H) \subseteq \text{synth } (\text{analz } H) \cup \text{parts } H$ "  
 $\langle proof \rangle$

**lemma** *Fake\_parts\_insert\_in\_Un*:  
 " $\llbracket Z \in \text{parts } (\text{insert } X \ H); \quad X \in \text{synth } (\text{analz } H) \rrbracket$   
 $\implies Z \in \text{synth } (\text{analz } H) \cup \text{parts } H$ "  
 $\langle proof \rangle$

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

**lemma** *Fake\_analz\_insert*:  
 " $X \in \text{synth } (\text{analz } G) \implies$

```

      analz (insert X H)  $\subseteq$  synth (analz G)  $\cup$  analz (G  $\cup$  H)"
    <proof>

```

```

lemma analz_conj_parts [simp]:
  "(X  $\in$  analz H  $\wedge$  X  $\in$  parts H) = (X  $\in$  analz H)"
  <proof>

```

```

lemma analz_disj_parts [simp]:
  "(X  $\in$  analz H | X  $\in$  parts H) = (X  $\in$  parts H)"
  <proof>

```

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

```

lemma MPair_synth_analz [iff]:
  "⟦X,Y⟧  $\in$  synth (analz H)  $\longleftrightarrow$  X  $\in$  synth (analz H)  $\wedge$  Y  $\in$  synth (analz H)"
  <proof>

```

```

lemma Crypt_synth_analz:
  "⟦Key K  $\in$  analz H; Key (invKey K)  $\in$  analz H⟧
   $\implies$  (Crypt K X  $\in$  synth (analz H)) = (X  $\in$  synth (analz H))"
  <proof>

```

```

lemma Hash_synth_analz [simp]:
  "X  $\notin$  synth (analz H)
   $\implies$  (Hash ⟨X,Y⟩  $\in$  synth (analz H)) = (Hash ⟨X,Y⟩  $\in$  analz H)"
  <proof>

```

## 1.7 HPair: a combination of Hash and MPair

### 1.7.1 Freeness

```

lemma Agent_neq_HPpair: "Agent A  $\neq$  Hash[X] Y"
  <proof>

```

```

lemma Nonce_neq_HPpair: "Nonce N  $\neq$  Hash[X] Y"
  <proof>

```

```

lemma Number_neq_HPpair: "Number N  $\neq$  Hash[X] Y"
  <proof>

```

```

lemma Key_neq_HPpair: "Key K  $\neq$  Hash[X] Y"
  <proof>

```

```

lemma Hash_neq_HPpair: "Hash Z  $\neq$  Hash[X] Y"
  <proof>

```

```

lemma Crypt_neq_HPpair: "Crypt K X'  $\neq$  Hash[X] Y"
  <proof>

```

```

lemmas HPair_neqs = Agent_neq_HPpair Nonce_neq_HPpair Number_neq_HPpair
  Key_neq_HPpair Hash_neq_HPpair Crypt_neq_HPpair

```

```

declare HPair_neqs [iff]
declare HPair_neqs [symmetric, iff]

```

```
lemma HPair_eq [iff]: "(Hash[X'] Y' = Hash[X] Y) = (X' = X ∧ Y'=Y)"
  <proof>
```

```
lemma MPair_eq_HPair [iff]:
  "({X',Y'} = Hash[X] Y) = (X' = Hash{X,Y} ∧ Y'=Y)"
  <proof>
```

```
lemma HPair_eq_MPair [iff]:
  "(Hash[X] Y = {X',Y'}) = (X' = Hash{X,Y} ∧ Y'=Y)"
  <proof>
```

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

```
lemma keysFor_insert_HPair [simp]: "keysFor (insert (Hash[X] Y) H) = keysFor
H"
  <proof>
```

```
lemma parts_insert_HPair [simp]:
  "parts (insert (Hash[X] Y) H) =
    insert (Hash[X] Y) (insert (Hash{X,Y}) (parts (insert Y H)))"
  <proof>
```

```
lemma analz_insert_HPair [simp]:
  "analz (insert (Hash[X] Y) H) =
    insert (Hash[X] Y) (insert (Hash{X,Y}) (analz (insert Y H)))"
  <proof>
```

```
lemma HPair_synth_analz [simp]:
  "X ∉ synth (analz H)
  ⇒ (Hash[X] Y ∈ synth (analz H)) =
    (Hash {X, Y} ∈ analz H ∧ Y ∈ synth (analz H))"
  <proof>
```

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

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

Rewrites to push in Key and Crypt messages, so that other messages can be pulled out using the `analz_insert` rules

```
lemmas pushKeys =
  insert_commute [of "Key K" "Agent C"]
  insert_commute [of "Key K" "Nonce N"]
  insert_commute [of "Key K" "Number N"]
  insert_commute [of "Key K" "Hash X"]
  insert_commute [of "Key K" "MPair X Y"]
  insert_commute [of "Key K" "Crypt X K'"]
  for K C N X Y K'
```

```
lemmas pushCrypts =
  insert_commute [of "Crypt X K" "Agent C"]
  insert_commute [of "Crypt X K" "Agent C"]
  insert_commute [of "Crypt X K" "Nonce N"]
  insert_commute [of "Crypt X K" "Number N"]
  insert_commute [of "Crypt X K" "Hash X'"]
```

```

insert_commute [of "Crypt X K" "MPair X' Y"]
for X K C N X' Y

```

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

```

lemmas pushes = pushKeys pushCrypts

```

## 1.8 The set of key-free messages

```

inductive_set
  keyfree :: "msg set"
  where
    Agent: "Agent A ∈ keyfree"
  | Number: "Number N ∈ keyfree"
  | Nonce: "Nonce N ∈ keyfree"
  | Hash: "Hash X ∈ keyfree"
  | MPair: "[X ∈ keyfree; Y ∈ keyfree] ⇒ {X,Y} ∈ keyfree"
  | Crypt: "[X ∈ keyfree] ⇒ Crypt K X ∈ keyfree"

```

```

declare keyfree.intros [intro]

```

```

inductive_cases keyfree_KeyE: "Key K ∈ keyfree"
inductive_cases keyfree_MPairE: "{X,Y} ∈ keyfree"
inductive_cases keyfree_CryptE: "Crypt K X ∈ keyfree"

```

```

lemma parts_keyfree: "parts (keyfree) ⊆ keyfree"
  <proof>

```

```

lemma analz_keyfree_into_Un: "[X ∈ analz (G ∪ H); G ⊆ keyfree] ⇒ X ∈
parts G ∪ analz H"
  <proof>

```

## 1.9 Tactics useful for many protocol proofs

*<ML>*

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 ∉ Key ' A"
  <proof>

```

```

lemma Hash_notin_image_Key [simp]: "Hash X ∉ Key ' A"
  <proof>

```

```

lemma synth_analz_mono: "G ⊆ H ⇒ synth (analz(G)) ⊆ synth (analz(H))"
  <proof>

```

```

lemma Fake_analz_eq [simp]:
  "X ∈ synth(analz H) ⇒ synth (analz (insert X H)) = synth (analz H)"

```



$\langle proof \rangle$

Two generalizations of `analz_insert_eq`

```
lemma gen_analz_insert_eq [rule_format]:
  "X ∈ analz H ⇒ ∀ G. H ⊆ G → analz (insert X G) = analz G"
  ⟨proof⟩
```

```
lemma synth_analz_insert_eq:
  "[X ∈ synth (analz H); H ⊆ G]
    ⇒ (Key K ∈ analz (insert X G)) ↔ (Key K ∈ analz G)"
  ⟨proof⟩
```

```
lemma Fake_parts_sing:
  "X ∈ synth (analz H) ⇒ parts{X} ⊆ synth (analz H) ∪ parts H"
  ⟨proof⟩
```

```
lemmas Fake_parts_sing_imp_Un = Fake_parts_sing [THEN [2] rev_subsetD]
```

$\langle ML \rangle$

end

## 2 Theory of Events for Security Protocols

```
theory Event imports Message begin
```

```
consts — Initial states of agents — a parameter of the construction
  initState :: "agent ⇒ msg set"
```

```
datatype
  event = Says agent agent msg
        | Gets agent msg
        | Notes agent msg
```

```
consts
  bad :: "agent set" — compromised agents
```

Spy has access to his own key for spoof messages, but Server is secure

```
specification (bad)
  Spy_in_bad [iff]: "Spy ∈ bad"
  Server_not_bad [iff]: "Server ∉ bad"
  ⟨proof⟩
```

```
primrec knows :: "agent ⇒ event list ⇒ msg set"
where
```

```
  knows_Nil: "knows A [] = initState A"
| knows_Cons:
  "knows A (ev # evs) =
    (if A = Spy then
      (case ev of
        Says A' B X ⇒ insert X (knows Spy evs)
      | Gets A' X ⇒ knows Spy evs
      | Notes A' X ⇒
```

```

      if A' ∈ bad then insert X (knows Spy evs) else knows Spy evs)
    else
      (case ev of
        Says A' B X ⇒
          if A'=A then insert X (knows A evs) else knows A evs
        | Gets A' X   ⇒
          if A'=A then insert X (knows A evs) else knows A evs
        | Notes A' X  ⇒
          if A'=A then insert X (knows A evs) else knows A evs))"

```

The constant "spies" is retained for compatibility's sake

**abbreviation** (input)

```

spies :: "event list ⇒ msg set" where
  "spies ≡ knows Spy"

```

Set of items that might be visible to somebody: complement of the set of fresh items

```

primrec used :: "event list ⇒ msg set"
where
  used_Nil:   "used [] = (UN B. parts (initState B))"
| used_Cons: "used (ev # evs) =
  (case ev of
    Says A B X ⇒ parts {X} ∪ used evs
    | Gets A X  ⇒ used evs
    | Notes A X ⇒ parts {X} ∪ used evs)"

```

— The case for *Gets* seems anomalous, but *Gets* always follows *Says* in real protocols. Seems difficult to change. See *Gets\_correct* in theory *Guard/Extensions.thy*.

```

lemma Notes_imp_used: "Notes A X ∈ set evs ⇒ X ∈ used evs"
  <proof>

```

```

lemma Says_imp_used: "Says A B X ∈ set evs ⇒ X ∈ used evs"
  <proof>

```

## 2.1 Function *knows*

```

lemmas parts_insert_knows_A = parts_insert [of _ "knows A evs"] for A evs

```

```

lemma knows_Spy_Says [simp]:
  "knows Spy (Says A B X # evs) = insert X (knows Spy evs)"
  <proof>

```

Letting the Spy see "bad" agents' notes avoids redundant case-splits on whether  $A = \text{Spy}$  and whether  $A \in \text{bad}$

```

lemma knows_Spy_Notes [simp]:
  "knows Spy (Notes A X # evs) =
    (if A ∈ bad then insert X (knows Spy evs) else knows Spy evs)"
  <proof>

```

```

lemma knows_Spy_Gets [simp]: "knows Spy (Gets A X # evs) = knows Spy evs"
  <proof>

```

```

lemma knows_Spy_subset_knows_Spy_Says:

```

"knows Spy evs  $\subseteq$  knows Spy (Says A B X # evs)"  
 <proof>

**lemma** knows\_Spy\_subset\_knows\_Spy\_Notes:  
 "knows Spy evs  $\subseteq$  knows Spy (Notes A X # evs)"  
 <proof>

**lemma** knows\_Spy\_subset\_knows\_Spy\_Gets:  
 "knows Spy evs  $\subseteq$  knows Spy (Gets A X # evs)"  
 <proof>

Spy sees what is sent on the traffic

**lemma** Says\_imp\_knows\_Spy:  
 "Says A B X  $\in$  set evs  $\implies$  X  $\in$  knows Spy evs"  
 <proof>

**lemma** Notes\_imp\_knows\_Spy [rule\_format]:  
 "Notes A X  $\in$  set evs  $\implies$  A  $\in$  bad  $\implies$  X  $\in$  knows Spy evs"  
 <proof>

Elimination rules: derive contradictions from old Says events containing items known to be fresh

**lemmas** Says\_imp\_parts\_knows\_Spy =  
 Says\_imp\_knows\_Spy [THEN parts.Inj, elim\_format]

**lemmas** knows\_Spy\_partsEs =  
 Says\_imp\_parts\_knows\_Spy parts.Body [elim\_format]

**lemmas** Says\_imp\_analz\_Spy = Says\_imp\_knows\_Spy [THEN analz.Inj]

Compatibility for the old "spies" function

**lemmas** spies\_partsEs = knows\_Spy\_partsEs  
**lemmas** Says\_imp\_spies = Says\_imp\_knows\_Spy  
**lemmas** parts\_insert\_spies = parts\_insert\_knows\_A [of \_ Spy]

## 2.2 Knowledge of Agents

**lemma** knows\_subset\_knows\_Says: "knows A evs  $\subseteq$  knows A (Says A' B X # evs)"  
 <proof>

**lemma** knows\_subset\_knows\_Notes: "knows A evs  $\subseteq$  knows A (Notes A' X # evs)"  
 <proof>

**lemma** knows\_subset\_knows\_Gets: "knows A evs  $\subseteq$  knows A (Gets A' X # evs)"  
 <proof>

Agents know what they say

**lemma** Says\_imp\_knows [rule\_format]: "Says A B X  $\in$  set evs  $\implies$  X  $\in$  knows A evs"  
 <proof>

Agents know what they note

**lemma** *Notes\_imp\_knows [rule\_format]: "Notes A X ∈ set evs  $\implies$  X ∈ knows A evs"*  
*<proof>*

Agents know what they receive

**lemma** *Gets\_imp\_knows\_agents [rule\_format]:*  
*"A  $\neq$  Spy  $\implies$  Gets A X ∈ set evs  $\implies$  X ∈ knows A evs"*  
*<proof>*

What agents DIFFERENT FROM Spy know was either said, or noted, or got, or known initially

**lemma** *knows\_imp\_Says\_Gets\_Notes\_initState:*  
*"[X ∈ knows A evs; A  $\neq$  Spy]  $\implies$*   
 *$\exists$  B. Says A B X ∈ set evs  $\vee$  Gets A X ∈ set evs  $\vee$  Notes A X ∈ set evs*  
 *$\vee$  X ∈ initState A"*  
*<proof>*

What the Spy knows – for the time being – was either said or noted, or known initially

**lemma** *knows\_Spy\_imp\_Says\_Notes\_initState:*  
*"X ∈ knows Spy evs  $\implies$*   
 *$\exists$  A B. Says A B X ∈ set evs  $\vee$  Notes A X ∈ set evs  $\vee$  X ∈ initState Spy"*  
*<proof>*

**lemma** *parts\_knows\_Spy\_subset\_used: "parts (knows Spy evs)  $\subseteq$  used evs"*  
*<proof>*

**lemmas** *usedI = parts\_knows\_Spy\_subset\_used [THEN subsetD, intro]*

**lemma** *initState\_into\_used: "X ∈ parts (initState B)  $\implies$  X ∈ used evs"*  
*<proof>*

New simprules to replace the primitive ones for *used* and *knows*

**lemma** *used\_Says [simp]: "used (Says A B X # evs) = parts{X}  $\cup$  used evs"*  
*<proof>*

**lemma** *used\_Notes [simp]: "used (Notes A X # evs) = parts{X}  $\cup$  used evs"*  
*<proof>*

**lemma** *used\_Gets [simp]: "used (Gets A X # evs) = used evs"*  
*<proof>*

**lemma** *used\_nil\_subset: "used []  $\subseteq$  used evs"*  
*<proof>*

NOTE REMOVAL: the laws above are cleaner, as they don't involve "case"

**declare** *knows\_Cons [simp del]*  
*used\_Nil [simp del] used\_Cons [simp del]*

For proving theorems of the form  $X \notin \text{analz} (\text{knows Spy evs}) \longrightarrow P$  New events added by induction to "evs" are discarded. Provided this information isn't needed, the proof will be much shorter, since it will omit complicated reasoning about *analz*.

```

lemmas analz_mono_contra =
  knows_Spy_subset_knows_Spy_Says [THEN analz_mono, THEN contra_subsetD]
  knows_Spy_subset_knows_Spy_Notes [THEN analz_mono, THEN contra_subsetD]
  knows_Spy_subset_knows_Spy_Gets [THEN analz_mono, THEN contra_subsetD]

```

```

lemma knows_subset_knows_Cons: "knows A evs  $\subseteq$  knows A (e # evs)"
  <proof>

```

```

lemma initState_subset_knows: "initState A  $\subseteq$  knows A evs"
  <proof>

```

For proving *new\_keys\_not\_used*

```

lemma keysFor_parts_insert:
  "[K  $\in$  keysFor (parts (insert X G)); X  $\in$  synth (analz H)]
   $\implies$  K  $\in$  keysFor (parts (G  $\cup$  H)) | Key (invKey K)  $\in$  parts H"
  <proof>

```

```

lemmas analz_impI = impI [where P = "Y  $\notin$  analz (knows Spy evs)"] for Y evs
  <ML>

```

Useful for case analysis on whether a hash is a spoof or not

```

lemmas syan_impI = impI [where P = "Y  $\notin$  synth (analz (knows Spy evs))"]
  for Y evs
  <ML>

```

**end**

```

theory Public
imports Event
begin

```

```

lemma invKey_K: "K  $\in$  symKeys  $\implies$  invKey K = K"
  <proof>

```

## 2.3 Asymmetric Keys

```

datatype keymode = Signature | Encryption

```

```

consts
  publicKey :: "[keymode, agent]  $\Rightarrow$  key"

```

```

abbreviation
  pubEK :: "agent  $\Rightarrow$  key" where
    "pubEK == publicKey Encryption"

```

```

abbreviation
  pubSK :: "agent  $\Rightarrow$  key" where
    "pubSK == publicKey Signature"

```

**abbreviation**

```
privateKey :: "[keymode, agent]  $\Rightarrow$  key" where
  "privateKey b A == invKey (publicKey b A)"
```

**abbreviation**

```
priEK :: "agent  $\Rightarrow$  key" where
  "priEK A == privateKey Encryption A"
```

**abbreviation**

```
priSK :: "agent  $\Rightarrow$  key" where
  "priSK A == privateKey Signature A"
```

These abbreviations give backward compatibility. They represent the simple situation where the signature and encryption keys are the same.

**abbreviation (input)**

```
pubK :: "agent  $\Rightarrow$  key" where
  "pubK A == pubEK A"
```

**abbreviation (input)**

```
priK :: "agent  $\Rightarrow$  key" where
  "priK A == invKey (pubEK A)"
```

By freeness of agents, no two agents have the same key. Since *True*  $\neq$  *False*, no agent has identical signing and encryption keys

**specification (publicKey)**

```
injective_publicKey:
  "publicKey b A = publicKey c A'  $\implies$  b=c  $\wedge$  A=A'"
  <proof>
```

**axiomatization where**

```
privateKey_neq_publicKey [iff]: "privateKey b A  $\neq$  publicKey c A'"
```

```
lemmas publicKey_neq_privateKey = privateKey_neq_publicKey [THEN not_sym]
declare publicKey_neq_privateKey [iff]
```

**2.4 Basic properties of pubEK and priEK**

```
lemma publicKey_inject [iff]: "(publicKey b A = publicKey c A') = (b=c  $\wedge$  A=A' )"
  <proof>
```

```
lemma not_symKeys_pubK [iff]: "publicKey b A  $\notin$  symKeys"
  <proof>
```

```
lemma not_symKeys_priK [iff]: "privateKey b A  $\notin$  symKeys"
  <proof>
```

```
lemma symKey_neq_priEK: "K  $\in$  symKeys  $\implies$  K  $\neq$  priEK A"
  <proof>
```

**lemma** *symKeys\_neq\_imp\_neq*: " $(K \in \text{symKeys}) \neq (K' \in \text{symKeys}) \implies K \neq K'$ "  
 <proof>

**lemma** *symKeys\_invKey\_iff [iff]*: " $(\text{invKey } K \in \text{symKeys}) = (K \in \text{symKeys})$ "  
 <proof>

**lemma** *analz\_symKeys\_Decrypt*:  
 " $\llbracket \text{Crypt } K \ X \in \text{analz } H; \ K \in \text{symKeys}; \ \text{Key } K \in \text{analz } H \rrbracket$   
 $\implies X \in \text{analz } H$ "  
 <proof>

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

**lemma** *invKey\_image\_eq [simp]*: " $(\text{invKey } x \in \text{invKey } A) = (x \in A)$ "  
 <proof>

**lemma** *publicKey\_image\_eq [simp]*:  
 " $(\text{publicKey } b \ x \in \text{publicKey } c \ ' \ AA) = (b=c \ \wedge \ x \in AA)$ "  
 <proof>

**lemma** *privateKey\_notin\_image\_publicKey [simp]*: " $\text{privateKey } b \ x \notin \text{publicKey } c \ ' \ AA$ "  
 <proof>

**lemma** *privateKey\_image\_eq [simp]*:  
 " $(\text{privateKey } b \ A \in \text{invKey } c \ ' \ AS) = (b=c \ \wedge \ A \in AS)$ "  
 <proof>

**lemma** *publicKey\_notin\_image\_privateKey [simp]*: " $\text{publicKey } b \ A \notin \text{invKey } c \ ' \ AS$ "  
 <proof>

## 2.6 Symmetric Keys

For some protocols, it is convenient to equip agents with symmetric as well as asymmetric keys. The theory *Shared* assumes that all keys are symmetric.

**consts**  
*shrK* :: "agent => key" — long-term shared keys

**specification** (*shrK*)  
*inj\_shrK*: "inj *shrK*"  
 — No two agents have the same long-term key  
 <proof>

**axiomatization where**  
*sym\_shrK [iff]*: "*shrK* *X* ∈ *symKeys*" — All shared keys are symmetric

Injectiveness: Agents' long-term keys are distinct.

**lemmas** *shrK\_injective* = *inj\_shrK [THEN inj\_eq]*  
**declare** *shrK\_injective [iff]*

**lemma** *invKey\_shrK [simp]*: "*invKey* (*shrK* *A*) = *shrK* *A*"

*<proof>*

**lemma** *analz\_shrK\_Decrypt*:

"[[Crypt (shrK A) X ∈ analz H; Key(shrK A) ∈ analz H]] ⇒ X ∈ analz H"  
*<proof>*

**lemma** *analz\_Decrypt'*:

"[[Crypt K X ∈ analz H; K ∈ symKeys; Key K ∈ analz H]] ⇒ X ∈ analz H"  
*<proof>*

**lemma** *priK\_neq\_shrK [iff]*: "shrK A ≠ privateKey b C"  
*<proof>*

**lemmas** *shrK\_neq\_priK* = *priK\_neq\_shrK [THEN not\_sym]*  
**declare** *shrK\_neq\_priK [simp]*

**lemma** *pubK\_neq\_shrK [iff]*: "shrK A ≠ publicKey b C"  
*<proof>*

**lemmas** *shrK\_neq\_pubK* = *pubK\_neq\_shrK [THEN not\_sym]*  
**declare** *shrK\_neq\_pubK [simp]*

**lemma** *priEK\_noteq\_shrK [simp]*: "priEK A ≠ shrK B"  
*<proof>*

**lemma** *publicKey\_notin\_image\_shrK [simp]*: "publicKey b x ∉ shrK ' AA"  
*<proof>*

**lemma** *privateKey\_notin\_image\_shrK [simp]*: "privateKey b x ∉ shrK ' AA"  
*<proof>*

**lemma** *shrK\_notin\_image\_publicKey [simp]*: "shrK x ∉ publicKey b ' AA"  
*<proof>*

**lemma** *shrK\_notin\_image\_privateKey [simp]*: "shrK x ∉ invKey ' publicKey b ' AA"  
*<proof>*

**lemma** *shrK\_image\_eq [simp]*: "(shrK x ∈ shrK ' AA) = (x ∈ AA)"  
*<proof>*

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

**declare** *invKey\_K [simp]*

## 2.7 Initial States of Agents

Note: for all practical purposes, all that matters is the initial knowledge of the Spy. All other agents are automata, merely following the protocol.

**overloading**

*initState* ≡ *initState*

**begin**



**primrec** *initState* **where**

```

initState_Server:
  "initState Server      =
    {Key (priEK Server), Key (priSK Server)} ∪
    (Key ' range pubEK) ∪ (Key ' range pubSK) ∪ (Key ' range shrK)"

| initState_Friend:
  "initState (Friend i) =
    {Key (priEK(Friend i)), Key (priSK(Friend i)), Key (shrK(Friend i))}
  ∪
    (Key ' range pubEK) ∪ (Key ' range pubSK)"

| initState_Spy:
  "initState Spy      =
    (Key ' invKey ' pubEK ' bad) ∪ (Key ' invKey ' pubSK ' bad) ∪
    (Key ' shrK ' bad) ∪
    (Key ' range pubEK) ∪ (Key ' range pubSK)"

```

**end**

These lemmas allow reasoning about *used evs* rather than *knows Spy evs*, which is useful when there are private Notes. Because they depend upon the definition of *initState*, they cannot be moved up.

**lemma** *used\_parts\_subset\_parts [rule\_format]:*  
 "∀ *X* ∈ *used evs*. *parts {X}* ⊆ *used evs*"  
 <proof>

**lemma** *MPair\_used\_D: "⌊X,Y⌋ ∈ used H ⇒ X ∈ used H ∧ Y ∈ used H"*  
 <proof>

There was a similar theorem in *Event.thy*, so perhaps this one can be moved up if proved directly by induction.

**lemma** *MPair\_used [elim!]:*  
 "⌊X,Y⌋ ∈ used H;  
 ⌊X ∈ used H; Y ∈ used H⌋ ⇒ *P*  
 ⇒ *P*"  
 <proof>

Rewrites should not refer to *initState (Friend i)* because that expression is not in normal form.

**lemma** *keysFor\_parts\_initState [simp]: "keysFor (parts (initState C)) = {}"*  
 <proof>

**lemma** *Crypt\_notin\_initState: "Crypt K X ∉ parts (initState B)"*  
 <proof>

**lemma** *Crypt\_notin\_used\_empty [simp]: "Crypt K X ∉ used []"*  
 <proof>

**lemma** *shrK\_in\_initState* [iff]: "Key (shrK A)  $\in$  initState A"  
 <proof>

**lemma** *shrK\_in\_knows* [iff]: "Key (shrK A)  $\in$  knows A evs"  
 <proof>

**lemma** *shrK\_in\_used* [iff]: "Key (shrK A)  $\in$  used evs"  
 <proof>

**lemma** *Key\_not\_used* [simp]: "Key K  $\notin$  used evs  $\implies$  K  $\notin$  range shrK"  
 <proof>

**lemma** *shrK\_neq*: "Key K  $\notin$  used evs  $\implies$  shrK B  $\neq$  K"  
 <proof>

**lemmas** *neq\_shrK* = *shrK\_neq* [THEN not\_sym]  
**declare** *neq\_shrK* [simp]

## 2.8 Function knows Spy

**lemma** *not\_SignatureE* [elim!]: "b  $\neq$  Signature  $\implies$  b = Encryption"  
 <proof>

Agents see their own private keys!

**lemma** *priK\_in\_initState* [iff]: "Key (privateKey b A)  $\in$  initState A"  
 <proof>

Agents see all public keys!

**lemma** *publicKey\_in\_initState* [iff]: "Key (publicKey b A)  $\in$  initState B"  
 <proof>

All public keys are visible

**lemma** *spies\_pubK* [iff]: "Key (publicKey b A)  $\in$  spies evs"  
 <proof>

**lemmas** *analz\_spies\_pubK* = *spies\_pubK* [THEN analz.Inj]  
**declare** *analz\_spies\_pubK* [iff]

Spy sees private keys of bad agents!

**lemma** *Spy\_spies\_bad\_privateKey* [intro!]:  
 "A  $\in$  bad  $\implies$  Key (privateKey b A)  $\in$  spies evs"  
 <proof>

Spy sees long-term shared keys of bad agents!

**lemma** *Spy\_spies\_bad\_shrK* [intro!]:  
 "A  $\in$  bad  $\implies$  Key (shrK A)  $\in$  spies evs"  
 <proof>

**lemma** *publicKey\_into\_used* [iff] : "Key (publicKey b A)  $\in$  used evs"

*<proof>*

**lemma** *privateKey\_into\_used [iff]: "Key (privateKey b A) ∈ used evs"*  
*<proof>*

**lemma** *Crypt\_Spy\_analz\_bad:*  
 "⟦Crypt (shrK A) X ∈ analz (knows Spy evs); A ∈ bad⟧  
 ⇒ X ∈ analz (knows Spy evs)"  
*<proof>*

## 2.9 Fresh Nonces

**lemma** *Nonce\_notin\_initState [iff]: "Nonce N ∉ parts (initState B)"*  
*<proof>*

**lemma** *Nonce\_notin\_used\_empty [simp]: "Nonce N ∉ used []"*  
*<proof>*

## 2.10 Supply fresh nonces for possibility theorems

In any trace, there is an upper bound N on the greatest nonce in use

**lemma** *Nonce\_supply\_lemma: "∃N. ∀n. N ≤ n ⇒ Nonce n ∉ used evs"*  
*<proof>*

**lemma** *Nonce\_supply1: "∃N. Nonce N ∉ used evs"*  
*<proof>*

**lemma** *Nonce\_supply: "Nonce (SOME N. Nonce N ∉ used evs) ∉ used evs"*  
*<proof>*

## 2.11 Specialized Rewriting for Theorems About analz and Image

**lemma** *insert\_Key\_singleton: "insert (Key K) H = Key ' {K} ∪ H"*  
*<proof>*

**lemma** *insert\_Key\_image: "insert (Key K) (Key 'KK ∪ C) = Key ' (insert K KK) ∪ C"*  
*<proof>*

**lemma** *Crypt\_imp\_keysFor : "⟦Crypt K X ∈ H; K ∈ symKeys⟧ ⇒ K ∈ keysFor H"*  
*<proof>*

Lemma for the trivial direction of the if-and-only-if of the Session Key Compromise Theorem

**lemma** *analz\_image\_freshK\_lemma:*  
 "(Key K ∈ analz (Key 'nE ∪ H)) ⇒ (K ∈ nE | Key K ∈ analz H) ⇒  
 (Key K ∈ analz (Key 'nE ∪ H)) = (K ∈ nE | Key K ∈ analz H)"  
*<proof>*

```

lemmas analz_image_freshK_simps =
  simp_thms mem_simps — these two allow its use with only:
  disj_comms
  image_insert [THEN sym] image_Un [THEN sym] empty_subsetI insert_subset
  analz_insert_eq Un_upper2 [THEN analz_mono, THEN subsetD]
  insert_Key_singleton
  Key_not_used insert_Key_image Un_assoc [THEN sym]

```

⟨ML⟩

## 2.12 Specialized Methods for Possibility Theorems

⟨ML⟩

**end**

## 3 Needham-Schroeder Shared-Key Protocol

```

theory NS_Shared imports Public begin

```

From page 247 of Burrows, Abadi and Needham (1989). A Logic of Authentication. Proc. Royal Soc. 426

**definition**

```

  Issues :: "[agent, agent, msg, event list] ⇒ bool"
    (<_ Issues _ with _ on _>) where
    "A Issues B with X on evs =
      (∃ Y. Says A B Y ∈ set evs ∧ X ∈ parts {Y} ∧
        X ∉ parts (spies (takeWhile (λz. z ≠ Says A B Y) (rev evs))))"

```

```

inductive_set ns_shared :: "event list set"

```

**where**

```

  Nil: "[] ∈ ns_shared"

```

```

  / Fake: "[[evsf ∈ ns_shared; X ∈ synth (analz (spies evsf))]]
    ⇒ Says Spy B X # evsf ∈ ns_shared"

```

```

  / NS1: "[[evs1 ∈ ns_shared; Nonce NA ∉ used evs1]]
    ⇒ Says A Server {Agent A, Agent B, Nonce NA} # evs1 ∈ ns_shared"

```

```

  / NS2: "[[evs2 ∈ ns_shared; Key KAB ∉ used evs2; KAB ∈ symKeys;
    Says A' Server {Agent A, Agent B, Nonce NA} ∈ set evs2]]
    ⇒ Says Server A
      (Crypt (shrK A)
        {Nonce NA, Agent B, Key KAB,
          (Crypt (shrK B) {Key KAB, Agent A})})
      # evs2 ∈ ns_shared"

```

```

/ NS3: "[[evs3 ∈ ns_shared; A ≠ Server;
        Says S A (Crypt (shrK A) {Nonce NA, Agent B, Key K, X}) ∈ set evs3;
        Says A Server {Agent A, Agent B, Nonce NA} ∈ set evs3]]
        ⇒ Says A B X # evs3 ∈ ns_shared"

/ NS4: "[[evs4 ∈ ns_shared; Nonce NB ∉ used evs4; K ∈ symKeys;
        Says A' B (Crypt (shrK B) {Key K, Agent A}) ∈ set evs4]]
        ⇒ Says B A (Crypt K (Nonce NB)) # evs4 ∈ ns_shared"

/ NS5: "[[evs5 ∈ ns_shared; K ∈ symKeys;
        Says B' A (Crypt K (Nonce NB)) ∈ set evs5;
        Says S A (Crypt (shrK A) {Nonce NA, Agent B, Key K, X})
          ∈ set evs5]]
        ⇒ Says A B (Crypt K {Nonce NB, Nonce NB}) # evs5 ∈ ns_shared"

/ Ops: "[[evso ∈ ns_shared; Says B A (Crypt K (Nonce NB)) ∈ set evso;
        Says Server A (Crypt (shrK A) {Nonce NA, Agent B, Key K, X})
          ∈ set evso]]
        ⇒ Notes Spy {Nonce NA, Nonce NB, Key K} # evso ∈ ns_shared"

```

```

declare Says_imp_knows_Spy [THEN parts.Inj, dest]
declare parts.Body [dest]
declare Fake_parts_insert_in_Un [dest]
declare analz_into_parts [dest]

```

A "possibility property": there are traces that reach the end

```

lemma "[A ≠ Server; Key K ∉ used []; K ∈ symKeys]
       ⇒ ∃ N. ∃ evs ∈ ns_shared.
          Says A B (Crypt K {Nonce N, Nonce N}) ∈ set evs"
<proof>

```

### 3.1 Inductive proofs about *ns\_shared*

#### 3.1.1 Forwarding lemmas, to aid simplification

For reasoning about the encrypted portion of message NS3

```

lemma NS3_msg_in_parts_spies:
  "Says S A (Crypt KA {N, B, K, X}) ∈ set evs ⇒ X ∈ parts (spies evs)"
<proof>

```

For reasoning about the Oops message

```

lemma Oops_parts_spies:
  "Says Server A (Crypt (shrK A) {NA, B, K, X}) ∈ set evs
   ⇒ K ∈ parts (spies evs)"
<proof>

```

Theorems of the form  $X \notin \text{parts } (\text{knows } \text{Spy } \text{evs})$  imply that NOBODY sends messages containing  $X$

Spy never sees another agent's shared key! (unless it's bad at start)

**lemma** *Spy\_see\_shrK [simp]:*  
 "evs ∈ ns\_shared ⇒ (Key (shrK A) ∈ parts (spies evs)) = (A ∈ bad)"  
 ⟨proof⟩

**lemma** *Spy\_analz\_shrK [simp]:*  
 "evs ∈ ns\_shared ⇒ (Key (shrK A) ∈ analz (spies evs)) = (A ∈ bad)"  
 ⟨proof⟩

Nobody can have used non-existent keys!

**lemma** *new\_keys\_not\_used [simp]:*  
 "Key K ∉ used evs; K ∈ symKeys; evs ∈ ns\_shared  
 ⇒ K ∉ keysFor (parts (spies evs))"  
 ⟨proof⟩

### 3.1.2 Lemmas concerning the form of items passed in messages

Describes the form of K, X and K' when the Server sends this message.

**lemma** *Says\_Server\_message\_form:*  
 "Says Server A (Crypt K' {N, Agent B, Key K, X}) ∈ set evs;  
 evs ∈ ns\_shared  
 ⇒ K ∉ range shrK ∧  
 X = (Crypt (shrK B) {Key K, Agent A}) ∧  
 K' = shrK A"  
 ⟨proof⟩

If the encrypted message appears then it originated with the Server

**lemma** *A\_trusts\_NS2:*  
 "Crypt (shrK A) {NA, Agent B, Key K, X} ∈ parts (spies evs);  
 A ∉ bad; evs ∈ ns\_shared  
 ⇒ Says Server A (Crypt (shrK A) {NA, Agent B, Key K, X}) ∈ set evs"  
 ⟨proof⟩

**lemma** *cert\_A\_form:*  
 "Crypt (shrK A) {NA, Agent B, Key K, X} ∈ parts (spies evs);  
 A ∉ bad; evs ∈ ns\_shared  
 ⇒ K ∉ range shrK ∧ X = (Crypt (shrK B) {Key K, Agent A})"  
 ⟨proof⟩

EITHER describes the form of X when the following message is sent, OR reduces it to the Fake case. Use *Says\_Server\_message\_form* if applicable.

**lemma** *Says\_S\_message\_form:*  
 "Says S A (Crypt (shrK A) {Nonce NA, Agent B, Key K, X}) ∈ set evs;  
 evs ∈ ns\_shared  
 ⇒ (K ∉ range shrK ∧ X = (Crypt (shrK B) {Key K, Agent A}))  
 ∨ X ∈ analz (spies evs)"  
 ⟨proof⟩

NOT useful in this form, but it says that session keys are not used to encrypt messages containing other keys, in the actual protocol. We require that agents should behave like this subsequently also.

**lemma** "evs ∈ ns\_shared; Kab ∉ range shrK ⇒  
 (Crypt KAB X) ∈ parts (spies evs) ∧

$\text{Key } K \in \text{parts } \{X\} \longrightarrow \text{Key } K \in \text{parts } (\text{spies } \text{evs})"$   
 $\langle \text{proof} \rangle$

### 3.1.3 Session keys are not used to encrypt other session keys

The equality makes the induction hypothesis easier to apply

**lemma** *analz\_image\_freshK* [rule\_format]:

" $\text{evs} \in \text{ns\_shared} \implies$   
 $\forall K \text{ KK. } \text{KK} \subseteq - (\text{range } \text{shrK}) \longrightarrow$   
 $(\text{Key } K \in \text{analz } (\text{Key}'\text{KK} \cup (\text{spies } \text{evs}))) =$   
 $(K \in \text{KK} \vee \text{Key } K \in \text{analz } (\text{spies } \text{evs}))"$

$\langle \text{proof} \rangle$

**lemma** *analz\_insert\_freshK*:

" $\llbracket \text{evs} \in \text{ns\_shared}; \text{KAB} \notin \text{range } \text{shrK} \rrbracket \implies$   
 $(\text{Key } K \in \text{analz } (\text{insert } (\text{Key } \text{KAB}) (\text{spies } \text{evs}))) =$   
 $(K = \text{KAB} \vee \text{Key } K \in \text{analz } (\text{spies } \text{evs}))"$

$\langle \text{proof} \rangle$

### 3.1.4 The session key K uniquely identifies the message

In messages of this form, the session key uniquely identifies the rest

**lemma** *unique\_session\_keys*:

" $\llbracket \text{Says Server } A \text{ (Crypt (shrK } A) \llbracket \text{NA, Agent } B, \text{Key } K, X \rrbracket)} \in \text{set } \text{evs};$   
 $\text{Says Server } A' \text{ (Crypt (shrK } A') \llbracket \text{NA}', \text{Agent } B', \text{Key } K, X' \rrbracket)} \in \text{set } \text{evs};$   
 $\text{evs} \in \text{ns\_shared} \rrbracket \implies A=A' \wedge \text{NA}=\text{NA}' \wedge B=B' \wedge X=X'"$

$\langle \text{proof} \rangle$

### 3.1.5 Crucial secrecy property: Spy doesn't see the keys sent in NS2

Beware of [rule\_format] and the universal quantifier!

**lemma** *secrecy\_lemma*:

" $\llbracket \text{Says Server } A \text{ (Crypt (shrK } A) \llbracket \text{NA, Agent } B, \text{Key } K,$   
 $\text{Crypt (shrK } B) \llbracket \text{Key } K, \text{Agent } A \rrbracket \rrbracket)} \in \text{set } \text{evs};$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{ns\_shared} \rrbracket$   
 $\implies (\forall \text{NB. Notes Spy } \llbracket \text{NA, NB, Key } K \rrbracket \notin \text{set } \text{evs}) \longrightarrow$   
 $\text{Key } K \notin \text{analz } (\text{spies } \text{evs})"$

$\langle \text{proof} \rangle$

Final version: Server's message in the most abstract form

**lemma** *Spy\_not\_see\_encrypted\_key*:

" $\llbracket \text{Says Server } A \text{ (Crypt } K' \llbracket \text{NA, Agent } B, \text{Key } K, X \rrbracket)} \in \text{set } \text{evs};$   
 $\forall \text{NB. Notes Spy } \llbracket \text{NA, NB, Key } K \rrbracket \notin \text{set } \text{evs};$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{ns\_shared} \rrbracket$   
 $\implies \text{Key } K \notin \text{analz } (\text{spies } \text{evs})"$

$\langle \text{proof} \rangle$

## 3.2 Guarantees available at various stages of protocol

If the encrypted message appears then it originated with the Server

**lemma** *B\_trusts\_NS3*:

```
"[[Crypt (shrK B) {Key K, Agent A} ∈ parts (spies evs);
  B ∉ bad; evs ∈ ns_shared]]
⇒ ∃ NA. Says Server A
    (Crypt (shrK A) {NA, Agent B, Key K,
                    Crypt (shrK B) {Key K, Agent A}})
  ∈ set evs"
```

*<proof>*

**lemma** *A\_trusts\_NS4\_lemma* [rule\_format]:

```
"evs ∈ ns_shared ⇒
  Key K ∉ analz (spies evs) →
  Says Server A (Crypt (shrK A) {NA, Agent B, Key K, X}) ∈ set evs →
  Crypt K (Nonce NB) ∈ parts (spies evs) →
  Says B A (Crypt K (Nonce NB)) ∈ set evs"
```

*<proof>*

This version no longer assumes that K is secure

**lemma** *A\_trusts\_NS4*:

```
"[[Crypt K (Nonce NB) ∈ parts (spies evs);
  Crypt (shrK A) {NA, Agent B, Key K, X} ∈ parts (spies evs);
  ∀ NB. Notes Spy {NA, NB, Key K} ∉ set evs;
  A ∉ bad; B ∉ bad; evs ∈ ns_shared]]
⇒ Says B A (Crypt K (Nonce NB)) ∈ set evs"
```

*<proof>*

If the session key has been used in NS4 then somebody has forwarded component X in some instance of NS4. Perhaps an interesting property, but not needed (after all) for the proofs below.

**theorem** *NS4\_implies\_NS3* [rule\_format]:

```
"evs ∈ ns_shared ⇒
  Key K ∉ analz (spies evs) →
  Says Server A (Crypt (shrK A) {NA, Agent B, Key K, X}) ∈ set evs →
  Crypt K (Nonce NB) ∈ parts (spies evs) →
  (∃ A'. Says A' B X ∈ set evs)"
```

*<proof>*

**lemma** *B\_trusts\_NS5\_lemma* [rule\_format]:

```
"[[B ∉ bad; evs ∈ ns_shared]] ⇒
  Key K ∉ analz (spies evs) →
  Says Server A
    (Crypt (shrK A) {NA, Agent B, Key K,
                    Crypt (shrK B) {Key K, Agent A}}) ∈ set evs →
  Crypt K {Nonce NB, Nonce NB} ∈ parts (spies evs) →
  Says A B (Crypt K {Nonce NB, Nonce NB}) ∈ set evs"
```

*<proof>*

Very strong Oops condition reveals protocol's weakness

**lemma** *B\_trusts\_NS5*:

```
"[[Crypt K {Nonce NB, Nonce NB} ∈ parts (spies evs);
  Crypt (shrK B) {Key K, Agent A} ∈ parts (spies evs);
```



```

     $\forall NA\ NB. \text{Notes Spy } \{NA, NB, \text{Key } K\} \notin \text{set evs};$ 
     $A \notin \text{bad};\ B \notin \text{bad};\ \text{evs} \in \text{ns\_shared}$ 
 $\implies \text{Says } A\ B\ (\text{Crypt } K\ \{\text{Nonce } NB, \text{Nonce } NB\}) \in \text{set evs}$ 
<proof>

```

Unaltered so far wrt original version

### 3.3 Lemmas for reasoning about predicate "Issues"

```

lemma spies_Says_rev: "spies (evs @ [Says A B X]) = insert X (spies evs)"
<proof>

```

```

lemma spies_Gets_rev: "spies (evs @ [Gets A X]) = spies evs"
<proof>

```

```

lemma spies_Notes_rev: "spies (evs @ [Notes A X]) =
    (if A ∈ bad then insert X (spies evs) else spies evs)"
<proof>

```

```

lemma spies_evs_rev: "spies evs = spies (rev evs)"
<proof>

```

```

lemmas parts_spies_evs_revD2 = spies_evs_rev [THEN equalityD2, THEN parts_mono]

```

```

lemma spies_takeWhile: "spies (takeWhile P evs) ⊆ spies evs"
<proof>

```

```

lemmas parts_spies_takeWhile_mono = spies_takeWhile [THEN parts_mono]

```

### 3.4 Guarantees of non-injective agreement on the session key, and of key distribution. They also express forms of freshness of certain messages, namely that agents were alive after something happened.

```

lemma B_Issues_A:
  "[[ Says B A (Crypt K (Nonce Nb)) ∈ set evs;
    Key K ∉ analz (spies evs);
    A ∉ bad; B ∉ bad; evs ∈ ns_shared ]]
   $\implies B \text{ Issues } A \text{ with } (\text{Crypt } K\ (\text{Nonce } Nb)) \text{ on evs}$ "
<proof>

```

Tells A that B was alive after she sent him the session key. The session key must be assumed confidential for this deduction to be meaningful, but that assumption can be relaxed by the appropriate argument.

Precisely, the theorem guarantees (to A) key distribution of the session key to B. It also guarantees (to A) non-injective agreement of B with A on the session key. Both goals are available to A in the sense of Goal Availability.

```

lemma A_authenticates_and_keydist_to_B:
  "[[Crypt K (Nonce NB) ∈ parts (spies evs);
    Crypt (shrK A) {NA, Agent B, Key K, X} ∈ parts (spies evs);
    Key K ∉ analz(knowns Spy evs);
    A ∉ bad; B ∉ bad; evs ∈ ns_shared]]

```

$\implies B \text{ Issues } A \text{ with } (\text{Crypt } K (\text{Nonce } NB)) \text{ on } \text{evs}"$   
 $\langle \text{proof} \rangle$

**lemma** *A\_trusts\_NS5*:

"[[  $\text{Crypt } K \{ \text{Nonce } NB, \text{Nonce } NB \} \in \text{parts}(\text{spies } \text{evs});$   
 $\text{Crypt } (\text{shrK } A) \{ \text{Nonce } NA, \text{Agent } B, \text{Key } K, X \} \in \text{parts}(\text{spies } \text{evs});$   
 $\text{Key } K \notin \text{analz } (\text{spies } \text{evs});$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{ns\_shared} ]]$   
 $\implies \text{Says } A \ B \ (\text{Crypt } K \{ \text{Nonce } NB, \text{Nonce } NB \}) \in \text{set } \text{evs}"$   
 $\langle \text{proof} \rangle$

**lemma** *A\_Issues\_B*:

"[[  $\text{Says } A \ B \ (\text{Crypt } K \{ \text{Nonce } NB, \text{Nonce } NB \}) \in \text{set } \text{evs};$   
 $\text{Key } K \notin \text{analz } (\text{spies } \text{evs});$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{ns\_shared} ]]$   
 $\implies A \text{ Issues } B \text{ with } (\text{Crypt } K \{ \text{Nonce } NB, \text{Nonce } NB \}) \text{ on } \text{evs}"$   
 $\langle \text{proof} \rangle$

Tells B that A was alive after B issued NB.

Precisely, the theorem guarantees (to B) key distribution of the session key to A. It also guarantees (to B) non-injective agreement of A with B on the session key. Both goals are available to B in the sense of Goal Availability.

**lemma** *B\_authenticates\_and\_keydist\_to\_A*:

"[[  $\text{Crypt } K \{ \text{Nonce } NB, \text{Nonce } NB \} \in \text{parts } (\text{spies } \text{evs});$   
 $\text{Crypt } (\text{shrK } B) \{ \text{Key } K, \text{Agent } A \} \in \text{parts } (\text{spies } \text{evs});$   
 $\text{Key } K \notin \text{analz } (\text{spies } \text{evs});$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{ns\_shared} ]]$   
 $\implies A \text{ Issues } B \text{ with } (\text{Crypt } K \{ \text{Nonce } NB, \text{Nonce } NB \}) \text{ on } \text{evs}"$   
 $\langle \text{proof} \rangle$

**end**

## 4 The Kerberos Protocol, BAN Version

**theory** *Kerberos\_BAN* **imports** *Public* **begin**

From page 251 of Burrows, Abadi and Needham (1989). A Logic of Authentication. Proc. Royal Soc. 426

Confidentiality (secrecy) and authentication properties are also given in a temporal version: strong guarantees in a little abstracted - but very realistic - model.

**consts**

*sesKlife*    :: nat

*authlife* :: nat

The ticket should remain fresh for two journeys on the network at least

**specification** (*sesKlife*)

*sesKlife\_LB* [iff]: " $2 \leq \text{sesKlife}$ "  
 $\langle \text{proof} \rangle$

The authenticator only for one journey

**specification** (authlife)

```
authlife_LB [iff]:    "authlife  $\neq$  0"
  <proof>
```

**abbreviation**

```
CT :: "event list  $\Rightarrow$  nat" where
  "CT == length "
```

**abbreviation**

```
expiredK :: "[nat, event list]  $\Rightarrow$  bool" where
  "expiredK T evs == sesKlife + T < CT evs"
```

**abbreviation**

```
expiredA :: "[nat, event list]  $\Rightarrow$  bool" where
  "expiredA T evs == authlife + T < CT evs"
```

**definition**

```
Issues :: "[agent, agent, msg, event list]  $\Rightarrow$  bool"
  (<_ Issues _ with _ on _>) where
  "A Issues B with X on evs =
    ( $\exists$  Y. Says A B Y  $\in$  set evs  $\wedge$  X  $\in$  parts {Y}  $\wedge$ 
      X  $\notin$  parts (spies (takeWhile ( $\lambda$ z. z  $\neq$  Says A B Y) (rev evs))))"
```

**definition**

```
before :: "[event, event list]  $\Rightarrow$  event list" (<before _ on _>)
  where "before ev on evs = takeWhile ( $\lambda$ z. z  $\neq$  ev) (rev evs)"
```

**definition**

```
Unique :: "[event, event list]  $\Rightarrow$  bool" (<Unique _ on _>)
  where "Unique ev on evs = (ev  $\notin$  set (tl (dropWhile ( $\lambda$ z. z  $\neq$  ev) evs)))"
```

**inductive\_set** bankerberos :: "event list set"

where

```
Nil:    "[ ]  $\in$  bankerberos"

/ Fake: "[ evsf  $\in$  bankerberos; X  $\in$  synth (analz (spies evsf)) ]
   $\Rightarrow$  Says Spy B X # evsf  $\in$  bankerberos"

/ BK1:  "[ evs1  $\in$  bankerberos ]
   $\Rightarrow$  Says A Server {Agent A, Agent B} # evs1
     $\in$  bankerberos"

/ BK2:  "[ evs2  $\in$  bankerberos; Key K  $\notin$  used evs2; K  $\in$  symKeys;
  Says A' Server {Agent A, Agent B}  $\in$  set evs2 ]
   $\Rightarrow$  Says Server A
```

```

      (Crypt (shrK A)
        {Number (CT evs2), Agent B, Key K,
         (Crypt (shrK B) {Number (CT evs2), Agent A, Key K})})
      # evs2 ∈ bankerberos"

/ BK3: "[ evs3 ∈ bankerberos;
  Says S A (Crypt (shrK A) {Number Tk, Agent B, Key K, Ticket})
  ∈ set evs3;
  Says A Server {Agent A, Agent B} ∈ set evs3;
  ¬ expiredK Tk evs3 ]
⇒ Says A B {Ticket, Crypt K {Agent A, Number (CT evs3)}}
  # evs3 ∈ bankerberos"

/ BK4: "[ evs4 ∈ bankerberos;
  Says A' B {(Crypt (shrK B) {Number Tk, Agent A, Key K}),
             (Crypt K {Agent A, Number Ta})} ∈ set evs4;
  ¬ expiredK Tk evs4; ¬ expiredA Ta evs4 ]
⇒ Says B A (Crypt K (Number Ta)) # evs4
  ∈ bankerberos"

/ Oops: "[ evso ∈ bankerberos;
  Says Server A (Crypt (shrK A) {Number Tk, Agent B, Key K, Ticket})
  ∈ set evso;
  expiredK Tk evso ]
⇒ Notes Spy {Number Tk, Key K} # evso ∈ bankerberos"

declare Says_imp_knows_Spy [THEN parts.Inj, dest]
declare parts.Body [dest]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]

A "possibility property": there are traces that reach the end.

lemma "[Key K ∉ used []; K ∈ symKeys]
  ⇒ ∃ Timestamp. ∃ evs ∈ bankerberos.
    Says B A (Crypt K (Number Timestamp))
    ∈ set evs"
<proof>

```

#### 4.1 Lemmas for reasoning about predicate "Issues"

```

lemma spies_Says_rev: "spies (evs @ [Says A B X]) = insert X (spies evs)"
<proof>

lemma spies_Gets_rev: "spies (evs @ [Gets A X]) = spies evs"
<proof>

lemma spies_Notes_rev: "spies (evs @ [Notes A X]) =
  (if A ∈ bad then insert X (spies evs) else spies evs)"
<proof>

```

**lemma** spies\_evs\_rev: "spies evs = spies (rev evs)"  
 <proof>

**lemmas** parts\_spies\_evs\_revD2 = spies\_evs\_rev [THEN equalityD2, THEN parts\_mono]

**lemma** spies\_takeWhile: "spies (takeWhile P evs)  $\subseteq$  spies evs"  
 <proof>

**lemmas** parts\_spies\_takeWhile\_mono = spies\_takeWhile [THEN parts\_mono]

Lemmas for reasoning about predicate "before"

**lemma** used\_Says\_rev: "used (evs @ [Says A B X]) = parts {X}  $\cup$  (used evs)"  
 <proof>

**lemma** used\_Notes\_rev: "used (evs @ [Notes A X]) = parts {X}  $\cup$  (used evs)"  
 <proof>

**lemma** used\_Gets\_rev: "used (evs @ [Gets B X]) = used evs"  
 <proof>

**lemma** used\_evs\_rev: "used evs = used (rev evs)"  
 <proof>

**lemma** used\_takeWhile\_used [rule\_format]:  
 "x  $\in$  used (takeWhile P X)  $\longrightarrow$  x  $\in$  used X"  
 <proof>

**lemma** set\_evs\_rev: "set evs = set (rev evs)"  
 <proof>

**lemma** takeWhile\_void [rule\_format]:  
 "x  $\notin$  set evs  $\longrightarrow$  takeWhile ( $\lambda z. z \neq x$ ) evs = evs"  
 <proof>

Forwarding Lemma for reasoning about the encrypted portion of message BK3

**lemma** BK3\_msg\_in\_parts\_spies:  
 "Says S A (Crypt KA {Timestamp, B, K, X})  $\in$  set evs  
 $\implies$  X  $\in$  parts (spies evs)"  
 <proof>

**lemma** Ops\_parts\_spies:  
 "Says Server A (Crypt (shrK A) {Timestamp, B, K, X})  $\in$  set evs  
 $\implies$  K  $\in$  parts (spies evs)"  
 <proof>

Spy never sees another agent's shared key! (unless it's bad at start)

**lemma** Spy\_see\_shrK [simp]:  
 "evs  $\in$  bankerberos  $\implies$  (Key (shrK A)  $\in$  parts (spies evs)) = (A  $\in$  bad)"  
 <proof>

**lemma** Spy\_analz\_shrK [simp]:  
 "evs  $\in$  bankerberos  $\implies$  (Key (shrK A)  $\in$  analz (spies evs)) = (A  $\in$  bad)"

$\langle proof \rangle$

**lemma** *Spy\_see\_shrK\_D* [dest!]:  
 "[[ Key (shrK A)  $\in$  parts (spies evs);  
       evs  $\in$  bankerberos ]]  $\implies$  A $\in$ bad"  
 $\langle proof \rangle$

**lemmas** *Spy\_analz\_shrK\_D* = analz\_subset\_parts [THEN subsetD, THEN *Spy\_see\_shrK\_D*, dest!]

Nobody can have used non-existent keys!

**lemma** *new\_keys\_not\_used* [simp]:  
 "[[Key K  $\notin$  used evs; K  $\in$  symKeys; evs  $\in$  bankerberos]]  
 $\implies$  K  $\notin$  keysFor (parts (spies evs))"  
 $\langle proof \rangle$

## 4.2 Lemmas concerning the form of items passed in messages

Describes the form of K, X and K' when the Server sends this message.

**lemma** *Says\_Server\_message\_form*:  
 "[[ Says Server A (Crypt K' {Number Tk, Agent B, Key K, Ticket})  
        $\in$  set evs; evs  $\in$  bankerberos ]]  
 $\implies$  K' = shrK A  $\wedge$  K  $\notin$  range shrK  $\wedge$   
       Ticket = (Crypt (shrK B) {Number Tk, Agent A, Key K})  $\wedge$   
       Key K  $\notin$  used(before  
           Says Server A (Crypt K' {Number Tk, Agent B, Key K, Ticket})  
           on evs)  $\wedge$   
       Tk = CT(before  
           Says Server A (Crypt K' {Number Tk, Agent B, Key K, Ticket})  
           on evs)"  
 $\langle proof \rangle$

If the encrypted message appears then it originated with the Server PROVIDED that A is NOT compromised! This allows A to verify freshness of the session key.

**lemma** *Kab\_authentic*:  
 "[[ Crypt (shrK A) {Number Tk, Agent B, Key K, X}  
        $\in$  parts (spies evs);  
       A  $\notin$  bad; evs  $\in$  bankerberos ]]  
 $\implies$  Says Server A (Crypt (shrK A) {Number Tk, Agent B, Key K, X})  
        $\in$  set evs"  
 $\langle proof \rangle$

If the TICKET appears then it originated with the Server

FRESHNESS OF THE SESSION KEY to B

**lemma** *ticket\_authentic*:  
 "[[ Crypt (shrK B) {Number Tk, Agent A, Key K}  $\in$  parts (spies evs);  
       B  $\notin$  bad; evs  $\in$  bankerberos ]]  
 $\implies$  Says Server A  
       (Crypt (shrK A) {Number Tk, Agent B, Key K,

### 4.3 Non-temporal guarantees, explicitly relying on non-occurrence of oops events - refined below by temporal guarantees

$\text{Crypt } (\text{shrK } B) \{ \text{Number } Tk, \text{ Agent } A, \text{ Key } K \}$

$\in \text{ set evs}$

*<proof>*

EITHER describes the form of X when the following message is sent, OR reduces it to the Fake case. Use *Says\_Server\_message\_form* if applicable.

**lemma** *Says\_S\_message\_form*:

$"[ \text{ Says } S \ A \ (\text{Crypt } (\text{shrK } A) \{ \text{Number } Tk, \text{ Agent } B, \text{ Key } K, X \})$   
 $\quad \in \text{ set evs};$   
 $\quad \text{ evs} \in \text{ bankerberos } ]$

$\implies (K \notin \text{ range shrK} \wedge X = (\text{Crypt } (\text{shrK } B) \{ \text{Number } Tk, \text{ Agent } A, \text{ Key } K \}))$   
 $\quad | X \in \text{ analz } (\text{spies evs})"$

*<proof>*

Session keys are not used to encrypt other session keys

**lemma** *analz\_image\_freshK [rule\_format (no\_asm)]*:

$"\text{ evs} \in \text{ bankerberos} \implies$   
 $\forall K \ KK. \ KK \subseteq - (\text{range shrK}) \longrightarrow$   
 $\quad (\text{Key } K \in \text{ analz } (\text{Key}'KK \cup (\text{spies evs}))) =$   
 $\quad (K \in KK \mid \text{Key } K \in \text{ analz } (\text{spies evs}))"$

*<proof>*

**lemma** *analz\_insert\_freshK*:

$"[ \text{ evs} \in \text{ bankerberos}; \ KAB \notin \text{ range shrK } ] \implies$   
 $\quad (\text{Key } K \in \text{ analz } (\text{insert } (\text{Key } KAB) (\text{spies evs}))) =$   
 $\quad (K = KAB \mid \text{Key } K \in \text{ analz } (\text{spies evs}))"$

*<proof>*

The session key K uniquely identifies the message

**lemma** *unique\_session\_keys*:

$"[ \text{ Says } \text{Server } A$   
 $\quad (\text{Crypt } (\text{shrK } A) \{ \text{Number } Tk, \text{ Agent } B, \text{ Key } K, X \}) \in \text{ set evs};$   
 $\quad \text{ Says } \text{Server } A'$   
 $\quad (\text{Crypt } (\text{shrK } A') \{ \text{Number } Tk', \text{ Agent } B', \text{ Key } K, X' \}) \in \text{ set evs};$   
 $\quad \text{ evs} \in \text{ bankerberos } ] \implies A=A' \wedge Tk= Tk' \wedge B=B' \wedge X = X'"$

*<proof>*

**lemma** *Server\_Unique*:

$"[ \text{ Says } \text{Server } A$   
 $\quad (\text{Crypt } (\text{shrK } A) \{ \text{Number } Tk, \text{ Agent } B, \text{ Key } K, \text{ Ticket} \}) \in \text{ set evs};$   
 $\quad \text{ evs} \in \text{ bankerberos } ] \implies$   
 $\quad \text{ Unique Says Server } A \ (\text{Crypt } (\text{shrK } A) \{ \text{Number } Tk, \text{ Agent } B, \text{ Key } K, \text{ Ticket} \})$   
 $\quad \text{ on evs}"$

*<proof>*

### 4.3 Non-temporal guarantees, explicitly relying on non-occurrence of oops events - refined below by temporal guarantees

Non temporal treatment of confidentiality

Lemma: the session key sent in msg BK2 would be lost by oops if the spy could see it!

```

lemma lemma_conf [rule_format (no_asm)]:
  "[[ A ∉ bad; B ∉ bad; evs ∈ bankerberos ]]
  ⇒ Says Server A
    (Crypt (shrK A) {Number Tk, Agent B, Key K,
                    Crypt (shrK B) {Number Tk, Agent A, Key K}})
    ∈ set evs →
    Key K ∈ analz (spies evs) → Notes Spy {Number Tk, Key K} ∈ set evs"
  <proof>

```

Confidentiality for the Server: Spy does not see the keys sent in msg BK2 as long as they have not expired.

```

lemma Confidentiality_S:
  "[[ Says Server A
    (Crypt K' {Number Tk, Agent B, Key K, Ticket}) ∈ set evs;
    Notes Spy {Number Tk, Key K} ∉ set evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerberos
  ]] ⇒ Key K ∉ analz (spies evs)"
  <proof>

```

Confidentiality for Alice

```

lemma Confidentiality_A:
  "[[ Crypt (shrK A) {Number Tk, Agent B, Key K, X} ∈ parts (spies evs);
    Notes Spy {Number Tk, Key K} ∉ set evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerberos
  ]] ⇒ Key K ∉ analz (spies evs)"
  <proof>

```

Confidentiality for Bob

```

lemma Confidentiality_B:
  "[[ Crypt (shrK B) {Number Tk, Agent A, Key K}
    ∈ parts (spies evs);
    Notes Spy {Number Tk, Key K} ∉ set evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerberos
  ]] ⇒ Key K ∉ analz (spies evs)"
  <proof>

```

Non temporal treatment of authentication

Lemmas *lemma\_A* and *lemma\_B* in fact are common to both temporal and non-temporal treatments

```

lemma lemma_A [rule_format]:
  "[[ A ∉ bad; B ∉ bad; evs ∈ bankerberos ]]
  ⇒
    Key K ∉ analz (spies evs) →
    Says Server A (Crypt (shrK A) {Number Tk, Agent B, Key K, X})
    ∈ set evs →
    Crypt K {Agent A, Number Ta} ∈ parts (spies evs) →
    Says A B {X, Crypt K {Agent A, Number Ta}}
    ∈ set evs"
  <proof>

```



### 4.3 Non-temporal guarantees, explicitly relying on non-occurrence of oops events - refined below by temporal guarantees

```

lemma lemma_B [rule_format]:
  "[ B ∉ bad; evs ∈ bankerberos ]
  ⇒ Key K ∉ analz (spies evs) ⇒
    Says Server A (Crypt (shrK A) {Number Tk, Agent B, Key K, X})
    ∈ set evs ⇒
    Crypt K (Number Ta) ∈ parts (spies evs) ⇒
    Says B A (Crypt K (Number Ta)) ∈ set evs"
⟨proof⟩

```

The "r" suffix indicates theorems where the confidentiality assumptions are relaxed by the corresponding arguments.

Authentication of A to B

```

lemma B_authenticates_A_r:
  "[ Crypt K {Agent A, Number Ta} ∈ parts (spies evs);
    Crypt (shrK B) {Number Tk, Agent A, Key K} ∈ parts (spies evs);
    Notes Spy {Number Tk, Key K} ∉ set evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerberos ]
  ⇒ Says A B {Crypt (shrK B) {Number Tk, Agent A, Key K},
    Crypt K {Agent A, Number Ta}} ∈ set evs"
⟨proof⟩

```

Authentication of B to A

```

lemma A_authenticates_B_r:
  "[ Crypt K (Number Ta) ∈ parts (spies evs);
    Crypt (shrK A) {Number Tk, Agent B, Key K, X} ∈ parts (spies evs);
    Notes Spy {Number Tk, Key K} ∉ set evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerberos ]
  ⇒ Says B A (Crypt K (Number Ta)) ∈ set evs"
⟨proof⟩

```

```

lemma B_authenticates_A:
  "[ Crypt K {Agent A, Number Ta} ∈ parts (spies evs);
    Crypt (shrK B) {Number Tk, Agent A, Key K} ∈ parts (spies evs);
    Key K ∉ analz (spies evs);
    A ∉ bad; B ∉ bad; evs ∈ bankerberos ]
  ⇒ Says A B {Crypt (shrK B) {Number Tk, Agent A, Key K},
    Crypt K {Agent A, Number Ta}} ∈ set evs"
⟨proof⟩

```

```

lemma A_authenticates_B:
  "[ Crypt K (Number Ta) ∈ parts (spies evs);
    Crypt (shrK A) {Number Tk, Agent B, Key K, X} ∈ parts (spies evs);
    Key K ∉ analz (spies evs);
    A ∉ bad; B ∉ bad; evs ∈ bankerberos ]
  ⇒ Says B A (Crypt K (Number Ta)) ∈ set evs"
⟨proof⟩

```

#### 4.4 Temporal guarantees, relying on a temporal check that insures that no oops event occurred. These are available in the sense of goal availability

Temporal treatment of confidentiality

Lemma: the session key sent in msg BK2 would be EXPIRED if the spy could see it!

```

lemma lemma_conf_temporal [rule_format (no_asm)]:
  "[[ A ∉ bad; B ∉ bad; evs ∈ bankerberos ]]
  ⇒ Says Server A
    (Crypt (shrK A) {Number Tk, Agent B, Key K,
                     Crypt (shrK B) {Number Tk, Agent A, Key K}})
    ∈ set evs →
    Key K ∈ analz (spies evs) → expiredK Tk evs"
  <proof>

```

Confidentiality for the Server: Spy does not see the keys sent in msg BK2 as long as they have not expired.

```

lemma Confidentiality_S_temporal:
  "[[ Says Server A
    (Crypt K' {Number T, Agent B, Key K, X}) ∈ set evs;
    ¬ expiredK T evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerberos
  ]] ⇒ Key K ∉ analz (spies evs)"
  <proof>

```

Confidentiality for Alice

```

lemma Confidentiality_A_temporal:
  "[[ Crypt (shrK A) {Number T, Agent B, Key K, X} ∈ parts (spies evs);
    ¬ expiredK T evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerberos
  ]] ⇒ Key K ∉ analz (spies evs)"
  <proof>

```

Confidentiality for Bob

```

lemma Confidentiality_B_temporal:
  "[[ Crypt (shrK B) {Number Tk, Agent A, Key K}
    ∈ parts (spies evs);
    ¬ expiredK Tk evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerberos
  ]] ⇒ Key K ∉ analz (spies evs)"
  <proof>

```

Temporal treatment of authentication

Authentication of A to B

```

lemma B_authenticates_A_temporal:
  "[[ Crypt K {Agent A, Number Ta} ∈ parts (spies evs);
    Crypt (shrK B) {Number Tk, Agent A, Key K}
    ∈ parts (spies evs);
    ¬ expiredK Tk evs;

```

4.5 Treatment of the key distribution goal using trace inspection. All guarantees are in non-temporal form, hence n

$$A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{bankerberos} \quad ]$$

$$\implies \text{Says } A \ B \ \{\!\{ \text{Crypt } (\text{shrK } B) \ \{\!\{ \text{Number } Tk, \text{Agent } A, \text{Key } K \}\!\} \},$$

$$\text{Crypt } K \ \{\!\{ \text{Agent } A, \text{Number } Ta \}\!\} \} \in \text{set evs}''$$

*<proof>*

Authentication of B to A

**lemma** *A\_authenticates\_B\_temporal:*

$$[ \text{Crypt } K \ (\text{Number } Ta) \in \text{parts } (\text{spies evs});$$

$$\text{Crypt } (\text{shrK } A) \ \{\!\{ \text{Number } Tk, \text{Agent } B, \text{Key } K, X \}\!\}$$

$$\in \text{parts } (\text{spies evs});$$

$$\neg \text{expiredK } Tk \ \text{evs};$$

$$A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{bankerberos} \quad ]$$

$$\implies \text{Says } B \ A \ (\text{Crypt } K \ (\text{Number } Ta)) \in \text{set evs}''$$

*<proof>*

**4.5 Treatment of the key distribution goal using trace inspection.** All guarantees are in non-temporal form, hence non available, though their temporal form is trivial to derive. These guarantees also convey a stronger form of authentication - non-injective agreement on the session key

**lemma** *B\_Issues\_A:*

$$[ \text{Says } B \ A \ (\text{Crypt } K \ (\text{Number } Ta)) \in \text{set evs};$$

$$\text{Key } K \notin \text{analz } (\text{spies evs});$$

$$A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{bankerberos} \quad ]$$

$$\implies B \text{ Issues } A \text{ with } (\text{Crypt } K \ (\text{Number } Ta)) \text{ on evs}''$$

*<proof>*

**lemma** *A\_authenticates\_and\_keydist\_to\_B:*

$$[ \text{Crypt } K \ (\text{Number } Ta) \in \text{parts } (\text{spies evs});$$

$$\text{Crypt } (\text{shrK } A) \ \{\!\{ \text{Number } Tk, \text{Agent } B, \text{Key } K, X \}\!\} \in \text{parts } (\text{spies evs});$$

$$\text{Key } K \notin \text{analz } (\text{spies evs});$$

$$A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{bankerberos} \quad ]$$

$$\implies B \text{ Issues } A \text{ with } (\text{Crypt } K \ (\text{Number } Ta)) \text{ on evs}''$$

*<proof>*

**lemma** *A\_Issues\_B:*

$$[ \text{Says } A \ B \ \{\!\{ \text{Ticket}, \text{Crypt } K \ \{\!\{ \text{Agent } A, \text{Number } Ta \}\!\} \}\!\}$$

$$\in \text{set evs};$$

$$\text{Key } K \notin \text{analz } (\text{spies evs});$$

$$A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{bankerberos} \quad ]$$

$$\implies A \text{ Issues } B \text{ with } (\text{Crypt } K \ \{\!\{ \text{Agent } A, \text{Number } Ta \}\!\}) \text{ on evs}''$$

*<proof>*

**lemma** *B\_authenticates\_and\_keydist\_to\_A:*

$$[ \text{Crypt } K \ \{\!\{ \text{Agent } A, \text{Number } Ta \}\!\} \in \text{parts } (\text{spies evs});$$

$$\text{Crypt } (\text{shrK } B) \ \{\!\{ \text{Number } Tk, \text{Agent } A, \text{Key } K \}\!\} \in \text{parts } (\text{spies evs});$$

$$\text{Key } K \notin \text{analz } (\text{spies evs});$$

$$A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{bankerberos} \quad ]$$

$$\implies A \text{ Issues } B \text{ with } (\text{Crypt } K \ \{\!\{ \text{Agent } A, \text{Number } Ta \}\!\}) \text{ on evs}''$$

*<proof>*

end

## 5 The Kerberos Protocol, BAN Version, with Gets event

**theory** *Kerberos\_BAN\_Gets* **imports** *Public* **begin**

From page 251 of Burrows, Abadi and Needham (1989). A Logic of Authentication. Proc. Royal Soc. 426

Confidentiality (secrecy) and authentication properties rely on temporal checks: strong guarantees in a little abstracted - but very realistic - model.

**consts**

*sesKlife* :: nat

*authlife* :: nat

The ticket should remain fresh for two journeys on the network at least

The Gets event causes longer traces for the protocol to reach its end

**specification** (*sesKlife*)

*sesKlife\_LB* [iff]: " $4 \leq \text{sesKlife}$ "

*<proof>*

The authenticator only for one journey

The Gets event causes longer traces for the protocol to reach its end

**specification** (*authlife*)

*authlife\_LB* [iff]: " $2 \leq \text{authlife}$ "

*<proof>*

**abbreviation**

*CT* :: "event list  $\Rightarrow$  nat" **where**

"*CT* == length"

**abbreviation**

*expiredK* :: "[nat, event list]  $\Rightarrow$  bool" **where**

"*expiredK* *T* *evs* == *sesKlife* + *T* < *CT* *evs*"

**abbreviation**

*expiredA* :: "[nat, event list]  $\Rightarrow$  bool" **where**

"*expiredA* *T* *evs* == *authlife* + *T* < *CT* *evs*"

**definition**

```

before :: "[event, event list] ⇒ event list" (<before _ on _>)
where "before ev on evs = takeWhile (λz. z ≠ ev) (rev evs)"

```

**definition**

```

Unique :: "[event, event list] ⇒ bool" (<Unique _ on _>)
where "Unique ev on evs = (ev ∉ set (tl (dropWhile (λz. z ≠ ev) evs)))"

```

**inductive\_set** bankerb\_gets :: "event list set"

**where**

```

Nil:  "[ ] ∈ bankerb_gets"

/ Fake: "[ evsf ∈ bankerb_gets; X ∈ synth (analz (knows Spy evsf)) ]
    ⇒ Says Spy B X # evsf ∈ bankerb_gets"

/ Reception: "[ evsr ∈ bankerb_gets; Says A B X ∈ set evsr ]
    ⇒ Gets B X # evsr ∈ bankerb_gets"

/ BK1: "[ evs1 ∈ bankerb_gets ]
    ⇒ Says A Server {Agent A, Agent B} # evs1
    ∈ bankerb_gets"

/ BK2: "[ evs2 ∈ bankerb_gets; Key K ∉ used evs2; K ∈ symKeys;
    Gets Server {Agent A, Agent B} ∈ set evs2 ]
    ⇒ Says Server A
    (Crypt (shrK A)
     {Number (CT evs2), Agent B, Key K,
      (Crypt (shrK B) {Number (CT evs2), Agent A, Key K})})
    # evs2 ∈ bankerb_gets"

/ BK3: "[ evs3 ∈ bankerb_gets;
    Gets A (Crypt (shrK A) {Number Tk, Agent B, Key K, Ticket})
    ∈ set evs3;
    Says A Server {Agent A, Agent B} ∈ set evs3;
    ¬ expiredK Tk evs3 ]
    ⇒ Says A B {Ticket, Crypt K {Agent A, Number (CT evs3)}}
    # evs3 ∈ bankerb_gets"

/ BK4: "[ evs4 ∈ bankerb_gets;
    Gets B {Crypt (shrK B) {Number Tk, Agent A, Key K}),
    (Crypt K {Agent A, Number Ta}) } ∈ set evs4;
    ¬ expiredK Tk evs4; ¬ expiredA Ta evs4 ]
    ⇒ Says B A (Crypt K (Number Ta)) # evs4
    ∈ bankerb_gets"

/ Ops: "[ evso ∈ bankerb_gets;

```

```

Says Server A (Crypt (shrK A) {Number Tk, Agent B, Key K, Ticket})
  ∈ set evso;
expiredK Tk evso
⇒ Notes Spy {Number Tk, Key K} # evso ∈ bankerb_gets"

```

```

declare Says_imp_knows_Spy [THEN parts.Inj, dest]
declare parts.Body [dest]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]
declare knows_Spy_partsEs [elim]

```

A "possibility property": there are traces that reach the end.

```

lemma "[Key K ∉ used []; K ∈ symKeys]
  ⇒ ∃ Timestamp. ∃ evs ∈ bankerb_gets.
    Says B A (Crypt K (Number Timestamp))
      ∈ set evs"

```

⟨proof⟩

Lemmas about reception invariant: if a message is received it certainly was sent

```

lemma Gets_imp_Says :
  "[Gets B X ∈ set evs; evs ∈ bankerb_gets] ⇒ ∃ A. Says A B X ∈ set
  evs"
⟨proof⟩

```

```

lemma Gets_imp_knows_Spy:
  "[Gets B X ∈ set evs; evs ∈ bankerb_gets] ⇒ X ∈ knows Spy evs"
⟨proof⟩

```

```

lemma Gets_imp_knows_Spy_parts[dest]:
  "[Gets B X ∈ set evs; evs ∈ bankerb_gets] ⇒ X ∈ parts (knows Spy
  evs)"
⟨proof⟩

```

```

lemma Gets_imp_knows:
  "[Gets B X ∈ set evs; evs ∈ bankerb_gets] ⇒ X ∈ knows B evs"
⟨proof⟩

```

```

lemma Gets_imp_knows_analz:
  "[Gets B X ∈ set evs; evs ∈ bankerb_gets] ⇒ X ∈ analz (knows B evs)"
⟨proof⟩

```

Lemmas for reasoning about predicate "before"

```

lemma used_Says_rev: "used (evs @ [Says A B X]) = parts {X} ∪ (used evs)"
⟨proof⟩

```

```

lemma used_Notes_rev: "used (evs @ [Notes A X]) = parts {X} ∪ (used evs)"
⟨proof⟩

```

```

lemma used_Gets_rev: "used (evs @ [Gets B X]) = used evs"
⟨proof⟩

```

```

lemma used_evs_rev: "used evs = used (rev evs)"

```

*<proof>*

**lemma** *used\_takeWhile\_used* [rule\_format]:  
 "x ∈ used (takeWhile P X) → x ∈ used X"  
*<proof>*

**lemma** *set\_evs\_rev*: "set evs = set (rev evs)"  
*<proof>*

**lemma** *takeWhile\_void* [rule\_format]:  
 "x ∉ set evs → takeWhile (λz. z ≠ x) evs = evs"  
*<proof>*

Forwarding Lemma for reasoning about the encrypted portion of message BK3

**lemma** *BK3\_msg\_in\_parts\_knows\_Spy*:  
 "[Gets A (Crypt KA {Timestamp, B, K, X}) ∈ set evs; evs ∈ bankerb\_gets  
 ]  
 ⇒ X ∈ parts (knows Spy evs)"  
*<proof>*

**lemma** *Ops\_parts\_knows\_Spy*:  
 "Says Server A (Crypt (shrK A) {Timestamp, B, K, X}) ∈ set evs  
 ⇒ K ∈ parts (knows Spy evs)"  
*<proof>*

Spy never sees another agent's shared key! (unless it's bad at start)

**lemma** *Spy\_see\_shrK* [simp]:  
 "evs ∈ bankerb\_gets ⇒ (Key (shrK A) ∈ parts (knows Spy evs)) = (A  
 ∈ bad)"  
*<proof>*

**lemma** *Spy\_analz\_shrK* [simp]:  
 "evs ∈ bankerb\_gets ⇒ (Key (shrK A) ∈ analz (knows Spy evs)) = (A  
 ∈ bad)"  
*<proof>*

**lemma** *Spy\_see\_shrK\_D* [dest!]:  
 "[Key (shrK A) ∈ parts (knows Spy evs);  
 evs ∈ bankerb\_gets ] ⇒ A ∈ bad"  
*<proof>*

**lemmas** *Spy\_analz\_shrK\_D = analz\_subset\_parts [THEN subsetD, THEN Spy\_see\_shrK\_D, dest!]*

Nobody can have used non-existent keys!

**lemma** *new\_keys\_not\_used* [simp]:  
 "[Key K ∉ used evs; K ∈ symKeys; evs ∈ bankerb\_gets]  
 ⇒ K ∉ keysFor (parts (knows Spy evs))"  
*<proof>*

## 5.1 Lemmas concerning the form of items passed in messages

Describes the form of  $K$ ,  $X$  and  $K'$  when the Server sends this message.

**lemma** *Says\_Server\_message\_form*:

```
"[[ Says Server A (Crypt K' {Number Tk, Agent B, Key K, Ticket})
  ∈ set evs; evs ∈ bankerb_gets ]]
⇒ K' = shrK A ∧ K ∉ range shrK ∧
  Ticket = (Crypt (shrK B) {Number Tk, Agent A, Key K}) ∧
  Key K ∉ used(before
    Says Server A (Crypt K' {Number Tk, Agent B, Key K, Ticket})
    on evs) ∧
  Tk = CT(before
    Says Server A (Crypt K' {Number Tk, Agent B, Key K, Ticket})
    on evs)"
```

*<proof>*

If the encrypted message appears then it originated with the Server PROVIDED that  $A$  is NOT compromised! This allows  $A$  to verify freshness of the session key.

**lemma** *Kab\_authentic*:

```
"[[ Crypt (shrK A) {Number Tk, Agent B, Key K, X}
  ∈ parts (knows Spy evs);
  A ∉ bad; evs ∈ bankerb_gets ]]
⇒ Says Server A (Crypt (shrK A) {Number Tk, Agent B, Key K, X})
  ∈ set evs"
```

*<proof>*

If the TICKET appears then it originated with the Server

### FRESHNESS OF THE SESSION KEY to B

**lemma** *ticket\_authentic*:

```
"[[ Crypt (shrK B) {Number Tk, Agent A, Key K} ∈ parts (knows Spy evs);
  B ∉ bad; evs ∈ bankerb_gets ]]
⇒ Says Server A
  (Crypt (shrK A) {Number Tk, Agent B, Key K,
    Crypt (shrK B) {Number Tk, Agent A, Key K}})
  ∈ set evs"
```

*<proof>*

EITHER describes the form of  $X$  when the following message is sent, OR reduces it to the Fake case. Use *Says\_Server\_message\_form* if applicable.

**lemma** *Gets\_Server\_message\_form*:

```
"[[ Gets A (Crypt (shrK A) {Number Tk, Agent B, Key K, X})
  ∈ set evs;
  evs ∈ bankerb_gets ]]
⇒ (K ∉ range shrK ∧ X = (Crypt (shrK B) {Number Tk, Agent A, Key K}))
  | X ∈ analz (knows Spy evs)"
```

*<proof>*

Reliability guarantees: honest agents act as we expect

**lemma** *BK3\_imp\_Gets*:



```

"[[ Says A B {|Ticket, Crypt K {|Agent A, Number Ta|}|} ∈ set evs;
  A ∉ bad; evs ∈ bankerb_gets ]
  ⇒ ∃ Tk. Gets A (Crypt (shrK A) {|Number Tk, Agent B, Key K, Ticket|})
    ∈ set evs"
⟨proof⟩

```

**lemma** *BK4\_imp\_Gets*:

```

"[[ Says B A (Crypt K (Number Ta)) ∈ set evs;
  B ∉ bad; evs ∈ bankerb_gets ]
  ⇒ ∃ Tk. Gets B {|Crypt (shrK B) {|Number Tk, Agent A, Key K|},
    Crypt K {|Agent A, Number Ta|}|} ∈ set evs"
⟨proof⟩

```

**lemma** *Gets\_A\_knows\_K*:

```

"[[ Gets A (Crypt (shrK A) {|Number Tk, Agent B, Key K, Ticket|}) ∈ set evs;
  evs ∈ bankerb_gets ]
  ⇒ Key K ∈ analz (knows A evs)"
⟨proof⟩

```

**lemma** *Gets\_B\_knows\_K*:

```

"[[ Gets B {|Crypt (shrK B) {|Number Tk, Agent A, Key K|},
  Crypt K {|Agent A, Number Ta|}|} ∈ set evs;
  evs ∈ bankerb_gets ]
  ⇒ Key K ∈ analz (knows B evs)"
⟨proof⟩

```

Session keys are not used to encrypt other session keys

**lemma** *analz\_image\_freshK* [rule\_format (no\_asm)]:

```

"evs ∈ bankerb_gets ⇒
  ∀ K KK. KK ⊆ - (range shrK) →
    (Key K ∈ analz (Key'KK ∪ (knows Spy evs))) =
    (K ∈ KK | Key K ∈ analz (knows Spy evs))"
⟨proof⟩

```

**lemma** *analz\_insert\_freshK*:

```

"[[ evs ∈ bankerb_gets; KAB ∉ range shrK ] ⇒
  (Key K ∈ analz (insert (Key KAB) (knows Spy evs))) =
  (K = KAB | Key K ∈ analz (knows Spy evs))"
⟨proof⟩

```

The session key K uniquely identifies the message

**lemma** *unique\_session\_keys*:

```

"[[ Says Server A
  (Crypt (shrK A) {|Number Tk, Agent B, Key K, X|}) ∈ set evs;
  Says Server A'
  (Crypt (shrK A') {|Number Tk', Agent B', Key K, X'|}) ∈ set evs;
  evs ∈ bankerb_gets ] ⇒ A=A' ∧ Tk=Tk' ∧ B=B' ∧ X=X'"
⟨proof⟩

```

**lemma** *unique\_session\_keys\_Gets*:

```

"[[ Gets A
  (Crypt (shrK A) {|Number Tk, Agent B, Key K, X|}) ∈ set evs;
  Gets A

```

$(\text{Crypt } (\text{shrK } A) \{ \text{Number } Tk', \text{ Agent } B', \text{ Key } K, X' \}) \in \text{set } \text{evs};$   
 $A \notin \text{bad}; \text{ evs} \in \text{bankerb\_gets} \implies Tk = Tk' \wedge B = B' \wedge X = X'$   
 <proof>

**lemma** *Server\_Unique*:  
 "[ Says Server A  
    $(\text{Crypt } (\text{shrK } A) \{ \text{Number } Tk, \text{ Agent } B, \text{ Key } K, \text{ Ticket} \}) \in \text{set } \text{evs};$   
    $\text{evs} \in \text{bankerb\_gets} \implies$   
   Unique Says Server A  $(\text{Crypt } (\text{shrK } A) \{ \text{Number } Tk, \text{ Agent } B, \text{ Key } K, \text{ Ticket} \})$   
   on evs"  
 <proof>

## 5.2 Non-temporal guarantees, explicitly relying on non-occurrence of oops events - refined below by temporal guarantees

Non temporal treatment of confidentiality

Lemma: the session key sent in msg BK2 would be lost by oops if the spy could see it!

**lemma** *lemma\_conf [rule\_format (no\_asm)]*:  
 "[  $A \notin \text{bad}; B \notin \text{bad}; \text{ evs} \in \text{bankerb\_gets} \implies$   
   Says Server A  
      $(\text{Crypt } (\text{shrK } A) \{ \text{Number } Tk, \text{ Agent } B, \text{ Key } K,$   
        $\text{Crypt } (\text{shrK } B) \{ \text{Number } Tk, \text{ Agent } A, \text{ Key } K \} \})$   
    $\in \text{set } \text{evs} \longrightarrow$   
    $\text{Key } K \in \text{analz } (\text{knows Spy evs}) \longrightarrow \text{Notes Spy } \{ \text{Number } Tk, \text{ Key } K \} \in \text{set}$   
    $\text{evs}$ "  
 <proof>

Confidentiality for the Server: Spy does not see the keys sent in msg BK2 as long as they have not expired.

**lemma** *Confidentiality\_S*:  
 "[ Says Server A  
    $(\text{Crypt } K' \{ \text{Number } Tk, \text{ Agent } B, \text{ Key } K, \text{ Ticket} \}) \in \text{set } \text{evs};$   
    $\text{Notes Spy } \{ \text{Number } Tk, \text{ Key } K \} \notin \text{set } \text{evs};$   
    $A \notin \text{bad}; B \notin \text{bad}; \text{ evs} \in \text{bankerb\_gets}$   
    $\implies \text{Key } K \notin \text{analz } (\text{knows Spy evs})$ "  
 <proof>

Confidentiality for Alice

**lemma** *Confidentiality\_A*:  
 "[  $\text{Crypt } (\text{shrK } A) \{ \text{Number } Tk, \text{ Agent } B, \text{ Key } K, X \} \in \text{parts } (\text{knows Spy evs});$   
    $\text{Notes Spy } \{ \text{Number } Tk, \text{ Key } K \} \notin \text{set } \text{evs};$   
    $A \notin \text{bad}; B \notin \text{bad}; \text{ evs} \in \text{bankerb\_gets}$   
    $\implies \text{Key } K \notin \text{analz } (\text{knows Spy evs})$ "  
 <proof>

Confidentiality for Bob

**lemma** *Confidentiality\_B*:  
 "[  $\text{Crypt } (\text{shrK } B) \{ \text{Number } Tk, \text{ Agent } A, \text{ Key } K \}$

## 5.2 Non-temporal guarantees, explicitly relying on non-occurrence of oops events - refined below by temporal guarantees

```

    ∈ parts (knows Spy evs);
    Notes Spy {Number Tk, Key K} ∉ set evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerb_gets
  ] ⇒ Key K ∉ analz (knows Spy evs)"
<proof>

```

Non temporal treatment of authentication

Lemmas `lemma_A` and `lemma_B` in fact are common to both temporal and non-temporal treatments

```

lemma lemma_A [rule_format]:
  "[ A ∉ bad; B ∉ bad; evs ∈ bankerb_gets ]
  ⇒
    Key K ∉ analz (knows Spy evs) →
    Says Server A (Crypt (shrK A) {Number Tk, Agent B, Key K, X})
    ∈ set evs →
    Crypt K {Agent A, Number Ta} ∈ parts (knows Spy evs) →
    Says A B {X, Crypt K {Agent A, Number Ta}}
    ∈ set evs"

```

<proof>

```

lemma lemma_B [rule_format]:
  "[ B ∉ bad; evs ∈ bankerb_gets ]
  ⇒ Key K ∉ analz (knows Spy evs) →
    Says Server A (Crypt (shrK A) {Number Tk, Agent B, Key K, X})
    ∈ set evs →
    Crypt K (Number Ta) ∈ parts (knows Spy evs) →
    Says B A (Crypt K (Number Ta)) ∈ set evs"

```

<proof>

The "r" suffix indicates theorems where the confidentiality assumptions are relaxed by the corresponding arguments.

Authentication of A to B

```

lemma B_authenticates_A_r:
  "[ Crypt K {Agent A, Number Ta} ∈ parts (knows Spy evs);
    Crypt (shrK B) {Number Tk, Agent A, Key K} ∈ parts (knows Spy evs);
    Notes Spy {Number Tk, Key K} ∉ set evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerb_gets ]
  ⇒ Says A B {Crypt (shrK B) {Number Tk, Agent A, Key K},
    Crypt K {Agent A, Number Ta}} ∈ set evs"

```

<proof>

Authentication of B to A

```

lemma A_authenticates_B_r:
  "[ Crypt K (Number Ta) ∈ parts (knows Spy evs);
    Crypt (shrK A) {Number Tk, Agent B, Key K, X} ∈ parts (knows Spy evs);
    Notes Spy {Number Tk, Key K} ∉ set evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerb_gets ]
  ⇒ Says B A (Crypt K (Number Ta)) ∈ set evs"

```

<proof>

```

lemma B_authenticates_A:
  "[ Crypt K {Agent A, Number Ta} ∈ parts (spies evs);

```

```

    Crypt (shrK B) {Number Tk, Agent A, Key K} ∈ parts (spies evs);
    Key K ∉ analz (spies evs);
    A ∉ bad; B ∉ bad; evs ∈ bankerb_gets ]
  ⇒ Says A B {Crypt (shrK B) {Number Tk, Agent A, Key K},
               Crypt K {Agent A, Number Ta}} ∈ set evs"
<proof>

```

```

lemma A_authenticates_B:
  "[ Crypt K (Number Ta) ∈ parts (spies evs);
    Crypt (shrK A) {Number Tk, Agent B, Key K, X} ∈ parts (spies evs);
    Key K ∉ analz (spies evs);
    A ∉ bad; B ∉ bad; evs ∈ bankerb_gets ]
  ⇒ Says B A (Crypt K (Number Ta)) ∈ set evs"
<proof>

```

### 5.3 Temporal guarantees, relying on a temporal check that insures that no oops event occurred. These are available in the sense of goal availability

Temporal treatment of confidentiality

Lemma: the session key sent in msg BK2 would be EXPIRED if the spy could see it!

```

lemma lemma_conf_temporal [rule_format (no_asm)]:
  "[ A ∉ bad; B ∉ bad; evs ∈ bankerb_gets ]
  ⇒ Says Server A
    (Crypt (shrK A) {Number Tk, Agent B, Key K,
                    Crypt (shrK B) {Number Tk, Agent A, Key K}})
    ∈ set evs →
    Key K ∈ analz (knows Spy evs) → expiredK Tk evs"
<proof>

```

Confidentiality for the Server: Spy does not see the keys sent in msg BK2 as long as they have not expired.

```

lemma Confidentiality_S_temporal:
  "[ Says Server A
    (Crypt K' {Number T, Agent B, Key K, X}) ∈ set evs;
    ¬ expiredK T evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerb_gets
  ] ⇒ Key K ∉ analz (knows Spy evs)"
<proof>

```

Confidentiality for Alice

```

lemma Confidentiality_A_temporal:
  "[ Crypt (shrK A) {Number T, Agent B, Key K, X} ∈ parts (knows Spy evs);
    ¬ expiredK T evs;
    A ∉ bad; B ∉ bad; evs ∈ bankerb_gets
  ] ⇒ Key K ∉ analz (knows Spy evs)"
<proof>

```

Confidentiality for Bob

```

lemma Confidentiality_B_temporal:

```

#### 5.4 Combined guarantees of key distribution and non-injective agreement on the session keys61

```

"[[ Crypt (shrK B) {Number Tk, Agent A, Key K}
   ∈ parts (knows Spy evs);
   ¬ expiredK Tk evs;
   A ∉ bad; B ∉ bad; evs ∈ bankerb_gets
]] ⇒ Key K ∉ analz (knows Spy evs)"
⟨proof⟩

```

Temporal treatment of authentication

Authentication of A to B

```

lemma B_authenticates_A_temporal:
  "[[ Crypt K {Agent A, Number Ta} ∈ parts (knows Spy evs);
     Crypt (shrK B) {Number Tk, Agent A, Key K}
     ∈ parts (knows Spy evs);
     ¬ expiredK Tk evs;
     A ∉ bad; B ∉ bad; evs ∈ bankerb_gets ]]
  ⇒ Says A B {Crypt (shrK B) {Number Tk, Agent A, Key K},
              Crypt K {Agent A, Number Ta}} ∈ set evs"
⟨proof⟩

```

Authentication of B to A

```

lemma A_authenticates_B_temporal:
  "[[ Crypt K (Number Ta) ∈ parts (knows Spy evs);
     Crypt (shrK A) {Number Tk, Agent B, Key K, X}
     ∈ parts (knows Spy evs);
     ¬ expiredK Tk evs;
     A ∉ bad; B ∉ bad; evs ∈ bankerb_gets ]]
  ⇒ Says B A (Crypt K (Number Ta)) ∈ set evs"
⟨proof⟩

```

#### 5.4 Combined guarantees of key distribution and non-injective agreement on the session keys

```

lemma B_authenticates_and_keydist_to_A:
  "[[ Gets B {Crypt (shrK B) {Number Tk, Agent A, Key K},
              Crypt K {Agent A, Number Ta}} ∈ set evs;
     Key K ∉ analz (spies evs);
     A ∉ bad; B ∉ bad; evs ∈ bankerb_gets ]]
  ⇒ Says A B {Crypt (shrK B) {Number Tk, Agent A, Key K},
              Crypt K {Agent A, Number Ta}} ∈ set evs
  ∧ Key K ∈ analz (knows A evs)"
⟨proof⟩

```

```

lemma A_authenticates_and_keydist_to_B:
  "[[ Gets A (Crypt (shrK A) {Number Tk, Agent B, Key K, Ticket}) ∈ set
  evs;
     Gets A (Crypt K (Number Ta)) ∈ set evs;
     Key K ∉ analz (spies evs);
     A ∉ bad; B ∉ bad; evs ∈ bankerb_gets ]]
  ⇒ Says B A (Crypt K (Number Ta)) ∈ set evs
  ∧ Key K ∈ analz (knows B evs)"
⟨proof⟩

```

end

## 6 The Kerberos Protocol, Version IV

theory *KerberosIV* imports *Public* begin

The "u" prefix indicates theorems referring to an updated version of the protocol. The "r" suffix indicates theorems where the confidentiality assumptions are relaxed by the corresponding arguments.

abbreviation

*Kas* :: agent where "Kas == Server"

abbreviation

*Tgs* :: agent where "Tgs == Friend 0"

axiomatization where

*Tgs\_not\_bad* [iff]: "*Tgs*  $\notin$  bad"  
 — *Tgs* is secure — we already know that *Kas* is secure

definition

*authKeys* :: "event list  $\Rightarrow$  key set" where  
 "authKeys evs = {authK.  $\exists$  A Peer Ta. Says Kas A  
     (Crypt (shrK A)  $\{\text{Key authK, Agent Peer, Number Ta,}$   
     (Crypt (shrK Peer)  $\{\text{Agent A, Agent Peer, Key authK, Number}$   
     Ta $\}$ )  
      $\}$ )  $\in$  set evs}"

definition

*Issues* :: "[agent, agent, msg, event list]  $\Rightarrow$  bool"  
 (<\_ Issues \_ with \_ on \_> [50, 0, 0, 50] 50) where  
 "(A Issues B with X on evs) =  
   ( $\exists$  Y. Says A B Y  $\in$  set evs  $\wedge$  X  $\in$  parts {Y}  $\wedge$   
   X  $\notin$  parts (spies (takeWhile ( $\lambda$ z. z  $\neq$  Says A B Y) (rev evs))))"

definition

*before* :: "[event, event list]  $\Rightarrow$  event list" (<before \_ on \_> [0, 50] 50)  
 where "(before ev on evs) = takeWhile ( $\lambda$ z. z  $\neq$  ev) (rev evs)"

definition

*Unique* :: "[event, event list]  $\Rightarrow$  bool" (<Unique \_ on \_> [0, 50] 50)  
 where "(Unique ev on evs) = (ev  $\notin$  set (tl (dropWhile ( $\lambda$ z. z  $\neq$  ev) evs))))"

consts

```

authKlife    :: nat

servKlife    :: nat

authlife     :: nat

replylife    :: nat

specification (authKlife)
  authKlife_LB [iff]: "2 ≤ authKlife"
  ⟨proof⟩

specification (servKlife)
  servKlife_LB [iff]: "2 + authKlife ≤ servKlife"
  ⟨proof⟩

specification (authlife)
  authlife_LB [iff]: "Suc 0 ≤ authlife"
  ⟨proof⟩

specification (replylife)
  replylife_LB [iff]: "Suc 0 ≤ replylife"
  ⟨proof⟩

abbreviation

  CT :: "event list ⇒ nat" where
    "CT == length"

abbreviation
  expiredAK :: "[nat, event list] ⇒ bool" where
    "expiredAK Ta evs == authKlife + Ta < CT evs"

abbreviation
  expiredSK :: "[nat, event list] ⇒ bool" where
    "expiredSK Ts evs == servKlife + Ts < CT evs"

abbreviation
  expiredA :: "[nat, event list] ⇒ bool" where
    "expiredA T evs == authlife + T < CT evs"

abbreviation
  valid :: "[nat, nat] ⇒ bool" (⟨valid _ wrt _⟩ [0, 50] 50) where
    "valid T1 wrt T2 == T1 ≤ replylife + T2"

definition AKcryptSK :: "[key, key, event list] ⇒ bool" where
  "AKcryptSK authK servK evs ==

```

```

     $\exists A B Ts.$ 
    Says Tgs A (Crypt authK
      {Key servK, Agent B, Number Ts,
       Crypt (shrK B) {Agent A, Agent B, Key servK, Number
Ts}}})
       $\in \text{set evs}$ 

inductive_set kerbIV :: "event list set"
  where

    Nil: "[]  $\in$  kerbIV"

    / Fake: "[[ evsf  $\in$  kerbIV; X  $\in$  synth (analz (spies evsf)) ] ]
       $\implies$  Says Spy B X # evsf  $\in$  kerbIV"

    / K1: "[[ evs1  $\in$  kerbIV ] ]
       $\implies$  Says A Kas {Agent A, Agent Tgs, Number (CT evs1)} # evs1
       $\in$  kerbIV"

    / K2: "[[ evs2  $\in$  kerbIV; Key authK  $\notin$  used evs2; authK  $\in$  symKeys;
      Says A' Kas {Agent A, Agent Tgs, Number T1}  $\in$  set evs2 ] ]
       $\implies$  Says Kas A
        (Crypt (shrK A) {Key authK, Agent Tgs, Number (CT evs2),
          (Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK,
            Number (CT evs2)}})) # evs2  $\in$  kerbIV"

    / K3: "[[ evs3  $\in$  kerbIV;
      Says A Kas {Agent A, Agent Tgs, Number T1}  $\in$  set evs3;
      Says Kas' A (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,
        authTicket})  $\in$  set evs3;
      valid Ta wrt T1
      ] ]
       $\implies$  Says A Tgs {authTicket,
        (Crypt authK {Agent A, Number (CT evs3)}),
        Agent B} # evs3  $\in$  kerbIV"

    / K4: "[[ evs4  $\in$  kerbIV; Key servK  $\notin$  used evs4; servK  $\in$  symKeys;
      B  $\neq$  Tgs; authK  $\in$  symKeys;
      Says A' Tgs {

```



```

      (Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK,
                        Number Ta}),
      (Crypt authK {Agent A, Number T2}), Agent B}
    ∈ set evs4;
  ¬ expiredAK Ta evs4;
  ¬ expiredA T2 evs4;
  servKlife + (CT evs4) ≤ authKlife + Ta
]
⇒ Says Tgs A
  (Crypt authK {Key servK, Agent B, Number (CT evs4),
                Crypt (shrK B) {Agent A, Agent B, Key servK,
                                Number (CT evs4)} }
   # evs4 ∈ kerbIV"

/ K5: "[ evs5 ∈ kerbIV; authK ∈ symKeys; servK ∈ symKeys;
  Says A Tgs
    {authTicket, Crypt authK {Agent A, Number T2},
     Agent B}
  ∈ set evs5;
  Says Tgs' A
    (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
  ∈ set evs5;
  valid Ts wrt T2 ]
⇒ Says A B {servTicket,
             Crypt servK {Agent A, Number (CT evs5)} }
  # evs5 ∈ kerbIV"

/ K6: "[ evs6 ∈ kerbIV;
  Says A' B {
    (Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}),
    (Crypt servK {Agent A, Number T3})}
  ∈ set evs6;
  ¬ expiredSK Ts evs6;
  ¬ expiredA T3 evs6
]
⇒ Says B A (Crypt servK (Number T3))
  # evs6 ∈ kerbIV"

/ Oops1: "[ evs01 ∈ kerbIV; A ≠ Spy;
  Says Kas A
    (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,
                    authTicket}) ∈ set evs01;

```

```

    expiredAK Ta evs01 ]]
  ⇒ Says A Spy {Agent A, Agent Tgs, Number Ta, Key authK}
    # evs01 ∈ kerbIV"

```

```

/ Oops2: "[ evs02 ∈ kerbIV; A ≠ Spy;
  Says Tgs A
    (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
    ∈ set evs02;
  expiredSK Ts evs02 ]
  ⇒ Says A Spy {Agent A, Agent B, Number Ts, Key servK}
    # evs02 ∈ kerbIV"

```

```

declare Says_imp_knows_Spy [THEN parts.Inj, dest]
declare parts.Body [dest]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]

```

### 6.1 Lemmas about lists, for reasoning about Issues

```

lemma spies_Says_rev: "spies (evs @ [Says A B X]) = insert X (spies evs)"
<proof>

```

```

lemma spies_Gets_rev: "spies (evs @ [Gets A X]) = spies evs"
<proof>

```

```

lemma spies_Notes_rev: "spies (evs @ [Notes A X]) =
  (if A ∈ bad then insert X (spies evs) else spies evs)"
<proof>

```

```

lemma spies_evs_rev: "spies evs = spies (rev evs)"
<proof>

```

```

lemmas parts_spies_evs_revD2 = spies_evs_rev [THEN equalityD2, THEN parts_mono]

```

```

lemma spies_takeWhile: "spies (takeWhile P evs) ⊆ spies evs"
<proof>

```

```

lemmas parts_spies_takeWhile_mono = spies_takeWhile [THEN parts_mono]

```

### 6.2 Lemmas about authKeys

```

lemma authKeys_empty: "authKeys [] = {}"
<proof>

```

```

lemma authKeys_not_insert:
  "(∀ A Ta akey Peer.
    ev ≠ Says Kas A (Crypt (shrK A) {akey, Agent Peer, Ta,
      (Crypt (shrK Peer) {Agent A, Agent Peer, akey, Ta})}))
  ⇒ authKeys (ev # evs) = authKeys evs"

```

$\langle proof \rangle$

**lemma** *authKeys\_insert*:

```
"authKeys
  (Says Kas A (Crypt (shrK A) {Key K, Agent Peer, Number Ta,
    (Crypt (shrK Peer) {Agent A, Agent Peer, Key K, Number Ta})}) # evs)
  = insert K (authKeys evs)"
 $\langle proof \rangle$ 
```

**lemma** *authKeys\_simp*:

```
"K ∈ authKeys
  (Says Kas A (Crypt (shrK A) {Key K', Agent Peer, Number Ta,
    (Crypt (shrK Peer) {Agent A, Agent Peer, Key K', Number Ta})}) # evs)
  ⇒ K = K' | K ∈ authKeys evs"
 $\langle proof \rangle$ 
```

**lemma** *authKeysI*:

```
"Says Kas A (Crypt (shrK A) {Key K, Agent Tgs, Number Ta,
  (Crypt (shrK Tgs) {Agent A, Agent Tgs, Key K, Number Ta})}) ∈ set evs
  ⇒ K ∈ authKeys evs"
 $\langle proof \rangle$ 
```

**lemma** *authKeys\_used*: "K ∈ authKeys evs ⇒ Key K ∈ used evs"

$\langle proof \rangle$

### 6.3 Forwarding Lemmas

–For reasoning about the encrypted portion of message K3–

**lemma** *K3\_msg\_in\_parts\_spies*:

```
"Says Kas' A (Crypt KeyA {authK, Peer, Ta, authTicket})
  ∈ set evs ⇒ authTicket ∈ parts (spies evs)"
 $\langle proof \rangle$ 
```

**lemma** *Oops\_range\_spies1*:

```
"[ Says Kas A (Crypt KeyA {Key authK, Peer, Ta, authTicket})
  ∈ set evs ;
  evs ∈ kerbIV ] ⇒ authK ∉ range shrK ∧ authK ∈ symKeys"
 $\langle proof \rangle$ 
```

–For reasoning about the encrypted portion of message K5–

**lemma** *K5\_msg\_in\_parts\_spies*:

```
"Says Tgs' A (Crypt authK {servK, Agent B, Ts, servTicket})
  ∈ set evs ⇒ servTicket ∈ parts (spies evs)"
 $\langle proof \rangle$ 
```

**lemma** *Oops\_range\_spies2*:

```
"[ Says Tgs A (Crypt authK {Key servK, Agent B, Ts, servTicket})
  ∈ set evs ;
  evs ∈ kerbIV ] ⇒ servK ∉ range shrK ∧ servK ∈ symKeys"
 $\langle proof \rangle$ 
```

**lemma** *Says\_ticket\_parts*:

```
"Says S A (Crypt K {SesKey, B, TimeStamp, Ticket}) ∈ set evs"
```

$\implies \text{Ticket} \in \text{parts } (\text{spies } \text{evs})"$   
 $\langle \text{proof} \rangle$

**lemma** *Spy\_see\_shrK [simp]:*  
 $"\text{evs} \in \text{kerbIV} \implies (\text{Key } (\text{shrK } A) \in \text{parts } (\text{spies } \text{evs})) = (A \in \text{bad})"$   
 $\langle \text{proof} \rangle$

**lemma** *Spy\_analz\_shrK [simp]:*  
 $"\text{evs} \in \text{kerbIV} \implies (\text{Key } (\text{shrK } A) \in \text{analz } (\text{spies } \text{evs})) = (A \in \text{bad})"$   
 $\langle \text{proof} \rangle$

**lemma** *Spy\_see\_shrK\_D [dest!]:*  
 $"\llbracket \text{Key } (\text{shrK } A) \in \text{parts } (\text{spies } \text{evs}); \text{ evs} \in \text{kerbIV} \rrbracket \implies A \in \text{bad}"$   
 $\langle \text{proof} \rangle$

**lemmas** *Spy\_analz\_shrK\_D = analz\_subset\_parts [THEN subsetD, THEN Spy\_see\_shrK\_D, dest!]*

Nobody can have used non-existent keys!

**lemma** *new\_keys\_not\_used [simp]:*  
 $"\llbracket \text{Key } K \notin \text{used } \text{evs}; K \in \text{symKeys}; \text{ evs} \in \text{kerbIV} \rrbracket$   
 $\implies K \notin \text{keysFor } (\text{parts } (\text{spies } \text{evs}))"$   
 $\langle \text{proof} \rangle$

**lemma** *new\_keys\_not\_analz:*  
 $"\llbracket \text{evs} \in \text{kerbIV}; K \in \text{symKeys}; \text{Key } K \notin \text{used } \text{evs} \rrbracket$   
 $\implies K \notin \text{keysFor } (\text{analz } (\text{spies } \text{evs}))"$   
 $\langle \text{proof} \rangle$

## 6.4 Lemmas for reasoning about predicate "before"

**lemma** *used\_Says\_rev:*  $"\text{used } (\text{evs} @ [\text{Says } A \ B \ X]) = \text{parts } \{X\} \cup (\text{used } \text{evs})"$   
 $\langle \text{proof} \rangle$

**lemma** *used\_Notes\_rev:*  $"\text{used } (\text{evs} @ [\text{Notes } A \ X]) = \text{parts } \{X\} \cup (\text{used } \text{evs})"$   
 $\langle \text{proof} \rangle$

**lemma** *used\_Gets\_rev:*  $"\text{used } (\text{evs} @ [\text{Gets } B \ X]) = \text{used } \text{evs}"$   
 $\langle \text{proof} \rangle$

**lemma** *used\_evs\_rev:*  $"\text{used } \text{evs} = \text{used } (\text{rev } \text{evs})"$   
 $\langle \text{proof} \rangle$

**lemma** *used\_takeWhile\_used [rule\_format]:*  
 $"x \in \text{used } (\text{takeWhile } P \ X) \longrightarrow x \in \text{used } X"$   
 $\langle \text{proof} \rangle$

**lemma** *set\_evs\_rev:*  $"\text{set } \text{evs} = \text{set } (\text{rev } \text{evs})"$   
 $\langle \text{proof} \rangle$

**lemma** *takeWhile\_void [rule\_format]:*  
 $"x \notin \text{set } \text{evs} \longrightarrow \text{takeWhile } (\lambda z. z \neq x) \text{ evs} = \text{evs}"$

*<proof>*

## 6.5 Regularity Lemmas

These concern the form of items passed in messages

Describes the form of all components sent by Kas

**lemma** *Says\_Kas\_message\_form*:

```
"[ Says Kas A (Crypt K {Key authK, Agent Peer, Number Ta, authTicket})
  ∈ set evs;
  evs ∈ kerbIV ] ⇒
K = shrK A ∧ Peer = Tgs ∧
authK ∉ range shrK ∧ authK ∈ authKeys evs ∧ authK ∈ symKeys ∧
authTicket = (Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta})
∧
Key authK ∉ used(before
  Says Kas A (Crypt K {Key authK, Agent Peer, Number Ta, authTicket})
  on evs) ∧
Ta = CT (before
  Says Kas A (Crypt K {Key authK, Agent Peer, Number Ta, authTicket})
  on evs)"
<proof>
```

**lemma** *SesKey\_is\_session\_key*:

```
"[ Crypt (shrK Tgs_B) {Agent A, Agent Tgs_B, Key SesKey, Number T}
  ∈ parts (spies evs); Tgs_B ∉ bad;
  evs ∈ kerbIV ]
⇒ SesKey ∉ range shrK"
<proof>
```

**lemma** *authTicket\_authentic*:

```
"[ Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}
  ∈ parts (spies evs);
  evs ∈ kerbIV ]
⇒ Says Kas A (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,
  Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}})
  ∈ set evs"
<proof>
```

**lemma** *authTicket\_crypt\_authK*:

```
"[ Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}
  ∈ parts (spies evs);
  evs ∈ kerbIV ]
⇒ authK ∈ authKeys evs"
<proof>
```

Describes the form of servK, servTicket and authK sent by Tgs

**lemma** *Says\_Tgs\_message\_form*:

```
"[ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
  ∈ set evs;
```

$$\begin{aligned} & \text{evs} \in \text{kerbIV} \ ] \\ \implies & B \neq \text{Tgs} \ \wedge \\ & \text{authK} \notin \text{range shrK} \ \wedge \ \text{authK} \in \text{authKeys evs} \ \wedge \ \text{authK} \in \text{symKeys} \ \wedge \\ & \text{servK} \notin \text{range shrK} \ \wedge \ \text{servK} \notin \text{authKeys evs} \ \wedge \ \text{servK} \in \text{symKeys} \ \wedge \\ & \text{servTicket} = (\text{Crypt} (\text{shrK } B) \ \{\!\{ \text{Agent } A, \text{Agent } B, \text{Key servK, Number Ts} \}\!\}) \\ \wedge & \\ & \text{Key servK} \notin \text{used (before} \\ & \quad \text{Says Tgs A (Crypt authK } \{\!\{ \text{Key servK, Agent B, Number Ts, servTicket} \}\!\}) \\ & \quad \text{on evs)} \ \wedge \\ & \text{Ts} = \text{CT(before} \\ & \quad \text{Says Tgs A (Crypt authK } \{\!\{ \text{Key servK, Agent B, Number Ts, servTicket} \}\!\}) \\ & \quad \text{on evs)} \ " \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma authTicket\_form:**

$$\begin{aligned} & "[ \text{Crypt (shrK A) } \{\!\{ \text{Key authK, Agent Tgs, Ta, authTicket} \}\!\} \\ & \quad \in \text{parts (spies evs)}; \\ & \quad A \notin \text{bad}; \\ & \quad \text{evs} \in \text{kerbIV} \ ] \\ \implies & \text{authK} \notin \text{range shrK} \ \wedge \ \text{authK} \in \text{symKeys} \ \wedge \\ & \text{authTicket} = \text{Crypt (shrK Tgs) } \{\!\{ \text{Agent A, Agent Tgs, Key authK, Ta} \}\!\} " \\ \langle \text{proof} \rangle & \end{aligned}$$

This form holds also over an authTicket, but is not needed below.

**lemma servTicket\_form:**

$$\begin{aligned} & "[ \text{Crypt authK } \{\!\{ \text{Key servK, Agent B, Ts, servTicket} \}\!\} \\ & \quad \in \text{parts (spies evs)}; \\ & \quad \text{Key authK} \notin \text{analz (spies evs)}; \\ & \quad \text{evs} \in \text{kerbIV} \ ] \\ \implies & \text{servK} \notin \text{range shrK} \ \wedge \ \text{servK} \in \text{symKeys} \ \wedge \\ & (\exists A. \text{servTicket} = \text{Crypt (shrK B) } \{\!\{ \text{Agent A, Agent B, Key servK, Ts} \}\!\}) " \\ \langle \text{proof} \rangle & \end{aligned}$$

Essentially the same as authTicket\_form

**lemma Says\_kas\_message\_form:**

$$\begin{aligned} & "[ \text{Says Kas' A (Crypt (shrK A) } \\ & \quad \{\!\{ \text{Key authK, Agent Tgs, Ta, authTicket} \}\!\}) \in \text{set evs}; \\ & \quad \text{evs} \in \text{kerbIV} \ ] \\ \implies & \text{authK} \notin \text{range shrK} \ \wedge \ \text{authK} \in \text{symKeys} \ \wedge \\ & \text{authTicket} = \\ & \quad \text{Crypt (shrK Tgs) } \{\!\{ \text{Agent A, Agent Tgs, Key authK, Ta} \}\!\} \\ & \quad \mid \text{authTicket} \in \text{analz (spies evs)} " \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma Says\_tgs\_message\_form:**

$$\begin{aligned} & "[ \text{Says Tgs' A (Crypt authK } \{\!\{ \text{Key servK, Agent B, Ts, servTicket} \}\!\}) \\ & \quad \in \text{set evs}; \ \text{authK} \in \text{symKeys}; \\ & \quad \text{evs} \in \text{kerbIV} \ ] \\ \implies & \text{servK} \notin \text{range shrK} \ \wedge \\ & (\exists A. \text{servTicket} = \\ & \quad \text{Crypt (shrK B) } \{\!\{ \text{Agent A, Agent B, Key servK, Ts} \}\!\}) \\ & \quad \mid \text{servTicket} \in \text{analz (spies evs)} " \\ \langle \text{proof} \rangle & \end{aligned}$$

## 6.6 Authenticity theorems: confirm origin of sensitive messages

**lemma** *authK\_authentic*:

```
"[ Crypt (shrK A) {Key authK, Peer, Ta, authTicket}
  ∈ parts (spies evs);
  A ∉ bad; evs ∈ kerbIV ]
⇒ Says Kas A (Crypt (shrK A) {Key authK, Peer, Ta, authTicket})
  ∈ set evs"
```

*<proof>*

If a certain encrypted message appears then it originated with Tgs

**lemma** *servK\_authentic*:

```
"[ Crypt authK {Key servK, Agent B, Number Ts, servTicket}
  ∈ parts (spies evs);
  Key authK ∉ analz (spies evs);
  authK ∉ range shrK;
  evs ∈ kerbIV ]
⇒ ∃ A. Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
  ∈ set evs"
```

*<proof>*

**lemma** *servK\_authentic\_bis*:

```
"[ Crypt authK {Key servK, Agent B, Number Ts, servTicket}
  ∈ parts (spies evs);
  Key authK ∉ analz (spies evs);
  B ≠ Tgs;
  evs ∈ kerbIV ]
⇒ ∃ A. Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
  ∈ set evs"
```

*<proof>*

Authenticity of servK for B

**lemma** *servTicket\_authentic\_Tgs*:

```
"[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
  ∈ parts (spies evs); B ≠ Tgs; B ∉ bad;
  evs ∈ kerbIV ]
⇒ ∃ authK.
  Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts,
    Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}})
  ∈ set evs"
```

*<proof>*

Anticipated here from next subsection

**lemma** *K4\_imp\_K2*:

```
"[ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
  ∈ set evs; evs ∈ kerbIV ]
⇒ ∃ Ta. Says Kas A
  (Crypt (shrK A)
    {Key authK, Agent Tgs, Number Ta,
     Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}})
  ∈ set evs"
```

*<proof>*

Anticipated here from next subsection

**lemma** *u\_K4\_imp\_K2*:

"[[ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})  
 $\in$  set evs; evs  $\in$  kerbIV ]]  
 $\implies \exists Ta. (Says Kas A (Crypt (shrK A) \{Key authK, Agent Tgs, Number Ta,$   
 $Crypt (shrK Tgs) \{Agent A, Agent Tgs, Key authK, Number Ta\}\})$   
 $\in$  set evs  
 $\wedge servKlife + Ts \leq authKlife + Ta)"$   
 <proof>

**lemma** *servTicket\_authentic\_Kas*:

"[[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}  
 $\in$  parts (spies evs); B  $\neq$  Tgs; B  $\notin$  bad;  
 evs  $\in$  kerbIV ]]  
 $\implies \exists authK Ta.$   
 Says Kas A  
 $(Crypt (shrK A) \{Key authK, Agent Tgs, Number Ta,$   
 $Crypt (shrK Tgs) \{Agent A, Agent Tgs, Key authK, Number Ta\}\})$   
 $\in$  set evs"  
 <proof>

**lemma** *u\_servTicket\_authentic\_Kas*:

"[[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}  
 $\in$  parts (spies evs); B  $\neq$  Tgs; B  $\notin$  bad;  
 evs  $\in$  kerbIV ]]  
 $\implies \exists authK Ta. Says Kas A (Crypt (shrK A) \{Key authK, Agent Tgs, Number Ta,$   
 $Crypt (shrK Tgs) \{Agent A, Agent Tgs, Key authK, Number Ta\}\})$   
 $\in$  set evs  
 $\wedge servKlife + Ts \leq authKlife + Ta"$   
 <proof>

**lemma** *servTicket\_authentic*:

"[[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}  
 $\in$  parts (spies evs); B  $\neq$  Tgs; B  $\notin$  bad;  
 evs  $\in$  kerbIV ]]  
 $\implies \exists Ta authK.$   
 Says Kas A (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,  
 $Crypt (shrK Tgs) \{Agent A, Agent Tgs, Key authK, Number$   
 $Ta\}\})$   
 $\in$  set evs  
 $\wedge Says Tgs A (Crypt authK \{Key servK, Agent B, Number Ts,$   
 $Crypt (shrK B) \{Agent A, Agent B, Key servK, Number Ts\}\})$   
 $\in$  set evs"  
 <proof>

**lemma** *u\_servTicket\_authentic*:

"[[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}  
 $\in$  parts (spies evs); B  $\neq$  Tgs; B  $\notin$  bad;  
 evs  $\in$  kerbIV ]]  
 $\implies \exists Ta authK.$   
 $(Says Kas A (Crypt (shrK A) \{Key authK, Agent Tgs, Number Ta,$   
 $Crypt (shrK Tgs) \{Agent A, Agent Tgs, Key authK, Number$   
 $Ta\}\})$   
 $\in$  set evs



## 6.7 Reliability: friendly agents send something if something else happened 73

$\wedge \text{ Says Tgs A (Crypt authK } \{ \text{Key servK, Agent B, Number Ts,} \\ \text{Crypt (shrK B) } \{ \text{Agent A, Agent B, Key servK, Number Ts} \} \})$   
 $\in \text{ set evs}$   
 $\wedge \text{ servKlife + Ts} \leq \text{authKlife + Ta}$ "  
 <proof>

**lemma** u\_NotexpiredSK\_NotexpiredAK:  
 $"[ \neg \text{ expiredSK Ts evs; servKlife + Ts} \leq \text{authKlife + Ta} ]$   
 $\implies \neg \text{ expiredAK Ta evs}"$   
 <proof>

## 6.7 Reliability: friendly agents send something if something else happened

**lemma** K3\_imp\_K2:  
 $"[ \text{ Says A Tgs} \\ \{ \text{authTicket, Crypt authK } \{ \text{Agent A, Number T2} \}, \text{ Agent B} \} \\ \in \text{ set evs;} \\ \text{ A} \notin \text{ bad; evs} \in \text{ kerbIV} ]$   
 $\implies \exists \text{ Ta. Says Kas A (Crypt (shrK A)} \\ \{ \text{Key authK, Agent Tgs, Number Ta, authTicket} \}) \\ \in \text{ set evs}"$   
 <proof>

Anticipated here from next subsection. An authK is encrypted by one and only one Shared key. A servK is encrypted by one and only one authK.

**lemma** Key\_unique\_SesKey:  
 $"[ \text{ Crypt K } \{ \text{Key SesKey, Agent B, T, Ticket} \} \\ \in \text{ parts (spies evs);} \\ \text{ Crypt K' } \{ \text{Key SesKey, Agent B', T', Ticket'} \} \\ \in \text{ parts (spies evs); Key SesKey} \notin \text{ analz (spies evs);} \\ \text{ evs} \in \text{ kerbIV} ]$   
 $\implies \text{ K=K'} \wedge \text{ B=B'} \wedge \text{ T=T'} \wedge \text{ Ticket=Ticket'}"$   
 <proof>

**lemma** Tgs\_authenticates\_A:  
 $"[ \text{ Crypt authK } \{ \text{Agent A, Number T2} \} \in \text{ parts (spies evs);} \\ \text{ Crypt (shrK Tgs) } \{ \text{Agent A, Agent Tgs, Key authK, Number Ta} \} \\ \in \text{ parts (spies evs);} \\ \text{ Key authK} \notin \text{ analz (spies evs); A} \notin \text{ bad; evs} \in \text{ kerbIV} ]$   
 $\implies \exists \text{ B. Says A Tgs } \{ \\ \text{ Crypt (shrK Tgs) } \{ \text{Agent A, Agent Tgs, Key authK, Number Ta} \}, \\ \text{ Crypt authK } \{ \text{Agent A, Number T2} \}, \text{ Agent B} \} \in \text{ set evs}"$   
 <proof>

**lemma** Says\_K5:  
 $"[ \text{ Crypt servK } \{ \text{Agent A, Number T3} \} \in \text{ parts (spies evs);} \\ \text{ Says Tgs A (Crypt authK } \{ \text{Key servK, Agent B, Number Ts,} \\ \text{servTicket} \}) \in \text{ set evs;} \\ \text{ Key servK} \notin \text{ analz (spies evs);} \\ \text{ A} \notin \text{ bad; B} \notin \text{ bad; evs} \in \text{ kerbIV} ]$   
 $\implies \text{ Says A B } \{ \text{servTicket, Crypt servK } \{ \text{Agent A, Number T3} \} \} \in \text{ set evs}"$   
 <proof>

Anticipated here from next subsection

**lemma** *unique\_CryptKey*:

```
"[[ Crypt (shrK B) {Agent A, Agent B, Key SesKey, T}
   ∈ parts (spies evs);
   Crypt (shrK B') {Agent A', Agent B', Key SesKey, T'}
   ∈ parts (spies evs); Key SesKey ∉ analz (spies evs);
   evs ∈ kerbIV ]]
⇒ A=A' ∧ B=B' ∧ T=T'"
⟨proof⟩
```

**lemma** *Says\_K6*:

```
"[[ Crypt servK (Number T3) ∈ parts (spies evs);
   Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts,
                           servTicket}) ∈ set evs;
   Key servK ∉ analz (spies evs);
   A ∉ bad; B ∉ bad; evs ∈ kerbIV ]]
⇒ Says B A (Crypt servK (Number T3)) ∈ set evs"
⟨proof⟩
```

Needs a unicity theorem, hence moved here

**lemma** *servK\_authentic\_ter*:

```
"[[ Says Kas A
   (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket}) ∈ set evs;
   Crypt authK {Key servK, Agent B, Number Ts, servTicket}
   ∈ parts (spies evs);
   Key authK ∉ analz (spies evs);
   evs ∈ kerbIV ]]
⇒ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
   ∈ set evs"
⟨proof⟩
```

## 6.8 Unicity Theorems

The session key, if secure, uniquely identifies the Ticket whether authTicket or servTicket. As a matter of fact, one can read also Tgs in the place of B.

**lemma** *unique\_authKeys*:

```
"[[ Says Kas A
   (Crypt Ka {Key authK, Agent Tgs, Ta, X}) ∈ set evs;
   Says Kas A'
   (Crypt Ka' {Key authK, Agent Tgs, Ta', X'}) ∈ set evs;
   evs ∈ kerbIV ]] ⇒ A=A' ∧ Ka=Ka' ∧ Ta=Ta' ∧ X=X'"
⟨proof⟩
```

servK uniquely identifies the message from Tgs

**lemma** *unique\_servKeys*:

```
"[[ Says Tgs A
   (Crypt K {Key servK, Agent B, Ts, X}) ∈ set evs;
   Says Tgs A'
   (Crypt K' {Key servK, Agent B', Ts', X'}) ∈ set evs;
   evs ∈ kerbIV ]] ⇒ A=A' ∧ B=B' ∧ K=K' ∧ Ts=Ts' ∧ X=X'"
⟨proof⟩
```

Revised unicity theorems

**lemma** *Kas\_Unique*:  
 "[[ Says Kas A  
     (Crypt Ka {Key authK, Agent Tgs, Ta, authTicket}) ∈ set evs;  
     evs ∈ kerbIV ] ] ⇒  
   Unique (Says Kas A (Crypt Ka {Key authK, Agent Tgs, Ta, authTicket}))  
   on evs"  
 <proof>

**lemma** *Tgs\_Unique*:  
 "[[ Says Tgs A  
     (Crypt authK {Key servK, Agent B, Ts, servTicket}) ∈ set evs;  
     evs ∈ kerbIV ] ] ⇒  
   Unique (Says Tgs A (Crypt authK {Key servK, Agent B, Ts, servTicket}))  
   on evs"  
 <proof>

## 6.9 Lemmas About the Predicate $AKcryptSK$

**lemma** *not\_AKcryptSK\_Nil* [iff]: " $\neg AKcryptSK \text{ authK servK []}$ "  
 <proof>

**lemma** *AKcryptSKI*:  
 "[[ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, X }) ∈ set evs;  
     evs ∈ kerbIV ] ] ⇒ AKcryptSK authK servK evs"  
 <proof>

**lemma** *AKcryptSK\_Says* [simp]:  
 "AKcryptSK authK servK (Says S A X # evs) =  
   (Tgs = S ∧  
     (∃ B Ts. X = Crypt authK  
       {Key servK, Agent B, Number Ts,  
       Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}})  
     | AKcryptSK authK servK evs)"  
 <proof>

**lemma** *Auth\_fresh\_not\_AKcryptSK*:  
 "[[ Key authK ∉ used evs; evs ∈ kerbIV ] ]  
   ⇒  $\neg AKcryptSK \text{ authK servK evs}$ "  
 <proof>

**lemma** *Serv\_fresh\_not\_AKcryptSK*:  
 "Key servK ∉ used evs ⇒  $\neg AKcryptSK \text{ authK servK evs}$ "  
 <proof>

**lemma** *authK\_not\_AKcryptSK*:  
 "[[ Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, tk}  
     ∈ parts (spies evs); evs ∈ kerbIV ] ]  
   ⇒  $\neg AKcryptSK K \text{ authK evs}$ "  
 <proof>

A secure serverkey cannot have been used to encrypt others

**lemma** *servK\_not\_AKcryptSK*:  
 "[ Crypt (shrK B) {Agent A, Agent B, Key SK, Number Ts} ∈ parts (spies evs);  
 Key SK ∉ analz (spies evs); SK ∈ symKeys;  
 B ≠ Tgs; evs ∈ kerbIV ]  
 ⇒ ¬ AKcryptSK SK K evs"  
 <proof>

Long term keys are not issued as servKeys

**lemma** *shrK\_not\_AKcryptSK*:  
 "evs ∈ kerbIV ⇒ ¬ AKcryptSK K (shrK A) evs"  
 <proof>

The Tgs message associates servK with authK and therefore not with any other key authK.

**lemma** *Says\_Tgs\_AKcryptSK*:  
 "[ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, X})  
 ∈ set evs;  
 authK' ≠ authK; evs ∈ kerbIV ]  
 ⇒ ¬ AKcryptSK authK' servK evs"  
 <proof>

Equivalently

**lemma** *not\_different\_AKcryptSK*:  
 "[ AKcryptSK authK servK evs;  
 authK' ≠ authK; evs ∈ kerbIV ]  
 ⇒ ¬ AKcryptSK authK' servK evs ∧ servK ∈ symKeys"  
 <proof>

**lemma** *AKcryptSK\_not\_AKcryptSK*:  
 "[ AKcryptSK authK servK evs; evs ∈ kerbIV ]  
 ⇒ ¬ AKcryptSK servK K evs"  
 <proof>

The only session keys that can be found with the help of session keys are those sent by Tgs in step K4.

We take some pains to express the property as a logical equivalence so that the simplifier can apply it.

**lemma** *Key\_analz\_image\_Key\_lemma*:  
 "P → (Key K ∈ analz (Key'KK ∪ H)) → (K ∈ KK | Key K ∈ analz H)  
 ⇒  
 P → (Key K ∈ analz (Key'KK ∪ H)) = (K ∈ KK | Key K ∈ analz H)"  
 <proof>

**lemma** *AKcryptSK\_analz\_insert*:  
 "[ AKcryptSK K K' evs; K ∈ symKeys; evs ∈ kerbIV ]  
 ⇒ Key K' ∈ analz (insert (Key K) (spies evs))"  
 <proof>

**lemma** *authKeys\_are\_not\_AKcryptSK*:

```

"[[ K ∈ authKeys evs ∪ range shrK; evs ∈ kerbIV ]]
  ⇒ ∀ SK. ¬ AKcryptSK SK K evs ∧ K ∈ symKeys"
⟨proof⟩

```

```

lemma not_authKeys_not_AKcryptSK:
  "[[ K ∉ authKeys evs;
      K ∉ range shrK; evs ∈ kerbIV ]]
    ⇒ ∀ SK. ¬ AKcryptSK K SK evs"
⟨proof⟩

```

## 6.10 Secrecy Theorems

For the Oops2 case of the next theorem

```

lemma Oops2_not_AKcryptSK:
  "[[ evs ∈ kerbIV;
      Says Tgs A (Crypt authK
                    {Key servK, Agent B, Number Ts, servTicket})
                ∈ set evs ]]
    ⇒ ¬ AKcryptSK servK SK evs"
⟨proof⟩

```

Big simplification law for keys SK that are not crypted by keys in KK It helps prove three, otherwise hard, facts about keys. These facts are exploited as simplification laws for analz, and also "limit the damage" in case of loss of a key to the spy. See ESORICS98. [simplified by LCP]

```

lemma Key_analz_image_Key [rule_format (no_asm)]:
  "evs ∈ kerbIV ⇒
    (∀ SK KK. SK ∈ symKeys ∧ KK ⊆ -(range shrK) →
     (∀ K ∈ KK. ¬ AKcryptSK K SK evs) →
     (Key SK ∈ analz (Key'KK ∪ (spies evs))) =
     (SK ∈ KK | Key SK ∈ analz (spies evs)))"
⟨proof⟩

```

First simplification law for analz: no session keys encrypt authentication keys or shared keys.

```

lemma analz_insert_freshK1:
  "[[ evs ∈ kerbIV; K ∈ authKeys evs ∪ range shrK;
      SesKey ∉ range shrK ]]
    ⇒ (Key K ∈ analz (insert (Key SesKey) (spies evs))) =
      (K = SesKey | Key K ∈ analz (spies evs))"
⟨proof⟩

```

Second simplification law for analz: no service keys encrypt any other keys.

```

lemma analz_insert_freshK2:
  "[[ evs ∈ kerbIV; servK ∉ (authKeys evs); servK ∉ range shrK;
      K ∈ symKeys ]]
    ⇒ (Key K ∈ analz (insert (Key servK) (spies evs))) =
      (K = servK | Key K ∈ analz (spies evs))"
⟨proof⟩

```

Third simplification law for analz: only one authentication key encrypts a certain service key.

**lemma** *analz\_insert\_freshK3*:

```
"[[ AKcryptSK authK servK evs;
  authK' ≠ authK; authK' ∉ range shrK; evs ∈ kerbIV ]]
  ⇒ (Key servK ∈ analz (insert (Key authK') (spies evs))) =
    (servK = authK' | Key servK ∈ analz (spies evs))"
⟨proof⟩
```

**lemma** *analz\_insert\_freshK3\_bis*:

```
"[[ Says Tgs A
  (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
  ∈ set evs;
  authK ≠ authK'; authK' ∉ range shrK; evs ∈ kerbIV ]]
  ⇒ (Key servK ∈ analz (insert (Key authK') (spies evs))) =
    (servK = authK' | Key servK ∈ analz (spies evs))"
⟨proof⟩
```

a weakness of the protocol

**lemma** *authK\_compromises\_servK*:

```
"[[ Says Tgs A
  (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
  ∈ set evs; authK ∈ symKeys;
  Key authK ∈ analz (spies evs); evs ∈ kerbIV ]]
  ⇒ Key servK ∈ analz (spies evs)"
⟨proof⟩
```

**lemma** *servK\_notin\_authKeysD*:

```
"[[ Crypt authK {Key servK, Agent B, Ts,
  Crypt (shrK B) {Agent A, Agent B, Key servK, Ts}}
  ∈ parts (spies evs);
  Key servK ∉ analz (spies evs);
  B ≠ Tgs; evs ∈ kerbIV ]]
  ⇒ servK ∉ authKeys evs"
⟨proof⟩
```

If Spy sees the Authentication Key sent in msg K2, then the Key has expired.

**lemma** *Confidentiality\_Kas\_lemma [rule\_format]*:

```
"[[ authK ∈ symKeys; A ∉ bad; evs ∈ kerbIV ]]
  ⇒ Says Kas A
    (Crypt (shrK A)
      {Key authK, Agent Tgs, Number Ta,
       Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}})
  ∈ set evs →
  Key authK ∈ analz (spies evs) →
  expiredAK Ta evs"
⟨proof⟩
```

**lemma** *Confidentiality\_Kas*:

```
"[[ Says Kas A
  (Crypt Ka {Key authK, Agent Tgs, Number Ta, authTicket})
  ∈ set evs;
  ¬ expiredAK Ta evs;
  A ∉ bad; evs ∈ kerbIV ]]
  ⇒ Key authK ∉ analz (spies evs)"
⟨proof⟩
```

If Spy sees the Service Key sent in msg K4, then the Key has expired.

**lemma Confidentiality\_lemma** [rule\_format]:

```
"[[ Says Tgs A
    (Crypt authK
      {Key servK, Agent B, Number Ts,
       Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}}})
  ∈ set evs;
  Key authK ∉ analz (spies evs);
  servK ∈ symKeys;
  A ∉ bad; B ∉ bad; evs ∈ kerbIV ]]
⇒ Key servK ∈ analz (spies evs) →
  expiredSK Ts evs"
```

⟨proof⟩

In the real world Tgs can't check wheter authK is secure!

**lemma Confidentiality\_Tgs:**

```
"[[ Says Tgs A
    (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
  ∈ set evs;
  Key authK ∉ analz (spies evs);
  ¬ expiredSK Ts evs;
  A ∉ bad; B ∉ bad; evs ∈ kerbIV ]]
⇒ Key servK ∉ analz (spies evs)"
```

⟨proof⟩

In the real world Tgs CAN check what Kas sends!

**lemma Confidentiality\_Tgs\_bis:**

```
"[[ Says Kas A
    (Crypt Ka {Key authK, Agent Tgs, Number Ta, authTicket})
  ∈ set evs;
  Says Tgs A
    (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
  ∈ set evs;
  ¬ expiredAK Ta evs; ¬ expiredSK Ts evs;
  A ∉ bad; B ∉ bad; evs ∈ kerbIV ]]
⇒ Key servK ∉ analz (spies evs)"
```

⟨proof⟩

Most general form

**lemmas Confidentiality\_Tgs\_ter** = authTicket\_authentic [THEN Confidentiality\_Tgs\_bis]

**lemmas Confidentiality\_Auth\_A** = authK\_authentic [THEN Confidentiality\_Kas]

Needs a confidentiality guarantee, hence moved here. Authenticity of servK for A

**lemma servK\_authentic\_bis\_r:**

```
"[[ Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket}
  ∈ parts (spies evs);
  Crypt authK {Key servK, Agent B, Number Ts, servTicket}
  ∈ parts (spies evs);
  ¬ expiredAK Ta evs; A ∉ bad; evs ∈ kerbIV ]]
⇒ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})"
```

$\in \text{set evs}$ "  
 $\langle \text{proof} \rangle$

**lemma Confidentiality\_Serv\_A:**

"[[ Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket}  
 $\in \text{parts (spies evs)}$ ;  
 Crypt authK {Key servK, Agent B, Number Ts, servTicket}  
 $\in \text{parts (spies evs)}$ ;  
 $\neg \text{expiredAK Ta evs}$ ;  $\neg \text{expiredSK Ts evs}$ ;  
 $A \notin \text{bad}$ ;  $B \notin \text{bad}$ ;  $\text{evs} \in \text{kerbIV}$  ]]  
 $\implies \text{Key servK} \notin \text{analz (spies evs)}$ "  
 $\langle \text{proof} \rangle$

**lemma Confidentiality\_B:**

"[[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}  
 $\in \text{parts (spies evs)}$ ;  
 Crypt authK {Key servK, Agent B, Number Ts, servTicket}  
 $\in \text{parts (spies evs)}$ ;  
 Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket}  
 $\in \text{parts (spies evs)}$ ;  
 $\neg \text{expiredSK Ts evs}$ ;  $\neg \text{expiredAK Ta evs}$ ;  
 $A \notin \text{bad}$ ;  $B \notin \text{bad}$ ;  $B \neq \text{Tgs}$ ;  $\text{evs} \in \text{kerbIV}$  ]]  
 $\implies \text{Key servK} \notin \text{analz (spies evs)}$ "  
 $\langle \text{proof} \rangle$

**lemma u\_Confidentiality\_B:**

"[[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}  
 $\in \text{parts (spies evs)}$ ;  
 $\neg \text{expiredSK Ts evs}$ ;  
 $A \notin \text{bad}$ ;  $B \notin \text{bad}$ ;  $B \neq \text{Tgs}$ ;  $\text{evs} \in \text{kerbIV}$  ]]  
 $\implies \text{Key servK} \notin \text{analz (spies evs)}$ "  
 $\langle \text{proof} \rangle$

### 6.11 Parties authentication: each party verifies "the identity of another party who generated some data" (quoted from Neuman and Ts'o).

These guarantees don't assess whether two parties agree on the same session key: sending a message containing a key doesn't a priori state knowledge of the key.

$\text{Tgs\_authenticates\_A}$  can be found above

**lemma A\_authenticates\_Tgs:**

"[[ Says Kas A  
 (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket})  $\in \text{set evs}$ ;  
 Crypt authK {Key servK, Agent B, Number Ts, servTicket}  
 $\in \text{parts (spies evs)}$ ;  
 $\text{Key authK} \notin \text{analz (spies evs)}$ ;  
 $\text{evs} \in \text{kerbIV}$  ]]  
 $\implies \text{Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})}$   
 $\in \text{set evs}$ "  
 $\langle \text{proof} \rangle$



6.11 Parties authentication: each party verifies "the identity of another party who generated some data" (quoted from

**lemma** *B\_authenticates\_A*:

```
"[ Crypt servK {Agent A, Number T3} ∈ parts (spies evs);
  Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
  ∈ parts (spies evs);
  Key servK ∉ analz (spies evs);
  A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbIV ]
⇒ Says A B {Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}},
  Crypt servK {Agent A, Number T3} ∈ set evs"
```

*<proof>*

The second assumption tells B what kind of key servK is.

**lemma** *B\_authenticates\_A\_r*:

```
"[ Crypt servK {Agent A, Number T3} ∈ parts (spies evs);
  Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
  ∈ parts (spies evs);
  Crypt authK {Key servK, Agent B, Number Ts, servTicket}
  ∈ parts (spies evs);
  Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket}
  ∈ parts (spies evs);
  ¬ expiredSK Ts evs; ¬ expiredAK Ta evs;
  B ≠ Tgs; A ∉ bad; B ∉ bad; evs ∈ kerbIV ]
⇒ Says A B {Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}},
  Crypt servK {Agent A, Number T3} ∈ set evs"
```

*<proof>*

*u\_B\_authenticates\_A* would be the same as *B\_authenticates\_A* because the servK confidentiality assumption is yet unrelaxed

**lemma** *u\_B\_authenticates\_A\_r*:

```
"[ Crypt servK {Agent A, Number T3} ∈ parts (spies evs);
  Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
  ∈ parts (spies evs);
  ¬ expiredSK Ts evs;
  B ≠ Tgs; A ∉ bad; B ∉ bad; evs ∈ kerbIV ]
⇒ Says A B {Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}},
  Crypt servK {Agent A, Number T3} ∈ set evs"
```

*<proof>*

**lemma** *A\_authenticates\_B*:

```
"[ Crypt servK (Number T3) ∈ parts (spies evs);
  Crypt authK {Key servK, Agent B, Number Ts, servTicket}
  ∈ parts (spies evs);
  Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket}
  ∈ parts (spies evs);
  Key authK ∉ analz (spies evs); Key servK ∉ analz (spies evs);
  A ∉ bad; B ∉ bad; evs ∈ kerbIV ]
⇒ Says B A (Crypt servK (Number T3)) ∈ set evs"
```

*<proof>*

**lemma** *A\_authenticates\_B\_r*:

```
"[ Crypt servK (Number T3) ∈ parts (spies evs);
  Crypt authK {Key servK, Agent B, Number Ts, servTicket}
  ∈ parts (spies evs);
```

$$\begin{aligned} & \text{Crypt (shrK A) } \{ \text{Key authK, Agent Tgs, Number Ta, authTicket} \} \\ & \in \text{parts (spies evs);} \\ & \neg \text{expiredAK Ta evs; } \neg \text{expiredSK Ts evs;} \\ & A \notin \text{bad; } B \notin \text{bad; evs} \in \text{kerbIV} \] \\ \implies & \text{Says B A (Crypt servK (Number T3))} \in \text{set evs}'' \\ \langle \text{proof} \rangle & \end{aligned}$$

### 6.12 Key distribution guarantees An agent knows a session key if he used it to issue a cipher. These guarantees also convey a stronger form of authentication - non-injective agreement on the session key

**lemma** *Kas\_Issues\_A*:

$$\begin{aligned} & "[ \text{Says Kas A (Crypt (shrK A) } \{ \text{Key authK, Peer, Ta, authTicket} \})} \in \text{set} \\ & \text{evs;} \\ & \text{evs} \in \text{kerbIV} \] \\ \implies & \text{Kas Issues A with (Crypt (shrK A) } \{ \text{Key authK, Peer, Ta, authTicket} \}) \\ & \text{on evs}'' \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma** *A\_authenticates\_and\_keydist\_to\_Kas*:

$$\begin{aligned} & "[ \text{Crypt (shrK A) } \{ \text{Key authK, Peer, Ta, authTicket} \} \in \text{parts (spies evs);} \\ & A \notin \text{bad; evs} \in \text{kerbIV} \] \\ \implies & \text{Kas Issues A with (Crypt (shrK A) } \{ \text{Key authK, Peer, Ta, authTicket} \}) \\ & \text{on evs}'' \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma** *honest\_never\_says\_newer\_timestamp\_in\_auth*:

$$\begin{aligned} & "[ (CT \text{ evs}) \leq T; A \notin \text{bad; Number T} \in \text{parts } \{X\}; \text{evs} \in \text{kerbIV} \] \\ \implies & \forall B Y. \text{Says A B } \{Y, X\} \notin \text{set evs}'' \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma** *honest\_never\_says\_current\_timestamp\_in\_auth*:

$$\begin{aligned} & "[ (CT \text{ evs}) = T; \text{Number T} \in \text{parts } \{X\}; \text{evs} \in \text{kerbIV} \] \\ \implies & \forall A B Y. A \notin \text{bad} \longrightarrow \text{Says A B } \{Y, X\} \notin \text{set evs}'' \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma** *A\_trusts\_secure\_authenticator*:

$$\begin{aligned} & "[ \text{Crypt K } \{ \text{Agent A, Number T} \} \in \text{parts (spies evs);} \\ & \text{Key K} \notin \text{analz (spies evs); evs} \in \text{kerbIV} \] \\ \implies & \exists B X. \text{Says A Tgs } \{X, \text{Crypt K } \{ \text{Agent A, Number T} \}, \text{Agent B}\} \in \text{set evs} \\ \vee & \\ & \text{Says A B } \{X, \text{Crypt K } \{ \text{Agent A, Number T} \}\} \in \text{set evs}'' \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma** *A\_Issues\_Tgs*:

$$\begin{aligned} & "[ \text{Says A Tgs } \{ \text{authTicket, Crypt authK } \{ \text{Agent A, Number T2} \}, \text{Agent B} \} \\ & \in \text{set evs;} \\ & \text{Key authK} \notin \text{analz (spies evs);} \\ & A \notin \text{bad; evs} \in \text{kerbIV} \] \\ \implies & \text{A Issues Tgs with (Crypt authK } \{ \text{Agent A, Number T2} \}) \text{ on evs}'' \end{aligned}$$

6.12 Key distribution guarantees An agent knows a session key if he used it to issue a cipher. These guarantees also

*<proof>*

**lemma** *Tgs\_authenticates\_and\_keydist\_to\_A:*

```
"[[ Crypt authK {Agent A, Number T2} ∈ parts (spies evs);
   Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}
   ∈ parts (spies evs);
   Key authK ∉ analz (spies evs);
   A ∉ bad; evs ∈ kerbIV ]]
⇒ A Issues Tgs with (Crypt authK {Agent A, Number T2}) on evs"
<proof>
```

**lemma** *Tgs\_Issues\_A:*

```
"[[ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket
})
   ∈ set evs;
   Key authK ∉ analz (spies evs); evs ∈ kerbIV ]]
⇒ Tgs Issues A with
   (Crypt authK {Key servK, Agent B, Number Ts, servTicket }) on evs"
<proof>
```

**lemma** *A\_authenticates\_and\_keydist\_to\_Tgs:*

```
"[[Crypt authK {Key servK, Agent B, Number Ts, servTicket} ∈ parts (spies evs);
   Key authK ∉ analz (spies evs); B ≠ Tgs; evs ∈ kerbIV ]]
⇒ ∃A. Tgs Issues A with
   (Crypt authK {Key servK, Agent B, Number Ts, servTicket }) on evs"
<proof>
```

**lemma** *B\_Issues\_A:*

```
"[[ Says B A (Crypt servK (Number T3)) ∈ set evs;
   Key servK ∉ analz (spies evs);
   A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbIV ]]
⇒ B Issues A with (Crypt servK (Number T3)) on evs"
<proof>
```

**lemma** *B\_Issues\_A\_r:*

```
"[[ Says B A (Crypt servK (Number T3)) ∈ set evs;
   Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
   ∈ parts (spies evs);
   Crypt authK {Key servK, Agent B, Number Ts, servTicket}
   ∈ parts (spies evs);
   Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket}
   ∈ parts (spies evs);
   ¬ expiredSK Ts evs; ¬ expiredAK Ta evs;
   A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbIV ]]
⇒ B Issues A with (Crypt servK (Number T3)) on evs"
<proof>
```

**lemma** *u\_B\_Issues\_A\_r:*

```
"[[ Says B A (Crypt servK (Number T3)) ∈ set evs;
   Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
   ∈ parts (spies evs);
   ¬ expiredSK Ts evs;
```

$A \notin \text{bad}; B \notin \text{bad}; B \neq \text{Tgs}; \text{evs} \in \text{kerbIV} \parallel$   
 $\implies B \text{ Issues } A \text{ with } (\text{Crypt servK (Number T3)}) \text{ on evs''}$   
 <proof>

**lemma** *A\_authenticates\_and\_keydist\_to\_B:*  
 $\parallel \text{Crypt servK (Number T3)} \in \text{parts (spies evs)};$   
 $\text{Crypt authK } \{ \text{Key servK, Agent B, Number Ts, servTicket} \}$   
 $\in \text{parts (spies evs)};$   
 $\text{Crypt (shrK A)} \{ \text{Key authK, Agent Tgs, Number Ta, authTicket} \}$   
 $\in \text{parts (spies evs)};$   
 $\text{Key authK} \notin \text{analz (spies evs)}; \text{Key servK} \notin \text{analz (spies evs)};$   
 $A \notin \text{bad}; B \notin \text{bad}; B \neq \text{Tgs}; \text{evs} \in \text{kerbIV} \parallel$   
 $\implies B \text{ Issues } A \text{ with } (\text{Crypt servK (Number T3)}) \text{ on evs''}$   
 <proof>

**lemma** *A\_authenticates\_and\_keydist\_to\_B\_r:*  
 $\parallel \text{Crypt servK (Number T3)} \in \text{parts (spies evs)};$   
 $\text{Crypt authK } \{ \text{Key servK, Agent B, Number Ts, servTicket} \}$   
 $\in \text{parts (spies evs)};$   
 $\text{Crypt (shrK A)} \{ \text{Key authK, Agent Tgs, Number Ta, authTicket} \}$   
 $\in \text{parts (spies evs)};$   
 $\neg \text{expiredAK Ta evs}; \neg \text{expiredSK Ts evs};$   
 $A \notin \text{bad}; B \notin \text{bad}; B \neq \text{Tgs}; \text{evs} \in \text{kerbIV} \parallel$   
 $\implies B \text{ Issues } A \text{ with } (\text{Crypt servK (Number T3)}) \text{ on evs''}$   
 <proof>

**lemma** *A\_Issues\_B:*  
 $\parallel \text{Says A B } \{ \text{servTicket, Crypt servK } \{ \text{Agent A, Number T3} \} \}$   
 $\in \text{set evs};$   
 $\text{Key servK} \notin \text{analz (spies evs)};$   
 $B \neq \text{Tgs}; A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{kerbIV} \parallel$   
 $\implies A \text{ Issues B with } (\text{Crypt servK } \{ \text{Agent A, Number T3} \}) \text{ on evs''}$   
 <proof>

**lemma** *A\_Issues\_B\_r:*  
 $\parallel \text{Says A B } \{ \text{servTicket, Crypt servK } \{ \text{Agent A, Number T3} \} \}$   
 $\in \text{set evs};$   
 $\text{Crypt (shrK A)} \{ \text{Key authK, Agent Tgs, Number Ta, authTicket} \}$   
 $\in \text{parts (spies evs)};$   
 $\text{Crypt authK } \{ \text{Key servK, Agent B, Number Ts, servTicket} \}$   
 $\in \text{parts (spies evs)};$   
 $\neg \text{expiredAK Ta evs}; \neg \text{expiredSK Ts evs};$   
 $B \neq \text{Tgs}; A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{kerbIV} \parallel$   
 $\implies A \text{ Issues B with } (\text{Crypt servK } \{ \text{Agent A, Number T3} \}) \text{ on evs''}$   
 <proof>

**lemma** *B\_authenticates\_and\_keydist\_to\_A:*  
 $\parallel \text{Crypt servK } \{ \text{Agent A, Number T3} \} \in \text{parts (spies evs)};$   
 $\text{Crypt (shrK B)} \{ \text{Agent A, Agent B, Key servK, Number Ts} \}$   
 $\in \text{parts (spies evs)};$   
 $\text{Key servK} \notin \text{analz (spies evs)};$   
 $B \neq \text{Tgs}; A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{kerbIV} \parallel$   
 $\implies A \text{ Issues B with } (\text{Crypt servK } \{ \text{Agent A, Number T3} \}) \text{ on evs''}$

*<proof>*

**lemma** *B\_authenticates\_and\_keydist\_to\_A\_r*:

```
"[[ Crypt servK {Agent A, Number T3} ∈ parts (spies evs);
   Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
   ∈ parts (spies evs);
   Crypt authK {Key servK, Agent B, Number Ts, servTicket}
   ∈ parts (spies evs);
   Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket}
   ∈ parts (spies evs);
   ¬ expiredSK Ts evs; ¬ expiredAK Ta evs;
   B ≠ Tgs; A ∉ bad; B ∉ bad; evs ∈ kerbIV ]]
⇒ A Issues B with (Crypt servK {Agent A, Number T3}) on evs"
```

*<proof>*

*u\_B\_authenticates\_and\_keydist\_to\_A* would be the same as *B\_authenticates\_and\_keydist\_to\_A* because the *servK* confidentiality assumption is yet unrelaxed

**lemma** *u\_B\_authenticates\_and\_keydist\_to\_A\_r*:

```
"[[ Crypt servK {Agent A, Number T3} ∈ parts (spies evs);
   Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
   ∈ parts (spies evs);
   ¬ expiredSK Ts evs;
   B ≠ Tgs; A ∉ bad; B ∉ bad; evs ∈ kerbIV ]]
⇒ A Issues B with (Crypt servK {Agent A, Number T3}) on evs"
```

*<proof>*

**end**

## 7 The Kerberos Protocol, Version IV

**theory** *KerberosIV\_Gets* **imports** *Public* **begin**

The "u" prefix indicates theorems referring to an updated version of the protocol. The "r" suffix indicates theorems where the confidentiality assumptions are relaxed by the corresponding arguments.

**abbreviation**

*Kas* :: agent where "Kas == Server"

**abbreviation**

*Tgs* :: agent where "Tgs == Friend 0"

**axiomatization** where

*Tgs\_not\_bad* [iff]: "*Tgs* ∉ bad"  
— *Tgs* is secure — we already know that *Kas* is secure

**definition**

```
authKeys :: "event list ⇒ key set" where
"authKeys evs = {authK. ∃ A Peer Ta. Says Kas A
  (Crypt (shrK A) {Key authK, Agent Peer, Number Ta,
    (Crypt (shrK Peer) {Agent A, Agent Peer, Key authK, Number
Ta})})"
```

$\mathbb{P}) \in \text{set evs}\}"$

#### definition

$\text{Unique} :: "[\text{event}, \text{event list}] \Rightarrow \text{bool}"$  ( $\langle \text{Unique } _ \text{ on } _ \rangle [0, 50] 50$ )  
 where  $"(\text{Unique } \text{ev} \text{ on } \text{evs}) = (\text{ev} \notin \text{set } (\text{tl } (\text{dropWhile } (\lambda z. z \neq \text{ev}) \text{evs})))"$

#### consts

$\text{authKlife} :: \text{nat}$

$\text{servKlife} :: \text{nat}$

$\text{authlife} :: \text{nat}$

$\text{replylife} :: \text{nat}$

#### specification (authKlife)

$\text{authKlife\_LB} [\text{iff}]: "2 \leq \text{authKlife}"$   
 $\langle \text{proof} \rangle$

#### specification (servKlife)

$\text{servKlife\_LB} [\text{iff}]: "2 + \text{authKlife} \leq \text{servKlife}"$   
 $\langle \text{proof} \rangle$

#### specification (authlife)

$\text{authlife\_LB} [\text{iff}]: "\text{Suc } 0 \leq \text{authlife}"$   
 $\langle \text{proof} \rangle$

#### specification (replylife)

$\text{replylife\_LB} [\text{iff}]: "\text{Suc } 0 \leq \text{replylife}"$   
 $\langle \text{proof} \rangle$

#### abbreviation

$\text{CT} :: "[\text{event list}] \Rightarrow \text{nat}"$  where  
 $"\text{CT} == \text{length}"$

#### abbreviation

$\text{expiredAK} :: "[\text{nat}, \text{event list}] \Rightarrow \text{bool}"$  where  
 $"\text{expiredAK } \text{Ta} \text{ evs} == \text{authKlife} + \text{Ta} < \text{CT } \text{evs}"$

#### abbreviation

$\text{expiredSK} :: "[\text{nat}, \text{event list}] \Rightarrow \text{bool}"$  where  
 $"\text{expiredSK } \text{Ts} \text{ evs} == \text{servKlife} + \text{Ts} < \text{CT } \text{evs}"$

#### abbreviation

$\text{expiredA} :: "[\text{nat}, \text{event list}] \Rightarrow \text{bool}"$  where  
 $"\text{expiredA } \text{T} \text{ evs} == \text{authlife} + \text{T} < \text{CT } \text{evs}"$

**abbreviation**

```
valid :: "[nat, nat] ⇒ bool" (<valid _ wrt _> [0, 50] 50) where
  "valid T1 wrt T2 == T1 ≤ replylife + T2"
```

**definition AKcryptSK :: "[key, key, event list] ⇒ bool" where**

```
"AKcryptSK authK servK evs ==
  ∃ A B Ts.
    Says Tgs A (Crypt authK
      {Key servK, Agent B, Number Ts,
       Crypt (shrK B) {Agent A, Agent B, Key servK, Number
Ts}})
      ∈ set evs"
```

**inductive\_set "kerbIV\_gets" :: "event list set"**

**where**

```
Nil: "[ ] ∈ kerbIV_gets"

/ Fake: "[ evsf ∈ kerbIV_gets; X ∈ synth (analz (spies evsf)) ]
  ⇒ Says Spy B X # evsf ∈ kerbIV_gets"

/ Reception: "[ evsr ∈ kerbIV_gets; Says A B X ∈ set evsr ]
  ⇒ Gets B X # evsr ∈ kerbIV_gets"

/ K1: "[ evs1 ∈ kerbIV_gets ]
  ⇒ Says A Kas {Agent A, Agent Tgs, Number (CT evs1)} # evs1
  ∈ kerbIV_gets"

/ K2: "[ evs2 ∈ kerbIV_gets; Key authK ∉ used evs2; authK ∈ symKeys;
  Gets Kas {Agent A, Agent Tgs, Number T1} ∈ set evs2 ]
  ⇒ Says Kas A
    (Crypt (shrK A) {Key authK, Agent Tgs, Number (CT evs2),
    (Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK,
    Number (CT evs2)}}) # evs2 ∈ kerbIV_gets"

/ K3: "[ evs3 ∈ kerbIV_gets;
  Says A Kas {Agent A, Agent Tgs, Number T1} ∈ set evs3;
  Gets A (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,
    authTicket}) ∈ set evs3;
  valid Ta wrt T1"
```

```

    ]
    ⇒ Says A Tgs {authTicket,
                  (Crypt authK {Agent A, Number (CT evs3)}),
                  Agent B} # evs3 ∈ kerbIV_gets"

/ K4: "[ evs4 ∈ kerbIV_gets; Key servK ∉ used evs4; servK ∈ symKeys;
        B ≠ Tgs; authK ∈ symKeys;
        Gets Tgs {
          (Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK,
                             Number Ta}),
          (Crypt authK {Agent A, Number T2}), Agent B}
        ∈ set evs4;
        ¬ expiredAK Ta evs4;
        ¬ expiredA T2 evs4;
        servKlife + (CT evs4) ≤ authKlife + Ta
      ]
    ⇒ Says Tgs A
      (Crypt authK {Key servK, Agent B, Number (CT evs4),
                  Crypt (shrK B) {Agent A, Agent B, Key servK,
                                   Number (CT evs4)}}
      # evs4 ∈ kerbIV_gets"

/ K5: "[ evs5 ∈ kerbIV_gets; authK ∈ symKeys; servK ∈ symKeys;
        Says A Tgs
          {authTicket, Crypt authK {Agent A, Number T2},
           Agent B}
        ∈ set evs5;
        Gets A
          (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
          ∈ set evs5;
        valid Ts wrt T2 ]
    ⇒ Says A B {servTicket,
                  Crypt servK {Agent A, Number (CT evs5)}}
      # evs5 ∈ kerbIV_gets"

/ K6: "[ evs6 ∈ kerbIV_gets;
        Gets B {
          (Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}),
          (Crypt servK {Agent A, Number T3})}
        ∈ set evs6;
        ¬ expiredSK Ts evs6;

```



```

      ¬ expiredA T3 evs6
    ]
  ⇒ Says B A (Crypt servK (Number T3))
    # evs6 ∈ kerbIV_gets"

/ Oops1: "[ evs01 ∈ kerbIV_gets; A ≠ Spy;
  Says Kas A
    (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,
      authTicket}) ∈ set evs01;
  expiredAK Ta evs01 ]
⇒ Says A Spy {Agent A, Agent Tgs, Number Ta, Key authK}
  # evs01 ∈ kerbIV_gets"

/ Oops2: "[ evs02 ∈ kerbIV_gets; A ≠ Spy;
  Says Tgs A
    (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
    ∈ set evs02;
  expiredSK Ts evs02 ]
⇒ Says A Spy {Agent A, Agent B, Number Ts, Key servK}
  # evs02 ∈ kerbIV_gets"

declare Says_imp_knows_Spy [THEN parts.Inj, dest]
declare parts.Body [dest]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]

```

## 7.1 Lemmas about reception event

```

lemma Gets_imp_Says :
  "[ Gets B X ∈ set evs; evs ∈ kerbIV_gets ] ⇒ ∃A. Says A B X ∈ set
  evs"
  <proof>

lemma Gets_imp_knows_Spy:
  "[ Gets B X ∈ set evs; evs ∈ kerbIV_gets ] ⇒ X ∈ knows Spy evs"
  <proof>

declare Gets_imp_knows_Spy [THEN parts.Inj, dest]

lemma Gets_imp_knows:
  "[ Gets B X ∈ set evs; evs ∈ kerbIV_gets ] ⇒ X ∈ knows B evs"
  <proof>

```

## 7.2 Lemmas about `authKeys`

**lemma** `authKeys_empty`: "`authKeys [] = {}`"  
 $\langle proof \rangle$

**lemma** `authKeys_not_insert`:  
 $(\forall A \text{ Ta akey Peer.}$   
 $\text{ev} \neq \text{Says Kas A (Crypt (shrK A) \{akey, Agent Peer, Ta,$   
 $\text{(Crypt (shrK Peer) \{Agent A, Agent Peer, akey, Ta\})\})}$   
 $\implies \text{authKeys (ev \# evs) = authKeys evs}$ "  
 $\langle proof \rangle$

**lemma** `authKeys_insert`:  
 $\text{"authKeys}$   
 $\text{(Says Kas A (Crypt (shrK A) \{Key K, Agent Peer, Number Ta,  
 $\text{(Crypt (shrK Peer) \{Agent A, Agent Peer, Key K, Number Ta\})\}) \# evs)}$   
 $\text{= insert K (authKeys evs)"}$   
 $\langle proof \rangle$$

**lemma** `authKeys_simp`:  
 $\text{"K} \in \text{authKeys}$   
 $\text{(Says Kas A (Crypt (shrK A) \{Key K', Agent Peer, Number Ta,  
 $\text{(Crypt (shrK Peer) \{Agent A, Agent Peer, Key K', Number Ta\})\}) \# evs)}$   
 $\implies K = K' \mid K \in \text{authKeys evs}"$   
 $\langle proof \rangle$$

**lemma** `authKeysI`:  
 $\text{"Says Kas A (Crypt (shrK A) \{Key K, Agent Tgs, Number Ta,  
 $\text{(Crypt (shrK Tgs) \{Agent A, Agent Tgs, Key K, Number Ta\})\})} \in \text{set evs}$   
 $\implies K \in \text{authKeys evs}"$   
 $\langle proof \rangle$$

**lemma** `authKeys_used`: " $K \in \text{authKeys evs} \implies \text{Key K} \in \text{used evs}$ "  
 $\langle proof \rangle$

## 7.3 Forwarding Lemmas

**lemma** `Says_ticket_parts`:  
 $\text{"Says S A (Crypt K \{SesKey, B, TimeStamp, Ticket\})} \in \text{set evs}$   
 $\implies \text{Ticket} \in \text{parts (spies evs)"}$   
 $\langle proof \rangle$

**lemma** `Gets_ticket_parts`:  
 $\text{"[[Gets A (Crypt K \{SesKey, Peer, Ta, Ticket\})} \in \text{set evs; evs} \in \text{kerbIV_gets}$   
 $\text{]]}$   
 $\implies \text{Ticket} \in \text{parts (spies evs)"}$   
 $\langle proof \rangle$

**lemma** `Ops_range_spies1`:  
 $\text{"[[ Says Kas A (Crypt KeyA \{Key authK, Peer, Ta, authTicket\})}$   
 $\text{} \in \text{set evs ;}$   
 $\text{evs} \in \text{kerbIV_gets ]} \implies \text{authK} \notin \text{range shrK} \wedge \text{authK} \in \text{symKeys}"$   
 $\langle proof \rangle$

**lemma** `Ops_range_spies2`:

```

"[[ Says Tgs A (Crypt authK {Key servK, Agent B, Ts, servTicket})
  ∈ set evs ;
  evs ∈ kerbIV_gets ]] ⇒ servK ∉ range shrK ∧ servK ∈ symKeys"
⟨proof⟩

```

```

lemma Spy_see_shrK [simp]:
  "evs ∈ kerbIV_gets ⇒ (Key (shrK A) ∈ parts (spies evs)) = (A ∈ bad)"
⟨proof⟩

```

```

lemma Spy_analz_shrK [simp]:
  "evs ∈ kerbIV_gets ⇒ (Key (shrK A) ∈ analz (spies evs)) = (A ∈ bad)"
⟨proof⟩

```

```

lemma Spy_see_shrK_D [dest!]:
  "[[ Key (shrK A) ∈ parts (spies evs); evs ∈ kerbIV_gets ]] ⇒ A ∈ bad"
⟨proof⟩
lemmas Spy_analz_shrK_D = analz_subset_parts [THEN subsetD, THEN Spy_see_shrK_D,
dest!]

```

Nobody can have used non-existent keys!

```

lemma new_keys_not_used [simp]:
  "[[Key K ∉ used evs; K ∈ symKeys; evs ∈ kerbIV_gets]]
  ⇒ K ∉ keysFor (parts (spies evs))"
⟨proof⟩

```

```

lemma new_keys_not_analzD:
  "[[evs ∈ kerbIV_gets; K ∈ symKeys; Key K ∉ used evs]
  ⇒ K ∉ keysFor (analz (spies evs))"
⟨proof⟩

```

## 7.4 Regularity Lemmas

These concern the form of items passed in messages

Describes the form of all components sent by Kas

```

lemma Says_Kas_message_form:
  "[[ Says Kas A (Crypt K {Key authK, Agent Peer, Number Ta, authTicket})
  ∈ set evs;
  evs ∈ kerbIV_gets ]] ⇒
  K = shrK A ∧ Peer = Tgs ∧
  authK ∉ range shrK ∧ authK ∈ authKeys evs ∧ authK ∈ symKeys ∧
  authTicket = (Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta})"
⟨proof⟩

```

```

lemma SesKey_is_session_key:
  "[[ Crypt (shrK Tgs_B) {Agent A, Agent Tgs_B, Key SesKey, Number T}
  ∈ parts (spies evs); Tgs_B ∉ bad;
  evs ∈ kerbIV_gets ]]
  ⇒ SesKey ∉ range shrK"

```

*<proof>*

**lemma** *authTicket\_authentic*:

"[[ Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}  
     ∈ parts (spies evs);  
     evs ∈ kerbIV\_gets ]]  
 ⇒ Says Kas A (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,  
     Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}})  
     ∈ set evs"

*<proof>*

**lemma** *authTicket\_crypt\_authK*:

"[[ Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}  
     ∈ parts (spies evs);  
     evs ∈ kerbIV\_gets ]]  
 ⇒ authK ∈ authKeys evs"

*<proof>*

**lemma** *Says\_Tgs\_message\_form*:

"[[ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})  
     ∈ set evs;  
     evs ∈ kerbIV\_gets ]]  
 ⇒ B ≠ Tgs ∧  
     authK ∉ range shrK ∧ authK ∈ authKeys evs ∧ authK ∈ symKeys ∧  
     servK ∉ range shrK ∧ servK ∉ authKeys evs ∧ servK ∈ symKeys ∧  
     servTicket = (Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts})"

*<proof>*

**lemma** *authTicket\_form*:

"[[ Crypt (shrK A) {Key authK, Agent Tgs, Ta, authTicket}  
     ∈ parts (spies evs);  
     A ∉ bad;  
     evs ∈ kerbIV\_gets ]]  
 ⇒ authK ∉ range shrK ∧ authK ∈ symKeys ∧  
     authTicket = Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Ta}"

*<proof>*

This form holds also over an authTicket, but is not needed below.

**lemma** *servTicket\_form*:

"[[ Crypt authK {Key servK, Agent B, Ts, servTicket}  
     ∈ parts (spies evs);  
     Key authK ∉ analz (spies evs);  
     evs ∈ kerbIV\_gets ]]  
 ⇒ servK ∉ range shrK ∧ servK ∈ symKeys ∧  
 (∃ A. servTicket = Crypt (shrK B) {Agent A, Agent B, Key servK, Ts})"

*<proof>*

Essentially the same as *authTicket\_form*

**lemma** *Says\_kas\_message\_form*:

"[[ Gets A (Crypt (shrK A)  
     {Key authK, Agent Tgs, Ta, authTicket}) ∈ set evs;  
     evs ∈ kerbIV\_gets ]]  
 ⇒ authK ∉ range shrK ∧ authK ∈ symKeys ∧

```

      authTicket =
        Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Ta}
      / authTicket ∈ analz (spies evs)"
⟨proof⟩

```

**lemma** *Says\_tgs\_message\_form*:

```

"[[ Gets A (Crypt authK {Key servK, Agent B, Ts, servTicket})
   ∈ set evs; authK ∈ symKeys;
   evs ∈ kerbIV_gets ]]
⇒ servK ∉ range shrK ∧
   (∃ A. servTicket =
     Crypt (shrK B) {Agent A, Agent B, Key servK, Ts})
   / servTicket ∈ analz (spies evs)"
⟨proof⟩

```

## 7.5 Authenticity theorems: confirm origin of sensitive messages

**lemma** *authK\_authentic*:

```

"[[ Crypt (shrK A) {Key authK, Peer, Ta, authTicket}
   ∈ parts (spies evs);
   A ∉ bad; evs ∈ kerbIV_gets ]]
⇒ Says Kas A (Crypt (shrK A) {Key authK, Peer, Ta, authTicket})
   ∈ set evs"
⟨proof⟩

```

If a certain encrypted message appears then it originated with Tgs

**lemma** *servK\_authentic*:

```

"[[ Crypt authK {Key servK, Agent B, Number Ts, servTicket}
   ∈ parts (spies evs);
   Key authK ∉ analz (spies evs);
   authK ∉ range shrK;
   evs ∈ kerbIV_gets ]]
⇒ ∃ A. Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
   ∈ set evs"
⟨proof⟩

```

**lemma** *servK\_authentic\_bis*:

```

"[[ Crypt authK {Key servK, Agent B, Number Ts, servTicket}
   ∈ parts (spies evs);
   Key authK ∉ analz (spies evs);
   B ≠ Tgs;
   evs ∈ kerbIV_gets ]]
⇒ ∃ A. Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
   ∈ set evs"
⟨proof⟩

```

Authenticity of servK for B

**lemma** *servTicket\_authentic\_Tgs*:

```

"[[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
   ∈ parts (spies evs); B ≠ Tgs; B ∉ bad;
   evs ∈ kerbIV_gets ]]
⇒ ∃ authK.

```

Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts,  
                   Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}})  
 ∈ set evs"  
 <proof>

Anticipated here from next subsection

**lemma** K4\_imp\_K2:  
 "[[ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})  
   ∈ set evs; evs ∈ kerbIV\_gets ]]  
 ⇒ ∃ Ta. Says Kas A  
   (Crypt (shrK A)  
     {Key authK, Agent Tgs, Number Ta,  
       Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}})  
   ∈ set evs"  
 <proof>

Anticipated here from next subsection

**lemma** u\_K4\_imp\_K2:  
 "[[ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})  
   ∈ set evs; evs ∈ kerbIV\_gets ]]  
 ⇒ ∃ Ta. (Says Kas A (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,  
                   Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}})  
   ∈ set evs  
   ∧ servKlife + Ts ≤ authKlife + Ta)"  
 <proof>

**lemma** servTicket\_authentic\_Kas:  
 "[[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}  
   ∈ parts (spies evs); B ≠ Tgs; B ∉ bad;  
   evs ∈ kerbIV\_gets ]]  
 ⇒ ∃ authK Ta.  
   Says Kas A  
     (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,  
       Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}})  
   ∈ set evs"  
 <proof>

**lemma** u\_servTicket\_authentic\_Kas:  
 "[[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}  
   ∈ parts (spies evs); B ≠ Tgs; B ∉ bad;  
   evs ∈ kerbIV\_gets ]]  
 ⇒ ∃ authK Ta. Says Kas A (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,  
                   Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}})  
   ∈ set evs  
   ∧ servKlife + Ts ≤ authKlife + Ta"  
 <proof>

**lemma** servTicket\_authentic:  
 "[[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}  
   ∈ parts (spies evs); B ≠ Tgs; B ∉ bad;  
   evs ∈ kerbIV\_gets ]]  
 ⇒ ∃ Ta authK.  
   Says Kas A (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,

## 7.6 Reliability: friendly agents send something if something else happened 95

```

      Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number
Ta}})
  ∈ set evs
  ∧ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts,
      Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}})
    ∈ set evs"
⟨proof⟩

```

**lemma** u\_servTicket\_authentic:

```

  "[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
    ∈ parts (spies evs); B ≠ Tgs; B ∉ bad;
    evs ∈ kerbIV_gets ]
  ⇒ ∃ Ta authK.
    (Says Kas A (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,
      Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number
Ta}})
      ∈ set evs
      ∧ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts,
      Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}})
        ∈ set evs
      ∧ servKlife + Ts ≤ authKlife + Ta)"
⟨proof⟩

```

**lemma** u\_NotexpiredSK\_NotexpiredAK:

```

  "[ ¬ expiredSK Ts evs; servKlife + Ts ≤ authKlife + Ta ]
  ⇒ ¬ expiredAK Ta evs"
⟨proof⟩

```

## 7.6 Reliability: friendly agents send something if something else happened

**lemma** K3\_imp\_K2:

```

  "[ Says A Tgs
    {authTicket, Crypt authK {Agent A, Number T2}, Agent B}
      ∈ set evs;
    A ∉ bad; evs ∈ kerbIV_gets ]
  ⇒ ∃ Ta. Says Kas A (Crypt (shrK A)
    {Key authK, Agent Tgs, Number Ta, authTicket})
      ∈ set evs"
⟨proof⟩

```

Anticipated here from next subsection. An authK is encrypted by one and only one Shared key. A servK is encrypted by one and only one authK.

**lemma** Key\_unique\_SesKey:

```

  "[ Crypt K {Key SesKey, Agent B, T, Ticket}
    ∈ parts (spies evs);
    Crypt K' {Key SesKey, Agent B', T', Ticket'}
      ∈ parts (spies evs); Key SesKey ∉ analz (spies evs);
    evs ∈ kerbIV_gets ]
  ⇒ K=K' ∧ B=B' ∧ T=T' ∧ Ticket=Ticket'"
⟨proof⟩

```

**lemma** Tgs\_authenticates\_A:

```

  "[ Crypt authK {Agent A, Number T2} ∈ parts (spies evs);

```

$$\begin{aligned} & \text{Crypt (shrK Tgs)} \{ \text{Agent A, Agent Tgs, Key authK, Number Ta} \} \\ & \quad \in \text{parts (spies evs)}; \\ & \text{Key authK} \notin \text{analz (spies evs)}; A \notin \text{bad}; \text{evs} \in \text{kerbIV\_gets} \] \\ \implies & \exists B. \text{Says A Tgs} \[ \\ & \quad \text{Crypt (shrK Tgs)} \{ \text{Agent A, Agent Tgs, Key authK, Number Ta} \}, \\ & \quad \text{Crypt authK} \{ \text{Agent A, Number T2} \}, \text{Agent B} \] \in \text{set evs}'' \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma Says\_K5:**

$$\begin{aligned} & "[ \text{Crypt servK} \{ \text{Agent A, Number T3} \} \in \text{parts (spies evs)}; \\ & \quad \text{Says Tgs A (Crypt authK} \{ \text{Key servK, Agent B, Number Ts,} \\ & \quad \quad \text{servTicket} \}) \in \text{set evs}; \\ & \quad \text{Key servK} \notin \text{analz (spies evs)}; \\ & \quad A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{kerbIV\_gets} \] \\ \implies & \text{Says A B} \{ \text{servTicket, Crypt servK} \{ \text{Agent A, Number T3} \} \} \in \text{set evs}'' \\ & \langle \text{proof} \rangle \end{aligned}$$

Anticipated here from next subsection

**lemma unique\_CryptKey:**

$$\begin{aligned} & "[ \text{Crypt (shrK B)} \{ \text{Agent A, Agent B, Key SesKey, T} \} \\ & \quad \in \text{parts (spies evs)}; \\ & \quad \text{Crypt (shrK B')} \{ \text{Agent A', Agent B', Key SesKey, T'} \} \\ & \quad \in \text{parts (spies evs)}; \text{Key SesKey} \notin \text{analz (spies evs)}; \\ & \quad \text{evs} \in \text{kerbIV\_gets} \] \\ \implies & A=A' \wedge B=B' \wedge T=T'' \\ & \langle \text{proof} \rangle \end{aligned}$$

**lemma Says\_K6:**

$$\begin{aligned} & "[ \text{Crypt servK (Number T3)} \in \text{parts (spies evs)}; \\ & \quad \text{Says Tgs A (Crypt authK} \{ \text{Key servK, Agent B, Number Ts,} \\ & \quad \quad \text{servTicket} \}) \in \text{set evs}; \\ & \quad \text{Key servK} \notin \text{analz (spies evs)}; \\ & \quad A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{kerbIV\_gets} \] \\ \implies & \text{Says B A (Crypt servK (Number T3))} \in \text{set evs}'' \\ & \langle \text{proof} \rangle \end{aligned}$$

Needs a unicity theorem, hence moved here

**lemma servK\_authentic\_ter:**

$$\begin{aligned} & "[ \text{Says Kas A} \\ & \quad (\text{Crypt (shrK A)} \{ \text{Key authK, Agent Tgs, Number Ta, authTicket} \}) \in \text{set evs}; \\ & \quad \text{Crypt authK} \{ \text{Key servK, Agent B, Number Ts, servTicket} \} \\ & \quad \in \text{parts (spies evs)}; \\ & \quad \text{Key authK} \notin \text{analz (spies evs)}; \\ & \quad \text{evs} \in \text{kerbIV\_gets} \] \\ \implies & \text{Says Tgs A (Crypt authK} \{ \text{Key servK, Agent B, Number Ts, servTicket} \}) \\ & \quad \in \text{set evs}'' \\ & \langle \text{proof} \rangle \end{aligned}$$

## 7.7 Unicity Theorems

The session key, if secure, uniquely identifies the Ticket whether authTicket or servTicket. As a matter of fact, one can read also Tgs in the place of B.

**lemma unique\_authKeys:**



```

"[[ Says Kas A
  (Crypt Ka {Key authK, Agent Tgs, Ta, X}) ∈ set evs;
  Says Kas A'
  (Crypt Ka' {Key authK, Agent Tgs, Ta', X'}) ∈ set evs;
  evs ∈ kerbIV_gets ]] ⇒ A=A' ∧ Ka=Ka' ∧ Ta=Ta' ∧ X=X'"
⟨proof⟩

```

servK uniquely identifies the message from Tgs

**lemma** *unique\_servKeys*:

```

"[[ Says Tgs A
  (Crypt K {Key servK, Agent B, Ts, X}) ∈ set evs;
  Says Tgs A'
  (Crypt K' {Key servK, Agent B', Ts', X'}) ∈ set evs;
  evs ∈ kerbIV_gets ]] ⇒ A=A' ∧ B=B' ∧ K=K' ∧ Ts=Ts' ∧ X=X'"
⟨proof⟩

```

Revised unicity theorems

**lemma** *Kas\_Unique*:

```

"[[ Says Kas A
  (Crypt Ka {Key authK, Agent Tgs, Ta, authTicket}) ∈ set evs;
  evs ∈ kerbIV_gets ]] ⇒
  Unique (Says Kas A (Crypt Ka {Key authK, Agent Tgs, Ta, authTicket}))
  on evs"
⟨proof⟩

```

**lemma** *Tgs\_Unique*:

```

"[[ Says Tgs A
  (Crypt authK {Key servK, Agent B, Ts, servTicket}) ∈ set evs;
  evs ∈ kerbIV_gets ]] ⇒
  Unique (Says Tgs A (Crypt authK {Key servK, Agent B, Ts, servTicket}))
  on evs"
⟨proof⟩

```

## 7.8 Lemmas About the Predicate $AKcryptSK$

**lemma** *not\_AKcryptSK\_Nil [iff]*: " $\neg AKcryptSK\ authK\ servK\ []$ "

⟨proof⟩

**lemma** *AKcryptSKI*:

```

"[[ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, X}) ∈ set evs;
  evs ∈ kerbIV_gets ]] ⇒ AKcryptSK authK servK evs"
⟨proof⟩

```

**lemma** *AKcryptSK\_Says [simp]*:

```

"AKcryptSK authK servK (Says S A X # evs) =
  (Tgs = S ∧
   (∃ B Ts. X = Crypt authK
    {Key servK, Agent B, Number Ts,
     Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}}
   )
  | AKcryptSK authK servK evs)"
⟨proof⟩

```

**lemma** *Auth\_fresh\_not\_AKcryptSK:*  
 "[ Key authK  $\notin$  used evs; evs  $\in$  kerbIV\_gets ]  
 $\implies \neg$  AKcryptSK authK servK evs"  
 <proof>

**lemma** *Serv\_fresh\_not\_AKcryptSK:*  
 "Key servK  $\notin$  used evs  $\implies \neg$  AKcryptSK authK servK evs"  
 <proof>

**lemma** *authK\_not\_AKcryptSK:*  
 "[ Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, tk}  
 $\in$  parts (spies evs); evs  $\in$  kerbIV\_gets ]  
 $\implies \neg$  AKcryptSK K authK evs"  
 <proof>

A secure serverkey cannot have been used to encrypt others

**lemma** *servK\_not\_AKcryptSK:*  
 "[ Crypt (shrK B) {Agent A, Agent B, Key SK, Number Ts}  $\in$  parts (spies evs);  
 Key SK  $\notin$  analz (spies evs); SK  $\in$  symKeys;  
 B  $\neq$  Tgs; evs  $\in$  kerbIV\_gets ]  
 $\implies \neg$  AKcryptSK SK K evs"  
 <proof>

Long term keys are not issued as servKeys

**lemma** *shrK\_not\_AKcryptSK:*  
 "evs  $\in$  kerbIV\_gets  $\implies \neg$  AKcryptSK K (shrK A) evs"  
 <proof>

The Tgs message associates servK with authK and therefore not with any other key authK.

**lemma** *Says\_Tgs\_AKcryptSK:*  
 "[ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, X})  
 $\in$  set evs;  
 authK'  $\neq$  authK; evs  $\in$  kerbIV\_gets ]  
 $\implies \neg$  AKcryptSK authK' servK evs"  
 <proof>

Equivalently

**lemma** *not\_different\_AKcryptSK:*  
 "[ AKcryptSK authK servK evs;  
 authK'  $\neq$  authK; evs  $\in$  kerbIV\_gets ]  
 $\implies \neg$  AKcryptSK authK' servK evs  $\wedge$  servK  $\in$  symKeys"  
 <proof>

**lemma** *AKcryptSK\_not\_AKcryptSK:*  
 "[ AKcryptSK authK servK evs; evs  $\in$  kerbIV\_gets ]  
 $\implies \neg$  AKcryptSK servK K evs"  
 <proof>

The only session keys that can be found with the help of session keys are those sent by Tgs in step K4.

We take some pains to express the property as a logical equivalence so that the simplifier can apply it.

```

lemma Key_analz_image_Key_lemma:
  "P  $\longrightarrow$  (Key K  $\in$  analz (Key'KK  $\cup$  H))  $\longrightarrow$  (K  $\in$  KK | Key K  $\in$  analz H)
   $\implies$ 
  P  $\longrightarrow$  (Key K  $\in$  analz (Key'KK  $\cup$  H)) = (K  $\in$  KK | Key K  $\in$  analz H)"
<proof>

```

```

lemma AKcryptSK_analz_insert:
  "[[ AKcryptSK K K' evs; K  $\in$  symKeys; evs  $\in$  kerbIV_gets ]]
   $\implies$  Key K'  $\in$  analz (insert (Key K) (spies evs))"
<proof>

```

```

lemma authKeys_are_not_AKcryptSK:
  "[[ K  $\in$  authKeys evs  $\cup$  range shrK; evs  $\in$  kerbIV_gets ]]
   $\implies$   $\forall$  SK.  $\neg$  AKcryptSK SK K evs  $\wedge$  K  $\in$  symKeys"
<proof>

```

```

lemma not_authKeys_not_AKcryptSK:
  "[[ K  $\notin$  authKeys evs;
    K  $\notin$  range shrK; evs  $\in$  kerbIV_gets ]]
   $\implies$   $\forall$  SK.  $\neg$  AKcryptSK K SK evs"
<proof>

```

## 7.9 Secrecy Theorems

For the Oops2 case of the next theorem

```

lemma Oops2_not_AKcryptSK:
  "[[ evs  $\in$  kerbIV_gets;
    Says Tgs A (Crypt authK
      {Key servK, Agent B, Number Ts, servTicket})
       $\in$  set evs ]]
   $\implies$   $\neg$  AKcryptSK servK SK evs"
<proof>

```

Big simplification law for keys SK that are not crypted by keys in KK It helps prove three, otherwise hard, facts about keys. These facts are exploited as simplification laws for analz, and also "limit the damage" in case of loss of a key to the spy. See ESORICS98.

```

lemma Key_analz_image_Key [rule_format (no_asm)]:
  "evs  $\in$  kerbIV_gets  $\implies$ 
  ( $\forall$  SK KK. SK  $\in$  symKeys  $\wedge$  KK  $\subseteq$   $\neg$ (range shrK)  $\longrightarrow$ 
  ( $\forall$  K  $\in$  KK.  $\neg$  AKcryptSK K SK evs)  $\longrightarrow$ 
  (Key SK  $\in$  analz (Key'KK  $\cup$  (spies evs))) =
  (SK  $\in$  KK | Key SK  $\in$  analz (spies evs)))"
<proof>

```

First simplification law for analz: no session keys encrypt authentication keys or shared keys.

```

lemma analz_insert_freshK1:
  "[[ evs  $\in$  kerbIV_gets; K  $\in$  authKeys evs  $\cup$  range shrK;

```

$$\begin{aligned} & \text{SesKey} \notin \text{range shrK} \parallel \\ \implies & (\text{Key } K \in \text{analz} (\text{insert} (\text{Key SesKey}) (\text{spies evs}))) = \\ & (K = \text{SesKey} \mid \text{Key } K \in \text{analz} (\text{spies evs}))" \\ \langle \text{proof} \rangle & \end{aligned}$$

Second simplification law for analz: no service keys encrypt any other keys.

**lemma** *analz\_insert\_freshK2*:  

$$\begin{aligned} & "[\text{ evs} \in \text{kerbIV\_gets}; \text{ servK} \notin (\text{authKeys evs}); \text{ servK} \notin \text{range shrK}; \\ & \quad K \in \text{symKeys} ] \\ \implies & (\text{Key } K \in \text{analz} (\text{insert} (\text{Key servK}) (\text{spies evs}))) = \\ & (K = \text{servK} \mid \text{Key } K \in \text{analz} (\text{spies evs}))" \\ \langle \text{proof} \rangle & \end{aligned}$$

Third simplification law for analz: only one authentication key encrypts a certain service key.

**lemma** *analz\_insert\_freshK3*:  

$$\begin{aligned} & "[\text{ AKcryptSK authK servK evs}; \\ & \quad \text{authK}' \neq \text{authK}; \text{authK}' \notin \text{range shrK}; \text{ evs} \in \text{kerbIV\_gets} ] \\ \implies & (\text{Key servK} \in \text{analz} (\text{insert} (\text{Key authK}') (\text{spies evs}))) = \\ & (\text{servK} = \text{authK}' \mid \text{Key servK} \in \text{analz} (\text{spies evs}))" \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma** *analz\_insert\_freshK3\_bis*:  

$$\begin{aligned} & "[\text{ Says Tgs A} \\ & \quad (\text{Crypt authK } \{\text{Key servK}, \text{Agent B}, \text{Number Ts}, \text{servTicket}\}) \\ & \quad \in \text{set evs}; \\ & \quad \text{authK} \neq \text{authK}'; \text{authK}' \notin \text{range shrK}; \text{ evs} \in \text{kerbIV\_gets} ] \\ \implies & (\text{Key servK} \in \text{analz} (\text{insert} (\text{Key authK}') (\text{spies evs}))) = \\ & (\text{servK} = \text{authK}' \mid \text{Key servK} \in \text{analz} (\text{spies evs}))" \\ \langle \text{proof} \rangle & \end{aligned}$$

a weakness of the protocol

**lemma** *authK\_compromises\_servK*:  

$$\begin{aligned} & "[\text{ Says Tgs A} \\ & \quad (\text{Crypt authK } \{\text{Key servK}, \text{Agent B}, \text{Number Ts}, \text{servTicket}\}) \\ & \quad \in \text{set evs}; \text{ authK} \in \text{symKeys}; \\ & \quad \text{Key authK} \in \text{analz} (\text{spies evs}); \text{ evs} \in \text{kerbIV\_gets} ] \\ \implies & \text{Key servK} \in \text{analz} (\text{spies evs})" \\ \langle \text{proof} \rangle & \end{aligned}$$

**lemma** *servK\_notin\_authKeysD*:  

$$\begin{aligned} & "[\text{ Crypt authK } \{\text{Key servK}, \text{Agent B}, \text{Ts}, \\ & \quad \text{Crypt (shrK B) } \{\text{Agent A}, \text{Agent B}, \text{Key servK}, \text{Ts}\}\} \\ & \quad \in \text{parts} (\text{spies evs}); \\ & \quad \text{Key servK} \notin \text{analz} (\text{spies evs}); \\ & \quad B \neq \text{Tgs}; \text{ evs} \in \text{kerbIV\_gets} ] \\ \implies & \text{servK} \notin \text{authKeys evs}" \\ \langle \text{proof} \rangle & \end{aligned}$$

If Spy sees the Authentication Key sent in msg K2, then the Key has expired.

**lemma** *Confidentiality\_Kas\_lemma* [*rule\_format*]:  

$$\begin{aligned} & "[\text{ authK} \in \text{symKeys}; A \notin \text{bad}; \text{ evs} \in \text{kerbIV\_gets} ] \\ \implies & \text{Says Kas A} \end{aligned}$$

```

      (Crypt (shrK A)
        {Key authK, Agent Tgs, Number Ta,
         Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}}})
    ∈ set evs →
    Key authK ∈ analz (spies evs) →
    expiredAK Ta evs"
⟨proof⟩

```

**lemma Confidentiality\_Kas:**

```

"[[ Says Kas A
   (Crypt Ka {Key authK, Agent Tgs, Number Ta, authTicket})
   ∈ set evs;
   ¬ expiredAK Ta evs;
   A ∉ bad; evs ∈ kerbIV_gets ]]
⇒ Key authK ∉ analz (spies evs)"
⟨proof⟩

```

If Spy sees the Service Key sent in msg K4, then the Key has expired.

**lemma Confidentiality\_lemma [rule\_format]:**

```

"[[ Says Tgs A
   (Crypt authK
    {Key servK, Agent B, Number Ts,
     Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}}})
   ∈ set evs;
   Key authK ∉ analz (spies evs);
   servK ∈ symKeys;
   A ∉ bad; B ∉ bad; evs ∈ kerbIV_gets ]]
⇒ Key servK ∈ analz (spies evs) →
   expiredSK Ts evs"
⟨proof⟩

```

In the real world Tgs can't check wheter authK is secure!

**lemma Confidentiality\_Tgs:**

```

"[[ Says Tgs A
   (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
   ∈ set evs;
   Key authK ∉ analz (spies evs);
   ¬ expiredSK Ts evs;
   A ∉ bad; B ∉ bad; evs ∈ kerbIV_gets ]]
⇒ Key servK ∉ analz (spies evs)"
⟨proof⟩

```

In the real world Tgs CAN check what Kas sends!

**lemma Confidentiality\_Tgs\_bis:**

```

"[[ Says Kas A
   (Crypt Ka {Key authK, Agent Tgs, Number Ta, authTicket})
   ∈ set evs;
   Says Tgs A
   (Crypt authK {Key servK, Agent B, Number Ts, servTicket})
   ∈ set evs;
   ¬ expiredAK Ta evs; ¬ expiredSK Ts evs;
   A ∉ bad; B ∉ bad; evs ∈ kerbIV_gets ]]
⇒ Key servK ∉ analz (spies evs)"

```

*<proof>*

Most general form

**lemmas** Confidentiality\_Tgs\_ter = authTicket\_authentic [THEN Confidentiality\_Tgs\_bis]

**lemmas** Confidentiality\_Auth\_A = authK\_authentic [THEN Confidentiality\_Kas]

Needs a confidentiality guarantee, hence moved here. Authenticity of servK for A

**lemma** servK\_authentic\_bis\_r:  
 "[ Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket}  
   ∈ parts (spies evs);  
   Crypt authK {Key servK, Agent B, Number Ts, servTicket}  
   ∈ parts (spies evs);  
   ¬ expiredAK Ta evs; A ∉ bad; evs ∈ kerbIV\_gets ]  
 ⇒ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})  
   ∈ set evs"  
*<proof>*

**lemma** Confidentiality\_Serv\_A:  
 "[ Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket}  
   ∈ parts (spies evs);  
   Crypt authK {Key servK, Agent B, Number Ts, servTicket}  
   ∈ parts (spies evs);  
   ¬ expiredAK Ta evs; ¬ expiredSK Ts evs;  
   A ∉ bad; B ∉ bad; evs ∈ kerbIV\_gets ]  
 ⇒ Key servK ∉ analz (spies evs)"  
*<proof>*

**lemma** u\_Confidentiality\_B:  
 "[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}  
   ∈ parts (spies evs);  
   ¬ expiredSK Ts evs;  
   A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbIV\_gets ]  
 ⇒ Key servK ∉ analz (spies evs)"  
*<proof>*

## 7.10 2. Parties' strong authentication: non-injective agreement on the session key. The same guarantees also express key distribution, hence their names

Authentication here still is weak agreement - of B with A

**lemma** A\_authenticates\_B:  
 "[ Crypt servK (Number T3) ∈ parts (spies evs);  
   Crypt authK {Key servK, Agent B, Number Ts, servTicket}  
   ∈ parts (spies evs);  
   Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket}  
   ∈ parts (spies evs);  
   Key authK ∉ analz (spies evs); Key servK ∉ analz (spies evs);  
   A ∉ bad; B ∉ bad; evs ∈ kerbIV\_gets ]

7.10 2. Parties' strong authentication: non-injective agreement on the session key. The same guarantees also express

$\implies \text{Says } B \ A \ (\text{Crypt } \text{servK} \ (\text{Number } T3)) \in \text{set } \text{evs}$   
 $\langle \text{proof} \rangle$

**lemma** *shrK\_in\_initState\_Server*[iff]: "Key (shrK A)  $\in$  initState Kas"  
 $\langle \text{proof} \rangle$

**lemma** *shrK\_in\_knows\_Server* [iff]: "Key (shrK A)  $\in$  knows Kas evs"  
 $\langle \text{proof} \rangle$

**lemma** *A\_authenticates\_and\_keydist\_to\_Kas*:  
 "[[ Gets A (Crypt (shrK A) {Key authK, Peer, Ta, authTicket})  $\in$  set evs;  
 A  $\notin$  bad; evs  $\in$  kerbIV\_gets ]]  
 $\implies$  Says Kas A (Crypt (shrK A) {Key authK, Peer, Ta, authTicket})  $\in$  set  
 evs  
 $\wedge$  Key authK  $\in$  analz(knows Kas evs)"  
 $\langle \text{proof} \rangle$

**lemma** *K3\_imp\_Gets\_evs*:  
 "[[ Says A Tgs {Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta},  
 Crypt authK {Agent A, Number T2}, Agent B}  
 $\in$  set evs; A  $\notin$  bad; evs  $\in$  kerbIV\_gets ]]  
 $\implies$  Gets A (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta,  
 Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}})  
 $\in$  set evs"  
 $\langle \text{proof} \rangle$

**lemma** *Tgs\_authenticates\_and\_keydist\_to\_A*:  
 "[[ Gets Tgs {  
 Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta},  
 Crypt authK {Agent A, Number T2}, Agent B }  $\in$  set evs;  
 Key authK  $\notin$  analz (spies evs); A  $\notin$  bad; evs  $\in$  kerbIV\_gets ]]  
 $\implies \exists B. \text{Says } A \ Tgs \{$   
 Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta},  
 Crypt authK {Agent A, Number T2}, Agent B }  $\in$  set evs  
 $\wedge$  Key authK  $\in$  analz (knows A evs)"  
 $\langle \text{proof} \rangle$

**lemma** *K4\_imp\_Gets*:  
 "[[ Says Tgs A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})  
 $\in$  set evs; evs  $\in$  kerbIV\_gets ]]  
 $\implies \exists Ta \ X.$   
 Gets Tgs {Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta},  
 X}  
 $\in$  set evs"  
 $\langle \text{proof} \rangle$

**lemma** *A\_authenticates\_and\_keydist\_to\_Tgs*:  
 "[[ Gets A (Crypt (shrK A) {Key authK, Agent Tgs, Number Ta, authTicket})  
 $\in$  set evs;  
 Gets A (Crypt authK {Key servK, Agent B, Number Ts, servTicket})

$\in \text{set evs};$   
 $\text{Key authK} \notin \text{analz}(\text{spies evs}); A \notin \text{bad};$   
 $\text{evs} \in \text{kerbIV\_gets} \parallel$   
 $\implies \text{Says Tgs A (Crypt authK \{Key servK, Agent B, Number Ts, servTicket\})}$   
 $\in \text{set evs}$   
 $\wedge \text{Key authK} \in \text{analz}(\text{knows Tgs evs})$   
 $\wedge \text{Key servK} \in \text{analz}(\text{knows Tgs evs})"$   
 $\langle \text{proof} \rangle$

**lemma K5\_imp\_Gets:**

$"\parallel \text{Says A B \{servTicket, Crypt servK \{Agent A, Number T3\}\} \in \text{set evs};$   
 $A \notin \text{bad}; \text{evs} \in \text{kerbIV\_gets} \parallel$   
 $\implies \exists \text{authK Ts authTicket T2.}$   
 $\text{Gets A (Crypt authK \{Key servK, Agent B, Number Ts, servTicket\})} \in \text{set}$   
 $\text{evs}$   
 $\wedge \text{Says A Tgs \{authTicket, Crypt authK \{Agent A, Number T2\}, Agent B\} \in}$   
 $\text{set evs}"$   
 $\langle \text{proof} \rangle$

**lemma K3\_imp\_Gets:**

$"\parallel \text{Says A Tgs \{authTicket, Crypt authK \{Agent A, Number T2\}, Agent B\}}$   
 $\in \text{set evs};$   
 $A \notin \text{bad}; \text{evs} \in \text{kerbIV\_gets} \parallel$   
 $\implies \exists \text{Ta. Gets A (Crypt (shrK A) \{Key authK, Agent Tgs, Number Ta, authTicket\})}$   
 $\in \text{set evs}"$   
 $\langle \text{proof} \rangle$

**lemma B\_authenticates\_and\_keydist\_to\_A:**

$"\parallel \text{Gets B \{Crypt (shrK B) \{Agent A, Agent B, Key servK, Number Ts\},}$   
 $\text{Crypt servK \{Agent A, Number T3\}\} \in \text{set evs};$   
 $\text{Key servK} \notin \text{analz}(\text{spies evs});$   
 $A \notin \text{bad}; B \notin \text{bad}; B \neq \text{Tgs}; \text{evs} \in \text{kerbIV\_gets} \parallel$   
 $\implies \text{Says A B \{Crypt (shrK B) \{Agent A, Agent B, Key servK, Number Ts\},}$   
 $\text{Crypt servK \{Agent A, Number T3\}\} \in \text{set evs}$   
 $\wedge \text{Key servK} \in \text{analz}(\text{knows A evs})"$   
 $\langle \text{proof} \rangle$

**lemma K6\_imp\_Gets:**

$"\parallel \text{Says B A (Crypt servK (Number T3))} \in \text{set evs};$   
 $B \notin \text{bad}; \text{evs} \in \text{kerbIV\_gets} \parallel$   
 $\implies \exists \text{Ts X. Gets B \{Crypt (shrK B) \{Agent A, Agent B, Key servK, Number Ts\}, X\}}$   
 $\in \text{set evs}"$   
 $\langle \text{proof} \rangle$

**lemma A\_authenticates\_and\_keydist\_to\_B:**

$"\parallel \text{Gets A \{Crypt authK \{Key servK, Agent B, Number Ts, servTicket\},}$   
 $\text{Crypt servK (Number T3)\} \in \text{set evs};$   
 $\text{Gets A (Crypt (shrK A) \{Key authK, Agent Tgs, Number Ta, authTicket\})}$   
 $\in \text{set evs};$   
 $\text{Key authK} \notin \text{analz}(\text{spies evs}); \text{Key servK} \notin \text{analz}(\text{spies evs});$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{kerbIV\_gets} \parallel$   
 $\implies \text{Says B A (Crypt servK (Number T3))} \in \text{set evs}$



```

     $\wedge$  Key servK  $\in$  analz (knows B evs)"
  <proof>

```

```

end

```

## 8 The Kerberos Protocol, Version V

```

theory KerberosV imports Public begin

```

The "u" prefix indicates theorems referring to an updated version of the protocol. The "r" suffix indicates theorems where the confidentiality assumptions are relaxed by the corresponding arguments.

**abbreviation**

```

  Kas :: agent where
    "Kas == Server"

```

**abbreviation**

```

  Tgs :: agent where
    "Tgs == Friend 0"

```

**axiomatization where**

```

  Tgs_not_bad [iff]: "Tgs  $\notin$  bad"
  — Tgs is secure — we already know that Kas is secure

```

**definition**

```

  authKeys :: "event list  $\Rightarrow$  key set" where
    "authKeys evs = {authK.  $\exists$  A Peer Ta.
      Says Kas A {Crypt (shrK A) {Key authK, Agent Peer, Ta}},
      Crypt (shrK Peer) {Agent A, Agent Peer, Key authK, Ta}}
       $\in$  set evs}"

```

**definition**

```

  Issues :: "[agent, agent, msg, event list]  $\Rightarrow$  bool"
    (<_ Issues _ with _ on _>) where
    "A Issues B with X on evs =
      ( $\exists$  Y. Says A B Y  $\in$  set evs  $\wedge$  X  $\in$  parts {Y}  $\wedge$ 
        X  $\notin$  parts (spies (takeWhile ( $\lambda$ z. z  $\neq$  Says A B Y) (rev evs))))"
```

**consts**

```

  authKlife  :: nat

```

```

  servKlife  :: nat

```

```

  authlife   :: nat

```

```

    replylife    :: nat

specification (authKlife)
  authKlife_LB [iff]: "2 ≤ authKlife"
    ⟨proof⟩

specification (servKlife)
  servKlife_LB [iff]: "2 + authKlife ≤ servKlife"
    ⟨proof⟩

specification (authlife)
  authlife_LB [iff]: "Suc 0 ≤ authlife"
    ⟨proof⟩

specification (replylife)
  replylife_LB [iff]: "Suc 0 ≤ replylife"
    ⟨proof⟩

abbreviation

  CT :: "event list ⇒ nat" where
    "CT == length"

abbreviation
  expiredAK :: "[nat, event list] ⇒ bool" where
    "expiredAK T evs == authKlife + T < CT evs"

abbreviation
  expiredSK :: "[nat, event list] ⇒ bool" where
    "expiredSK T evs == servKlife + T < CT evs"

abbreviation
  expiredA :: "[nat, event list] ⇒ bool" where
    "expiredA T evs == authlife + T < CT evs"

abbreviation
  valid :: "[nat, nat] ⇒ bool"  (⟨valid _ wrt _⟩) where
    "valid T1 wrt T2 == T1 ≤ replylife + T2"


definition AKcryptSK :: "[key, key, event list] ⇒ bool" where
  "AKcryptSK authK servK evs ==
    ∃ A B tt.
      Says Tgs A {Crypt authK {Key servK, Agent B, tt}},
      Crypt (shrK B) {Agent A, Agent B, Key servK, tt}}
    ∈ set evs"

inductive_set kerbV :: "event list set"
  where

    Nil: "[] ∈ kerbV"

```

```

/ Fake: "[ evsf ∈ kerbV; X ∈ synth (analz (spies evsf)) ]
    ⇒ Says Spy B X # evsf ∈ kerbV"

/ KV1: "[ evs1 ∈ kerbV ]
    ⇒ Says A Kas {Agent A, Agent Tgs, Number (CT evs1)} # evs1
    ∈ kerbV"

/ KV2: "[ evs2 ∈ kerbV; Key authK ∉ used evs2; authK ∈ symKeys;
    Says A' Kas {Agent A, Agent Tgs, Number T1} ∈ set evs2 ]
    ⇒ Says Kas A {
    Crypt (shrK A) {Key authK, Agent Tgs, Number (CT evs2)},
    Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number (CT evs2)}
    } # evs2 ∈ kerbV"

/ KV3: "[ evs3 ∈ kerbV; A ≠ Kas; A ≠ Tgs;
    Says A Kas {Agent A, Agent Tgs, Number T1} ∈ set evs3;
    Says Kas' A {Crypt (shrK A) {Key authK, Agent Tgs, Number Ta},
    authTicket} ∈ set evs3;
    valid Ta wrt T1
    ]
    ⇒ Says A Tgs {authTicket,
    (Crypt authK {Agent A, Number (CT evs3)}),
    Agent B} # evs3 ∈ kerbV"

/ KV4: "[ evs4 ∈ kerbV; Key servK ∉ used evs4; servK ∈ symKeys;
    B ≠ Tgs; authK ∈ symKeys;
    Says A' Tgs {
    (Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK,
    Number Ta}),
    (Crypt authK {Agent A, Number T2}), Agent B}
    ∈ set evs4;
    ¬ expiredAK Ta evs4;
    ¬ expiredA T2 evs4;
    servKlife + (CT evs4) ≤ authKlife + Ta
    ]
    ⇒ Says Tgs A {
    Crypt authK {Key servK, Agent B, Number (CT evs4)},
    Crypt (shrK B) {Agent A, Agent B, Key servK, Number (CT evs4)}
    } # evs4 ∈ kerbV"

/ KV5: "[ evs5 ∈ kerbV; authK ∈ symKeys; servK ∈ symKeys;
    A ≠ Kas; A ≠ Tgs;
    Says A Tgs
    {authTicket, Crypt authK {Agent A, Number T2},
    Agent B}
    ∈ set evs5;

```

```

    Says Tgs' A {Crypt authK {Key servK, Agent B, Number Ts},
                  servTicket}
    ∈ set evs5;
    valid Ts wrt T2 ]
  ⇒ Says A B {servTicket,
               Crypt servK {Agent A, Number (CT evs5)}}
    # evs5 ∈ kerbV"

/ KV6: "[ evs6 ∈ kerbV; B ≠ Kas; B ≠ Tgs;
    Says A' B {
      (Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}),
      (Crypt servK {Agent A, Number T3})}
    ∈ set evs6;
    ¬ expiredSK Ts evs6;
    ¬ expiredA T3 evs6
  ]
  ⇒ Says B A (Crypt servK (Number Ta2))
    # evs6 ∈ kerbV"

/ Oops1: "[ evs01 ∈ kerbV; A ≠ Spy;
    Says Kas A {Crypt (shrK A) {Key authK, Agent Tgs, Number Ta},
                  authTicket} ∈ set evs01;
    expiredAK Ta evs01 ]
  ⇒ Notes Spy {Agent A, Agent Tgs, Number Ta, Key authK}
    # evs01 ∈ kerbV"

/ Oops2: "[ evs02 ∈ kerbV; A ≠ Spy;
    Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts},
                  servTicket} ∈ set evs02;
    expiredSK Ts evs02 ]
  ⇒ Notes Spy {Agent A, Agent B, Number Ts, Key servK}
    # evs02 ∈ kerbV"

```

```

declare Says_imp_knows_Spy [THEN parts.Inj, dest]
declare parts.Body [dest]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]

```

### 8.1 Lemmas about lists, for reasoning about Issues

```

lemma spies_Says_rev: "spies (evs @ [Says A B X]) = insert X (spies evs)"
<proof>

```

```

lemma spies_Gets_rev: "spies (evs @ [Gets A X]) = spies evs"
<proof>

```

```

lemma spies_Notes_rev: "spies (evs @ [Notes A X]) =
  (if A ∈ bad then insert X (spies evs) else spies evs)"

```

*<proof>*

**lemma** spies\_evs\_rev: "spies evs = spies (rev evs)"

*<proof>*

**lemmas** parts\_spies\_evs\_revD2 = spies\_evs\_rev [THEN equalityD2, THEN parts\_mono]

**lemma** spies\_takeWhile: "spies (takeWhile P evs)  $\subseteq$  spies evs"

*<proof>*

**lemmas** parts\_spies\_takeWhile\_mono = spies\_takeWhile [THEN parts\_mono]

## 8.2 Lemmas about authKeys

**lemma** authKeys\_empty: "authKeys [] = {}"

*<proof>*

**lemma** authKeys\_not\_insert:

" $(\forall A \text{ Ta akey Peer.}$

$\text{ev} \neq \text{Says Kas A } \{\{\text{Crypt (shrK A) } \{\{\text{akey, Agent Peer, Ta}\},$   
 $\text{Crypt (shrK Peer) } \{\{\text{Agent A, Agent Peer, akey, Ta}\}\}\}$

$\implies \text{authKeys (ev \# evs) = authKeys evs}$ "

*<proof>*

**lemma** authKeys\_insert:

"authKeys

$(\text{Says Kas A } \{\{\text{Crypt (shrK A) } \{\{\text{Key K, Agent Peer, Number Ta}\},$   
 $\text{Crypt (shrK Peer) } \{\{\text{Agent A, Agent Peer, Key K, Number Ta}\}\}\} \# \text{evs})$   
 $= \text{insert K (authKeys evs)}$ "

*<proof>*

**lemma** authKeys\_simp:

" $K \in \text{authKeys}$

$(\text{Says Kas A } \{\{\text{Crypt (shrK A) } \{\{\text{Key K'}, Agent Peer, Number Ta}\},$   
 $\text{Crypt (shrK Peer) } \{\{\text{Agent A, Agent Peer, Key K'}, Number Ta}\}\} \# \text{evs})$   
 $\implies K = K' \mid K \in \text{authKeys evs}$ "

*<proof>*

**lemma** authKeysI:

"Says Kas A  $\{\{\text{Crypt (shrK A) } \{\{\text{Key K, Agent Tgs, Number Ta}\},$   
 $\text{Crypt (shrK Tgs) } \{\{\text{Agent A, Agent Tgs, Key K, Number Ta}\}\}\} \in \text{set evs}$   
 $\implies K \in \text{authKeys evs}$ "

*<proof>*

**lemma** authKeys\_used: " $K \in \text{authKeys evs} \implies \text{Key K} \in \text{used evs}$ "

*<proof>*

## 8.3 Forwarding Lemmas

**lemma** Says\_ticket\_parts:

"Says S A  $\{\{\text{Crypt K } \{\{\text{SesKey, B, TimeStamp}\}, \text{Ticket}\}$   
 $\in \text{set evs} \implies \text{Ticket} \in \text{parts (spies evs)}$ "

*<proof>*

**lemma** Says\_ticket\_analz:

"Says S A {Crypt K {SesKey, B, TimeStamp}, Ticket}  
 $\in \text{set evs} \implies \text{Ticket} \in \text{analz} (\text{spies evs})"$

*<proof>*

**lemma** Ops\_range\_spies1:

"[ Says Kas A {Crypt KeyA {Key authK, Peer, Ta}, authTicket}  
 $\in \text{set evs}$  ;  
 $\text{evs} \in \text{kerbV}$  ]  $\implies \text{authK} \notin \text{range shrK} \wedge \text{authK} \in \text{symKeys}"$

*<proof>*

**lemma** Ops\_range\_spies2:

"[ Says Tgs A {Crypt authK {Key servK, Agent B, Ts}, servTicket}  
 $\in \text{set evs}$  ;  
 $\text{evs} \in \text{kerbV}$  ]  $\implies \text{servK} \notin \text{range shrK} \wedge \text{servK} \in \text{symKeys}"$

*<proof>*

**lemma** Spy\_see\_shrK [simp]:

" $\text{evs} \in \text{kerbV} \implies (\text{Key} (\text{shrK } A) \in \text{parts} (\text{spies evs})) = (A \in \text{bad})"$

*<proof>*

**lemma** Spy\_analz\_shrK [simp]:

" $\text{evs} \in \text{kerbV} \implies (\text{Key} (\text{shrK } A) \in \text{analz} (\text{spies evs})) = (A \in \text{bad})"$

*<proof>*

**lemma** Spy\_see\_shrK\_D [dest!]:

"[ Key (shrK A)  $\in \text{parts} (\text{spies evs})$ ;  $\text{evs} \in \text{kerbV}$  ]  $\implies A \in \text{bad}"$

*<proof>*

**lemmas** Spy\_analz\_shrK\_D = analz\_subset\_parts [THEN subsetD, THEN Spy\_see\_shrK\_D, dest!]

Nobody can have used non-existent keys!

**lemma** new\_keys\_not\_used [simp]:

"[Key K  $\notin \text{used evs}$ ;  $K \in \text{symKeys}$ ;  $\text{evs} \in \text{kerbV}$ ]  
 $\implies K \notin \text{keysFor} (\text{parts} (\text{spies evs}))"$

*<proof>*

**lemma** new\_keys\_not\_analz:

"[ $\text{evs} \in \text{kerbV}$ ;  $K \in \text{symKeys}$ ; Key K  $\notin \text{used evs}$ ]  
 $\implies K \notin \text{keysFor} (\text{analz} (\text{spies evs}))"$

*<proof>*

## 8.4 Regularity Lemmas

These concern the form of items passed in messages

Describes the form of all components sent by Kas

**lemma** Says\_Kas\_message\_form:

"[ Says Kas A {Crypt K {Key authK, Agent Peer, Ta}, authTicket}

```

    ∈ set evs;
    evs ∈ kerbV ]
    ⇒ authK ∉ range shrK ∧ authK ∈ authKeys evs ∧ authK ∈ symKeys ∧

    authTicket = (Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Ta}) ∧
    K = shrK A ∧ Peer = Tgs"
  <proof>

```

```

lemma SesKey_is_session_key:
  "[ Crypt (shrK Tgs_B) {Agent A, Agent Tgs_B, Key SesKey, Number T}
    ∈ parts (spies evs); Tgs_B ∉ bad;
    evs ∈ kerbV ]
  ⇒ SesKey ∉ range shrK"
  <proof>

```

```

lemma authTicket_authentic:
  "[ Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Ta}
    ∈ parts (spies evs);
    evs ∈ kerbV ]
  ⇒ Says Kas A {Crypt (shrK A) {Key authK, Agent Tgs, Ta},
    Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Ta}}
    ∈ set evs"
  <proof>

```

```

lemma authTicket_crypt_authK:
  "[ Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}
    ∈ parts (spies evs);
    evs ∈ kerbV ]
  ⇒ authK ∈ authKeys evs"
  <proof>

```

Describes the form of servK, servTicket and authK sent by Tgs

```

lemma Says_Tgs_message_form:
  "[ Says Tgs A {Crypt authK {Key servK, Agent B, Ts}, servTicket}
    ∈ set evs;
    evs ∈ kerbV ]
  ⇒ B ≠ Tgs ∧
    servK ∉ range shrK ∧ servK ∉ authKeys evs ∧ servK ∈ symKeys ∧
    servTicket = (Crypt (shrK B) {Agent A, Agent B, Key servK, Ts}) ∧
    authK ∉ range shrK ∧ authK ∈ authKeys evs ∧ authK ∈ symKeys"
  <proof>

```

## 8.5 Authenticity theorems: confirm origin of sensitive messages

```

lemma authK_authentic:
  "[ Crypt (shrK A) {Key authK, Peer, Ta}
    ∈ parts (spies evs);
    A ∉ bad; evs ∈ kerbV ]
  ⇒ ∃ AT. Says Kas A {Crypt (shrK A) {Key authK, Peer, Ta}, AT}
    ∈ set evs"

```

*<proof>*

If a certain encrypted message appears then it originated with Tgs

**lemma** *servK\_authentic*:

```
"[[ Crypt authK {Key servK, Agent B, Ts}
   ∈ parts (spies evs);
   Key authK ∉ analz (spies evs);
   authK ∉ range shrK;
   evs ∈ kerbV ]]
⇒ ∃ A ST. Says Tgs A {Crypt authK {Key servK, Agent B, Ts}, ST}
   ∈ set evs"
```

*<proof>*

**lemma** *servK\_authentic\_bis*:

```
"[[ Crypt authK {Key servK, Agent B, Ts}
   ∈ parts (spies evs);
   Key authK ∉ analz (spies evs);
   B ≠ Tgs;
   evs ∈ kerbV ]]
⇒ ∃ A ST. Says Tgs A {Crypt authK {Key servK, Agent B, Ts}, ST}
   ∈ set evs"
```

*<proof>*

Authenticity of servK for B

**lemma** *servTicket\_authentic\_Tgs*:

```
"[[ Crypt (shrK B) {Agent A, Agent B, Key servK, Ts}
   ∈ parts (spies evs); B ≠ Tgs; B ∉ bad;
   evs ∈ kerbV ]]
⇒ ∃ authK.
   Says Tgs A {Crypt authK {Key servK, Agent B, Ts},
               Crypt (shrK B) {Agent A, Agent B, Key servK, Ts}}
   ∈ set evs"
```

*<proof>*

Anticipated here from next subsection

**lemma** *K4\_imp\_K2*:

```
"[[ Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts}, servTicket}
   ∈ set evs; evs ∈ kerbV ]]
⇒ ∃ Ta. Says Kas A
   {Crypt (shrK A) {Key authK, Agent Tgs, Number Ta},
    Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}}
   ∈ set evs"
```

*<proof>*

Anticipated here from next subsection

**lemma** *u\_K4\_imp\_K2*:

```
"[[ Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts}, servTicket} ∈
set evs; evs ∈ kerbV ]]
⇒ ∃ Ta. Says Kas A {Crypt (shrK A) {Key authK, Agent Tgs, Number Ta},
                    Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}} ∈
set evs
   ∧ servKlife + Ts ≤ authKlife + Ta"
```

*<proof>*



**lemma** *servTicket\_authentic\_Kas*:

```
"[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
  ∈ parts (spies evs); B ≠ Tgs; B ∉ bad;
  evs ∈ kerbV ]
⇒ ∃ authK Ta.
  Says Kas A
    {Crypt (shrK A) {Key authK, Agent Tgs, Number Ta},
     Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}}
  ∈ set evs"
⟨proof⟩
```

**lemma** *u\_servTicket\_authentic\_Kas*:

```
"[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
  ∈ parts (spies evs); B ≠ Tgs; B ∉ bad;
  evs ∈ kerbV ]
⇒ ∃ authK Ta.
  Says Kas A
    {Crypt (shrK A) {Key authK, Agent Tgs, Number Ta},
     Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}}
  ∈ set evs ∧
  servKlife + Ts ≤ authKlife + Ta"
⟨proof⟩
```

**lemma** *servTicket\_authentic*:

```
"[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
  ∈ parts (spies evs); B ≠ Tgs; B ∉ bad;
  evs ∈ kerbV ]
⇒ ∃ Ta authK.
  Says Kas A {Crypt (shrK A) {Key authK, Agent Tgs, Number Ta},
              Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number
Ta}} ∈ set evs
  ∧ Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts},
                Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}}
  ∈ set evs"
⟨proof⟩
```

**lemma** *u\_servTicket\_authentic*:

```
"[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
  ∈ parts (spies evs); B ≠ Tgs; B ∉ bad;
  evs ∈ kerbV ]
⇒ ∃ Ta authK.
  Says Kas A {Crypt (shrK A) {Key authK, Agent Tgs, Number Ta},
              Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number
Ta}} ∈ set evs
  ∧ Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts},
                Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}}
  ∈ set evs
  ∧ servKlife + Ts ≤ authKlife + Ta"
⟨proof⟩
```

**lemma** *u\_NotexpiredSK\_NotexpiredAK*:

```
"[ ¬ expiredSK Ts evs; servKlife + Ts ≤ authKlife + Ta ]
⇒ ¬ expiredAK Ta evs"
```

*<proof>*

## 8.6 Reliability: friendly agents send something if something else happened

**lemma** *K3\_imp\_K2:*

```
"[[ Says A Tgs
    {authTicket, Crypt authK {Agent A, Number T2}}, Agent B}
   ∈ set evs;
   A ∉ bad; evs ∈ kerbV ]]
⇒ ∃ Ta AT. Says Kas A {Crypt (shrK A) {Key authK, Agent Tgs, Ta}},
                                     AT} ∈ set evs"
```

*<proof>*

Anticipated here from next subsection. An authK is encrypted by one and only one Shared key. A servK is encrypted by one and only one authK.

**lemma** *Key\_unique\_SesKey:*

```
"[[ Crypt K {Key SesKey, Agent B, T}
   ∈ parts (spies evs);
   Crypt K' {Key SesKey, Agent B', T'}
   ∈ parts (spies evs); Key SesKey ∉ analz (spies evs);
   evs ∈ kerbV ]]
⇒ K=K' ∧ B=B' ∧ T=T'"
```

*<proof>*

This inevitably has an existential form in version V

**lemma** *Says\_K5:*

```
"[[ Crypt servK {Agent A, Number T3} ∈ parts (spies evs);
   Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts},
               servTicket} ∈ set evs;
   Key servK ∉ analz (spies evs);
   A ∉ bad; B ∉ bad; evs ∈ kerbV ]]
⇒ ∃ ST. Says A B {ST, Crypt servK {Agent A, Number T3}} ∈ set evs"
```

*<proof>*

Anticipated here from next subsection

**lemma** *unique\_CryptKey:*

```
"[[ Crypt (shrK B) {Agent A, Agent B, Key SesKey, T}
   ∈ parts (spies evs);
   Crypt (shrK B') {Agent A', Agent B', Key SesKey, T'}
   ∈ parts (spies evs); Key SesKey ∉ analz (spies evs);
   evs ∈ kerbV ]]
⇒ A=A' ∧ B=B' ∧ T=T'"
```

*<proof>*

**lemma** *Says\_K6:*

```
"[[ Crypt servK (Number T3) ∈ parts (spies evs);
   Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts},
               servTicket} ∈ set evs;
   Key servK ∉ analz (spies evs);
   A ∉ bad; B ∉ bad; evs ∈ kerbV ]]
⇒ Says B A (Crypt servK (Number T3)) ∈ set evs"
```

*<proof>*

Needs a unicity theorem, hence moved here

**lemma** *servK\_authentic\_ter*:

```
"[[ Says Kas A
  {Crypt (shrK A) {Key authK, Agent Tgs, Ta}}, authTicket} ∈ set evs;
  Crypt authK {Key servK, Agent B, Ts}
  ∈ parts (spies evs);
  Key authK ∉ analz (spies evs);
  evs ∈ kerbV ] ]
⇒ Says Tgs A {Crypt authK {Key servK, Agent B, Ts},
  Crypt (shrK B) {Agent A, Agent B, Key servK, Ts} }
  ∈ set evs"
```

*<proof>*

## 8.7 Unicity Theorems

The session key, if secure, uniquely identifies the Ticket whether authTicket or servTicket. As a matter of fact, one can read also Tgs in the place of B.

**lemma** *unique\_authKeys*:

```
"[[ Says Kas A
  {Crypt Ka {Key authK, Agent Tgs, Ta}}, X} ∈ set evs;
  Says Kas A'
  {Crypt Ka' {Key authK, Agent Tgs, Ta'}}, X'} ∈ set evs;
  evs ∈ kerbV ] ] ⇒ A=A' ∧ Ka=Ka' ∧ Ta=Ta' ∧ X=X'"
```

*<proof>*

servK uniquely identifies the message from Tgs

**lemma** *unique\_servKeys*:

```
"[[ Says Tgs A
  {Crypt K {Key servK, Agent B, Ts}}, X} ∈ set evs;
  Says Tgs A'
  {Crypt K' {Key servK, Agent B', Ts'}}, X'} ∈ set evs;
  evs ∈ kerbV ] ] ⇒ A=A' ∧ B=B' ∧ K=K' ∧ Ts=Ts' ∧ X=X'"
```

*<proof>*

## 8.8 Lemmas About the Predicate *AKcryptSK*

**lemma** *not\_AKcryptSK\_Nil [iff]*: " $\neg$  *AKcryptSK* authK servK []"

*<proof>*

**lemma** *AKcryptSKI*:

```
"[[ Says Tgs A {Crypt authK {Key servK, Agent B, tt}}, X } ∈ set evs;
  evs ∈ kerbV ] ] ⇒ AKcryptSK authK servK evs"
```

*<proof>*

**lemma** *AKcryptSK\_Says [simp]*:

```
"AKcryptSK authK servK (Says S A X # evs) =
  (S = Tgs ∧
   (∃ B tt. X = {Crypt authK {Key servK, Agent B, tt},
    Crypt (shrK B) {Agent A, Agent B, Key servK, tt} }
    | AKcryptSK authK servK evs)"
```

*<proof>*

**lemma** *AKcryptSK\_Notes [simp]:*  
 "AKcryptSK authK servK (Notes A X # evs) =  
 AKcryptSK authK servK evs"  
 <proof>

**lemma** *Auth\_fresh\_not\_AKcryptSK:*  
 "[[ Key authK  $\notin$  used evs; evs  $\in$  kerbV ]]  
 $\implies \neg$  AKcryptSK authK servK evs"  
 <proof>

**lemma** *Serv\_fresh\_not\_AKcryptSK:*  
 "Key servK  $\notin$  used evs  $\implies \neg$  AKcryptSK authK servK evs"  
 <proof>

**lemma** *authK\_not\_AKcryptSK:*  
 "[[ Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, tk}  
 $\in$  parts (spies evs); evs  $\in$  kerbV ]]  
 $\implies \neg$  AKcryptSK K authK evs"  
 <proof>

A secure serverkey cannot have been used to encrypt others

**lemma** *servK\_not\_AKcryptSK:*  
 "[[ Crypt (shrK B) {Agent A, Agent B, Key SK, tt}  $\in$  parts (spies evs);  
 Key SK  $\notin$  analz (spies evs); SK  $\in$  symKeys;  
 B  $\neq$  Tgs; evs  $\in$  kerbV ]]  
 $\implies \neg$  AKcryptSK SK K evs"  
 <proof>

Long term keys are not issued as servKeys

**lemma** *shrK\_not\_AKcryptSK:*  
 "evs  $\in$  kerbV  $\implies \neg$  AKcryptSK K (shrK A) evs"  
 <proof>

The Tgs message associates servK with authK and therefore not with any other key authK.

**lemma** *Says\_Tgs\_AKcryptSK:*  
 "[[ Says Tgs A {Crypt authK {Key servK, Agent B, tt}, X}  
 $\in$  set evs;  
 authK'  $\neq$  authK; evs  $\in$  kerbV ]]  
 $\implies \neg$  AKcryptSK authK' servK evs"  
 <proof>

**lemma** *AKcryptSK\_not\_AKcryptSK:*  
 "[[ AKcryptSK authK servK evs; evs  $\in$  kerbV ]]  
 $\implies \neg$  AKcryptSK servK K evs"  
 <proof>

**lemma** *not\_different\_AKcryptSK:*  
 "[[ AKcryptSK authK servK evs;  
 authK'  $\neq$  authK; evs  $\in$  kerbV ]]

$\implies \neg \text{AKcryptSK } \text{authK}' \text{ servK } \text{evs} \wedge \text{servK} \in \text{symKeys}$   
 $\langle \text{proof} \rangle$

The only session keys that can be found with the help of session keys are those sent by Tgs in step K4.

We take some pains to express the property as a logical equivalence so that the simplifier can apply it.

**lemma** *Key\_analz\_image\_Key\_lemma*:  
 $"P \longrightarrow (\text{Key } K \in \text{analz } (\text{Key}'KK \cup H)) \longrightarrow (K \in KK \vee \text{Key } K \in \text{analz } H)$   
 $\implies$   
 $P \longrightarrow (\text{Key } K \in \text{analz } (\text{Key}'KK \cup H)) = (K \in KK \vee \text{Key } K \in \text{analz } H)"$   
 $\langle \text{proof} \rangle$

**lemma** *AKcryptSK\_analz\_insert*:  
 $"[ \text{AKcryptSK } K \text{ K}' \text{ evs}; K \in \text{symKeys}; \text{evs} \in \text{kerbV} ]$   
 $\implies \text{Key } K' \in \text{analz } (\text{insert } (\text{Key } K) (\text{spies evs}))"$   
 $\langle \text{proof} \rangle$

**lemma** *authKeys\_are\_not\_AKcryptSK*:  
 $"[ K \in \text{authKeys evs} \cup \text{range shrK}; \text{evs} \in \text{kerbV} ]$   
 $\implies \forall SK. \neg \text{AKcryptSK } SK \text{ K evs} \wedge K \in \text{symKeys}"$   
 $\langle \text{proof} \rangle$

**lemma** *not\_authKeys\_not\_AKcryptSK*:  
 $"[ K \notin \text{authKeys evs};$   
 $K \notin \text{range shrK}; \text{evs} \in \text{kerbV} ]$   
 $\implies \forall SK. \neg \text{AKcryptSK } K \text{ SK evs}"$   
 $\langle \text{proof} \rangle$

## 8.9 Secrecy Theorems

For the OOps2 case of the next theorem

**lemma** *Ops2\_not\_AKcryptSK*:  
 $"[ \text{evs} \in \text{kerbV};$   
 $\text{Says Tgs A } \{ \text{Crypt authK}$   
 $\{ \text{Key servK, Agent B, Number Ts} \}, \text{servTicket} \}$   
 $\in \text{set evs} ]$   
 $\implies \neg \text{AKcryptSK servK SK evs}"$   
 $\langle \text{proof} \rangle$

Big simplification law for keys SK that are not crypted by keys in KK It helps prove three, otherwise hard, facts about keys. These facts are exploited as simplification laws for analz, and also "limit the damage" in case of loss of a key to the spy. See ESORICS98.

**lemma** *Key\_analz\_image\_Key [rule\_format (no\_asm)]*:  
 $"\text{evs} \in \text{kerbV} \implies$   
 $(\forall SK \text{ KK}. SK \in \text{symKeys} \wedge KK \subseteq \neg(\text{range shrK}) \longrightarrow$   
 $(\forall K \in KK. \neg \text{AKcryptSK } K \text{ SK evs}) \longrightarrow$   
 $(\text{Key } SK \in \text{analz } (\text{Key}'KK \cup (\text{spies evs}))) =$   
 $(SK \in KK \mid \text{Key } SK \in \text{analz } (\text{spies evs}))"$   
 $\langle \text{proof} \rangle$

First simplification law for *analz*: no session keys encrypt authentication keys or shared keys.

```

lemma analz_insert_freshK1:
  "[[ evs ∈ kerbV; K ∈ authKeys evs ∪ range shrK;
    SesKey ∉ range shrK ]]"
  ⇒ (Key K ∈ analz (insert (Key SesKey) (spies evs))) =
    (K = SesKey | Key K ∈ analz (spies evs))"
<proof>

```

Second simplification law for *analz*: no service keys encrypt any other keys.

```

lemma analz_insert_freshK2:
  "[[ evs ∈ kerbV; servK ∉ (authKeys evs); servK ∉ range shrK;
    K ∈ symKeys ]]"
  ⇒ (Key K ∈ analz (insert (Key servK) (spies evs))) =
    (K = servK | Key K ∈ analz (spies evs))"
<proof>

```

Third simplification law for *analz*: only one authentication key encrypts a certain service key.

```

lemma analz_insert_freshK3:
  "[[ AKcryptSK authK servK evs;
    authK' ≠ authK; authK' ∉ range shrK; evs ∈ kerbV ]]"
  ⇒ (Key servK ∈ analz (insert (Key authK') (spies evs))) =
    (servK = authK' | Key servK ∈ analz (spies evs))"
<proof>

```

```

lemma analz_insert_freshK3_bis:
  "[[ Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts}}, servTicket}
    ∈ set evs;
    authK ≠ authK'; authK' ∉ range shrK; evs ∈ kerbV ]]"
  ⇒ (Key servK ∈ analz (insert (Key authK') (spies evs))) =
    (servK = authK' | Key servK ∈ analz (spies evs))"
<proof>

```

a weakness of the protocol

```

lemma authK_compromises_servK:
  "[[ Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts}}, servTicket}
    ∈ set evs; authK ∈ symKeys;
    Key authK ∈ analz (spies evs); evs ∈ kerbV ]]"
  ⇒ Key servK ∈ analz (spies evs)"
<proof>

```

*lemma servK\_notin\_authKeysD* not needed in version V

If Spy sees the Authentication Key sent in msg K2, then the Key has expired.

```

lemma Confidentiality_Kas_lemma [rule_format]:
  "[[ authK ∈ symKeys; A ∉ bad; evs ∈ kerbV ]]"
  ⇒ Says Kas A
    {Crypt (shrK A) {Key authK, Agent Tgs, Number Ta}},
    Crypt (shrK Tgs) {Agent A, Agent Tgs, Key authK, Number Ta}}
    ∈ set evs →
    Key authK ∈ analz (spies evs) →

```

```

    expiredAK Ta evs"
<proof>

lemma Confidentiality_Kas:
  "[ Says Kas A
    {Crypt Ka {Key authK, Agent Tgs, Number Ta}}, authTicket}
    ∈ set evs;
    ¬ expiredAK Ta evs;
    A ∉ bad; evs ∈ kerbV ]
  ⇒ Key authK ∉ analz (spies evs)"
<proof>

```

If Spy sees the Service Key sent in msg K4, then the Key has expired.

```

lemma Confidentiality_lemma [rule_format]:
  "[ Says Tgs A
    {Crypt authK {Key servK, Agent B, Number Ts}},
    Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}}
    ∈ set evs;
    Key authK ∉ analz (spies evs);
    servK ∈ symKeys;
    A ∉ bad; B ∉ bad; evs ∈ kerbV ]
  ⇒ Key servK ∈ analz (spies evs) →
    expiredSK Ts evs"
<proof>

```

In the real world Tgs can't check wheter authK is secure!

```

lemma Confidentiality_Tgs:
  "[ Says Tgs A
    {Crypt authK {Key servK, Agent B, Number Ts}}, servTicket}
    ∈ set evs;
    Key authK ∉ analz (spies evs);
    ¬ expiredSK Ts evs;
    A ∉ bad; B ∉ bad; evs ∈ kerbV ]
  ⇒ Key servK ∉ analz (spies evs)"
<proof>

```

In the real world Tgs CAN check what Kas sends!

```

lemma Confidentiality_Tgs_bis:
  "[ Says Kas A
    {Crypt Ka {Key authK, Agent Tgs, Number Ta}}, authTicket}
    ∈ set evs;
    Says Tgs A
    {Crypt authK {Key servK, Agent B, Number Ts}}, servTicket}
    ∈ set evs;
    ¬ expiredAK Ta evs; ¬ expiredSK Ts evs;
    A ∉ bad; B ∉ bad; evs ∈ kerbV ]
  ⇒ Key servK ∉ analz (spies evs)"
<proof>

```

Most general form

```

lemmas Confidentiality_Tgs_ter = authTicket_authentic [THEN Confidentiality_Tgs_bis]

```

```

lemmas Confidentiality_Auth_A = authK_authentic [THEN exE, THEN Confidentiality_Kas]

```

Needs a confidentiality guarantee, hence moved here. Authenticity of servK for A

```

lemma servK_authentic_bis_r:
  "[ Crypt (shrK A) {Key authK, Agent Tgs, Number Ta}
    ∈ parts (spies evs);
    Crypt authK {Key servK, Agent B, Number Ts}
    ∈ parts (spies evs);
    ¬ expiredAK Ta evs; A ∉ bad; evs ∈ kerbV ]
  ⇒ Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts},
    Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts} }
    ∈ set evs"
<proof>

```

```

lemma Confidentiality_Serv_A:
  "[ Crypt (shrK A) {Key authK, Agent Tgs, Number Ta}
    ∈ parts (spies evs);
    Crypt authK {Key servK, Agent B, Number Ts}
    ∈ parts (spies evs);
    ¬ expiredAK Ta evs; ¬ expiredSK Ts evs;
    A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbV ]
  ⇒ Key servK ∉ analz (spies evs)"
<proof>

```

```

lemma Confidentiality_B:
  "[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
    ∈ parts (spies evs);
    Crypt authK {Key servK, Agent B, Number Ts}
    ∈ parts (spies evs);
    Crypt (shrK A) {Key authK, Agent Tgs, Number Ta}
    ∈ parts (spies evs);
    ¬ expiredSK Ts evs; ¬ expiredAK Ta evs;
    A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbV ]
  ⇒ Key servK ∉ analz (spies evs)"
<proof>

```

```

lemma u_Confidentiality_B:
  "[ Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
    ∈ parts (spies evs);
    ¬ expiredSK Ts evs;
    A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbV ]
  ⇒ Key servK ∉ analz (spies evs)"
<proof>

```

## 8.10 Authentication

Each party verifies "the identity of another party who generated some data" (quoted from Neuman and Ts'o).

These guarantees don't assess whether two parties agree on the same session key: sending a message containing a key doesn't a priori state knowledge of the key.

These didn't have existential form in version IV



**lemma** *B\_authenticates\_A*:

```
"[ Crypt servK {Agent A, Number T3} ∈ parts (spies evs);
  Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
    ∈ parts (spies evs);
  Key servK ∉ analz (spies evs);
  A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbV ]
⇒ ∃ ST. Says A B {ST, Crypt servK {Agent A, Number T3}} ∈ set evs"
⟨proof⟩
```

The second assumption tells B what kind of key servK is.

**lemma** *B\_authenticates\_A\_r*:

```
"[ Crypt servK {Agent A, Number T3} ∈ parts (spies evs);
  Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
    ∈ parts (spies evs);
  Crypt authK {Key servK, Agent B, Number Ts}
    ∈ parts (spies evs);
  Crypt (shrK A) {Key authK, Agent Tgs, Number Ta}
    ∈ parts (spies evs);
  ¬ expiredSK Ts evs; ¬ expiredAK Ta evs;
  B ≠ Tgs; A ∉ bad; B ∉ bad; evs ∈ kerbV ]
⇒ ∃ ST. Says A B {ST, Crypt servK {Agent A, Number T3}} ∈ set evs"
⟨proof⟩
```

*u\_B\_authenticates\_A* would be the same as *B\_authenticates\_A* because the servK confidentiality assumption is yet unrelaxed

**lemma** *u\_B\_authenticates\_A\_r*:

```
"[ Crypt servK {Agent A, Number T3} ∈ parts (spies evs);
  Crypt (shrK B) {Agent A, Agent B, Key servK, Number Ts}
    ∈ parts (spies evs);
  ¬ expiredSK Ts evs;
  B ≠ Tgs; A ∉ bad; B ∉ bad; evs ∈ kerbV ]
⇒ ∃ ST. Says A B {ST, Crypt servK {Agent A, Number T3}} ∈ set evs"
⟨proof⟩
```

**lemma** *A\_authenticates\_B*:

```
"[ Crypt servK (Number T3) ∈ parts (spies evs);
  Crypt authK {Key servK, Agent B, Number Ts}
    ∈ parts (spies evs);
  Crypt (shrK A) {Key authK, Agent Tgs, Number Ta}
    ∈ parts (spies evs);
  Key authK ∉ analz (spies evs); Key servK ∉ analz (spies evs);
  A ∉ bad; B ∉ bad; evs ∈ kerbV ]
⇒ Says B A (Crypt servK (Number T3)) ∈ set evs"
⟨proof⟩
```

**lemma** *A\_authenticates\_B\_r*:

```
"[ Crypt servK (Number T3) ∈ parts (spies evs);
  Crypt authK {Key servK, Agent B, Number Ts}
    ∈ parts (spies evs);
  Crypt (shrK A) {Key authK, Agent Tgs, Number Ta}
    ∈ parts (spies evs);
  ¬ expiredAK Ta evs; ¬ expiredSK Ts evs;
  A ∉ bad; B ∉ bad; evs ∈ kerbV ]
⇒ Says B A (Crypt servK (Number T3)) ∈ set evs"
```

*<proof>*

### 8.11 Parties' knowledge of session keys

An agent knows a session key if he used it to issue a cipher. These guarantees can be interpreted both in terms of key distribution and of non-injective agreement on the session key.

**lemma** *Kas\_Issues\_A:*

"[ Says Kas A {Crypt (shrK A) {Key authK, Peer, Ta}}, authTicket} ∈ set evs;

evs ∈ kerbV ]

⇒ Kas Issues A with (Crypt (shrK A) {Key authK, Peer, Ta})  
on evs"

*<proof>*

**lemma** *A\_authenticates\_and\_keydist\_to\_Kas:*

"[ Crypt (shrK A) {Key authK, Peer, Ta} ∈ parts (spies evs);

A ∉ bad; evs ∈ kerbV ]

⇒ Kas Issues A with (Crypt (shrK A) {Key authK, Peer, Ta})  
on evs"

*<proof>*

**lemma** *Tgs\_Issues\_A:*

"[ Says Tgs A {Crypt authK {Key servK, Agent B, Number Ts}}, servTicket}  
∈ set evs;

Key authK ∉ analz (spies evs); evs ∈ kerbV ]

⇒ Tgs Issues A with  
(Crypt authK {Key servK, Agent B, Number Ts}) on evs"

*<proof>*

**lemma** *A\_authenticates\_and\_keydist\_to\_Tgs:*

"[ Crypt authK {Key servK, Agent B, Number Ts}  
∈ parts (spies evs);

Key authK ∉ analz (spies evs); B ≠ Tgs; evs ∈ kerbV ]

⇒ ∃ A. Tgs Issues A with  
(Crypt authK {Key servK, Agent B, Number Ts}) on evs"

*<proof>*

**lemma** *B\_Issues\_A:*

"[ Says B A (Crypt servK (Number T3)) ∈ set evs;

Key servK ∉ analz (spies evs);

A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbV ]

⇒ B Issues A with (Crypt servK (Number T3)) on evs"

*<proof>*

**lemma** *A\_authenticates\_and\_keydist\_to\_B:*

"[ Crypt servK (Number T3) ∈ parts (spies evs);

Crypt authK {Key servK, Agent B, Number Ts}

∈ parts (spies evs);

Crypt (shrK A) {Key authK, Agent Tgs, Number Ta}

∈ parts (spies evs);

Key authK ∉ analz (spies evs); Key servK ∉ analz (spies evs);

A ∉ bad; B ∉ bad; B ≠ Tgs; evs ∈ kerbV ]

$\implies B \text{ Issues } A \text{ with } (\text{Crypt servK } (\text{Number } T3)) \text{ on evs}$   
 <proof>

But can prove a less general fact concerning only authenticators!

**lemma** *honest\_never\_says\_newer\_timestamp\_in\_auth*:  
 "[ (CT evs)  $\leq T$ ; Number  $T \in \text{parts } \{X\}$ ;  $A \notin \text{bad}$ ;  $\text{evs} \in \text{kerbV}$  ]  
 $\implies \text{Says } A \ B \ \{\{Y, X\}\} \notin \text{set evs}$   
 <proof>

**lemma** *honest\_never\_says\_current\_timestamp\_in\_auth*:  
 "[ (CT evs) =  $T$ ; Number  $T \in \text{parts } \{X\}$ ;  $A \notin \text{bad}$ ;  $\text{evs} \in \text{kerbV}$  ]  
 $\implies \text{Says } A \ B \ \{\{Y, X\}\} \notin \text{set evs}$   
 <proof>

**lemma** *A\_Issues\_B*:  
 "[ Says  $A \ B \ \{\{ST, \text{Crypt servK } \{\{Agent \ A, \text{Number } T3\}\}\} \in \text{set evs}$ ;  
 Key  $\text{servK} \notin \text{analz } (\text{spies evs})$ ;  
 $B \neq Tgs$ ;  $A \notin \text{bad}$ ;  $B \notin \text{bad}$ ;  $\text{evs} \in \text{kerbV}$  ]  
 $\implies A \text{ Issues } B \text{ with } (\text{Crypt servK } \{\{Agent \ A, \text{Number } T3\}\}) \text{ on evs}$   
 <proof>

**lemma** *B\_authenticates\_and\_keydist\_to\_A*:  
 "[ Crypt  $\text{servK } \{\{Agent \ A, \text{Number } T3\}\} \in \text{parts } (\text{spies evs})$ ;  
 Crypt  $(\text{shrK } B) \ \{\{Agent \ A, \text{Agent } B, \text{Key servK, Number } Ts\}\}$   
 $\in \text{parts } (\text{spies evs})$ ;  
 Key  $\text{servK} \notin \text{analz } (\text{spies evs})$ ;  
 $B \neq Tgs$ ;  $A \notin \text{bad}$ ;  $B \notin \text{bad}$ ;  $\text{evs} \in \text{kerbV}$  ]  
 $\implies A \text{ Issues } B \text{ with } (\text{Crypt servK } \{\{Agent \ A, \text{Number } T3\}\}) \text{ on evs}$   
 <proof>

## 8.12 Novel guarantees, never studied before

Because honest agents always say the right timestamp in authenticators, we can prove unicity guarantees based exactly on timestamps. Classical unicity guarantees are based on nonces. Of course assuming the agent to be different from the Spy, rather than not in bad, would suffice below. Similar guarantees must also hold of Kerberos IV.

Notice that an honest agent can send the same timestamp on two different traces of the same length, but not on the same trace!

**lemma** *unique\_timestamp\_authenticator1*:  
 "[ Says  $A \ Kas \ \{\{Agent \ A, \text{Agent } Tgs, \text{Number } T1\}\} \in \text{set evs}$ ;  
 Says  $A \ Kas' \ \{\{Agent \ A, \text{Agent } Tgs', \text{Number } T1\}\} \in \text{set evs}$ ;  
 $A \notin \text{bad}$ ;  $\text{evs} \in \text{kerbV}$  ]  
 $\implies Kas=Kas' \wedge Tgs=Tgs'$   
 <proof>

**lemma** *unique\_timestamp\_authenticator2*:  
 "[ Says  $A \ Tgs \ \{\{AT, \text{Crypt } AK \ \{\{Agent \ A, \text{Number } T2\}\}, \text{Agent } B\}\} \in \text{set evs}$ ;  
 Says  $A \ Tgs' \ \{\{AT', \text{Crypt } AK' \ \{\{Agent \ A, \text{Number } T2\}\}, \text{Agent } B'\}\} \in \text{set evs}$ ;  
 $A \notin \text{bad}$ ;  $\text{evs} \in \text{kerbV}$  ]  
 $\implies Tgs=Tgs' \wedge AT=AT' \wedge AK=AK' \wedge B=B'$

$\langle proof \rangle$

**lemma** *unique\_timestamp\_authenticator3*:

"[ Says A B {ST, Crypt SK {Agent A, Number T}} ∈ set evs;  
 Says A B' {ST', Crypt SK' {Agent A, Number T}} ∈ set evs;  
 A ∉ bad; evs ∈ kerbV ]

⇒ B=B' ∧ ST=ST' ∧ SK=SK' "

$\langle proof \rangle$

The second part of the message is treated as an authenticator by the last simplification step, even if it is not an authenticator!

**lemma** *unique\_timestamp\_authticket*:

"[ Says Kas A {X, Crypt (shrK Tgs) {Agent A, Agent Tgs, Key AK, T}} ∈ set evs;

Says Kas A' {X', Crypt (shrK Tgs') {Agent A', Agent Tgs', Key AK', T}} ∈ set evs;

evs ∈ kerbV ]

⇒ A=A' ∧ X=X' ∧ Tgs=Tgs' ∧ AK=AK' "

$\langle proof \rangle$

The second part of the message is treated as an authenticator by the last simplification step, even if it is not an authenticator!

**lemma** *unique\_timestamp\_servticket*:

"[ Says Tgs A {X, Crypt (shrK B) {Agent A, Agent B, Key SK, T}} ∈ set evs;

Says Tgs A' {X', Crypt (shrK B') {Agent A', Agent B', Key SK', T}} ∈ set evs;

evs ∈ kerbV ]

⇒ A=A' ∧ X=X' ∧ B=B' ∧ SK=SK' "

$\langle proof \rangle$

**lemma** *Kas\_never\_says\_newer\_timestamp*:

"[ (CT evs) ≤ T; Number T ∈ parts {X}; evs ∈ kerbV ]

⇒ ∀ A. Says Kas A X ∉ set evs "

$\langle proof \rangle$

**lemma** *Kas\_never\_says\_current\_timestamp*:

"[ (CT evs) = T; Number T ∈ parts {X}; evs ∈ kerbV ]

⇒ ∀ A. Says Kas A X ∉ set evs "

$\langle proof \rangle$

**lemma** *unique\_timestamp\_msg2*:

"[ Says Kas A {Crypt (shrK A) {Key AK, Agent Tgs, T}, AT} ∈ set evs;

Says Kas A' {Crypt (shrK A') {Key AK', Agent Tgs', T}, AT'} ∈ set evs;

evs ∈ kerbV ]

⇒ A=A' ∧ AK=AK' ∧ Tgs=Tgs' ∧ AT=AT' "

$\langle proof \rangle$

**lemma** *Tgs\_never\_says\_newer\_timestamp*:

"[ (CT evs) ≤ T; Number T ∈ parts {X}; evs ∈ kerbV ]

⇒ ∀ A. Says Tgs A X ∉ set evs "

$\langle proof \rangle$

```

lemma Tgs_never_says_current_timestamp:
  "[ (CT evs) = T; Number T ∈ parts {X}; evs ∈ kerbV ]
  ⇒ ∀ A. Says Tgs A X ∉ set evs"
  <proof>

lemma unique_timestamp_msg4:
  "[ Says Tgs A {Crypt (shrK A) {Key SK, Agent B, T}, ST} ∈ set evs;
    Says Tgs A' {Crypt (shrK A') {Key SK', Agent B', T}, ST'} ∈ set evs;
    evs ∈ kerbV ]
  ⇒ A=A' ∧ SK=SK' ∧ B=B' ∧ ST=ST'"
  <proof>

end

```

## 9 The Original Otway-Rees Protocol

**theory** *OtwayRees* **imports** *Public* **begin**

From page 244 of Burrows, Abadi and Needham (1989). A Logic of Authentication. Proc. Royal Soc. 426

This is the original version, which encrypts Nonce NB.

```

inductive_set otway :: "event list set"
  where
    Nil: "[ ] ∈ otway"
    — Initial trace is empty
  / Fake: "[ evsf ∈ otway; X ∈ synth (analz (knows Spy evsf)) ]
    ⇒ Says Spy B X # evsf ∈ otway"
    — The spy can say almost anything.
  / Reception: "[ evsr ∈ otway; Says A B X ∈ set evsr ] ⇒ Gets B X # evsr
  ∈ otway"
    — A message that has been sent can be received by the intended recipient.
  / OR1: "[ evs1 ∈ otway; Nonce NA ∉ used evs1 ]
    ⇒ Says A B {Nonce NA, Agent A, Agent B,
      Crypt (shrK A) {Nonce NA, Agent A, Agent B} }
    # evs1 ∈ otway"
    — Alice initiates a protocol run
  / OR2: "[ evs2 ∈ otway; Nonce NB ∉ used evs2;
    Gets B {Nonce NA, Agent A, Agent B, X} ∈ set evs2 ]
    ⇒ Says B Server
      {Nonce NA, Agent A, Agent B, X,
      Crypt (shrK B)
      {Nonce NA, Nonce NB, Agent A, Agent B} }
    # evs2 ∈ otway"
    — Bob's response to Alice's message. Note that NB is encrypted.
  / OR3: "[ evs3 ∈ otway; Key KAB ∉ used evs3;
    Gets Server
      {Nonce NA, Agent A, Agent B,
      Crypt (shrK A) {Nonce NA, Agent A, Agent B},
      Crypt (shrK B) {Nonce NA, Nonce NB, Agent A, Agent B} }
    ∈ set evs3 ]
    ⇒ Says Server B
      {Nonce NA,

```

$\text{Crypt } (\text{shrK } A) \{ \text{Nonce } NA, \text{Key } KAB \},$   
 $\text{Crypt } (\text{shrK } B) \{ \text{Nonce } NB, \text{Key } KAB \}$   
 $\# \text{ evs3} \in \text{otway}$

— The Server receives Bob's message and checks that the three NAs match. Then he sends a new session key to Bob with a packet for forwarding to Alice

/ OR4: " $\llbracket \text{evs4} \in \text{otway}; B \neq \text{Server};$   
 $\text{Says } B \text{ Server } \{ \text{Nonce } NA, \text{Agent } A, \text{Agent } B, X',$   
 $\text{Crypt } (\text{shrK } B)$   
 $\{ \text{Nonce } NA, \text{Nonce } NB, \text{Agent } A, \text{Agent } B \} \}$   
 $\in \text{set evs4};$   
 $\text{Gets } B \{ \text{Nonce } NA, X, \text{Crypt } (\text{shrK } B) \{ \text{Nonce } NB, \text{Key } K \} \}$   
 $\in \text{set evs4} \rrbracket$   
 $\implies \text{Says } B A \{ \text{Nonce } NA, X \} \# \text{ evs4} \in \text{otway}$ "

— Bob receives the Server's (?) message and compares the Nonces with those in the message he previously sent the Server. Need  $B \neq \text{Server}$  because we allow messages to self.

/ Ops: " $\llbracket \text{evso} \in \text{otway};$   
 $\text{Says } \text{Server } B \{ \text{Nonce } NA, X, \text{Crypt } (\text{shrK } B) \{ \text{Nonce } NB, \text{Key } K \} \}$   
 $\in \text{set evso} \rrbracket$   
 $\implies \text{Notes } \text{Spy } \{ \text{Nonce } NA, \text{Nonce } NB, \text{Key } K \} \# \text{ evso} \in \text{otway}$ "

— This message models possible leaks of session keys. The nonces identify the protocol run

```

declare Says_imp_analz_Spy [dest]
declare parts.Body [dest]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]

```

A "possibility property": there are traces that reach the end

```

lemma " $\llbracket B \neq \text{Server}; \text{Key } K \notin \text{used } [] \rrbracket$   

 $\implies \exists \text{ evs} \in \text{otway}.$   

 $\text{Says } B A \{ \text{Nonce } NA, \text{Crypt } (\text{shrK } A) \{ \text{Nonce } NA, \text{Key } K \} \}$   

 $\in \text{set evs}$ "

```

*<proof>*

```

lemma Gets_imp_Says [dest!]:
  " $\llbracket \text{Gets } B X \in \text{set evs}; \text{ evs} \in \text{otway} \rrbracket \implies \exists A. \text{Says } A B X \in \text{set evs}$ "

```

*<proof>*

```

lemma OR2_analz_knows_Spy:
  " $\llbracket \text{Gets } B \{ N, \text{Agent } A, \text{Agent } B, X \} \in \text{set evs}; \text{ evs} \in \text{otway} \rrbracket$   

 $\implies X \in \text{analz } (\text{knows } \text{Spy } \text{evs})$ "

```

*<proof>*

```

lemma OR4_analz_knows_Spy:
  " $\llbracket \text{Gets } B \{ N, X, \text{Crypt } (\text{shrK } B) X' \} \in \text{set evs}; \text{ evs} \in \text{otway} \rrbracket$   

 $\implies X \in \text{analz } (\text{knows } \text{Spy } \text{evs})$ "

```

*<proof>*

```

lemmas OR2_parts_knows_Spy =

```

*OR2\_analz\_knows\_Spy [THEN analz\_into\_parts]*

Theorems of the form  $X \notin \text{parts } (\text{knows Spy evs})$  imply that NOBODY sends messages containing X!

Spy never sees a good agent's shared key!

**lemma** *Spy\_see\_shrK [simp]:*  
 $"\text{evs} \in \text{otway} \implies (\text{Key } (\text{shrK } A) \in \text{parts } (\text{knows Spy evs})) = (A \in \text{bad})"$   
*<proof>*

**lemma** *Spy\_analz\_shrK [simp]:*  
 $"\text{evs} \in \text{otway} \implies (\text{Key } (\text{shrK } A) \in \text{analz } (\text{knows Spy evs})) = (A \in \text{bad})"$   
*<proof>*

**lemma** *Spy\_see\_shrK\_D [dest!]:*  
 $"[\text{Key } (\text{shrK } A) \in \text{parts } (\text{knows Spy evs}); \text{ evs} \in \text{otway}] \implies A \in \text{bad}"$   
*<proof>*

## 9.1 Towards Secrecy: Proofs Involving *analz*

Describes the form of K and NA when the Server sends this message. Also for Oops case.

**lemma** *Says\_Server\_message\_form:*  
 $"[\text{Says Server } B \{NA, X, \text{Crypt } (\text{shrK } B) \{NB, \text{Key } K\}\} \in \text{set evs};$   
 $\text{evs} \in \text{otway}]$   
 $\implies K \notin \text{range shrK} \wedge (\exists i. NA = \text{Nonce } i) \wedge (\exists j. NB = \text{Nonce } j)"$   
*<proof>*

Session keys are not used to encrypt other session keys

The equality makes the induction hypothesis easier to apply

**lemma** *analz\_image\_freshK [rule\_format]:*  
 $"\text{evs} \in \text{otway} \implies$   
 $\forall K KK. KK \subseteq \neg(\text{range shrK}) \longrightarrow$   
 $(\text{Key } K \in \text{analz } (\text{Key } KK \cup (\text{knows Spy evs}))) =$   
 $(K \in KK \mid \text{Key } K \in \text{analz } (\text{knows Spy evs}))"$   
*<proof>*

**lemma** *analz\_insert\_freshK:*  
 $"[\text{evs} \in \text{otway}; KAB \notin \text{range shrK}] \implies$   
 $(\text{Key } K \in \text{analz } (\text{insert } (\text{Key } KAB) (\text{knows Spy evs}))) =$   
 $(K = KAB \mid \text{Key } K \in \text{analz } (\text{knows Spy evs}))"$   
*<proof>*

The Key K uniquely identifies the Server's message.

**lemma** *unique\_session\_keys:*  
 $"[\text{Says Server } B \{NA, X, \text{Crypt } (\text{shrK } B) \{NB, K\}\} \in \text{set evs};$   
 $\text{Says Server } B' \{NA', X', \text{Crypt } (\text{shrK } B') \{NB', K\}\} \in \text{set evs};$   
 $\text{evs} \in \text{otway}] \implies X=X' \wedge B=B' \wedge NA=NA' \wedge NB=NB'"$   
*<proof>*

## 9.2 Authenticity properties relating to NA

Only OR1 can have caused such a part of a message to appear.

**lemma** *Crypt\_imp\_OR1 [rule\_format]:*  
 "[A  $\notin$  bad; evs  $\in$  otway]  
 $\implies$  Crypt (shrK A) {NA, Agent A, Agent B}  $\in$  parts (knows Spy evs)  $\longrightarrow$   
 Says A B {NA, Agent A, Agent B,  
 Crypt (shrK A) {NA, Agent A, Agent B}}  
 $\in$  set evs"  
 <proof>

**lemma** *Crypt\_imp\_OR1\_Gets:*  
 "[Gets B {NA, Agent A, Agent B,  
 Crypt (shrK A) {NA, Agent A, Agent B}}  $\in$  set evs;  
 A  $\notin$  bad; evs  $\in$  otway]  
 $\implies$  Says A B {NA, Agent A, Agent B,  
 Crypt (shrK A) {NA, Agent A, Agent B}}  
 $\in$  set evs"  
 <proof>

The Nonce NA uniquely identifies A's message

**lemma** *unique\_NA:*  
 "[Crypt (shrK A) {NA, Agent A, Agent B}  $\in$  parts (knows Spy evs);  
 Crypt (shrK A) {NA, Agent A, Agent C}  $\in$  parts (knows Spy evs);  
 evs  $\in$  otway; A  $\notin$  bad]  
 $\implies$  B = C"  
 <proof>

It is impossible to re-use a nonce in both OR1 and OR2. This holds because OR2 encrypts Nonce NB. It prevents the attack that can occur in the over-simplified version of this protocol: see *OtwayRees\_Bad*.

**lemma** *no\_nonce\_OR1\_OR2:*  
 "[Crypt (shrK A) {NA, Agent A, Agent B}  $\in$  parts (knows Spy evs);  
 A  $\notin$  bad; evs  $\in$  otway]  
 $\implies$  Crypt (shrK A) {NA', NA, Agent A', Agent A}  $\notin$  parts (knows Spy evs)"  
 <proof>

Crucial property: If the encrypted message appears, and A has used NA to start a run, then it originated with the Server!

**lemma** *NA\_Crypt\_imp\_Server\_msg [rule\_format]:*  
 "[A  $\notin$  bad; evs  $\in$  otway]  
 $\implies$  Says A B {NA, Agent A, Agent B,  
 Crypt (shrK A) {NA, Agent A, Agent B}}  $\in$  set evs  $\longrightarrow$   
 Crypt (shrK A) {NA, Key K}  $\in$  parts (knows Spy evs)  
 $\longrightarrow$  ( $\exists$  NB. Says Server B  
 {NA,  
 Crypt (shrK A) {NA, Key K},  
 Crypt (shrK B) {NB, Key K}}  $\in$  set evs)"  
 <proof>

Corollary: if A receives B's OR4 message and the nonce NA agrees then the key really did come from the Server! CANNOT prove this of the bad form of this protocol, even though we can prove *Spy\_not\_see\_encrypted\_key*



**lemma** *A\_trusts\_OR4*:

```
"[[Says A B {NA, Agent A, Agent B,
      Crypt (shrK A) {NA, Agent A, Agent B}}] ∈ set evs;
   Says B' A {NA, Crypt (shrK A) {NA, Key K}}] ∈ set evs;
   A ∉ bad; evs ∈ otway]]
⇒ ∃ NB. Says Server B
    {NA,
     Crypt (shrK A) {NA, Key K},
     Crypt (shrK B) {NB, Key K}}
   ∈ set evs"
```

*<proof>*

Crucial secrecy property: Spy does not see the keys sent in msg OR3 Does not in itself guarantee security: an attack could violate the premises, e.g. by having  $A = \text{Spy}$

**lemma** *secrecy\_lemma*:

```
"[A ∉ bad; B ∉ bad; evs ∈ otway]]
⇒ Says Server B
   {NA, Crypt (shrK A) {NA, Key K},
    Crypt (shrK B) {NB, Key K}} ∈ set evs →
   Notes Spy {NA, NB, Key K} ∉ set evs →
   Key K ∉ analz (knows Spy evs)"
```

*<proof>*

**theorem** *Spy\_not\_see\_encrypted\_key*:

```
"[[Says Server B
   {NA, Crypt (shrK A) {NA, Key K},
    Crypt (shrK B) {NB, Key K}}] ∈ set evs;
   Notes Spy {NA, NB, Key K} ∉ set evs;
   A ∉ bad; B ∉ bad; evs ∈ otway]]
⇒ Key K ∉ analz (knows Spy evs)"
```

*<proof>*

This form is an immediate consequence of the previous result. It is similar to the assertions established by other methods. It is equivalent to the previous result in that the Spy already has *analz* and *synth* at his disposal. However, the conclusion  $\text{Key } K \notin \text{knows Spy evs}$  appears not to be inductive: all the cases other than Fake are trivial, while Fake requires  $\text{Key } K \notin \text{analz (knows Spy evs)}$ .

**lemma** *Spy\_not\_know\_encrypted\_key*:

```
"[[Says Server B
   {NA, Crypt (shrK A) {NA, Key K},
    Crypt (shrK B) {NB, Key K}}] ∈ set evs;
   Notes Spy {NA, NB, Key K} ∉ set evs;
   A ∉ bad; B ∉ bad; evs ∈ otway]]
⇒ Key K ∉ knows Spy evs"
```

*<proof>*

A's guarantee. The Oops premise quantifies over NB because A cannot know what it is.

**lemma** *A\_gets\_good\_key*:

```
"[[Says A B {NA, Agent A, Agent B,
      Crypt (shrK A) {NA, Agent A, Agent B}}] ∈ set evs;
   Says B' A {NA, Crypt (shrK A) {NA, Key K}}] ∈ set evs;
```

$\forall NB. \text{Notes Spy } \{NA, NB, \text{Key } K\} \notin \text{set evs};$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{otway}$   
 $\implies \text{Key } K \notin \text{analz}(\text{knows Spy evs})"$   
 <proof>

### 9.3 Authenticity properties relating to NB

Only OR2 can have caused such a part of a message to appear. We do not know anything about X: it does NOT have to have the right form.

**lemma** *Crypt\_imp\_OR2*:

$"\llbracket \text{Crypt}(\text{shrK } B) \{NA, NB, \text{Agent } A, \text{Agent } B\} \in \text{parts}(\text{knows Spy evs});$   
 $B \notin \text{bad}; \text{evs} \in \text{otway} \rrbracket$   
 $\implies \exists X. \text{Says } B \text{ Server}$   
 $\quad \{NA, \text{Agent } A, \text{Agent } B, X,$   
 $\quad \text{Crypt}(\text{shrK } B) \{NA, NB, \text{Agent } A, \text{Agent } B\}\}$   
 $\in \text{set evs}"$   
 <proof>

The Nonce NB uniquely identifies B's message

**lemma** *unique\_NB*:

$"\llbracket \text{Crypt}(\text{shrK } B) \{NA, NB, \text{Agent } A, \text{Agent } B\} \in \text{parts}(\text{knows Spy evs});$   
 $\text{Crypt}(\text{shrK } B) \{NC, NB, \text{Agent } C, \text{Agent } B\} \in \text{parts}(\text{knows Spy evs});$   
 $\text{evs} \in \text{otway}; B \notin \text{bad} \rrbracket$   
 $\implies NC = NA \wedge C = A"$   
 <proof>

If the encrypted message appears, and B has used Nonce NB, then it originated with the Server! Quite messy proof.

**lemma** *NB\_Crypt\_imp\_Server\_msg* [rule\_format]:

$"\llbracket B \notin \text{bad}; \text{evs} \in \text{otway} \rrbracket$   
 $\implies \text{Crypt}(\text{shrK } B) \{NB, \text{Key } K\} \in \text{parts}(\text{knows Spy evs})$   
 $\longrightarrow (\forall X'. \text{Says } B \text{ Server}$   
 $\quad \{NA, \text{Agent } A, \text{Agent } B, X',$   
 $\quad \text{Crypt}(\text{shrK } B) \{NA, NB, \text{Agent } A, \text{Agent } B\}\}$   
 $\in \text{set evs}$   
 $\longrightarrow \text{Says Server } B$   
 $\quad \{NA, \text{Crypt}(\text{shrK } A) \{NA, \text{Key } K\},$   
 $\quad \text{Crypt}(\text{shrK } B) \{NB, \text{Key } K\}\}$   
 $\in \text{set evs})"$   
 <proof>

Guarantee for B: if it gets a message with matching NB then the Server has sent the correct message.

**theorem** *B\_trusts\_OR3*:

$"\llbracket \text{Says } B \text{ Server } \{NA, \text{Agent } A, \text{Agent } B, X',$   
 $\quad \text{Crypt}(\text{shrK } B) \{NA, NB, \text{Agent } A, \text{Agent } B\}\}$   
 $\in \text{set evs};$   
 $\text{Gets } B \{NA, X, \text{Crypt}(\text{shrK } B) \{NB, \text{Key } K\}\} \in \text{set evs};$   
 $B \notin \text{bad}; \text{evs} \in \text{otway} \rrbracket$   
 $\implies \text{Says Server } B$   
 $\quad \{NA,$   
 $\quad \text{Crypt}(\text{shrK } A) \{NA, \text{Key } K\},$

```

      Crypt (shrK B) {NB, Key K}
    ∈ set evs"
⟨proof⟩

The obvious combination of B_trusts_OR3 with Spy_not_see_encrypted_key

lemma B_gets_good_key:
  "[Says B Server {NA, Agent A, Agent B, X',
    Crypt (shrK B) {NA, NB, Agent A, Agent B}}
    ∈ set evs;
   Gets B {NA, X, Crypt (shrK B) {NB, Key K}} ∈ set evs;
   Notes Spy {NA, NB, Key K} ∉ set evs;
   A ∉ bad; B ∉ bad; evs ∈ otway]
  ⇒ Key K ∉ analz (knows Spy evs)"
⟨proof⟩

```

```

lemma OR3_imp_OR2:
  "[Says Server B
    {NA, Crypt (shrK A) {NA, Key K},
     Crypt (shrK B) {NB, Key K}} ∈ set evs;
   B ∉ bad; evs ∈ otway]
  ⇒ ∃ X. Says B Server {NA, Agent A, Agent B, X,
    Crypt (shrK B) {NA, NB, Agent A, Agent B}}
    ∈ set evs"
⟨proof⟩

```

After getting and checking OR4, agent A can trust that B has been active. We could probably prove that X has the expected form, but that is not strictly necessary for authentication.

```

theorem A_auths_B:
  "[Says B' A {NA, Crypt (shrK A) {NA, Key K}} ∈ set evs;
   Says A B {NA, Agent A, Agent B,
    Crypt (shrK A) {NA, Agent A, Agent B}} ∈ set evs;
   A ∉ bad; B ∉ bad; evs ∈ otway]
  ⇒ ∃ NB X. Says B Server {NA, Agent A, Agent B, X,
    Crypt (shrK B) {NA, NB, Agent A, Agent B}}
    ∈ set evs"
⟨proof⟩

```

**end**

## 10 The Otway-Rees Protocol as Modified by Abadi and Needham

**theory** OtwayRees\_AN **imports** Public **begin**

This simplified version has minimal encryption and explicit messages.

Note that the formalization does not even assume that nonces are fresh. This is because the protocol does not rely on uniqueness of nonces for security, only for freshness, and the proof script does not prove freshness properties.

From page 11 of Abadi and Needham (1996). Prudent Engineering Practice for Cryptographic Protocols. IEEE Trans. SE 22 (1)

```

inductive_set otway :: "event list set"
  where
    Nil: — The empty trace
          "[ ] ∈ otway"

    / Fake: — The Spy may say anything he can say. The sender field is correct, but
    agents don't use that information.
          "[[evsf ∈ otway; X ∈ synth (analz (knows Spy evsf))]]
          ⇒ Says Spy B X # evsf ∈ otway"

    / Reception: — A message that has been sent can be received by the intended
    recipient.
          "[[evsr ∈ otway; Says A B X ∈ set evsr]]
          ⇒ Gets B X # evsr ∈ otway"

    / OR1: — Alice initiates a protocol run
          "evs1 ∈ otway
          ⇒ Says A B {Agent A, Agent B, Nonce NA} # evs1 ∈ otway"

    / OR2: — Bob's response to Alice's message.
          "[[evs2 ∈ otway;
            Gets B {Agent A, Agent B, Nonce NA} ∈ set evs2]]
          ⇒ Says B Server {Agent A, Agent B, Nonce NA, Nonce NB}
            # evs2 ∈ otway"

    / OR3: — The Server receives Bob's message. Then he sends a new session key to
    Bob with a packet for forwarding to Alice.
          "[[evs3 ∈ otway; Key KAB ∉ used evs3;
            Gets Server {Agent A, Agent B, Nonce NA, Nonce NB}
              ∈ set evs3]]
          ⇒ Says Server B
            {Crypt (shrK A) {Nonce NA, Agent A, Agent B, Key KAB},
             Crypt (shrK B) {Nonce NB, Agent A, Agent B, Key KAB}}
            # evs3 ∈ otway"

    / OR4: — Bob receives the Server's (?) message and compares the Nonces with
    those in the message he previously sent the Server. Need B ≠ Server because we
    allow messages to self.
          "[[evs4 ∈ otway; B ≠ Server;
            Says B Server {Agent A, Agent B, Nonce NA, Nonce NB} ∈ set evs4;
            Gets B {X, Crypt (shrK B) {Nonce NB, Agent A, Agent B, Key K}}
              ∈ set evs4]]
          ⇒ Says B A X # evs4 ∈ otway"

    / Oops: — This message models possible leaks of session keys. The nonces identify
    the protocol run.
          "[[evso ∈ otway;
            Says Server B
              {Crypt (shrK A) {Nonce NA, Agent A, Agent B, Key K},
               Crypt (shrK B) {Nonce NB, Agent A, Agent B, Key K}}
              ∈ set evso]]
          ⇒ Notes Spy {Nonce NA, Nonce NB, Key K} # evso ∈ otway"

```

```

declare Says_imp_knows_Spy [THEN analz.Inj, dest]
declare parts.Body [dest]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]

```

A "possibility property": there are traces that reach the end

```

lemma "[B ≠ Server; Key K ∉ used []]
  ⇒ ∃ evs ∈ otway.
    Says B A (Crypt (shrK A) {Nonce NA, Agent A, Agent B, Key K})
    ∈ set evs"
<proof>

```

```

lemma Gets_imp_Says [dest!]:
  "[Gets B X ∈ set evs; evs ∈ otway] ⇒ ∃ A. Says A B X ∈ set evs"
<proof>

```

For reasoning about the encrypted portion of messages

```

lemma OR4_analz_knows_Spy:
  "[Gets B {X, Crypt (shrK B) X'} ∈ set evs; evs ∈ otway]
  ⇒ X ∈ analz (knows Spy evs)"
<proof>

```

Theorems of the form  $X \notin \text{parts } (\text{knows Spy evs})$  imply that NOBODY sends messages containing X!

Spy never sees a good agent's shared key!

```

lemma Spy_see_shrK [simp]:
  "evs ∈ otway ⇒ (Key (shrK A) ∈ parts (knows Spy evs)) = (A ∈ bad)"
<proof>

```

```

lemma Spy_analz_shrK [simp]:
  "evs ∈ otway ⇒ (Key (shrK A) ∈ analz (knows Spy evs)) = (A ∈ bad)"
<proof>

```

```

lemma Spy_see_shrK_D [dest!]:
  "[Key (shrK A) ∈ parts (knows Spy evs); evs ∈ otway] ⇒ A ∈ bad"
<proof>

```

## 10.1 Proofs involving *analz*

Describes the form of K and NA when the Server sends this message.

```

lemma Says_Server_message_form:
  "[Says Server B
    {Crypt (shrK A) {NA, Agent A, Agent B, Key K},
     Crypt (shrK B) {NB, Agent A, Agent B, Key K}}
    ∈ set evs;
   evs ∈ otway]
  ⇒ K ∉ range shrK ∧ (∃ i. NA = Nonce i) ∧ (∃ j. NB = Nonce j)"
<proof>

```

Session keys are not used to encrypt other session keys

The equality makes the induction hypothesis easier to apply

```

lemma analz_image_freshK [rule_format]:
  "evs ∈ otway ⇒
    ∀K KK. KK ⊆ -(range shrK) ⇒
      (Key K ∈ analz (Key'KK ∪ (knows Spy evs))) =
      (K ∈ KK | Key K ∈ analz (knows Spy evs))"
  <proof>

lemma analz_insert_freshK:
  "[[evs ∈ otway; KAB ∉ range shrK]] ⇒
    (Key K ∈ analz (insert (Key KAB) (knows Spy evs))) =
    (K = KAB | Key K ∈ analz (knows Spy evs))"
  <proof>

```

The Key K uniquely identifies the Server's message.

```

lemma unique_session_keys:
  "[[Says Server B
    {Crypt (shrK A) {NA, Agent A, Agent B, K}},
     Crypt (shrK B) {NB, Agent A, Agent B, K}}
   ∈ set evs;
   Says Server B'
    {Crypt (shrK A') {NA', Agent A', Agent B', K}},
     Crypt (shrK B') {NB', Agent A', Agent B', K}}
   ∈ set evs;
   evs ∈ otway]]
  ⇒ A=A' ∧ B=B' ∧ NA=NA' ∧ NB=NB'"
  <proof>

```

## 10.2 Authenticity properties relating to NA

If the encrypted message appears then it originated with the Server!

```

lemma NA_Crypt_imp_Server_msg [rule_format]:
  "[[A ∉ bad; A ≠ B; evs ∈ otway]]
  ⇒ Crypt (shrK A) {NA, Agent A, Agent B, Key K} ∈ parts (knows Spy evs)
  → (∃NB. Says Server B
    {Crypt (shrK A) {NA, Agent A, Agent B, Key K}},
     Crypt (shrK B) {NB, Agent A, Agent B, Key K}}
   ∈ set evs)"
  <proof>

```

Corollary: if A receives B's OR4 message then it originated with the Server.  
Freshness may be inferred from nonce NA.

```

lemma A_trusts_OR4:
  "[[Says B' A (Crypt (shrK A) {NA, Agent A, Agent B, Key K}) ∈ set evs;
    A ∉ bad; A ≠ B; evs ∈ otway]]
  ⇒ ∃NB. Says Server B
    {Crypt (shrK A) {NA, Agent A, Agent B, Key K}},
     Crypt (shrK B) {NB, Agent A, Agent B, Key K}}
   ∈ set evs"
  <proof>

```

Crucial secrecy property: Spy does not see the keys sent in msg OR3 Does not

in itself guarantee security: an attack could violate the premises, e.g. by having  $A = \text{Spy}$

```

lemma secrecy_lemma:
  "[[A ∉ bad; B ∉ bad; evs ∈ otway]]
  ⇒ Says Server B
    {Crypt (shrK A) {NA, Agent A, Agent B, Key K},
     Crypt (shrK B) {NB, Agent A, Agent B, Key K}}
  ∈ set evs →
  Notes Spy {NA, NB, Key K} ∉ set evs →
  Key K ∉ analz (knows Spy evs)"
<proof>

```

```

lemma Spy_not_see_encrypted_key:
  "[[Says Server B
    {Crypt (shrK A) {NA, Agent A, Agent B, Key K},
     Crypt (shrK B) {NB, Agent A, Agent B, Key K}}
  ∈ set evs;
  Notes Spy {NA, NB, Key K} ∉ set evs;
  A ∉ bad; B ∉ bad; evs ∈ otway]]
  ⇒ Key K ∉ analz (knows Spy evs)"
<proof>

```

A's guarantee. The Oops premise quantifies over NB because A cannot know what it is.

```

lemma A_gets_good_key:
  "[[Says B' A (Crypt (shrK A) {NA, Agent A, Agent B, Key K}) ∈ set evs;
  ∀ NB. Notes Spy {NA, NB, Key K} ∉ set evs;
  A ∉ bad; B ∉ bad; A ≠ B; evs ∈ otway]]
  ⇒ Key K ∉ analz (knows Spy evs)"
<proof>

```

### 10.3 Authenticity properties relating to NB

If the encrypted message appears then it originated with the Server!

```

lemma NB_Crypt_imp_Server_msg [rule_format]:
  "[[B ∉ bad; A ≠ B; evs ∈ otway]]
  ⇒ Crypt (shrK B) {NB, Agent A, Agent B, Key K} ∈ parts (knows Spy evs)
  → (∃ NA. Says Server B
    {Crypt (shrK A) {NA, Agent A, Agent B, Key K},
     Crypt (shrK B) {NB, Agent A, Agent B, Key K}}
  ∈ set evs)"
<proof>

```

Guarantee for B: if it gets a well-formed certificate then the Server has sent the correct message in round 3.

```

lemma B_trusts_OR3:
  "[[Says S B {X, Crypt (shrK B) {NB, Agent A, Agent B, Key K}}
  ∈ set evs;
  B ∉ bad; A ≠ B; evs ∈ otway]]
  ⇒ ∃ NA. Says Server B
    {Crypt (shrK A) {NA, Agent A, Agent B, Key K},

```

```

      Crypt (shrK B) {NB, Agent A, Agent B, Key K}
    ∈ set evs"
⟨proof⟩

```

The obvious combination of *B\_trusts\_OR3* with *Spy\_not\_see\_encrypted\_key*

```

lemma B_gets_good_key:
  "[Gets B {X, Crypt (shrK B) {NB, Agent A, Agent B, Key K}}
   ∈ set evs;
   ∀NA. Notes Spy {NA, NB, Key K} ∉ set evs;
   A ∉ bad; B ∉ bad; A ≠ B; evs ∈ otway]
  ⇒ Key K ∉ analz (knows Spy evs)"
⟨proof⟩

```

**end**

## 11 The Otway-Rees Protocol: The Faulty BAN Version

**theory** *OtwayRees\_Bad* **imports** *Public* **begin**

The FAULTY version omitting encryption of Nonce NB, as suggested on page 247 of Burrows, Abadi and Needham (1988). A Logic of Authentication. Proc. Royal Soc. 426

This file illustrates the consequences of such errors. We can still prove impressive-looking properties such as *Spy\_not\_see\_encrypted\_key*, yet the protocol is open to a middleperson attack. Attempting to prove some key lemmas indicates the possibility of this attack.

**inductive\_set** *otway* :: "event list set"

**where**

*Nil*: — The empty trace

"[] ∈ *otway*"

*/ Fake*: — The Spy may say anything he can say. The sender field is correct, but agents don't use that information.

```

  "[evsf ∈ otway; X ∈ synth (analz (knows Spy evsf))]
  ⇒ Says Spy B X # evsf ∈ otway"

```

*/ Reception*: — A message that has been sent can be received by the intended recipient.

```

  "[evsr ∈ otway; Says A B X ∈ set evsr]
  ⇒ Gets B X # evsr ∈ otway"

```

*/ OR1*: — Alice initiates a protocol run

```

  "[evs1 ∈ otway; Nonce NA ∉ used evs1]
  ⇒ Says A B {Nonce NA, Agent A, Agent B,
               Crypt (shrK A) {Nonce NA, Agent A, Agent B}}
  # evs1 ∈ otway"

```

*/ OR2*: — Bob's response to Alice's message. This variant of the protocol does NOT encrypt NB.

```

  "[evs2 ∈ otway; Nonce NB ∉ used evs2;

```



```

    Gets B {Nonce NA, Agent A, Agent B, X} ∈ set evs2
  ⇒ Says B Server
    {Nonce NA, Agent A, Agent B, X, Nonce NB,
     Crypt (shrK B) {Nonce NA, Agent A, Agent B}}
    # evs2 ∈ otway"

```

/ OR3: — The Server receives Bob's message and checks that the three NAs match. Then he sends a new session key to Bob with a packet for forwarding to Alice.

```

  "[evs3 ∈ otway; Key KAB ∉ used evs3;
   Gets Server
     {Nonce NA, Agent A, Agent B,
      Crypt (shrK A) {Nonce NA, Agent A, Agent B},
      Nonce NB,
      Crypt (shrK B) {Nonce NA, Agent A, Agent B}}
     ∈ set evs3]
  ⇒ Says Server B
     {Nonce NA,
      Crypt (shrK A) {Nonce NA, Key KAB},
      Crypt (shrK B) {Nonce NB, Key KAB}}
     # evs3 ∈ otway"

```

/ OR4: — Bob receives the Server's (?) message and compares the Nonces with those in the message he previously sent the Server. Need  $B \neq \text{Server}$  because we allow messages to self.

```

  "[evs4 ∈ otway; B ≠ Server;
   Says B Server {Nonce NA, Agent A, Agent B, X', Nonce NB,
                  Crypt (shrK B) {Nonce NA, Agent A, Agent B}}
   ∈ set evs4;
   Gets B {Nonce NA, X, Crypt (shrK B) {Nonce NB, Key K}}
   ∈ set evs4]
  ⇒ Says B A {Nonce NA, X} # evs4 ∈ otway"

```

/ Ops: — This message models possible leaks of session keys. The nonces identify the protocol run.

```

  "[evso ∈ otway;
   Says Server B {Nonce NA, X, Crypt (shrK B) {Nonce NB, Key K}}
   ∈ set evso]
  ⇒ Notes Spy {Nonce NA, Nonce NB, Key K} # evso ∈ otway"

```

```

declare Says_imp_knows_Spy [THEN analz.Inj, dest]
declare parts.Body [dest]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]

```

A "possibility property": there are traces that reach the end

```

lemma "[B ≠ Server; Key K ∉ used []]
  ⇒ ∃ NA. ∃ evs ∈ otway.
    Says B A {Nonce NA, Crypt (shrK A) {Nonce NA, Key K}}
    ∈ set evs"

```

⟨proof⟩

```

lemma Gets_imp_Says [dest!]:
  "[Gets B X ∈ set evs; evs ∈ otway] ⇒ ∃ A. Says A B X ∈ set evs"

```

*<proof>*

### 11.1 For reasoning about the encrypted portion of messages

**lemma** *OR2\_analz\_knows\_Spy*:

"[[Gets B {N, Agent A, Agent B, X}] ∈ set evs; evs ∈ otway]  
 ⇒ X ∈ analz (knows Spy evs)"

*<proof>*

**lemma** *OR4\_analz\_knows\_Spy*:

"[[Gets B {N, X, Crypt (shrK B) X'}] ∈ set evs; evs ∈ otway]  
 ⇒ X ∈ analz (knows Spy evs)"

*<proof>*

**lemma** *Oops\_parts\_knows\_Spy*:

"Says Server B {NA, X, Crypt K' {NB, K}} ∈ set evs  
 ⇒ K ∈ parts (knows Spy evs)"

*<proof>*

Forwarding lemma: see comments in OtwayRees.thy

**lemmas** *OR2\_parts\_knows\_Spy* =

*OR2\_analz\_knows\_Spy [THEN analz\_into\_parts]*

Theorems of the form  $X \notin \text{parts (knows Spy evs)}$  imply that NOBODY sends messages containing X!

Spy never sees a good agent's shared key!

**lemma** *Spy\_see\_shrK [simp]*:

"evs ∈ otway ⇒ (Key (shrK A) ∈ parts (knows Spy evs)) = (A ∈ bad)"

*<proof>*

**lemma** *Spy\_analz\_shrK [simp]*:

"evs ∈ otway ⇒ (Key (shrK A) ∈ analz (knows Spy evs)) = (A ∈ bad)"

*<proof>*

**lemma** *Spy\_see\_shrK\_D [dest!]*:

"[[Key (shrK A) ∈ parts (knows Spy evs); evs ∈ otway] ⇒ A ∈ bad"

*<proof>*

### 11.2 Proofs involving analz

Describes the form of K and NA when the Server sends this message. Also for Oops case.

**lemma** *Says\_Server\_message\_form*:

"[[Says Server B {NA, X, Crypt (shrK B) {NB, Key K}}] ∈ set evs;  
 evs ∈ otway]  
 ⇒ K ∉ range shrK ∧ (∃ i. NA = Nonce i) ∧ (∃ j. NB = Nonce j)"

*<proof>*

Session keys are not used to encrypt other session keys

The equality makes the induction hypothesis easier to apply

```
lemma analz_image_freshK [rule_format]:
  "evs ∈ otway ⇒
    ∀ K KK. KK ⊆ -(range shrK) →
      (Key K ∈ analz (Key'KK ∪ (knows Spy evs))) =
      (K ∈ KK | Key K ∈ analz (knows Spy evs))"
  <proof>
```

```
lemma analz_insert_freshK:
  "[evs ∈ otway; KAB ∉ range shrK] ⇒
    (Key K ∈ analz (insert (Key KAB) (knows Spy evs))) =
    (K = KAB | Key K ∈ analz (knows Spy evs))"
  <proof>
```

The Key K uniquely identifies the Server's message.

```
lemma unique_session_keys:
  "[Says Server B {NA, X, Crypt (shrK B) {NB, K}} ∈ set evs;
   Says Server B' {NA', X', Crypt (shrK B') {NB', K}} ∈ set evs;
   evs ∈ otway] ⇒ X=X' ∧ B=B' ∧ NA=NA' ∧ NB=NB'"
  <proof>
```

Crucial secrecy property: Spy does not see the keys sent in msg OR3 Does not in itself guarantee security: an attack could violate the premises, e.g. by having  $A = \text{Spy}$

```
lemma secrecy_lemma:
  "[A ∉ bad; B ∉ bad; evs ∈ otway]
   ⇒ Says Server B
      {NA, Crypt (shrK A) {NA, Key K}},
      Crypt (shrK B) {NB, Key K}} ∈ set evs →
    Notes Spy {NA, NB, Key K} ∉ set evs →
    Key K ∉ analz (knows Spy evs)"
  <proof>
```

```
lemma Spy_not_see_encrypted_key:
  "[Says Server B
    {NA, Crypt (shrK A) {NA, Key K}},
    Crypt (shrK B) {NB, Key K}} ∈ set evs;
   Notes Spy {NA, NB, Key K} ∉ set evs;
   A ∉ bad; B ∉ bad; evs ∈ otway]
   ⇒ Key K ∉ analz (knows Spy evs)"
  <proof>
```

### 11.3 Attempting to prove stronger properties

Only OR1 can have caused such a part of a message to appear. The premise  $A \neq B$  prevents OR2's similar-looking cryptogram from being picked up. Original Otway-Rees doesn't need it.

```
lemma Crypt_imp_OR1 [rule_format]:
  "[A ∉ bad; A ≠ B; evs ∈ otway]
   ⇒ Crypt (shrK A) {NA, Agent A, Agent B} ∈ parts (knows Spy evs) →
    Says A B {NA, Agent A, Agent B,
```

$\text{Crypt } (\text{shrK } A) \{ \{NA, \text{Agent } A, \text{Agent } B\} \} \in \text{set evs}$ "

$\langle \text{proof} \rangle$

Crucial property: If the encrypted message appears, and A has used NA to start a run, then it originated with the Server! The premise  $A \neq B$  allows use of *Crypt\_imp\_OR1*

Only it is FALSE. Somebody could make a fake message to Server substituting some other nonce NA' for NB.

**lemma** " $\llbracket A \notin \text{bad}; A \neq B; \text{evs} \in \text{otway} \rrbracket$   
 $\implies \text{Crypt } (\text{shrK } A) \{ \{NA, \text{Key } K\} \} \in \text{parts } (\text{knows Spy evs}) \implies$   
 $\text{Says } A \ B \ \{ \{NA, \text{Agent } A, \text{Agent } B, \text{Crypt } (\text{shrK } A) \{ \{NA, \text{Agent } A, \text{Agent } B\} \} \}$   
 $\in \text{set evs} \implies$   
 $(\exists B \ NB. \text{Says Server } B$   
 $\{ \{NA,$   
 $\text{Crypt } (\text{shrK } A) \{ \{NA, \text{Key } K\} \},$   
 $\text{Crypt } (\text{shrK } B) \{ \{NB, \text{Key } K\} \} \in \text{set evs})"$

$\langle \text{proof} \rangle$

**end**

## 12 Bella's version of the Otway-Rees protocol

**theory** *OtwayReesBella* **imports** *Public* **begin**

Bella's modifications to a version of the Otway-Rees protocol taken from the BAN paper only concern message 7. The updated protocol makes the goal of key distribution of the session key available to A. Investigating the principle of Goal Availability undermines the BAN claim about the original protocol, that "this protocol does not make use of Kab as an encryption key, so neither principal can know whether the key is known to the other". The updated protocol makes no use of the session key to encrypt but informs A that B knows it.

**inductive\_set** *orb* :: "event list set"  
**where**

*Nil*: " $\llbracket \rrbracket \in \text{orb}$ "

*Fake*: " $\llbracket \text{evsa} \in \text{orb}; X \in \text{synth } (\text{analz } (\text{knows Spy evsa})) \rrbracket$   
 $\implies \text{Says Spy } B \ X \ \# \text{ evsa} \in \text{orb}$ "

*Reception*: " $\llbracket \text{evsr} \in \text{orb}; \text{Says } A \ B \ X \in \text{set evsr} \rrbracket$   
 $\implies \text{Gets } B \ X \ \# \text{ evsr} \in \text{orb}$ "

*OR1*: " $\llbracket \text{evs1} \in \text{orb}; \text{Nonce } NA \notin \text{used evs1} \rrbracket$   
 $\implies \text{Says } A \ B \ \{ \{ \text{Nonce } M, \text{Agent } A, \text{Agent } B,$   
 $\text{Crypt } (\text{shrK } A) \{ \{ \text{Nonce } NA, \text{Nonce } M, \text{Agent } A, \text{Agent } B \} \}$   
 $\# \text{ evs1} \in \text{orb} \rrbracket$ "

*OR2*: " $\llbracket \text{evs2} \in \text{orb}; \text{Nonce } NB \notin \text{used evs2};$   
 $\text{Gets } B \ \{ \{ \text{Nonce } M, \text{Agent } A, \text{Agent } B, X \} \} \in \text{set evs2} \rrbracket$   
 $\implies \text{Says } B \ \text{Server}$ "

```

    {Nonce M, Agent A, Agent B, X,
    Crypt (shrK B) {Nonce NB, Nonce M, Nonce M, Agent A, Agent B}}
    # evs2 ∈ orb"

/ OR3: "[[evs3 ∈ orb; Key KAB ∉ used evs3;
    Gets Server
    {Nonce M, Agent A, Agent B,
    Crypt (shrK A) {Nonce NA, Nonce M, Agent A, Agent B},
    Crypt (shrK B) {Nonce NB, Nonce M, Nonce M, Agent A, Agent
B}}]
    ∈ set evs3]
⇒ Says Server B {Nonce M,
    Crypt (shrK B) {Crypt (shrK A) {Nonce NA, Key KAB},
    Nonce NB, Key KAB}}
    # evs3 ∈ orb"

/ OR4: "[[evs4 ∈ orb; B ≠ Server; ∀ p q. X ≠ {p, q};
    Says B Server {Nonce M, Agent A, Agent B, X'},
    Crypt (shrK B) {Nonce NB, Nonce M, Nonce M, Agent A, Agent
B}}]
    ∈ set evs4;
    Gets B {Nonce M, Crypt (shrK B) {X, Nonce NB, Key KAB}}
    ∈ set evs4]
⇒ Says B A {Nonce M, X} # evs4 ∈ orb"

/ Ops: "[[evso ∈ orb;
    Says Server B {Nonce M,
    Crypt (shrK B) {Crypt (shrK A) {Nonce NA, Key KAB},
    Nonce NB, Key KAB}}
    ∈ set evso]
⇒ Notes Spy {Agent A, Agent B, Nonce NA, Nonce NB, Key KAB} # evso
    ∈ orb"

```

```

declare knows_Spy_partsEs [elim]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]

```

Fragile proof, with backtracking in the possibility call.

```

lemma possibility_thm: "[[A ≠ Server; B ≠ Server; Key K ∉ used[]]
    ⇒ ∃ evs ∈ orb.
    Says B A {Nonce M, Crypt (shrK A) {Nonce Na, Key K}} ∈ set evs"
<proof>

```

```

lemma Gets_imp_Says :
    "[[Gets B X ∈ set evs; evs ∈ orb] ⇒ ∃ A. Says A B X ∈ set evs"
<proof>

```

```

lemma Gets_imp_knows_Spy:

```

```

    "[Gets B X ∈ set evs; evs ∈ orb] ⇒ X ∈ knows Spy evs"
  <proof>

declare Gets_imp_knows_Spy [THEN parts.Inj, dest]

lemma Gets_imp_knows:
  "[Gets B X ∈ set evs; evs ∈ orb] ⇒ X ∈ knows B evs"
  <proof>

lemma OR2_analz_knows_Spy:
  "[Gets B {Nonce M, Agent A, Agent B, X} ∈ set evs; evs ∈ orb]
  ⇒ X ∈ analz (knows Spy evs)"
  <proof>

lemma OR4_parts_knows_Spy:
  "[Gets B {Nonce M, Crypt (shrK B) {X, Nonce Nb, Key Kab}} ∈ set evs;
  evs ∈ orb] ⇒ X ∈ parts (knows Spy evs)"
  <proof>

lemma Ops_parts_knows_Spy:
  "Says Server B {Nonce M, Crypt K' {X, Nonce Nb, K}} ∈ set evs
  ⇒ K ∈ parts (knows Spy evs)"
  <proof>

lemmas OR2_parts_knows_Spy =
  OR2_analz_knows_Spy [THEN analz_into_parts]

  <ML>

lemma Spy_see_shrK [simp]:
  "evs ∈ orb ⇒ (Key (shrK A) ∈ parts (knows Spy evs)) = (A ∈ bad)"
  <proof>

lemma Spy_analz_shrK [simp]:
  "evs ∈ orb ⇒ (Key (shrK A) ∈ analz (knows Spy evs)) = (A ∈ bad)"
  <proof>

lemma Spy_see_shrK_D [dest!]:
  "[Key (shrK A) ∈ parts (knows Spy evs); evs ∈ orb] ⇒ A ∈ bad"
  <proof>

lemma new_keys_not_used [simp]:
  "[Key K ∉ used evs; K ∈ symKeys; evs ∈ orb] ⇒ K ∉ keysFor (parts (knows
  Spy evs))"
  <proof>

```

## 12.1 Proofs involving analz

Describes the form of K and NA when the Server sends this message. Also for Ops case.

```

lemma Says_Server_message_form:
  "[Says Server B {Nonce M, Crypt (shrK B) {X, Nonce Nb, Key K}} ∈ set evs;

```

$evs \in orb$   
 $\implies K \notin \text{range shrK} \wedge (\exists A \text{ Na. } X = (\text{Crypt (shrK A) } \{ \text{Nonce Na, Key K} \} ))$   
 $\langle \text{proof} \rangle$

**lemma Says\_Server\_imp\_Gets:**

$\llbracket \text{Says Server B } \{ \text{Nonce M, Crypt (shrK B) } \{ \text{Crypt (shrK A) } \{ \text{Nonce Na, Key K} \}, \text{Nonce Nb, Key K} \} \} \in \text{set evs};$   
 $evs \in orb$   
 $\implies \text{Gets Server } \{ \text{Nonce M, Agent A, Agent B,}$   
 $\text{Crypt (shrK A) } \{ \text{Nonce Na, Nonce M, Agent A, Agent B} \},$   
 $\text{Crypt (shrK B) } \{ \text{Nonce Nb, Nonce M, Nonce M, Agent A, Agent}$   
 $\text{B} \} \}$   
 $\in \text{set evs}$   
 $\langle \text{proof} \rangle$

**lemma A\_trusts\_OR1:**

$\llbracket \text{Crypt (shrK A) } \{ \text{Nonce Na, Nonce M, Agent A, Agent B} \} \in \text{parts (knows Spy evs)};$   
 $A \notin \text{bad}; evs \in orb$   
 $\implies \text{Says A B } \{ \text{Nonce M, Agent A, Agent B, Crypt (shrK A) } \{ \text{Nonce Na, Nonce}$   
 $\text{M, Agent A, Agent B} \} \} \in \text{set evs}$   
 $\langle \text{proof} \rangle$

**lemma B\_trusts\_OR2:**

$\llbracket \text{Crypt (shrK B) } \{ \text{Nonce Nb, Nonce M, Nonce M, Agent A, Agent B} \}$   
 $\in \text{parts (knows Spy evs)}; B \notin \text{bad}; evs \in orb$   
 $\implies (\exists X. \text{Says B Server } \{ \text{Nonce M, Agent A, Agent B, X,}$   
 $\text{Crypt (shrK B) } \{ \text{Nonce Nb, Nonce M, Nonce M, Agent A, Agent B} \} \}$   
 $\in \text{set evs})$   
 $\langle \text{proof} \rangle$

**lemma B\_trusts\_OR3:**

$\llbracket \text{Crypt (shrK B) } \{ X, \text{Nonce Nb, Key K} \} \in \text{parts (knows Spy evs)};$   
 $B \notin \text{bad}; evs \in orb$   
 $\implies \exists M. \text{Says Server B } \{ \text{Nonce M, Crypt (shrK B) } \{ X, \text{Nonce Nb, Key K} \} \}$   
 $\in \text{set evs}$   
 $\langle \text{proof} \rangle$

**lemma Gets\_Server\_message\_form:**

$\llbracket \text{Gets B } \{ \text{Nonce M, Crypt (shrK B) } \{ X, \text{Nonce Nb, Key K} \} \} \in \text{set evs};$   
 $evs \in orb$   
 $\implies (K \notin \text{range shrK} \wedge (\exists A \text{ Na. } X = (\text{Crypt (shrK A) } \{ \text{Nonce Na, Key K} \} ))$   
 $\mid X \in \text{analz (knows Spy evs)})$   
 $\langle \text{proof} \rangle$

**lemma unique\_Na:**  $\llbracket \text{Says A B } \{ \text{Nonce M, Agent A, Agent B, Crypt (shrK A) } \{ \text{Nonce}$   
 $\text{Na, Nonce M, Agent A, Agent B} \} \} \in \text{set evs};$

$\text{Says A B' } \{ \text{Nonce M', Agent A, Agent B', Crypt (shrK A) } \{ \text{Nonce Na,}$

Nonce  $M'$ , Agent  $A$ , Agent  $B'$ }}  $\in \text{set evs}$ ;  
 $A \notin \text{bad}; \text{evs} \in \text{orb} \implies B=B' \wedge M=M'$ "  
 <proof>

**lemma** unique\_Nb: "[Says B Server {Nonce  $M$ , Agent  $A$ , Agent  $B$ ,  $X$ , Crypt (shrK  $B$ ) {Nonce  $Nb$ , Nonce  $M$ , Nonce  $M$ , Agent  $A$ , Agent  $B$ }}  $\in \text{set evs}$ ;  
 Says B Server {Nonce  $M'$ , Agent  $A'$ , Agent  $B$ ,  $X'$ , Crypt (shrK  $B$ ) {Nonce  $Nb$ , Nonce  $M'$ , Nonce  $M'$ , Agent  $A'$ , Agent  $B$ }}  $\in \text{set evs}$ ;  
 $B \notin \text{bad}; \text{evs} \in \text{orb} \implies M=M' \wedge A=A' \wedge X=X'$ "  
 <proof>

**lemma** analz\_image\_freshCryptK\_lemma:  
 "(Crypt  $K$   $X \in \text{analz} (\text{Key}'nE \cup H)) \longrightarrow (\text{Crypt } K \text{ } X \in \text{analz } H) \implies$   
 (Crypt  $K$   $X \in \text{analz} (\text{Key}'nE \cup H)) = (\text{Crypt } K \text{ } X \in \text{analz } H)"$   
 <proof>  
 <ML>

**lemma** analz\_image\_freshCryptK [rule\_format]:  
 "evs  $\in \text{orb} \implies$   
 Key  $K \notin \text{analz} (\text{knows Spy evs}) \longrightarrow$   
 ( $\forall KK. KK \subseteq - (\text{range shrK}) \longrightarrow$   
 (Crypt  $K$   $X \in \text{analz} (\text{Key}'KK \cup (\text{knows Spy evs}))) =$   
 (Crypt  $K$   $X \in \text{analz} (\text{knows Spy evs})))"$   
 <proof>

**lemma** analz\_insert\_freshCryptK:  
 "[[evs  $\in \text{orb};$  Key  $K \notin \text{analz} (\text{knows Spy evs});$   
 Seskey  $\notin \text{range shrK}] \implies$   
 (Crypt  $K$   $X \in \text{analz} (\text{insert} (\text{Key Seskey}) (\text{knows Spy evs}))) =$   
 (Crypt  $K$   $X \in \text{analz} (\text{knows Spy evs}))"$   
 <proof>

**lemma** analz\_hard:  
 "[[Says A B {Nonce  $M$ , Agent  $A$ , Agent  $B$ ,  
 Crypt (shrK  $A$ ) {Nonce  $Na$ , Nonce  $M$ , Agent  $A$ , Agent  $B$ }}  $\in \text{set evs}$ ;  
 Crypt (shrK  $A$ ) {Nonce  $Na$ , Key  $K$ }}  $\in \text{analz} (\text{knows Spy evs});$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{orb}]$   
 $\implies$  Says B A {Nonce  $M$ , Crypt (shrK  $A$ ) {Nonce  $Na$ , Key  $K$ }}  $\in \text{set evs}"$   
 <proof>

**lemma** Gets\_Server\_message\_form':  
 "[[Gets B {Nonce  $M$ , Crypt (shrK  $B$ ) { $X$ , Nonce  $Nb$ , Key  $K$ }}  $\in \text{set evs}$ ;  
 $B \notin \text{bad}; \text{evs} \in \text{orb}]$   
 $\implies K \notin \text{range shrK} \wedge (\exists A Na. X = (\text{Crypt} (\text{shrK } A) \{\text{Nonce } Na, \text{Key } K\}))"$   
 <proof>



```

lemma OR4_imp_Gets:
  "[[Says B A {Nonce M, Crypt (shrK A) {Nonce Na, Key K}}] ∈ set evs;
   B ∉ bad; evs ∈ orb]
   ⇒ (∃ Nb. Gets B {Nonce M, Crypt (shrK B) {Crypt (shrK A) {Nonce Na, Key
K}},
                                           Nonce Nb, Key K}} ∈ set evs)"
  <proof>

```

```

lemma A_keydist_to_B:
  "[Says A B {Nonce M, Agent A, Agent B,
    Crypt (shrK A) {Nonce Na, Nonce M, Agent A, Agent B}}] ∈ set evs;

  Gets A {Nonce M, Crypt (shrK A) {Nonce Na, Key K}} ∈ set evs;
  A ∉ bad; B ∉ bad; evs ∈ orb]
  ⇒ Key K ∈ analz (knows B evs)"
  <proof>

```

Other properties as for the original protocol

end

## 13 The Woo-Lam Protocol

```
theory WooLam imports Public begin
```

Simplified version from page 11 of Abadi and Needham (1996). Prudent Engineering Practice for Cryptographic Protocols. IEEE Trans. S.E. 22(1), pages 6-15.

Note: this differs from the Woo-Lam protocol discussed by Lowe (1996): Some New Attacks upon Security Protocols. Computer Security Foundations Workshop

```

inductive_set woolam :: "event list set"
  where

    Nil:  "[ ] ∈ woolam"

    / Fake: "[[evsf ∈ woolam; X ∈ synth (analz (spies evsf))]]
              ⇒ Says Spy B X # evsf ∈ woolam"

    / WL1:  "evs1 ∈ woolam ⇒ Says A B (Agent A) # evs1 ∈ woolam"

    / WL2:  "[[evs2 ∈ woolam; Says A' B (Agent A) ∈ set evs2]]
              ⇒ Says B A (Nonce NB) # evs2 ∈ woolam"

    / WL3:  "[[evs3 ∈ woolam;

```

```

      Says A B (Agent A) ∈ set evs3;
      Says B' A (Nonce NB) ∈ set evs3]]
    ⇒ Says A B (Crypt (shrK A) (Nonce NB)) # evs3 ∈ woolam"

/ WL4: "[[evs4 ∈ woolam;
      Says A' B X          ∈ set evs4;
      Says A'' B (Agent A) ∈ set evs4]]
    ⇒ Says B Server {Agent A, Agent B, X} # evs4 ∈ woolam"

/ WL5: "[[evs5 ∈ woolam;
      Says B' Server {Agent A, Agent B, Crypt (shrK A) (Nonce NB)}
      ∈ set evs5]]
    ⇒ Says Server B (Crypt (shrK B) {Agent A, Nonce NB})
      # evs5 ∈ woolam"

declare Says_imp_knows_Spy [THEN analz.Inj, dest]
declare parts.Body [dest]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]

lemma "∃ NB. ∃ evs ∈ woolam.
      Says Server B (Crypt (shrK B) {Agent A, Nonce NB}) ∈ set evs"
  <proof>

lemma Spy_see_shrK [simp]:
  "evs ∈ woolam ⇒ (Key (shrK A) ∈ parts (spies evs)) = (A ∈ bad)"
  <proof>

lemma Spy_analz_shrK [simp]:
  "evs ∈ woolam ⇒ (Key (shrK A) ∈ analz (spies evs)) = (A ∈ bad)"
  <proof>

lemma Spy_see_shrK_D [dest!]:
  "[[Key (shrK A) ∈ parts (knows Spy evs); evs ∈ woolam]] ⇒ A ∈ bad"
  <proof>

```

**lemma** *NB\_Crypt\_imp\_Alice\_msg*:  
 "[Crypt (shrK A) (Nonce NB) ∈ parts (spies evs);  
 A ∉ bad; evs ∈ woolam]  
 ⇒ ∃ B. Says A B (Crypt (shrK A) (Nonce NB)) ∈ set evs"  
 <proof>

**lemma** *Server\_trusts\_WL4 [dest]*:  
 "[Says B' Server {Agent A, Agent B, Crypt (shrK A) (Nonce NB)}  
 ∈ set evs;  
 A ∉ bad; evs ∈ woolam]  
 ⇒ ∃ B. Says A B (Crypt (shrK A) (Nonce NB)) ∈ set evs"  
 <proof>

**lemma** *Server\_sent\_WL5 [dest]*:  
 "[Says Server B (Crypt (shrK B) {Agent A, NB}) ∈ set evs;  
 evs ∈ woolam]  
 ⇒ ∃ B'. Says B' Server {Agent A, Agent B, Crypt (shrK A) NB}  
 ∈ set evs"  
 <proof>

**lemma** *NB\_Crypt\_imp\_Server\_msg [rule\_format]*:  
 "[Crypt (shrK B) {Agent A, NB} ∈ parts (spies evs);  
 B ∉ bad; evs ∈ woolam]  
 ⇒ Says Server B (Crypt (shrK B) {Agent A, NB}) ∈ set evs"  
 <proof>

**lemma** *B\_trusts\_WL5*:  
 "[Says S B (Crypt (shrK B) {Agent A, Nonce NB}) ∈ set evs;  
 A ∉ bad; B ∉ bad; evs ∈ woolam]  
 ⇒ ∃ B. Says A B (Crypt (shrK A) (Nonce NB)) ∈ set evs"  
 <proof>

**lemma** *B\_said\_WL2*:  
 "[Says B A (Nonce NB) ∈ set evs; B ≠ Spy; evs ∈ woolam]  
 ⇒ ∃ A'. Says A' B (Agent A) ∈ set evs"  
 <proof>

**lemma** "[A ∉ bad; B ≠ Spy; evs ∈ woolam]  
 ⇒ Crypt (shrK A) (Nonce NB) ∈ parts (spies evs) ∧  
 Says B A (Nonce NB) ∈ set evs  
 → Says A B (Crypt (shrK A) (Nonce NB)) ∈ set evs"  
 <proof>

end

## 14 The Otway-Bull Recursive Authentication Protocol

theory *Recur* imports *Public* begin

End marker for message bundles

abbreviation

*END* :: "msg" where  
*"END == Number 0"*

inductive\_set

*respond* :: "event list  $\Rightarrow$  (msg\*msg\*key)set"  
 for *evs* :: "event list"  
 where  
   *One*: "Key *KAB*  $\notin$  used *evs*  
        $\Rightarrow$  (Hash[Key(shrK *A*)] {Agent *A*, Agent *B*, Nonce *NA*, *END*},  
           {Crypt (shrK *A*) {Key *KAB*, Agent *B*, Nonce *NA*}, *END*},  
           *KAB*)  $\in$  respond *evs*"

/ *Cons*: "{(PA, RA, KAB)  $\in$  respond *evs*;  
       Key *KBC*  $\notin$  used *evs*; Key *KBC*  $\notin$  parts {RA};  
       PA = Hash[Key(shrK *A*)] {Agent *A*, Agent *B*, Nonce *NA*, *P*}}  
        $\Rightarrow$  (Hash[Key(shrK *B*)] {Agent *B*, Agent *C*, Nonce *NB*, PA},  
           {Crypt (shrK *B*) {Key *KBC*, Agent *C*, Nonce *NB*},  
           Crypt (shrK *B*) {Key *KAB*, Agent *A*, Nonce *NB*},  
           RA},  
       *KBC*)  
        $\in$  respond *evs*"

inductive\_set

*responses* :: "event list  $\Rightarrow$  msg set"  
 for *evs* :: "event list"  
 where

*Nil*: "*END*  $\in$  responses *evs*"

/ *Cons*: "{RA  $\in$  responses *evs*; Key *KAB*  $\notin$  used *evs*}"  
        $\Rightarrow$  {Crypt (shrK *B*) {Key *KAB*, Agent *A*, Nonce *NB*},  
           RA}  $\in$  responses *evs*"

inductive\_set *recur* :: "event list set"

where

*Nil*: "[ ]  $\in$  recur"

```

/ Fake: "[[evsf ∈ recur; X ∈ synth (analz (knows Spy evsf))]]
        ⇒ Says Spy B X # evsf ∈ recur"

/ RA1: "[[evs1 ∈ recur; Nonce NA ∉ used evs1]]
        ⇒ Says A B (Hash[Key(shrK A)] {Agent A, Agent B, Nonce NA, END})
        # evs1 ∈ recur"

/ RA2: "[[evs2 ∈ recur; Nonce NB ∉ used evs2;
        Says A' B PA ∈ set evs2]]
        ⇒ Says B C (Hash[Key(shrK B)] {Agent B, Agent C, Nonce NB, PA})
        # evs2 ∈ recur"

/ RA3: "[[evs3 ∈ recur; Says B' Server PB ∈ set evs3;
        (PB, RB, K) ∈ respond evs3]]
        ⇒ Says Server B RB # evs3 ∈ recur"

/ RA4: "[[evs4 ∈ recur;
        Says B C {XH, Agent B, Agent C, Nonce NB,
                  XA, Agent A, Agent B, Nonce NA, P} ∈ set evs4;
        Says C' B {Crypt (shrK B) {Key KBC, Agent C, Nonce NB},
                  Crypt (shrK B) {Key KAB, Agent A, Nonce NB},
                  RA} ∈ set evs4]]
        ⇒ Says B A RA # evs4 ∈ recur"

```

```

declare Says_imp_knows_Spy [THEN analz.Inj, dest]
declare parts.Body [dest]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]

```

Simplest case: Alice goes directly to the server

```

lemma "Key K ∉ used [];
        ⇒ ∃ NA. ∃ evs ∈ recur.
           Says Server A {Crypt (shrK A) {Key K, Agent Server, Nonce NA},
                        END} ∈ set evs"

```

⟨proof⟩

Case two: Alice, Bob and the server

```

lemma "[[Key K ∉ used []; Key K' ∉ used []; K ≠ K';
        Nonce NA ∉ used []; Nonce NB ∉ used []; NA < NB]]
        ⇒ ∃ NA. ∃ evs ∈ recur.
           Says B A {Crypt (shrK A) {Key K, Agent B, Nonce NA},
                  END} ∈ set evs"

```

⟨proof⟩

```

lemma "[[Key K ∉ used []; Key K' ∉ used [];
        Key K'' ∉ used []; K ≠ K'; K' ≠ K''; K ≠ K''];

```

```

      Nonce NA  $\notin$  used []; Nonce NB  $\notin$  used []; Nonce NC  $\notin$  used [];
      NA < NB; NB < NC
     $\implies \exists K. \exists NA. \exists \text{evs} \in \text{recur}.$ 
      Says B A  $\{\{\text{Crypt (shrK A) \{Key K, Agent B, Nonce NA\},$ 
        END\}\} \in \text{set evs}"
  <proof>

```

```

lemma respond_imp_not_used: "(PA, RB, KAB)  $\in$  respond evs  $\implies$  Key KAB  $\notin$  used
evs"
  <proof>

```

```

lemma Key_in_parts_respond [rule_format]:
  "[Key K  $\in$  parts {RB}; (PB, RB, K')  $\in$  respond evs]  $\implies$  Key K  $\notin$  used evs"
  <proof>

```

Simple inductive reasoning about responses

```

lemma respond_imp_responses:
  "(PA, RB, KAB)  $\in$  respond evs  $\implies$  RB  $\in$  responses evs"
  <proof>

```

```

lemmas RA2_analz_spies = Says_imp_spies [THEN analz.Inj]

```

```

lemma RA4_analz_spies:
  "Says C' B  $\{\{\text{Crypt K X, X', RA}\} \in \text{set evs} \implies \text{RA} \in \text{analz (spies evs)}"$ 
  <proof>

```

```

lemmas RA2_parts_spies = RA2_analz_spies [THEN analz_into_parts]
lemmas RA4_parts_spies = RA4_analz_spies [THEN analz_into_parts]

```

```

lemma Spy_see_shrK [simp]:
  "evs  $\in$  recur  $\implies$  (Key (shrK A)  $\in$  parts (spies evs)) = (A  $\in$  bad)"
  <proof>

```

```

lemma Spy_analz_shrK [simp]:
  "evs  $\in$  recur  $\implies$  (Key (shrK A)  $\in$  analz (spies evs)) = (A  $\in$  bad)"
  <proof>

```

```

lemma Spy_see_shrK_D [dest!]:
  "[Key (shrK A)  $\in$  parts (knows Spy evs); evs  $\in$  recur]  $\implies$  A  $\in$  bad"
  <proof>

```

```

lemma resp_analz_image_freshK_lemma:
  "[RB ∈ responses evs;
   ∀ K KK. KK ⊆ - (range shrK) ⟶
    (Key K ∈ analz (Key'KK ∪ H)) =
    (K ∈ KK | Key K ∈ analz H)]
  ⟹ ∀ K KK. KK ⊆ - (range shrK) ⟶
    (Key K ∈ analz (insert RB (Key'KK ∪ H))) =
    (K ∈ KK | Key K ∈ analz (insert RB H))"
<proof>

```

Version for the protocol. Proof is easy, thanks to the lemma.

```

lemma raw_analz_image_freshK:
  "evs ∈ recur ⟹
   ∀ K KK. KK ⊆ - (range shrK) ⟶
    (Key K ∈ analz (Key'KK ∪ (spies evs))) =
    (K ∈ KK | Key K ∈ analz (spies evs))"
<proof>

```

```

lemmas resp_analz_image_freshK =
  resp_analz_image_freshK_lemma [OF _ raw_analz_image_freshK]

```

```

lemma analz_insert_freshK:
  "[evs ∈ recur; KAB ∉ range shrK]
  ⟹ (Key K ∈ analz (insert (Key KAB) (spies evs))) =
    (K = KAB | Key K ∈ analz (spies evs))"
<proof>

```

Everything that's hashed is already in past traffic.

```

lemma Hash_imp_body:
  "[Hash {Key(shrK A), X} ∈ parts (spies evs);
   evs ∈ recur; A ∉ bad] ⟹ X ∈ parts (spies evs)"
<proof>

```

```

lemma unique_NA:
  "[Hash {Key(shrK A), Agent A, B, NA, P} ∈ parts (spies evs);
   Hash {Key(shrK A), Agent A, B', NA, P'} ∈ parts (spies evs);
   evs ∈ recur; A ∉ bad]
  ⟹ B=B' ∧ P=P'"
<proof>

```

```

lemma shrK_in_analz_respond [simp]:

```

```

"[[RB ∈ responses evs; evs ∈ recur]]
⇒ (Key (shrK B) ∈ analz (insert RB (spies evs))) = (B ∈ bad)"
⟨proof⟩

```

```

lemma resp_analz_insert_lemma:
  "[[Key K ∈ analz (insert RB H);
    ∀ K KK. KK ⊆ - (range shrK) →
      (Key K ∈ analz (Key KK ∪ H)) =
      (K ∈ KK | Key K ∈ analz H);
    RB ∈ responses evs]]
  ⇒ (Key K ∈ parts{RB} | Key K ∈ analz H)"
⟨proof⟩

```

```

lemmas resp_analz_insert =
  resp_analz_insert_lemma [OF _ raw_analz_image_freshK]

```

The last key returned by respond indeed appears in a certificate

```

lemma respond_certificate:
  "(Hash[Key(shrK A)] {Agent A, B, NA, P}, RA, K) ∈ respond evs
  ⇒ Crypt (shrK A) {Key K, B, NA} ∈ parts {RA}"
⟨proof⟩

```

```

lemma unique_lemma [rule_format]:
  "(PB, RB, KXY) ∈ respond evs ⇒
    ∀ A B N. Crypt (shrK A) {Key K, Agent B, N} ∈ parts {RB} →
    (∀ A' B' N'. Crypt (shrK A') {Key K, Agent B', N'} ∈ parts {RB} →
    (A'=A ∧ B'=B) | (A'=B ∧ B'=A))"
⟨proof⟩

```

```

lemma unique_session_keys:
  "[[Crypt (shrK A) {Key K, Agent B, N} ∈ parts {RB};
    Crypt (shrK A') {Key K, Agent B', N'} ∈ parts {RB};
    (PB, RB, KXY) ∈ respond evs]]
  ⇒ (A'=A ∧ B'=B) | (A'=B ∧ B'=A)"
⟨proof⟩

```

```

lemma respond_Spy_not_see_session_key [rule_format]:
  "[[ (PB, RB, KAB) ∈ respond evs; evs ∈ recur]]
  ⇒ ∀ A A' N. A ∉ bad ∧ A' ∉ bad →
    Crypt (shrK A) {Key K, Agent A', N} ∈ parts{RB} →
    Key K ∉ analz (insert RB (spies evs))"
⟨proof⟩

```

```

lemma Spy_not_see_session_key:
  "[[Crypt (shrK A) {Key K, Agent A', N} ∈ parts (spies evs);
    A ∉ bad; A' ∉ bad; evs ∈ recur]]
  ⇒ Key K ∉ analz (spies evs)"
⟨proof⟩

```



The response never contains Hashes

**lemma** *Hash\_in\_parts\_respond*:  
 "[[Hash {Key (shrK B), M} ∈ parts (insert RB H);  
 (PB, RB, K) ∈ respond evs]]  
 ⇒ Hash {Key (shrK B), M} ∈ parts H"  
 <proof>

Only RA1 or RA2 can have caused such a part of a message to appear. This result is of no use to B, who cannot verify the Hash. Moreover, it can say nothing about how recent A's message is. It might later be used to prove B's presence to A at the run's conclusion.

**lemma** *Hash\_auth\_sender [rule\_format]*:  
 "[[Hash {Key (shrK A), Agent A, Agent B, NA, P} ∈ parts (spies evs);  
 A ∉ bad; evs ∈ recur]]  
 ⇒ Says A B (Hash [Key (shrK A)] {Agent A, Agent B, NA, P}) ∈ set evs"  
 <proof>

Certificates can only originate with the Server.

**lemma** *Cert\_imp\_Server\_msg*:  
 "[[Crypt (shrK A) Y ∈ parts (spies evs);  
 A ∉ bad; evs ∈ recur]]  
 ⇒ ∃ C RC. Says Server C RC ∈ set evs ∧  
 Crypt (shrK A) Y ∈ parts {RC}"  
 <proof>

**end**

## 15 The Yahalom Protocol

**theory** *Yahalom* **imports** *Public* **begin**

From page 257 of Burrows, Abadi and Needham (1989). A Logic of Authentication. Proc. Royal Soc. 426

This theory has the prototypical example of a secrecy relation, KeyCryptNonce.

**inductive\_set** *yahalom* :: "event list set"  
**where**  
  
 Nil: "[ ] ∈ yahalom"  
  
 / Fake: "[[evsf ∈ yahalom; X ∈ synth (analz (knows Spy evsf))]]  
 ⇒ Says Spy B X # evsf ∈ yahalom"  
  
 / Reception: "[[evsr ∈ yahalom; Says A B X ∈ set evsr]]  
 ⇒ Gets B X # evsr ∈ yahalom"  
  
 / YM1: "[[evs1 ∈ yahalom; Nonce NA ∉ used evs1]]  
 ⇒ Says A B {Agent A, Nonce NA} # evs1 ∈ yahalom"

```

/ YM2: "[[evs2 ∈ yahalom; Nonce NB ∉ used evs2;
        Gets B {Agent A, Nonce NA} ∈ set evs2]]
        ⇒ Says B Server
           {Agent B, Crypt (shrK B) {Agent A, Nonce NA, Nonce NB}}
           # evs2 ∈ yahalom"

/ YM3: "[[evs3 ∈ yahalom; Key KAB ∉ used evs3; KAB ∈ symKeys;
        Gets Server
           {Agent B, Crypt (shrK B) {Agent A, Nonce NA, Nonce NB}}
           ∈ set evs3]]
        ⇒ Says Server A
           {Crypt (shrK A) {Agent B, Key KAB, Nonce NA, Nonce NB},
            Crypt (shrK B) {Agent A, Key KAB}}
           # evs3 ∈ yahalom"

/ YM4:
  — Alice receives the Server's (?) message, checks her Nonce, and uses the
  new session key to send Bob his Nonce. The premise  $A \neq \text{Server}$  is needed for
  Says_Server_not_range. Alice can check that K is symmetric by its length.
  "[[evs4 ∈ yahalom; A ≠ Server; K ∈ symKeys;
    Gets A {Crypt(shrK A) {Agent B, Key K, Nonce NA, Nonce NB}, X}
           ∈ set evs4;
    Says A B {Agent A, Nonce NA} ∈ set evs4]]
    ⇒ Says A B {X, Crypt K (Nonce NB)} # evs4 ∈ yahalom"

/ Oops: "[[evso ∈ yahalom;
        Says Server A {Crypt (shrK A)
                       {Agent B, Key K, Nonce NA, Nonce NB},
                       X} ∈ set evso]]
        ⇒ Notes Spy {Nonce NA, Nonce NB, Key K} # evso ∈ yahalom"

definition KeyWithNonce :: "[key, nat, event list] ⇒ bool" where
  "KeyWithNonce K NB evs ==
    ∃ A B na X.
      Says Server A {Crypt (shrK A) {Agent B, Key K, na, Nonce NB}, X}
      ∈ set evs"

```

```

declare Says_imp_analz_Spy [dest]
declare parts.Body [dest]
declare Fake_parts_insert_in_Un [dest]
declare analz_into_parts [dest]

```

A "possibility property": there are traces that reach the end

```

lemma "[A ≠ Server; K ∈ symKeys; Key K ∉ used []]
        ⇒ ∃ X NB. ∃ evs ∈ yahalom.
           Says A B {X, Crypt K (Nonce NB)} ∈ set evs"
<proof>

```

## 15.1 Regularity Lemmas for Yahalom

**lemma** *Gets\_imp\_Says*:

"[[Gets B X ∈ set evs; evs ∈ yahalom]] ⇒ ∃ A. Says A B X ∈ set evs"  
 <proof>

Must be proved separately for each protocol

**lemma** *Gets\_imp\_knows\_Spy*:

"[[Gets B X ∈ set evs; evs ∈ yahalom]] ⇒ X ∈ knows Spy evs"  
 <proof>

**lemmas** *Gets\_imp\_analz\_Spy* = *Gets\_imp\_knows\_Spy* [THEN analz.Inj]  
**declare** *Gets\_imp\_analz\_Spy* [dest]

Lets us treat YM4 using a similar argument as for the Fake case.

**lemma** *YM4\_analz\_knows\_Spy*:

"[[Gets A {Crypt (shrK A) Y, X} ∈ set evs; evs ∈ yahalom]]  
 ⇒ X ∈ analz (knows Spy evs)"  
 <proof>

**lemmas** *YM4\_parts\_knows\_Spy* =  
*YM4\_analz\_knows\_Spy* [THEN analz\_into\_parts]

For Oops

**lemma** *YM4\_Key\_parts\_knows\_Spy*:

"Says Server A {Crypt (shrK A) {B,K,NA,NB}, X} ∈ set evs  
 ⇒ K ∈ parts (knows Spy evs)"  
 <proof>

Theorems of the form  $X \notin \text{parts (knows Spy evs)}$  imply that NOBODY sends messages containing X!

Spy never sees a good agent's shared key!

**lemma** *Spy\_see\_shrK [simp]*:

"evs ∈ yahalom ⇒ (Key (shrK A) ∈ parts (knows Spy evs)) = (A ∈ bad)"  
 <proof>

**lemma** *Spy\_analz\_shrK [simp]*:

"evs ∈ yahalom ⇒ (Key (shrK A) ∈ analz (knows Spy evs)) = (A ∈ bad)"  
 <proof>

**lemma** *Spy\_see\_shrK\_D [dest!]*:

"[Key (shrK A) ∈ parts (knows Spy evs); evs ∈ yahalom] ⇒ A ∈ bad"  
 <proof>

Nobody can have used non-existent keys! Needed to apply *analz\_insert\_Key*

**lemma** *new\_keys\_not\_used [simp]*:

"[Key K ∉ used evs; K ∈ symKeys; evs ∈ yahalom]  
 ⇒ K ∉ keysFor (parts (spies evs))"  
 <proof>

Earlier, all protocol proofs declared this theorem. But only a few proofs need it, e.g. Yahalom and Kerberos IV.

**lemma** *new\_keys\_not\_analz*:  
 "[ $K \in \text{symKeys}; \text{evs} \in \text{yahalom}; \text{Key } K \notin \text{used evs}$ ]  
 $\implies K \notin \text{keysFor } (\text{analz } (\text{knows Spy evs}))$ "  
 <proof>

Describes the form of  $K$  when the Server sends this message. Useful for Oops as well as main secrecy property.

**lemma** *Says\_Server\_not\_range* [simp]:  
 "[ $\text{Says Server } A \ \{\{\text{Crypt } (\text{shrK } A) \ \{\text{Agent } B, \text{Key } K, \text{na}, \text{nb}\}, X\}\}$   
 $\in \text{set evs}; \text{evs} \in \text{yahalom}$ ]  
 $\implies K \notin \text{range shrK}$ "  
 <proof>

## 15.2 Secrecy Theorems

Session keys are not used to encrypt other session keys

**lemma** *analz\_image\_freshK* [rule\_format]:  
 " $\text{evs} \in \text{yahalom} \implies$   
 $\forall K \text{ KK}. \text{KK} \subseteq - (\text{range shrK}) \longrightarrow$   
 $(\text{Key } K \in \text{analz } (\text{Key 'KK} \cup (\text{knows Spy evs}))) =$   
 $(K \in \text{KK} \mid \text{Key } K \in \text{analz } (\text{knows Spy evs}))$ "  
 <proof>

**lemma** *analz\_insert\_freshK*:  
 " $[\text{evs} \in \text{yahalom}; \text{KAB} \notin \text{range shrK}] \implies$   
 $(\text{Key } K \in \text{analz } (\text{insert } (\text{Key KAB}) (\text{knows Spy evs}))) =$   
 $(K = \text{KAB} \mid \text{Key } K \in \text{analz } (\text{knows Spy evs}))$ "  
 <proof>

The Key  $K$  uniquely identifies the Server's message.

**lemma** *unique\_session\_keys*:  
 "[ $\text{Says Server } A$   
 $\{\{\text{Crypt } (\text{shrK } A) \ \{\text{Agent } B, \text{Key } K, \text{na}, \text{nb}\}, X\}\} \in \text{set evs};$   
 $\text{Says Server } A'$   
 $\{\{\text{Crypt } (\text{shrK } A') \ \{\text{Agent } B', \text{Key } K, \text{na}', \text{nb}'\}, X'\}\} \in \text{set evs};$   
 $\text{evs} \in \text{yahalom}$ ]  
 $\implies A=A' \wedge B=B' \wedge \text{na}=\text{na}' \wedge \text{nb}=\text{nb}'$ "  
 <proof>

Crucial secrecy property: Spy does not see the keys sent in msg YM3

**lemma** *secrecy\_lemma*:  
 "[ $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{yahalom}$ ]  
 $\implies \text{Says Server } A$   
 $\{\{\text{Crypt } (\text{shrK } A) \ \{\text{Agent } B, \text{Key } K, \text{na}, \text{nb}\},$   
 $\text{Crypt } (\text{shrK } B) \ \{\text{Agent } A, \text{Key } K\}\}$   
 $\in \text{set evs} \longrightarrow$   
 $\text{Notes Spy } \{\text{na}, \text{nb}, \text{Key } K\} \notin \text{set evs} \longrightarrow$   
 $\text{Key } K \notin \text{analz } (\text{knows Spy evs})$ "  
 <proof>

Final version

**lemma** *Spy\_not\_see\_encrypted\_key*:

```

"[[Says Server A
  {Crypt (shrK A) {Agent B, Key K, na, nb}},
   Crypt (shrK B) {Agent A, Key K}}
 ∈ set evs;
 Notes Spy {na, nb, Key K} ∉ set evs;
 A ∉ bad; B ∉ bad; evs ∈ yahalom]]
⇒ Key K ∉ analz (knows Spy evs)"
<proof>

```

### 15.2.1 Security Guarantee for A upon receiving YM3

If the encrypted message appears then it originated with the Server

**lemma** *A\_trusts\_YM3*:

```

"[[Crypt (shrK A) {Agent B, Key K, na, nb} ∈ parts (knows Spy evs);
  A ∉ bad; evs ∈ yahalom]]
⇒ Says Server A
  {Crypt (shrK A) {Agent B, Key K, na, nb}},
  Crypt (shrK B) {Agent A, Key K}}
 ∈ set evs"
<proof>

```

The obvious combination of *A\_trusts\_YM3* with *Spy\_not\_see\_encrypted\_key*

**lemma** *A\_gets\_good\_key*:

```

"[[Crypt (shrK A) {Agent B, Key K, na, nb} ∈ parts (knows Spy evs);
  Notes Spy {na, nb, Key K} ∉ set evs;
  A ∉ bad; B ∉ bad; evs ∈ yahalom]]
⇒ Key K ∉ analz (knows Spy evs)"
<proof>

```

### 15.2.2 Security Guarantees for B upon receiving YM4

B knows, by the first part of A's message, that the Server distributed the key for A and B. But this part says nothing about nonces.

**lemma** *B\_trusts\_YM4\_shrK*:

```

"[[Crypt (shrK B) {Agent A, Key K} ∈ parts (knows Spy evs);
  B ∉ bad; evs ∈ yahalom]]
⇒ ∃ NA NB. Says Server A
  {Crypt (shrK A) {Agent B, Key K,
    Nonce NA, Nonce NB}},
  Crypt (shrK B) {Agent A, Key K}}
 ∈ set evs"
<proof>

```

B knows, by the second part of A's message, that the Server distributed the key quoting nonce NB. This part says nothing about agent names. Secrecy of NB is crucial. Note that *Nonce NB ∉ analz (knows Spy evs)* must be the FIRST antecedent of the induction formula.

**lemma** *B\_trusts\_YM4\_newK [rule\_format]*:

```

"[[Crypt K (Nonce NB) ∈ parts (knows Spy evs);
  Nonce NB ∉ analz (knows Spy evs); evs ∈ yahalom]]
⇒ ∃ A B NA. Says Server A
  {Crypt (shrK A) {Agent B, Key K, Nonce NA, Nonce NB}},

```

$$\text{Crypt } (\text{shrK } B) \{ \text{Agent } A, \text{Key } K \}$$

$$\in \text{set evs}$$

$\langle \text{proof} \rangle$

### 15.2.3 Towards proving secrecy of Nonce NB

Lemmas about the predicate `KeyWithNonce`

**lemma** `KeyWithNonceI:`

$$\text{"Says Server } A$$

$$\quad \{ \text{Crypt } (\text{shrK } A) \{ \text{Agent } B, \text{Key } K, na, \text{Nonce } NB \}, X \}$$

$$\in \text{set evs} \implies \text{KeyWithNonce } K \text{ NB evs"}$$

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

**lemma** `KeyWithNonce_Says [simp]:`

$$\text{"KeyWithNonce } K \text{ NB (Says } S \text{ A } X \# \text{ evs)} =$$

$$\quad (\text{Server} = S \wedge$$

$$\quad (\exists B \text{ n } X'. X = \{ \text{Crypt } (\text{shrK } A) \{ \text{Agent } B, \text{Key } K, n, \text{Nonce } NB \}, X' \})$$

$$\quad | \text{KeyWithNonce } K \text{ NB evs)"}$$

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

**lemma** `KeyWithNonce_Notes [simp]:`

$$\text{"KeyWithNonce } K \text{ NB (Notes } A \text{ X \# evs)} = \text{KeyWithNonce } K \text{ NB evs"}$$

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

**lemma** `KeyWithNonce_Gets [simp]:`

$$\text{"KeyWithNonce } K \text{ NB (Gets } A \text{ X \# evs)} = \text{KeyWithNonce } K \text{ NB evs"}$$

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

A fresh key cannot be associated with any nonce (with respect to a given trace).

**lemma** `fresh_not_KeyWithNonce:`

$$\text{"Key } K \notin \text{used evs} \implies \neg \text{KeyWithNonce } K \text{ NB evs"}$$

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

The Server message associates  $K$  with  $NB'$  and therefore not with any other nonce  $NB$ .

**lemma** `Says_Server_KeyWithNonce:`

$$\text{"[Says Server } A \{ \text{Crypt } (\text{shrK } A) \{ \text{Agent } B, \text{Key } K, na, \text{Nonce } NB' \}, X \}$$

$$\in \text{set evs};$$

$$\quad NB \neq NB'; \text{ evs} \in \text{yahalom}]$$

$$\implies \neg \text{KeyWithNonce } K \text{ NB evs"}$$

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

The only nonces that can be found with the help of session keys are those distributed as nonce  $NB$  by the Server. The form of the theorem recalls `analz_image_freshK`, but it is much more complicated.

As with `analz_image_freshK`, we take some pains to express the property as a logical equivalence so that the simplifier can apply it.

**lemma** `Nonce_secrecy_lemma:`

$$P \longrightarrow (X \in \text{analz } (G \cup H)) \longrightarrow (X \in \text{analz } H) \implies$$

$$P \longrightarrow (X \in \text{analz } (G \cup H)) = (X \in \text{analz } H)"$$

$\langle \text{proof} \rangle$

**lemma** *Nonce\_secrecy*:

" $\text{evs} \in \text{yahalom} \implies$   
 $(\forall KK. KK \subseteq - (\text{range shrK}) \longrightarrow$   
 $(\forall K \in KK. K \in \text{symKeys} \longrightarrow \neg \text{KeyWithNonce } K \text{ NB evs}) \longrightarrow$   
 $(\text{Nonce NB} \in \text{analz } (\text{Key}'KK \cup (\text{knows Spy evs}))) =$   
 $(\text{Nonce NB} \in \text{analz } (\text{knows Spy evs}))$ "

$\langle \text{proof} \rangle$

Version required below: if NB can be decrypted using a session key then it was distributed with that key. The more general form above is required for the induction to carry through.

**lemma** *single\_Nonce\_secrecy*:

" $\llbracket \text{Says Server A}$   
 $\quad \llbracket \text{Crypt } (\text{shrK A}) \llbracket \text{Agent B, Key KAB, na, Nonce NB}' \rrbracket, X \rrbracket$   
 $\in \text{set evs};$   
 $\text{NB} \neq \text{NB}'; \text{KAB} \notin \text{range shrK}; \text{evs} \in \text{yahalom} \rrbracket$   
 $\implies (\text{Nonce NB} \in \text{analz } (\text{insert } (\text{Key KAB}) (\text{knows Spy evs}))) =$   
 $(\text{Nonce NB} \in \text{analz } (\text{knows Spy evs}))$ "

$\langle \text{proof} \rangle$

#### 15.2.4 The Nonce NB uniquely identifies B's message.

**lemma** *unique\_NB*:

" $\llbracket \text{Crypt } (\text{shrK B}) \llbracket \text{Agent A, Nonce NA, nb} \rrbracket \in \text{parts } (\text{knows Spy evs});$   
 $\text{Crypt } (\text{shrK B}') \llbracket \text{Agent A}', \text{Nonce NA}', \text{nb} \rrbracket \in \text{parts } (\text{knows Spy evs});$   
 $\text{evs} \in \text{yahalom}; \text{B} \notin \text{bad}; \text{B}' \notin \text{bad} \rrbracket$   
 $\implies \text{NA}' = \text{NA} \wedge \text{A}' = \text{A} \wedge \text{B}' = \text{B}$ "

$\langle \text{proof} \rangle$

Variant useful for proving secrecy of NB. Because nb is assumed to be secret, we no longer must assume B, B' not bad.

**lemma** *Says\_unique\_NB*:

" $\llbracket \text{Says C S} \quad \llbracket X, \text{Crypt } (\text{shrK B}) \llbracket \text{Agent A, Nonce NA, nb} \rrbracket \rrbracket$   
 $\in \text{set evs};$   
 $\text{Gets S}' \llbracket X', \text{Crypt } (\text{shrK B}') \llbracket \text{Agent A}', \text{Nonce NA}', \text{nb} \rrbracket \rrbracket$   
 $\in \text{set evs};$   
 $\text{nb} \notin \text{analz } (\text{knows Spy evs}); \text{evs} \in \text{yahalom} \rrbracket$   
 $\implies \text{NA}' = \text{NA} \wedge \text{A}' = \text{A} \wedge \text{B}' = \text{B}$ "

$\langle \text{proof} \rangle$

#### 15.2.5 A nonce value is never used both as NA and as NB

**lemma** *no\_nonce\_YM1\_YM2*:

" $\llbracket \text{Crypt } (\text{shrK B}') \llbracket \text{Agent A}', \text{Nonce NB, nb}' \rrbracket \in \text{parts}(\text{knows Spy evs});$   
 $\text{Nonce NB} \notin \text{analz } (\text{knows Spy evs}); \text{evs} \in \text{yahalom} \rrbracket$   
 $\implies \text{Crypt } (\text{shrK B}) \llbracket \text{Agent A, na, Nonce NB} \rrbracket \notin \text{parts}(\text{knows Spy evs})$ "

$\langle \text{proof} \rangle$

The Server sends YM3 only in response to YM2.

**lemma** *Says\_Server\_imp\_YM2*:

" $\llbracket \text{Says Server A} \quad \llbracket \text{Crypt } (\text{shrK A}) \llbracket \text{Agent B, k, na, nb} \rrbracket, X \rrbracket \in \text{set evs};$

$$\begin{aligned} & \text{evs} \in \text{yahalom} \\ \implies & \text{Gets Server } \{\!\{ \text{Agent B, Crypt (shrK B) } \{\!\{ \text{Agent A, na, nb} \}\!\} \\ & \in \text{set evs} \}\!\} \\ \langle \text{proof} \rangle & \end{aligned}$$

A vital theorem for B, that nonce NB remains secure from the Spy.

**theorem** *Spy\_not\_see\_NB* :

$$\begin{aligned} & \text{"}\llbracket \text{Says B Server} \\ & \quad \{\!\{ \text{Agent B, Crypt (shrK B) } \{\!\{ \text{Agent A, Nonce NA, Nonce NB} \}\!\} \\ & \quad \in \text{set evs}; \\ & \quad (\forall k. \text{Notes Spy } \{\!\{ \text{Nonce NA, Nonce NB, k} \}\!\} \notin \text{set evs}); \\ & \quad A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{yahalom} \rrbracket \\ \implies & \text{Nonce NB} \notin \text{analz (knows Spy evs)} \text{"} \\ \langle \text{proof} \rangle & \end{aligned}$$

B's session key guarantee from YM4. The two certificates contribute to a single conclusion about the Server's message. Note that the "Notes Spy" assumption must quantify over  $\forall$  POSSIBLE keys instead of our particular K. If this run is broken and the spy substitutes a certificate containing an old key, B has no means of telling.

**lemma** *B\_trusts\_YM4*:

$$\begin{aligned} & \text{"}\llbracket \text{Gets B } \{\!\{ \text{Crypt (shrK B) } \{\!\{ \text{Agent A, Key K} \}\!\}, \\ & \quad \text{Crypt K (Nonce NB)} \}\!\} \in \text{set evs}; \\ & \quad \text{Says B Server} \\ & \quad \{\!\{ \text{Agent B, Crypt (shrK B) } \{\!\{ \text{Agent A, Nonce NA, Nonce NB} \}\!\} \\ & \quad \in \text{set evs}; \\ & \quad \forall k. \text{Notes Spy } \{\!\{ \text{Nonce NA, Nonce NB, k} \}\!\} \notin \text{set evs}; \\ & \quad A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{yahalom} \rrbracket \\ \implies & \text{Says Server A} \\ & \quad \{\!\{ \text{Crypt (shrK A) } \{\!\{ \text{Agent B, Key K,} \\ & \quad \quad \text{Nonce NA, Nonce NB} \}\!\}, \\ & \quad \text{Crypt (shrK B) } \{\!\{ \text{Agent A, Key K} \}\!\} \\ & \quad \in \text{set evs} \text{"} \\ \langle \text{proof} \rangle & \end{aligned}$$

The obvious combination of *B\_trusts\_YM4* with *Spy\_not\_see\_encrypted\_key*

**lemma** *B\_gets\_good\_key*:

$$\begin{aligned} & \text{"}\llbracket \text{Gets B } \{\!\{ \text{Crypt (shrK B) } \{\!\{ \text{Agent A, Key K} \}\!\}, \\ & \quad \text{Crypt K (Nonce NB)} \}\!\} \in \text{set evs}; \\ & \quad \text{Says B Server} \\ & \quad \{\!\{ \text{Agent B, Crypt (shrK B) } \{\!\{ \text{Agent A, Nonce NA, Nonce NB} \}\!\} \\ & \quad \in \text{set evs}; \\ & \quad \forall k. \text{Notes Spy } \{\!\{ \text{Nonce NA, Nonce NB, k} \}\!\} \notin \text{set evs}; \\ & \quad A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{yahalom} \rrbracket \\ \implies & \text{Key K} \notin \text{analz (knows Spy evs)} \text{"} \\ \langle \text{proof} \rangle & \end{aligned}$$

### 15.3 Authenticating B to A

The encryption in message YM2 tells us it cannot be faked.

**lemma** *B\_Said\_YM2 [rule\_format]*:

$$\text{"}\llbracket \text{Crypt (shrK B) } \{\!\{ \text{Agent A, Nonce NA, nb} \}\!\} \in \text{parts (knows Spy evs)} \rrbracket$$



```

    evs ∈ yahalom
  ⇒ B ∉ bad ⇒
    Says B Server {Agent B, Crypt (shrK B) {Agent A, Nonce NA, nb}}
    ∈ set evs"

```

⟨proof⟩

If the server sends YM3 then B sent YM2

**lemma** *YM3\_auth\_B\_to\_A\_lemma*:

```

  "[Says Server A {Crypt (shrK A) {Agent B, Key K, Nonce NA, nb}}, X}
   ∈ set evs; evs ∈ yahalom]
  ⇒ B ∉ bad ⇒
    Says B Server {Agent B, Crypt (shrK B) {Agent A, Nonce NA, nb}}
    ∈ set evs"

```

⟨proof⟩

If A receives YM3 then B has used nonce NA (and therefore is alive)

**theorem** *YM3\_auth\_B\_to\_A*:

```

  "[Gets A {Crypt (shrK A) {Agent B, Key K, Nonce NA, nb}}, X}
   ∈ set evs;
   A ∉ bad; B ∉ bad; evs ∈ yahalom]
  ⇒ Says B Server {Agent B, Crypt (shrK B) {Agent A, Nonce NA, nb}}
   ∈ set evs"

```

⟨proof⟩

## 15.4 Authenticating A to B using the certificate *Crypt K (Nonce NB)*

Assuming the session key is secure, if both certificates are present then A has said NB. We can't be sure about the rest of A's message, but only NB matters for freshness.

**theorem** *A\_Said\_YM3\_lemma [rule\_format]*:

```

  "evs ∈ yahalom
  ⇒ Key K ∉ analz (knows Spy evs) ⇒
    Crypt K (Nonce NB) ∈ parts (knows Spy evs) ⇒
    Crypt (shrK B) {Agent A, Key K} ∈ parts (knows Spy evs) ⇒
    B ∉ bad ⇒
    (∃ X. Says A B {X, Crypt K (Nonce NB)} ∈ set evs)"

```

⟨proof⟩

If B receives YM4 then A has used nonce NB (and therefore is alive). Moreover, A associates K with NB (thus is talking about the same run). Other premises guarantee secrecy of K.

**theorem** *YM4\_imp\_A\_Said\_YM3 [rule\_format]*:

```

  "[Gets B {Crypt (shrK B) {Agent A, Key K},
    Crypt K (Nonce NB)} ∈ set evs;
   Says B Server
    {Agent B, Crypt (shrK B) {Agent A, Nonce NA, Nonce NB}}
    ∈ set evs;
   (∀ NA k. Notes Spy {Nonce NA, Nonce NB, k} ∉ set evs);
   A ∉ bad; B ∉ bad; evs ∈ yahalom]
  ⇒ ∃ X. Says A B {X, Crypt K (Nonce NB)} ∈ set evs"

```

⟨proof⟩

end

## 16 The Yahalom Protocol, Variant 2

theory Yahalom2 imports Public begin

This version trades encryption of NB for additional explicitness in YM3. Also in YM3, care is taken to make the two certificates distinct.

From page 259 of Burrows, Abadi and Needham (1989). A Logic of Authentication. Proc. Royal Soc. 426

This theory has the prototypical example of a secrecy relation, KeyCryptNonce.

```

inductive_set yahalom :: "event list set"
  where

    Nil: "[] ∈ yahalom"

    / Fake: "[evsf ∈ yahalom; X ∈ synth (analz (knows Spy evsf))]"
      ⇒ "Says Spy B X # evsf ∈ yahalom"

    / Reception: "[evsr ∈ yahalom; Says A B X ∈ set evsr]"
      ⇒ "Gets B X # evsr ∈ yahalom"

    / YM1: "[evs1 ∈ yahalom; Nonce NA ∉ used evs1]"
      ⇒ "Says A B {Agent A, Nonce NA} # evs1 ∈ yahalom"

    / YM2: "[evs2 ∈ yahalom; Nonce NB ∉ used evs2;
      Gets B {Agent A, Nonce NA} ∈ set evs2]"
      ⇒ "Says B Server
        {Agent B, Nonce NB, Crypt (shrK B) {Agent A, Nonce NA}}
        # evs2 ∈ yahalom"

    / YM3: "[evs3 ∈ yahalom; Key KAB ∉ used evs3;
      Gets Server {Agent B, Nonce NB,
        Crypt (shrK B) {Agent A, Nonce NA}}
        ∈ set evs3]"
      ⇒ "Says Server A
        {Nonce NB,
        Crypt (shrK A) {Agent B, Key KAB, Nonce NA},
        Crypt (shrK B) {Agent A, Agent B, Key KAB, Nonce NB}}
        # evs3 ∈ yahalom"

    / YM4: "[evs4 ∈ yahalom;
      Gets A {Nonce NB, Crypt (shrK A) {Agent B, Key K, Nonce NA},
        X} ∈ set evs4;
      Says A B {Agent A, Nonce NA} ∈ set evs4]"

```

$\implies \text{Says } A \ B \ \{\{X, \text{Crypt } K \ (\text{Nonce } NB)\}\} \ \# \ \text{evs4} \in \text{yahalom}$

```

/ Oops: "[[evso ∈ yahalom;
        Says Server A {Nonce NB,
                      Crypt (shrK A) {Agent B, Key K, Nonce NA}},
        X} ∈ set evso]]
   $\implies \text{Notes Spy } \{\{\text{Nonce } NA, \text{Nonce } NB, \text{Key } K\}\} \ \# \ \text{evso} \in \text{yahalom}$ 

```

```

declare Says_imp_knows_Spy [THEN analz.Inj, dest]
declare parts.Body [dest]
declare Fake_parts_insert_in_Un [dest]
declare analz_into_parts [dest]

```

A "possibility property": there are traces that reach the end

```

lemma "Key K ∉ used []
   $\implies \exists X \ NB. \exists \text{evs} \in \text{yahalom}. \text{Says } A \ B \ \{\{X, \text{Crypt } K \ (\text{Nonce } NB)\}\} \in \text{set evs}$ "
<proof>

```

```

lemma Gets_imp_Says:
  "[[Gets B X ∈ set evs; evs ∈ yahalom]]  $\implies \exists A. \text{Says } A \ B \ X \in \text{set evs}$ "
<proof>

```

Must be proved separately for each protocol

```

lemma Gets_imp_knows_Spy:
  "[[Gets B X ∈ set evs; evs ∈ yahalom]]  $\implies X \in \text{knows Spy evs}$ "
<proof>

```

```

declare Gets_imp_knows_Spy [THEN analz.Inj, dest]

```

## 16.1 Inductive Proofs

Result for reasoning about the encrypted portion of messages. Lets us treat YM4 using a similar argument as for the Fake case.

```

lemma YM4_analz_knows_Spy:
  "[[Gets A {NB, Crypt (shrK A) Y, X} ∈ set evs; evs ∈ yahalom]]
   $\implies X \in \text{analz } (\text{knows Spy evs})$ "
<proof>

```

```

lemmas YM4_parts_knows_Spy =
  YM4_analz_knows_Spy [THEN analz_into_parts]

```

Spy never sees a good agent's shared key!

```

lemma Spy_see_shrK [simp]:
  " $\text{evs} \in \text{yahalom} \implies (\text{Key } (\text{shrK } A) \in \text{parts } (\text{knows Spy evs})) = (A \in \text{bad})$ "
<proof>

```

```

lemma Spy_analz_shrK [simp]:
  " $\text{evs} \in \text{yahalom} \implies (\text{Key } (\text{shrK } A) \in \text{analz } (\text{knows Spy evs})) = (A \in \text{bad})$ "
<proof>

```

**lemma** *Spy\_see\_shrK\_D [dest!]*:  
 "[Key (shrK A) ∈ parts (knows Spy evs); evs ∈ yahalom] ⇒ A ∈ bad"  
 <proof>

Nobody can have used non-existent keys! Needed to apply *analz\_insert\_Key*

**lemma** *new\_keys\_not\_used [simp]*:  
 "[Key K ∉ used evs; K ∈ symKeys; evs ∈ yahalom]  
 ⇒ K ∉ keysFor (parts (spies evs))"  
 <proof>

Describes the form of K when the Server sends this message. Useful for Oops as well as main secrecy property.

**lemma** *Says\_Server\_message\_form*:  
 "[Says Server A {nb', Crypt (shrK A) {Agent B, Key K, na}}, X}  
 ∈ set evs; evs ∈ yahalom]  
 ⇒ K ∉ range shrK"  
 <proof>

**lemma** *analz\_image\_freshK [rule\_format]*:  
 "evs ∈ yahalom ⇒  
 ∀ K KK. KK ⊆ - (range shrK) →  
 (Key K ∈ analz (Key 'KK ∪ (knows Spy evs))) =  
 (K ∈ KK | Key K ∈ analz (knows Spy evs))"  
 <proof>

**lemma** *analz\_insert\_freshK*:  
 "[evs ∈ yahalom; KAB ∉ range shrK] ⇒  
 (Key K ∈ analz (insert (Key KAB) (knows Spy evs))) =  
 (K = KAB | Key K ∈ analz (knows Spy evs))"  
 <proof>

The Key K uniquely identifies the Server's message

**lemma** *unique\_session\_keys*:  
 "[Says Server A  
 {nb, Crypt (shrK A) {Agent B, Key K, na}}, X} ∈ set evs;  
 Says Server A'  
 {nb', Crypt (shrK A') {Agent B', Key K, na'}}, X'} ∈ set evs;  
 evs ∈ yahalom]  
 ⇒ A=A' ∧ B=B' ∧ na=na' ∧ nb=nb'"  
 <proof>

## 16.2 Crucial Secrecy Property: Spy Does Not See Key $K_{AB}$

**lemma** *secrecy\_lemma*:  
 "[A ∉ bad; B ∉ bad; evs ∈ yahalom]  
 ⇒ Says Server A  
 {nb, Crypt (shrK A) {Agent B, Key K, na}},

$\text{Crypt } (\text{shrK } B) \{ \text{Agent } A, \text{Agent } B, \text{Key } K, \text{nb} \}$   
 $\in \text{set evs} \longrightarrow$   
 $\text{Notes Spy } \{ \text{na}, \text{nb}, \text{Key } K \} \notin \text{set evs} \longrightarrow$   
 $\text{Key } K \notin \text{analz } (\text{knows Spy evs})"$

*<proof>*

Final version

**lemma** *Spy\_not\_see\_encrypted\_key*:

$"\llbracket \text{Says Server } A$   
 $\quad \{ \text{nb}, \text{Crypt } (\text{shrK } A) \{ \text{Agent } B, \text{Key } K, \text{na} \},$   
 $\quad \text{Crypt } (\text{shrK } B) \{ \text{Agent } A, \text{Agent } B, \text{Key } K, \text{nb} \} \}$   
 $\in \text{set evs};$   
 $\text{Notes Spy } \{ \text{na}, \text{nb}, \text{Key } K \} \notin \text{set evs};$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{yahalom} \rrbracket$   
 $\implies \text{Key } K \notin \text{analz } (\text{knows Spy evs})"$

*<proof>*

This form is an immediate consequence of the previous result. It is similar to the assertions established by other methods. It is equivalent to the previous result in that the Spy already has *analz* and *synth* at his disposal. However, the conclusion  $\text{Key } K \notin \text{knows Spy evs}$  appears not to be inductive: all the cases other than Fake are trivial, while Fake requires  $\text{Key } K \notin \text{analz } (\text{knows Spy evs})$ .

**lemma** *Spy\_not\_know\_encrypted\_key*:

$"\llbracket \text{Says Server } A$   
 $\quad \{ \text{nb}, \text{Crypt } (\text{shrK } A) \{ \text{Agent } B, \text{Key } K, \text{na} \},$   
 $\quad \text{Crypt } (\text{shrK } B) \{ \text{Agent } A, \text{Agent } B, \text{Key } K, \text{nb} \} \}$   
 $\in \text{set evs};$   
 $\text{Notes Spy } \{ \text{na}, \text{nb}, \text{Key } K \} \notin \text{set evs};$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{yahalom} \rrbracket$   
 $\implies \text{Key } K \notin \text{knows Spy evs}"$

*<proof>*

### 16.3 Security Guarantee for A upon receiving YM3

If the encrypted message appears then it originated with the Server. May now apply *Spy\_not\_see\_encrypted\_key*, subject to its conditions.

**lemma** *A\_trusts\_YM3*:

$"\llbracket \text{Crypt } (\text{shrK } A) \{ \text{Agent } B, \text{Key } K, \text{na} \} \in \text{parts } (\text{knows Spy evs});$   
 $A \notin \text{bad}; \text{evs} \in \text{yahalom} \rrbracket$   
 $\implies \exists \text{nb. Says Server } A$   
 $\quad \{ \text{nb}, \text{Crypt } (\text{shrK } A) \{ \text{Agent } B, \text{Key } K, \text{na} \},$   
 $\quad \text{Crypt } (\text{shrK } B) \{ \text{Agent } A, \text{Agent } B, \text{Key } K, \text{nb} \} \}$   
 $\in \text{set evs}"$

*<proof>*

The obvious combination of *A\_trusts\_YM3* with *Spy\_not\_see\_encrypted\_key*

**theorem** *A\_gets\_good\_key*:

$"\llbracket \text{Crypt } (\text{shrK } A) \{ \text{Agent } B, \text{Key } K, \text{na} \} \in \text{parts } (\text{knows Spy evs});$   
 $\forall \text{nb. Notes Spy } \{ \text{na}, \text{nb}, \text{Key } K \} \notin \text{set evs};$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{evs} \in \text{yahalom} \rrbracket$   
 $\implies \text{Key } K \notin \text{analz } (\text{knows Spy evs})"$

*<proof>*

### 16.4 Security Guarantee for B upon receiving YM4

B knows, by the first part of A's message, that the Server distributed the key for A and B, and has associated it with NB.

**lemma** *B\_trusts\_YM4\_shrK*:

"[[Crypt (shrK B) {Agent A, Agent B, Key K, Nonce NB}  
 $\in$  parts (knows Spy evs);  
 $B \notin \text{bad}; \text{ evs} \in \text{yahalom}$ ]]  
 $\implies \exists NA. \text{ Says Server A}$   
 {Nonce NB,  
 Crypt (shrK A) {Agent B, Key K, Nonce NA},  
 Crypt (shrK B) {Agent A, Agent B, Key K, Nonce NB}}  
 $\in \text{set evs}$ "

*<proof>*

With this protocol variant, we don't need the 2nd part of YM4 at all: Nonce NB is available in the first part.

What can B deduce from receipt of YM4? Stronger and simpler than Yahalom because we do not have to show that NB is secret.

**lemma** *B\_trusts\_YM4*:

"[[Gets B {Crypt (shrK B) {Agent A, Agent B, Key K, Nonce NB}, X}  
 $\in \text{set evs};$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{ evs} \in \text{yahalom}$ ]]  
 $\implies \exists NA. \text{ Says Server A}$   
 {Nonce NB,  
 Crypt (shrK A) {Agent B, Key K, Nonce NA},  
 Crypt (shrK B) {Agent A, Agent B, Key K, Nonce NB}}  
 $\in \text{set evs}$ "

*<proof>*

The obvious combination of *B\_trusts\_YM4* with *Spy\_not\_see\_encrypted\_key*

**theorem** *B\_gets\_good\_key*:

"[[Gets B {Crypt (shrK B) {Agent A, Agent B, Key K, Nonce NB}, X}  
 $\in \text{set evs};$   
 $\forall na. \text{ Notes Spy } \{na, \text{ Nonce NB, Key K}\} \notin \text{set evs};$   
 $A \notin \text{bad}; B \notin \text{bad}; \text{ evs} \in \text{yahalom}$ ]]  
 $\implies \text{Key K} \notin \text{analz} (\text{knows Spy evs})"$

*<proof>*

### 16.5 Authenticating B to A

The encryption in message YM2 tells us it cannot be faked.

**lemma** *B\_Said\_YM2*:

"[[Crypt (shrK B) {Agent A, Nonce NA}  $\in$  parts (knows Spy evs);  
 $B \notin \text{bad}; \text{ evs} \in \text{yahalom}$ ]]  
 $\implies \exists NB. \text{ Says B Server } \{Agent B, \text{ Nonce NB,}$   
 Crypt (shrK B) {Agent A, Nonce NA}  
 $\in \text{set evs}$ "

*<proof>*

If the server sends YM3 then B sent YM2, perhaps with a different NB

**lemma** *YM3\_auth\_B\_to\_A\_lemma*:

```
"[[Says Server A {nb, Crypt (shrK A) {Agent B, Key K, Nonce NA}}, X]
  ∈ set evs;
  B ∉ bad; evs ∈ yahalom]]
⇒ ∃nb'. Says B Server {Agent B, nb',
                        Crypt (shrK B) {Agent A, Nonce NA}}
  ∈ set evs"
```

*<proof>*

If A receives YM3 then B has used nonce NA (and therefore is alive)

**theorem** *YM3\_auth\_B\_to\_A*:

```
"[[Gets A {nb, Crypt (shrK A) {Agent B, Key K, Nonce NA}}, X]
  ∈ set evs;
  A ∉ bad; B ∉ bad; evs ∈ yahalom]]
⇒ ∃nb'. Says B Server
    {Agent B, nb', Crypt (shrK B) {Agent A, Nonce NA}}
  ∈ set evs"
```

*<proof>*

## 16.6 Authenticating A to B

using the certificate *Crypt K (Nonce NB)*

Assuming the session key is secure, if both certificates are present then A has said NB. We can't be sure about the rest of A's message, but only NB matters for freshness. Note that *Key K ∉ analz (knows Spy evs)* must be the FIRST antecedent of the induction formula.

This lemma allows a use of *unique\_session\_keys* in the next proof, which otherwise is extremely slow.

**lemma** *secure\_unique\_session\_keys*:

```
"[[Crypt (shrK A) {Agent B, Key K, na} ∈ analz (spies evs);
  Crypt (shrK A') {Agent B', Key K, na'} ∈ analz (spies evs);
  Key K ∉ analz (knows Spy evs); evs ∈ yahalom]]
⇒ A=A' ∧ B=B'"
```

*<proof>*

**lemma** *Auth\_A\_to\_B\_lemma [rule\_format]*:

```
"evs ∈ yahalom
⇒ Key K ∉ analz (knows Spy evs) →
  K ∈ symKeys →
  Crypt K (Nonce NB) ∈ parts (knows Spy evs) →
  Crypt (shrK B) {Agent A, Agent B, Key K, Nonce NB}
  ∈ parts (knows Spy evs) →
  B ∉ bad →
  (∃X. Says A B {X, Crypt K (Nonce NB)}) ∈ set evs"
```

*<proof>*

If B receives YM4 then A has used nonce NB (and therefore is alive). Moreover, A associates K with NB (thus is talking about the same run). Other premises guarantee secrecy of K.

**theorem** *YM4\_imp\_A\_Said\_YM3 [rule\_format]*:

```

    "[Gets B {Crypt (shrK B) {Agent A, Agent B, Key K, Nonce NB}},
      Crypt K (Nonce NB)} ∈ set evs;
      (∀ NA. Notes Spy {Nonce NA, Nonce NB, Key K} ∉ set evs);
      K ∈ symKeys; A ∉ bad; B ∉ bad; evs ∈ yahalom]
    ⇒ ∃ X. Says A B {X, Crypt K (Nonce NB)} ∈ set evs"
  <proof>

end

```

## 17 The Yahalom Protocol: A Flawed Version

theory Yahalom\_Bad imports Public begin

Demonstrates of why Oops is necessary. This protocol can be attacked because it doesn't keep NB secret, but without Oops it can be "verified" anyway. The issues are discussed in lcp's LICS 2000 invited lecture.

inductive\_set yahalom :: "event list set"  
 where

Nil: "[ ] ∈ yahalom"

/ Fake: "[evsf ∈ yahalom; X ∈ synth (analz (knows Spy evsf))]  
 ⇒ Says Spy B X # evsf ∈ yahalom"

/ Reception: "[evsr ∈ yahalom; Says A B X ∈ set evsr]  
 ⇒ Gets B X # evsr ∈ yahalom"

/ YM1: "[evs1 ∈ yahalom; Nonce NA ∉ used evs1]  
 ⇒ Says A B {Agent A, Nonce NA} # evs1 ∈ yahalom"

/ YM2: "[evs2 ∈ yahalom; Nonce NB ∉ used evs2;  
 Gets B {Agent A, Nonce NA} ∈ set evs2]  
 ⇒ Says B Server  
 {Agent B, Nonce NB, Crypt (shrK B) {Agent A, Nonce NA}}  
 # evs2 ∈ yahalom"

/ YM3: "[evs3 ∈ yahalom; Key KAB ∉ used evs3; KAB ∈ symKeys;  
 Gets Server  
 {Agent B, Nonce NB, Crypt (shrK B) {Agent A, Nonce NA}}  
 ∈ set evs3]  
 ⇒ Says Server A  
 {Crypt (shrK A) {Agent B, Key KAB, Nonce NA, Nonce NB},  
 Crypt (shrK B) {Agent A, Key KAB}}  
 # evs3 ∈ yahalom"

/ YM4: "[evs4 ∈ yahalom; A ≠ Server; K ∈ symKeys;  
 Gets A {Crypt (shrK A) {Agent B, Key K, Nonce NA, Nonce NB}, X}



```

    ∈ set evs4;
    Says A B {|Agent A, Nonce NA|} ∈ set evs4]]
    ⇒ Says A B {|X, Crypt K (Nonce NB)|} # evs4 ∈ yahalom"

```

```

declare Says_imp_knows_Spy [THEN analz.Inj, dest]
declare parts.Body [dest]
declare Fake_parts_insert_in_Un [dest]
declare analz_into_parts [dest]

```

A "possibility property": there are traces that reach the end

```

lemma "[|A ≠ Server; Key K ∉ used []; K ∈ symKeys|]
    ⇒ ∃ X NB. ∃ evs ∈ yahalom.
        Says A B {|X, Crypt K (Nonce NB)|} ∈ set evs"
<proof>

```

## 17.1 Regularity Lemmas for Yahalom

```

lemma Gets_imp_Says:
    "[|Gets B X ∈ set evs; evs ∈ yahalom|] ⇒ ∃ A. Says A B X ∈ set evs"
<proof>

```

```

lemma Gets_imp_knows_Spy:
    "[|Gets B X ∈ set evs; evs ∈ yahalom|] ⇒ X ∈ knows Spy evs"
<proof>

```

```

declare Gets_imp_knows_Spy [THEN analz.Inj, dest]

```

## 17.2 For reasoning about the encrypted portion of messages

Lets us treat YM4 using a similar argument as for the Fake case.

```

lemma YM4_analz_knows_Spy:
    "[|Gets A {|Crypt (shrK A) Y, X|} ∈ set evs; evs ∈ yahalom|]
    ⇒ X ∈ analz (knows Spy evs)"
<proof>

```

```

lemmas YM4_parts_knows_Spy =
    YM4_analz_knows_Spy [THEN analz_into_parts]

```

Theorems of the form  $X \notin \text{parts } (\text{knows Spy evs})$  imply that NOBODY sends messages containing X!

Spy never sees a good agent's shared key!

```

lemma Spy_see_shrK [simp]:
    "evs ∈ yahalom ⇒ (Key (shrK A) ∈ parts (knows Spy evs)) = (A ∈ bad)"
<proof>

```

```

lemma Spy_analz_shrK [simp]:
    "evs ∈ yahalom ⇒ (Key (shrK A) ∈ analz (knows Spy evs)) = (A ∈ bad)"
<proof>

```

```

lemma Spy_see_shrK_D [dest!]:
  "[[Key (shrK A) ∈ parts (knows Spy evs); evs ∈ yahalom]] ⇒ A ∈ bad"
  <proof>

```

Nobody can have used non-existent keys! Needed to apply `analz_insert_Key`

```

lemma new_keys_not_used [simp]:
  "[[Key K ∉ used evs; K ∈ symKeys; evs ∈ yahalom]]
   ⇒ K ∉ keysFor (parts (spies evs))"
  <proof>

```

### 17.3 Secrecy Theorems

#### 17.4 Session keys are not used to encrypt other session keys

```

lemma analz_image_freshK [rule_format]:
  "evs ∈ yahalom ⇒
   ∀K KK. KK ⊆ - (range shrK) →
   (Key K ∈ analz (Key'KK ∪ (knows Spy evs))) =
   (K ∈ KK | Key K ∈ analz (knows Spy evs))"
  <proof>

```

```

lemma analz_insert_freshK:
  "[[evs ∈ yahalom; KAB ∉ range shrK]] ⇒
   (Key K ∈ analz (insert (Key KAB) (knows Spy evs))) =
   (K = KAB | Key K ∈ analz (knows Spy evs))"
  <proof>

```

The Key K uniquely identifies the Server's message.

```

lemma unique_session_keys:
  "[[Says Server A
   {Crypt (shrK A) {Agent B, Key K, na, nb}}, X} ∈ set evs;
   Says Server A'
   {Crypt (shrK A') {Agent B', Key K, na', nb'}}, X'} ∈ set evs;
   evs ∈ yahalom]]
   ⇒ A=A' ∧ B=B' ∧ na=na' ∧ nb=nb'"
  <proof>

```

Crucial secrecy property: Spy does not see the keys sent in msg YM3

```

lemma secrecy_lemma:
  "[[A ∉ bad; B ∉ bad; evs ∈ yahalom]]
   ⇒ Says Server A
   {Crypt (shrK A) {Agent B, Key K, na, nb}},
   Crypt (shrK B) {Agent A, Key K}}
   ∈ set evs →
   Key K ∉ analz (knows Spy evs)"
  <proof>

```

Final version

```

lemma Spy_not_see_encrypted_key:
  "[[Says Server A
   {Crypt (shrK A) {Agent B, Key K, na, nb}},
   Crypt (shrK B) {Agent A, Key K}}]

```

$$\begin{aligned} & \in \text{set evs}; \\ & A \notin \text{bad}; \quad B \notin \text{bad}; \quad \text{evs} \in \text{yahalom} \\ & \implies \text{Key } K \notin \text{analz } (\text{knows Spy evs}) \end{aligned}$$
 <proof>

## 17.5 Security Guarantee for A upon receiving YM3

If the encrypted message appears then it originated with the Server

**lemma** *A\_trusts\_YM3*:  

$$\begin{aligned} & "[\text{Crypt } (\text{shrK } A) \{ \text{Agent } B, \text{Key } K, na, nb \} \in \text{parts } (\text{knows Spy evs}); \\ & \quad A \notin \text{bad}; \quad \text{evs} \in \text{yahalom} ] \\ & \implies \text{Says Server A} \\ & \quad [\text{Crypt } (\text{shrK } A) \{ \text{Agent } B, \text{Key } K, na, nb \}, \\ & \quad \quad \text{Crypt } (\text{shrK } B) \{ \text{Agent } A, \text{Key } K \} ] \\ & \quad \in \text{set evs} \end{aligned}$$
 <proof>

The obvious combination of *A\_trusts\_YM3* with *Spy\_not\_see\_encrypted\_key*

**lemma** *A\_gets\_good\_key*:  

$$\begin{aligned} & "[\text{Crypt } (\text{shrK } A) \{ \text{Agent } B, \text{Key } K, na, nb \} \in \text{parts } (\text{knows Spy evs}); \\ & \quad A \notin \text{bad}; \quad B \notin \text{bad}; \quad \text{evs} \in \text{yahalom} ] \\ & \implies \text{Key } K \notin \text{analz } (\text{knows Spy evs}) \end{aligned}$$
 <proof>

## 17.6 Security Guarantees for B upon receiving YM4

B knows, by the first part of A's message, that the Server distributed the key for A and B. But this part says nothing about nonces.

**lemma** *B\_trusts\_YM4\_shrK*:  

$$\begin{aligned} & "[\text{Crypt } (\text{shrK } B) \{ \text{Agent } A, \text{Key } K \} \in \text{parts } (\text{knows Spy evs}); \\ & \quad B \notin \text{bad}; \quad \text{evs} \in \text{yahalom} ] \\ & \implies \exists NA NB. \text{Says Server A} \\ & \quad [\text{Crypt } (\text{shrK } A) \{ \text{Agent } B, \text{Key } K, \text{Nonce } NA, \text{Nonce } NB \}, \\ & \quad \quad \text{Crypt } (\text{shrK } B) \{ \text{Agent } A, \text{Key } K \} ] \\ & \quad \in \text{set evs} \end{aligned}$$
 <proof>

## 17.7 The Flaw in the Model

Up to now, the reasoning is similar to standard Yahalom. Now the doubtful reasoning occurs. We should not be assuming that an unknown key is secure, but the model allows us to: there is no *Oops* rule to let session keys become compromised.

B knows, by the second part of A's message, that the Server distributed the key quoting nonce NB. This part says nothing about agent names. Secrecy of K is assumed; the valid Yahalom proof uses (and later proves) the secrecy of NB.

**lemma** *B\_trusts\_YM4\_newK [rule\_format]*:  

$$\begin{aligned} & "[\text{Key } K \notin \text{analz } (\text{knows Spy evs}); \quad \text{evs} \in \text{yahalom} ] \\ & \implies \text{Crypt } K (\text{Nonce } NB) \in \text{parts } (\text{knows Spy evs}) \longrightarrow \\ & \quad (\exists A B NA. \text{Says Server A} \end{aligned}$$

$\{\!\{ \text{Crypt } (\text{shr}K \ A) \ \{\!\{ \text{Agent } B, \text{Key } K, \\ \text{Nonce } NA, \text{Nonce } NB \}\!\}, \\ \text{Crypt } (\text{shr}K \ B) \ \{\!\{ \text{Agent } A, \text{Key } K \}\!\} \\ \in \text{set evs} \}\!\}$

$\langle \text{proof} \rangle$

B's session key guarantee from YM4. The two certificates contribute to a single conclusion about the Server's message.

**lemma** *B\_trusts\_YM4*:

$\llbracket \text{Gets } B \ \{\!\{ \text{Crypt } (\text{shr}K \ B) \ \{\!\{ \text{Agent } A, \text{Key } K \}\!\}, \\ \text{Crypt } K \ (\text{Nonce } NB) \}\!\} \in \text{set evs}; \\ \text{Says } B \ \text{Server} \\ \{\!\{ \text{Agent } B, \text{Nonce } NB, \text{Crypt } (\text{shr}K \ B) \ \{\!\{ \text{Agent } A, \text{Nonce } NA \}\!\} \\ \in \text{set evs}; \\ A \notin \text{bad}; \ B \notin \text{bad}; \ \text{evs} \in \text{yahalom} \rrbracket \\ \implies \exists na \ nb. \ \text{Says } \text{Server } A \\ \{\!\{ \text{Crypt } (\text{shr}K \ A) \ \{\!\{ \text{Agent } B, \text{Key } K, na, nb \}\!\}, \\ \text{Crypt } (\text{shr}K \ B) \ \{\!\{ \text{Agent } A, \text{Key } K \}\!\} \\ \in \text{set evs} \}\!\}$

$\langle \text{proof} \rangle$

The obvious combination of *B\_trusts\_YM4* with *Spy\_not\_see\_encrypted\_key*

**lemma** *B\_gets\_good\_key*:

$\llbracket \text{Gets } B \ \{\!\{ \text{Crypt } (\text{shr}K \ B) \ \{\!\{ \text{Agent } A, \text{Key } K \}\!\}, \\ \text{Crypt } K \ (\text{Nonce } NB) \}\!\} \in \text{set evs}; \\ \text{Says } B \ \text{Server} \\ \{\!\{ \text{Agent } B, \text{Nonce } NB, \text{Crypt } (\text{shr}K \ B) \ \{\!\{ \text{Agent } A, \text{Nonce } NA \}\!\} \\ \in \text{set evs}; \\ A \notin \text{bad}; \ B \notin \text{bad}; \ \text{evs} \in \text{yahalom} \rrbracket \\ \implies \text{Key } K \notin \text{analz } (\text{knows } \text{Spy } \text{evs})$

$\langle \text{proof} \rangle$

Assuming the session key is secure, if both certificates are present then A has said NB. We can't be sure about the rest of A's message, but only NB matters for freshness.

**lemma** *A\_Said\_YM3\_lemma* [rule\_format]:

$\llbracket \text{evs} \in \text{yahalom} \\ \implies \text{Key } K \notin \text{analz } (\text{knows } \text{Spy } \text{evs}) \longrightarrow \\ \text{Crypt } K \ (\text{Nonce } NB) \in \text{parts } (\text{knows } \text{Spy } \text{evs}) \longrightarrow \\ \text{Crypt } (\text{shr}K \ B) \ \{\!\{ \text{Agent } A, \text{Key } K \}\!\} \in \text{parts } (\text{knows } \text{Spy } \text{evs}) \longrightarrow \\ B \notin \text{bad} \longrightarrow \\ (\exists X. \ \text{Says } A \ B \ \{\!\{ X, \text{Crypt } K \ (\text{Nonce } NB) \}\!\} \in \text{set evs}) \rrbracket$

$\langle \text{proof} \rangle$

If B receives YM4 then A has used nonce NB (and therefore is alive). Moreover, A associates K with NB (thus is talking about the same run). Other premises guarantee secrecy of K.

**lemma** *YM4\_imp\_A\_Said\_YM3* [rule\_format]:

$\llbracket \text{Gets } B \ \{\!\{ \text{Crypt } (\text{shr}K \ B) \ \{\!\{ \text{Agent } A, \text{Key } K \}\!\}, \\ \text{Crypt } K \ (\text{Nonce } NB) \}\!\} \in \text{set evs}; \\ \text{Says } B \ \text{Server} \\ \{\!\{ \text{Agent } B, \text{Nonce } NB, \text{Crypt } (\text{shr}K \ B) \ \{\!\{ \text{Agent } A, \text{Nonce } NA \}\!\} \\ \in \text{set evs};$

```

      A ∉ bad; B ∉ bad; evs ∈ yahalom
    ⇒ ∃ X. Says A B {X, Crypt K (Nonce NB)} ∈ set evs"
  <proof>

```

```

end

```

```

theory ZhouGollmann imports Public begin

```

```

abbreviation

```

```

  TTP :: agent where "TTP == Server"

```

```

abbreviation f_sub :: nat where "f_sub == 5"

```

```

abbreviation f_nro :: nat where "f_nro == 2"

```

```

abbreviation f_nrr :: nat where "f_nrr == 3"

```

```

abbreviation f_con :: nat where "f_con == 4"

```

```

definition broken :: "agent set" where

```

```

  — the compromised honest agents; TTP is included as it's not allowed to use the
  protocol

```

```

  "broken == bad - {Spy}"

```

```

declare broken_def [simp]

```

```

inductive_set zg :: "event list set"

```

```

  where

```

```

    Nil: "[] ∈ zg"

```

```

  / Fake: "[[evsf ∈ zg; X ∈ synth (analz (spies evsf))]]
    ⇒ Says Spy B X # evsf ∈ zg"

```

```

  / Reception: "[[evsr ∈ zg; Says A B X ∈ set evsr]] ⇒ Gets B X # evsr ∈ zg"

```

```

  / ZG1: "[[evs1 ∈ zg; Nonce L ∉ used evs1; C = Crypt K (Number m);
    K ∈ symKeys;
    NRO = Crypt (priK A) {Number f_nro, Agent B, Nonce L, C}]]
    ⇒ Says A B {Number f_nro, Agent B, Nonce L, C, NRO} # evs1 ∈ zg"

```

```

  / ZG2: "[[evs2 ∈ zg;
    Gets B {Number f_nro, Agent B, Nonce L, C, NRO} ∈ set evs2;
    NRO = Crypt (priK A) {Number f_nro, Agent B, Nonce L, C};
    NRR = Crypt (priK B) {Number f_nrr, Agent A, Nonce L, C}]]
    ⇒ Says B A {Number f_nrr, Agent A, Nonce L, NRR} # evs2 ∈ zg"

```

```

  / ZG3: "[[evs3 ∈ zg; C = Crypt K M; K ∈ symKeys;
    Says A B {Number f_nro, Agent B, Nonce L, C, NRO} ∈ set evs3;
    Gets A {Number f_nrr, Agent A, Nonce L, NRR} ∈ set evs3;
    NRR = Crypt (priK B) {Number f_nrr, Agent A, Nonce L, C};
    sub_K = Crypt (priK A) {Number f_sub, Agent B, Nonce L, Key K}]]

```

```

⇒ Says A TTP {Number f_sub, Agent B, Nonce L, Key K, sub_K}
    # evs3 ∈ zg"

/ ZG4: "[[evs4 ∈ zg; K ∈ symKeys;
    Gets TTP {Number f_sub, Agent B, Nonce L, Key K, sub_K}
    ∈ set evs4;
    sub_K = Crypt (priK A) {Number f_sub, Agent B, Nonce L, Key K};
    con_K = Crypt (priK TTP) {Number f_con, Agent A, Agent B,
        Nonce L, Key K}]]
⇒ Says TTP Spy con_K
    #
    Notes TTP {Number f_con, Agent A, Agent B, Nonce L, Key K, con_K}
    # evs4 ∈ zg"

```

```

declare Says_imp_knows_Spy [THEN analz.Inj, dest]
declare Fake_parts_insert_in_Un [dest]
declare analz_into_parts [dest]

```

```

declare symKey_neq_priEK [simp]
declare symKey_neq_priEK [THEN not_sym, simp]

```

A "possibility property": there are traces that reach the end

```

lemma "[[A ≠ B; TTP ≠ A; TTP ≠ B; K ∈ symKeys]] ⇒
  ∃ L. ∃ evs ∈ zg.
    Notes TTP {Number f_con, Agent A, Agent B, Nonce L, Key K,
      Crypt (priK TTP) {Number f_con, Agent A, Agent B, Nonce L,
Key K}}
    ∈ set evs"
⟨proof⟩

```

## 17.8 Basic Lemmas

```

lemma Gets_imp_Says:
  "[[Gets B X ∈ set evs; evs ∈ zg]] ⇒ ∃ A. Says A B X ∈ set evs"
⟨proof⟩

```

```

lemma Gets_imp_knows_Spy:
  "[[Gets B X ∈ set evs; evs ∈ zg]] ⇒ X ∈ spies evs"
⟨proof⟩

```

Lets us replace proofs about *used evs* by simpler proofs about *parts (knows Spy evs)*.

```

lemma Crypt_used_imp_spies:
  "[[Crypt K X ∈ used evs; evs ∈ zg]]
  ⇒ Crypt K X ∈ parts (spies evs)"
⟨proof⟩

```

```

lemma Notes_TTP_imp_Gets:
  "[[Notes TTP {Number f_con, Agent A, Agent B, Nonce L, Key K, con_K}
    ∈ set evs;
    sub_K = Crypt (priK A) {Number f_sub, Agent B, Nonce L, Key K};
    evs ∈ zg]]

```

$\Rightarrow \text{Gets TTP } \{ \text{Number } f\_sub, \text{ Agent } B, \text{ Nonce } L, \text{ Key } K, \text{ sub\_K} \} \in \text{set evs}$ "  
 <proof>

For reasoning about C, which is encrypted in message ZG2

**lemma** *ZG2\_msg\_in\_parts\_spies*:  
 "[Gets B {F, B', L, C, X}]  $\in$  set evs; evs  $\in$  zg]  
 $\Rightarrow C \in \text{parts (spies evs)}$ "  
 <proof>

**lemma** *Spy\_see\_priK [simp]*:  
 "evs  $\in$  zg  $\Rightarrow (\text{Key (priK A)} \in \text{parts (spies evs)}) = (A \in \text{bad})$ "  
 <proof>

So that blast can use it too

**declare** *Spy\_see\_priK [THEN [2] rev\_iffD1, dest!]*

**lemma** *Spy\_analz\_priK [simp]*:  
 "evs  $\in$  zg  $\Rightarrow (\text{Key (priK A)} \in \text{analz (spies evs)}) = (A \in \text{bad})$ "  
 <proof>

## 17.9 About NRO: Validity for B

Below we prove that if *NRO* exists then A definitely sent it, provided A is not broken.

Strong conclusion for a good agent

**lemma** *NRO\_validity\_good*:  
 "[NRO = Crypt (priK A) {Number f\_nro, Agent B, Nonce L, C}];  
 NRO  $\in$  parts (spies evs);  
 A  $\notin$  bad; evs  $\in$  zg]  
 $\Rightarrow \text{Says A B } \{ \text{Number } f\_nro, \text{ Agent } B, \text{ Nonce } L, \text{ C, NRO} \} \in \text{set evs}$ "  
 <proof>

**lemma** *NRO\_sender*:  
 "[Says A' B {n, b, l, C, Crypt (priK A) X}]  $\in$  set evs; evs  $\in$  zg]  
 $\Rightarrow A' \in \{A, \text{Spy}\}$ "  
 <proof>

Holds also for A = Spy!

**theorem** *NRO\_validity*:  
 "[Gets B {Number f\_nro, Agent B, Nonce L, C, NRO}]  $\in$  set evs;  
 NRO = Crypt (priK A) {Number f\_nro, Agent B, Nonce L, C};  
 A  $\notin$  broken; evs  $\in$  zg]  
 $\Rightarrow \text{Says A B } \{ \text{Number } f\_nro, \text{ Agent } B, \text{ Nonce } L, \text{ C, NRO} \} \in \text{set evs}$ "  
 <proof>

## 17.10 About NRR: Validity for A

Below we prove that if *NRR* exists then B definitely sent it, provided B is not broken.

Strong conclusion for a good agent

**lemma** *NRR\_validity\_good*:

"[[NRR = Crypt (priK B) {Number f\_nrr, Agent A, Nonce L, C}];  
 NRR ∈ parts (spies evs);  
 B ∉ bad; evs ∈ zg]]  
 ⇒ Says B A {Number f\_nrr, Agent A, Nonce L, NRR} ∈ set evs"  
 <proof>

**lemma** *NRR\_sender*:

"[[Says B' A {n, a, l, Crypt (priK B) X} ∈ set evs; evs ∈ zg]]  
 ⇒ B' ∈ {B, Spy}"  
 <proof>

Holds also for  $B = \text{Spy}$ !

**theorem** *NRR\_validity*:

"[[Says B' A {Number f\_nrr, Agent A, Nonce L, NRR} ∈ set evs;  
 NRR = Crypt (priK B) {Number f\_nrr, Agent A, Nonce L, C};  
 B ∉ broken; evs ∈ zg]]  
 ⇒ Says B A {Number f\_nrr, Agent A, Nonce L, NRR} ∈ set evs"  
 <proof>

### 17.11 Proofs About *sub\_K*

Below we prove that if *sub\_K* exists then *A* definitely sent it, provided *A* is not broken.

Strong conclusion for a good agent

**lemma** *sub\_K\_validity\_good*:

"[[sub\_K = Crypt (priK A) {Number f\_sub, Agent B, Nonce L, Key K};  
 sub\_K ∈ parts (spies evs);  
 A ∉ bad; evs ∈ zg]]  
 ⇒ Says A TTP {Number f\_sub, Agent B, Nonce L, Key K, sub\_K} ∈ set evs"  
 <proof>

**lemma** *sub\_K\_sender*:

"[[Says A' TTP {n, b, l, k, Crypt (priK A) X} ∈ set evs; evs ∈ zg]]  
 ⇒ A' ∈ {A, Spy}"  
 <proof>

Holds also for  $A = \text{Spy}$ !

**theorem** *sub\_K\_validity*:

"[[Gets TTP {Number f\_sub, Agent B, Nonce L, Key K, sub\_K} ∈ set evs;  
 sub\_K = Crypt (priK A) {Number f\_sub, Agent B, Nonce L, Key K};  
 A ∉ broken; evs ∈ zg]]  
 ⇒ Says A TTP {Number f\_sub, Agent B, Nonce L, Key K, sub\_K} ∈ set evs"  
 <proof>

### 17.12 Proofs About *con\_K*

Below we prove that if *con\_K* exists, then *TTP* has it, and therefore *A* and *B*) can get it too. Moreover, we know that *A* sent *sub\_K*

**lemma** *con\_K\_validity*:

"[[con\_K ∈ used evs;



```

con_K = Crypt (priK TTP)
          {Number f_con, Agent A, Agent B, Nonce L, Key K};
evs ∈ zg
⇒ Notes TTP {Number f_con, Agent A, Agent B, Nonce L, Key K, con_K}
    ∈ set evs"
⟨proof⟩

```

If *TTP* holds *con\_K* then *A* sent *sub\_K*. We assume that *A* is not broken. Importantly, nothing needs to be assumed about the form of *con\_K*!

**lemma** *Notes\_TTP\_imp\_Says\_A*:

```

"[[Notes TTP {Number f_con, Agent A, Agent B, Nonce L, Key K, con_K}
   ∈ set evs;
   sub_K = Crypt (priK A) {Number f_sub, Agent B, Nonce L, Key K};
   A ∉ broken; evs ∈ zg]]
⇒ Says A TTP {Number f_sub, Agent B, Nonce L, Key K, sub_K} ∈ set evs"
⟨proof⟩

```

If *con\_K* exists, then *A* sent *sub\_K*. We again assume that *A* is not broken.

**theorem** *B\_sub\_K\_validity*:

```

"[[con_K ∈ used evs;
   con_K = Crypt (priK TTP) {Number f_con, Agent A, Agent B,
                               Nonce L, Key K};
   sub_K = Crypt (priK A) {Number f_sub, Agent B, Nonce L, Key K};
   A ∉ broken; evs ∈ zg]]
⇒ Says A TTP {Number f_sub, Agent B, Nonce L, Key K, sub_K} ∈ set evs"
⟨proof⟩

```

### 17.13 Proving fairness

Cannot prove that, if *B* has NRO, then *A* has her NRR. It would appear that *B* has a small advantage, though it is useless to win disputes: *B* needs to present *con\_K* as well.

Strange: unicity of the label protects *A*?

**lemma** *A\_unicity*:

```

"[[NRO = Crypt (priK A) {Number f_nro, Agent B, Nonce L, Crypt K M};
   NRO ∈ parts (spies evs);
   Says A B {Number f_nro, Agent B, Nonce L, Crypt K M', NRO'}
   ∈ set evs;
   A ∉ bad; evs ∈ zg]]
⇒ M'=M"
⟨proof⟩

```

Fairness lemma: if *sub\_K* exists, then *A* holds NRR. Relies on unicity of labels.

**lemma** *sub\_K\_implies\_NRR*:

```

"[[NRO = Crypt (priK A) {Number f_nro, Agent B, Nonce L, Crypt K M};
   NRR = Crypt (priK B) {Number f_nrr, Agent A, Nonce L, Crypt K M};
   sub_K ∈ parts (spies evs);
   NRO ∈ parts (spies evs);
   sub_K = Crypt (priK A) {Number f_sub, Agent B, Nonce L, Key K};
   A ∉ bad; evs ∈ zg]]
⇒ Gets A {Number f_nrr, Agent A, Nonce L, NRR} ∈ set evs"

```

*<proof>*

**lemma** *Crypt\_used\_imp\_L\_used:*

"[[Crypt (priK TTP) {F, A, B, L, K} ∈ used evs; evs ∈ zg]]  
 ⇒ L ∈ used evs"

*<proof>*

Fairness for *A*: if *con\_K* and *NRO* exist, then *A* holds *NRR*. *A* must be uncompromised, but there is no assumption about *B*.

**theorem** *A\_fairness\_NRR:*

"[[con\_K ∈ used evs;  
 NRO ∈ parts (spies evs);  
 con\_K = Crypt (priK TTP)  
 {Number f\_con, Agent A, Agent B, Nonce L, Key K};  
 NRO = Crypt (priK A) {Number f\_nro, Agent B, Nonce L, Crypt K M};  
 NRR = Crypt (priK B) {Number f\_nrr, Agent A, Nonce L, Crypt K M};  
 A ∉ bad; evs ∈ zg]]  
 ⇒ Gets A {Number f\_nrr, Agent A, Nonce L, NRR} ∈ set evs"

*<proof>*

Fairness for *B*: *NRR* exists at all, then *B* holds *NRO*. *B* must be uncompromised, but there is no assumption about *A*.

**theorem** *B\_fairness\_NRR:*

"[[NRR ∈ used evs;  
 NRR = Crypt (priK B) {Number f\_nrr, Agent A, Nonce L, C};  
 NRO = Crypt (priK A) {Number f\_nro, Agent B, Nonce L, C};  
 B ∉ bad; evs ∈ zg]]  
 ⇒ Gets B {Number f\_nro, Agent B, Nonce L, C, NRO} ∈ set evs"

*<proof>*

If *con\_K* exists at all, then *B* can get it, by *con\_K\_validity*. Cannot conclude that also *NRO* is available to *B*, because if *A* were unfair, *A* could build message 3 without building message 1, which contains *NRO*.

**end**

## 18 Conventional protocols: rely on conventional Message, Event and Public – Shared-key protocols

**theory** *Auth\_Shared*

**imports**

*NS\_Shared*  
*Kerberos\_BAN*  
*Kerberos\_BAN\_Gets*  
*KerberosIV*  
*KerberosIV\_Gets*  
*KerberosV*  
*OtwayRees*  
*OtwayRees\_AN*  
*OtwayRees\_Bad*

```

OtwayReesBella
WooLam
Recur
Yahalom
Yahalom2
Yahalom_Bad
ZhouGollmann
begin

end

```

## 19 The Needham-Schroeder Public-Key Protocol (Flawed)

Flawed version, vulnerable to Lowe's attack. From Burrows, Abadi and Needham. A Logic of Authentication. Proc. Royal Soc. 426 (1989), p. 260

```

theory NS_Public_Bad imports Public begin

inductive_set ns_public :: "event list set"
  where
    Nil: "[ ] ∈ ns_public"
    — Initial trace is empty
  | Fake: "[[evsf ∈ ns_public; X ∈ synth (analz (spies evsf))]]
    ⇒ Says Spy B X # evsf ∈ ns_public"
    — The spy can say almost anything.
  | NS1: "[[evs1 ∈ ns_public; Nonce NA ∉ used evs1]]
    ⇒ Says A B (Crypt (pubEK B) {Nonce NA, Agent A})
    # evs1 ∈ ns_public"
    — Alice initiates a protocol run, sending a nonce to Bob
  | NS2: "[[evs2 ∈ ns_public; Nonce NB ∉ used evs2;
    Says A' B (Crypt (pubEK B) {Nonce NA, Agent A}) ∈ set evs2]]
    ⇒ Says B A (Crypt (pubEK A) {Nonce NA, Nonce NB})
    # evs2 ∈ ns_public"
    — Bob responds to Alice's message with a further nonce
  | NS3: "[[evs3 ∈ ns_public;
    Says A B (Crypt (pubEK B) {Nonce NA, Agent A}) ∈ set evs3;
    Says B' A (Crypt (pubEK A) {Nonce NA, Nonce NB}) ∈ set evs3]]
    ⇒ Says A B (Crypt (pubEK B) (Nonce NB)) # evs3 ∈ ns_public"
    — Alice proves her existence by sending NB back to Bob.

declare knows_Spy_partsEs [elim]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]

A "possibility property": there are traces that reach the end

lemma "∃ NB. ∃ evs ∈ ns_public. Says A B (Crypt (pubEK B) (Nonce NB)) ∈ set evs"
  ⟨proof⟩

```

### 19.1 Inductive proofs about *ns\_public*

Spy never sees another agent's private key! (unless it's bad at start)

```

lemma Spy_see_priEK [simp]:
  "evs ∈ ns_public ⇒ (Key (priEK A) ∈ parts (spies evs)) = (A ∈ bad)"
  <proof>

```

```

lemma Spy_analz_priEK [simp]:
  "evs ∈ ns_public ⇒ (Key (priEK A) ∈ analz (spies evs)) = (A ∈ bad)"
  <proof>

```

## 19.2 Authenticity properties obtained from term NS1

It is impossible to re-use a nonce in both term NS1 and term NS2, provided the nonce is secret. (Honest users generate fresh nonces.)

```

lemma no_nonce_NS1_NS2:
  "[[evs ∈ ns_public;
    Crypt (pubEK C) {NA', Nonce NA} ∈ parts (spies evs);
    Crypt (pubEK B) {Nonce NA, Agent A} ∈ parts (spies evs)]
  ⇒ Nonce NA ∈ analz (spies evs)"
  <proof>

```

Unicity for term NS1: nonce term NA identifies agents term A and term B

```

lemma unique_NA:
  assumes NA: "Crypt(pubEK B) {Nonce NA, Agent A} ∈ parts(spies evs)"
    "Crypt(pubEK B') {Nonce NA, Agent A'} ∈ parts(spies evs)"
    "Nonce NA ∉ analz (spies evs)"
  and evs: "evs ∈ ns_public"
  shows "A=A' ∧ B=B'"
  <proof>

```

Secrecy: Spy does not see the nonce sent in msg term NS1 if term A and term B are secure The major premise "Says A B ..." makes it a dest-rule, hence the given assumption order.

```

theorem Spy_not_see_NA:
  assumes NA: "Says A B (Crypt(pubEK B) {Nonce NA, Agent A}) ∈ set evs"
    "A ∉ bad" "B ∉ bad"
  and evs: "evs ∈ ns_public"
  shows "Nonce NA ∉ analz (spies evs)"
  <proof>

```

Authentication for term A: if she receives message 2 and has used term NA to start a run, then term B has sent message 2.

```

lemma A_trusts_NS2_lemma:
  "[[evs ∈ ns_public;
    Crypt (pubEK A) {Nonce NA, Nonce NB} ∈ parts (spies evs);
    Says A B (Crypt(pubEK B) {Nonce NA, Agent A}) ∈ set evs;
    A ∉ bad; B ∉ bad]
  ⇒ Says B A (Crypt(pubEK A) {Nonce NA, Nonce NB}) ∈ set evs"
  <proof>

```

```

theorem A_trusts_NS2:
  "[[Says A B (Crypt(pubEK B) {Nonce NA, Agent A}) ∈ set evs;
    Says B' A (Crypt(pubEK A) {Nonce NA, Nonce NB}) ∈ set evs;
    A ∉ bad; B ∉ bad; evs ∈ ns_public]

```

$\Rightarrow \text{Says } B \ A \ (\text{Crypt}(\text{pubEK } A) \ \{\!\!\{ \text{Nonce } NA, \text{Nonce } NB \}\!\!\}) \in \text{set } \text{evs}"$   
 $\langle \text{proof} \rangle$

If the encrypted message appears then it originated with Alice in term NS1

**lemma** *B\_trusts\_NS1*:

" $\llbracket \text{evs} \in \text{ns\_public};$   
 $\text{Crypt}(\text{pubEK } B) \ \{\!\!\{ \text{Nonce } NA, \text{Agent } A \}\!\!\} \in \text{parts}(\text{spies } \text{evs});$   
 $\text{Nonce } NA \notin \text{analz}(\text{spies } \text{evs}) \rrbracket$   
 $\Rightarrow \text{Says } A \ B \ (\text{Crypt}(\text{pubEK } B) \ \{\!\!\{ \text{Nonce } NA, \text{Agent } A \}\!\!\}) \in \text{set } \text{evs}"$   
 $\langle \text{proof} \rangle$

### 19.3 Authenticity properties obtained from term NS2

Unicity for term NS2: nonce term NB identifies nonce term NA and agent term A [proof closely follows that for *unique\_NA*]

**lemma** *unique\_NB [dest]*:

**assumes** NB: " $\text{Crypt}(\text{pubEK } A) \ \{\!\!\{ \text{Nonce } NA, \text{Nonce } NB \}\!\!\} \in \text{parts}(\text{spies } \text{evs})"$   
 $\text{Crypt}(\text{pubEK } A') \ \{\!\!\{ \text{Nonce } NA', \text{Nonce } NB \}\!\!\} \in \text{parts}(\text{spies } \text{evs})"$   
 $\text{Nonce } NB \notin \text{analz}(\text{spies } \text{evs})"$   
**and** *evs*: " $\text{evs} \in \text{ns\_public}"$   
**shows** " $A=A' \ \wedge \ NA=NA'"$   
 $\langle \text{proof} \rangle$

term NB remains secret *provided* Alice never responds with round 3

**theorem** *Spy\_not\_see\_NB [dest]*:

**assumes** NB: " $\text{Says } B \ A \ (\text{Crypt}(\text{pubEK } A) \ \{\!\!\{ \text{Nonce } NA, \text{Nonce } NB \}\!\!\}) \in \text{set } \text{evs}"$   
 $\forall C. \text{Says } A \ C \ (\text{Crypt}(\text{pubEK } C) \ (\text{Nonce } NB)) \notin \text{set } \text{evs}"$   
 $A \notin \text{bad} \ \text{ " } B \notin \text{bad}"$   
**and** *evs*: " $\text{evs} \in \text{ns\_public}"$   
**shows** " $\text{Nonce } NB \notin \text{analz}(\text{spies } \text{evs})"$   
 $\langle \text{proof} \rangle$

Authentication for term B: if he receives message 3 and has used term NB in message 2, then term A has sent message 3 (to somebody)

**lemma** *B\_trusts\_NS3\_lemma*:

" $\llbracket \text{evs} \in \text{ns\_public};$   
 $\text{Crypt}(\text{pubEK } B) \ (\text{Nonce } NB) \in \text{parts}(\text{spies } \text{evs});$   
 $\text{Says } B \ A \ (\text{Crypt}(\text{pubEK } A) \ \{\!\!\{ \text{Nonce } NA, \text{Nonce } NB \}\!\!\}) \in \text{set } \text{evs};$   
 $A \notin \text{bad}; \ B \notin \text{bad} \rrbracket$   
 $\Rightarrow \exists C. \text{Says } A \ C \ (\text{Crypt}(\text{pubEK } C) \ (\text{Nonce } NB)) \in \text{set } \text{evs}"$   
 $\langle \text{proof} \rangle$

**theorem** *B\_trusts\_NS3*:

" $\llbracket \text{Says } B \ A \ (\text{Crypt}(\text{pubEK } A) \ \{\!\!\{ \text{Nonce } NA, \text{Nonce } NB \}\!\!\}) \in \text{set } \text{evs};$   
 $\text{Says } A' \ B \ (\text{Crypt}(\text{pubEK } B) \ (\text{Nonce } NB)) \in \text{set } \text{evs};$   
 $A \notin \text{bad}; \ B \notin \text{bad}; \ \text{evs} \in \text{ns\_public} \rrbracket$   
 $\Rightarrow \exists C. \text{Says } A \ C \ (\text{Crypt}(\text{pubEK } C) \ (\text{Nonce } NB)) \in \text{set } \text{evs}"$   
 $\langle \text{proof} \rangle$

Can we strengthen the secrecy theorem *Spy\_not\_see\_NB*? NO

**lemma** " $\llbracket \text{evs} \in \text{ns\_public};$

$\text{Says } B \ A \ (\text{Crypt}(\text{pubEK } A) \ \{\!\!\{ \text{Nonce } NA, \text{Nonce } NB \}\!\!\}) \in \text{set } \text{evs};$

```

      A ∉ bad; B ∉ bad]]
    ⇒ Nonce NB ∉ analz (spies evs)"
  <proof>

```

```

end

```

## 20 The Needham-Schroeder Public-Key Protocol

Flawed version, vulnerable to Lowe's attack. From Burrows, Abadi and Needham. A Logic of Authentication. Proc. Royal Soc. 426 (1989), p. 260

```

theory NS_Public imports Public begin

```

```

inductive_set ns_public :: "event list set"
  where
    Nil: "[ ] ∈ ns_public"
    — Initial trace is empty
  | Fake: "[[evsf ∈ ns_public; X ∈ synth (analz (spies evsf))]]
    ⇒ Says Spy B X # evsf ∈ ns_public"
    — The spy can say almost anything.
  | NS1: "[[evs1 ∈ ns_public; Nonce NA ∉ used evs1]]
    ⇒ Says A B (Crypt (pubEK B) {Nonce NA, Agent A})
    # evs1 ∈ ns_public"
    — Alice initiates a protocol run, sending a nonce to Bob
  | NS2: "[[evs2 ∈ ns_public; Nonce NB ∉ used evs2;
    Says A' B (Crypt (pubEK B) {Nonce NA, Agent A}) ∈ set evs2]]
    ⇒ Says B A (Crypt (pubEK A) {Nonce NA, Nonce NB, Agent B})
    # evs2 ∈ ns_public"
    — Bob responds to Alice's message with a further nonce
  | NS3: "[[evs3 ∈ ns_public;
    Says A B (Crypt (pubEK B) {Nonce NA, Agent A}) ∈ set evs3;
    Says B' A (Crypt (pubEK A) {Nonce NA, Nonce NB, Agent B}) ∈ set
    evs3]]
    ⇒ Says A B (Crypt (pubEK B) (Nonce NB)) # evs3 ∈ ns_public"
    — Alice proves her existence by sending NB back to Bob.

```

```

declare knows_Spy_partsEs [elim]
declare analz_into_parts [dest]
declare Fake_parts_insert_in_Un [dest]

```

A "possibility property": there are traces that reach the end

```

lemma "∃ NB. ∃ evs ∈ ns_public. Says A B (Crypt (pubEK B) (Nonce NB)) ∈ set
    evs"
  <proof>

```

### 20.1 Inductive proofs about ns\_public

Spy never sees another agent's private key! (unless it's bad at start)

```

lemma Spy_see_priEK [simp]:

```

```
"evs ∈ ns_public ⇒ (Key (priEK A) ∈ parts (spies evs)) = (A ∈ bad)"
⟨proof⟩
```

**lemma** *Spy\_analz\_priEK [simp]:*

```
"evs ∈ ns_public ⇒ (Key (priEK A) ∈ analz (spies evs)) = (A ∈ bad)"
⟨proof⟩
```

## 20.2 Authenticity properties obtained from term NS1

It is impossible to re-use a nonce in both term NS1 and term NS2, provided the nonce is secret. (Honest users generate fresh nonces.)

**lemma** *no\_nonce\_NS1\_NS2:*

```
"[[evs ∈ ns_public;
  Crypt (pubEK C) {NA', Nonce NA, Agent D} ∈ parts (spies evs);
  Crypt (pubEK B) {Nonce NA, Agent A} ∈ parts (spies evs)]
⇒ Nonce NA ∈ analz (spies evs)"
⟨proof⟩
```

Unicity for term NS1: nonce term NA identifies agents term A and term B

**lemma** *unique\_NA:*

```
assumes NA: "Crypt(pubEK B) {Nonce NA, Agent A} ∈ parts(spies evs)"
          "Crypt(pubEK B') {Nonce NA, Agent A'} ∈ parts(spies evs)"
          "Nonce NA ∉ analz (spies evs)"
and evs: "evs ∈ ns_public"
shows "A=A' ∧ B=B'"
⟨proof⟩
```

Secrecy: Spy does not see the nonce sent in msg term NS1 if term A and term B are secure The major premise "Says A B ..." makes it a dest-rule, hence the given assumption order.

**theorem** *Spy\_not\_see\_NA:*

```
assumes NA: "Says A B (Crypt(pubEK B) {Nonce NA, Agent A}) ∈ set evs"
          "A ∉ bad" "B ∉ bad"
and evs: "evs ∈ ns_public"
shows "Nonce NA ∉ analz (spies evs)"
⟨proof⟩
```

Authentication for term A: if she receives message 2 and has used term NA to start a run, then term B has sent message 2.

**lemma** *A\_trusts\_NS2\_lemma:*

```
"[[evs ∈ ns_public;
  Crypt (pubEK A) {Nonce NA, Nonce NB, Agent B} ∈ parts (spies evs);
  Says A B (Crypt(pubEK B) {Nonce NA, Agent A}) ∈ set evs;
  A ∉ bad; B ∉ bad]
⇒ Says B A (Crypt(pubEK A) {Nonce NA, Nonce NB, Agent B}) ∈ set evs"
⟨proof⟩
```

**theorem** *A\_trusts\_NS2:*

```
"[[Says A B (Crypt(pubEK B) {Nonce NA, Agent A}) ∈ set evs;
  Says B' A (Crypt(pubEK A) {Nonce NA, Nonce NB, Agent B}) ∈ set evs;
  A ∉ bad; B ∉ bad; evs ∈ ns_public]
⇒ Says B A (Crypt(pubEK A) {Nonce NA, Nonce NB, Agent B}) ∈ set evs"
```

*<proof>*

If the encrypted message appears then it originated with Alice in term NS1

**lemma** *B\_trusts\_NS1*:  
 "[*evs* ∈ *ns\_public*;  
   *Crypt* (*pubEK* *B*) {*Nonce* *NA*, *Agent* *A*} ∈ *parts* (*spies evs*);  
   *Nonce* *NA* ∉ *analz* (*spies evs*)]  
 ⇒ *Says* *A* *B* (*Crypt* (*pubEK* *B*) {*Nonce* *NA*, *Agent* *A*}) ∈ *set evs*"  
*<proof>*

### 20.3 Authenticity properties obtained from term NS2

Unicity for term NS2: nonce term NB identifies nonce term NA and agent term A [proof closely follows that for *unique\_NA*]

**lemma** *unique\_NB [dest]*:  
*assumes* *NB*: "*Crypt*(*pubEK* *A*) {*Nonce* *NA*, *Nonce* *NB*, *Agent* *B*} ∈ *parts*(*spies evs*)"  
           "*Crypt*(*pubEK* *A'*) {*Nonce* *NA'*, *Nonce* *NB*, *Agent* *B'*} ∈ *parts*(*spies evs*)"  
           "*Nonce* *NB* ∉ *analz* (*spies evs*)"  
*and evs*: "*evs* ∈ *ns\_public*"  
*shows* "*A=A'* ∧ *NA=NA'* ∧ *B=B'*"  
*<proof>*

term NB remains secret

**theorem** *Spy\_not\_see\_NB [dest]*:  
*assumes* *NB*: "*Says* *B* *A* (*Crypt* (*pubEK* *A*) {*Nonce* *NA*, *Nonce* *NB*, *Agent* *B*}) ∈ *set evs*"  
           "*A* ∉ *bad*" "*B* ∉ *bad*"  
*and evs*: "*evs* ∈ *ns\_public*"  
*shows* "*Nonce* *NB* ∉ *analz* (*spies evs*)"  
*<proof>*

Authentication for term B: if he receives message 3 and has used term NB in message 2, then term A has sent message 3.

**lemma** *B\_trusts\_NS3\_lemma*:  
 "[*evs* ∈ *ns\_public*;  
   *Crypt* (*pubEK* *B*) (*Nonce* *NB*) ∈ *parts* (*spies evs*);  
   *Says* *B* *A* (*Crypt* (*pubEK* *A*) {*Nonce* *NA*, *Nonce* *NB*, *Agent* *B*}) ∈ *set evs*;  
   *A* ∉ *bad*; *B* ∉ *bad*]  
 ⇒ *Says* *A* *B* (*Crypt* (*pubEK* *B*) (*Nonce* *NB*)) ∈ *set evs*"  
*<proof>*

**theorem** *B\_trusts\_NS3*:  
 "[*Says* *B* *A* (*Crypt* (*pubEK* *A*) {*Nonce* *NA*, *Nonce* *NB*, *Agent* *B*}) ∈ *set evs*;  
   *Says* *A'* *B* (*Crypt* (*pubEK* *B*) (*Nonce* *NB*)) ∈ *set evs*;  
   *A* ∉ *bad*; *B* ∉ *bad*; *evs* ∈ *ns\_public*]  
 ⇒ *Says* *A* *B* (*Crypt* (*pubEK* *B*) (*Nonce* *NB*)) ∈ *set evs*"  
*<proof>*



## 20.4 Overall guarantee for term B

If NS3 has been sent and the nonce NB agrees with the nonce B joined with NA, then A initiated the run using NA.

```

theorem B_trusts_protocol:
  "[[A ∉ bad; B ∉ bad; evs ∈ ns_public]] ⇒
    Crypt (pubEK B) (Nonce NB) ∈ parts (spies evs) →
    Says B A (Crypt (pubEK A) {Nonce NA, Nonce NB, Agent B}) ∈ set evs
  →
    Says A B (Crypt (pubEK B) {Nonce NA, Agent A}) ∈ set evs"
  <proof>

end

```

## 21 The TLS Protocol: Transport Layer Security

```

theory TLS imports Public "HOL-Library.Nat_Bijection" begin

```

```

definition certificate :: "[agent,key] ⇒ msg" where
  "certificate A KA == Crypt (priSK Server) {Agent A, Key KA}"

```

TLS apparently does not require separate keypairs for encryption and signature. Therefore, we formalize signature as encryption using the private encryption key.

```

datatype role = ClientRole | ServerRole

```

```

consts

```

```

  PRF :: "nat*nat*nat ⇒ nat"

```

```

  sessionK :: "(nat*nat*nat) * role ⇒ key"

```

```

abbreviation

```

```

  clientK :: "nat*nat*nat ⇒ key" where
  "clientK X == sessionK(X, ClientRole)"

```

```

abbreviation

```

```

  serverK :: "nat*nat*nat ⇒ key" where
  "serverK X == sessionK(X, ServerRole)"

```

```

specification (PRF)

```

```

  inj_PRF: "inj PRF"
  — the pseudo-random function is collision-free
  <proof>

```

```

specification (sessionK)

```

```

  inj_sessionK: "inj sessionK"
  — sessionK is collision-free; also, no clientK clashes with any serverK.
  <proof>

```

```

axiomatization where

```

```

  — sessionK makes symmetric keys

```

*isSym\_sessionK*: "*sessionK nonces*  $\in$  *symKeys*" and

— *sessionK* never clashes with a long-term symmetric key (they don't exist in TLS anyway)

*sessionK\_neq\_shrK* [iff]: "*sessionK nonces*  $\neq$  *shrK A*"

**inductive\_set** *tls* :: "event list set"

**where**

*Nil*: — The initial, empty trace

"[]  $\in$  *tls*"

/ *Fake*: — The Spy may say anything he can say. The sender field is correct, but agents don't use that information.

"[*evsf*  $\in$  *tls*; *X*  $\in$  synth (*analz* (*spies evsf*))]"

$\implies$  Says Spy *B X* # *evsf*  $\in$  *tls*"

/ *SpyKeys*: — The spy may apply *PRF* and *sessionK* to available nonces

"[*evsSK*  $\in$  *tls*;

{*Nonce NA*, *Nonce NB*, *Nonce M*}  $\subseteq$  *analz* (*spies evsSK*)]"

$\implies$  Notes Spy { *Nonce* (*PRF*(*M*,*NA*,*NB*)),

*Key* (*sessionK*((*NA*,*NB*,*M*),*role*)) } # *evsSK*  $\in$  *tls*"

/ *ClientHello*:

— (7.4.1.2) PA represents *CLIENT\_VERSION*, *CIPHER\_SUITES* and *COMPRESSION\_METHODS*. It is uninterpreted but will be confirmed in the FINISHED messages. NA is CLIENT RANDOM, while SID is *SESSION\_ID*. UNIX TIME is omitted because the protocol doesn't use it. May assume *NA*  $\notin$  range *PRF* because CLIENT RANDOM is 28 bytes while MASTER SECRET is 48 bytes

"[*evsCH*  $\in$  *tls*; *Nonce NA*  $\notin$  used *evsCH*; *NA*  $\notin$  range *PRF*]"

$\implies$  Says A B {*Agent A*, *Nonce NA*, *Number SID*, *Number PA*}

# *evsCH*  $\in$  *tls*"

/ *ServerHello*:

— 7.4.1.3 of the TLS Internet-Draft PB represents *CLIENT\_VERSION*, *CIPHER\_SUITE* and *COMPRESSION\_METHOD*. SERVER CERTIFICATE (7.4.2) is always present. *CERTIFICATE\_REQUEST* (7.4.4) is implied.

"[*evsSH*  $\in$  *tls*; *Nonce NB*  $\notin$  used *evsSH*; *NB*  $\notin$  range *PRF*;

Says A' B {*Agent A*, *Nonce NA*, *Number SID*, *Number PA*}

$\in$  set *evsSH*]"

$\implies$  Says B A {*Nonce NB*, *Number SID*, *Number PB*} # *evsSH*  $\in$  *tls*"

/ *Certificate*:

— SERVER (7.4.2) or CLIENT (7.4.6) CERTIFICATE.

"*evsC*  $\in$  *tls*  $\implies$  Says B A (*certificate B* (*pubK B*)) # *evsC*  $\in$  *tls*"

/ *ClientKeyExch*:

— CLIENT KEY EXCHANGE (7.4.7). The client, A, chooses PMS, the PREMASTER SECRET. She encrypts PMS using the supplied KB, which ought to be pubK B. We assume *PMS*  $\notin$  range *PRF* because a clash between the PMS and another MASTER SECRET is highly unlikely (even though both items have the same length, 48 bytes). The Note event records in the trace that she knows PMS (see REMARK at top).

"[*evsCX*  $\in$  *tls*; *Nonce PMS*  $\notin$  used *evsCX*; *PMS*  $\notin$  range *PRF*;

```

    Says B' A (certificate B KB) ∈ set evsCX]]
  ⇒ Says A B (Crypt KB (Nonce PMS))
    # Notes A {Agent B, Nonce PMS}
    # evsCX ∈ tls"

```

/ CertVerify:

— The optional Certificate Verify (7.4.8) message contains the specific components listed in the security analysis, F.1.1.2. It adds the pre-master-secret, which is also essential! Checking the signature, which is the only use of A's certificate, assures B of A's presence

```

  "[evsCV ∈ tls;
    Says B' A {Nonce NB, Number SID, Number PB} ∈ set evsCV;
    Notes A {Agent B, Nonce PMS} ∈ set evsCV]]
  ⇒ Says A B (Crypt (priK A) (Hash{Nonce NB, Agent B, Nonce PMS}))
    # evsCV ∈ tls"

```

— Finally come the FINISHED messages (7.4.8), confirming PA and PB among other things. The master-secret is PRF(PMS,NA,NB). Either party may send its message first.

/ ClientFinished:

— The occurrence of Notes A {Agent B, Nonce PMS} stops the rule's applying when the Spy has satisfied the Says A B by repaying messages sent by the true client; in that case, the Spy does not know PMS and could not send ClientFinished. One could simply put  $A \neq \text{Spy}$  into the rule, but one should not expect the spy to be well-behaved.

```

  "[evsCF ∈ tls;
    Says A B {Agent A, Nonce NA, Number SID, Number PA}
      ∈ set evsCF;
    Says B' A {Nonce NB, Number SID, Number PB} ∈ set evsCF;
    Notes A {Agent B, Nonce PMS} ∈ set evsCF;
    M = PRF(PMS,NA,NB)]
  ⇒ Says A B (Crypt (clientK(NA,NB,M))
    (Hash{Number SID, Nonce M,
      Nonce NA, Number PA, Agent A,
      Nonce NB, Number PB, Agent B}))
    # evsCF ∈ tls"

```

/ ServerFinished:

— Keeping A' and A'' distinct means B cannot even check that the two messages originate from the same source.

```

  "[evsSF ∈ tls;
    Says A' B {Agent A, Nonce NA, Number SID, Number PA}
      ∈ set evsSF;
    Says B A {Nonce NB, Number SID, Number PB} ∈ set evsSF;
    Says A'' B (Crypt (pubK B) (Nonce PMS)) ∈ set evsSF;
    M = PRF(PMS,NA,NB)]
  ⇒ Says B A (Crypt (serverK(NA,NB,M))
    (Hash{Number SID, Nonce M,
      Nonce NA, Number PA, Agent A,
      Nonce NB, Number PB, Agent B}))
    # evsSF ∈ tls"

```

/ ClientAccepts:

— Having transmitted ClientFinished and received an identical message encrypted with serverK, the client stores the parameters needed to resume this session. The "Notes A ..." premise is used to prove *Notes\_master\_imp\_Crypt\_PMS*.

```
"[evsCA ∈ tls;
  Notes A {Agent B, Nonce PMS} ∈ set evsCA;
  M = PRF(PMS, NA, NB);
  X = Hash{Number SID, Nonce M,
           Nonce NA, Number PA, Agent A,
           Nonce NB, Number PB, Agent B};
  Says A B (Crypt (clientK(NA, NB, M)) X) ∈ set evsCA;
  Says B' A (Crypt (serverK(NA, NB, M)) X) ∈ set evsCA]
⇒
  Notes A {Number SID, Agent A, Agent B, Nonce M} # evsCA ∈ tls"
```

/ *ServerAccepts*:

— Having transmitted ServerFinished and received an identical message encrypted with clientK, the server stores the parameters needed to resume this session. The "Says A" B ..." premise is used to prove *Notes\_master\_imp\_Crypt\_PMS*.

```
"[evsSA ∈ tls;
  A ≠ B;
  Says A' B (Crypt (pubK B) (Nonce PMS)) ∈ set evsSA;
  M = PRF(PMS, NA, NB);
  X = Hash{Number SID, Nonce M,
           Nonce NA, Number PA, Agent A,
           Nonce NB, Number PB, Agent B};
  Says B A (Crypt (serverK(NA, NB, M)) X) ∈ set evsSA;
  Says A' B (Crypt (clientK(NA, NB, M)) X) ∈ set evsSA]
⇒
  Notes B {Number SID, Agent A, Agent B, Nonce M} # evsSA ∈ tls"
```

/ *ClientResume*:

— If A recalls the *SESSION\_ID*, then she sends a FINISHED message using the new nonces and stored MASTER SECRET.

```
"[evsCR ∈ tls;
  Says A B {Agent A, Nonce NA, Number SID, Number PA} ∈ set evsCR;
  Says B' A {Nonce NB, Number SID, Number PB} ∈ set evsCR;
  Notes A {Number SID, Agent A, Agent B, Nonce M} ∈ set evsCR]
⇒ Says A B (Crypt (clientK(NA, NB, M))
              (Hash{Number SID, Nonce M,
                   Nonce NA, Number PA, Agent A,
                   Nonce NB, Number PB, Agent B}))
  # evsCR ∈ tls"
```

/ *ServerResume*:

— Resumption (7.3): If B finds the *SESSION\_ID* then he can send a FINISHED message using the recovered MASTER SECRET

```
"[evsSR ∈ tls;
  Says A' B {Agent A, Nonce NA, Number SID, Number PA} ∈ set evsSR;
  Says B A {Nonce NB, Number SID, Number PB} ∈ set evsSR;
  Notes B {Number SID, Agent A, Agent B, Nonce M} ∈ set evsSR]
⇒ Says B A (Crypt (serverK(NA, NB, M))
              (Hash{Number SID, Nonce M,
                   Nonce NA, Number PA, Agent A,
                   Nonce NB, Number PB, Agent B})) # evsSR"
```

$\in \text{tls}$ "

*/ Ops:*

— The most plausible compromise is of an old session key. Losing the MASTER SECRET or PREMASTER SECRET is more serious but rather unlikely. The assumption  $A \neq \text{Spy}$  is essential: otherwise the Spy could learn session keys merely by replaying messages!

$\llbracket \text{evso} \in \text{tls}; A \neq \text{Spy};$   
 $\text{Says } A \ B \ (\text{Crypt } (\text{sessionK}((NA, NB, M), \text{role})) \ X) \in \text{set evso} \rrbracket$   
 $\implies \text{Says } A \ \text{Spy} \ (\text{Key } (\text{sessionK}((NA, NB, M), \text{role}))) \ \# \ \text{evso} \in \text{tls}$

**declare** *Says\_imp\_knows\_Spy* [THEN *analz.Inj*, *dest*]  
**declare** *parts.Body* [*dest*]  
**declare** *analz\_into\_parts* [*dest*]  
**declare** *Fake\_parts\_insert\_in\_Un* [*dest*]

Automatically unfold the definition of "certificate"

**declare** *certificate\_def* [*simp*]

Injectiveness of key-generating functions

**declare** *inj\_PRF* [THEN *inj\_eq*, *iff*]  
**declare** *inj\_sessionK* [THEN *inj\_eq*, *iff*]  
**declare** *isSym\_sessionK* [*simp*]

**lemma** *pubK\_neq\_sessionK* [*iff*]: "publicKey b A  $\neq$  sessionK arg"  
 $\langle \text{proof} \rangle$

**declare** *pubK\_neq\_sessionK* [THEN *not\_sym*, *iff*]

**lemma** *priK\_neq\_sessionK* [*iff*]: "invKey (publicKey b A)  $\neq$  sessionK arg"  
 $\langle \text{proof} \rangle$

**declare** *priK\_neq\_sessionK* [THEN *not\_sym*, *iff*]

**lemmas** *keys\_distinct* = *pubK\_neq\_sessionK priK\_neq\_sessionK*

## 21.1 Protocol Proofs

Possibility properties state that some traces run the protocol to the end. Four paths and 12 rules are considered.

Possibility property ending with ClientAccepts.

**lemma** " $\llbracket \forall \text{evs}. (\text{SOME } N. \text{Nonce } N \notin \text{used evs}) \notin \text{range PRF}; A \neq B \rrbracket$   
 $\implies \exists \text{SID } M. \exists \text{evs} \in \text{tls}.$   
 $\text{Notes } A \ \{ \text{Number } \text{SID}, \text{Agent } A, \text{Agent } B, \text{Nonce } M \} \in \text{set evs}$ "  
 $\langle \text{proof} \rangle$

And one for ServerAccepts. Either FINISHED message may come first.

**lemma** "[ $\forall \text{ evs. (SOME } N. \text{ Nonce } N \notin \text{ used evs}) \notin \text{ range PRF; } A \neq B$ ]  
 $\implies \exists \text{ SID NA PA NB PB M. } \exists \text{ evs} \in \text{ tls.}$   
 $\text{Notes } B \{ \text{Number SID, Agent A, Agent B, Nonce M} \} \in \text{ set evs}$ "  
 <proof>

Another one, for CertVerify (which is optional)

**lemma** "[ $\forall \text{ evs. (SOME } N. \text{ Nonce } N \notin \text{ used evs}) \notin \text{ range PRF; } A \neq B$ ]  
 $\implies \exists \text{ NB PMS. } \exists \text{ evs} \in \text{ tls.}$   
 $\text{Says } A \text{ } B \text{ (Crypt (priK } A) \text{ (Hash } \{ \text{Nonce NB, Agent B, Nonce PMS} \}))}$   
 $\in \text{ set evs}$ "  
 <proof>

Another one, for session resumption (both ServerResume and ClientResume).  
 NO tls.Nil here: we refer to a previous session, not the empty trace.

**lemma** "[ $\text{evs0} \in \text{ tls};$   
 $\text{Notes } A \{ \text{Number SID, Agent A, Agent B, Nonce M} \} \in \text{ set evs0};$   
 $\text{Notes } B \{ \text{Number SID, Agent A, Agent B, Nonce M} \} \in \text{ set evs0};$   
 $\forall \text{ evs. (SOME } N. \text{ Nonce } N \notin \text{ used evs}) \notin \text{ range PRF;}$   
 $A \neq B$ ]  
 $\implies \exists \text{ NA PA NB PB X. } \exists \text{ evs} \in \text{ tls.}$   
 $X = \text{Hash } \{ \text{Number SID, Nonce M,}$   
 $\text{Nonce NA, Number PA, Agent A,}$   
 $\text{Nonce NB, Number PB, Agent B} \} \wedge$   
 $\text{Says } A \text{ } B \text{ (Crypt (clientK(NA,NB,M)) } X) \in \text{ set evs} \wedge$   
 $\text{Says } B \text{ } A \text{ (Crypt (serverK(NA,NB,M)) } X) \in \text{ set evs}$ "  
 <proof>

## 21.2 Inductive proofs about tls

Spy never sees a good agent's private key!

**lemma** *Spy\_see\_priK [simp]:*  
 $"\text{evs} \in \text{ tls} \implies (\text{Key (privateKey } b \text{ } A) \in \text{ parts (spies evs)}) = (A \in \text{ bad})"$   
 <proof>

**lemma** *Spy\_analz\_priK [simp]:*  
 $"\text{evs} \in \text{ tls} \implies (\text{Key (privateKey } b \text{ } A) \in \text{ analz (spies evs)}) = (A \in \text{ bad})"$   
 <proof>

**lemma** *Spy\_see\_priK\_D [dest!]:*  
 $"[\text{Key (privateKey } b \text{ } A) \in \text{ parts (knows Spy evs); } \text{evs} \in \text{ tls}] \implies A \in \text{ bad}"$   
 <proof>

This lemma says that no false certificates exist. One might extend the model to include bogus certificates for the agents, but there seems little point in doing so: the loss of their private keys is a worse breach of security.

**lemma** *certificate\_valid:*  
 $"[\text{certificate } B \text{ } KB \in \text{ parts (spies evs); } \text{evs} \in \text{ tls}] \implies KB = \text{pubK } B"$   
 <proof>

**lemmas** *CX\_KB\_is\_pubKB = Says\_imp\_spies [THEN parts.Inj, THEN certificate\_valid]*

### 21.2.1 Properties of items found in Notes

**lemma** *Notes\_Crypt\_parts\_spies*:  
 "[Notes A {Agent B, X} ∈ set evs; evs ∈ tls]  
 ⇒ Crypt (pubK B) X ∈ parts (spies evs)"  
 <proof>

C may be either A or B

**lemma** *Notes\_master\_imp\_Crypt\_PMS*:  
 "[Notes C {s, Agent A, Agent B, Nonce(PRF(PMS, NA, NB))} ∈ set evs;  
 evs ∈ tls]  
 ⇒ Crypt (pubK B) (Nonce PMS) ∈ parts (spies evs)"  
 <proof>

Compared with the theorem above, both premise and conclusion are stronger

**lemma** *Notes\_master\_imp\_Notes\_PMS*:  
 "[Notes A {s, Agent A, Agent B, Nonce(PRF(PMS, NA, NB))} ∈ set evs;  
 evs ∈ tls]  
 ⇒ Notes A {Agent B, Nonce PMS} ∈ set evs"  
 <proof>

### 21.2.2 Protocol goal: if B receives CertVerify, then A sent it

B can check A's signature if he has received A's certificate.

**lemma** *TrustCertVerify\_lemma*:  
 "[X ∈ parts (spies evs);  
 X = Crypt (priK A) (Hash{nb, Agent B, pms});  
 evs ∈ tls; A ∉ bad]  
 ⇒ Says A B X ∈ set evs"  
 <proof>

Final version: B checks X using the distributed KA instead of priK A

**lemma** *TrustCertVerify*:  
 "[X ∈ parts (spies evs);  
 X = Crypt (invKey KA) (Hash{nb, Agent B, pms});  
 certificate A KA ∈ parts (spies evs);  
 evs ∈ tls; A ∉ bad]  
 ⇒ Says A B X ∈ set evs"  
 <proof>

If CertVerify is present then A has chosen PMS.

**lemma** *UseCertVerify\_lemma*:  
 "[Crypt (priK A) (Hash{nb, Agent B, Nonce PMS}) ∈ parts (spies evs);  
 evs ∈ tls; A ∉ bad]  
 ⇒ Notes A {Agent B, Nonce PMS} ∈ set evs"  
 <proof>

Final version using the distributed KA instead of priK A

**lemma** *UseCertVerify*:  
 "[Crypt (invKey KA) (Hash{nb, Agent B, Nonce PMS})  
 ∈ parts (spies evs);  
 certificate A KA ∈ parts (spies evs);

```

    evs ∈ tls; A ∉ bad]]
  ⇒ Notes A {Agent B, Nonce PMS} ∈ set evs"
⟨proof⟩

```

```

lemma no_Notes_A_PRF [simp]:
  "evs ∈ tls ⇒ Notes A {Agent B, Nonce (PRF x)} ∉ set evs"
⟨proof⟩

```

```

lemma MS_imp_PMS [dest!]:
  "[Nonce (PRF (PMS,NA,NB)) ∈ parts (spies evs); evs ∈ tls]
  ⇒ Nonce PMS ∈ parts (spies evs)"
⟨proof⟩

```

### 21.2.3 Unicity results for PMS, the pre-master-secret

PMS determines B.

```

lemma Crypt_unique_PMS:
  "[Crypt(pubK B) (Nonce PMS) ∈ parts (spies evs);
   Crypt(pubK B') (Nonce PMS) ∈ parts (spies evs);
   Nonce PMS ∉ analz (spies evs);
   evs ∈ tls]
  ⇒ B=B'"
⟨proof⟩

```

In A's internal Note, PMS determines A and B.

```

lemma Notes_unique_PMS:
  "[Notes A {Agent B, Nonce PMS} ∈ set evs;
   Notes A' {Agent B', Nonce PMS} ∈ set evs;
   evs ∈ tls]
  ⇒ A=A' ∧ B=B'"
⟨proof⟩

```

## 21.3 Secrecy Theorems

Key compromise lemma needed to prove *analz\_image\_keys*. No collection of keys can help the spy get new private keys.

```

lemma analz_image_priK [rule_format]:
  "evs ∈ tls
  ⇒ ∀ KK. (Key(priK B) ∈ analz (Key'KK ∪ (spies evs))) =
    (priK B ∈ KK | B ∈ bad)"
⟨proof⟩

```

slightly speeds up the big simplification below

```

lemma range_sessionkeys_not_priK:
  "KK ⊆ range sessionK ⇒ priK B ∉ KK"
⟨proof⟩

```

Lemma for the trivial direction of the if-and-only-if

```

lemma analz_image_keys_lemma:
  "(X ∈ analz (G ∪ H)) ⇒ (X ∈ analz H) ⇒

```



$(X \in \text{analz } (G \cup H)) = (X \in \text{analz } H)$ "  
 <proof>

**lemma** *analz\_image\_keys* [rule\_format]:  
 "evs ∈ tls ⇒  
 ∀ KK. KK ⊆ range sessionK →  
 (Nonce N ∈ analz (Key'KK ∪ (spies evs))) =  
 (Nonce N ∈ analz (spies evs))"  
 <proof>

Knowing some session keys is no help in getting new nonces

**lemma** *analz\_insert\_key* [simp]:  
 "evs ∈ tls ⇒  
 (Nonce N ∈ analz (insert (Key (sessionK z)) (spies evs))) =  
 (Nonce N ∈ analz (spies evs))"  
 <proof>

### 21.3.1 Protocol goal: serverK(Na,Nb,M) and clientK(Na,Nb,M) remain secure

Lemma: session keys are never used if PMS is fresh. Nonces don't have to agree, allowing session resumption. Converse doesn't hold; revealing PMS doesn't force the keys to be sent. THEY ARE NOT SUITABLE AS SAFE ELIM RULES.

**lemma** *PMS\_lemma*:  
 "[Nonce PMS ∉ parts (spies evs);  
 K = sessionK((Na, Nb, PRF(PMS,NA,NB)), role);  
 evs ∈ tls]  
 ⇒ Key K ∉ parts (spies evs) ∧ (∀ Y. Crypt K Y ∉ parts (spies evs))"  
 <proof>

**lemma** *PMS\_sessionK\_not\_spied*:  
 "[Key (sessionK((Na, Nb, PRF(PMS,NA,NB)), role)) ∈ parts (spies evs);  
 evs ∈ tls]  
 ⇒ Nonce PMS ∈ parts (spies evs)"  
 <proof>

**lemma** *PMS\_Crypt\_sessionK\_not\_spied*:  
 "[Crypt (sessionK((Na, Nb, PRF(PMS,NA,NB)), role)) Y  
 ∈ parts (spies evs); evs ∈ tls]  
 ⇒ Nonce PMS ∈ parts (spies evs)"  
 <proof>

Write keys are never sent if M (MASTER SECRET) is secure. Converse fails; betraying M doesn't force the keys to be sent! The strong Oops condition can be weakened later by unicity reasoning, with some effort. NO LONGER USED: see *clientK\_not\_spied* and *serverK\_not\_spied*

**lemma** *sessionK\_not\_spied*:  
 "[∀ A. Says A Spy (Key (sessionK((NA,NB,M),role))) ∉ set evs;  
 Nonce M ∉ analz (spies evs); evs ∈ tls]  
 ⇒ Key (sessionK((NA,NB,M),role)) ∉ parts (spies evs)"

*<proof>*

If A sends ClientKeyExch to an honest B, then the PMS will stay secret.

**lemma** *Spy\_not\_see\_PMS*:  
 "[Notes A {Agent B, Nonce PMS} ∈ set evs;  
 evs ∈ tls; A ∉ bad; B ∉ bad]  
 ⇒ Nonce PMS ∉ analz (spies evs)"  
*<proof>*

If A sends ClientKeyExch to an honest B, then the MASTER SECRET will stay secret.

**lemma** *Spy\_not\_see\_MS*:  
 "[Notes A {Agent B, Nonce PMS} ∈ set evs;  
 evs ∈ tls; A ∉ bad; B ∉ bad]  
 ⇒ Nonce (PRF(PMS, NA, NB)) ∉ analz (spies evs)"  
*<proof>*

### 21.3.2 Weakening the Oops conditions for leakage of clientK

If A created PMS then nobody else (except the Spy in replays) would send a message using a clientK generated from that PMS.

**lemma** *Says\_clientK\_unique*:  
 "[Says A' B' (Crypt (clientK(Na, Nb, PRF(PMS, NA, NB))) Y) ∈ set evs;  
 Notes A {Agent B, Nonce PMS} ∈ set evs;  
 evs ∈ tls; A' ≠ Spy]  
 ⇒ A = A'"  
*<proof>*

If A created PMS and has not leaked her clientK to the Spy, then it is completely secure: not even in parts!

**lemma** *clientK\_not\_spied*:  
 "[Notes A {Agent B, Nonce PMS} ∈ set evs;  
 Says A Spy (Key (clientK(Na, Nb, PRF(PMS, NA, NB)))) ∉ set evs;  
 A ∉ bad; B ∉ bad;  
 evs ∈ tls]  
 ⇒ Key (clientK(Na, Nb, PRF(PMS, NA, NB))) ∉ parts (spies evs)"  
*<proof>*

### 21.3.3 Weakening the Oops conditions for leakage of serverK

If A created PMS for B, then nobody other than B or the Spy would send a message using a serverK generated from that PMS.

**lemma** *Says\_serverK\_unique*:  
 "[Says B' A' (Crypt (serverK(Na, Nb, PRF(PMS, NA, NB))) Y) ∈ set evs;  
 Notes A {Agent B, Nonce PMS} ∈ set evs;  
 evs ∈ tls; A ∉ bad; B ∉ bad; B' ≠ Spy]  
 ⇒ B = B'"  
*<proof>*

If A created PMS for B, and B has not leaked his serverK to the Spy, then it is completely secure: not even in parts!

**lemma** *serverK\_not\_spied*:

```
"[[Notes A {Agent B, Nonce PMS} ∈ set evs;
  Says B Spy (Key(serverK(Na,Nb,PRF(PMS,NA,NB)))) ∉ set evs;
  A ∉ bad; B ∉ bad; evs ∈ tls]]
⇒ Key (serverK(Na,Nb,PRF(PMS,NA,NB))) ∉ parts (spies evs)"
⟨proof⟩
```

**21.3.4 Protocol goals: if A receives ServerFinished, then B is present and has used the quoted values PA, PB, etc. Note that it is up to A to compare PA with what she originally sent.**

The mention of her name (A) in X assures A that B knows who she is.

**lemma** *TrustServerFinished [rule\_format]*:

```
"[[X = Crypt (serverK(Na,Nb,M))
  (Hash {Number SID, Nonce M,
         Nonce Na, Number PA, Agent A,
         Nonce Nb, Number PB, Agent B})];
  M = PRF(PMS,NA,NB);
  evs ∈ tls; A ∉ bad; B ∉ bad]]
⇒ Says B Spy (Key(serverK(Na,Nb,M))) ∉ set evs →
  Notes A {Agent B, Nonce PMS} ∈ set evs →
  X ∈ parts (spies evs) → Says B A X ∈ set evs"
⟨proof⟩
```

This version refers not to ServerFinished but to any message from B. We don't assume B has received CertVerify, and an intruder could have changed A's identity in all other messages, so we can't be sure that B sends his message to A. If CLIENT KEY EXCHANGE were augmented to bind A's identity with PMS, then we could replace A' by A below.

**lemma** *TrustServerMsg [rule\_format]*:

```
"[[M = PRF(PMS,NA,NB); evs ∈ tls; A ∉ bad; B ∉ bad]]
⇒ Says B Spy (Key(serverK(Na,Nb,M))) ∉ set evs →
  Notes A {Agent B, Nonce PMS} ∈ set evs →
  Crypt (serverK(Na,Nb,M)) Y ∈ parts (spies evs) →
  (∃ A'. Says B A' (Crypt (serverK(Na,Nb,M)) Y) ∈ set evs)"
⟨proof⟩
```

**21.3.5 Protocol goal: if B receives any message encrypted with clientK then A has sent it**

ASSUMING that A chose PMS. Authentication is assumed here; B cannot verify it. But if the message is ClientFinished, then B can then check the quoted values PA, PB, etc.

**lemma** *TrustClientMsg [rule\_format]*:

```
"[[M = PRF(PMS,NA,NB); evs ∈ tls; A ∉ bad; B ∉ bad]]
⇒ Says A Spy (Key(clientK(Na,Nb,M))) ∉ set evs →
  Notes A {Agent B, Nonce PMS} ∈ set evs →
  Crypt (clientK(Na,Nb,M)) Y ∈ parts (spies evs) →
  Says A B (Crypt (clientK(Na,Nb,M)) Y) ∈ set evs"
⟨proof⟩
```

**21.3.6 Protocol goal:** if B receives ClientFinished, and if B is able to check a CertVerify from A, then A has used the quoted values PA, PB, etc. Even this one requires A to be uncompromised.

```

lemma AuthClientFinished:
  "  $\llbracket M = \text{PRF}(PMS, NA, NB);$ 
    Says A Spy (Key(clientK(Na,Nb,M)))  $\notin$  set evs;
    Says A' B (Crypt (clientK(Na,Nb,M)) Y)  $\in$  set evs;
    certificate A KA  $\in$  parts (spies evs);
    Says A'' B (Crypt (invKey KA) (Hash  $\{nb, \text{Agent } B, \text{Nonce } PMS\}$ ))
       $\in$  set evs;
    evs  $\in$  tls; A  $\notin$  bad; B  $\notin$  bad  $\rrbracket$ 
 $\implies$  Says A B (Crypt (clientK(Na,Nb,M)) Y)  $\in$  set evs"
<proof>

```

end

## 22 The Certified Electronic Mail Protocol by Abadi et al.

```

theory CertifiedEmail imports Public begin

```

**abbreviation**

```

  TTP :: agent where
  "TTP == Server"

```

**abbreviation**

```

  RPwd :: "agent  $\Rightarrow$  key" where
  "RPwd == shrK"

```

**consts**

```

  NoAuth    :: nat
  TTPAuth   :: nat
  SAuth     :: nat
  BothAuth  :: nat

```

We formalize a fixed way of computing responses. Could be better.

**definition** "response" :: "agent  $\Rightarrow$  agent  $\Rightarrow$  nat  $\Rightarrow$  msg" where  
 "response S R q == Hash {Agent S, Key (shrK R), Nonce q}"

**inductive\_set** certified\_mail :: "event list set"  
 where

Nil: — The empty trace  
 "[]  $\in$  certified\_mail"

/ Fake: — The Spy may say anything he can say. The sender field is correct, but agents don't use that information.

"[evsf  $\in$  certified\_mail; X  $\in$  synth(analz(spies evsf))]  
 $\Rightarrow$  Says Spy B X # evsf  $\in$  certified\_mail"

/ FakeSSL: — The Spy may open SSL sessions with TTP, who is the only agent equipped with the necessary credentials to serve as an SSL server.

"[evsfssl  $\in$  certified\_mail; X  $\in$  synth(analz(spies evsfssl))]  
 $\Rightarrow$  Notes TTP {Agent Spy, Agent TTP, X} # evsfssl  $\in$  certified\_mail"

/ CM1: — The sender approaches the recipient. The message is a number.

"[evs1  $\in$  certified\_mail;  
 Key K  $\notin$  used evs1;  
 K  $\in$  symKeys;  
 Nonce q  $\notin$  used evs1;  
 hs = Hash{Number cleartext, Nonce q, response S R q, Crypt K (Number m)};  
 S2TTP = Crypt(pubEK TTP) {Agent S, Number BothAuth, Key K, Agent R, hs};  
 $\Rightarrow$  Says S R {Agent S, Agent TTP, Crypt K (Number m), Number BothAuth,  
 Number cleartext, Nonce q, S2TTP} # evs1  
 $\in$  certified\_mail"

/ CM2: — The recipient records S2TTP while transmitting it and her password to TTP over an SSL channel.

"[evs2  $\in$  certified\_mail;  
 Gets R {Agent S, Agent TTP, em, Number BothAuth, Number cleartext,  
 Nonce q, S2TTP}  $\in$  set evs2;  
 TTP  $\neq$  R;  
 hr = Hash {Number cleartext, Nonce q, response S R q, em};  
 $\Rightarrow$   
 Notes TTP {Agent R, Agent TTP, S2TTP, Key(RPwd R), hr} # evs2  
 $\in$  certified\_mail"

/ CM3: — TTP simultaneously reveals the key to the recipient and gives a receipt to the sender. The SSL channel does not authenticate the client (R), but TTP accepts the message only if the given password is that of the claimed sender, R. He replies over the established SSL channel.

"[evs3  $\in$  certified\_mail;  
 Notes TTP {Agent R, Agent TTP, S2TTP, Key(RPwd R), hr}  $\in$  set evs3;  
 S2TTP = Crypt (pubEK TTP)  
 {Agent S, Number BothAuth, Key k, Agent R, hs};  
 TTP  $\neq$  R; hs = hr; k  $\in$  symKeys]  
 $\Rightarrow$   
 Notes R {Agent TTP, Agent R, Key k, hr} #  
 Gets S (Crypt (priSK TTP) S2TTP) #

```

    Says TTP S (Crypt (priSK TTP) S2TTP) # evs3 ∈ certified_mail"

/ Reception:
"[[evsr ∈ certified_mail; Says A B X ∈ set evsr]]
  ⇒ Gets B X#evsr ∈ certified_mail"

declare Says_imp_knows_Spy [THEN analz.Inj, dest]
declare analz_into_parts [dest]

lemma "[Key K ∉ used []; K ∈ symKeys] ⇒
  ∃ S2TTP. ∃ evs ∈ certified_mail.
    Says TTP S (Crypt (priSK TTP) S2TTP) ∈ set evs"
⟨proof⟩

lemma Gets_imp_Says:
"[[Gets B X ∈ set evs; evs ∈ certified_mail]] ⇒ ∃ A. Says A B X ∈ set evs"
⟨proof⟩

lemma Gets_imp_parts_knows_Spy:
"[[Gets A X ∈ set evs; evs ∈ certified_mail]] ⇒ X ∈ parts(spies evs)"
⟨proof⟩

lemma CM2_S2TTP_analz_knows_Spy:
"[[Gets R {Agent A, Agent B, em, Number A0, Number cleartext,
  Nonce q, S2TTP} ∈ set evs;
  evs ∈ certified_mail]]
  ⇒ S2TTP ∈ analz(spies evs)"
⟨proof⟩

lemmas CM2_S2TTP_parts_knows_Spy =
  CM2_S2TTP_analz_knows_Spy [THEN analz_subset_parts [THEN subsetD]]

lemma hr_form_lemma [rule_format]:
"evs ∈ certified_mail
  ⇒ hr ∉ synth (analz (spies evs)) →
  (∀ S2TTP. Notes TTP {Agent R, Agent TTP, S2TTP, pwd, hr}
    ∈ set evs →
    (∃ clt q S em. hr = Hash {Number clt, Nonce q, response S R q, em}))"
⟨proof⟩

Cannot strengthen the first disjunct to  $R \neq \text{Spy}$  because the fakessl rule allows
Spy to spoof the sender's name. Maybe can strengthen the second disjunct with
 $R \neq \text{Spy}$ .

lemma hr_form:
"[[Notes TTP {Agent R, Agent TTP, S2TTP, pwd, hr} ∈ set evs;
  evs ∈ certified_mail]]
  ⇒ hr ∈ synth (analz (spies evs)) /
  (∃ clt q S em. hr = Hash {Number clt, Nonce q, response S R q, em}))"
⟨proof⟩

```

```

lemma Spy_dont_know_private_keys [dest!]:
  "[[Key (privateKey b A) ∈ parts (spies evs); evs ∈ certified_mail]]
  ⇒ A ∈ bad"
<proof>

lemma Spy_know_private_keys_iff [simp]:
  "evs ∈ certified_mail
  ⇒ (Key (privateKey b A) ∈ parts (spies evs)) = (A ∈ bad)"
<proof>

lemma Spy_dont_know_TTPKey_parts [simp]:
  "evs ∈ certified_mail ⇒ Key (privateKey b TTP) ∉ parts(spies evs)"
<proof>

lemma Spy_dont_know_TTPKey_analz [simp]:
  "evs ∈ certified_mail ⇒ Key (privateKey b TTP) ∉ analz(spies evs)"
<proof>

Thus, prove any goal that assumes that Spy knows a private key belonging to
TTP

declare Spy_dont_know_TTPKey_parts [THEN [2] rev_notE, elim!]

lemma CM3_k_parts_knows_Spy:
  "[[evs ∈ certified_mail;
    Notes TTP {Agent A, Agent TTP,
      Crypt (pubEK TTP) {Agent S, Number AO, Key K,
        Agent R, hs}, Key (RPwd R), hs} ∈ set evs]]
  ⇒ Key K ∈ parts(spies evs)"
<proof>

lemma Spy_dont_know_RPwd [rule_format]:
  "evs ∈ certified_mail ⇒ Key (RPwd A) ∈ parts(spies evs) → A ∈ bad"
<proof>

lemma Spy_know_RPwd_iff [simp]:
  "evs ∈ certified_mail ⇒ (Key (RPwd A) ∈ parts(spies evs)) = (A ∈ bad)"
<proof>

lemma Spy_analz_RPwd_iff [simp]:
  "evs ∈ certified_mail ⇒ (Key (RPwd A) ∈ analz(spies evs)) = (A ∈ bad)"
<proof>

Unused, but a guarantee of sorts

theorem CertAutenticity:
  "[[Crypt (priSK TTP) X ∈ parts (spies evs); evs ∈ certified_mail]]
  ⇒ ∃ A. Says TTP A (Crypt (priSK TTP) X) ∈ set evs"
<proof>

```

## 22.1 Proving Confidentiality Results

```

lemma analz_image_freshK [rule_format]:

```

```

"evs ∈ certified_mail ⇒
  ∀ K KK. invKey (pubEK TTP) ∉ KK →
    (Key K ∈ analz (Key 'KK ∪ (spies evs))) =
      (K ∈ KK | Key K ∈ analz (spies evs))"
⟨proof⟩

```

```

lemma analz_insert_freshK:
  "[[evs ∈ certified_mail; KAB ≠ invKey (pubEK TTP)] ⇒
    (Key K ∈ analz (insert (Key KAB) (spies evs))) =
      (K = KAB | Key K ∈ analz (spies evs))]"
⟨proof⟩

```

S2TTP must have originated from a valid sender provided  $K$  is secure. Proof is surprisingly hard.

```

lemma Notes_SSL_imp_used:
  "[[Notes B {Agent A, Agent B, X} ∈ set evs] ⇒ X ∈ used evs]"
⟨proof⟩

```

```

lemma S2TTP_sender_lemma [rule_format]:
  "evs ∈ certified_mail ⇒
    Key K ∉ analz (spies evs) →
    (∀ AO. Crypt (pubEK TTP)
      {Agent S, Number AO, Key K, Agent R, hs} ∈ used evs →
    (∃ m ctxt q.
      hs = Hash {Number ctxt, Nonce q, response S R q, Crypt K (Number m)}
    ∧
      Says S R
        {Agent S, Agent TTP, Crypt K (Number m), Number AO,
         Number ctxt, Nonce q,
         Crypt (pubEK TTP)
         {Agent S, Number AO, Key K, Agent R, hs}} ∈ set evs))"
⟨proof⟩

```

```

lemma S2TTP_sender:
  "[[Crypt (pubEK TTP) {Agent S, Number AO, Key K, Agent R, hs} ∈ used evs;
    Key K ∉ analz (spies evs);
    evs ∈ certified_mail]
  ⇒ ∃ m ctxt q.
    hs = Hash {Number ctxt, Nonce q, response S R q, Crypt K (Number m)}
  ∧
    Says S R
      {Agent S, Agent TTP, Crypt K (Number m), Number AO,
       Number ctxt, Nonce q,
       Crypt (pubEK TTP)
       {Agent S, Number AO, Key K, Agent R, hs}} ∈ set evs]"
⟨proof⟩

```

Nobody can have used non-existent keys!

```

lemma new_keys_not_used [simp]:
  "[[Key K ∉ used evs; K ∈ symKeys; evs ∈ certified_mail]
  ⇒ K ∉ keysFor (parts (spies evs))]"

```



*<proof>*

Less easy to prove  $m' = m$ . Maybe needs a separate unicity theorem for ciphertexts of the form  $\text{Crypt } K \text{ (Number } m)$ , where  $K$  is secure.

**lemma** *Key\_unique\_lemma* [rule\_format]:

```
"evs ∈ certified_mail ⇒
  Key K ∉ analz (spies evs) →
  (∀m cleartext q hs.
    Says S R
      {Agent S, Agent TTP, Crypt K (Number m), Number AO,
       Number cleartext, Nonce q,
       Crypt (pubEK TTP) {Agent S, Number AO, Key K, Agent R, hs}}
    ∈ set evs →
    (∀m' cleartext' q' hs'.
      Says S' R'
        {Agent S', Agent TTP, Crypt K (Number m'), Number AO',
         Number cleartext', Nonce q',
         Crypt (pubEK TTP) {Agent S', Number AO', Key K, Agent R', hs'}}
      ∈ set evs → R' = R ∧ S' = S ∧ AO' = AO ∧ hs' = hs))"
```

*<proof>*

The key determines the sender, recipient and protocol options.

**lemma** *Key\_unique*:

```
"[Says S R
  {Agent S, Agent TTP, Crypt K (Number m), Number AO,
   Number cleartext, Nonce q,
   Crypt (pubEK TTP) {Agent S, Number AO, Key K, Agent R, hs}}
 ∈ set evs;
 Says S' R'
  {Agent S', Agent TTP, Crypt K (Number m'), Number AO',
   Number cleartext', Nonce q',
   Crypt (pubEK TTP) {Agent S', Number AO', Key K, Agent R', hs'}}
 ∈ set evs;
 Key K ∉ analz (spies evs);
 evs ∈ certified_mail]
 ⇒ R' = R ∧ S' = S ∧ AO' = AO ∧ hs' = hs"
```

*<proof>*

## 22.2 The Guarantees for Sender and Recipient

A Sender's guarantee: If Spy gets the key then  $R$  is bad and  $S$  moreover gets his return receipt (and therefore has no grounds for complaint).

**theorem** *S\_fairness\_bad\_R*:

```
"[Says S R {Agent S, Agent TTP, Crypt K (Number m), Number AO,
            Number cleartext, Nonce q, S2TTP} ∈ set evs;
 S2TTP = Crypt (pubEK TTP) {Agent S, Number AO, Key K, Agent R, hs};
 Key K ∈ analz (spies evs);
 evs ∈ certified_mail;
 S ≠ Spy]
 ⇒ R ∈ bad ∧ Gets S (Crypt (priSK TTP) S2TTP) ∈ set evs"
```

*<proof>*

Confidentially for the symmetric key

**theorem** *Spy\_not\_see\_encrypted\_key*:

```
"[[Says S R {Agent S, Agent TTP, Crypt K (Number m), Number AO,
      Number cleartext, Nonce q, S2TTP} ∈ set evs;
   S2TTP = Crypt (pubEK TTP) {Agent S, Number AO, Key K, Agent R, hs};
   evs ∈ certified_mail;
   S ≠ Spy; R ∉ bad]]
⇒ Key K ∉ analz(spies evs)"
⟨proof⟩
```

Agent *R*, who may be the Spy, doesn't receive the key until *S* has access to the return receipt.

**theorem** *S\_guarantee*:

```
"[[Says S R {Agent S, Agent TTP, Crypt K (Number m), Number AO,
      Number cleartext, Nonce q, S2TTP} ∈ set evs;
   S2TTP = Crypt (pubEK TTP) {Agent S, Number AO, Key K, Agent R, hs};
   Notes R {Agent TTP, Agent R, Key K, hs} ∈ set evs;
   S ≠ Spy; evs ∈ certified_mail]]
⇒ Gets S (Crypt (priSK TTP) S2TTP) ∈ set evs"
⟨proof⟩
```

If *R* sends message 2, and a delivery certificate exists, then *R* receives the necessary key. This result is also important to *S*, as it confirms the validity of the return receipt.

**theorem** *RR\_validity*:

```
"[[Crypt (priSK TTP) S2TTP ∈ used evs;
   S2TTP = Crypt (pubEK TTP)
      {Agent S, Number AO, Key K, Agent R,
       Hash {Number cleartext, Nonce q, r, em}}];
   hr = Hash {Number cleartext, Nonce q, r, em};
   R ≠ Spy; evs ∈ certified_mail]]
⇒ Notes R {Agent TTP, Agent R, Key K, hr} ∈ set evs"
⟨proof⟩
```

**end**

## 23 Conventional protocols: rely on conventional Message, Event and Public – Public-key protocols

**theory** *Auth\_Public*

**imports**

*NS\_Public\_Bad*

*NS\_Public*

*TLS*

*CertifiedEmail*

**begin**

**end**

## 24 Theory of Events for Security Protocols that use smartcards

```

theory EventSC
imports
  "../Message"
  "HOL-Library.Simps_Case_Conv"
begin

consts
  initState :: "agent => msg set"

datatype card = Card agent

```

Four new events express the traffic between an agent and his card

```

datatype
  event = Says agent agent msg
        | Notes agent msg
        | Gets agent msg
        | Inputs agent card msg
        | C_Gets card msg
        | Outpts card agent msg
        | A_Gets agent msg

```

```

consts
  bad      :: "agent set"
  stolen   :: "card set"
  cloned   :: "card set"
  secureM  :: "bool"

```

```

abbreviation
  insecureM :: bool where
    "insecureM == ¬secureM"

```

Spy has access to his own key for spoof messages, but Server is secure

```

specification (bad)
  Spy_in_bad      [iff]: "Spy ∈ bad"
  Server_not_bad [iff]: "Server ∉ bad"
  ⟨proof⟩

```

```

specification (stolen)

```

```

  Card_Server_not_stolen [iff]: "Card Server ∉ stolen"
  Card_Spy_not_stolen   [iff]: "Card Spy ∉ stolen"
  ⟨proof⟩

```

```

specification (cloned)

```

```

  Card_Server_not_cloned [iff]: "Card Server ∉ cloned"
  Card_Spy_not_cloned   [iff]: "Card Spy ∉ cloned"
  ⟨proof⟩

```

```

primrec

```

```

knows  :: "agent => event list => msg set" where
knows_Nil:  "knows A [] = initState A" |
knows_Cons: "knows A (ev # evs) =
  (case ev of
    Says A' B X =>
      if (A=A' | A=Spy) then insert X (knows A evs) else knows A
evs
    | Notes A' X =>
      if (A=A' | (A=Spy & A'∈bad)) then insert X (knows A evs)
      else knows A evs
    | Gets A' X =>
      if (A=A' & A ≠ Spy) then insert X (knows A evs)
      else knows A evs
    | Inputs A' C X =>
      if secureM then
        if A=A' then insert X (knows A evs) else knows A evs
      else
        if (A=A' | A=Spy) then insert X (knows A evs) else knows A
evs
    | C_Gets C X => knows A evs
    | Outpts C A' X =>
      if secureM then
        if A=A' then insert X (knows A evs) else knows A evs
      else
        if A=Spy then insert X (knows A evs) else knows A evs
    | A_Gets A' X =>
      if (A=A' & A ≠ Spy) then insert X (knows A evs)
      else knows A evs)"

```

### primrec

```

used :: "event list => msg set" where
used_Nil:  "used [] = (UN B. parts (initState B))" |
used_Cons: "used (ev # evs) =
  (case ev of
    Says A B X => parts {X} ∪ (used evs)
    | Notes A X => parts {X} ∪ (used evs)
    | Gets A X => used evs
    | Inputs A C X => parts{X} ∪ (used evs)
    | C_Gets C X => used evs
    | Outpts C A X => parts{X} ∪ (used evs)
    | A_Gets A X => used evs)"

```

— Gets always follows Says in real protocols. Likewise, C\_Gets will always have to follow Inputs and A\_Gets will always have to follow Outpts

**lemma** Notes\_imp\_used [rule\_format]: "Notes A X ∈ set evs  $\longrightarrow$  X ∈ used evs"  
 <proof>

**lemma** Says\_imp\_used [rule\_format]: "Says A B X ∈ set evs  $\longrightarrow$  X ∈ used evs"  
 <proof>

**lemma** MPair\_used [rule\_format]:

"MPair  $X Y \in \text{used evs} \longrightarrow X \in \text{used evs} \ \& \ Y \in \text{used evs}$ "  
 <proof>

### 24.1 Function knows

lemmas parts\_insert\_knows\_A = parts\_insert [of \_ "knows A evs"] for A evs

lemma knows\_Spy\_Says [simp]:  
 "knows Spy (Says A B X # evs) = insert X (knows Spy evs)"  
 <proof>

Letting the Spy see "bad" agents' notes avoids redundant case-splits on whether  
 $A = \text{Spy}$  and whether  $A \in \text{bad}$

lemma knows\_Spy\_Notes [simp]:  
 "knows Spy (Notes A X # evs) =  
 (if  $A \in \text{bad}$  then insert X (knows Spy evs) else knows Spy evs)"  
 <proof>

lemma knows\_Spy\_Gets [simp]: "knows Spy (Gets A X # evs) = knows Spy evs"  
 <proof>

lemma knows\_Spy\_Inputs\_secureM [simp]:  
 "secureM  $\implies$  knows Spy (Inputs A C X # evs) =  
 (if  $A = \text{Spy}$  then insert X (knows Spy evs) else knows Spy evs)"  
 <proof>

lemma knows\_Spy\_Inputs\_insecureM [simp]:  
 "insecureM  $\implies$  knows Spy (Inputs A C X # evs) = insert X (knows Spy evs)"  
 <proof>

lemma knows\_Spy\_C\_Gets [simp]: "knows Spy (C\_Gets C X # evs) = knows Spy  
 evs"  
 <proof>

lemma knows\_Spy\_Outpts\_secureM [simp]:  
 "secureM  $\implies$  knows Spy (Outpts C A X # evs) =  
 (if  $A = \text{Spy}$  then insert X (knows Spy evs) else knows Spy evs)"  
 <proof>

lemma knows\_Spy\_Outpts\_insecureM [simp]:  
 "insecureM  $\implies$  knows Spy (Outpts C A X # evs) = insert X (knows Spy  
 evs)"  
 <proof>

lemma knows\_Spy\_A\_Gets [simp]: "knows Spy (A\_Gets A X # evs) = knows Spy  
 evs"  
 <proof>

lemma knows\_Spy\_subset\_knows\_Spy\_Says:  
 "knows Spy evs  $\subseteq$  knows Spy (Says A B X # evs)"  
 <proof>

**lemma** *knows\_Spy\_subset\_knows\_Spy\_Notes*:  
 "knows Spy evs  $\subseteq$  knows Spy (Notes A X # evs)"  
 <proof>

**lemma** *knows\_Spy\_subset\_knows\_Spy\_Gets*:  
 "knows Spy evs  $\subseteq$  knows Spy (Gets A X # evs)"  
 <proof>

**lemma** *knows\_Spy\_subset\_knows\_Spy\_Inputs*:  
 "knows Spy evs  $\subseteq$  knows Spy (Inputs A C X # evs)"  
 <proof>

**lemma** *knows\_Spy\_equals\_knows\_Spy\_Gets*:  
 "knows Spy evs = knows Spy (C\_Gets C X # evs)"  
 <proof>

**lemma** *knows\_Spy\_subset\_knows\_Spy\_Outpts*: "knows Spy evs  $\subseteq$  knows Spy (Outpts C A X # evs)"  
 <proof>

**lemma** *knows\_Spy\_subset\_knows\_Spy\_A\_Gets*: "knows Spy evs  $\subseteq$  knows Spy (A\_Gets A X # evs)"  
 <proof>

Spy sees what is sent on the traffic

**lemma** *Says\_imp\_knows\_Spy [rule\_format]*:  
 "Says A B X  $\in$  set evs  $\longrightarrow$  X  $\in$  knows Spy evs"  
 <proof>

**lemma** *Notes\_imp\_knows\_Spy [rule\_format]*:  
 "Notes A X  $\in$  set evs  $\longrightarrow$  A  $\in$  bad  $\longrightarrow$  X  $\in$  knows Spy evs"  
 <proof>

**lemma** *Inputs\_imp\_knows\_Spy\_secureM [rule\_format (no\_asm)]*:  
 "Inputs Spy C X  $\in$  set evs  $\longrightarrow$  secureM  $\longrightarrow$  X  $\in$  knows Spy evs"  
 <proof>

**lemma** *Inputs\_imp\_knows\_Spy\_insecureM [rule\_format (no\_asm)]*:  
 "Inputs A C X  $\in$  set evs  $\longrightarrow$  insecureM  $\longrightarrow$  X  $\in$  knows Spy evs"  
 <proof>

**lemma** *Outpts\_imp\_knows\_Spy\_secureM [rule\_format (no\_asm)]*:  
 "Outpts C Spy X  $\in$  set evs  $\longrightarrow$  secureM  $\longrightarrow$  X  $\in$  knows Spy evs"  
 <proof>

**lemma** *Outpts\_imp\_knows\_Spy\_insecureM [rule\_format (no\_asm)]*:  
 "Outpts C A X  $\in$  set evs  $\longrightarrow$  insecureM  $\longrightarrow$  X  $\in$  knows Spy evs"  
 <proof>

Elimination rules: derive contradictions from old Says events containing items known to be fresh

```
lemmas knows_Spy_partsEs =
  Says_imp_knows_Spy [THEN parts.Inj, elim_format]
  parts.Body [elim_format]
```

## 24.2 Knowledge of Agents

```
lemma knows_Inputs: "knows A (Inputs A C X # evs) = insert X (knows A evs)"
  <proof>
```

```
lemma knows_C_Gets: "knows A (C_Gets C X # evs) = knows A evs"
  <proof>
```

```
lemma knows_Outpts_secureM:
  "secureM  $\longrightarrow$  knows A (Outpts C A X # evs) = insert X (knows A evs)"
  <proof>
```

```
lemma knows_Outpts_insecureM:
  "insecureM  $\longrightarrow$  knows Spy (Outpts C A X # evs) = insert X (knows Spy
  evs)"
  <proof>
```

```
lemma knows_subset_knows_Says: "knows A evs  $\subseteq$  knows A (Says A' B X # evs)"
  <proof>
```

```
lemma knows_subset_knows_Notes: "knows A evs  $\subseteq$  knows A (Notes A' X # evs)"
  <proof>
```

```
lemma knows_subset_knows_Gets: "knows A evs  $\subseteq$  knows A (Gets A' X # evs)"
  <proof>
```

```
lemma knows_subset_knows_Inputs: "knows A evs  $\subseteq$  knows A (Inputs A' C X #
  evs)"
  <proof>
```

```
lemma knows_subset_knows_C_Gets: "knows A evs  $\subseteq$  knows A (C_Gets C X # evs)"
  <proof>
```

```
lemma knows_subset_knows_Outpts: "knows A evs  $\subseteq$  knows A (Outpts C A' X #
  evs)"
  <proof>
```

```
lemma knows_subset_knows_A_Gets: "knows A evs  $\subseteq$  knows A (A_Gets A' X # evs)"
  <proof>
```

Agents know what they say

```
lemma Says_imp_knows [rule_format]: "Says A B X  $\in$  set evs  $\longrightarrow$  X  $\in$  knows
  A evs"
```

*<proof>*

Agents know what they note

**lemma** *Notes\_imp\_knows [rule\_format]: "Notes A X ∈ set evs → X ∈ knows A evs"*  
*<proof>*

Agents know what they receive

**lemma** *Gets\_imp\_knows\_agents [rule\_format]:*  
*"A ≠ Spy → Gets A X ∈ set evs → X ∈ knows A evs"*  
*<proof>*

**lemma** *Inputs\_imp\_knows\_agents [rule\_format (no\_asm)]:*  
*"Inputs A (Card A) X ∈ set evs → X ∈ knows A evs"*  
*<proof>*

**lemma** *Outpts\_imp\_knows\_agents\_secureM [rule\_format (no\_asm)]:*  
*"secureM → Outpts (Card A) A X ∈ set evs → X ∈ knows A evs"*  
*<proof>*

**lemma** *Outpts\_imp\_knows\_agents\_insecureM [rule\_format (no\_asm)]:*  
*"insecureM → Outpts (Card A) A X ∈ set evs → X ∈ knows Spy evs"*  
*<proof>*

**lemma** *parts\_knows\_Spy\_subset\_used: "parts (knows Spy evs) ⊆ used evs"*  
*<proof>*

**lemmas** *usedI = parts\_knows\_Spy\_subset\_used [THEN subsetD, intro]*

**lemma** *initState\_into\_used: "X ∈ parts (initState B) ⇒ X ∈ used evs"*  
*<proof>*

**simps\_of\_case** *used\_Cons\_simps[simp]: used\_Cons*

**lemma** *used\_nil\_subset: "used [] ⊆ used evs"*  
*<proof>*

**lemma** *Says\_parts\_used [rule\_format (no\_asm)]:*  
*"Says A B X ∈ set evs → (parts {X}) ⊆ used evs"*  
*<proof>*

**lemma** *Notes\_parts\_used [rule\_format (no\_asm)]:*



```

    "Notes A X ∈ set evs → (parts {X}) ⊆ used evs"
  <proof>

lemma Outpts_parts_used [rule_format (no_asm)]:
  "Outpts C A X ∈ set evs → (parts {X}) ⊆ used evs"
  <proof>

lemma Inputs_parts_used [rule_format (no_asm)]:
  "Inputs A C X ∈ set evs → (parts {X}) ⊆ used evs"
  <proof>

NOTE REMOVAL—laws above are cleaner, as they don't involve "case"

declare knows_Cons [simp del]
        used_Nil [simp del] used_Cons [simp del]

lemma knows_subset_knows_Cons: "knows A evs ⊆ knows A (e # evs)"
  <proof>

lemma initState_subset_knows: "initState A ⊆ knows A evs"
  <proof>

For proving new_keys_not_used

lemma keysFor_parts_insert:
  "[[ K ∈ keysFor (parts (insert X G)); X ∈ synth (analz H) ] ]
   ⇒ K ∈ keysFor (parts (G ∪ H)) ∨ Key (invKey K) ∈ parts H"
  <proof>

end
theory All_Symmetric
imports Message
begin

All keys are symmetric

overloading all_symmetric ≡ all_symmetric
begin
  definition "all_symmetric ≡ True"
end

lemma isSym_keys: "K ∈ symKeys"
  <proof>

end

```

## 25 Theory of smartcards

```

theory Smartcard
imports EventSC "../All_Symmetric"
begin

```

As smartcards handle long-term (symmetric) keys, this theory extends and supersedes theory Private.thy

An agent is bad if she reveals her PIN to the spy, not the shared key that is embedded in her card. An agent's being bad implies nothing about her smartcard, which independently may be stolen or cloned.

#### axiomatization

```
shrK    :: "agent => key" and
crdK    :: "card  => key" and
pin     :: "agent => key" and
```

```
Pairkey :: "agent * agent => nat" and
pairK   :: "agent * agent => key"
```

#### where

```
inj_shrK: "inj shrK" and — No two smartcards store the same key
inj_crdK: "inj crdK" and — Nor do two cards
inj_pin  : "inj pin" and  — Nor do two agents have the same pin
```

```
inj_pairK [iff]: "(pairK(A,B) = pairK(A',B')) = (A = A' & B = B' )" and
comm_Pairkey [iff]: "Pairkey(A,B) = Pairkey(B,A)" and
```

```
pairK_disj_crdK [iff]: "pairK(A,B) ≠ crdK C" and
pairK_disj_shrK [iff]: "pairK(A,B) ≠ shrK P" and
pairK_disj_pin [iff]:  "pairK(A,B) ≠ pin P" and
shrK_disj_crdK [iff]:  "shrK P ≠ crdK C" and
shrK_disj_pin [iff]:   "shrK P ≠ pin Q" and
crdK_disj_pin [iff]:   "crdK C ≠ pin P"
```

```
definition legalUse :: "card => bool" (<legalUse _>) where
  "legalUse C == C ∉ stolen"
```

```
primrec illegalUse :: "card => bool" where
  illegalUse_def: "illegalUse (Card A) = ( (Card A ∈ stolen ∧ A ∈ bad) ∨
Card A ∈ cloned )"

```

initState must be defined with care

#### overloading

```
initState ≡ initState
```

#### begin

```
primrec initState where
```

```
initState_Server: "initState Server =
  (Key'(range shrK ∪ range crdK ∪ range pin ∪ range pairK)) ∪
  (Nonce'(range Pairkey))" |
```

```
initState_Friend: "initState (Friend i) = {Key (pin (Friend i))}" |
```

```
initState_Spy: "initState Spy =
  (Key'((pin'bad) ∪ (pin '{A. Card A ∈ cloned}) ∪
    (shrK'{A. Card A ∈ cloned}) ∪
```

$$\begin{aligned} & (\text{crdK}'\text{cloned}) \cup \\ & (\text{pairK}'\{(X,A). \text{Card } A \in \text{cloned}\})) \\ \cup & (\text{Nonce}'(\text{Pairkey}'\{(A,B). \text{Card } A \in \text{cloned} \ \& \ \text{Card } B \in \text{cloned}\})) \end{aligned}$$

**end**

Still relying on axioms

**axiomatization where**

**Key\_supply\_ax:** "finite KK  $\implies \exists K. K \notin KK \ \& \ \text{Key } K \notin \text{used evs}$ " **and**

**Nonce\_supply\_ax:** "finite NN  $\implies \exists N. N \notin NN \ \& \ \text{Nonce } N \notin \text{used evs}$ "

## 25.1 Basic properties of shrK

**declare** inj\_shrK [THEN inj\_eq, iff]  
**declare** inj\_crdK [THEN inj\_eq, iff]  
**declare** inj\_pin [THEN inj\_eq, iff]

**lemma** invKey\_K [simp]: "invKey K = K"  
 <proof>

**lemma** analz\_Decrypt' [dest]:  
 "[ Crypt K X  $\in$  analz H; Key K  $\in$  analz H ]  $\implies X \in$  analz H"  
 <proof>

Now cancel the *dest* attribute given to *analz.Decrypt* in its declaration.

**declare** analz.Decrypt [rule del]

Rewrites should not refer to *initState (Friend i)* because that expression is not in normal form.

Added to extend initState with set of nonces

**lemma** parts\_image\_Nonce [simp]: "parts (Nonce'N) = Nonce'N"  
 <proof>

**lemma** keysFor\_parts\_initState [simp]: "keysFor (parts (initState C)) = {}"  
 <proof>

**lemma** keysFor\_parts\_insert:  
 "[ K  $\in$  keysFor (parts (insert X G)); X  $\in$  synth (analz H) ]  
 $\implies K \in$  keysFor (parts (G  $\cup$  H)) | Key K  $\in$  parts H"  
 <proof>

**lemma** Crypt\_imp\_keysFor: "Crypt K X  $\in$  H  $\implies K \in$  keysFor H"  
 <proof>

## 25.2 Function "knows"

**lemma** Spy\_knows\_bad [intro!]: "A  $\in$  bad  $\implies \text{Key (pin A)} \in \text{knows Spy evs}$ "  
 <proof>

**lemma** *Spy\_knows\_cloned* [intro!]:

"Card A  $\in$  cloned  $\implies$  Key (crdK (Card A))  $\in$  knows Spy evs &  
 Key (shrK A)  $\in$  knows Spy evs &  
 Key (pin A)  $\in$  knows Spy evs &  
 ( $\forall$  B. Key (pairK(B,A))  $\in$  knows Spy evs)"

*<proof>*

**lemma** *Spy\_knows\_cloned1* [intro!]: "C  $\in$  cloned  $\implies$  Key (crdK C)  $\in$  knows Spy evs"

*<proof>*

**lemma** *Spy\_knows\_cloned2* [intro!]: "[ Card A  $\in$  cloned; Card B  $\in$  cloned ]

$\implies$  Nonce (Pairkey(A,B))  $\in$  knows Spy evs"

*<proof>*

**lemma** *Spy\_knows\_Spy\_bad* [intro!]: "A  $\in$  bad  $\implies$  Key (pin A)  $\in$  knows Spy evs"

*<proof>*

**lemma** *Crypt\_Spy\_analz\_bad*:

"[ Crypt (pin A) X  $\in$  analz (knows Spy evs); A  $\in$  bad ]  
 $\implies$  X  $\in$  analz (knows Spy evs)"

*<proof>*

**lemma** *shrK\_in\_initState* [iff]: "Key (shrK A)  $\in$  initState Server"

*<proof>*

**lemma** *shrK\_in\_used* [iff]: "Key (shrK A)  $\in$  used evs"

*<proof>*

**lemma** *crdK\_in\_initState* [iff]: "Key (crdK A)  $\in$  initState Server"

*<proof>*

**lemma** *crdK\_in\_used* [iff]: "Key (crdK A)  $\in$  used evs"

*<proof>*

**lemma** *pin\_in\_initState* [iff]: "Key (pin A)  $\in$  initState A"

*<proof>*

**lemma** *pin\_in\_used* [iff]: "Key (pin A)  $\in$  used evs"

*<proof>*

**lemma** *pairK\_in\_initState* [iff]: "Key (pairK X)  $\in$  initState Server"

*<proof>*

**lemma** *pairK\_in\_used* [iff]: "Key (pairK X)  $\in$  used evs"

*<proof>*

**lemma** *Key\_not\_used [simp]: "Key K  $\notin$  used evs  $\implies$  K  $\notin$  range shrK"*  
*<proof>*

**lemma** *shrK\_neq [simp]: "Key K  $\notin$  used evs  $\implies$  shrK B  $\neq$  K"*  
*<proof>*

**lemma** *crdK\_not\_used [simp]: "Key K  $\notin$  used evs  $\implies$  K  $\notin$  range crdK"*  
*<proof>*

**lemma** *crdK\_neq [simp]: "Key K  $\notin$  used evs  $\implies$  crdK C  $\neq$  K"*  
*<proof>*

**lemma** *pin\_not\_used [simp]: "Key K  $\notin$  used evs  $\implies$  K  $\notin$  range pin"*  
*<proof>*

**lemma** *pin\_neq [simp]: "Key K  $\notin$  used evs  $\implies$  pin A  $\neq$  K"*  
*<proof>*

**lemma** *pairK\_not\_used [simp]: "Key K  $\notin$  used evs  $\implies$  K  $\notin$  range pairK"*  
*<proof>*

**lemma** *pairK\_neq [simp]: "Key K  $\notin$  used evs  $\implies$  pairK(A,B)  $\neq$  K"*  
*<proof>*

**declare** *shrK\_neq [THEN not\_sym, simp]*  
**declare** *crdK\_neq [THEN not\_sym, simp]*  
**declare** *pin\_neq [THEN not\_sym, simp]*  
**declare** *pairK\_neq [THEN not\_sym, simp]*

### 25.3 Fresh nonces

**lemma** *Nonce\_notin\_initState [iff]: "Nonce N  $\notin$  parts (initState (Friend i))"*  
*<proof>*

### 25.4 Supply fresh nonces for possibility theorems.

**lemma** *Nonce\_supply1: " $\exists N$ . Nonce N  $\notin$  used evs"*  
*<proof>*

**lemma** *Nonce\_supply2:*  
*" $\exists N N'$ . Nonce N  $\notin$  used evs & Nonce N'  $\notin$  used evs' & N  $\neq$  N'"*  
*<proof>*

**lemma** *Nonce\_supply3: " $\exists N N' N''$ . Nonce N  $\notin$  used evs & Nonce N'  $\notin$  used evs' &*  
*Nonce N''  $\notin$  used evs'' & N  $\neq$  N' & N'  $\neq$  N'' & N  $\neq$  N''"*  
*<proof>*

**lemma** *Nonce\_supply: "Nonce (SOME N. Nonce N  $\notin$  used evs)  $\notin$  used evs"*

*<proof>*

Unlike the corresponding property of nonces, we cannot prove  $\text{finite } KK \implies \exists K. K \notin KK \wedge \text{Key } K \notin \text{used evs}$ . We have infinitely many agents and there is nothing to stop their long-term keys from exhausting all the natural numbers. Instead, possibility theorems must assume the existence of a few keys.

## 25.5 Specialized Rewriting for Theorems About *analz* and Image

**lemma** *subset\_Compl\_range\_shrK*: " $A \subseteq - (\text{range shrK}) \implies \text{shrK } x \notin A$ "  
*<proof>*

**lemma** *subset\_Compl\_range\_crdK*: " $A \subseteq - (\text{range crdK}) \implies \text{crdK } x \notin A$ "  
*<proof>*

**lemma** *subset\_Compl\_range\_pin*: " $A \subseteq - (\text{range pin}) \implies \text{pin } x \notin A$ "  
*<proof>*

**lemma** *subset\_Compl\_range\_pairK*: " $A \subseteq - (\text{range pairK}) \implies \text{pairK } x \notin A$ "  
*<proof>*

**lemma** *insert\_Key\_singleton*: " $\text{insert } (\text{Key } K) H = \text{Key } ' \{K\} \cup H$ "  
*<proof>*

**lemma** *insert\_Key\_image*: " $\text{insert } (\text{Key } K) (\text{Key}'KK \cup C) = \text{Key}'(\text{insert } K KK) \cup C$ "  
*<proof>*

**lemmas** *analz\_image\_freshK\_simps* =  
   *simp\_thms mem\_simps* — these two allow its use with only:  
   *disj\_comms*  
   *image\_insert [THEN sym] image\_Un [THEN sym] empty\_subsetI insert\_subset*  
   *analz\_insert\_eq Un\_upper2 [THEN analz\_mono, THEN [2] rev\_subsetD]*  
   *insert\_Key\_singleton subset\_Compl\_range\_shrK subset\_Compl\_range\_crdK*  
   *subset\_Compl\_range\_pin subset\_Compl\_range\_pairK*  
   *Key\_not\_used insert\_Key\_image Un\_assoc [THEN sym]*

**lemma** *analz\_image\_freshK\_lemma*:  
   " $(\text{Key } K \in \text{analz } (\text{Key}'nE \cup H)) \longrightarrow (K \in nE \mid \text{Key } K \in \text{analz } H) \implies$   
    $(\text{Key } K \in \text{analz } (\text{Key}'nE \cup H)) = (K \in nE \mid \text{Key } K \in \text{analz } H)$ "  
*<proof>*

## 25.6 Tactics for possibility theorems

*<ML>*

**lemma** *invKey\_shrK\_iff [iff]*:  
   " $(\text{Key } (\text{invKey } K) \in X) = (\text{Key } K \in X)$ "

$\langle proof \rangle$

$\langle ML \rangle$

**lemma** *knows\_subset\_knows\_Cons*: " $knows\ A\ evs \subseteq knows\ A\ (e\ \# \ evs)$ "  
 $\langle proof \rangle$

```

declare shrK_disj_crdK[THEN not_sym, iff]
declare shrK_disj_pin[THEN not_sym, iff]
declare pairK_disj_shrK[THEN not_sym, iff]
declare pairK_disj_crdK[THEN not_sym, iff]
declare pairK_disj_pin[THEN not_sym, iff]
declare crdK_disj_pin[THEN not_sym, iff]

declare legalUse_def [iff] illegalUse_def [iff]

end

```

## 26 Original Shoup-Rubin protocol

**theory** *ShoupRubin* **imports** *Smartcard* **begin**

**axiomatization** *sesK* :: " $nat*key \Rightarrow key$ "  
**where**

*inj\_sesK* [iff]: " $(sesK(m,k) = sesK(m',k')) = (m = m' \wedge k = k')$ " **and**

*shrK\_disj\_sesK* [iff]: " $shrK\ A \neq sesK(m,pk)$ " **and**  
*crdK\_disj\_sesK* [iff]: " $crdK\ C \neq sesK(m,pk)$ " **and**  
*pin\_disj\_sesK* [iff]: " $pin\ P \neq sesK(m,pk)$ " **and**  
*pairK\_disj\_sesK* [iff]: " $pairK(A,B) \neq sesK(m,pk)$ " **and**

*Atomic\_distrib* [iff]: " $Atomic'(KEY'K \cup NONCE'N) =$   
 $Atomic'(KEY'K) \cup Atomic'(NONCE'N)$ " **and**

*shouprubin\_assumes\_securemeans* [iff]: " $evs \in sr \Longrightarrow secureM$ "

**definition** *Unique* :: " $[event, event\ list] \Rightarrow bool$ " ( $\langle Unique\_on\_ \rangle$ ) **where**  
 $Unique\ ev\ on\ evs ==$   
 $ev \notin set\ (tl\ (dropWhile\ (\% z. z \neq ev)\ evs))$

**inductive\_set** *sr* :: " $event\ list\ set$ "  
**where**

*Nil*: " $[] \in sr$ "

```

/ Fake: "[ evsF ∈ sr; X ∈ synth (analz (knows Spy evsF));
         illegalUse(Card B) ]
    ⇒ Says Spy A X #
       Inputs Spy (Card B) X # evsF ∈ sr"

/ Forge:
    "[ evsFo ∈ sr; Nonce Nb ∈ analz (knows Spy evsFo);
       Key (pairK(A,B)) ∈ knows Spy evsFo ]
    ⇒ Notes Spy (Key (sesK(Nb,pairK(A,B)))) # evsFo ∈ sr"

/ Reception: "[ evsR ∈ sr; Says A B X ∈ set evsR ]
    ⇒ Gets B X # evsR ∈ sr"

/ SR1: "[ evs1 ∈ sr; A ≠ Server]
    ⇒ Says A Server {Agent A, Agent B}
       # evs1 ∈ sr"

/ SR2: "[ evs2 ∈ sr;
         Gets Server {Agent A, Agent B} ∈ set evs2 ]
    ⇒ Says Server A {Nonce (Pairkey(A,B)),
                     Crypt (shrK A) {Nonce (Pairkey(A,B)), Agent B}
                     }
       # evs2 ∈ sr"

/ SR3: "[ evs3 ∈ sr; legalUse(Card A);
         Says A Server {Agent A, Agent B} ∈ set evs3;
         Gets A {Nonce Pk, Certificate} ∈ set evs3 ]
    ⇒ Inputs A (Card A) (Agent A)
       # evs3 ∈ sr"

/ SR4: "[ evs4 ∈ sr; A ≠ Server;
         Nonce Na ∉ used evs4; legalUse(Card A);
         Inputs A (Card A) (Agent A) ∈ set evs4 ]
    ⇒ Outputs (Card A) A {Nonce Na, Crypt (crdK (Card A)) (Nonce Na)}
       # evs4 ∈ sr"

/ SR4Fake: "[ evs4F ∈ sr; Nonce Na ∉ used evs4F;
             illegalUse(Card A);
             Inputs Spy (Card A) (Agent A) ∈ set evs4F ]

```



$\Rightarrow$  *Outpts* (Card A) *Spy*  $\{\text{Nonce Na}, \text{Crypt}(\text{crdK}(\text{Card A}))(\text{Nonce Na})\}$   
 $\# \text{ evs4F} \in \text{sr}$ "

/ SR5: "[ evs5  $\in$  sr;  
*Outpts* (Card A) A  $\{\text{Nonce Na}, \text{Certificate}\} \in \text{set evs5};$   
 $\forall p q. \text{Certificate} \neq \{p, q\}$  ]  
 $\Rightarrow$  *Says* A B  $\{\text{Agent A}, \text{Nonce Na}\} \# \text{ evs5} \in \text{sr}$ "

/ SR6: "[ evs6  $\in$  sr; *legalUse*(Card B);  
*Gets* B  $\{\text{Agent A}, \text{Nonce Na}\} \in \text{set evs6}$  ]  
 $\Rightarrow$  *Inputs* B (Card B)  $\{\text{Agent A}, \text{Nonce Na}\}$   
 $\# \text{ evs6} \in \text{sr}$ "

/ SR7: "[ evs7  $\in$  sr;  
 $\text{Nonce Nb} \notin \text{used evs7};$  *legalUse*(Card B); B  $\neq$  Server;  
 $K = \text{sesK}(\text{Nb}, \text{pairK}(\text{A}, \text{B}));$   
 $\text{Key K} \notin \text{used evs7};$   
*Inputs* B (Card B)  $\{\text{Agent A}, \text{Nonce Na}\} \in \text{set evs7}$  ]  
 $\Rightarrow$  *Outpts* (Card B) B  $\{\text{Nonce Nb}, \text{Key K},$   
 $\text{Crypt}(\text{pairK}(\text{A}, \text{B}))(\text{Nonce Na}, \text{Nonce Nb}),$   
 $\text{Crypt}(\text{pairK}(\text{A}, \text{B}))(\text{Nonce Nb})\}$   
 $\# \text{ evs7} \in \text{sr}$ "

/ SR7Fake: "[ evs7F  $\in$  sr;  $\text{Nonce Nb} \notin \text{used evs7F};$   
*illegalUse*(Card B);  
 $K = \text{sesK}(\text{Nb}, \text{pairK}(\text{A}, \text{B}));$   
 $\text{Key K} \notin \text{used evs7F};$   
*Inputs* *Spy* (Card B)  $\{\text{Agent A}, \text{Nonce Na}\} \in \text{set evs7F}$  ]  
 $\Rightarrow$  *Outpts* (Card B) *Spy*  $\{\text{Nonce Nb}, \text{Key K},$   
 $\text{Crypt}(\text{pairK}(\text{A}, \text{B}))(\text{Nonce Na}, \text{Nonce Nb}),$   
 $\text{Crypt}(\text{pairK}(\text{A}, \text{B}))(\text{Nonce Nb})\}$   
 $\# \text{ evs7F} \in \text{sr}$ "

/ SR8: "[ evs8  $\in$  sr;  
*Inputs* B (Card B)  $\{\text{Agent A}, \text{Nonce Na}\} \in \text{set evs8};$   
*Outpts* (Card B) B  $\{\text{Nonce Nb}, \text{Key K},$   
 $\text{Cert1}, \text{Cert2}\} \in \text{set evs8}$  ]

$\Rightarrow \text{Says } B \ A \ \{\{ \text{Nonce } Nb, \text{Cert1} \} \# \text{ evs8} \in sr\}$

/ SR9: " $\ll \text{ evs9} \in sr; \text{ legalUse}(\text{Card } A);$   
 $\text{Gets } A \ \{\{ \text{Nonce } Pk, \text{Cert1} \} \in \text{set evs9};$   
 $\text{Outpts } (\text{Card } A) \ A \ \{\{ \text{Nonce } Na, \text{Cert2} \} \in \text{set evs9};$   
 $\text{Gets } A \ \{\{ \text{Nonce } Nb, \text{Cert3} \} \in \text{set evs9};$   
 $\forall p \ q. \text{Cert2} \neq \{p, q\} \}$   
 $\Rightarrow \text{Inputs } A \ (\text{Card } A)$   
 $\{\{ \text{Agent } B, \text{Nonce } Na, \text{Nonce } Nb, \text{Nonce } Pk,$   
 $\text{Cert1}, \text{Cert3}, \text{Cert2} \}$   
 $\# \text{ evs9} \in sr\}$

/ SR10: " $\ll \text{ evs10} \in sr; \text{ legalUse}(\text{Card } A); A \neq \text{Server};$   
 $K = \text{sesK}(Nb, \text{pairK}(A, B));$   
 $\text{Inputs } A \ (\text{Card } A) \ \{\{ \text{Agent } B, \text{Nonce } Na, \text{Nonce } Nb,$   
 $\text{Nonce } (\text{Pairkey}(A, B)),$   
 $\text{Crypt } (\text{shrK } A) \ \{\{ \text{Nonce } (\text{Pairkey}(A, B)),$   
 $\text{Agent } B \},$   
 $\text{Crypt } (\text{pairK}(A, B)) \ \{\{ \text{Nonce } Na, \text{Nonce } Nb \},$   
 $\text{Crypt } (\text{crdK } (\text{Card } A)) (\text{Nonce } Na) \}$   
 $\in \text{set evs10} \}$   
 $\Rightarrow \text{Outpts } (\text{Card } A) \ A \ \{\{ \text{Key } K, \text{Crypt } (\text{pairK}(A, B)) (\text{Nonce } Nb) \}$   
 $\# \text{ evs10} \in sr\}$

/ SR10Fake: " $\ll \text{ evs10F} \in sr;$   
 $\text{illegalUse}(\text{Card } A);$   
 $K = \text{sesK}(Nb, \text{pairK}(A, B));$   
 $\text{Inputs } \text{Spy } (\text{Card } A) \ \{\{ \text{Agent } B, \text{Nonce } Na, \text{Nonce } Nb,$   
 $\text{Nonce } (\text{Pairkey}(A, B)),$   
 $\text{Crypt } (\text{shrK } A) \ \{\{ \text{Nonce } (\text{Pairkey}(A, B)),$   
 $\text{Agent } B \},$   
 $\text{Crypt } (\text{pairK}(A, B)) \ \{\{ \text{Nonce } Na, \text{Nonce } Nb \},$   
 $\text{Crypt } (\text{crdK } (\text{Card } A)) (\text{Nonce } Na) \}$   
 $\in \text{set evs10F} \}$   
 $\Rightarrow \text{Outpts } (\text{Card } A) \ \text{Spy} \ \{\{ \text{Key } K, \text{Crypt } (\text{pairK}(A, B)) (\text{Nonce } Nb) \}$   
 $\# \text{ evs10F} \in sr\}$

/ SR11: " $\ll \text{ evs11} \in sr;$

```

    Says A Server {Agent A, Agent B} ∈ set evs11;
    Outpts (Card A) A {Key K, Certificate} ∈ set evs11
  ⇒ Says A B (Certificate)
    # evs11 ∈ sr"

```

```

/ Ops1:
  "[ evs01 ∈ sr;
    Outpts (Card B) B {Nonce Nb, Key K, Certificate,
                      Crypt (pairK(A,B)) (Nonce Nb)} ∈ set evs01 ]
  ⇒ Notes Spy {Key K, Nonce Nb, Agent A, Agent B} # evs01 ∈ sr"

/ Ops2:
  "[ evs02 ∈ sr;
    Outpts (Card A) A {Key K, Crypt (pairK(A,B)) (Nonce Nb)}
      ∈ set evs02 ]
  ⇒ Notes Spy {Key K, Nonce Nb, Agent A, Agent B} # evs02 ∈ sr"

```

```

declare Fake_parts_insert_in_Un [dest]
declare analz_into_parts [dest]

```

```

lemma Gets_imp_Says:
  "[ Gets B X ∈ set evs; evs ∈ sr ] ⇒ ∃ A. Says A B X ∈ set evs"
<proof>

```

```

lemma Gets_imp_knows_Spy:
  "[ Gets B X ∈ set evs; evs ∈ sr ] ⇒ X ∈ knows Spy evs"
<proof>

```

```

lemma Gets_imp_knows_Spy_parts_Snd:
  "[ Gets B {X, Y} ∈ set evs; evs ∈ sr ] ⇒ Y ∈ parts (knows Spy evs)"
<proof>

```

```

lemma Gets_imp_knows_Spy_analz_Snd:
  "[ Gets B {X, Y} ∈ set evs; evs ∈ sr ] ⇒ Y ∈ analz (knows Spy evs)"
<proof>

```

```

lemma Inputs_imp_knows_Spy_secureM_sr:
  "[ Inputs Spy C X ∈ set evs; evs ∈ sr ] ⇒ X ∈ knows Spy evs"
  <proof>

lemma knows_Spy_Inputs_secureM_sr_Spy:
  "evs ∈ sr ⇒ knows Spy (Inputs Spy C X # evs) = insert X (knows Spy
  evs)"
  <proof>

lemma knows_Spy_Inputs_secureM_sr:
  "[ A ≠ Spy; evs ∈ sr ] ⇒ knows Spy (Inputs A C X # evs) = knows Spy
  evs"
  <proof>

lemma knows_Spy_Outpts_secureM_sr_Spy:
  "evs ∈ sr ⇒ knows Spy (Outpts C Spy X # evs) = insert X (knows Spy
  evs)"
  <proof>

lemma knows_Spy_Outpts_secureM_sr:
  "[ A ≠ Spy; evs ∈ sr ] ⇒ knows Spy (Outpts C A X # evs) = knows Spy
  evs"
  <proof>


lemma Inputs_A_Card_3:
  "[ Inputs A C (Agent A) ∈ set evs; A ≠ Spy; evs ∈ sr ]
  ⇒ legalUse(C) ∧ C = (Card A) ∧
  (∃ Pk Certificate. Gets A {Pk, Certificate} ∈ set evs)"
  <proof>

lemma Inputs_B_Card_6:
  "[ Inputs B C {Agent A, Nonce Na} ∈ set evs; B ≠ Spy; evs ∈ sr ]
  ⇒ legalUse(C) ∧ C = (Card B) ∧ Gets B {Agent A, Nonce Na} ∈ set evs"
  <proof>

lemma Inputs_A_Card_9:
  "[ Inputs A C {Agent B, Nonce Na, Nonce Nb, Nonce Pk,
  Cert1, Cert2, Cert3} ∈ set evs;

  A ≠ Spy; evs ∈ sr ]
  ⇒ legalUse(C) ∧ C = (Card A) ∧
  Gets A {Nonce Pk, Cert1} ∈ set evs      ∧
  Outpts (Card A) A {Nonce Na, Cert3} ∈ set evs      ∧
  Gets A {Nonce Nb, Cert2} ∈ set evs"

```

$\langle proof \rangle$

**lemma** *Outpts\_A\_Card\_4:*

" $\llbracket$  Outpts C A  $\{ \text{Nonce Na}, (\text{Crypt} (\text{crdK} (\text{Card A})) (\text{Nonce Na})) \} \in \text{set evs};$

$\text{evs} \in \text{sr} \rrbracket$

$\implies \text{legalUse}(C) \wedge C = (\text{Card A}) \wedge$   
 $\text{Inputs A} (\text{Card A}) (\text{Agent A}) \in \text{set evs}"$

$\langle proof \rangle$

**lemma** *Outpts\_B\_Card\_7:*

" $\llbracket$  Outpts C B  $\{ \text{Nonce Nb}, \text{Key K},$   
 $\text{Crypt} (\text{pairK}(A,B)) \{ \text{Nonce Na}, \text{Nonce Nb} \},$   
 $\text{Cert2} \} \in \text{set evs};$

$\text{evs} \in \text{sr} \rrbracket$

$\implies \text{legalUse}(C) \wedge C = (\text{Card B}) \wedge$   
 $\text{Inputs B} (\text{Card B}) \{ \text{Agent A}, \text{Nonce Na} \} \in \text{set evs}"$

$\langle proof \rangle$

**lemma** *Outpts\_A\_Card\_10:*

" $\llbracket$  Outpts C A  $\{ \text{Key K}, (\text{Crypt} (\text{pairK}(A,B)) (\text{Nonce Nb})) \} \in \text{set evs};$

$\text{evs} \in \text{sr} \rrbracket$

$\implies \text{legalUse}(C) \wedge C = (\text{Card A}) \wedge$   
 $(\exists \text{ Na Ver1 Ver2 Ver3}.$   
 $\text{Inputs A} (\text{Card A}) \{ \text{Agent B}, \text{Nonce Na}, \text{Nonce Nb}, \text{Nonce} (\text{Pairkey}(A,B)),$

$\text{Ver1}, \text{Ver2}, \text{Ver3} \} \in \text{set evs})"$

$\langle proof \rangle$

**lemma** *Outpts\_A\_Card\_10\_imp\_Inputs:*

" $\llbracket$  Outpts (Card A) A  $\{ \text{Key K}, \text{Certificate} \} \in \text{set evs}; \text{evs} \in \text{sr} \rrbracket$

$\implies (\exists \text{ B Na Nb Ver1 Ver2 Ver3}.$

$\text{Inputs A} (\text{Card A}) \{ \text{Agent B}, \text{Nonce Na}, \text{Nonce Nb}, \text{Nonce} (\text{Pairkey}(A,B)),$

$\text{Ver1}, \text{Ver2}, \text{Ver3} \} \in \text{set evs})"$

$\langle proof \rangle$

**lemma** *Outpts\_honest\_A\_Card\_4:*

" $\llbracket$  Outpts C A  $\{ \text{Nonce Na}, \text{Crypt K X} \} \in \text{set evs};$

$A \neq \text{Spy}; \text{evs} \in \text{sr} \rrbracket$

$\implies \text{legalUse}(C) \wedge C = (\text{Card A}) \wedge$   
 $\text{Inputs A} (\text{Card A}) (\text{Agent A}) \in \text{set evs}"$

$\langle proof \rangle$

**lemma** *Outpts\_honest\_B\_Card\_7:*

"[[ Outpts C B {Nonce Nb, Key K, Cert1, Cert2} ∈ set evs;  
 B ≠ Spy; evs ∈ sr ]]  
 ⇒ legalUse(C) ∧ C = (Card B) ∧  
 (∃ A Na. Inputs B (Card B) {Agent A, Nonce Na} ∈ set evs)"

⟨proof⟩

**lemma** *Outpts\_honest\_A\_Card\_10:*

"[[ Outpts C A {Key K, Certificate} ∈ set evs;  
 A ≠ Spy; evs ∈ sr ]]  
 ⇒ legalUse(C) ∧ C = (Card A) ∧  
 (∃ B Na Nb Pk Ver1 Ver2 Ver3.  
 Inputs A (Card A) {Agent B, Nonce Na, Nonce Nb, Pk,  
 Ver1, Ver2, Ver3} ∈ set evs)"

⟨proof⟩

**lemma** *Outpts\_which\_Card\_4:*

"[[ Outpts (Card A) A {Nonce Na, Crypt K X} ∈ set evs; evs ∈ sr ]]  
 ⇒ Inputs A (Card A) (Agent A) ∈ set evs"

⟨proof⟩

**lemma** *Outpts\_which\_Card\_7:*

"[[ Outpts (Card B) B {Nonce Nb, Key K, Cert1, Cert2} ∈ set evs;  
 evs ∈ sr ]]  
 ⇒ ∃ A Na. Inputs B (Card B) {Agent A, Nonce Na} ∈ set evs"

⟨proof⟩

**lemma** *Outpts\_which\_Card\_10:*

"[[ Outpts (Card A) A {Key (sesK(Nb, pairK(A, B))),  
 Crypt (pairK(A, B)) (Nonce Nb)} ∈ set evs;  
 evs ∈ sr ]]  
 ⇒ ∃ Na. Inputs A (Card A) {Agent B, Nonce Na, Nonce Nb, Nonce (Pairkey(A, B)),  
 Crypt (shrK A) {Nonce (Pairkey(A, B)), Agent B},  
 Crypt (pairK(A, B)) {Nonce Na, Nonce Nb},  
 Crypt (crdK (Card A)) (Nonce Na)} ∈ set evs"

⟨proof⟩

**lemma** *Outpts\_A\_Card\_form\_4:*

"[[ Outpts (Card A) A {Nonce Na, Certificate} ∈ set evs;  
 ∀ p q. Certificate ≠ {p, q}; evs ∈ sr ]]

$\implies \text{Certificate} = (\text{Crypt } (\text{crdK } (\text{Card } A)) \text{ (Nonce Na)})$ "  
 $\langle \text{proof} \rangle$

**lemma** *Outpts\_B\_Card\_form\_7:*  
 $"\llbracket \text{Outpts } (\text{Card } B) \text{ } B \text{ } \{\text{Nonce Nb, Key K, Cert1, Cert2}\} \in \text{set evs};$   
 $\text{evs} \in \text{sr} \rrbracket$   
 $\implies \exists A \text{ Na.}$   
 $K = \text{sesK}(\text{Nb}, \text{pairK}(A, B)) \wedge$   
 $\text{Cert1} = (\text{Crypt } (\text{pairK}(A, B)) \{\text{Nonce Na, Nonce Nb}\}) \wedge$   
 $\text{Cert2} = (\text{Crypt } (\text{pairK}(A, B)) \text{ (Nonce Nb)})"$   
 $\langle \text{proof} \rangle$

**lemma** *Outpts\_A\_Card\_form\_10:*  
 $"\llbracket \text{Outpts } (\text{Card } A) \text{ } A \text{ } \{\text{Key K, Certificate}\} \in \text{set evs}; \text{evs} \in \text{sr} \rrbracket$   
 $\implies \exists B \text{ Nb.}$   
 $K = \text{sesK}(\text{Nb}, \text{pairK}(A, B)) \wedge$   
 $\text{Certificate} = (\text{Crypt } (\text{pairK}(A, B)) \text{ (Nonce Nb)})"$   
 $\langle \text{proof} \rangle$

**lemma** *Outpts\_A\_Card\_form\_bis:*  
 $"\llbracket \text{Outpts } (\text{Card } A') \text{ } A' \text{ } \{\text{Key } (\text{sesK}(\text{Nb}, \text{pairK}(A, B))), \text{Certificate}\} \in \text{set evs};$   
 $\text{evs} \in \text{sr} \rrbracket$   
 $\implies A' = A \wedge$   
 $\text{Certificate} = (\text{Crypt } (\text{pairK}(A, B)) \text{ (Nonce Nb)})"$   
 $\langle \text{proof} \rangle$

**lemma** *Inputs\_A\_Card\_form\_9:*  
 $"\llbracket \text{Inputs } A \text{ (Card } A) \text{ } \{\text{Agent B, Nonce Na, Nonce Nb, Nonce Pk,}$   
 $\text{Cert1, Cert2, Cert3}\} \in \text{set evs};$   
 $\text{evs} \in \text{sr} \rrbracket$   
 $\implies \text{Cert3} = \text{Crypt } (\text{crdK } (\text{Card } A)) \text{ (Nonce Na)}"$   
 $\langle \text{proof} \rangle$

**lemma** *Inputs\_Card\_legalUse:*  
 $"\llbracket \text{Inputs } A \text{ (Card } A) \text{ } X \in \text{set evs}; \text{evs} \in \text{sr} \rrbracket \implies \text{legalUse}(\text{Card } A)"$   
 $\langle \text{proof} \rangle$

**lemma** *Outpts\_Card\_legalUse:*  
 $"\llbracket \text{Outpts } (\text{Card } A) \text{ } A \text{ } X \in \text{set evs}; \text{evs} \in \text{sr} \rrbracket \implies \text{legalUse}(\text{Card } A)"$   
 $\langle \text{proof} \rangle$

**lemma** *Inputs\_Card:*  $"\llbracket \text{Inputs } A \text{ } C \text{ } X \in \text{set evs}; A \neq \text{Spy}; \text{evs} \in \text{sr} \rrbracket$   
 $\implies C = (\text{Card } A) \wedge \text{legalUse}(C)"$

$\langle proof \rangle$

**lemma** *Outpts\_Card*: " $\llbracket \text{Outpts } C \ A \ X \in \text{set evs}; A \neq \text{Spy}; \text{evs} \in \text{sr} \rrbracket$   
 $\implies C = (\text{Card } A) \wedge \text{legalUse}(C)$ "

$\langle proof \rangle$

**lemma** *Inputs\_Outpts\_Card*:  
 $\llbracket \text{Inputs } A \ C \ X \in \text{set evs} \vee \text{Outpts } C \ A \ Y \in \text{set evs};$   
 $A \neq \text{Spy}; \text{evs} \in \text{sr} \rrbracket$   
 $\implies C = (\text{Card } A) \wedge \text{legalUse}(\text{Card } A)$ "

$\langle proof \rangle$

**lemma** *Inputs\_Card\_Spy*:  
 $\llbracket \text{Inputs } \text{Spy} \ C \ X \in \text{set evs} \vee \text{Outpts } C \ \text{Spy} \ X \in \text{set evs}; \text{evs} \in \text{sr} \rrbracket$   
 $\implies C = (\text{Card } \text{Spy}) \wedge \text{legalUse}(\text{Card } \text{Spy}) \vee$   
 $(\exists A. C = (\text{Card } A) \wedge \text{illegalUse}(\text{Card } A))$ "

$\langle proof \rangle$

**lemma** *Outpts\_A\_Card\_unique\_nonce*:  
 $\llbracket \text{Outpts } (\text{Card } A) \ A \ \{\text{Nonce } Na, \text{Crypt } (\text{crdK } (\text{Card } A)) \ (\text{Nonce } Na)\} \in \text{set evs};$   
 $\text{Outpts } (\text{Card } A') \ A' \ \{\text{Nonce } Na, \text{Crypt } (\text{crdK } (\text{Card } A')) \ (\text{Nonce } Na)\} \in \text{set evs};$   
 $\text{evs} \in \text{sr} \rrbracket \implies A=A'$ "

$\langle proof \rangle$

**lemma** *Outpts\_B\_Card\_unique\_nonce*:  
 $\llbracket \text{Outpts } (\text{Card } B) \ B \ \{\text{Nonce } Nb, \text{Key } SK, \text{Cert1}, \text{Cert2}\} \in \text{set evs};$   
 $\text{Outpts } (\text{Card } B') \ B' \ \{\text{Nonce } Nb, \text{Key } SK', \text{Cert1}', \text{Cert2}'\} \in \text{set evs};$   
 $\text{evs} \in \text{sr} \rrbracket \implies B=B' \wedge SK=SK' \wedge \text{Cert1}=\text{Cert1}' \wedge \text{Cert2}=\text{Cert2}'$ "

$\langle proof \rangle$

**lemma** *Outpts\_B\_Card\_unique\_key*:  
 $\llbracket \text{Outpts } (\text{Card } B) \ B \ \{\text{Nonce } Nb, \text{Key } SK, \text{Cert1}, \text{Cert2}\} \in \text{set evs};$



$$\text{Outpts (Card } B') \ B' \ \{\text{Nonce } Nb', \text{ Key } SK, \text{ Cert1}', \text{ Cert2}'\} \in \text{set evs};$$

$$\text{evs} \in \text{sr} \implies B=B' \wedge Nb=Nb' \wedge \text{Cert1}=\text{Cert1}' \wedge \text{Cert2}=\text{Cert2}'$$

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

**lemma** *Outpts\_A\_Card\_unique\_key*: " $\llbracket \text{Outpts (Card } A) \ A \ \{\text{Key } K, V\} \in \text{set evs};$   
 $\text{evs} \in \text{sr} \implies A=A' \wedge V=V'$ "
$$\langle \text{proof} \rangle$$

**lemma** *Outpts\_A\_Card\_Unique*:  

$$\llbracket \text{Outpts (Card } A) \ A \ \{\text{Nonce } Na, \text{ rest}\} \in \text{set evs}; \text{evs} \in \text{sr} \rrbracket$$

$$\implies \text{Unique (Outpts (Card } A) \ A \ \{\text{Nonce } Na, \text{ rest}\}) \text{ on evs}$$

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

**lemma** *Spy\_knows\_Na*:  

$$\llbracket \text{Says } A \ B \ \{\text{Agent } A, \text{ Nonce } Na\} \in \text{set evs}; \text{evs} \in \text{sr} \rrbracket$$

$$\implies \text{Nonce } Na \in \text{analz (knows Spy evs)}$$

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

**lemma** *Spy\_knows\_Nb*:  

$$\llbracket \text{Says } B \ A \ \{\text{Nonce } Nb, \text{ Certificate}\} \in \text{set evs}; \text{evs} \in \text{sr} \rrbracket$$

$$\implies \text{Nonce } Nb \in \text{analz (knows Spy evs)}$$

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

**lemma** *Pairkey\_Gets\_analz\_knows\_Spy*:  

$$\llbracket \text{Gets } A \ \{\text{Nonce (Pairkey(A,B)), Certificate}\} \in \text{set evs}; \text{evs} \in \text{sr} \rrbracket$$

$$\implies \text{Nonce (Pairkey(A,B))} \in \text{analz (knows Spy evs)}$$

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

**lemma** *Pairkey\_Inputs\_imp\_Gets*:  

$$\llbracket \text{Inputs } A \ (\text{Card } A) \ \{\text{Agent } B, \text{ Nonce } Na, \text{ Nonce } Nb, \text{ Nonce (Pairkey(A,B)),}$$

$$\text{Cert1, Cert3, Cert2}\} \in \text{set evs};$$

$$A \neq \text{Spy}; \text{evs} \in \text{sr} \rrbracket$$

$$\implies \text{Gets } A \ \{\text{Nonce (Pairkey(A,B)), Cert1}\} \in \text{set evs}$$

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

```

lemma Pairkey_Inputs_analz_knows_Spy:
  "[[ Inputs A (Card A)
    {Agent B, Nonce Na, Nonce Nb, Nonce (Pairkey(A,B)),
      Cert1, Cert3, Cert2} ∈ set evs;
    evs ∈ sr ]]"
  ⇒ Nonce (Pairkey(A,B)) ∈ analz (knows Spy evs)"
<proof>

```

```

declare shrK_disj_sesK [THEN not_sym, iff]
declare pin_disj_sesK [THEN not_sym, iff]
declare crdK_disj_sesK [THEN not_sym, iff]
declare pairK_disj_sesK [THEN not_sym, iff]

```

<ML>

```

lemma Spy_parts_keys [simp]: "evs ∈ sr ⇒
  (Key (shrK P) ∈ parts (knows Spy evs)) = (Card P ∈ cloned) ∧
  (Key (pin P) ∈ parts (knows Spy evs)) = (P ∈ bad ∨ Card P ∈ cloned) ∧

  (Key (crdK C) ∈ parts (knows Spy evs)) = (C ∈ cloned) ∧
  (Key (pairK(A,B)) ∈ parts (knows Spy evs)) = (Card B ∈ cloned)"
<proof>

```

```

lemma Spy_analz_shrK[simp]: "evs ∈ sr ⇒
  (Key (shrK P) ∈ analz (knows Spy evs)) = (Card P ∈ cloned)"
<proof>

```

```

lemma Spy_analz_crdK[simp]: "evs ∈ sr ⇒
  (Key (crdK C) ∈ analz (knows Spy evs)) = (C ∈ cloned)"
<proof>

```

```

lemma Spy_analz_pairK[simp]: "evs ∈ sr ⇒
  (Key (pairK(A,B)) ∈ analz (knows Spy evs)) = (Card B ∈ cloned)"
<proof>

```

**lemma** *analz\_image\_Key\_Un\_Nonce*:

"*analz* (*Key* ' *K*  $\cup$  *Nonce* ' *N*) = *Key* ' *K*  $\cup$  *Nonce* ' *N*"  
 <*proof*>

<*ML*>

**lemma** *analz\_image\_freshK* [*rule\_format*]:

"*evs*  $\in$  *sr*  $\implies$   $\forall$  *K* *KK*.  
 (*Key* *K*  $\in$  *analz* (*Key*'*KK*  $\cup$  (*knows* *Spy* *evs*))) =  
 (*K*  $\in$  *KK*  $\vee$  *Key* *K*  $\in$  *analz* (*knows* *Spy* *evs*))"

<*proof*>

**lemma** *analz\_insert\_freshK*: "*evs*  $\in$  *sr*  $\implies$

*Key* *K*  $\in$  *analz* (*insert* (*Key* *K'*) (*knows* *Spy* *evs*)) =  
 (*K* = *K'*  $\vee$  *Key* *K*  $\in$  *analz* (*knows* *Spy* *evs*))"

<*proof*>

**lemma** *Na\_Nb\_certificate\_authentic*:

"[ *Crypt* (*pairK*(*A*,*B*)) {*Nonce* *Na*, *Nonce* *Nb*}  $\in$  *parts* (*knows* *Spy* *evs*);  
 $\neg$ *illegalUse*(*Card* *B*);  
*evs*  $\in$  *sr* ]  
 $\implies$  *Outpts* (*Card* *B*) *B* {*Nonce* *Nb*, *Key* (*sesK*(*Nb*,*pairK*(*A*,*B*))),  
*Crypt* (*pairK*(*A*,*B*)) {*Nonce* *Na*, *Nonce* *Nb*},  
*Crypt* (*pairK*(*A*,*B*)) (*Nonce* *Nb*)}  $\in$  *set* *evs*"

<*proof*>

**lemma** *Nb\_certificate\_authentic*:

"[ *Crypt* (*pairK*(*A*,*B*)) (*Nonce* *Nb*)  $\in$  *parts* (*knows* *Spy* *evs*);  
*B*  $\neq$  *Spy*;  $\neg$ *illegalUse*(*Card* *A*);  $\neg$ *illegalUse*(*Card* *B*);  
*evs*  $\in$  *sr* ]  
 $\implies$  *Outpts* (*Card* *A*) *A* {*Key* (*sesK*(*Nb*,*pairK*(*A*,*B*))),  
*Crypt* (*pairK*(*A*,*B*)) (*Nonce* *Nb*)}  $\in$  *set* *evs*"

<*proof*>

**lemma** *Outpts\_A\_Card\_imp\_pairK\_parts*:

"[ *Outpts* (*Card* *A*) *A*  
 {*Key* *K*, *Crypt* (*pairK*(*A*,*B*)) (*Nonce* *Nb*)}  $\in$  *set* *evs*;

```

      evs ∈ sr ]
    ⇒ ∃ Na. Crypt (pairK(A,B)) {Nonce Na, Nonce Nb} ∈ parts (knows Spy
    evs)"
  <proof>

```

```

lemma Nb_certificate_authentic_bis:
  "[ Crypt (pairK(A,B)) (Nonce Nb) ∈ parts (knows Spy evs);
    B ≠ Spy; ¬illegalUse(Card B);
    evs ∈ sr ]
  ⇒ ∃ Na. Outpts (Card B) B {Nonce Nb, Key (sesK(Nb,pairK(A,B))),
    Crypt (pairK(A,B)) {Nonce Na, Nonce Nb},
    Crypt (pairK(A,B)) (Nonce Nb)} ∈ set evs"
  <proof>

```

```

lemma Pairkey_certificate_authentic:
  "[ Crypt (shrK A) {Nonce Pk, Agent B} ∈ parts (knows Spy evs);
    Card A ∉ cloned; evs ∈ sr ]
  ⇒ Pk = Pairkey(A,B) ∧
    Says Server A {Nonce Pk,
    Crypt (shrK A) {Nonce Pk, Agent B}}
    ∈ set evs"
  <proof>

```

```

lemma sesK_authentic:
  "[ Key (sesK(Nb,pairK(A,B))) ∈ parts (knows Spy evs);
    A ≠ Spy; B ≠ Spy; ¬illegalUse(Card A); ¬illegalUse(Card B);
    evs ∈ sr ]
  ⇒ Notes Spy {Key (sesK(Nb,pairK(A,B))), Nonce Nb, Agent A, Agent B}
    ∈ set evs"
  <proof>

```

```

lemma Confidentiality:
  "[ Notes Spy {Key (sesK(Nb,pairK(A,B))), Nonce Nb, Agent A, Agent B}
    ∉ set evs;
    A ≠ Spy; B ≠ Spy; ¬illegalUse(Card A); ¬illegalUse(Card B);
    evs ∈ sr ]
  ⇒ Key (sesK(Nb,pairK(A,B))) ∉ analz (knows Spy evs)"
  <proof>

```

**lemma Confidentiality\_B:**

```
"[[ Outputs (Card B) B {Nonce Nb, Key K, Certificate,
      Crypt (pairK(A,B)) (Nonce Nb)} ∈ set evs;
   Notes Spy {Key K, Nonce Nb, Agent A, Agent B} ∉ set evs;
   A ≠ Spy; B ≠ Spy; ¬illegalUse(Card A); Card B ∉ cloned;
   evs ∈ sr ]]
⇒ Key K ∉ analz (knows Spy evs)"
⟨proof⟩
```

**lemma A\_authenticates\_B:**

```
"[[ Outputs (Card A) A {Key K, Crypt (pairK(A,B)) (Nonce Nb)} ∈ set evs;
   ¬illegalUse(Card B);
   evs ∈ sr ]]
⇒ ∃ Na.
   Outputs (Card B) B {Nonce Nb, Key K,
     Crypt (pairK(A,B)) {Nonce Na, Nonce Nb},
     Crypt (pairK(A,B)) (Nonce Nb)} ∈ set evs"
⟨proof⟩
```

**lemma A\_authenticates\_B\_Gets:**

```
"[[ Gets A {Nonce Nb, Crypt (pairK(A,B)) {Nonce Na, Nonce Nb}}
   ∈ set evs;
   ¬illegalUse(Card B);
   evs ∈ sr ]]
⇒ Outputs (Card B) B {Nonce Nb, Key (sesK(Nb, pairK (A, B))),
     Crypt (pairK(A,B)) {Nonce Na, Nonce Nb},
     Crypt (pairK(A,B)) (Nonce Nb)} ∈ set evs"
⟨proof⟩
```

**lemma B\_authenticates\_A:**

```
"[[ Gets B (Crypt (pairK(A,B)) (Nonce Nb)) ∈ set evs;
   B ≠ Spy; ¬illegalUse(Card A); ¬illegalUse(Card B);
   evs ∈ sr ]]
⇒ Outputs (Card A) A
   {Key (sesK(Nb,pairK(A,B))), Crypt (pairK(A,B)) (Nonce Nb)} ∈ set evs"
⟨proof⟩
```

**lemma Confidentiality\_A:** "[[ Outputs (Card A) A

$\{ \text{Key } K, \text{Crypt } (\text{pairK}(A,B)) (\text{Nonce } Nb) \} \in \text{set evs};$   
 $\text{Notes Spy } \{ \text{Key } K, \text{Nonce } Nb, \text{Agent } A, \text{Agent } B \} \notin \text{set evs};$   
 $A \neq \text{Spy}; B \neq \text{Spy}; \neg \text{illegalUse}(\text{Card } A); \neg \text{illegalUse}(\text{Card } B);$   
 $\text{evs} \in \text{sr} \parallel$   
 $\implies \text{Key } K \notin \text{analz } (\text{knows Spy evs})"$   
 $\langle \text{proof} \rangle$

**lemma** *Outpts\_imp\_knows\_agents\_secureM\_sr*:  
 $" \parallel \text{Outpts } (\text{Card } A) A X \in \text{set evs}; \text{evs} \in \text{sr} \parallel \implies X \in \text{knows } A \text{ evs}"$   
 $\langle \text{proof} \rangle$

**lemma** *A\_keydist\_to\_B*:  
 $" \parallel \text{Outpts } (\text{Card } A) A$   
 $\{ \text{Key } K, \text{Crypt } (\text{pairK}(A,B)) (\text{Nonce } Nb) \} \in \text{set evs};$   
 $\neg \text{illegalUse}(\text{Card } B);$   
 $\text{evs} \in \text{sr} \parallel$   
 $\implies \text{Key } K \in \text{analz } (\text{knows } B \text{ evs})"$   
 $\langle \text{proof} \rangle$

**lemma** *B\_keydist\_to\_A*:  
 $" \parallel \text{Outpts } (\text{Card } B) B \{ \text{Nonce } Nb, \text{Key } K, \text{Certificate},$   
 $\text{Crypt } (\text{pairK}(A,B)) (\text{Nonce } Nb) \} \in \text{set evs};$   
 $\text{Gets } B (\text{Crypt } (\text{pairK}(A,B)) (\text{Nonce } Nb)) \in \text{set evs};$   
 $B \neq \text{Spy}; \neg \text{illegalUse}(\text{Card } A); \neg \text{illegalUse}(\text{Card } B);$   
 $\text{evs} \in \text{sr} \parallel$   
 $\implies \text{Key } K \in \text{analz } (\text{knows } A \text{ evs})"$   
 $\langle \text{proof} \rangle$

**lemma** *Nb\_certificate\_authentic\_B*:  
 $" \parallel \text{Gets } B (\text{Crypt } (\text{pairK}(A,B)) (\text{Nonce } Nb)) \in \text{set evs};$   
 $B \neq \text{Spy}; \neg \text{illegalUse}(\text{Card } B);$   
 $\text{evs} \in \text{sr} \parallel$   
 $\implies \exists Na.$   
 $\text{Outpts } (\text{Card } B) B \{ \text{Nonce } Nb, \text{Key } (\text{sesK}(Nb, \text{pairK}(A,B))),$   
 $\text{Crypt } (\text{pairK}(A,B)) \{ \text{Nonce } Na, \text{Nonce } Nb \} \},$

$\langle proof \rangle$   $Crypt\ (pairK(A,B))\ (Nonce\ Nb)\} \in set\ evs"$

**lemma** *Pairkey\_certificate\_authentic\_A\_Card:*  
 "[[ Inputs A (Card A)  
   {Agent B, Nonce Na, Nonce Nb, Nonce Pk,  
   Crypt (shrK A) {Nonce Pk, Agent B},  
   Cert2, Cert3} ∈ set evs;  
   A ≠ Spy; Card A ∉ cloned; evs ∈ sr ]]  
 ⇒ Pk = Pairkey(A,B) ∧  
   Says Server A {Nonce (Pairkey(A,B)),  
   Crypt (shrK A) {Nonce (Pairkey(A,B)), Agent B}}  
   ∈ set evs "  
 $\langle proof \rangle$

**lemma** *Na\_Nb\_certificate\_authentic\_A\_Card:*  
 "[[ Inputs A (Card A)  
   {Agent B, Nonce Na, Nonce Nb, Nonce Pk,  
   Cert1,  
   Crypt (pairK(A,B)) {Nonce Na, Nonce Nb}, Cert3} ∈ set evs;  
  
   A ≠ Spy; ¬illegalUse(Card B); evs ∈ sr ]]  
 ⇒ Outputs (Card B) B {Nonce Nb, Key (sesK(Nb, pairK (A, B))),  
   Crypt (pairK(A,B)) {Nonce Na, Nonce Nb},  
   Crypt (pairK(A,B)) (Nonce Nb)}  
   ∈ set evs "  
 $\langle proof \rangle$

**lemma** *Na\_authentic\_A\_Card:*  
 "[[ Inputs A (Card A)  
   {Agent B, Nonce Na, Nonce Nb, Nonce Pk,  
   Cert1, Cert2, Cert3} ∈ set evs;  
   A ≠ Spy; evs ∈ sr ]]  
 ⇒ Outputs (Card A) A {Nonce Na, Cert3}  
   ∈ set evs "  
 $\langle proof \rangle$

**lemma** *Inputs\_A\_Card\_9\_authentic:*  
 "[[ Inputs A (Card A)  
   {Agent B, Nonce Na, Nonce Nb, Nonce Pk,  
   Crypt (shrK A) {Nonce Pk, Agent B},  
   Crypt (pairK(A,B)) {Nonce Na, Nonce Nb}, Cert3} ∈ set evs;

$A \neq \text{Spy}; \text{Card } A \notin \text{cloned}; \neg \text{illegalUse}(\text{Card } B); \text{evs} \in \text{sr} \parallel$   
 $\implies \text{Says Server } A \{ \{ \text{Nonce } Pk, \text{Crypt } (\text{shrK } A) \{ \text{Nonce } Pk, \text{Agent } B \} \} \}$   
 $\quad \in \text{set evs} \wedge$   
 $\text{Outpts } (\text{Card } B) B \{ \{ \text{Nonce } Nb, \text{Key } (\text{sesK}(Nb, \text{pairK } (A, B))),$   
 $\quad \quad \text{Crypt } (\text{pairK}(A, B)) \{ \text{Nonce } Na, \text{Nonce } Nb \},$   
 $\quad \quad \text{Crypt } (\text{pairK}(A, B)) (\text{Nonce } Nb) \}$   
 $\quad \in \text{set evs} \wedge$   
 $\text{Outpts } (\text{Card } A) A \{ \{ \text{Nonce } Na, \text{Cert3} \}$   
 $\quad \in \text{set evs} "$   
 $\langle \text{proof} \rangle$

**lemma SR4\_imp:**  
 $" \parallel \text{Outpts } (\text{Card } A) A \{ \{ \text{Nonce } Na, \text{Crypt } (\text{crdK } (\text{Card } A)) (\text{Nonce } Na) \} \}$   
 $\quad \in \text{set evs};$   
 $A \neq \text{Spy}; \text{evs} \in \text{sr} \parallel$   
 $\implies \exists Pk V. \text{Gets } A \{ \{ Pk, V \} \} \in \text{set evs} "$   
 $\langle \text{proof} \rangle$

**lemma SR7\_imp:**  
 $" \parallel \text{Outpts } (\text{Card } B) B \{ \{ \text{Nonce } Nb, \text{Key } K,$   
 $\quad \quad \text{Crypt } (\text{pairK}(A, B)) \{ \text{Nonce } Na, \text{Nonce } Nb \},$   
 $\quad \quad \text{Cert2} \} \in \text{set evs};$   
 $B \neq \text{Spy}; \text{evs} \in \text{sr} \parallel$   
 $\implies \text{Gets } B \{ \{ \text{Agent } A, \text{Nonce } Na \} \} \in \text{set evs} "$   
 $\langle \text{proof} \rangle$

**lemma SR10\_imp:**  
 $" \parallel \text{Outpts } (\text{Card } A) A \{ \{ \text{Key } K, \text{Crypt } (\text{pairK}(A, B)) (\text{Nonce } Nb) \} \}$   
 $\quad \in \text{set evs};$   
 $A \neq \text{Spy}; \text{evs} \in \text{sr} \parallel$   
 $\implies \exists \text{Cert1 } \text{Cert2}.$   
 $\quad \text{Gets } A \{ \{ \text{Nonce } (\text{Pairkey } (A, B)), \text{Cert1} \} \} \in \text{set evs} \wedge$   
 $\quad \text{Gets } A \{ \{ \text{Nonce } Nb, \text{Cert2} \} \} \in \text{set evs} "$   
 $\langle \text{proof} \rangle$



**lemma** *Outpts\_Server\_not\_evs*: "evs  $\in$  sr  $\implies$  Outpts (Card Server) P X  $\notin$  set evs"  
 <proof>

*step2\_integrity* also is a reliability theorem

**lemma** *Says\_Server\_message\_form*:  
 "[ Says Server A {Pk, Certificate}  $\in$  set evs;  
   evs  $\in$  sr ]  
 $\implies \exists B. Pk = \text{Nonce } (\text{Pairkey}(A,B)) \wedge$   
   Certificate = Crypt (shrK A) {Nonce (Pairkey(A,B)), Agent B}"  
 <proof>

*step4integrity* is *Outpts\_A\_Card\_form\_4*

*step7integrity* is *Outpts\_B\_Card\_form\_7*

**lemma** *step8\_integrity*:  
 "[ Says B A {Nonce Nb, Certificate}  $\in$  set evs;  
   B  $\neq$  Server; B  $\neq$  Spy; evs  $\in$  sr ]  
 $\implies \exists \text{Cert2 } K.$   
   Outpts (Card B) B {Nonce Nb, Key K, Certificate, Cert2}  $\in$  set evs"  
 <proof>

*step9integrity* is *Inputs\_A\_Card\_form\_9*

*step10integrity* is *Outpts\_A\_Card\_form\_10*.

**lemma** *step11\_integrity*:  
 "[ Says A B (Certificate)  $\in$  set evs;  
    $\forall p \ q. \text{Certificate} \neq \{p, q\};$   
   A  $\neq$  Spy; evs  $\in$  sr ]  
 $\implies \exists K.$   
   Outpts (Card A) A {Key K, Certificate}  $\in$  set evs"  
 <proof>

end

## 27 Bella's modification of the Shoup-Rubin protocol

**theory** *ShoupRubinBella* imports *Smartcard* begin

The modifications are that message 7 now mentions A, while message 10 now mentions Nb and B. The lack of explicitness of the original version was discovered by investigating adherence to the principle of Goal Availability. Only the updated version makes the goals of confidentiality, authentication and key distribution available to both peers.

**axiomatization** *sesK* :: "nat\*key  $\Rightarrow$  key"  
**where**

*inj\_sesK [iff]*: "(sesK(m,k) = sesK(m',k')) = (m = m'  $\wedge$  k = k')" and

```

shrK_disj_sesK [iff]: "shrK A  $\neq$  sesK(m,pk)" and
crdK_disj_sesK [iff]: "crdK C  $\neq$  sesK(m,pk)" and
pin_disj_sesK [iff]: "pin P  $\neq$  sesK(m,pk)" and
pairK_disj_sesK [iff]: "pairK(A,B)  $\neq$  sesK(m,pk)" and

Atomic_distrib [iff]: "Atomic'(KEY'K  $\cup$  NONCE'N) =
                        Atomic'(KEY'K)  $\cup$  Atomic'(NONCE'N)" and

shouprubin_assumes_securemeans [iff]: "evs  $\in$  srb  $\implies$  secureM"

definition Unique :: "[event, event list] => bool" (<Unique _ on _>) where
  "Unique ev on evs ==
   ev  $\notin$  set (tl (dropWhile (% z. z  $\neq$  ev) evs))"

inductive_set srb :: "event list set"
where

  Nil: "[]  $\in$  srb"

  / Fake: "[[ evsF  $\in$  srb; X  $\in$  synth (analz (knows Spy evsF));
             illegalUse(Card B) ]
             $\implies$  Says Spy A X #
             Inputs Spy (Card B) X # evsF  $\in$  srb"

  / Forge:
    "[[ evsFo  $\in$  srb; Nonce Nb  $\in$  analz (knows Spy evsFo);
       Key (pairK(A,B))  $\in$  knows Spy evsFo ]
       $\implies$  Notes Spy (Key (sesK(Nb,pairK(A,B)))) # evsFo  $\in$  srb"

  / Reception: "[[ evsrb  $\in$  srb; Says A B X  $\in$  set evsrb ]
                 $\implies$  Gets B X # evsrb  $\in$  srb"

  / SR_U1: "[[ evs1  $\in$  srb; A  $\neq$  Server ]
             $\implies$  Says A Server {Agent A, Agent B}
             # evs1  $\in$  srb"

  / SR_U2: "[[ evs2  $\in$  srb;
             Gets Server {Agent A, Agent B}  $\in$  set evs2 ]
             $\implies$  Says Server A {Nonce (Pairkey(A,B)),
                               Crypt (shrK A) {Nonce (Pairkey(A,B)), Agent B}
             }
             # evs2  $\in$  srb"

```

```

/ SR_U3: "[ evs3 ∈ srb; legalUse(Card A);
           Says A Server {Agent A, Agent B} ∈ set evs3;
           Gets A {Nonce Pk, Certificate} ∈ set evs3 ]
⇒ Inputs A (Card A) (Agent A)
    # evs3 ∈ srb"

/ SR_U4: "[ evs4 ∈ srb;
           Nonce Na ∉ used evs4; legalUse(Card A); A ≠ Server;
           Inputs A (Card A) (Agent A) ∈ set evs4 ]
⇒ Outpts (Card A) A {Nonce Na, Crypt (crdK (Card A)) (Nonce Na)}
    # evs4 ∈ srb"

/ SR_U4Fake: "[ evs4F ∈ srb; Nonce Na ∉ used evs4F;
               illegalUse(Card A);
               Inputs Spy (Card A) (Agent A) ∈ set evs4F ]
⇒ Outpts (Card A) Spy {Nonce Na, Crypt (crdK (Card A)) (Nonce Na)}
    # evs4F ∈ srb"

/ SR_U5: "[ evs5 ∈ srb;
           Outpts (Card A) A {Nonce Na, Certificate} ∈ set evs5;
           ∀ p q. Certificate ≠ {p, q} ]
⇒ Says A B {Agent A, Nonce Na} # evs5 ∈ srb"

/ SR_U6: "[ evs6 ∈ srb; legalUse(Card B);
           Gets B {Agent A, Nonce Na} ∈ set evs6 ]
⇒ Inputs B (Card B) {Agent A, Nonce Na}
    # evs6 ∈ srb"

/ SR_U7: "[ evs7 ∈ srb;
           Nonce Nb ∉ used evs7; legalUse(Card B); B ≠ Server;
           K = sesK(Nb, pairK(A, B));
           Key K ∉ used evs7;
           Inputs B (Card B) {Agent A, Nonce Na} ∈ set evs7 ]
⇒ Outpts (Card B) B {Nonce Nb, Agent A, Key K,
                    Crypt (pairK(A, B)) {Nonce Na, Nonce Nb},
                    Crypt (pairK(A, B)) (Nonce Nb)}

```

```

# evs7 ∈ srb"

/ SR_U7Fake: "[ evs7F ∈ srb; Nonce Nb ∉ used evs7F;
  illegalUse(Card B);
  K = sesK(Nb, pairK(A, B));
  Key K ∉ used evs7F;
  Inputs Spy (Card B) {Agent A, Nonce Na} ∈ set evs7F ]
⇒ Outputs (Card B) Spy {Nonce Nb, Agent A, Key K,
  Crypt (pairK(A, B)) {Nonce Na, Nonce Nb},
  Crypt (pairK(A, B)) (Nonce Nb)}
  # evs7F ∈ srb"

/ SR_U8: "[ evs8 ∈ srb;
  Inputs B (Card B) {Agent A, Nonce Na} ∈ set evs8;
  Outputs (Card B) B {Nonce Nb, Agent A, Key K,
  Cert1, Cert2} ∈ set evs8 ]
⇒ Says B A {Nonce Nb, Cert1} # evs8 ∈ srb"

/ SR_U9: "[ evs9 ∈ srb; legalUse(Card A);
  Gets A {Nonce Pk, Cert1} ∈ set evs9;
  Outputs (Card A) A {Nonce Na, Cert2} ∈ set evs9;
  Gets A {Nonce Nb, Cert3} ∈ set evs9;
  ∀ p q. Cert2 ≠ {p, q} ]
⇒ Inputs A (Card A)
  {Agent B, Nonce Na, Nonce Nb, Nonce Pk,
  Cert1, Cert3, Cert2}
  # evs9 ∈ srb"

/ SR_U10: "[ evs10 ∈ srb; legalUse(Card A); A ≠ Server;
  K = sesK(Nb, pairK(A, B));
  Inputs A (Card A) {Agent B, Nonce Na, Nonce Nb,
  Nonce (Pairkey(A, B)),
  Crypt (shrK A) {Nonce (Pairkey(A, B)),
  Agent B},
  Crypt (pairK(A, B)) {Nonce Na, Nonce Nb},
  Crypt (crdK (Card A)) (Nonce Na)}
  ∈ set evs10 ]
⇒ Outputs (Card A) A {Agent B, Nonce Nb,
  Key K, Crypt (pairK(A, B)) (Nonce Nb)}
  # evs10 ∈ srb"

```

```

/ SR_U10Fake: "[ evs10F ∈ srb;
  illegalUse(Card A);
  K = sesK(Nb, pairK(A,B));
  Inputs Spy (Card A) {Agent B, Nonce Na, Nonce Nb,
    Nonce (Pairkey(A,B)),
    Crypt (shrK A) {Nonce (Pairkey(A,B)),
      Agent B},
    Crypt (pairK(A,B)) {Nonce Na, Nonce Nb},
    Crypt (crdK (Card A)) (Nonce Na)}
  ∈ set evs10F ]
⇒ Outputs (Card A) Spy {Agent B, Nonce Nb,
  Key K, Crypt (pairK(A,B)) (Nonce Nb)}
  # evs10F ∈ srb"

```

```

/ SR_U11: "[ evs11 ∈ srb;
  Says A Server {Agent A, Agent B} ∈ set evs11;
  Outputs (Card A) A {Agent B, Nonce Nb, Key K, Certificate}
  ∈ set evs11 ]
⇒ Says A B (Certificate)
  # evs11 ∈ srb"

```

```

/ Ops1:
  "[ evs01 ∈ srb;
    Outputs (Card B) B {Nonce Nb, Agent A, Key K, Cert1, Cert2}
    ∈ set evs01 ]
  ⇒ Notes Spy {Key K, Nonce Nb, Agent A, Agent B} # evs01 ∈ srb"

```

```

/ Ops2:
  "[ evs02 ∈ srb;
    Outputs (Card A) A {Agent B, Nonce Nb, Key K, Certificate}
    ∈ set evs02 ]
  ⇒ Notes Spy {Key K, Nonce Nb, Agent A, Agent B} # evs02 ∈ srb"

```

```

declare Fake_parts_insert_in_Un [dest]
declare analz_into_parts [dest]

```

**lemma** *Gets\_imp\_Says*:  
 "[[ Gets B X ∈ set evs; evs ∈ srb ]] ⇒ ∃ A. Says A B X ∈ set evs"  
 <proof>

**lemma** *Gets\_imp\_knows\_Spy*:  
 "[[ Gets B X ∈ set evs; evs ∈ srb ]] ⇒ X ∈ knows Spy evs"  
 <proof>

**lemma** *Gets\_imp\_knows\_Spy\_parts\_Snd*:  
 "[[ Gets B {X, Y} ∈ set evs; evs ∈ srb ]] ⇒ Y ∈ parts (knows Spy evs)"  
 <proof>

**lemma** *Gets\_imp\_knows\_Spy\_analz\_Snd*:  
 "[[ Gets B {X, Y} ∈ set evs; evs ∈ srb ]] ⇒ Y ∈ analz (knows Spy evs)"  
 <proof>

**lemma** *Inputs\_imp\_knows\_Spy\_secureM\_srb*:  
 "[[ Inputs Spy C X ∈ set evs; evs ∈ srb ]] ⇒ X ∈ knows Spy evs"  
 <proof>

**lemma** *knows\_Spy\_Inputs\_secureM\_srb\_Spy*:  
 "evs ∈ srb ⇒ knows Spy (Inputs Spy C X # evs) = insert X (knows Spy evs)"  
 <proof>

**lemma** *knows\_Spy\_Inputs\_secureM\_srb*:  
 "[[ A ≠ Spy; evs ∈ srb ]] ⇒ knows Spy (Inputs A C X # evs) = knows Spy evs"  
 <proof>

**lemma** *knows\_Spy\_Outpts\_secureM\_srb\_Spy*:  
 "evs ∈ srb ⇒ knows Spy (Outpts C Spy X # evs) = insert X (knows Spy evs)"  
 <proof>

**lemma** *knows\_Spy\_Outpts\_secureM\_srb*:  
 "[[ A ≠ Spy; evs ∈ srb ]] ⇒ knows Spy (Outpts C A X # evs) = knows Spy evs"  
 <proof>

**lemma** *Inputs\_A\_Card\_3:*

"[ Inputs A C (Agent A) ∈ set evs; A ≠ Spy; evs ∈ srb ]  
 ⇒ legalUse(C) ∧ C = (Card A) ∧  
 (∃ Pk Certificate. Gets A {Pk, Certificate} ∈ set evs)"  
 <proof>

**lemma** *Inputs\_B\_Card\_6:*

"[ Inputs B C {Agent A, Nonce Na} ∈ set evs; B ≠ Spy; evs ∈ srb ]  
 ⇒ legalUse(C) ∧ C = (Card B) ∧ Gets B {Agent A, Nonce Na} ∈ set evs"  
 <proof>

**lemma** *Inputs\_A\_Card\_9:*

"[ Inputs A C {Agent B, Nonce Na, Nonce Nb, Nonce Pk,  
 Cert1, Cert2, Cert3} ∈ set evs;  
 A ≠ Spy; evs ∈ srb ]  
 ⇒ legalUse(C) ∧ C = (Card A) ∧  
 Gets A {Nonce Pk, Cert1} ∈ set evs ∧  
 Outpts (Card A) A {Nonce Na, Cert3} ∈ set evs ∧  
 Gets A {Nonce Nb, Cert2} ∈ set evs"  
 <proof>

**lemma** *Outpts\_A\_Card\_4:*

"[ Outpts C A {Nonce Na, (Crypt (crdK (Card A)) (Nonce Na))} ∈ set evs;  
 evs ∈ srb ]  
 ⇒ legalUse(C) ∧ C = (Card A) ∧  
 Inputs A (Card A) (Agent A) ∈ set evs"  
 <proof>

**lemma** *Outpts\_B\_Card\_7:*

"[ Outpts C B {Nonce Nb, Agent A, Key K,  
 Crypt (pairK(A,B)) {Nonce Na, Nonce Nb},  
 Cert2} ∈ set evs;  
 evs ∈ srb ]  
 ⇒ legalUse(C) ∧ C = (Card B) ∧  
 Inputs B (Card B) {Agent A, Nonce Na} ∈ set evs"  
 <proof>

**lemma** *Outpts\_A\_Card\_10:*

"[ Outpts C A {Agent B, Nonce Nb,  
 Key K, (Crypt (pairK(A,B)) (Nonce Nb))} ∈ set evs;  
 evs ∈ srb ]  
 ⇒ legalUse(C) ∧ C = (Card A) ∧  
 (∃ Na Ver1 Ver2 Ver3.  
 Inputs A (Card A) {Agent B, Nonce Na, Nonce Nb, Nonce (Pairkey(A,B)),  
 Ver1, Ver2, Ver3} ∈ set evs)"

```
lemma Outpts_which_Card_7:
  "[[ Outpts (Card B) B {Nonce Nb, Agent A, Key K, Cert1, Cert2}
```



$\in \text{set evs}; \text{ evs} \in \text{srb} \ ]$   
 $\implies \exists \text{ Na. Inputs } B \text{ (Card } B) \ \{ \text{Agent } A, \text{ Nonce Na} \} \in \text{set evs}$ "  
 <proof>

**lemma** *Outpts\_which\_Card\_10:*

$"[ \text{Outpts (Card } A) \ A \ \{ \text{Agent } B, \text{ Nonce Nb, Key } K, \text{ Certificate} \} \in \text{set evs};$   
 $\text{ evs} \in \text{srb} \ ]$   
 $\implies \exists \text{ Na. Inputs } A \text{ (Card } A) \ \{ \text{Agent } B, \text{ Nonce Na, Nonce Nb, Nonce (Pairkey(A,B)),}$   
 $\text{ Crypt (shrK } A) \ \{ \text{Nonce (Pairkey(A,B)), Agent } B \},$   
 $\text{ Crypt (pairK(A,B)) } \{ \text{Nonce Na, Nonce Nb} \},$   
 $\text{ Crypt (crdK (Card } A)) \text{ (Nonce Na)} \} \in \text{set evs}"$   
 <proof>

**lemma** *Outpts\_A\_Card\_form\_4:*

$"[ \text{Outpts (Card } A) \ A \ \{ \text{Nonce Na, Certificate} \} \in \text{set evs};$   
 $\forall p \ q. \text{ Certificate} \neq \{ p, q \}; \text{ evs} \in \text{srb} \ ]$   
 $\implies \text{ Certificate} = (\text{Crypt (crdK (Card } A)) \text{ (Nonce Na)})"$   
 <proof>

**lemma** *Outpts\_B\_Card\_form\_7:*

$"[ \text{Outpts (Card } B) \ B \ \{ \text{Nonce Nb, Agent } A, \text{ Key } K, \text{ Cert1, Cert2} \}$   
 $\in \text{set evs}; \text{ evs} \in \text{srb} \ ]$   
 $\implies \exists \text{ Na.}$   
 $\text{ K} = \text{sesK(Nb, pairK(A,B))} \wedge$   
 $\text{ Cert1} = (\text{Crypt (pairK(A,B)) } \{ \text{Nonce Na, Nonce Nb} \}) \wedge$   
 $\text{ Cert2} = (\text{Crypt (pairK(A,B)) (Nonce Nb)})"$   
 <proof>

**lemma** *Outpts\_A\_Card\_form\_10:*

$"[ \text{Outpts (Card } A) \ A \ \{ \text{Agent } B, \text{ Nonce Nb, Key } K, \text{ Certificate} \}$   
 $\in \text{set evs}; \text{ evs} \in \text{srb} \ ]$   
 $\implies \text{ K} = \text{sesK(Nb, pairK(A,B))} \wedge$   
 $\text{ Certificate} = (\text{Crypt (pairK(A,B)) (Nonce Nb)})"$   
 <proof>

**lemma** *Outpts\_A\_Card\_form\_bis:*

$"[ \text{Outpts (Card } A') \ A' \ \{ \text{Agent } B', \text{ Nonce Nb', Key (sesK(Nb, pairK(A,B))),}$   
 $\text{ Certificate} \} \in \text{set evs};$   
 $\text{ evs} \in \text{srb} \ ]$   
 $\implies \text{ A' } = \text{ A} \wedge \text{ B' } = \text{ B} \wedge \text{ Nb } = \text{ Nb' } \wedge$   
 $\text{ Certificate} = (\text{Crypt (pairK(A,B)) (Nonce Nb)})"$   
 <proof>

**lemma** *Inputs\_A\_Card\_form\_9:*

"[[ Inputs A (Card A) {Agent B, Nonce Na, Nonce Nb, Nonce Pk,  
Cert1, Cert2, Cert3} ∈ set evs;  
evs ∈ srb ]]  
⇒ Cert3 = Crypt (crdK (Card A)) (Nonce Na)"  
⟨proof⟩

**lemma** *Inputs\_Card\_legalUse:*

"[[ Inputs A (Card A) X ∈ set evs; evs ∈ srb ] ⇒ legalUse(Card A)"  
⟨proof⟩

**lemma** *Outpts\_Card\_legalUse:*

"[[ Outpts (Card A) A X ∈ set evs; evs ∈ srb ] ⇒ legalUse(Card A)"  
⟨proof⟩

**lemma** *Inputs\_Card:* "[[ Inputs A C X ∈ set evs; A ≠ Spy; evs ∈ srb ]]  
⇒ C = (Card A) ∧ legalUse(C)"  
⟨proof⟩

**lemma** *Outpts\_Card:* "[[ Outpts C A X ∈ set evs; A ≠ Spy; evs ∈ srb ]]  
⇒ C = (Card A) ∧ legalUse(C)"  
⟨proof⟩

**lemma** *Inputs\_Outpts\_Card:*

"[[ Inputs A C X ∈ set evs ∨ Outpts C A Y ∈ set evs;  
A ≠ Spy; evs ∈ srb ]]  
⇒ C = (Card A) ∧ legalUse(Card A)"  
⟨proof⟩

**lemma** *Inputs\_Card\_Spy:*

"[[ Inputs Spy C X ∈ set evs ∨ Outpts C Spy X ∈ set evs; evs ∈ srb ]]  
⇒ C = (Card Spy) ∧ legalUse(Card Spy) ∨  
(∃ A. C = (Card A) ∧ illegalUse(Card A))"  
⟨proof⟩

**lemma** *Outpts\_A\_Card\_unique\_nonce:*

"[[ Outpts (Card A) A {Nonce Na, Crypt (crdK (Card A)) (Nonce Na)}  
     ∈ set evs;  
     Outpts (Card A') A' {Nonce Na, Crypt (crdK (Card A')) (Nonce Na)}  
     ∈ set evs;  
     evs ∈ srb ] ⇒ A=A'"]

⟨proof⟩

**lemma** *Outpts\_B\_Card\_unique\_nonce:*

"[[ Outpts (Card B) B {Nonce Nb, Agent A, Key SK, Cert1, Cert2} ∈ set  
     evs;  
     Outpts (Card B') B' {Nonce Nb, Agent A', Key SK', Cert1', Cert2'} ∈  
     set evs;  
     evs ∈ srb ] ⇒ B=B' ∧ A=A' ∧ SK=SK' ∧ Cert1=Cert1' ∧ Cert2=Cert2'"]

⟨proof⟩

**lemma** *Outpts\_B\_Card\_unique\_key:*

"[[ Outpts (Card B) B {Nonce Nb, Agent A, Key SK, Cert1, Cert2} ∈ set  
     evs;  
     Outpts (Card B') B' {Nonce Nb', Agent A', Key SK, Cert1', Cert2'} ∈  
     set evs;  
     evs ∈ srb ] ⇒ B=B' ∧ A=A' ∧ Nb=Nb' ∧ Cert1=Cert1' ∧ Cert2=Cert2'"]

⟨proof⟩

**lemma** *Outpts\_A\_Card\_unique\_key:*

"[[ Outpts (Card A) A {Agent B, Nonce Nb, Key K, V} ∈ set evs;  
     Outpts (Card A') A' {Agent B', Nonce Nb', Key K, V'} ∈ set evs;  
     evs ∈ srb ] ⇒ A=A' ∧ B=B' ∧ Nb=Nb' ∧ V=V'"]

⟨proof⟩

**lemma** *Outpts\_A\_Card\_Unique:*

"[[ Outpts (Card A) A {Nonce Na, rest} ∈ set evs; evs ∈ srb ]  
     ⇒ Unique (Outpts (Card A) A {Nonce Na, rest}) on evs"]

⟨proof⟩

**lemma** *Spy\_knows\_Na*:  
 "[[ Says A B {Agent A, Nonce Na} ∈ set evs; evs ∈ srb ]]  
 ⇒ Nonce Na ∈ analz (knows Spy evs)"  
 <proof>

**lemma** *Spy\_knows\_Nb*:  
 "[[ Says B A {Nonce Nb, Certificate} ∈ set evs; evs ∈ srb ]]  
 ⇒ Nonce Nb ∈ analz (knows Spy evs)"  
 <proof>

**lemma** *Pairkey\_Gets\_analz\_knows\_Spy*:  
 "[[ Gets A {Nonce (Pairkey(A,B)), Certificate} ∈ set evs; evs ∈ srb ]]  
 ⇒ Nonce (Pairkey(A,B)) ∈ analz (knows Spy evs)"  
 <proof>

**lemma** *Pairkey\_Inputs\_imp\_Gets*:  
 "[[ Inputs A (Card A)  
 {Agent B, Nonce Na, Nonce Nb, Nonce (Pairkey(A,B)),  
 Cert1, Cert3, Cert2} ∈ set evs;  
 A ≠ Spy; evs ∈ srb ]]  
 ⇒ Gets A {Nonce (Pairkey(A,B)), Cert1} ∈ set evs"  
 <proof>

**lemma** *Pairkey\_Inputs\_analz\_knows\_Spy*:  
 "[[ Inputs A (Card A)  
 {Agent B, Nonce Na, Nonce Nb, Nonce (Pairkey(A,B)),  
 Cert1, Cert3, Cert2} ∈ set evs;  
 evs ∈ srb ]]  
 ⇒ Nonce (Pairkey(A,B)) ∈ analz (knows Spy evs)"  
 <proof>

**declare** *shrK\_disj\_sesK* [THEN not\_sym, iff]  
**declare** *pin\_disj\_sesK* [THEN not\_sym, iff]  
**declare** *crdK\_disj\_sesK* [THEN not\_sym, iff]  
**declare** *pairK\_disj\_sesK* [THEN not\_sym, iff]

<ML>

**lemma** *Spy\_parts\_keys [simp]*: "evs  $\in$  srb  $\implies$   
 (Key (shrK P)  $\in$  parts (knows Spy evs)) = (Card P  $\in$  cloned)  $\wedge$   
 (Key (pin P)  $\in$  parts (knows Spy evs)) = (P  $\in$  bad  $\vee$  Card P  $\in$  cloned)  $\wedge$   
 (Key (crdK C)  $\in$  parts (knows Spy evs)) = (C  $\in$  cloned)  $\wedge$   
 (Key (pairK(A,B))  $\in$  parts (knows Spy evs)) = (Card B  $\in$  cloned)"  
 <proof>

**lemma** *Spy\_analz\_shrK[simp]*: "evs  $\in$  srb  $\implies$   
 (Key (shrK P)  $\in$  analz (knows Spy evs)) = (Card P  $\in$  cloned)"  
 <proof>

**lemma** *Spy\_analz\_crdK[simp]*: "evs  $\in$  srb  $\implies$   
 (Key (crdK C)  $\in$  analz (knows Spy evs)) = (C  $\in$  cloned)"  
 <proof>

**lemma** *Spy\_analz\_pairK[simp]*: "evs  $\in$  srb  $\implies$   
 (Key (pairK(A,B))  $\in$  analz (knows Spy evs)) = (Card B  $\in$  cloned)"  
 <proof>

**lemma** *analz\_image\_Key\_Un\_Nonce*:  
 "analz (Key ' K  $\cup$  Nonce ' N) = Key ' K  $\cup$  Nonce ' N"  
 <proof>

<ML>

**lemma** *analz\_image\_freshK [rule\_format]*:  
 "evs  $\in$  srb  $\implies \quad \forall K KK.$   
 (Key K  $\in$  analz (Key'KK  $\cup$  (knows Spy evs))) =  
 (K  $\in$  KK  $\vee$  Key K  $\in$  analz (knows Spy evs))"  
 <proof>

**lemma** *analz\_insert\_freshK*: "evs  $\in$  srb  $\implies$   
 Key K  $\in$  analz (insert (Key K') (knows Spy evs)) =  
 (K = K'  $\vee$  Key K  $\in$  analz (knows Spy evs))"  
 <proof>

**lemma** *Na\_Nb\_certificate\_authentic:*

"[[ Crypt (pairK(A,B)) {Nonce Na, Nonce Nb} ∈ parts (knows Spy evs);  
 ¬illegalUse(Card B);  
 evs ∈ srb ]]  
 ⇒ Outputs (Card B) B {Nonce Nb, Agent A, Key (sesK(Nb,pairK(A,B))),

Crypt (pairK(A,B)) {Nonce Na, Nonce Nb},  
 Crypt (pairK(A,B)) (Nonce Nb)} ∈ set evs"

⟨proof⟩

**lemma** *Nb\_certificate\_authentic:*

"[[ Crypt (pairK(A,B)) (Nonce Nb) ∈ parts (knows Spy evs);  
 B ≠ Spy; ¬illegalUse(Card A); ¬illegalUse(Card B);  
 evs ∈ srb ]]  
 ⇒ Outputs (Card A) A {Agent B, Nonce Nb, Key (sesK(Nb,pairK(A,B))),

Crypt (pairK(A,B)) (Nonce Nb)} ∈ set evs"

⟨proof⟩

**lemma** *Outputs\_A\_Card\_imp\_pairK\_parts:*

"[[ Outputs (Card A) A {Agent B, Nonce Nb,  
 Key K, Certificate} ∈ set evs;  
 evs ∈ srb ]]  
 ⇒ ∃ Na. Crypt (pairK(A,B)) {Nonce Na, Nonce Nb} ∈ parts (knows Spy  
 evs)"

⟨proof⟩

**lemma** *Nb\_certificate\_authentic\_bis:*

"[[ Crypt (pairK(A,B)) (Nonce Nb) ∈ parts (knows Spy evs);  
 B ≠ Spy; ¬illegalUse(Card B);  
 evs ∈ srb ]]  
 ⇒ ∃ Na. Outputs (Card B) B {Nonce Nb, Agent A, Key (sesK(Nb,pairK(A,B))),

Crypt (pairK(A,B)) {Nonce Na, Nonce Nb},  
 Crypt (pairK(A,B)) (Nonce Nb)} ∈ set evs"

⟨proof⟩

**lemma** *Pairkey\_certificate\_authentic:*

"[[ Crypt (shrK A) {Nonce Pk, Agent B} ∈ parts (knows Spy evs);  
 Card A ∉ cloned; evs ∈ srb ]]  
 ⇒ Pk = Pairkey(A,B) ∧  
 Says Server A {Nonce Pk,  
 Crypt (shrK A) {Nonce Pk, Agent B}}  
 ∈ set evs"

⟨proof⟩

**lemma** *sesK\_authentic*:

```
"[[ Key (sesK(Nb,pairK(A,B))) ∈ parts (knows Spy evs);
   A ≠ Spy; B ≠ Spy; ¬illegalUse(Card A); ¬illegalUse(Card B);
   evs ∈ srb ]]
⇒ Notes Spy {Key (sesK(Nb,pairK(A,B))), Nonce Nb, Agent A, Agent B}

   ∈ set evs"
⟨proof⟩
```

**lemma** *Confidentiality*:

```
"[[ Notes Spy {Key (sesK(Nb,pairK(A,B))), Nonce Nb, Agent A, Agent B}

   ∉ set evs;
   A ≠ Spy; B ≠ Spy; ¬illegalUse(Card A); ¬illegalUse(Card B);
   evs ∈ srb ]]
⇒ Key (sesK(Nb,pairK(A,B))) ∉ analz (knows Spy evs)"
⟨proof⟩
```

**lemma** *Confidentiality\_B*:

```
"[[ Outpts (Card B) B {Nonce Nb, Agent A, Key K, Cert1, Cert2}
   ∈ set evs;
   Notes Spy {Key K, Nonce Nb, Agent A, Agent B} ∉ set evs;
   A ≠ Spy; B ≠ Spy; ¬illegalUse(Card A); Card B ∉ cloned;
   evs ∈ srb ]]
⇒ Key K ∉ analz (knows Spy evs)"
⟨proof⟩
```

**lemma** *A\_authenticates\_B*:

```
"[[ Outpts (Card A) A {Agent B, Nonce Nb, Key K, Certificate} ∈ set evs;
   ¬illegalUse(Card B);
   evs ∈ srb ]]
⇒ ∃ Na. Outpts (Card B) B {Nonce Nb, Agent A, Key K,
   Crypt (pairK(A,B)) {Nonce Na, Nonce Nb},
   Crypt (pairK(A,B)) (Nonce Nb)} ∈ set evs"
⟨proof⟩
```

**lemma** *A\_authenticates\_B\_Gets*:

```
"[[ Gets A {Nonce Nb, Crypt (pairK(A,B)) {Nonce Na, Nonce Nb}}
   ∈ set evs;
```

$\neg \text{illegalUse}(\text{Card } B);$   
 $\text{evs} \in \text{srb} \parallel$   
 $\implies \text{Outpts}(\text{Card } B) B \{ \text{Nonce } Nb, \text{Agent } A, \text{Key}(\text{sesK}(Nb, \text{pairK}(A, B))) \},$   
 $\text{Crypt}(\text{pairK}(A, B)) \{ \text{Nonce } Na, \text{Nonce } Nb \},$   
 $\text{Crypt}(\text{pairK}(A, B)) (\text{Nonce } Nb) \in \text{set evs}"$   
 $\langle \text{proof} \rangle$

**lemma** *A\_authenticates\_B\_bis*:  
 $" \parallel \text{Outpts}(\text{Card } A) A \{ \text{Agent } B, \text{Nonce } Nb, \text{Key } K, \text{Cert2} \} \in \text{set evs};$   
 $\neg \text{illegalUse}(\text{Card } B);$   
 $\text{evs} \in \text{srb} \parallel$   
 $\implies \exists \text{Cert1}. \text{Outpts}(\text{Card } B) B \{ \text{Nonce } Nb, \text{Agent } A, \text{Key } K, \text{Cert1}, \text{Cert2} \}$   
 $\in \text{set evs}"$   
 $\langle \text{proof} \rangle$

**lemma** *B\_authenticates\_A*:  
 $" \parallel \text{Gets } B (\text{Crypt}(\text{pairK}(A, B)) (\text{Nonce } Nb)) \in \text{set evs};$   
 $B \neq \text{Spy}; \neg \text{illegalUse}(\text{Card } A); \neg \text{illegalUse}(\text{Card } B);$   
 $\text{evs} \in \text{srb} \parallel$   
 $\implies \text{Outpts}(\text{Card } A) A \{ \text{Agent } B, \text{Nonce } Nb,$   
 $\text{Key}(\text{sesK}(Nb, \text{pairK}(A, B))), \text{Crypt}(\text{pairK}(A, B)) (\text{Nonce } Nb) \} \in \text{set evs}"$   
 $\langle \text{proof} \rangle$

**lemma** *B\_authenticates\_A\_bis*:  
 $" \parallel \text{Outpts}(\text{Card } B) B \{ \text{Nonce } Nb, \text{Agent } A, \text{Key } K, \text{Cert1}, \text{Cert2} \} \in \text{set evs};$   
 $\text{Gets } B (\text{Cert2}) \in \text{set evs};$   
 $B \neq \text{Spy}; \neg \text{illegalUse}(\text{Card } A); \neg \text{illegalUse}(\text{Card } B);$   
 $\text{evs} \in \text{srb} \parallel$   
 $\implies \text{Outpts}(\text{Card } A) A \{ \text{Agent } B, \text{Nonce } Nb, \text{Key } K, \text{Cert2} \} \in \text{set evs}"$   
 $\langle \text{proof} \rangle$

**lemma** *Confidentiality\_A*:  
 $" \parallel \text{Outpts}(\text{Card } A) A \{ \text{Agent } B, \text{Nonce } Nb,$   
 $\text{Key } K, \text{Certificate} \} \in \text{set evs};$   
 $\text{Notes Spy} \{ \text{Key } K, \text{Nonce } Nb, \text{Agent } A, \text{Agent } B \} \notin \text{set evs};$   
 $A \neq \text{Spy}; B \neq \text{Spy}; \neg \text{illegalUse}(\text{Card } A); \neg \text{illegalUse}(\text{Card } B);$   
 $\text{evs} \in \text{srb} \parallel$   
 $\implies \text{Key } K \notin \text{analz}(\text{knows Spy evs})"$   
 $\langle \text{proof} \rangle$



**lemma** *Outpts\_imp\_knows\_agents\_secureM\_srb:*

"[[ Outpts (Card A) A X ∈ set evs; evs ∈ srb ]] ⇒ X ∈ knows A evs"  
 <proof>

**lemma** *A\_keydist\_to\_B:*

"[[ Outpts (Card A) A {Agent B, Nonce Nb, Key K, Certificate} ∈ set evs;  
     ¬illegalUse(Card B);  
     evs ∈ srb ]]  
 ⇒ Key K ∈ analz (knows B evs)"  
 <proof>

**lemma** *B\_keydist\_to\_A:*

"[[ Outpts (Card B) B {Nonce Nb, Agent A, Key K, Cert1, Cert2} ∈ set evs;  
     Gets B (Cert2) ∈ set evs;  
     B ≠ Spy; ¬illegalUse(Card A); ¬illegalUse(Card B);  
     evs ∈ srb ]]  
 ⇒ Key K ∈ analz (knows A evs)"  
 <proof>

**lemma** *Nb\_certificate\_authentic\_B:*

"[[ Gets B (Crypt (pairK(A,B)) (Nonce Nb)) ∈ set evs;  
     B ≠ Spy; ¬illegalUse(Card B);  
     evs ∈ srb ]]  
 ⇒ ∃ Na.  
     Outpts (Card B) B {Nonce Nb, Agent A, Key (sesK(Nb,pairK(A,B))),  
         Crypt (pairK(A,B)) {Nonce Na, Nonce Nb},  
         Crypt (pairK(A,B)) (Nonce Nb)} ∈ set evs"  
 <proof>

**lemma** *Pairkey\_certificate\_authentic\_A\_Card:*

"[[ Inputs A (Card A)  
     {Agent B, Nonce Na, Nonce Nb, Nonce Pk,  
     Crypt (shrK A) {Nonce Pk, Agent B},  
     Cert2, Cert3} ∈ set evs;  
     A ≠ Spy; Card A ∉ cloned; evs ∈ srb ]]  
 ⇒ Pk = Pairkey(A,B) ∧

Says Server A  $\{\text{Nonce } (\text{Pairkey}(A,B)),$   
 $\text{Crypt } (\text{shrK } A) \{\text{Nonce } (\text{Pairkey}(A,B)), \text{Agent } B\}\}$   
 $\in \text{set evs "}$   
 $\langle \text{proof} \rangle$

**lemma** Na\_Nb\_certificate\_authentic\_A\_Card:  
 $"\llbracket \text{Inputs } A \text{ (Card } A)$   
 $\{\text{Agent } B, \text{Nonce } Na, \text{Nonce } Nb, \text{Nonce } Pk,$   
 $\text{Cert1, Crypt } (\text{pairK}(A,B)) \{\text{Nonce } Na, \text{Nonce } Nb\}, \text{Cert3}\} \in \text{set evs};$   
 $A \neq \text{Spy}; \neg \text{illegalUse}(\text{Card } B); \text{evs} \in \text{srb} \rrbracket$   
 $\implies \text{Outputs } (\text{Card } B) B \{\text{Nonce } Nb, \text{Agent } A, \text{Key } (\text{sesK}(Nb, \text{pairK } (A, B))),$   
 $\text{Crypt } (\text{pairK}(A,B)) \{\text{Nonce } Na, \text{Nonce } Nb\},$   
 $\text{Crypt } (\text{pairK}(A,B)) (\text{Nonce } Nb)\}$   
 $\in \text{set evs "}$   
 $\langle \text{proof} \rangle$

**lemma** Na\_authentic\_A\_Card:  
 $"\llbracket \text{Inputs } A \text{ (Card } A)$   
 $\{\text{Agent } B, \text{Nonce } Na, \text{Nonce } Nb, \text{Nonce } Pk,$   
 $\text{Cert1, Cert2, Cert3}\} \in \text{set evs};$   
 $A \neq \text{Spy}; \text{evs} \in \text{srb} \rrbracket$   
 $\implies \text{Outputs } (\text{Card } A) A \{\text{Nonce } Na, \text{Cert3}\}$   
 $\in \text{set evs "}$   
 $\langle \text{proof} \rangle$

**lemma** Inputs\_A\_Card\_9\_authentic:  
 $"\llbracket \text{Inputs } A \text{ (Card } A)$   
 $\{\text{Agent } B, \text{Nonce } Na, \text{Nonce } Nb, \text{Nonce } Pk,$   
 $\text{Crypt } (\text{shrK } A) \{\text{Nonce } Pk, \text{Agent } B\},$   
 $\text{Crypt } (\text{pairK}(A,B)) \{\text{Nonce } Na, \text{Nonce } Nb\}, \text{Cert3}\} \in \text{set evs};$   
 $A \neq \text{Spy}; \text{Card } A \notin \text{cloned}; \neg \text{illegalUse}(\text{Card } B); \text{evs} \in \text{srb} \rrbracket$   
 $\implies \text{Says Server } A \{\text{Nonce } Pk, \text{Crypt } (\text{shrK } A) \{\text{Nonce } Pk, \text{Agent } B\}\}$   
 $\in \text{set evs} \wedge$   
 $\text{Outputs } (\text{Card } B) B \{\text{Nonce } Nb, \text{Agent } A, \text{Key } (\text{sesK}(Nb, \text{pairK } (A, B))),$   
 $\text{Crypt } (\text{pairK}(A,B)) \{\text{Nonce } Na, \text{Nonce } Nb\},$   
 $\text{Crypt } (\text{pairK}(A,B)) (\text{Nonce } Nb)\}$   
 $\in \text{set evs} \wedge$   
 $\text{Outputs } (\text{Card } A) A \{\text{Nonce } Na, \text{Cert3}\}$   
 $\in \text{set evs "}$   
 $\langle \text{proof} \rangle$

**lemma** *SR\_U4\_imp*:  
 "[ Outpts (Card A) A {Nonce Na, Crypt (crdK (Card A)) (Nonce Na)}  
   ∈ set evs;  
   A ≠ Spy; evs ∈ srb ]  
 ⇒ ∃ Pk V. Gets A {Pk, V} ∈ set evs"  
 <proof>

**lemma** *SR\_U7\_imp*:  
 "[ Outpts (Card B) B {Nonce Nb, Agent A, Key K,  
                       Crypt (pairK(A,B)) {Nonce Na, Nonce Nb},  
                       Cert2} ∈ set evs;  
   B ≠ Spy; evs ∈ srb ]  
 ⇒ Gets B {Agent A, Nonce Na} ∈ set evs"  
 <proof>

**lemma** *SR\_U10\_imp*:  
 "[ Outpts (Card A) A {Agent B, Nonce Nb,  
                       Key K, Crypt (pairK(A,B)) (Nonce Nb)}  
   ∈ set evs;  
   A ≠ Spy; evs ∈ srb ]  
 ⇒ ∃ Cert1 Cert2.  
      Gets A {Nonce (Pairkey (A, B)), Cert1} ∈ set evs ∧  
      Gets A {Nonce Nb, Cert2} ∈ set evs"  
 <proof>

**lemma** *Outpts\_Server\_not\_evs*:  
 "evs ∈ srb ⇒ Outpts (Card Server) P X ∉ set evs"  
 <proof>

*step2\_integrity* also is a reliability theorem

**lemma** *Says\_Server\_message\_form*:  
 "[ Says Server A {Pk, Certificate} ∈ set evs;  
   evs ∈ srb ]  
 ⇒ ∃ B. Pk = Nonce (Pairkey(A,B)) ∧  
      Certificate = Crypt (shrK A) {Nonce (Pairkey(A,B)), Agent B}"  
 <proof>

step4integrity is *Outpts\_A\_Card\_form\_4*

step7integrity is *Outpts\_B\_Card\_form\_7*

```
lemma step8_integrity:
  "[[ Says B A {Nonce Nb, Certificate} ∈ set evs;
    B ≠ Server; B ≠ Spy; evs ∈ srb ]]
  ⇒ ∃ Cert2 K.
    Outpts (Card B) B {Nonce Nb, Agent A, Key K, Certificate, Cert2} ∈ set
  evs"
  <proof>
```

step9integrity is *Inputs\_A\_Card\_form\_9* step10integrity is *Outpts\_A\_Card\_form\_10*.

```
lemma step11_integrity:
  "[[ Says A B (Certificate) ∈ set evs;
    ∀ p q. Certificate ≠ {p, q};
    A ≠ Spy; evs ∈ srb ]]
  ⇒ ∃ K Nb.
    Outpts (Card A) A {Agent B, Nonce Nb, Key K, Certificate} ∈ set evs"
  <proof>
```

end

## 28 Smartcard protocols: rely on conventional Message and on new EventSC and Smartcard

```
theory Auth_Smartcard
imports
  ShoupRubin
  ShoupRubinBella
begin

end
```

## 29 Extensions to Standard Theories

```
theory Extensions
imports "../Event"
begin
```

### 29.1 Extensions to Theory Set

```
lemma eq: "[[ ∧x. x∈A ⇒ x∈B; ∧x. x∈B ⇒ x∈A ]] ⇒ A=B"
  <proof>
```

```
lemma insert_Un: "P ({x} ∪ A) ⇒ P (insert x A)"
  <proof>
```

```
lemma in_sub: "x∈A ⇒ {x}⊆A"
  <proof>
```

## 29.2 Extensions to Theory List

### 29.2.1 "remove l x" erase the first element of "l" equal to "x"

```

primrec remove :: "'a list => 'a => 'a list" where
  "remove [] y = []" |
  "remove (x#xs) y = (if x=y then xs else x # remove xs y)"

lemma set_remove: "set (remove l x) <= set l"
  <proof>

```

## 29.3 Extensions to Theory Message

### 29.3.1 declarations for tactics

```

declare analz_subset_parts [THEN subsetD, dest]
declare parts_insert2 [simp]
declare analz_cut [dest]
declare if_split_asm [split]
declare analz_insertI [intro]
declare Un_Diff [simp]

```

### 29.3.2 extract the agent number of an Agent message

```

primrec agt_nb :: "msg => agent" where
  "agt_nb (Agent A) = A"

```

### 29.3.3 messages that are pairs

```

definition is_MPair :: "msg => bool" where
  "is_MPair X ==  $\exists Y Z. X = \langle Y, Z \rangle$ "

```

```

declare is_MPair_def [simp]

```

```

lemma MPair_is_MPair [iff]: "is_MPair  $\langle X, Y \rangle$ "
  <proof>

```

```

lemma Agent_isnt_MPair [iff]: "~ is_MPair (Agent A)"
  <proof>

```

```

lemma Number_isnt_MPair [iff]: "~ is_MPair (Number n)"
  <proof>

```

```

lemma Key_isnt_MPair [iff]: "~ is_MPair (Key K)"
  <proof>

```

```

lemma Nonce_isnt_MPair [iff]: "~ is_MPair (Nonce n)"
  <proof>

```

```

lemma Hash_isnt_MPair [iff]: "~ is_MPair (Hash X)"
  <proof>

```

```

lemma Crypt_isnt_MPair [iff]: "~ is_MPair (Crypt K X)"
  <proof>

```

**abbreviation**

```

not_MPair :: "msg => bool" where
  "not_MPair X == ~ is_MPair X"

```

```

lemma is_MPairE: "[is_MPair X ==> P; not_MPair X ==> P] ==> P"
<proof>

```

```

declare is_MPair_def [simp del]

```

```

definition has_no_pair :: "msg set => bool" where
  "has_no_pair H ==  $\forall X Y. \{X, Y\} \notin H$ "

```

```

declare has_no_pair_def [simp]

```

### 29.3.4 well-foundedness of messages

```

lemma wf_Crypt1 [iff]: "Crypt K X ~= X"
<proof>

```

```

lemma wf_Crypt2 [iff]: "X ~= Crypt K X"
<proof>

```

```

lemma parts_size: "X  $\in$  parts {Y} ==> X=Y  $\vee$  size X < size Y"
<proof>

```

```

lemma wf_Crypt_parts [iff]: "Crypt K X  $\notin$  parts {X}"
<proof>

```

### 29.3.5 lemmas on keysFor

```

definition usekeys :: "msg set => key set" where
  "usekeys G  $\equiv$  {K.  $\exists Y. \text{Crypt } K Y \in G$ }"

```

```

lemma finite_keysFor [intro]: "finite G ==> finite (keysFor G)"
<proof>

```

### 29.3.6 lemmas on parts

```

lemma parts_sub: "[X  $\in$  parts G; G  $\subseteq$  H] ==> X  $\in$  parts H"
<proof>

```

```

lemma parts_Diff [dest]: "X  $\in$  parts (G - H) ==> X  $\in$  parts G"
<proof>

```

```

lemma parts_Diff_notin: "[Y  $\notin$  H; Nonce n  $\notin$  parts (H - {Y})]
==> Nonce n  $\notin$  parts H"
<proof>

```

```

lemmas parts_insert_substI = parts_insert [THEN ssubst]
lemmas parts_insert_substD = parts_insert [THEN sym, THEN ssubst]

```

```

lemma finite_parts_msg [iff]: "finite (parts {X})"
<proof>

```

```

lemma finite_parts [intro]: "finite H ==> finite (parts H)"
<proof>

```

**lemma** parts\_parts: " $\llbracket X \in \text{parts } \{Y\}; Y \in \text{parts } G \rrbracket \implies X \in \text{parts } G$ "  
 <proof>

**lemma** parts\_parts\_parts: " $\llbracket X \in \text{parts } \{Y\}; Y \in \text{parts } \{Z\}; Z \in \text{parts } G \rrbracket \implies X \in \text{parts } G$ "  
 <proof>

**lemma** parts\_parts\_Crypt: " $\llbracket \text{Crypt } K X \in \text{parts } G; \text{Nonce } n \in \text{parts } \{X\} \rrbracket \implies \text{Nonce } n \in \text{parts } G$ "  
 <proof>

### 29.3.7 lemmas on synth

**lemma** synth\_sub: " $\llbracket X \in \text{synth } G; G \subseteq H \rrbracket \implies X \in \text{synth } H$ "  
 <proof>

**lemma** Crypt\_synth [rule\_format]: " $\llbracket X \in \text{synth } G; \text{Key } K \notin G \rrbracket \implies \text{Crypt } K Y \in \text{parts } \{X\} \longrightarrow \text{Crypt } K Y \in \text{parts } G$ "  
 <proof>

### 29.3.8 lemmas on analz

**lemma** analz\_UnI1 [intro]: " $X \in \text{analz } G \implies X \in \text{analz } (G \cup H)$ "  
 <proof>

**lemma** analz\_sub: " $\llbracket X \in \text{analz } G; G \subseteq H \rrbracket \implies X \in \text{analz } H$ "  
 <proof>

**lemma** analz\_Diff [dest]: " $X \in \text{analz } (G - H) \implies X \in \text{analz } G$ "  
 <proof>

**lemmas** in\_analz\_subset\_cong = analz\_subset\_cong [THEN subsetD]

**lemma** analz\_eq: " $A=A' \implies \text{analz } A = \text{analz } A'$ "  
 <proof>

**lemmas** insert\_commute\_substI = insert\_commute [THEN ssubst]

**lemma** analz\_insertD:  
 " $\llbracket \text{Crypt } K Y \in H; \text{Key } (\text{invKey } K) \in H \rrbracket \implies \text{analz } (\text{insert } Y H) = \text{analz } H$ "  
 <proof>

**lemma** must\_decrypt [rule\_format,dest]: " $\llbracket X \in \text{analz } H; \text{has\_no\_pair } H \rrbracket \implies X \notin H \longrightarrow (\exists K Y. \text{Crypt } K Y \in H \wedge \text{Key } (\text{invKey } K) \in H)$ "  
 <proof>

**lemma** analz\_needs\_only\_finite: " $X \in \text{analz } H \implies \exists G. G \subseteq H \wedge \text{finite } G$ "  
 <proof>

**lemma** notin\_analz\_insert: " $X \notin \text{analz } (\text{insert } Y G) \implies X \notin \text{analz } G$ "  
 <proof>

### 29.3.9 lemmas on parts, synth and analz

**lemma** *parts\_invKey* [rule\_format,dest]: " $X \in \text{parts } \{Y\} \implies X \in \text{analz } (\text{insert } (\text{Crypt } K \ Y) \ H) \longrightarrow X \notin \text{analz } H \longrightarrow \text{Key } (\text{invKey } K) \in \text{analz } H$ "  
 <proof>

**lemma** *in\_analz*: " $Y \in \text{analz } H \implies \exists X. X \in H \wedge Y \in \text{parts } \{X\}$ "  
 <proof>

**lemmas** *in\_analz\_subset\_parts* = *analz\_subset\_parts* [THEN subsetD]

**lemma** *Crypt\_synth\_insert*: " $\llbracket \text{Crypt } K \ X \in \text{parts } (\text{insert } Y \ H); Y \in \text{synth } (\text{analz } H); \text{Key } K \notin \text{analz } H \rrbracket \implies \text{Crypt } K \ X \in \text{parts } H$ "  
 <proof>

### 29.3.10 greatest nonce used in a message

**fun** *greatest\_msg* :: "msg => nat"  
**where**  
   "*greatest\_msg* (Nonce *n*) = *n*"  
   | "*greatest\_msg*  $\{X, Y\}$  = max (*greatest\_msg* *X*) (*greatest\_msg* *Y*)"  
   | "*greatest\_msg* (Crypt *K* *X*) = *greatest\_msg* *X*"  
   | "*greatest\_msg* *other* = 0"

**lemma** *greatest\_msg\_is\_greatest*: "Nonce *n*  $\in \text{parts } \{X\} \implies n \leq \text{greatest\_msg } X$ "  
 <proof>

### 29.3.11 sets of keys

**definition** *keyset* :: "msg set => bool" **where**  
 "*keyset* *G*  $\equiv \forall X. X \in G \longrightarrow (\exists K. X = \text{Key } K)$ "

**lemma** *keyset\_in* [dest]: " $\llbracket \text{keyset } G; X \in G \rrbracket \implies \exists K. X = \text{Key } K$ "  
 <proof>

**lemma** *MPair\_notin\_keyset* [simp]: "*keyset* *G*  $\implies \{X, Y\} \notin G$ "  
 <proof>

**lemma** *Crypt\_notin\_keyset* [simp]: "*keyset* *G*  $\implies \text{Crypt } K \ X \notin G$ "  
 <proof>

**lemma** *Nonce\_notin\_keyset* [simp]: "*keyset* *G*  $\implies \text{Nonce } n \notin G$ "  
 <proof>

**lemma** *parts\_keyset* [simp]: "*keyset* *G*  $\implies \text{parts } G = G$ "  
 <proof>

### 29.3.12 keys a priori necessary for decrypting the messages of G

**definition** *keysfor* :: "msg set => msg set" **where**  
 "*keysfor* *G* == Key ' *keysFor* (parts *G*)"

**lemma** *keyset\_keysfor* [iff]: "*keyset* (*keysfor* *G*)"



*<proof>*

**lemma** *keyset\_Diff\_keysfor [simp]: "keyset H  $\implies$  keyset (H - keysfor G)"*  
*<proof>*

**lemma** *keysfor\_Crypt: "Crypt K X  $\in$  parts G  $\implies$  Key (invKey K)  $\in$  keysfor G"*  
*<proof>*

**lemma** *no\_key\_no\_Crypt: "Key K  $\notin$  keysfor G  $\implies$  Crypt (invKey K) X  $\notin$  parts G"*  
*<proof>*

**lemma** *finite\_keysfor [intro]: "finite G  $\implies$  finite (keysfor G)"*  
*<proof>*

### 29.3.13 only the keys necessary for G are useful in analz

**lemma** *analz\_keyset: "keyset H  $\implies$   
 analz (G Un H) = H - keysfor G Un (analz (G Un (H Int keysfor G)))"*  
*<proof>*

**lemmas** *analz\_keyset\_substD = analz\_keyset [THEN sym, THEN ssubst]*

## 29.4 Extensions to Theory Event

### 29.4.1 general protocol properties

**definition** *is\_Says :: "event  $\Rightarrow$  bool" where*  
*"is\_Says ev == ( $\exists$  A B X. ev = Says A B X)"*

**lemma** *is\_Says\_Says [iff]: "is\_Says (Says A B X)"*  
*<proof>*

**definition** *Gets\_correct :: "event list set  $\Rightarrow$  bool" where*  
*"Gets\_correct p ==  $\forall$  evs B X. evs  $\in$  p  $\longrightarrow$  Gets B X  $\in$  set evs*  
 *$\longrightarrow$  ( $\exists$  A. Says A B X  $\in$  set evs)"*

**lemma** *Gets\_correct\_Says: "[[Gets\_correct p; Gets B X # evs  $\in$  p]]*  
 *$\implies \exists$  A. Says A B X  $\in$  set evs"*  
*<proof>*

**definition** *one\_step :: "event list set  $\Rightarrow$  bool" where*  
*"one\_step p ==  $\forall$  evs ev. ev#evs  $\in$  p  $\longrightarrow$  evs  $\in$  p"*

**lemma** *one\_step\_Cons [dest]: "[one\_step p; ev#evs  $\in$  p]  $\implies$  evs  $\in$  p"*  
*<proof>*

**lemma** *one\_step\_app: "[[evs@evs'  $\in$  p; one\_step p; []  $\in$  p]]  $\implies$  evs'  $\in$  p"*  
*<proof>*

**lemma** *trunc: "[[evs @ evs'  $\in$  p; one\_step p]]  $\implies$  evs'  $\in$  p"*  
*<proof>*

**definition** *has\_only\_Says :: "event list set  $\Rightarrow$  bool" where*

"has\_only\_Says p  $\equiv \forall \text{ evs } \text{ev}. \text{ evs} \in p \longrightarrow \text{ev} \in \text{set evs}$   
 $\longrightarrow (\exists A B X. \text{ ev} = \text{Says } A B X)$ "

**lemma** has\_only\_SaysD: "[[ev  $\in$  set evs; evs  $\in$  p; has\_only\_Says p]]  
 $\implies \exists A B X. \text{ ev} = \text{Says } A B X$ "  
 <proof>

**lemma** in\_has\_only\_Says [dest]: "[[has\_only\_Says p; evs  $\in$  p; ev  $\in$  set evs]]  
 $\implies \exists A B X. \text{ ev} = \text{Says } A B X$ "  
 <proof>

**lemma** has\_only\_Says\_imp\_Gets\_correct [simp]: "has\_only\_Says p  
 $\implies \text{Gets\_correct } p$ "  
 <proof>

#### 29.4.2 lemma on knows

**lemma** Says\_imp\_spies2: "Says A B {X,Y}  $\in$  set evs  $\implies Y \in \text{parts (spies evs)}$ "  
 <proof>

**lemma** Says\_not\_parts: "[[Says A B X  $\in$  set evs; Y  $\notin$  parts (spies evs)]]  
 $\implies Y \notin \text{parts } \{X\}$ "  
 <proof>

#### 29.4.3 knows without initState

**primrec** knows' :: "agent  $\Rightarrow$  event list  $\Rightarrow$  msg set" where  
 knows'\_Nil: "knows' A [] = {}" |  
 knows'\_Cons0:  
 "knows' A (ev # evs) = (  
 if A = Spy then (  
 case ev of  
 Says A' B X  $\Rightarrow$  insert X (knows' A evs)  
 | Gets A' X  $\Rightarrow$  knows' A evs  
 | Notes A' X  $\Rightarrow$  if A'  $\in$  bad then insert X (knows' A evs) else knows'  
 A evs  
 ) else (  
 case ev of  
 Says A' B X  $\Rightarrow$  if A=A' then insert X (knows' A evs) else knows' A evs  
 | Gets A' X  $\Rightarrow$  if A=A' then insert X (knows' A evs) else knows' A evs  
 | Notes A' X  $\Rightarrow$  if A=A' then insert X (knows' A evs) else knows' A evs  
 ))"

#### abbreviation

spies' :: "event list  $\Rightarrow$  msg set" where  
 "spies' == knows' Spy"

#### 29.4.4 decomposition of knows into knows' and initState

**lemma** knows\_decomp: "knows A evs = knows' A evs Un (initState A)"  
 <proof>

**lemmas** knows\_decomp\_substI = knows\_decomp [THEN ssubst]  
 knows\_decomp\_substD = knows\_decomp [THEN sym, THEN ssubst]

**lemma** knows'\_sub\_knows: "knows' A evs <= knows A evs"  
 <proof>

**lemma** knows'\_Cons: "knows' A (ev#evs) = knows' A [ev] Un knows' A evs"  
 <proof>

**lemmas** knows'\_Cons\_substI = knows'\_Cons [THEN ssubst]  
**lemmas** knows'\_Cons\_substD = knows'\_Cons [THEN sym, THEN ssubst]

**lemma** knows\_Cons: "knows A (ev#evs) = initState A Un knows' A [ev]  
 Un knows A evs"  
 <proof>

**lemmas** knows\_Cons\_substI = knows\_Cons [THEN ssubst]  
**lemmas** knows\_Cons\_substD = knows\_Cons [THEN sym, THEN ssubst]

**lemma** knows'\_sub\_spies': "[evs ∈ p; has\_only\_Says p; one\_step p]  
 ⇒ knows' A evs ⊆ spies' evs"  
 <proof>

#### 29.4.5 knows' is finite

**lemma** finite\_knows' [iff]: "finite (knows' A evs)"  
 <proof>

#### 29.4.6 monotonicity of knows

**lemma** knows\_sub\_Cons: "knows A evs <= knows A (ev#evs)"  
 <proof>

**lemma** knows\_ConsI: "X ∈ knows A evs ⇒ X ∈ knows A (ev#evs)"  
 <proof>

**lemma** knows\_sub\_app: "knows A evs <= knows A (evs @ evs')"  
 <proof>

#### 29.4.7 maximum knowledge an agent can have includes messages sent to the agent

**primrec** knows\_max' :: "agent => event list => msg set" where  
 knows\_max'\_def\_Nil: "knows\_max' A [] = {}" |  
 knows\_max'\_def\_Cons: "knows\_max' A (ev # evs) = (  
 if A=Spy then (  
 case ev of  
 Says A' B X => insert X (knows\_max' A evs)  
 | Gets A' X => knows\_max' A evs  
 | Notes A' X =>  
 if A' ∈ bad then insert X (knows\_max' A evs) else knows\_max' A evs  
 ) else (  
 case ev of  
 Says A' B X =>  
 if A=A' | A=B then insert X (knows\_max' A evs) else knows\_max' A evs  
 | Gets A' X =>  
 if A=A' then insert X (knows\_max' A evs) else knows\_max' A evs

```

    | Notes A' X =>
      if A=A' then insert X (knows_max' A evs) else knows_max' A evs
  ))"

```

**definition** knows\_max :: "agent => event list => msg set" where  
 "knows\_max A evs == knows\_max' A evs Un initState A"

**abbreviation**

```

  spies_max :: "event list => msg set" where
  "spies_max evs == knows_max Spy evs"

```

#### 29.4.8 basic facts about knows\_max

**lemma** spies\_max\_spies [iff]: "spies\_max evs = spies evs"  
 <proof>

**lemma** knows\_max'\_Cons: "knows\_max' A (ev#evs)  
 = knows\_max' A [ev] Un knows\_max' A evs"  
 <proof>

**lemmas** knows\_max'\_Cons\_substI = knows\_max'\_Cons [THEN ssubst]  
 knows\_max'\_Cons\_substD = knows\_max'\_Cons [THEN sym, THEN ssubst]

**lemma** knows\_max\_Cons: "knows\_max A (ev#evs)  
 = knows\_max' A [ev] Un knows\_max A evs"  
 <proof>

**lemmas** knows\_max\_Cons\_substI = knows\_max\_Cons [THEN ssubst]  
 knows\_max\_Cons\_substD = knows\_max\_Cons [THEN sym, THEN ssubst]

**lemma** finite\_knows\_max' [iff]: "finite (knows\_max' A evs)"  
 <proof>

**lemma** knows\_max'\_sub\_spies': "[evs ∈ p; has\_only\_Says p; one\_step p]  
 ⇒ knows\_max' A evs ⊆ spies' evs"  
 <proof>

**lemma** knows\_max'\_in\_spies' [dest]: "[evs ∈ p; X ∈ knows\_max' A evs;  
 has\_only\_Says p; one\_step p] ⇒ X ∈ spies' evs"  
 <proof>

**lemma** knows\_max'\_app: "knows\_max' A (evs @ evs')  
 = knows\_max' A evs Un knows\_max' A evs'"  
 <proof>

**lemma** Says\_to\_knows\_max': "Says A B X ∈ set evs ⇒ X ∈ knows\_max' B evs"  
 <proof>

**lemma** Says\_from\_knows\_max': "Says A B X ∈ set evs ⇒ X ∈ knows\_max' A evs"  
 <proof>

#### 29.4.9 used without initState

**primrec** used' :: "event list => msg set" where  
 "used' [] = {}" |

```

"used' (ev # evs) = (
  case ev of
    Says A B X => parts {X} Un used' evs
  | Gets A X => used' evs
  | Notes A X => parts {X} Un used' evs
)"

```

```

definition init :: "msg set" where
  "init == used []"

```

```

lemma used_decomp: "used evs = init Un used' evs"
<proof>

```

```

lemma used'_sub_app: "used' evs  $\subseteq$  used' (evs@evs')"
<proof>

```

```

lemma used'_parts [rule_format]: "X  $\in$  used' evs  $\implies$  Y  $\in$  parts {X}  $\longrightarrow$  Y
 $\in$  used' evs"
<proof>

```

#### 29.4.10 monotonicity of used

```

lemma used_sub_Cons: "used evs  $\leq$  used (ev#evs)"
<proof>

```

```

lemma used_ConsI: "X  $\in$  used evs  $\implies$  X  $\in$  used (ev#evs)"
<proof>

```

```

lemma notin_used_ConsD: "X  $\notin$  used (ev#evs)  $\implies$  X  $\notin$  used evs"
<proof>

```

```

lemma used_appD [dest]: "X  $\in$  used (evs @ evs')  $\implies$  X  $\in$  used evs  $\vee$  X  $\in$  used
evs'"
<proof>

```

```

lemma used_ConsD: "X  $\in$  used (ev#evs)  $\implies$  X  $\in$  used [ev]  $\vee$  X  $\in$  used evs"
<proof>

```

```

lemma used_sub_app: "used evs  $\leq$  used (evs@evs')"
<proof>

```

```

lemma used_appIL: "X  $\in$  used evs  $\implies$  X  $\in$  used (evs' @ evs)"
<proof>

```

```

lemma used_appIR: "X  $\in$  used evs  $\implies$  X  $\in$  used (evs @ evs')"
<proof>

```

```

lemma used_parts: "[X  $\in$  parts {Y}; Y  $\in$  used evs]  $\implies$  X  $\in$  used evs"
<proof>

```

```

lemma parts_Says_used: "[Says A B X  $\in$  set evs; Y  $\in$  parts {X}]  $\implies$  Y  $\in$  used
evs"
<proof>

```

**lemma** *parts\_used\_app*: " $X \in \text{parts } \{Y\} \implies X \in \text{used } (\text{evs } @ \text{ Says } A \ B \ Y \ \# \ \text{evs}')$ "  
 $\langle \text{proof} \rangle$

#### 29.4.11 lemmas on used and knows

**lemma** *initState\_used*: " $X \in \text{parts } (\text{initState } A) \implies X \in \text{used evs}$ "  
 $\langle \text{proof} \rangle$

**lemma** *Says\_imp\_used*: " $\text{Says } A \ B \ X \in \text{set evs} \implies \text{parts } \{X\} \subseteq \text{used evs}$ "  
 $\langle \text{proof} \rangle$

**lemma** *not\_used\_not\_spied*: " $X \notin \text{used evs} \implies X \notin \text{parts } (\text{spies evs})$ "  
 $\langle \text{proof} \rangle$

**lemma** *not\_used\_not\_parts*: " $\llbracket Y \notin \text{used evs}; \text{Says } A \ B \ X \in \text{set evs} \rrbracket \implies Y \notin \text{parts } \{X\}$ "  
 $\langle \text{proof} \rangle$

**lemma** *not\_used\_parts\_false*: " $\llbracket X \notin \text{used evs}; Y \in \text{parts } (\text{spies evs}) \rrbracket \implies X \notin \text{parts } \{Y\}$ "  
 $\langle \text{proof} \rangle$

**lemma** *known\_used [rule\_format]*: " $\llbracket \text{evs} \in p; \text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies X \in \text{parts } (\text{knows } A \ \text{evs}) \longrightarrow X \in \text{used evs}$ "  
 $\langle \text{proof} \rangle$

**lemma** *known\_max\_used [rule\_format]*: " $\llbracket \text{evs} \in p; \text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies X \in \text{parts } (\text{knows\_max } A \ \text{evs}) \longrightarrow X \in \text{used evs}$ "  
 $\langle \text{proof} \rangle$

**lemma** *not\_used\_not\_known*: " $\llbracket \text{evs} \in p; X \notin \text{used evs}; \text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies X \notin \text{parts } (\text{knows } A \ \text{evs})$ "  
 $\langle \text{proof} \rangle$

**lemma** *not\_used\_not\_known\_max*: " $\llbracket \text{evs} \in p; X \notin \text{used evs}; \text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies X \notin \text{parts } (\text{knows\_max } A \ \text{evs})$ "  
 $\langle \text{proof} \rangle$

#### 29.4.12 a nonce or key in a message cannot equal a fresh nonce or key

**lemma** *Nonce\_neq [dest]*: " $\llbracket \text{Nonce } n' \notin \text{used evs}; \text{Says } A \ B \ X \in \text{set evs}; \text{Nonce } n \in \text{parts } \{X\} \rrbracket \implies n \neq n'$ "  
 $\langle \text{proof} \rangle$

**lemma** *Key\_neq [dest]*: " $\llbracket \text{Key } n' \notin \text{used evs}; \text{Says } A \ B \ X \in \text{set evs}; \text{Key } n \in \text{parts } \{X\} \rrbracket \implies n \neq n'$ "  
 $\langle \text{proof} \rangle$

#### 29.4.13 message of an event

**primrec** *msg* :: "event  $\Rightarrow$  msg"  
**where**

"*msg* (Says A B X) = X"

```
| "msg (Gets A X) = X"
| "msg (Notes A X) = X"
```

```
lemma used_sub_parts_used: "X ∈ used (ev # evs) ⇒ X ∈ parts {msg ev} ∪
used evs"
⟨proof⟩
```

```
end
```

## 30 Decomposition of Analz into two parts

```
theory Analz imports Extensions begin
```

decomposition of *analz* into two parts: *pparts* (for pairs) and *analz* of *kparts*

### 30.1 messages that do not contribute to analz

```
inductive_set
  pparts :: "msg set => msg set"
  for H :: "msg set"
where
  Inj [intro]: "[X ∈ H; is_MPair X] ⇒ X ∈ pparts H"
| Fst [dest]: "[{X,Y} ∈ pparts H; is_MPair X] ⇒ X ∈ pparts H"
| Snd [dest]: "[{X,Y} ∈ pparts H; is_MPair Y] ⇒ Y ∈ pparts H"
```

### 30.2 basic facts about pparts

```
lemma pparts_is_MPair [dest]: "X ∈ pparts H ⇒ is_MPair X"
⟨proof⟩
```

```
lemma Crypt_notin_pparts [iff]: "Crypt K X ∉ pparts H"
⟨proof⟩
```

```
lemma Key_notin_pparts [iff]: "Key K ∉ pparts H"
⟨proof⟩
```

```
lemma Nonce_notin_pparts [iff]: "Nonce n ∉ pparts H"
⟨proof⟩
```

```
lemma Number_notin_pparts [iff]: "Number n ∉ pparts H"
⟨proof⟩
```

```
lemma Agent_notin_pparts [iff]: "Agent A ∉ pparts H"
⟨proof⟩
```

```
lemma pparts_empty [iff]: "pparts {} = {}"
⟨proof⟩
```

```
lemma pparts_insertI [intro]: "X ∈ pparts H ⇒ X ∈ pparts (insert Y H)"
⟨proof⟩
```

```
lemma pparts_sub: "[X ∈ pparts G; G ⊆ H] ⇒ X ∈ pparts H"
⟨proof⟩
```

```

lemma pparts_insert2 [iff]: "pparts (insert X (insert Y H))
= pparts {X} Un pparts {Y} Un pparts H"
<proof>

lemma pparts_insert_MPair [iff]: "pparts (insert {X,Y} H)
= insert {X,Y} (pparts ({X,Y} ∪ H))"
<proof>

lemma pparts_insert_Nonce [iff]: "pparts (insert (Nonce n) H) = pparts H"
<proof>

lemma pparts_insert_Crypt [iff]: "pparts (insert (Crypt K X) H) = pparts
H"
<proof>

lemma pparts_insert_Key [iff]: "pparts (insert (Key K) H) = pparts H"
<proof>

lemma pparts_insert_Agent [iff]: "pparts (insert (Agent A) H) = pparts H"
<proof>

lemma pparts_insert_Number [iff]: "pparts (insert (Number n) H) = pparts
H"
<proof>

lemma pparts_insert_Hash [iff]: "pparts (insert (Hash X) H) = pparts H"
<proof>

lemma pparts_insert: "X ∈ pparts (insert Y H) ⇒ X ∈ pparts {Y} ∪ pparts
H"
<proof>

lemma insert_pparts: "X ∈ pparts {Y} ∪ pparts H ⇒ X ∈ pparts (insert
Y H)"
<proof>

lemma pparts_Un [iff]: "pparts (G ∪ H) = pparts G ∪ pparts H"
<proof>

lemma pparts_pparts [iff]: "pparts (pparts H) = pparts H"
<proof>

lemma pparts_insert_eq: "pparts (insert X H) = pparts {X} Un pparts H"
<proof>

lemmas pparts_insert_substI = pparts_insert_eq [THEN ssubst]

lemma in_pparts: "Y ∈ pparts H ⇒ ∃X. X ∈ H ∧ Y ∈ pparts {X}"
<proof>

```

### 30.3 facts about *pparts* and *parts*

```

lemma pparts_no_Nonce [dest]: "[X ∈ pparts {Y}; Nonce n ∉ parts {Y}]

```



$\Rightarrow \text{Nonce } n \notin \text{parts } \{X\}$ "  
 $\langle \text{proof} \rangle$

### 30.4 facts about pparts and analz

**lemma** pparts\_analz: " $X \in \text{pparts } H \Rightarrow X \in \text{analz } H$ "  
 $\langle \text{proof} \rangle$

**lemma** pparts\_analz\_sub: " $\llbracket X \in \text{pparts } G; G \subseteq H \rrbracket \Rightarrow X \in \text{analz } H$ "  
 $\langle \text{proof} \rangle$

### 30.5 messages that contribute to analz

**inductive\_set**  
 kparts :: "msg set  $\Rightarrow$  msg set"  
 for H :: "msg set"  
**where**  
 Inj [intro]: " $\llbracket X \in H; \text{not\_MPair } X \rrbracket \Rightarrow X \in \text{kparts } H$ "  
 | Fst [intro]: " $\llbracket \{X, Y\} \in \text{pparts } H; \text{not\_MPair } X \rrbracket \Rightarrow X \in \text{kparts } H$ "  
 | Snd [intro]: " $\llbracket \{X, Y\} \in \text{pparts } H; \text{not\_MPair } Y \rrbracket \Rightarrow Y \in \text{kparts } H$ "

### 30.6 basic facts about kparts

**lemma** kparts\_not\_MPair [dest]: " $X \in \text{kparts } H \Rightarrow \text{not\_MPair } X$ "  
 $\langle \text{proof} \rangle$

**lemma** kparts\_empty [iff]: " $\text{kparts } \{\} = \{\}$ "  
 $\langle \text{proof} \rangle$

**lemma** kparts\_insertI [intro]: " $X \in \text{kparts } H \Rightarrow X \in \text{kparts } (\text{insert } Y H)$ "  
 $\langle \text{proof} \rangle$

**lemma** kparts\_insert2 [iff]: " $\text{kparts } (\text{insert } X (\text{insert } Y H))$   
 $= \text{kparts } \{X\} \cup \text{kparts } \{Y\} \cup \text{kparts } H$ "  
 $\langle \text{proof} \rangle$

**lemma** kparts\_insert\_MPair [iff]: " $\text{kparts } (\text{insert } \{X, Y\} H)$   
 $= \text{kparts } (\{X, Y\} \cup H)$ "  
 $\langle \text{proof} \rangle$

**lemma** kparts\_insert\_Nonce [iff]: " $\text{kparts } (\text{insert } (\text{Nonce } n) H)$   
 $= \text{insert } (\text{Nonce } n) (\text{kparts } H)$ "  
 $\langle \text{proof} \rangle$

**lemma** kparts\_insert\_Crypt [iff]: " $\text{kparts } (\text{insert } (\text{Crypt } K X) H)$   
 $= \text{insert } (\text{Crypt } K X) (\text{kparts } H)$ "  
 $\langle \text{proof} \rangle$

**lemma** kparts\_insert\_Key [iff]: " $\text{kparts } (\text{insert } (\text{Key } K) H)$   
 $= \text{insert } (\text{Key } K) (\text{kparts } H)$ "  
 $\langle \text{proof} \rangle$

**lemma** kparts\_insert\_Agent [iff]: " $\text{kparts } (\text{insert } (\text{Agent } A) H)$   
 $= \text{insert } (\text{Agent } A) (\text{kparts } H)$ "

*<proof>*

**lemma** *kparts\_insert\_Number* [iff]: "kparts (insert (Number n) H)  
= insert (Number n) (kparts H)"  
*<proof>*

**lemma** *kparts\_insert\_Hash* [iff]: "kparts (insert (Hash X) H)  
= insert (Hash X) (kparts H)"  
*<proof>*

**lemma** *kparts\_insert*: " $X \in \text{kparts } (\text{insert } X \ H) \implies X \in \text{kparts } \{X\} \cup \text{kparts } H$ "  
*<proof>*

**lemma** *kparts\_insert\_fst* [rule\_format,dest]: " $X \in \text{kparts } (\text{insert } Z \ H) \implies X \notin \text{kparts } H \longrightarrow X \in \text{kparts } \{Z\}$ "  
*<proof>*

**lemma** *kparts\_sub*: " $\llbracket X \in \text{kparts } G; G \subseteq H \rrbracket \implies X \in \text{kparts } H$ "  
*<proof>*

**lemma** *kparts\_Un* [iff]: " $\text{kparts } (G \cup H) = \text{kparts } G \cup \text{kparts } H$ "  
*<proof>*

**lemma** *pparts\_kparts* [iff]: " $\text{pparts } (\text{kparts } H) = \{\}$ "  
*<proof>*

**lemma** *kparts\_kparts* [iff]: " $\text{kparts } (\text{kparts } H) = \text{kparts } H$ "  
*<proof>*

**lemma** *kparts\_insert\_eq*: " $\text{kparts } (\text{insert } X \ H) = \text{kparts } \{X\} \cup \text{kparts } H$ "  
*<proof>*

**lemmas** *kparts\_insert\_substI* = *kparts\_insert\_eq* [THEN *ssubst*]

**lemma** *in\_kparts*: " $Y \in \text{kparts } H \implies \exists X. X \in H \wedge Y \in \text{kparts } \{X\}$ "  
*<proof>*

**lemma** *kparts\_has\_no\_pair* [iff]: " $\text{has\_no\_pair } (\text{kparts } H)$ "  
*<proof>*

### 30.7 facts about *kparts* and *parts*

**lemma** *kparts\_no\_Nonce* [dest]: " $\llbracket X \in \text{kparts } \{Y\}; \text{Nonce } n \notin \text{parts } \{Y\} \rrbracket \implies \text{Nonce } n \notin \text{parts } \{X\}$ "  
*<proof>*

**lemma** *kparts\_parts*: " $X \in \text{kparts } H \implies X \in \text{parts } H$ "  
*<proof>*

**lemma** *parts\_kparts*: " $X \in \text{parts } (\text{kparts } H) \implies X \in \text{parts } H$ "  
*<proof>*

**lemma** *Crypt\_kparts\_Nonce\_parts* [dest]: " $\llbracket \text{Crypt } K \ Y \in \text{kparts } \{Z\};$

*Nonce*  $n \in \text{parts } \{Y\} \implies \text{Nonce } n \in \text{parts } \{Z\}$ "  
 <proof>

### 30.8 facts about *kparts* and *analz*

**lemma** *kparts\_analz*: " $X \in \text{kparts } H \implies X \in \text{analz } H$ "  
 <proof>

**lemma** *kparts\_analz\_sub*: " $\llbracket X \in \text{kparts } G; G \subseteq H \rrbracket \implies X \in \text{analz } H$ "  
 <proof>

**lemma** *analz\_kparts* [rule\_format, dest]: " $X \in \text{analz } H \implies$   
 $Y \in \text{kparts } \{X\} \longrightarrow Y \in \text{analz } H$ "  
 <proof>

**lemma** *analz\_kparts\_analz*: " $X \in \text{analz } (\text{kparts } H) \implies X \in \text{analz } H$ "  
 <proof>

**lemma** *analz\_kparts\_insert*: " $X \in \text{analz } (\text{kparts } (\text{insert } Z H)) \implies X \in \text{analz } (\text{kparts } \{Z\} \cup \text{kparts } H)$ "  
 <proof>

**lemma** *Nonce\_kparts\_synth* [rule\_format]: " $Y \in \text{synth } (\text{analz } G) \implies$   
 $\text{Nonce } n \in \text{kparts } \{Y\} \longrightarrow \text{Nonce } n \in \text{analz } G$ "  
 <proof>

**lemma** *kparts\_insert\_synth*: " $\llbracket Y \in \text{parts } (\text{insert } X G); X \in \text{synth } (\text{analz } G);$   
 $\text{Nonce } n \in \text{kparts } \{Y\}; \text{Nonce } n \notin \text{analz } G \rrbracket \implies Y \in \text{parts } G$ "  
 <proof>

**lemma** *Crypt\_insert\_synth*:  
 " $\llbracket \text{Crypt } K Y \in \text{parts } (\text{insert } X G); X \in \text{synth } (\text{analz } G); \text{Nonce } n \in \text{kparts } \{Y\};$   
 $\text{Nonce } n \notin \text{analz } G \rrbracket \implies \text{Crypt } K Y \in \text{parts } G$ "  
 <proof>

### 30.9 *analz* is *pparts* + *analz* of *kparts*

**lemma** *analz\_pparts\_kparts*: " $X \in \text{analz } H \implies X \in \text{pparts } H \vee X \in \text{analz } (\text{kparts } H)$ "  
 <proof>

**lemma** *analz\_pparts\_kparts\_eq*: " $\text{analz } H = \text{pparts } H \cup \text{analz } (\text{kparts } H)$ "  
 <proof>

**lemmas** *analz\_pparts\_kparts\_substI* = *analz\_pparts\_kparts\_eq* [THEN *ssubst*]  
**lemmas** *analz\_pparts\_kparts\_substD* = *analz\_pparts\_kparts\_eq* [THEN *sym*, THEN  
*ssubst*]

**end**

## 31 Protocol-Independent Confidentiality Theorem on Nonces

theory Guard imports Analz Extensions begin

inductive\_set

guard :: "nat  $\Rightarrow$  key set  $\Rightarrow$  msg set"

for n :: nat and Ks :: "key set"

where

No\_Nonce [intro]: "Nonce n  $\notin$  parts {X}  $\Longrightarrow$  X  $\in$  guard n Ks"

| Guard\_Nonce [intro]: "invKey K  $\in$  Ks  $\Longrightarrow$  Crypt K X  $\in$  guard n Ks"

| Crypt [intro]: "X  $\in$  guard n Ks  $\Longrightarrow$  Crypt K X  $\in$  guard n Ks"

| Pair [intro]: "[X  $\in$  guard n Ks; Y  $\in$  guard n Ks]  $\Longrightarrow$  {X,Y}  $\in$  guard n Ks"

### 31.1 basic facts about guard

lemma Key\_is\_guard [iff]: "Key K  $\in$  guard n Ks"

$\langle$ proof $\rangle$

lemma Agent\_is\_guard [iff]: "Agent A  $\in$  guard n Ks"

$\langle$ proof $\rangle$

lemma Number\_is\_guard [iff]: "Number r  $\in$  guard n Ks"

$\langle$ proof $\rangle$

lemma Nonce\_notin\_guard: "X  $\in$  guard n Ks  $\Longrightarrow$  X  $\neq$  Nonce n"

$\langle$ proof $\rangle$

lemma Nonce\_notin\_guard\_iff [iff]: "Nonce n  $\notin$  guard n Ks"

$\langle$ proof $\rangle$

lemma guard\_has\_Crypt [rule\_format]: "X  $\in$  guard n Ks  $\Longrightarrow$  Nonce n  $\in$  parts {X}"

$\longrightarrow (\exists K Y. \text{Crypt } K Y \in \text{kparts } \{X\} \wedge \text{Nonce } n \in \text{parts } \{Y\})"$

$\langle$ proof $\rangle$

lemma Nonce\_notin\_kparts\_msg: "X  $\in$  guard n Ks  $\Longrightarrow$  Nonce n  $\notin$  kparts {X}"

$\langle$ proof $\rangle$

lemma Nonce\_in\_kparts\_imp\_no\_guard: "Nonce n  $\in$  kparts H

$\Longrightarrow \exists X. X \in H \wedge X \notin$  guard n Ks"

$\langle$ proof $\rangle$

lemma guard\_kparts [rule\_format]: "X  $\in$  guard n Ks  $\Longrightarrow$

Y  $\in$  kparts {X}  $\longrightarrow$  Y  $\in$  guard n Ks"

$\langle$ proof $\rangle$

lemma guard\_Crypt: "[Crypt K Y  $\in$  guard n Ks; K  $\notin$  invKey'Ks]  $\Longrightarrow$  Y  $\in$  guard n Ks"

$\langle$ proof $\rangle$

lemma guard\_MPair [iff]: "({X,Y}  $\in$  guard n Ks) = (X  $\in$  guard n Ks  $\wedge$  Y  $\in$

*guard n Ks*)"  
 <proof>

**lemma** *guard\_not\_guard* [rule\_format]: " $X \in \text{guard } n \text{ } Ks \implies$   
 $\text{Crypt } K \ Y \in \text{kparts } \{X\} \longrightarrow \text{Nonce } n \in \text{kparts } \{Y\} \longrightarrow Y \notin \text{guard } n \text{ } Ks$ "  
 <proof>

**lemma** *guard\_extand*: " $\llbracket X \in \text{guard } n \text{ } Ks; Ks \subseteq Ks' \rrbracket \implies X \in \text{guard } n \text{ } Ks'$ "  
 <proof>

## 31.2 guarded sets

**definition** *Guard* :: " $\text{nat} \Rightarrow \text{key set} \Rightarrow \text{msg set} \Rightarrow \text{bool}$ " where  
 " $\text{Guard } n \text{ } Ks \ H \equiv \forall X. X \in H \longrightarrow X \in \text{guard } n \text{ } Ks$ "

## 31.3 basic facts about *Guard*

**lemma** *Guard\_empty* [iff]: " $\text{Guard } n \text{ } Ks \ \{\}$ "  
 <proof>

**lemma** *notin\_parts\_Guard* [intro]: " $\text{Nonce } n \notin \text{parts } G \implies \text{Guard } n \text{ } Ks \ G$ "  
 <proof>

**lemma** *Nonce\_notin\_kparts* [simplified]: " $\text{Guard } n \text{ } Ks \ H \implies \text{Nonce } n \notin \text{kparts } H$ "  
 <proof>

**lemma** *Guard\_must\_decrypt*: " $\llbracket \text{Guard } n \text{ } Ks \ H; \text{Nonce } n \in \text{analz } H \rrbracket \implies$   
 $\exists K \ Y. \text{Crypt } K \ Y \in \text{kparts } H \wedge \text{Key } (\text{invKey } K) \in \text{kparts } H$ "  
 <proof>

**lemma** *Guard\_kparts* [intro]: " $\text{Guard } n \text{ } Ks \ H \implies \text{Guard } n \text{ } Ks \ (\text{kparts } H)$ "  
 <proof>

**lemma** *Guard\_mono*: " $\llbracket \text{Guard } n \text{ } Ks \ H; G \leq H \rrbracket \implies \text{Guard } n \text{ } Ks \ G$ "  
 <proof>

**lemma** *Guard\_insert* [iff]: " $\text{Guard } n \text{ } Ks \ (\text{insert } X \ H)$   
 $= (\text{Guard } n \text{ } Ks \ H \wedge X \in \text{guard } n \text{ } Ks)$ "  
 <proof>

**lemma** *Guard\_Un* [iff]: " $\text{Guard } n \text{ } Ks \ (G \text{ Un } H) = (\text{Guard } n \text{ } Ks \ G \ \& \ \text{Guard } n \text{ } Ks \ H)$ "  
 <proof>

**lemma** *Guard\_synth* [intro]: " $\text{Guard } n \text{ } Ks \ G \implies \text{Guard } n \text{ } Ks \ (\text{synth } G)$ "  
 <proof>

**lemma** *Guard\_analz* [intro]: " $\llbracket \text{Guard } n \text{ } Ks \ G; \forall K. K \in Ks \longrightarrow \text{Key } K \notin \text{analz } G \rrbracket$   
 $\implies \text{Guard } n \text{ } Ks \ (\text{analz } G)$ "  
 <proof>

**lemma** *in\_Guard* [dest]: " $\llbracket X \in G; \text{Guard } n \text{ } Ks \ G \rrbracket \implies X \in \text{guard } n \text{ } Ks$ "  
 <proof>

**lemma** *in\_synth\_Guard*: " $\llbracket X \in \text{synth } G; \text{Guard } n \text{ Ks } G \rrbracket \implies X \in \text{guard } n \text{ Ks}$ "  
 $\langle \text{proof} \rangle$

**lemma** *in\_analz\_Guard*: " $\llbracket X \in \text{analz } G; \text{Guard } n \text{ Ks } G; \forall K. K \in \text{Ks} \longrightarrow \text{Key } K \notin \text{analz } G \rrbracket \implies X \in \text{guard } n \text{ Ks}$ "  
 $\langle \text{proof} \rangle$

**lemma** *Guard\_keyset [simp]*: " $\text{keyset } G \implies \text{Guard } n \text{ Ks } G$ "  
 $\langle \text{proof} \rangle$

**lemma** *Guard\_Un\_keyset*: " $\llbracket \text{Guard } n \text{ Ks } G; \text{keyset } H \rrbracket \implies \text{Guard } n \text{ Ks } (G \cup H)$ "  
 $\langle \text{proof} \rangle$

**lemma** *in\_Guard\_kparts*: " $\llbracket X \in G; \text{Guard } n \text{ Ks } G; Y \in \text{kparts } \{X\} \rrbracket \implies Y \in \text{guard } n \text{ Ks}$ "  
 $\langle \text{proof} \rangle$

**lemma** *in\_Guard\_kparts\_neq*: " $\llbracket X \in G; \text{Guard } n \text{ Ks } G; \text{Nonce } n' \in \text{kparts } \{X\} \rrbracket \implies n \neq n'$ "  
 $\langle \text{proof} \rangle$

**lemma** *in\_Guard\_kparts\_Crypt*: " $\llbracket X \in G; \text{Guard } n \text{ Ks } G; \text{is\_MPair } X; \text{Crypt } K \ Y \in \text{kparts } \{X\}; \text{Nonce } n \in \text{kparts } \{Y\} \rrbracket \implies \text{invKey } K \in \text{Ks}$ "  
 $\langle \text{proof} \rangle$

**lemma** *Guard\_extand*: " $\llbracket \text{Guard } n \text{ Ks } G; \text{Ks} \subseteq \text{Ks}' \rrbracket \implies \text{Guard } n \text{ Ks}' \ G$ "  
 $\langle \text{proof} \rangle$

**lemma** *guard\_invKey [rule\_format]*: " $\llbracket X \in \text{guard } n \text{ Ks}; \text{Nonce } n \in \text{kparts } \{Y\} \rrbracket \implies \text{Crypt } K \ Y \in \text{kparts } \{X\} \longrightarrow \text{invKey } K \in \text{Ks}$ "  
 $\langle \text{proof} \rangle$

**lemma** *Crypt\_guard\_invKey [rule\_format]*: " $\llbracket \text{Crypt } K \ Y \in \text{guard } n \text{ Ks}; \text{Nonce } n \in \text{kparts } \{Y\} \rrbracket \implies \text{invKey } K \in \text{Ks}$ "  
 $\langle \text{proof} \rangle$

### 31.4 set obtained by decrypting a message

**abbreviation** (*input*)  
 $\text{decrypt} :: \text{"msg set} \Rightarrow \text{key} \Rightarrow \text{msg} \Rightarrow \text{msg set" where}$   
 $\text{"decrypt } H \ K \ Y == \text{insert } Y \ (H - \{\text{Crypt } K \ Y\})"$

**lemma** *analz\_decrypt*: " $\llbracket \text{Crypt } K \ Y \in H; \text{Key } (\text{invKey } K) \in H; \text{Nonce } n \in \text{analz } H \rrbracket \implies \text{Nonce } n \in \text{analz } (\text{decrypt } H \ K \ Y)$ "  
 $\langle \text{proof} \rangle$

**lemma** *parts\_decrypt*: " $\llbracket \text{Crypt } K \ Y \in H; X \in \text{parts } (\text{decrypt } H \ K \ Y) \rrbracket \implies X \in \text{parts } H$ "  
 $\langle \text{proof} \rangle$

### 31.5 number of Crypt's in a message

```

fun crypt_nb :: "msg => nat"
where
  "crypt_nb (Crypt K X) = Suc (crypt_nb X)"
| "crypt_nb {X,Y} = crypt_nb X + crypt_nb Y"
| "crypt_nb X = 0"

```

### 31.6 basic facts about crypt\_nb

```

lemma non_empty_crypt_msg: "Crypt K Y ∈ parts {X} ⟹ crypt_nb X ≠ 0"
⟨proof⟩

```

### 31.7 number of Crypt's in a message list

```

primrec cnb :: "msg list => nat"
where
  "cnb [] = 0"
| "cnb (X#l) = crypt_nb X + cnb l"

```

### 31.8 basic facts about cnb

```

lemma cnb_app [simp]: "cnb (l @ l') = cnb l + cnb l'"
⟨proof⟩

```

```

lemma mem_cnb_minus: "x ∈ set l ⟹ cnb l = crypt_nb x + (cnb l - crypt_nb x)"
⟨proof⟩

```

```

lemmas mem_cnb_minus_substI = mem_cnb_minus [THEN ssubst]

```

```

lemma cnb_minus [simp]: "x ∈ set l ⟹ cnb (remove l x) = cnb l - crypt_nb x"
⟨proof⟩

```

```

lemma parts_cnb: "Z ∈ parts (set l) ⟹
cnb l = (cnb l - crypt_nb Z) + crypt_nb Z"
⟨proof⟩

```

```

lemma non_empty_crypt: "Crypt K Y ∈ parts (set l) ⟹ cnb l ≠ 0"
⟨proof⟩

```

### 31.9 list of kparts

```

lemma kparts_msg_set: "∃ l. kparts {X} = set l ∧ cnb l = crypt_nb X"
⟨proof⟩

```

```

lemma kparts_set: "∃ l'. kparts (set l) = set l' ∧ cnb l' = cnb l"
⟨proof⟩

```

### 31.10 list corresponding to "decrypt"

```

definition decrypt' :: "msg list => key => msg => msg list" where
"decrypt' l K Y == Y # remove l (Crypt K Y)"

```

```
declare decrypt'_def [simp]
```

### 31.11 basic facts about *decrypt'*

```
lemma decrypt_minus: "decrypt (set l) K Y <= set (decrypt' l K Y)"
<proof>
```

### 31.12 if the analyse of a finite guarded set gives *n* then it must also gives one of the keys of *Ks*

```
lemma Guard_invKey_by_list [rule_format]: "∀l. cnb l = p
→ Guard n Ks (set l) → Nonce n ∈ analz (set l)
→ (∃K. K ∈ Ks ∧ Key K ∈ analz (set l))"
<proof>
```

```
lemma Guard_invKey_finite: "[Nonce n ∈ analz G; Guard n Ks G; finite G]
⇒ ∃K. K ∈ Ks ∧ Key K ∈ analz G"
<proof>
```

```
lemma Guard_invKey: "[Nonce n ∈ analz G; Guard n Ks G]
⇒ ∃K. K ∈ Ks ∧ Key K ∈ analz G"
<proof>
```

### 31.13 if the analyse of a finite guarded set and a (possibly infinite) set of keys gives *n* then it must also gives *Ks*

```
lemma Guard_invKey_keyset: "[Nonce n ∈ analz (G ∪ H); Guard n Ks G; finite
G;
keyset H] ⇒ ∃K. K ∈ Ks ∧ Key K ∈ analz (G ∪ H)"
<proof>
```

```
end
```

## 32 protocol-independent confidentiality theorem on keys

```
theory GuardK
imports Analz Extensions
begin
```

```
inductive_set
```

```
  guardK :: "nat => key set => msg set"
  for n :: nat and Ks :: "key set"
```

```
where
```

```
  No_Key [intro]: "Key n ∉ parts {X} ⇒ X ∈ guardK n Ks"
  | Guard_Key [intro]: "invKey K ∈ Ks ⇒ Crypt K X ∈ guardK n Ks"
  | Crypt [intro]: "X ∈ guardK n Ks ⇒ Crypt K X ∈ guardK n Ks"
  | Pair [intro]: "[X ∈ guardK n Ks; Y ∈ guardK n Ks] ⇒ {X,Y} ∈ guardK n
Ks"
```



### 32.1 basic facts about *guardK*

**lemma** *Nonce\_is\_guardK [iff]: "Nonce p ∈ guardK n Ks"*  
*<proof>*

**lemma** *Agent\_is\_guardK [iff]: "Agent A ∈ guardK n Ks"*  
*<proof>*

**lemma** *Number\_is\_guardK [iff]: "Number r ∈ guardK n Ks"*  
*<proof>*

**lemma** *Key\_notin\_guardK: "X ∈ guardK n Ks ⇒ X ≠ Key n"*  
*<proof>*

**lemma** *Key\_notin\_guardK\_iff [iff]: "Key n ∉ guardK n Ks"*  
*<proof>*

**lemma** *guardK\_has\_Crypt [rule\_format]: "X ∈ guardK n Ks ⇒ Key n ∈ parts {X}"*  
 $\longrightarrow (\exists K Y. \text{Crypt } K Y \in \text{kparts } \{X\} \wedge \text{Key } n \in \text{parts } \{Y\})"$   
*<proof>*

**lemma** *Key\_notin\_kparts\_msg: "X ∈ guardK n Ks ⇒ Key n ∉ kparts {X}"*  
*<proof>*

**lemma** *Key\_in\_kparts\_imp\_no\_guardK: "Key n ∈ kparts H*  
 $\Rightarrow \exists X. X \in H \wedge X \notin \text{guardK } n Ks"$   
*<proof>*

**lemma** *guardK\_kparts [rule\_format]: "X ∈ guardK n Ks ⇒*  
 $Y \in \text{kparts } \{X\} \longrightarrow Y \in \text{guardK } n Ks"$   
*<proof>*

**lemma** *guardK\_Crypt: "[Crypt K Y ∈ guardK n Ks; K ∉ invKey'Ks] ⇒ Y ∈ guardK n Ks"*  
*<proof>*

**lemma** *guardK\_MPair [iff]: "(⟦X, Y⟧ ∈ guardK n Ks)*  
 $= (X \in \text{guardK } n Ks \wedge Y \in \text{guardK } n Ks)"$   
*<proof>*

**lemma** *guardK\_not\_guardK [rule\_format]: "X ∈ guardK n Ks ⇒*  
 $\text{Crypt } K Y \in \text{kparts } \{X\} \longrightarrow \text{Key } n \in \text{kparts } \{Y\} \longrightarrow Y \notin \text{guardK } n Ks"$   
*<proof>*

**lemma** *guardK\_extand: "[X ∈ guardK n Ks; Ks ⊆ Ks';*  
 $\llbracket K \in Ks'; K \notin Ks \rrbracket \Rightarrow \text{Key } K \notin \text{parts } \{X\} \rrbracket \Rightarrow X \in \text{guardK } n Ks'"$   
*<proof>*

### 32.2 guarded sets

**definition** *GuardK :: "nat ⇒ key set ⇒ msg set ⇒ bool" where*  
 $"\text{GuardK } n Ks H \equiv \forall X. X \in H \longrightarrow X \in \text{guardK } n Ks"$

### 32.3 basic facts about *GuardK*

**lemma** *GuardK\_empty* [iff]: "*GuardK* *n* *Ks* {}"

⟨*proof*⟩

**lemma** *Key\_notin\_kparts* [simplified]: "*GuardK* *n* *Ks* *H*  $\implies$  *Key* *n*  $\notin$  *kparts* *H*"

⟨*proof*⟩

**lemma** *GuardK\_must\_decrypt*: " $\llbracket \text{GuardK } n \text{ } Ks \text{ } H; \text{Key } n \in \text{analz } H \rrbracket \implies \exists K \ Y. \text{Crypt } K \ Y \in \text{kparts } H \wedge \text{Key } (\text{invKey } K) \in \text{kparts } H$ "

⟨*proof*⟩

**lemma** *GuardK\_kparts* [intro]: "*GuardK* *n* *Ks* *H*  $\implies$  *GuardK* *n* *Ks* (*kparts* *H*)"

⟨*proof*⟩

**lemma** *GuardK\_mono*: " $\llbracket \text{GuardK } n \text{ } Ks \text{ } H; G \subseteq H \rrbracket \implies \text{GuardK } n \text{ } Ks \text{ } G$ "

⟨*proof*⟩

**lemma** *GuardK\_insert* [iff]: "*GuardK* *n* *Ks* (*insert* *X* *H*)

= (*GuardK* *n* *Ks* *H*  $\wedge$  *X*  $\in$  *guardK* *n* *Ks*)"

⟨*proof*⟩

**lemma** *GuardK\_Un* [iff]: "*GuardK* *n* *Ks* (*G* *Un* *H*) = (*GuardK* *n* *Ks* *G* & *GuardK* *n* *Ks* *H*)"

⟨*proof*⟩

**lemma** *GuardK\_synth* [intro]: "*GuardK* *n* *Ks* *G*  $\implies$  *GuardK* *n* *Ks* (*synth* *G*)"

⟨*proof*⟩

**lemma** *GuardK\_analz* [intro]: " $\llbracket \text{GuardK } n \text{ } Ks \text{ } G; \forall K. K \in Ks \longrightarrow \text{Key } K \notin \text{analz } G \rrbracket$

$\implies \text{GuardK } n \text{ } Ks \text{ } (\text{analz } G)$ "

⟨*proof*⟩

**lemma** *in\_GuardK* [dest]: " $\llbracket X \in G; \text{GuardK } n \text{ } Ks \text{ } G \rrbracket \implies X \in \text{guardK } n \text{ } Ks$ "

⟨*proof*⟩

**lemma** *in\_synth\_GuardK*: " $\llbracket X \in \text{synth } G; \text{GuardK } n \text{ } Ks \text{ } G \rrbracket \implies X \in \text{guardK } n \text{ } Ks$ "

⟨*proof*⟩

**lemma** *in\_analz\_GuardK*: " $\llbracket X \in \text{analz } G; \text{GuardK } n \text{ } Ks \text{ } G; \forall K. K \in Ks \longrightarrow \text{Key } K \notin \text{analz } G \rrbracket \implies X \in \text{guardK } n \text{ } Ks$ "

⟨*proof*⟩

**lemma** *GuardK\_keyset* [simp]: " $\llbracket \text{keyset } G; \text{Key } n \notin G \rrbracket \implies \text{GuardK } n \text{ } Ks \text{ } G$ "

⟨*proof*⟩

**lemma** *GuardK\_Un\_keyset*: " $\llbracket \text{GuardK } n \text{ } Ks \text{ } G; \text{keyset } H; \text{Key } n \notin H \rrbracket$

$\implies \text{GuardK } n \text{ } Ks \text{ } (G \text{ Un } H)$ "

⟨*proof*⟩

**lemma** *in\_GuardK\_kparts*: " $\llbracket X \in G; \text{GuardK } n \text{ } Ks \text{ } G; Y \in \text{kparts } \{X\} \rrbracket \implies Y \in \text{guardK } n \text{ } Ks$ "

⟨*proof*⟩

**lemma** *in\_GuardK\_kparts\_neq*: " $\llbracket X \in G; \text{GuardK } n \text{ Ks } G; \text{Key } n' \in \text{kparts } \{X\} \rrbracket \implies n \neq n'$ "  
 <proof>

**lemma** *in\_GuardK\_kparts\_Crypt*: " $\llbracket X \in G; \text{GuardK } n \text{ Ks } G; \text{is\_MPair } X; \text{Crypt } K \text{ Y} \in \text{kparts } \{X\}; \text{Key } n \in \text{kparts } \{Y\} \rrbracket \implies \text{invKey } K \in \text{Ks}$ "  
 <proof>

**lemma** *GuardK\_extand*: " $\llbracket \text{GuardK } n \text{ Ks } G; \text{Ks} \subseteq \text{Ks}'; \llbracket K \in \text{Ks}'; K \notin \text{Ks} \rrbracket \implies \text{Key } K \notin \text{parts } G \rrbracket \implies \text{GuardK } n \text{ Ks}' \text{ } G$ "  
 <proof>

## 32.4 set obtained by decrypting a message

**abbreviation** (*input*)

*decrypt* :: "*msg set*  $\Rightarrow$  *key*  $\Rightarrow$  *msg*  $\Rightarrow$  *msg set*" **where**  
 "*decrypt* *H K Y*  $\equiv$  *insert Y (H - {Crypt K Y})*"

**lemma** *analz\_decrypt*: " $\llbracket \text{Crypt } K \text{ Y} \in H; \text{Key } (\text{invKey } K) \in H; \text{Key } n \in \text{analz } H \rrbracket \implies \text{Key } n \in \text{analz } (\text{decrypt } H \text{ K Y})$ "  
 <proof>

**lemma** *parts\_decrypt*: " $\llbracket \text{Crypt } K \text{ Y} \in H; X \in \text{parts } (\text{decrypt } H \text{ K Y}) \rrbracket \implies X \in \text{parts } H$ "  
 <proof>

## 32.5 number of Crypt's in a message

**fun** *crypt\_nb* :: "*msg*  $\Rightarrow$  *nat*" **where**  
 "*crypt\_nb* (*Crypt K X*) = *Suc (crypt\_nb X)*" |  
 "*crypt\_nb*  $\{X, Y\}$  = *crypt\_nb X* + *crypt\_nb Y*" |  
 "*crypt\_nb X* = 0"

## 32.6 basic facts about *crypt\_nb*

**lemma** *non\_empty\_crypt\_msg*: "*Crypt K Y*  $\in \text{parts } \{X\} \implies \text{crypt\_nb } X \neq 0$ "  
 <proof>

## 32.7 number of Crypt's in a message list

**primrec** *cnb* :: "*msg list*  $\Rightarrow$  *nat*" **where**  
 "*cnb []* = 0" |  
 "*cnb (X#l)* = *crypt\_nb X* + *cnb l*"

## 32.8 basic facts about *cnb*

**lemma** *cnb\_app [simp]*: "*cnb (l @ l')* = *cnb l* + *cnb l'*"  
 <proof>

**lemma** *mem\_cnb\_minus*: "*x*  $\in \text{set } l \implies \text{cnb } l = \text{crypt\_nb } x + (\text{cnb } l - \text{crypt\_nb } x)$ "  
 <proof>

**lemmas** *mem\_cnb\_minus\_substI* = *mem\_cnb\_minus* [THEN *ssubst*]

**lemma** *cnb\_minus* [*simp*]: " $x \in \text{set } l \implies \text{cnb } (\text{remove } l \ x) = \text{cnb } l - \text{crypt\_nb } x$ "  
 <proof>

**lemma** *parts\_cnb*: " $Z \in \text{parts } (\text{set } l) \implies \text{cnb } l = (\text{cnb } l - \text{crypt\_nb } Z) + \text{crypt\_nb } Z$ "  
 <proof>

**lemma** *non\_empty\_crypt*: " $\text{Crypt } K \ Y \in \text{parts } (\text{set } l) \implies \text{cnb } l \neq 0$ "  
 <proof>

### 32.9 list of kparts

**lemma** *kparts\_msg\_set*: " $\exists l. \text{kparts } \{X\} = \text{set } l \wedge \text{cnb } l = \text{crypt\_nb } X$ "  
 <proof>

**lemma** *kparts\_set*: " $\exists l'. \text{kparts } (\text{set } l) = \text{set } l' \ \& \ \text{cnb } l' = \text{cnb } l$ "  
 <proof>

### 32.10 list corresponding to "decrypt"

**definition** *decrypt'* :: " $\text{msg list} \Rightarrow \text{key} \Rightarrow \text{msg} \Rightarrow \text{msg list}$ " **where**  
 " $\text{decrypt}' \ l \ K \ Y == Y \ \# \ \text{remove } l \ (\text{Crypt } K \ Y)$ "

**declare** *decrypt'\_def* [*simp*]

### 32.11 basic facts about *decrypt'*

**lemma** *decrypt\_minus*: " $\text{decrypt } (\text{set } l) \ K \ Y \leq \text{set } (\text{decrypt}' \ l \ K \ Y)$ "  
 <proof>

if the analysis of a finite guarded set gives *n* then it must also give one of the keys of *Ks*

**lemma** *GuardK\_invKey\_by\_list* [*rule\_format*]: " $\forall l. \text{cnb } l = p \longrightarrow \text{GuardK } n \ Ks \ (\text{set } l) \longrightarrow \text{Key } n \in \text{analz } (\text{set } l) \longrightarrow (\exists K. K \in Ks \wedge \text{Key } K \in \text{analz } (\text{set } l))$ "  
 <proof>

**lemma** *GuardK\_invKey\_finite*: " $\llbracket \text{Key } n \in \text{analz } G; \text{GuardK } n \ Ks \ G; \text{finite } G \rrbracket \implies \exists K. K \in Ks \wedge \text{Key } K \in \text{analz } G$ "  
 <proof>

**lemma** *GuardK\_invKey*: " $\llbracket \text{Key } n \in \text{analz } G; \text{GuardK } n \ Ks \ G \rrbracket \implies \exists K. K \in Ks \wedge \text{Key } K \in \text{analz } G$ "  
 <proof>

if the analyse of a finite guarded set and a (possibly infinite) set of keys gives *n* then it must also gives *Ks*

**lemma** *GuardK\_invKey\_keyset*: " $\llbracket \text{Key } n \in \text{analz } (G \cup H); \text{GuardK } n \ Ks \ G; \text{finite } G; \text{keyset } H; \text{Key } n \notin H \rrbracket \implies \exists K. K \in Ks \wedge \text{Key } K \in \text{analz } (G \cup H)$ "

*<proof>*

**end**

```
theory Shared
imports Event All_Symmetric
begin
```

```
consts
  shrK      :: "agent  $\Rightarrow$  key"
```

```
specification (shrK)
  inj_shrK: "inj shrK"
  — No two agents have the same long-term key
  <proof>
```

Server knows all long-term keys; other agents know only their own

```
overloading
  initState  $\equiv$  initState
begin
```

```
primrec initState where
  initState_Server: "initState Server      = Key ` range shrK"
| initState_Friend: "initState (Friend i) = {Key (shrK (Friend i))}"
| initState_Spy:    "initState Spy        = Key ` shrK ` bad"
```

**end**

## 32.12 Basic properties of shrK

```
lemmas shrK_injective = inj_shrK [THEN inj_eq]
declare shrK_injective [iff]
```

```
lemma invKey_K [simp]: "invKey K = K"
<proof>
```

```
lemma analz_Decrypt' [dest]:
  "[Crypt K X  $\in$  analz H; Key K  $\in$  analz H]  $\Longrightarrow$  X  $\in$  analz H"
<proof>
```

Now cancel the *dest* attribute given to *analz.Decrypt* in its declaration.

```
declare analz.Decrypt [rule del]
```

Rewrites should not refer to *initState (Friend i)* because that expression is not in normal form.

```
lemma keysFor_parts_initState [simp]: "keysFor (parts (initState C)) = {}"
<proof>
```

```
lemma keysFor_parts_insert:
  "[K  $\in$  keysFor (parts (insert X G)); X  $\in$  synth (analz H)]
```

$\implies K \in \text{keysFor } (\text{parts } (G \cup H)) \mid \text{Key } K \in \text{parts } H$   
 $\langle \text{proof} \rangle$

**lemma** *Crypt\_imp\_keysFor*: "Crypt  $K$   $X \in H \implies K \in \text{keysFor } H$ "  
 $\langle \text{proof} \rangle$

### 32.13 Function "knows"

**lemma** *Spy\_knows\_Spy\_bad* [intro!]: " $A \in \text{bad} \implies \text{Key } (\text{shrK } A) \in \text{knows } \text{Spy evs}$ "  
 $\langle \text{proof} \rangle$

**lemma** *Crypt\_Spy\_analz\_bad*: " $\llbracket \text{Crypt } (\text{shrK } A) X \in \text{analz } (\text{knows } \text{Spy evs}); A \in \text{bad} \rrbracket$   
 $\implies X \in \text{analz } (\text{knows } \text{Spy evs})$ "  
 $\langle \text{proof} \rangle$

**lemma** *shrK\_in\_initState* [iff]: " $\text{Key } (\text{shrK } A) \in \text{initState } A$ "  
 $\langle \text{proof} \rangle$

**lemma** *shrK\_in\_used* [iff]: " $\text{Key } (\text{shrK } A) \in \text{used evs}$ "  
 $\langle \text{proof} \rangle$

**lemma** *Key\_not\_used* [simp]: " $\text{Key } K \notin \text{used evs} \implies K \notin \text{range shrK}$ "  
 $\langle \text{proof} \rangle$

**lemma** *shrK\_neq* [simp]: " $\text{Key } K \notin \text{used evs} \implies \text{shrK } B \neq K$ "  
 $\langle \text{proof} \rangle$

**lemmas** *shrK\_sym\_neq* = *shrK\_neq* [THEN not\_sym]  
**declare** *shrK\_sym\_neq* [simp]

### 32.14 Fresh nonces

**lemma** *Nonce\_notin\_initState* [iff]: " $\text{Nonce } N \notin \text{parts } (\text{initState } B)$ "  
 $\langle \text{proof} \rangle$

**lemma** *Nonce\_notin\_used\_empty* [simp]: " $\text{Nonce } N \notin \text{used } []$ "  
 $\langle \text{proof} \rangle$

### 32.15 Supply fresh nonces for possibility theorems.

**lemma** *Nonce\_supply\_lemma*: " $\exists N. \forall n. N \leq n \longrightarrow \text{Nonce } n \notin \text{used evs}$ "  
 $\langle \text{proof} \rangle$

**lemma** *Nonce\_supply1*: " $\exists N. \text{Nonce } N \notin \text{used evs}$ "  
 $\langle \text{proof} \rangle$

**lemma** *Nonce\_supply2*: " $\exists N\ N'. \text{Nonce } N \notin \text{used evs} \wedge \text{Nonce } N' \notin \text{used evs}'$   
 $\wedge N \neq N'$ "  
 <proof>

**lemma** *Nonce\_supply3*: " $\exists N\ N'\ N''. \text{Nonce } N \notin \text{used evs} \wedge \text{Nonce } N' \notin \text{used evs}'$   
 $\wedge$   
 $\text{Nonce } N'' \notin \text{used evs}'' \wedge N \neq N' \wedge N' \neq N'' \wedge N \neq N''$ "  
 <proof>

**lemma** *Nonce\_supply*: "*Nonce* (*SOME* *N*. *Nonce* *N*  $\notin$  *used evs*)  $\notin$  *used evs*"  
 <proof>

Unlike the corresponding property of nonces, we cannot prove *finite* *KK*  $\implies$   $\exists K. K \notin \text{used evs} \wedge \text{Key } K \notin \text{used evs}$ . We have infinitely many agents and there is nothing to stop their long-term keys from exhausting all the natural numbers. Instead, possibility theorems must assume the existence of a few keys.

## 32.16 Specialized Rewriting for Theorems About *analz* and *Image*

**lemma** *subset\_Compl\_range*: " $A \subseteq -(\text{range shrK}) \implies \text{shrK } x \notin A$ "  
 <proof>

**lemma** *insert\_Key\_singleton*: "*insert* (*Key* *K*) *H* = *Key* ' {*K*}  $\cup$  *H*"  
 <proof>

**lemma** *insert\_Key\_image*: "*insert* (*Key* *K*) (*Key*'*KK*  $\cup$  *C*) = *Key*'(*insert* *K* *KK*)  
 $\cup$  *C*"  
 <proof>

**lemmas** *analz\_image\_freshK\_simps* =  
*simp\_thms mem\_simps* — these two allow its use with *only*:  
*disj\_comms*  
*image\_insert* [THEN *sym*] *image\_Un* [THEN *sym*] *empty\_subsetI* *insert\_subset*  
*analz\_insert\_eq* *Un\_upper2* [THEN *analz\_mono*, THEN [2] *rev\_subsetD*]  
*insert\_Key\_singleton* *subset\_Compl\_range*  
*Key\_not\_used* *insert\_Key\_image* *Un\_assoc* [THEN *sym*]

**lemma** *analz\_image\_freshK\_lemma*:  
 " $(\text{Key } K \in \text{analz } (\text{Key}'nE \cup H)) \longrightarrow (K \in nE \mid \text{Key } K \in \text{analz } H) \implies$   
 $(\text{Key } K \in \text{analz } (\text{Key}'nE \cup H)) = (K \in nE \mid \text{Key } K \in \text{analz } H)$ "  
 <proof>

## 32.17 Tactics for possibility theorems

<ML>

```

lemma invKey_shrK_iff [iff]:
  "(Key (invKey K) ∈ X) = (Key K ∈ X)"
  ⟨proof⟩

```

⟨ML⟩

```

lemma knows_subset_knows_Cons: "knows A evs ⊆ knows A (e # evs)"
  ⟨proof⟩

```

**end**

### 33 lemmas on guarded messages for protocols with symmetric keys

```

theory Guard_Shared imports Guard GuardK "../Shared" begin

```

#### 33.1 Extensions to Theory Shared

```

declare initState.simps [simp del]

```

##### 33.1.1 a little abbreviation

```

abbreviation
  Ciph :: "agent => msg => msg" where
    "Ciph A X == Crypt (shrK A) X"

```

##### 33.1.2 agent associated to a key

```

definition agt :: "key => agent" where
  "agt K == SOME A. K = shrK A"

```

```

lemma agt_shrK [simp]: "agt (shrK A) = A"
  ⟨proof⟩

```

##### 33.1.3 basic facts about initState

```

lemma no_Crypt_in_parts_init [simp]: "Crypt K X ∉ parts (initState A)"
  ⟨proof⟩

```

```

lemma no_Crypt_in_analz_init [simp]: "Crypt K X ∉ analz (initState A)"
  ⟨proof⟩

```

```

lemma no_shrK_in_analz_init [simp]: "A ∉ bad
  ⇒ Key (shrK A) ∉ analz (initState Spy)"
  ⟨proof⟩

```

```

lemma shrK_notin_initState_Friend [simp]: "A ≠ Friend C
  ⇒ Key (shrK A) ∉ parts (initState (Friend C))"
  ⟨proof⟩

```

```

lemma keyset_init [iff]: "keyset (initState A)"
  ⟨proof⟩

```



### 33.1.4 sets of symmetric keys

**definition** *shrK\_set* :: "key set  $\Rightarrow$  bool" where  
 "shrK\_set Ks  $\equiv \forall K. K \in Ks \longrightarrow (\exists A. K = \text{shrK } A)$ "

**lemma** *in\_shrK\_set*: " $\llbracket \text{shrK\_set } Ks; K \in Ks \rrbracket \Longrightarrow \exists A. K = \text{shrK } A$ "  
 <proof>

**lemma** *shrK\_set1* [iff]: "*shrK\_set* {*shrK A*}"  
 <proof>

**lemma** *shrK\_set2* [iff]: "*shrK\_set* {*shrK A*, *shrK B*}"  
 <proof>

### 33.1.5 sets of good keys

**definition** *good* :: "key set  $\Rightarrow$  bool" where  
 "good Ks  $\equiv \forall K. K \in Ks \longrightarrow \text{agt } K \notin \text{bad}$ "

**lemma** *in\_good*: " $\llbracket \text{good } Ks; K \in Ks \rrbracket \Longrightarrow \text{agt } K \notin \text{bad}$ "  
 <proof>

**lemma** *good1* [simp]: " $A \notin \text{bad} \Longrightarrow \text{good } \{\text{shrK } A\}$ "  
 <proof>

**lemma** *good2* [simp]: " $\llbracket A \notin \text{bad}; B \notin \text{bad} \rrbracket \Longrightarrow \text{good } \{\text{shrK } A, \text{shrK } B\}$ "  
 <proof>

## 33.2 Proofs About Guarded Messages

### 33.2.1 small hack

**lemma** *shrK\_is\_invKey\_shrK*: "*shrK A* = *invKey (shrK A)*"  
 <proof>

**lemmas** *shrK\_is\_invKey\_shrK\_substI* = *shrK\_is\_invKey\_shrK* [THEN *ssubst*]

**lemmas** *invKey\_invKey\_substI* = *invKey* [THEN *ssubst*]

**lemma** "Nonce *n*  $\in$  *parts* {*X*}  $\Longrightarrow$  *Crypt (shrK A) X*  $\in$  *guard n {shrK A}*"  
 <proof>

### 33.2.2 guardedness results on nonces

**lemma** *guard\_ciph* [simp]: "*shrK A*  $\in$  *Ks*  $\Longrightarrow$  *Ciph A X*  $\in$  *guard n Ks*"  
 <proof>

**lemma** *guardK\_ciph* [simp]: "*shrK A*  $\in$  *Ks*  $\Longrightarrow$  *Ciph A X*  $\in$  *guardK n Ks*"  
 <proof>

**lemma** *Guard\_init* [iff]: "*Guard n Ks (initState B)*"  
 <proof>

**lemma** *Guard\_knows\_max'*: "*Guard n Ks (knows\_max' C evs)*  
 $\Longrightarrow$  *Guard n Ks (knows\_max C evs)*"

*<proof>*

**lemma** *Nonce\_not\_used\_Guard\_spies* [dest]: "Nonce  $n \notin \text{used evs}$   
 $\implies \text{Guard } n \text{ Ks } (\text{spies evs})"$   
*<proof>*

**lemma** *Nonce\_not\_used\_Guard* [dest]: " $\llbracket \text{evs} \in p; \text{Nonce } n \notin \text{used evs};$   
 $\text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies \text{Guard } n \text{ Ks } (\text{knows } (\text{Friend } C) \text{ evs})"$   
*<proof>*

**lemma** *Nonce\_not\_used\_Guard\_max* [dest]: " $\llbracket \text{evs} \in p; \text{Nonce } n \notin \text{used evs};$   
 $\text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies \text{Guard } n \text{ Ks } (\text{knows\_max } (\text{Friend } C) \text{ evs})"$   
*<proof>*

**lemma** *Nonce\_not\_used\_Guard\_max'* [dest]: " $\llbracket \text{evs} \in p; \text{Nonce } n \notin \text{used evs};$   
 $\text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies \text{Guard } n \text{ Ks } (\text{knows\_max'} (\text{Friend } C) \text{ evs})"$   
*<proof>*

### 33.2.3 guardedness results on keys

**lemma** *GuardK\_init* [simp]: " $n \notin \text{range shrK} \implies \text{GuardK } n \text{ Ks } (\text{initState } B)"$   
*<proof>*

**lemma** *GuardK\_knows\_max'*: " $\llbracket \text{GuardK } n \text{ A } (\text{knows\_max'} C \text{ evs}); n \notin \text{range shrK} \rrbracket$   
 $\implies \text{GuardK } n \text{ A } (\text{knows\_max } C \text{ evs})"$   
*<proof>*

**lemma** *Key\_not\_used\_GuardK\_spies* [dest]: " $\text{Key } n \notin \text{used evs}$   
 $\implies \text{GuardK } n \text{ A } (\text{spies evs})"$   
*<proof>*

**lemma** *Key\_not\_used\_GuardK* [dest]: " $\llbracket \text{evs} \in p; \text{Key } n \notin \text{used evs};$   
 $\text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies \text{GuardK } n \text{ A } (\text{knows } (\text{Friend } C) \text{ evs})"$   
*<proof>*

**lemma** *Key\_not\_used\_GuardK\_max* [dest]: " $\llbracket \text{evs} \in p; \text{Key } n \notin \text{used evs};$   
 $\text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies \text{GuardK } n \text{ A } (\text{knows\_max } (\text{Friend } C) \text{ evs})"$   
*<proof>*

**lemma** *Key\_not\_used\_GuardK\_max'* [dest]: " $\llbracket \text{evs} \in p; \text{Key } n \notin \text{used evs};$   
 $\text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies \text{GuardK } n \text{ A } (\text{knows\_max'} (\text{Friend } C) \text{ evs})"$   
*<proof>*

### 33.2.4 regular protocols

**definition** *regular* :: "event list set  $\Rightarrow$  bool" where  
 $\text{"regular } p \equiv \forall \text{ evs A. evs} \in p \longrightarrow (\text{Key } (\text{shrK } A) \in \text{parts } (\text{spies evs})) = (A \in \text{bad})"$

**lemma** *shrK\_parts\_iff\_bad* [simp]: " $\llbracket \text{evs} \in p; \text{regular } p \rrbracket \implies$   
 $(\text{Key } (\text{shrK } A) \in \text{parts } (\text{spies evs})) = (A \in \text{bad})"$   
*<proof>*

**lemma** *shrK\_analz\_iff\_bad* [simp]: " $\llbracket \text{evs} \in p; \text{regular } p \rrbracket \implies$   
 $(\text{Key } (\text{shrK } A) \in \text{analz } (\text{spies evs})) = (A \in \text{bad})"$

*<proof>*

**lemma** *Guard\_Nonce\_analz*: "[[Guard n Ks (spies evs); evs ∈ p;  
shrK\_set Ks; good Ks; regular p]] ⇒ Nonce n ∉ analz (spies evs)"  
*<proof>*

**lemma** *GuardK\_Key\_analz*:  
 assumes "GuardK n Ks (spies evs)" "evs ∈ p" "shrK\_set Ks"  
 "good Ks" "regular p" "n ∉ range shrK"  
 shows "Key n ∉ analz (spies evs)"  
*<proof>*

**end**

## 34 Otway-Rees Protocol

**theory** *Guard\_OtwayRees* imports *Guard\_Shared* **begin**

### 34.1 messages used in the protocol

**abbreviation**

*nil* :: "msg" where  
 "nil == Number 0"

**abbreviation**

*or1* :: "agent ⇒ agent ⇒ nat ⇒ event" where  
 "or1 A B NA ==  
 Says A B {Nonce NA, Agent A, Agent B, Ciph A {Nonce NA, Agent A, Agent  
B}}"

**abbreviation**

*or1'* :: "agent ⇒ agent ⇒ agent ⇒ nat ⇒ msg ⇒ event" where  
 "or1' A' A B NA X == Says A' B {Nonce NA, Agent A, Agent B, X}"

**abbreviation**

*or2* :: "agent ⇒ agent ⇒ nat ⇒ nat ⇒ msg ⇒ event" where  
 "or2 A B NA NB X ==  
 Says B Server {Nonce NA, Agent A, Agent B, X,  
 Ciph B {Nonce NA, Nonce NB, Agent A, Agent B}}"

**abbreviation**

*or2'* :: "agent ⇒ agent ⇒ agent ⇒ nat ⇒ nat ⇒ event" where  
 "or2' B' A B NA NB ==  
 Says B' Server {Nonce NA, Agent A, Agent B,  
 Ciph A {Nonce NA, Agent A, Agent B},  
 Ciph B {Nonce NA, Nonce NB, Agent A, Agent B}}"

**abbreviation**

*or3* :: "agent ⇒ agent ⇒ nat ⇒ nat ⇒ key ⇒ event" where  
 "or3 A B NA NB K ==  
 Says Server B {Nonce NA, Ciph A {Nonce NA, Key K},  
 Ciph B {Nonce NB, Key K}}"

**abbreviation**

```

or3' :: "agent => msg => agent => agent => nat => nat => key => event" where
"or3' S Y A B NA NB K ==
  Says S B {Nonce NA, Y, Ciph B {Nonce NB, Key K}}"
```

**abbreviation**

```

or4 :: "agent => agent => nat => msg => event" where
"or4 A B NA X == Says B A {Nonce NA, X, nil}"
```

**abbreviation**

```

or4' :: "agent => agent => nat => key => event" where
"or4' B' A NA K == Says B' A {Nonce NA, Ciph A {Nonce NA, Key K}, nil}"
```

**34.2 definition of the protocol**

```

inductive_set or :: "event list set"
where
```

```

  Nil: "[] ∈ or"
```

```

  / Fake: "[[evs ∈ or; X ∈ synth (analz (spies evs))]] ⇒ Says Spy B X # evs
    ∈ or"
```

```

  / OR1: "[[evs1 ∈ or; Nonce NA ∉ used evs1]] ⇒ or1 A B NA # evs1 ∈ or"
```

```

  / OR2: "[[evs2 ∈ or; or1' A' A B NA X ∈ set evs2; Nonce NB ∉ used evs2]]
    ⇒ or2 A B NA NB X # evs2 ∈ or"
```

```

  / OR3: "[[evs3 ∈ or; or2' B' A B NA NB ∈ set evs3; Key K ∉ used evs3]]
    ⇒ or3 A B NA NB K # evs3 ∈ or"
```

```

  / OR4: "[[evs4 ∈ or; or2 A B NA NB X ∈ set evs4; or3' S Y A B NA NB K ∈ set
    evs4]]
    ⇒ or4 A B NA X # evs4 ∈ or"
```

**34.3 declarations for tactics**

```

declare knows_Spy_partsEs [elim]
declare Fake_parts_insert [THEN subsetD, dest]
declare initState.simps [simp del]
```

**34.4 general properties of or**

```

lemma or_has_no_Gets: "evs ∈ or ⇒ ∀ A X. Gets A X ∉ set evs"
<proof>
```

```

lemma or_is_Gets_correct [iff]: "Gets_correct or"
<proof>
```

```

lemma or_is_one_step [iff]: "one_step or"
<proof>
```

```

lemma or_has_only_Says' [rule_format]: "evs ∈ or ⇒
  ev ∈ set evs → (∃ A B X. ev=Says A B X)"
```

*<proof>*

**lemma** *or\_has\_only\_Says [iff]: "has\_only\_Says or"*

*<proof>*

### 34.5 or is regular

**lemma** *or1'\_parts\_spies [dest]: "or1' A' A B NA X ∈ set evs*

*⇒ X ∈ parts (spies evs)"*

*<proof>*

**lemma** *or2\_parts\_spies [dest]: "or2 A B NA NB X ∈ set evs*

*⇒ X ∈ parts (spies evs)"*

*<proof>*

**lemma** *or3\_parts\_spies [dest]: "Says S B {NA, Y, Ciph B {NB, K}} ∈ set evs*

*⇒ K ∈ parts (spies evs)"*

*<proof>*

**lemma** *or\_is\_regular [iff]: "regular or"*

*<proof>*

### 34.6 guardedness of KAB

**lemma** *Guard\_KAB [rule\_format]: "[evs ∈ or; A ∉ bad; B ∉ bad] ⇒*

*or3 A B NA NB K ∈ set evs → GuardK K {shrK A, shrK B} (spies evs)"*

*<proof>*

### 34.7 guardedness of NB

**lemma** *Guard\_NB [rule\_format]: "[evs ∈ or; B ∉ bad] ⇒*

*or2 A B NA NB X ∈ set evs → Guard NB {shrK B} (spies evs)"*

*<proof>*

**end**

## 35 Yahalom Protocol

**theory** *Guard\_Yahalom imports "../Shared" Guard\_Shared begin*

### 35.1 messages used in the protocol

**abbreviation** *(input)*

*ya1 :: "agent => agent => nat => event" where*

*"ya1 A B NA == Says A B {Agent A, Nonce NA}"*

**abbreviation** *(input)*

*ya1' :: "agent => agent => agent => nat => event" where*

*"ya1' A' A B NA == Says A' B {Agent A, Nonce NA}"*

**abbreviation** *(input)*

*ya2 :: "agent => agent => nat => nat => event" where*

*"ya2 A B NA NB == Says B Server {Agent B, Ciph B {Agent A, Nonce NA, Nonce NB}}"*

abbreviation (input)

```
ya2' :: "agent => agent => agent => nat => nat => event" where
  "ya2' B' A B NA NB == Says B' Server {Agent B, Ciph B {Agent A, Nonce NA,
  Nonce NB}}"
```

abbreviation (input)

```
ya3 :: "agent => agent => nat => nat => key => event" where
  "ya3 A B NA NB K ==
    Says Server A {Ciph A {Agent B, Key K, Nonce NA, Nonce NB},
    Ciph B {Agent A, Key K}}"
```

abbreviation (input)

```
ya3' :: "agent => msg => agent => agent => nat => nat => key => event" where
  "ya3' S Y A B NA NB K ==
    Says S A {Ciph A {Agent B, Key K, Nonce NA, Nonce NB}, Y}"
```

abbreviation (input)

```
ya4 :: "agent => agent => nat => nat => msg => event" where
  "ya4 A B K NB Y == Says A B {Y, Crypt K (Nonce NB)}"
```

abbreviation (input)

```
ya4' :: "agent => agent => nat => nat => msg => event" where
  "ya4' A' B K NB Y == Says A' B {Y, Crypt K (Nonce NB)}"
```

## 35.2 definition of the protocol

inductive\_set ya :: "event list set"

where

Nil: "[ ] ∈ ya"

/ Fake: "[ evs ∈ ya; X ∈ synth (analz (spies evs)) ] ⇒ Says Spy B X # evs ∈ ya"

/ YA1: "[ evs1 ∈ ya; Nonce NA ∉ used evs1 ] ⇒ ya1 A B NA # evs1 ∈ ya"

/ YA2: "[ evs2 ∈ ya; ya1' A' A B NA ∈ set evs2; Nonce NB ∉ used evs2 ]  
⇒ ya2 A B NA NB # evs2 ∈ ya"

/ YA3: "[ evs3 ∈ ya; ya2' B' A B NA NB ∈ set evs3; Key K ∉ used evs3 ]  
⇒ ya3 A B NA NB K # evs3 ∈ ya"

/ YA4: "[ evs4 ∈ ya; ya1 A B NA ∈ set evs4; ya3' S Y A B NA NB K ∈ set evs4 ]  
⇒ ya4 A B K NB Y # evs4 ∈ ya"

## 35.3 declarations for tactics

declare knows\_Spy\_partsEs [elim]

declare Fake\_parts\_insert [THEN subsetD, dest]

declare initState.simps [simp del]

### 35.4 general properties of ya

**lemma** *ya\_has\_no\_Gets*: "evs  $\in$  ya  $\implies \forall A X. \text{Gets } A X \notin \text{set evs}$ "  
 <proof>

**lemma** *ya\_is\_Gets\_correct* [iff]: "Gets\_correct ya"  
 <proof>

**lemma** *ya\_is\_one\_step* [iff]: "one\_step ya"  
 <proof>

**lemma** *ya\_has\_only\_Says'* [rule\_format]: "evs  $\in$  ya  $\implies$   
 ev  $\in$  set evs  $\longrightarrow (\exists A B X. \text{ev} = \text{Says } A B X)$ "  
 <proof>

**lemma** *ya\_has\_only\_Says* [iff]: "has\_only\_Says ya"  
 <proof>

**lemma** *ya\_is\_regular* [iff]: "regular ya"  
 <proof>

### 35.5 guardedness of KAB

**lemma** *Guard\_KAB* [rule\_format]: "[[evs  $\in$  ya;  $A \notin \text{bad}$ ;  $B \notin \text{bad}$ ]]  $\implies$   
 ya3 A B NA NB K  $\in$  set evs  $\longrightarrow \text{GuardK } K \{ \text{shrK } A, \text{shrK } B \} (\text{spies evs})$ "  
 <proof>

### 35.6 session keys are not symmetric keys

**lemma** *KAB\_isnt\_shrK* [rule\_format]: "evs  $\in$  ya  $\implies$   
 ya3 A B NA NB K  $\in$  set evs  $\longrightarrow K \notin \text{range shrK}$ "  
 <proof>

**lemma** *ya3\_shrK*: "evs  $\in$  ya  $\implies$  ya3 A B NA NB (shrK C)  $\notin$  set evs"  
 <proof>

### 35.7 ya2' implies ya1'

**lemma** *ya2'\_parts\_imp\_ya1'\_parts* [rule\_format]:  
 "[[evs  $\in$  ya;  $B \notin \text{bad}$ ]]  $\implies$   
 Ciph B {Agent A, Nonce NA, Nonce NB}  $\in$  parts (spies evs)  $\longrightarrow$   
 {Agent A, Nonce NA}  $\in$  spies evs"  
 <proof>

**lemma** *ya2'\_imp\_ya1'\_parts*: "[[ya2' B' A B NA NB  $\in$  set evs; evs  $\in$  ya;  $B \notin$   
 bad]]  $\implies$   
 {Agent A, Nonce NA}  $\in$  spies evs"  
 <proof>

### 35.8 uniqueness of NB

**lemma** *NB\_is\_uniq\_in\_ya2'\_parts* [rule\_format]: "[[evs  $\in$  ya;  $B \notin \text{bad}$ ;  $B' \notin$   
 bad]]  $\implies$   
 Ciph B {Agent A, Nonce NA, Nonce NB}  $\in$  parts (spies evs)  $\longrightarrow$   
 Ciph B' {Agent A', Nonce NA', Nonce NB}  $\in$  parts (spies evs)  $\longrightarrow$

$A=A' \wedge B=B' \wedge NA=NA'$   
 $\langle proof \rangle$

**lemma** *NB\_is\_uniq\_in\_ya2'*: " $\llbracket ya2' \ C \ A \ B \ NA \ NB \in \text{set evs};$   
 $ya2' \ C' \ A' \ B' \ NA' \ NB \in \text{set evs}; \text{evs} \in ya; B \notin \text{bad}; B' \notin \text{bad} \rrbracket$   
 $\implies A=A' \wedge B=B' \wedge NA=NA'$ "  
 $\langle proof \rangle$

### 35.9 ya3' implies ya2'

**lemma** *ya3'\_parts\_imp\_ya2'\_parts* [rule\_format]: " $\llbracket \text{evs} \in ya; A \notin \text{bad} \rrbracket \implies$   
 $\text{Ciph } A \ \{Agent \ B, \text{Key } K, \text{Nonce } NA, \text{Nonce } NB\} \in \text{parts}(\text{spies evs})$   
 $\longrightarrow \text{Ciph } B \ \{Agent \ A, \text{Nonce } NA, \text{Nonce } NB\} \in \text{parts}(\text{spies evs})$ "  
 $\langle proof \rangle$

**lemma** *ya3'\_parts\_imp\_ya2'* [rule\_format]: " $\llbracket \text{evs} \in ya; A \notin \text{bad} \rrbracket \implies$   
 $\text{Ciph } A \ \{Agent \ B, \text{Key } K, \text{Nonce } NA, \text{Nonce } NB\} \in \text{parts}(\text{spies evs})$   
 $\longrightarrow (\exists B'. ya2' \ B' \ A \ B \ NA \ NB \in \text{set evs})$ "  
 $\langle proof \rangle$

**lemma** *ya3'\_imp\_ya2'*: " $\llbracket ya3' \ S \ Y \ A \ B \ NA \ NB \ K \in \text{set evs}; \text{evs} \in ya; A \notin \text{bad} \rrbracket$   
 $\implies (\exists B'. ya2' \ B' \ A \ B \ NA \ NB \in \text{set evs})$ "  
 $\langle proof \rangle$

### 35.10 ya3' implies ya3

**lemma** *ya3'\_parts\_imp\_ya3* [rule\_format]: " $\llbracket \text{evs} \in ya; A \notin \text{bad} \rrbracket \implies$   
 $\text{Ciph } A \ \{Agent \ B, \text{Key } K, \text{Nonce } NA, \text{Nonce } NB\} \in \text{parts}(\text{spies evs})$   
 $\longrightarrow ya3 \ A \ B \ NA \ NB \ K \in \text{set evs}$ "  
 $\langle proof \rangle$

**lemma** *ya3'\_imp\_ya3*: " $\llbracket ya3' \ S \ Y \ A \ B \ NA \ NB \ K \in \text{set evs}; \text{evs} \in ya; A \notin \text{bad} \rrbracket$   
 $\implies ya3 \ A \ B \ NA \ NB \ K \in \text{set evs}$ "  
 $\langle proof \rangle$

### 35.11 guardedness of NB

**definition** *ya\_keys* :: "agent  $\Rightarrow$  agent  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  event list  $\Rightarrow$  key set"  
**where**  
 $"ya\_keys \ A \ B \ NA \ NB \ evs \equiv \{shrK \ A, shrK \ B\} \cup \{K. ya3 \ A \ B \ NA \ NB \ K \in \text{set evs}\}"$

**lemma** *Guard\_NB* [rule\_format]: " $\llbracket \text{evs} \in ya; A \notin \text{bad}; B \notin \text{bad} \rrbracket \implies$   
 $ya2 \ A \ B \ NA \ NB \in \text{set evs} \longrightarrow \text{Guard } NB \ (ya\_keys \ A \ B \ NA \ NB \ evs) \ (\text{spies evs})$ "  
 $\langle proof \rangle$

**end**

## 36 Blanqui's "guard" concept: protocol-independent secrecy

**theory** *Auth\_Guard\_Shared*  
**imports**  
*Guard\_OtwayRees*



```

    Guard_Yahalom
begin
end

```

```

theory Guard_Public imports Guard "../Public" Extensions begin

```

### 36.1 Extensions to Theory Public

```

declare initState.simps [simp del]

```

#### 36.1.1 signature

```

definition sign :: "agent => msg => msg" where
"sign A X == {Agent A, X, Crypt (priK A) (Hash X)}"

lemma sign_inj [iff]: "(sign A X = sign A' X') = (A=A' & X=X')"
<proof>

```

#### 36.1.2 agent associated to a key

```

definition agt :: "key => agent" where
"agt K == SOME A. K = priK A | K = pubK A"

lemma agt_priK [simp]: "agt (priK A) = A"
<proof>

lemma agt_pubK [simp]: "agt (pubK A) = A"
<proof>

```

#### 36.1.3 basic facts about initState

```

lemma no_Crypt_in_parts_init [simp]: "Crypt K X ∉ parts (initState A)"
<proof>

lemma no_Crypt_in_analz_init [simp]: "Crypt K X ∉ analz (initState A)"
<proof>

lemma no_priK_in_analz_init [simp]: "A ∉ bad
⇒ Key (priK A) ∉ analz (initState Spy)"
<proof>

lemma priK_notin_initState_Friend [simp]: "A ≠ Friend C
⇒ Key (priK A) ∉ parts (initState (Friend C))"
<proof>

lemma keyset_init [iff]: "keyset (initState A)"
<proof>

```

#### 36.1.4 sets of private keys

```

definition priK_set :: "key set => bool" where
"priK_set Ks ≡ ∀K. K ∈ Ks → (∃A. K = priK A)"

```

**lemma** *in\_priK\_set*: " $\llbracket \text{priK\_set } Ks; K \in Ks \rrbracket \implies \exists A. K = \text{priK } A$ "  
 $\langle \text{proof} \rangle$

**lemma** *priK\_set1 [iff]*: " $\text{priK\_set } \{\text{priK } A\}$ "  
 $\langle \text{proof} \rangle$

**lemma** *priK\_set2 [iff]*: " $\text{priK\_set } \{\text{priK } A, \text{priK } B\}$ "  
 $\langle \text{proof} \rangle$

### 36.1.5 sets of good keys

**definition** *good* :: " $\text{key set} \Rightarrow \text{bool}$ " **where**  
 $\text{"good } Ks == \forall K. K \in Ks \longrightarrow \text{agt } K \notin \text{bad}"$

**lemma** *in\_good*: " $\llbracket \text{good } Ks; K \in Ks \rrbracket \implies \text{agt } K \notin \text{bad}$ "  
 $\langle \text{proof} \rangle$

**lemma** *good1 [simp]*: " $A \notin \text{bad} \implies \text{good } \{\text{priK } A\}$ "  
 $\langle \text{proof} \rangle$

**lemma** *good2 [simp]*: " $\llbracket A \notin \text{bad}; B \notin \text{bad} \rrbracket \implies \text{good } \{\text{priK } A, \text{priK } B\}$ "  
 $\langle \text{proof} \rangle$

### 36.1.6 greatest nonce used in a trace, 0 if there is no nonce

**primrec** *greatest* :: " $\text{event list} \Rightarrow \text{nat}$ "  
**where**  
 $\text{"greatest } [] = 0"$   
 $| \text{"greatest } (\text{ev} \# \text{evs}) = \max (\text{greatest\_msg } (\text{msg ev})) (\text{greatest evs})"$

**lemma** *greatest\_is\_greatest*: " $\text{Nonce } n \in \text{used evs} \implies n \leq \text{greatest evs}$ "  
 $\langle \text{proof} \rangle$

### 36.1.7 function giving a new nonce

**definition** *new* :: " $\text{event list} \Rightarrow \text{nat}$ " **where**  
 $\text{"new evs} \equiv \text{Suc } (\text{greatest evs})"$

**lemma** *new\_isnt\_used [iff]*: " $\text{Nonce } (\text{new evs}) \notin \text{used evs}$ "  
 $\langle \text{proof} \rangle$

## 36.2 Proofs About Guarded Messages

### 36.2.1 small hack necessary because priK is defined as the inverse of pubK

**lemma** *pubK\_is\_invKey\_priK*: " $\text{pubK } A = \text{invKey } (\text{priK } A)$ "  
 $\langle \text{proof} \rangle$

**lemmas** *pubK\_is\_invKey\_priK\_substI* = *pubK\_is\_invKey\_priK* [THEN *ssubst*]

**lemmas** *invKey\_invKey\_substI* = *invKey* [THEN *ssubst*]

**lemma** " $\text{Nonce } n \in \text{parts } \{X\} \implies \text{Crypt } (\text{pubK } A) X \in \text{guard } n \{\text{priK } A\}$ "  
 $\langle \text{proof} \rangle$

### 36.2.2 guardedness results

**lemma** *sign\_guard* [intro]: " $X \in \text{guard } n \text{ } Ks \implies \text{sign } A \text{ } X \in \text{guard } n \text{ } Ks$ "  
 <proof>

**lemma** *Guard\_init* [iff]: " $\text{Guard } n \text{ } Ks \text{ (initState } B)$ "  
 <proof>

**lemma** *Guard\_knows\_max'*: " $\text{Guard } n \text{ } Ks \text{ (knows\_max' } C \text{ evs)}$   
 $\implies \text{Guard } n \text{ } Ks \text{ (knows\_max } C \text{ evs)}$ "  
 <proof>

**lemma** *Nonce\_not\_used\_Guard\_spies* [dest]: " $\text{Nonce } n \notin \text{used evs}$   
 $\implies \text{Guard } n \text{ } Ks \text{ (spies evs)}$ "  
 <proof>

**lemma** *Nonce\_not\_used\_Guard* [dest]: " $\llbracket \text{evs} \in p; \text{Nonce } n \notin \text{used evs};$   
 $\text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies \text{Guard } n \text{ } Ks \text{ (knows (Friend } C) \text{ evs)}$ "  
 <proof>

**lemma** *Nonce\_not\_used\_Guard\_max* [dest]: " $\llbracket \text{evs} \in p; \text{Nonce } n \notin \text{used evs};$   
 $\text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies \text{Guard } n \text{ } Ks \text{ (knows\_max (Friend } C) \text{ evs)}$ "  
 <proof>

**lemma** *Nonce\_not\_used\_Guard\_max'* [dest]: " $\llbracket \text{evs} \in p; \text{Nonce } n \notin \text{used evs};$   
 $\text{Gets\_correct } p; \text{one\_step } p \rrbracket \implies \text{Guard } n \text{ } Ks \text{ (knows\_max' (Friend } C) \text{ evs)}$ "  
 <proof>

### 36.2.3 regular protocols

**definition** *regular* :: "event list set  $\Rightarrow$  bool" where  
 " $\text{regular } p \equiv \forall \text{evs } A. \text{evs} \in p \longrightarrow (\text{Key (priK } A) \in \text{parts (spies evs)}) = (A \in \text{bad})$ "

**lemma** *priK\_parts\_iff\_bad* [simp]: " $\llbracket \text{evs} \in p; \text{regular } p \rrbracket \implies$   
 $(\text{Key (priK } A) \in \text{parts (spies evs)}) = (A \in \text{bad})$ "  
 <proof>

**lemma** *priK\_analz\_iff\_bad* [simp]: " $\llbracket \text{evs} \in p; \text{regular } p \rrbracket \implies$   
 $(\text{Key (priK } A) \in \text{analz (spies evs)}) = (A \in \text{bad})$ "  
 <proof>

**lemma** *Guard\_Nonce\_analz*: " $\llbracket \text{Guard } n \text{ } Ks \text{ (spies evs); evs} \in p;$   
 $\text{priK\_set } Ks; \text{good } Ks; \text{regular } p \rrbracket \implies \text{Nonce } n \notin \text{analz (spies evs)}$ "  
 <proof>

end

## 37 Lists of Messages and Lists of Agents

theory *List\_Msg* imports *Extensions* begin

### 37.1 Implementation of Lists by Messages

#### 37.1.1 nil is represented by any message which is not a pair

abbreviation (input)

```
cons :: "msg => msg => msg" where
  "cons x l == {x,l}"
```

#### 37.1.2 induction principle

```
lemma lmsg_induct: "[[!x. not_MPair x ==> P x; !x l. P l ==> P (cons x
l)]]
==> P l"
<proof>
```

#### 37.1.3 head

```
primrec head :: "msg => msg" where
  "head (cons x l) = x"
```

#### 37.1.4 tail

```
primrec tail :: "msg => msg" where
  "tail (cons x l) = l"
```

#### 37.1.5 length

```
fun len :: "msg => nat" where
  "len (cons x l) = Suc (len l)" |
  "len other = 0"
```

```
lemma len_not_empty: "n < len l ==> ∃ x l'. l = cons x l'"
<proof>
```

#### 37.1.6 membership

```
fun isin :: "msg * msg => bool" where
  "isin (x, cons y l) = (x=y | isin (x,l))" |
  "isin (x, other) = False"
```

#### 37.1.7 delete an element

```
fun del :: "msg * msg => msg" where
  "del (x, cons y l) = (if x=y then l else cons y (del (x,l)))" |
  "del (x, other) = other"
```

```
lemma notin_del [simp]: "~ isin (x,l) ==> del (x,l) = l"
<proof>
```

```
lemma isin_del [rule_format]: "isin (y, del (x,l)) --> isin (y,l)"
<proof>
```

#### 37.1.8 concatenation

```
fun app :: "msg * msg => msg" where
  "app (cons x l, l') = cons x (app (l,l'))" |
  "app (other, l') = l'"
```

```
lemma isin_app [iff]: "isin (x, app(l,l')) = (isin (x,l) | isin (x,l'))"
<proof>
```

### 37.1.9 replacement

```
fun repl :: "msg * nat * msg => msg" where
  "repl (cons x l, Suc i, x') = cons x (repl (l,i,x'))" |
  "repl (cons x l, 0, x') = cons x' l" |
  "repl (other, i, M') = other"
```

#### 37.1.10 ith element

```
fun ith :: "msg * nat => msg" where
  "ith (cons x l, Suc i) = ith (l,i)" |
  "ith (cons x l, 0) = x" |
  "ith (other, i) = other"
```

```
lemma ith_head: "0 < len l ==> ith (l,0) = head l"
<proof>
```

#### 37.1.11 insertion

```
fun ins :: "msg * nat * msg => msg" where
  "ins (cons x l, Suc i, y) = cons x (ins (l,i,y))" |
  "ins (l, 0, y) = cons y l"
```

```
lemma ins_head [simp]: "ins (l,0,y) = cons y l"
<proof>
```

#### 37.1.12 truncation

```
fun trunc :: "msg * nat => msg" where
  "trunc (l,0) = l" |
  "trunc (cons x l, Suc i) = trunc (l,i)"
```

```
lemma trunc_zero [simp]: "trunc (l,0) = l"
<proof>
```

## 37.2 Agent Lists

### 37.2.1 set of well-formed agent-list messages

#### abbreviation

```
nil :: msg where
  "nil == Number 0"
```

```
inductive_set agl :: "msg set"
```

```
where
```

```
  Nil[intro]: "nil ∈ agl"
  | Cons[intro]: "[[A ∈ agent; I ∈ agl]] ==> cons (Agent A) I ∈ agl"
```

### 37.2.2 basic facts about agent lists

```
lemma del_in_agl [intro]: "I ∈ agl ==> del (a,I) ∈ agl"
<proof>
```

**lemma** *app\_in\_agl* [*intro*]: " $\llbracket I \in \text{agl}; J \in \text{agl} \rrbracket \implies \text{app } (I, J) \in \text{agl}$ "  
 <proof>

**lemma** *no\_Key\_in\_agl*: " $I \in \text{agl} \implies \text{Key } K \notin \text{parts } \{I\}$ "  
 <proof>

**lemma** *no\_Nonce\_in\_agl*: " $I \in \text{agl} \implies \text{Nonce } n \notin \text{parts } \{I\}$ "  
 <proof>

**lemma** *no\_Key\_in\_appdel*: " $\llbracket I \in \text{agl}; J \in \text{agl} \rrbracket \implies$   
 $\text{Key } K \notin \text{parts } \{\text{app } (J, \text{del } (\text{Agent } B, I))\}$ "  
 <proof>

**lemma** *no\_Nonce\_in\_appdel*: " $\llbracket I \in \text{agl}; J \in \text{agl} \rrbracket \implies$   
 $\text{Nonce } n \notin \text{parts } \{\text{app } (J, \text{del } (\text{Agent } B, I))\}$ "  
 <proof>

**lemma** *no\_Crypt\_in\_agl*: " $I \in \text{agl} \implies \text{Crypt } K X \notin \text{parts } \{I\}$ "  
 <proof>

**lemma** *no\_Crypt\_in\_appdel*: " $\llbracket I \in \text{agl}; J \in \text{agl} \rrbracket \implies$   
 $\text{Crypt } K X \notin \text{parts } \{\text{app } (J, \text{del } (\text{Agent } B, I))\}$ "  
 <proof>

**end**

## 38 Protocol P1

**theory** *P1* **imports** "../Public" *Guard\_Public List\_Msg* **begin**

### 38.1 Protocol Definition

**38.1.1 offer chaining: B chains his offer for A with the head offer of L for sending it to C**

**definition** *chain* :: " $\text{agent} \Rightarrow \text{nat} \Rightarrow \text{agent} \Rightarrow \text{msg} \Rightarrow \text{agent} \Rightarrow \text{msg}$ " **where**  
 "*chain* *B ofr A L C* ==  
 let *m1* = *Crypt* (*pubK A*) (*Nonce ofr*) in  
 let *m2* = *Hash* {*head L*, *Agent C*} in  
*sign B* {*m1, m2*}"

**declare** *Let\_def* [*simp*]

**lemma** *chain\_inj* [*iff*]: " $(\text{chain } B \text{ ofr } A \text{ L } C = \text{chain } B' \text{ ofr}' A' \text{ L}' C') \implies$   
 $(B=B' \ \& \ \text{ofr}=\text{ofr}' \ \& \ A=A' \ \& \ \text{head } L = \text{head } L' \ \& \ C=C')$ "  
 <proof>

**lemma** *Nonce\_in\_chain* [*iff*]: " $\text{Nonce ofr} \in \text{parts } \{\text{chain } B \text{ ofr } A \text{ L } C\}$ "  
 <proof>

**38.1.2 agent whose key is used to sign an offer**

**fun** *shop* :: " $\text{msg} \Rightarrow \text{msg}$ " **where**

"shop  $\{B, X, \text{Crypt } K \ H\} = \text{Agent } (\text{agt } K)"$

**lemma** shop\_chain [simp]: "shop (chain B ofr A L C) = Agent B"  
 <proof>

### 38.1.3 nonce used in an offer

**fun** nonce :: "msg => msg" **where**  
 "nonce  $\{B, \{ \text{Crypt } K \text{ ofr, m2} \}, \text{CryptH} \} = \text{ofr}"$

**lemma** nonce\_chain [simp]: "nonce (chain B ofr A L C) = Nonce ofr"  
 <proof>

### 38.1.4 next shop

**fun** next\_shop :: "msg => agent" **where**  
 "next\_shop  $\{B, \{m1, \text{Hash } \{ \text{headL, Agent } C \} \}, \text{CryptH} \} = C"$

**lemma** next\_shop\_chain [iff]: "next\_shop (chain B ofr A L C) = C"  
 <proof>

### 38.1.5 anchor of the offer list

**definition** anchor :: "agent => nat => agent => msg" **where**  
 "anchor A n B == chain A n A (cons nil nil) B"

**lemma** anchor\_inj [iff]: "(anchor A n B = anchor A' n' B')  
 = (A=A' & n=n' & B=B')"  
 <proof>

**lemma** Nonce\_in\_anchor [iff]: "Nonce n  $\in$  parts {anchor A n B}"  
 <proof>

**lemma** shop\_anchor [simp]: "shop (anchor A n B) = Agent A"  
 <proof>

**lemma** nonce\_anchor [simp]: "nonce (anchor A n B) = Nonce n"  
 <proof>

**lemma** next\_shop\_anchor [iff]: "next\_shop (anchor A n B) = B"  
 <proof>

### 38.1.6 request event

**definition** reqm :: "agent => nat => nat => msg => agent => msg" **where**  
 "reqm A r n I B ==  $\{ \text{Agent } A, \text{Number } r, \text{cons } (\text{Agent } A) (\text{cons } (\text{Agent } B) I), \text{cons } (\text{anchor } A \ n \ B) \text{ nil} \}$ "

**lemma** reqm\_inj [iff]: "(reqm A r n I B = reqm A' r' n' I' B')  
 = (A=A' & r=r' & n=n' & I=I' & B=B')"  
 <proof>

**lemma** Nonce\_in\_reqm [iff]: "Nonce n  $\in$  parts {reqm A r n I B}"  
 <proof>

**definition** req :: "agent => nat => nat => msg => agent => event" where  
 "req A r n I B == Says A B (reqm A r n I B)"

**lemma** req\_inj [iff]: "(req A r n I B = req A' r' n' I' B')  
 = (A=A' & r=r' & n=n' & I=I' & B=B')"  
 <proof>

### 38.1.7 propose event

**definition** prom :: "agent => nat => agent => nat => msg => msg =>  
 msg => agent => msg" where  
 "prom B ofr A r I L J C == {Agent A, Number r,  
 app (J, del (Agent B, I)), cons (chain B ofr A L C) L}"

**lemma** prom\_inj [dest]: "prom B ofr A r I L J C  
 = prom B' ofr' A' r' I' L' J' C'  
 ==> B=B' & ofr=ofr' & A=A' & r=r' & L=L' & C=C'"  
 <proof>

**lemma** Nonce\_in\_prom [iff]: "Nonce ofr ∈ parts {prom B ofr A r I L J C}"  
 <proof>

**definition** pro :: "agent => nat => agent => nat => msg => msg =>  
 msg => agent => event" where  
 "pro B ofr A r I L J C == Says B C (prom B ofr A r I L J C)"

**lemma** pro\_inj [dest]: "pro B ofr A r I L J C = pro B' ofr' A' r' I' L' J'  
 C'  
 ==> B=B' & ofr=ofr' & A=A' & r=r' & L=L' & C=C'"  
 <proof>

### 38.1.8 protocol

**inductive\_set** p1 :: "event list set"  
**where**

Nil: "[] ∈ p1"

/ Fake: "[[evsf ∈ p1; X ∈ synth (analz (spies evsf))]] ==> Says Spy B X # evsf  
 ∈ p1"

/ Request: "[[evsr ∈ p1; Nonce n ∉ used evsr; I ∈ agl]] ==> req A r n I B #  
 evsr ∈ p1"

/ Propose: "[[evsp ∈ p1; Says A' B {Agent A, Number r, I, cons M L} ∈ set evsp;  
 I ∈ agl; J ∈ agl; isin (Agent C, app (J, del (Agent B, I)))];  
 Nonce ofr ∉ used evsp] ==> pro B ofr A r I (cons M L) J C # evsp ∈ p1"

### 38.1.9 Composition of Traces

**lemma** "evs' ∈ p1 ==>  
 evs ∈ p1 ∧ (∀n. Nonce n ∈ used evs' → Nonce n ∉ used evs) →  
 evs' @ evs ∈ p1"  
 <proof>



### 38.1.10 Valid Offer Lists

```

inductive_set
  valid :: "agent  $\Rightarrow$  nat  $\Rightarrow$  agent  $\Rightarrow$  msg set"
  for A :: agent and n :: nat and B :: agent
where
  Request [intro]: "cons (anchor A n B) nil  $\in$  valid A n B"

  | Propose [intro]: "L  $\in$  valid A n B
 $\Rightarrow$  cons (chain (next_shop (head L)) ofr A L C) L  $\in$  valid A n B"

```

#### 38.1.11 basic properties of valid

```

lemma valid_not_empty: "L  $\in$  valid A n B  $\Rightarrow \exists M L'. L = \text{cons } M L' "$ "
<proof>

```

```

lemma valid_pos_len: "L  $\in$  valid A n B  $\Rightarrow 0 < \text{len } L "$ "
<proof>

```

#### 38.1.12 offers of an offer list

```

definition offer_nonces :: "msg  $\Rightarrow$  msg set" where
"offer_nonces L  $\equiv \{X. X \in \text{parts } \{L\} \wedge (\exists n. X = \text{Nonce } n)\} "$ 

```

#### 38.1.13 the originator can get the offers

```

lemma "L  $\in$  valid A n B  $\Rightarrow \text{offer\_nonces } L \subseteq \text{analz } (\text{insert } L (\text{initState } A)) "$ "
<proof>

```

#### 38.1.14 list of offers

```

fun offers :: "msg  $\Rightarrow$  msg" where
"offers (cons M L) = cons {shop M, nonce M} (offers L)" |
"offers other = nil"

```

#### 38.1.15 list of agents whose keys are used to sign a list of offers

```

fun shops :: "msg  $\Rightarrow$  msg" where
"shops (cons M L) = cons (shop M) (shops L)" |
"shops other = other"

```

```

lemma shops_in_agl: "L  $\in$  valid A n B  $\Rightarrow \text{shops } L \in \text{agl} "$ "
<proof>

```

#### 38.1.16 builds a trace from an itinerary

```

fun offer_list :: "agent  $\times$  nat  $\times$  agent  $\times$  msg  $\times$  nat  $\Rightarrow$  msg" where
"offer_list (A,n,B,nil,ofr) = cons (anchor A n B) nil" |
"offer_list (A,n,B,cons (Agent C) I,ofr) = (
  let L = offer_list (A,n,B,I,Suc ofr) in
  cons (chain (next_shop (head L)) ofr A L C) L)"

```

```

lemma "I  $\in$  agl  $\Rightarrow \forall \text{ofr. offer\_list } (A,n,B,I,\text{ofr}) \in \text{valid } A \text{ n } B "$ "
<proof>

```

```

fun trace :: "agent  $\times$  nat  $\times$  agent  $\times$  nat  $\times$  msg  $\times$  msg  $\times$  msg
 $\Rightarrow$  event list" where
  "trace (B,ofr,A,r,I,L,nil) = []" |
  "trace (B,ofr,A,r,I,L,cons (Agent D) K) = (
    let C = (if K=nil then B else agt_nb (head K)) in
    let I' = (if K=nil then cons (Agent A) (cons (Agent B) I)
              else cons (Agent A) (app (I, cons (head K) nil))) in
    let I'' = app (I, cons (head K) nil) in
    pro C (Suc ofr) A r I' L nil D
    # trace (B,Suc ofr,A,r,I'',tail L,K))"

```

```

definition trace' :: "agent  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  msg  $\Rightarrow$  agent  $\Rightarrow$  nat  $\Rightarrow$  event
list" where
  "trace' A r n I B ofr  $\equiv$  (
    let AI = cons (Agent A) I in
    let L = offer_list (A,n,B,AI,ofr) in
    trace (B,ofr,A,r,nil,L,AI))"

```

```

declare trace'_def [simp]

```

### 38.1.17 there is a trace in which the originator receives a valid answer

```

lemma p1_not_empty: "evs  $\in$  p1  $\implies$  req A r n I B  $\in$  set evs  $\longrightarrow$ 
  ( $\exists$  evs'. evs'  $\in$  p1  $\wedge$  pro B' ofr A r I' L J A  $\in$  set evs'  $\wedge$  L  $\in$  valid
  A n B)"
  <proof>

```

## 38.2 properties of protocol P1

publicly verifiable forward integrity: anyone can verify the validity of an offer list

### 38.2.1 strong forward integrity: except the last one, no offer can be modified

```

lemma strong_forward_integrity: " $\forall$  L. Suc i < len L
 $\longrightarrow$  L  $\in$  valid A n B  $\wedge$  repl (L,Suc i,M)  $\in$  valid A n B  $\longrightarrow$  M = ith (L,Suc i)"
  <proof>

```

### 38.2.2 insertion resilience: except at the beginning, no offer can be inserted

```

lemma chain_isnt_head [simp]: "L  $\in$  valid A n B  $\implies$ 
head L  $\neq$  chain (next_shop (head L)) ofr A L C"
  <proof>

```

```

lemma insertion_resilience: " $\forall$  L. L  $\in$  valid A n B  $\longrightarrow$  Suc i < len L
 $\longrightarrow$  ins (L,Suc i,M)  $\notin$  valid A n B"
  <proof>

```

### 38.2.3 truncation resilience: only shop i can truncate at offer i

```

lemma truncation_resilience: " $\forall$  L. L  $\in$  valid A n B  $\longrightarrow$  Suc i < len L

```

$\longrightarrow \text{cons } M (\text{trunc } (L, \text{Suc } i)) \in \text{valid } A \text{ n } B \longrightarrow \text{shop } M = \text{shop } (\text{ith } (L, i))"$   
 $\langle \text{proof} \rangle$

#### 38.2.4 declarations for tactics

```
declare knows_Spy_partsEs [elim]
declare Fake_parts_insert [THEN subsetD, dest]
declare initState.simps [simp del]
```

#### 38.2.5 get components of a message

**lemma** *get\_ML* [dest]: "Says A' B {A,r,I,M,L}  $\in$  set evs  $\implies$   
 $M \in \text{parts } (\text{spies evs}) \wedge L \in \text{parts } (\text{spies evs})"$   
 $\langle \text{proof} \rangle$

#### 38.2.6 general properties of p1

**lemma** *reqm\_neq\_prom* [iff]:  
 $"\text{reqm } A \text{ r n } I \text{ B} \neq \text{prom } B' \text{ ofr } A' \text{ r}' \text{ I}' (\text{cons } M \text{ L}) \text{ J } C"$   
 $\langle \text{proof} \rangle$

**lemma** *prom\_neq\_reqm* [iff]:  
 $"\text{prom } B' \text{ ofr } A' \text{ r}' \text{ I}' (\text{cons } M \text{ L}) \text{ J } C \neq \text{reqm } A \text{ r n } I \text{ B}"$   
 $\langle \text{proof} \rangle$

**lemma** *req\_neq\_pro* [iff]:  $"\text{req } A \text{ r n } I \text{ B} \neq \text{pro } B' \text{ ofr } A' \text{ r}' \text{ I}' (\text{cons } M \text{ L}) \text{ J } C"$   
 $\langle \text{proof} \rangle$

**lemma** *pro\_neq\_req* [iff]:  $"\text{pro } B' \text{ ofr } A' \text{ r}' \text{ I}' (\text{cons } M \text{ L}) \text{ J } C \neq \text{req } A \text{ r n } I \text{ B}"$   
 $\langle \text{proof} \rangle$

**lemma** *p1\_has\_no\_Gets*:  $"\text{evs} \in p1 \implies \forall A \text{ X. Gets } A \text{ X} \notin \text{set evs}"$   
 $\langle \text{proof} \rangle$

**lemma** *p1\_is\_Gets\_correct* [iff]:  $"\text{Gets\_correct } p1"$   
 $\langle \text{proof} \rangle$

**lemma** *p1\_is\_one\_step* [iff]:  $"\text{one\_step } p1"$   
 $\langle \text{proof} \rangle$

**lemma** *p1\_has\_only\_Says'* [rule\_format]:  $"\text{evs} \in p1 \implies$   
 $\text{ev} \in \text{set evs} \longrightarrow (\exists A \text{ B } X. \text{ev} = \text{Says } A \text{ B } X)"$   
 $\langle \text{proof} \rangle$

**lemma** *p1\_has\_only\_Says* [iff]:  $"\text{has\_only\_Says } p1"$   
 $\langle \text{proof} \rangle$

**lemma** *p1\_is\_regular* [iff]:  $"\text{regular } p1"$   
 $\langle \text{proof} \rangle$

#### 38.2.7 private keys are safe

**lemma** *priK\_parts\_Friend\_imp\_bad* [rule\_format, dest]:

" $\llbracket \text{evs} \in p1; \text{Friend } B \neq A \rrbracket$   
 $\implies (\text{Key } (\text{priK } A) \in \text{parts } (\text{knows } (\text{Friend } B) \text{ evs})) \longrightarrow (A \in \text{bad})$ "  
 $\langle \text{proof} \rangle$

**lemma** *priK\_analz\_Friend\_imp\_bad* [rule\_format,dest]:  
 " $\llbracket \text{evs} \in p1; \text{Friend } B \neq A \rrbracket$   
 $\implies (\text{Key } (\text{priK } A) \in \text{analz } (\text{knows } (\text{Friend } B) \text{ evs})) \longrightarrow (A \in \text{bad})$ "  
 $\langle \text{proof} \rangle$

**lemma** *priK\_notin\_knows\_max\_Friend*: " $\llbracket \text{evs} \in p1; A \notin \text{bad}; A \neq \text{Friend } C \rrbracket$   
 $\implies \text{Key } (\text{priK } A) \notin \text{analz } (\text{knows\_max } (\text{Friend } C) \text{ evs})$ "  
 $\langle \text{proof} \rangle$

### 38.2.8 general guardedness properties

**lemma** *agl\_guard* [intro]: " $I \in \text{agl} \implies I \in \text{guard } n \text{ Ks}$ "  
 $\langle \text{proof} \rangle$

**lemma** *Says\_to\_knows\_max'\_guard*: " $\llbracket \text{Says } A' \ C \ \{A'', r, I, L\} \in \text{set evs};$   
 $\text{Guard } n \text{ Ks } (\text{knows\_max}' \ C \ \text{evs}) \rrbracket \implies L \in \text{guard } n \text{ Ks}$ "  
 $\langle \text{proof} \rangle$

**lemma** *Says\_from\_knows\_max'\_guard*: " $\llbracket \text{Says } C \ A' \ \{A'', r, I, L\} \in \text{set evs};$   
 $\text{Guard } n \text{ Ks } (\text{knows\_max}' \ C \ \text{evs}) \rrbracket \implies L \in \text{guard } n \text{ Ks}$ "  
 $\langle \text{proof} \rangle$

**lemma** *Says\_Nonce\_not\_used\_guard*: " $\llbracket \text{Says } A' \ B \ \{A'', r, I, L\} \in \text{set evs};$   
 $\text{Nonce } n \notin \text{used evs} \rrbracket \implies L \in \text{guard } n \text{ Ks}$ "  
 $\langle \text{proof} \rangle$

### 38.2.9 guardedness of messages

**lemma** *chain\_guard* [iff]: " $\text{chain } B \ \text{ofr } A \ L \ C \in \text{guard } n \ \{\text{priK } A\}$ "  
 $\langle \text{proof} \rangle$

**lemma** *chain\_guard\_Nonce\_neq* [intro]: " $n \neq \text{ofr}$   
 $\implies \text{chain } B \ \text{ofr } A' \ L \ C \in \text{guard } n \ \{\text{priK } A\}$ "  
 $\langle \text{proof} \rangle$

**lemma** *anchor\_guard* [iff]: " $\text{anchor } A \ n' \ B \in \text{guard } n \ \{\text{priK } A\}$ "  
 $\langle \text{proof} \rangle$

**lemma** *anchor\_guard\_Nonce\_neq* [intro]: " $n \neq n'$   
 $\implies \text{anchor } A' \ n' \ B \in \text{guard } n \ \{\text{priK } A\}$ "  
 $\langle \text{proof} \rangle$

**lemma** *reqm\_guard* [intro]: " $I \in \text{agl} \implies \text{reqm } A \ r \ n' \ I \ B \in \text{guard } n \ \{\text{priK } A\}$ "  
 $\langle \text{proof} \rangle$

**lemma** *reqm\_guard\_Nonce\_neq* [intro]: " $\llbracket n \neq n'; I \in \text{agl} \rrbracket$   
 $\implies \text{reqm } A' \ r \ n' \ I \ B \in \text{guard } n \ \{\text{priK } A\}$ "  
 $\langle \text{proof} \rangle$

**lemma** *prom\_guard* [intro]: " $\llbracket I \in \text{agl}; J \in \text{agl}; L \in \text{guard } n \ \{\text{priK } A\} \rrbracket$   
 $\implies \text{prom } B \ \text{ofr } A \ r \ I \ L \ J \ C \in \text{guard } n \ \{\text{priK } A\}$ "

*<proof>*

**lemma** *prom\_guard\_Nonce\_neq* [intro]: " $\llbracket n \neq \text{ofr}; I \in \text{agl}; J \in \text{agl};$   
 $L \in \text{guard } n \{ \text{priK } A \} \rrbracket \implies \text{prom } B \text{ ofr } A' \text{ r } I \text{ L } J \text{ C} \in \text{guard } n \{ \text{priK } A \}$ "  
*<proof>*

### 38.2.10 Nonce uniqueness

**lemma** *uniq\_Nonce\_in\_chain* [dest]: " $\text{Nonce } k \in \text{parts } \{ \text{chain } B \text{ ofr } A \text{ L } C \} \implies$   
 $k = \text{ofr}$ "  
*<proof>*

**lemma** *uniq\_Nonce\_in\_anchor* [dest]: " $\text{Nonce } k \in \text{parts } \{ \text{anchor } A \text{ n } B \} \implies k = n$ "  
*<proof>*

**lemma** *uniq\_Nonce\_in\_reqm* [dest]: " $\llbracket \text{Nonce } k \in \text{parts } \{ \text{reqm } A \text{ r } n \text{ I } B \};$   
 $I \in \text{agl} \rrbracket \implies k = n$ "  
*<proof>*

**lemma** *uniq\_Nonce\_in\_prom* [dest]: " $\llbracket \text{Nonce } k \in \text{parts } \{ \text{prom } B \text{ ofr } A \text{ r } I \text{ L } J$   
 $C \};$   
 $I \in \text{agl}; J \in \text{agl}; \text{Nonce } k \notin \text{parts } \{ L \} \rrbracket \implies k = \text{ofr}$ "  
*<proof>*

### 38.2.11 requests are guarded

**lemma** *req\_imp\_Guard* [rule\_format]: " $\llbracket \text{evs} \in p1; A \notin \text{bad} \rrbracket \implies$   
 $\text{req } A \text{ r } n \text{ I } B \in \text{set evs} \longrightarrow \text{Guard } n \{ \text{priK } A \} (\text{spies evs})$ "  
*<proof>*

**lemma** *req\_imp\_Guard\_Friend*: " $\llbracket \text{evs} \in p1; A \notin \text{bad}; \text{req } A \text{ r } n \text{ I } B \in \text{set evs} \rrbracket$   
 $\implies \text{Guard } n \{ \text{priK } A \} (\text{knows\_max } (\text{Friend } C) \text{ evs})$ "  
*<proof>*

### 38.2.12 propositions are guarded

**lemma** *pro\_imp\_Guard* [rule\_format]: " $\llbracket \text{evs} \in p1; B \notin \text{bad}; A \notin \text{bad} \rrbracket \implies$   
 $\text{pro } B \text{ ofr } A \text{ r } I (\text{cons } M \text{ L}) J \text{ C} \in \text{set evs} \longrightarrow \text{Guard ofr } \{ \text{priK } A \} (\text{spies evs})$ "  
*<proof>*

**lemma** *pro\_imp\_Guard\_Friend*: " $\llbracket \text{evs} \in p1; B \notin \text{bad}; A \notin \text{bad};$   
 $\text{pro } B \text{ ofr } A \text{ r } I (\text{cons } M \text{ L}) J \text{ C} \in \text{set evs} \rrbracket$   
 $\implies \text{Guard ofr } \{ \text{priK } A \} (\text{knows\_max } (\text{Friend } D) \text{ evs})$ "  
*<proof>*

### 38.2.13 data confidentiality: no one other than the originator can decrypt the offers

**lemma** *Nonce\_req\_notin\_spies*: " $\llbracket \text{evs} \in p1; \text{req } A \text{ r } n \text{ I } B \in \text{set evs}; A \notin \text{bad} \rrbracket$   
 $\implies \text{Nonce } n \notin \text{analz } (\text{spies evs})$ "  
*<proof>*

**lemma** *Nonce\_req\_notin\_knows\_max\_Friend*: " $\llbracket \text{evs} \in p1; \text{req } A \text{ r } n \text{ I } B \in \text{set}$   
 $\text{evs};$   
 $A \notin \text{bad}; A \neq \text{Friend } C \rrbracket \implies \text{Nonce } n \notin \text{analz } (\text{knows\_max } (\text{Friend } C) \text{ evs})$ "

*<proof>*

**lemma** *Nonce\_pro\_notin\_spies*: "[*evs* ∈ *p1*; *B* ∉ *bad*; *A* ∉ *bad*;  
 pro *B* ofr *A* r *I* (cons *M L*) *J C* ∈ set *evs*] ⇒ Nonce ofr ∉ analz (*spies evs*)"  
*<proof>*

**lemma** *Nonce\_pro\_notin\_knows\_max\_Friend*: "[*evs* ∈ *p1*; *B* ∉ *bad*; *A* ∉ *bad*;  
*A* ≠ *Friend D*; pro *B* ofr *A* r *I* (cons *M L*) *J C* ∈ set *evs*]  
 ⇒ Nonce ofr ∉ analz (*knows\_max (Friend D) evs*)"  
*<proof>*

### 38.2.14 non repudiability: an offer signed by B has been sent by B

**lemma** *Crypt\_reqm*: "[Crypt (*priK A*) *X* ∈ parts {*reqm A' r n I B*}; *I* ∈ *agl*]  
 ⇒ *A=A'*"  
*<proof>*

**lemma** *Crypt\_prom*: "[Crypt (*priK A*) *X* ∈ parts {*prom B ofr A' r I L J C*};  
*I* ∈ *agl*; *J* ∈ *agl*] ⇒ *A=B* ∨ Crypt (*priK A*) *X* ∈ parts {*L*}"  
*<proof>*

**lemma** *Crypt\_safeness*: "[*evs* ∈ *p1*; *A* ∉ *bad*] ⇒ Crypt (*priK A*) *X* ∈ parts  
 (*spies evs*)  
 → (∃ *B Y*. Says *A B Y* ∈ set *evs* ∧ Crypt (*priK A*) *X* ∈ parts {*Y*})"  
*<proof>*

**lemma** *Crypt\_Hash\_imp\_sign*: "[*evs* ∈ *p1*; *A* ∉ *bad*] ⇒  
 Crypt (*priK A*) (Hash *X*) ∈ parts (*spies evs*)  
 → (∃ *B Y*. Says *A B Y* ∈ set *evs* ∧ sign *A X* ∈ parts {*Y*})"  
*<proof>*

**lemma** *sign\_safeness*: "[*evs* ∈ *p1*; *A* ∉ *bad*] ⇒ sign *A X* ∈ parts (*spies evs*)  
 → (∃ *B Y*. Says *A B Y* ∈ set *evs* ∧ sign *A X* ∈ parts {*Y*})"  
*<proof>*

end

## 39 Protocol P2

theory *P2* imports *Guard\_Public List\_Msg* begin

### 39.1 Protocol Definition

Like *P1* except the definitions of *chain*, *shop*, *next\_shop* and *nonce*

#### 39.1.1 offer chaining: B chains his offer for A with the head offer of L for sending it to C

**definition** *chain* :: "agent ⇒ nat ⇒ agent ⇒ msg ⇒ agent ⇒ msg" where  
 "chain *B ofr A L C* ==  
 let *m1* = sign *B (Nonce ofr)* in  
 let *m2* = Hash {head *L*, Agent *C*} in  
 {Crypt (*pubK A*) *m1*, *m2*}"

**declare** *Let\_def* [*simp*]

**lemma** *chain\_inj* [*iff*]: " $(\text{chain } B \text{ ofr } A \text{ L } C = \text{chain } B' \text{ ofr}' A' \text{ L}' C') = (B=B' \ \& \ \text{ofr}=\text{ofr}' \ \& \ A=A' \ \& \ \text{head } L = \text{head } L' \ \& \ C=C')$ "  
 <proof>

**lemma** *Nonce\_in\_chain* [*iff*]: " $\text{Nonce ofr} \in \text{parts } \{\text{chain } B \text{ ofr } A \text{ L } C\}$ "  
 <proof>

### 39.1.2 agent whose key is used to sign an offer

**fun** *shop* :: "*msg* => *msg*" **where**  
 "*shop*  $\{\{\text{Crypt } K \ \{\{B, \text{ofr}, \text{Crypt } K' \ H\}, m2\}\} = \text{Agent } (\text{agt } K')\}$ "

**lemma** *shop\_chain* [*simp*]: "*shop* (*chain* *B ofr A L C*) = *Agent B*"  
 <proof>

### 39.1.3 nonce used in an offer

**fun** *nonce* :: "*msg* => *msg*" **where**  
 "*nonce*  $\{\{\text{Crypt } K \ \{\{B, \text{ofr}, \text{CryptH}\}, m2\}\} = \text{ofr}\}$ "

**lemma** *nonce\_chain* [*simp*]: "*nonce* (*chain* *B ofr A L C*) = *Nonce ofr*"  
 <proof>

### 39.1.4 next shop

**fun** *next\_shop* :: "*msg* => *agent*" **where**  
 "*next\_shop*  $\{\{m1, \text{Hash } \{\{\text{headL}, \text{Agent } C\}\}\} = C\}$ "

**lemma** "*next\_shop* (*chain* *B ofr A L C*) = *C*"  
 <proof>

### 39.1.5 anchor of the offer list

**definition** *anchor* :: "*agent* => *nat* => *agent* => *msg*" **where**  
 "*anchor* *A n B* == *chain* *A n A* (*cons* *nil nil*) *B*"

**lemma** *anchor\_inj* [*iff*]:  
 " $(\text{anchor } A \text{ n } B = \text{anchor } A' \text{ n}' B') = (A=A' \ \wedge \ n=n' \ \wedge \ B=B')$ "  
 <proof>

**lemma** *Nonce\_in\_anchor* [*iff*]: " $\text{Nonce } n \in \text{parts } \{\text{anchor } A \text{ n } B\}$ "  
 <proof>

**lemma** *shop\_anchor* [*simp*]: "*shop* (*anchor* *A n B*) = *Agent A*"  
 <proof>

### 39.1.6 request event

**definition** *reqm* :: "*agent* => *nat* => *nat* => *msg* => *agent* => *msg*" **where**  
 "*reqm* *A r n I B* ==  $\{\{\text{Agent } A, \text{Number } r, \text{cons } (\text{Agent } A) (\text{cons } (\text{Agent } B) I), \text{cons } (\text{anchor } A \text{ n } B) \text{ nil}\}\}$ "

**lemma** reqm\_inj [iff]: "(reqm A r n I B = reqm A' r' n' I' B')  
 = (A=A' & r=r' & n=n' & I=I' & B=B')"  
 <proof>

**lemma** Nonce\_in\_reqm [iff]: "Nonce n ∈ parts {reqm A r n I B}"  
 <proof>

**definition** req :: "agent => nat => nat => msg => agent => event" where  
 "req A r n I B == Says A B (reqm A r n I B)"

**lemma** req\_inj [iff]: "(req A r n I B = req A' r' n' I' B')  
 = (A=A' & r=r' & n=n' & I=I' & B=B')"  
 <proof>

### 39.1.7 propose event

**definition** prom :: "agent => nat => agent => nat => msg => msg =>  
 msg => agent => msg" where  
 "prom B ofr A r I L J C == {Agent A, Number r,  
 app (J, del (Agent B, I)), cons (chain B ofr A L C) L}"

**lemma** prom\_inj [dest]: "prom B ofr A r I L J C = prom B' ofr' A' r' I' L'  
 J' C'  
 ==> B=B' & ofr=ofr' & A=A' & r=r' & L=L' & C=C'"  
 <proof>

**lemma** Nonce\_in\_prom [iff]: "Nonce ofr ∈ parts {prom B ofr A r I L J C}"  
 <proof>

**definition** pro :: "agent => nat => agent => nat => msg => msg =>  
 msg => agent => event" where  
 "pro B ofr A r I L J C == Says B C (prom B ofr A r I L J C)"

**lemma** pro\_inj [dest]: "pro B ofr A r I L J C = pro B' ofr' A' r' I' L' J'  
 C'  
 ==> B=B' & ofr=ofr' & A=A' & r=r' & L=L' & C=C'"  
 <proof>

### 39.1.8 protocol

**inductive\_set** p2 :: "event list set"  
**where**

Nil: "[ ] ∈ p2"

/ Fake: "[evsf ∈ p2; X ∈ synth (analz (spies evsf))] ==> Says Spy B X # evsf  
 ∈ p2"

/ Request: "[evsr ∈ p2; Nonce n ∉ used evsr; I ∈ agl] ==> req A r n I B #  
 evsr ∈ p2"

/ Propose: "[evsp ∈ p2; Says A' B {Agent A, Number r, I, cons M L} ∈ set evsp;  
 I ∈ agl; J ∈ agl; isin (Agent C, app (J, del (Agent B, I))) ;  
 Nonce ofr ∉ used evsp] ==> pro B ofr A r I (cons M L) J C # evsp ∈ p2"



**39.1.9 valid offer lists**

```

inductive_set
  valid :: "agent  $\Rightarrow$  nat  $\Rightarrow$  agent  $\Rightarrow$  msg set"
  for A :: agent and n :: nat and B :: agent
where
  Request [intro]: "cons (anchor A n B) nil  $\in$  valid A n B"

  | Propose [intro]: "L  $\in$  valid A n B
 $\implies$  cons (chain (next_shop (head L)) ofr A L C) L  $\in$  valid A n B"

```

**39.1.10 basic properties of valid**

```

lemma valid_not_empty: "L  $\in$  valid A n B  $\implies \exists M L'. L = \text{cons } M L'"
<proof>$ 
```

```

lemma valid_pos_len: "L  $\in$  valid A n B  $\implies 0 < \text{len } L"
<proof>$ 
```

**39.1.11 list of offers**

```

fun offers :: "msg  $\Rightarrow$  msg"
where
  "offers (cons M L) = cons {shop M, nonce M} (offers L)"
  | "offers other = nil"

```

**39.2 Properties of Protocol P2**

same as *P1\_Prop* except that publicly verifiable forward integrity is replaced by forward privacy

**39.3 strong forward integrity: except the last one, no offer can be modified**

```

lemma strong_forward_integrity: " $\forall L. \text{Suc } i < \text{len } L$ 
 $\longrightarrow L \in \text{valid } A \text{ n } B \longrightarrow \text{repl } (L, \text{Suc } i, M) \in \text{valid } A \text{ n } B \longrightarrow M = \text{ith } (L, \text{Suc } i)"
<proof>$ 
```

**39.4 insertion resilience: except at the beginning, no offer can be inserted**

```

lemma chain_isnt_head [simp]: "L  $\in$  valid A n B  $\implies$ 
head L  $\neq$  chain (next_shop (head L)) ofr A L C"
<proof>

```

```

lemma insertion_resilience: " $\forall L. L \in \text{valid } A \text{ n } B \longrightarrow \text{Suc } i < \text{len } L$ 
 $\longrightarrow \text{ins } (L, \text{Suc } i, M) \notin \text{valid } A \text{ n } B"
<proof>$ 
```

**39.5 truncation resilience: only shop i can truncate at offer i**

```

lemma truncation_resilience: " $\forall L. L \in \text{valid } A \text{ n } B \longrightarrow \text{Suc } i < \text{len } L$ 

```

$\longrightarrow \text{cons } M (\text{trunc } (L, \text{Suc } i)) \in \text{valid } A \text{ n } B \longrightarrow \text{shop } M = \text{shop } (\text{ith } (L, i))"$   
 $\langle \text{proof} \rangle$

### 39.6 declarations for tactics

**declare** *knows\_Spy\_partsEs* [elim]  
**declare** *Fake\_parts\_insert* [THEN subsetD, dest]  
**declare** *initState.simps* [simp del]

### 39.7 get components of a message

**lemma** *get\_ML* [dest]: "Says A' B  $\{A, R, I, M, L\} \in \text{set evs} \implies$   
 $M \in \text{parts } (\text{spies evs}) \wedge L \in \text{parts } (\text{spies evs})"$   
 $\langle \text{proof} \rangle$

### 39.8 general properties of p2

**lemma** *reqm\_neq\_prom* [iff]:  
 $"\text{reqm } A \text{ r n } I \text{ B} \neq \text{prom } B' \text{ ofr } A' \text{ r}' \text{ I}' (\text{cons } M \text{ L}) \text{ J } C"$   
 $\langle \text{proof} \rangle$

**lemma** *prom\_neq\_reqm* [iff]:  
 $"\text{prom } B' \text{ ofr } A' \text{ r}' \text{ I}' (\text{cons } M \text{ L}) \text{ J } C \neq \text{reqm } A \text{ r n } I \text{ B}"$   
 $\langle \text{proof} \rangle$

**lemma** *req\_neq\_pro* [iff]:  $"\text{req } A \text{ r n } I \text{ B} \neq \text{pro } B' \text{ ofr } A' \text{ r}' \text{ I}' (\text{cons } M \text{ L}) \text{ J } C"$   
 $\langle \text{proof} \rangle$

**lemma** *pro\_neq\_req* [iff]:  $"\text{pro } B' \text{ ofr } A' \text{ r}' \text{ I}' (\text{cons } M \text{ L}) \text{ J } C \neq \text{req } A \text{ r n } I \text{ B}"$   
 $\langle \text{proof} \rangle$

**lemma** *p2\_has\_no\_Gets*:  $"\text{evs} \in \text{p2} \implies \forall A \text{ X. Gets } A \text{ X} \notin \text{set evs}"$   
 $\langle \text{proof} \rangle$

**lemma** *p2\_is\_Gets\_correct* [iff]:  $"\text{Gets\_correct p2}"$   
 $\langle \text{proof} \rangle$

**lemma** *p2\_is\_one\_step* [iff]:  $"\text{one\_step p2}"$   
 $\langle \text{proof} \rangle$

**lemma** *p2\_has\_only\_Says'* [rule\_format]:  $"\text{evs} \in \text{p2} \implies$   
 $\text{ev} \in \text{set evs} \longrightarrow (\exists A \text{ B } X. \text{ev} = \text{Says } A \text{ B } X)"$   
 $\langle \text{proof} \rangle$

**lemma** *p2\_has\_only\_Says* [iff]:  $"\text{has\_only\_Says p2}"$   
 $\langle \text{proof} \rangle$

**lemma** *p2\_is\_regular* [iff]:  $"\text{regular p2}"$   
 $\langle \text{proof} \rangle$

### 39.9 private keys are safe

**lemma** *priK\_parts\_Friend\_imp\_bad* [rule\_format,dest]:  
 "[[evs ∈ p2; Friend B ≠ A]]  
 ⇒ (Key (priK A) ∈ parts (knows (Friend B) evs)) → (A ∈ bad)"  
 <proof>

**lemma** *priK\_analz\_Friend\_imp\_bad* [rule\_format,dest]:  
 "[[evs ∈ p2; Friend B ≠ A]]  
 ⇒ (Key (priK A) ∈ analz (knows (Friend B) evs)) → (A ∈ bad)"  
 <proof>

**lemma** *priK\_notin\_knows\_max\_Friend*:  
 "[[evs ∈ p2; A ∉ bad; A ≠ Friend C]]  
 ⇒ Key (priK A) ∉ analz (knows\_max (Friend C) evs)"  
 <proof>

### 39.10 general guardedness properties

**lemma** *agl\_guard* [intro]: "I ∈ agl ⇒ I ∈ guard n Ks"  
 <proof>

**lemma** *Says\_to\_knows\_max'\_guard*: "[[Says A' C {A'',r,I,L} ∈ set evs;  
 Guard n Ks (knows\_max' C evs)]] ⇒ L ∈ guard n Ks"  
 <proof>

**lemma** *Says\_from\_knows\_max'\_guard*: "[[Says C A' {A'',r,I,L} ∈ set evs;  
 Guard n Ks (knows\_max' C evs)]] ⇒ L ∈ guard n Ks"  
 <proof>

**lemma** *Says\_Nonce\_not\_used\_guard*: "[[Says A' B {A'',r,I,L} ∈ set evs;  
 Nonce n ∉ used evs]] ⇒ L ∈ guard n Ks"  
 <proof>

### 39.11 guardedness of messages

**lemma** *chain\_guard* [iff]: "chain B ofr A L C ∈ guard n {priK A}"  
 <proof>

**lemma** *chain\_guard\_Nonce\_neq* [intro]: "n ≠ n'  
 ⇒ chain B ofr A' L C ∈ guard n {priK A}"  
 <proof>

**lemma** *anchor\_guard* [iff]: "anchor A n' B ∈ guard n {priK A}"  
 <proof>

**lemma** *anchor\_guard\_Nonce\_neq* [intro]: "n ≠ n'  
 ⇒ anchor A' n' B ∈ guard n {priK A}"  
 <proof>

**lemma** *reqm\_guard* [intro]: "I ∈ agl ⇒ reqm A r n' I B ∈ guard n {priK A}"  
 <proof>

**lemma** *reqm\_guard\_Nonce\_neq* [intro]: "[[n ≠ n'; I ∈ agl]]  
 ⇒ reqm A' r n' I B ∈ guard n {priK A}"

*<proof>*

**lemma** *prom\_guard [intro]:* " $\llbracket I \in \text{agl}; J \in \text{agl}; L \in \text{guard } n \{ \text{priK } A \} \rrbracket$   
 $\implies \text{prom } B \text{ ofr } A \text{ r } I \text{ L } J \text{ C} \in \text{guard } n \{ \text{priK } A \}$ "  
*<proof>*

**lemma** *prom\_guard\_Nonce\_neq [intro]:* " $\llbracket n \neq \text{ofr}; I \in \text{agl}; J \in \text{agl};$   
 $L \in \text{guard } n \{ \text{priK } A \} \rrbracket \implies \text{prom } B \text{ ofr } A' \text{ r } I \text{ L } J \text{ C} \in \text{guard } n \{ \text{priK } A \}$ "  
*<proof>*

### 39.12 Nonce uniqueness

**lemma** *uniq\_Nonce\_in\_chain [dest]:* " $\text{Nonce } k \in \text{parts } \{ \text{chain } B \text{ ofr } A \text{ L } C \} \implies$   
 $k = \text{ofr}$ "  
*<proof>*

**lemma** *uniq\_Nonce\_in\_anchor [dest]:* " $\text{Nonce } k \in \text{parts } \{ \text{anchor } A \text{ n } B \} \implies k = n$ "  
*<proof>*

**lemma** *uniq\_Nonce\_in\_reqm [dest]:* " $\llbracket \text{Nonce } k \in \text{parts } \{ \text{reqm } A \text{ r } n \text{ I } B \};$   
 $I \in \text{agl} \rrbracket \implies k = n$ "  
*<proof>*

**lemma** *uniq\_Nonce\_in\_prom [dest]:* " $\llbracket \text{Nonce } k \in \text{parts } \{ \text{prom } B \text{ ofr } A \text{ r } I \text{ L } J$   
 $C \};$   
 $I \in \text{agl}; J \in \text{agl}; \text{Nonce } k \notin \text{parts } \{ L \} \rrbracket \implies k = \text{ofr}$ "  
*<proof>*

### 39.13 requests are guarded

**lemma** *req\_imp\_Guard [rule\_format]:* " $\llbracket \text{evs} \in p2; A \notin \text{bad} \rrbracket \implies$   
 $\text{req } A \text{ r } n \text{ I } B \in \text{set evs} \longrightarrow \text{Guard } n \{ \text{priK } A \} (\text{spies evs})$ "  
*<proof>*

**lemma** *req\_imp\_Guard\_Friend:* " $\llbracket \text{evs} \in p2; A \notin \text{bad}; \text{req } A \text{ r } n \text{ I } B \in \text{set evs} \rrbracket$   
 $\implies \text{Guard } n \{ \text{priK } A \} (\text{knows\_max } (\text{Friend } C) \text{ evs})$ "  
*<proof>*

### 39.14 propositions are guarded

**lemma** *pro\_imp\_Guard [rule\_format]:* " $\llbracket \text{evs} \in p2; B \notin \text{bad}; A \notin \text{bad} \rrbracket \implies$   
 $\text{pro } B \text{ ofr } A \text{ r } I (\text{cons } M \text{ L}) J \text{ C} \in \text{set evs} \longrightarrow \text{Guard ofr } \{ \text{priK } A \} (\text{spies evs})$ "  
*<proof>*

**lemma** *pro\_imp\_Guard\_Friend:* " $\llbracket \text{evs} \in p2; B \notin \text{bad}; A \notin \text{bad};$   
 $\text{pro } B \text{ ofr } A \text{ r } I (\text{cons } M \text{ L}) J \text{ C} \in \text{set evs} \rrbracket$   
 $\implies \text{Guard ofr } \{ \text{priK } A \} (\text{knows\_max } (\text{Friend } D) \text{ evs})$ "  
*<proof>*

### 39.15 data confidentiality: no one other than the originator can decrypt the offers

**lemma** *Nonce\_req\_notin\_spies:* " $\llbracket \text{evs} \in p2; \text{req } A \text{ r } n \text{ I } B \in \text{set evs}; A \notin \text{bad} \rrbracket$   
 $\implies \text{Nonce } n \notin \text{analz } (\text{spies evs})$ "

### 39.16 forward privacy: only the originator can know the identity of the shops 309

*<proof>*

**lemma** *Nonce\_req\_notin\_knows\_max\_Friend*: "[ $\text{evs} \in p2$ ;  $\text{req } A \text{ r n } I \ B \in \text{set evs}$ ;  
 $A \notin \text{bad}$ ;  $A \neq \text{Friend } C$ ]  $\implies \text{Nonce } n \notin \text{analz } (\text{knows\_max } (\text{Friend } C) \text{ evs})$ "  
*<proof>*

**lemma** *Nonce\_pro\_notin\_spies*: "[ $\text{evs} \in p2$ ;  $B \notin \text{bad}$ ;  $A \notin \text{bad}$ ;  
 $\text{pro } B \text{ ofr } A \text{ r } I \ (\text{cons } M \ L) \ J \ C \in \text{set evs}$ ]  $\implies \text{Nonce ofr} \notin \text{analz } (\text{spies evs})$ "  
*<proof>*

**lemma** *Nonce\_pro\_notin\_knows\_max\_Friend*: "[ $\text{evs} \in p2$ ;  $B \notin \text{bad}$ ;  $A \notin \text{bad}$ ;  
 $A \neq \text{Friend } D$ ;  $\text{pro } B \text{ ofr } A \text{ r } I \ (\text{cons } M \ L) \ J \ C \in \text{set evs}$ ]  $\implies \text{Nonce ofr} \notin \text{analz } (\text{knows\_max } (\text{Friend } D) \text{ evs})$ "  
*<proof>*

### 39.16 forward privacy: only the originator can know the identity of the shops

**lemma** *forward\_privacy\_Spy*: "[ $\text{evs} \in p2$ ;  $B \notin \text{bad}$ ;  $A \notin \text{bad}$ ;  
 $\text{pro } B \text{ ofr } A \text{ r } I \ (\text{cons } M \ L) \ J \ C \in \text{set evs}$ ]  $\implies \text{sign } B \ (\text{Nonce ofr}) \notin \text{analz } (\text{spies evs})$ "  
*<proof>*

**lemma** *forward\_privacy\_Friend*: "[ $\text{evs} \in p2$ ;  $B \notin \text{bad}$ ;  $A \notin \text{bad}$ ;  $A \neq \text{Friend } D$ ;  
 $\text{pro } B \text{ ofr } A \text{ r } I \ (\text{cons } M \ L) \ J \ C \in \text{set evs}$ ]  $\implies \text{sign } B \ (\text{Nonce ofr}) \notin \text{analz } (\text{knows\_max } (\text{Friend } D) \text{ evs})$ "  
*<proof>*

### 39.17 non repudiability: an offer signed by B has been sent by B

**lemma** *Crypt\_reqm*: "[ $\text{Crypt } (\text{priK } A) \ X \in \text{parts } \{\text{reqm } A' \text{ r n } I \ B\}$ ;  $I \in \text{agl}$ ]  $\implies A=A'$ "  
*<proof>*

**lemma** *Crypt\_prom*: "[ $\text{Crypt } (\text{priK } A) \ X \in \text{parts } \{\text{prom } B \text{ ofr } A' \text{ r } I \ L \ J \ C\}$ ;  
 $I \in \text{agl}$ ;  $J \in \text{agl}$ ]  $\implies A=B \mid \text{Crypt } (\text{priK } A) \ X \in \text{parts } \{L\}$ "  
*<proof>*

**lemma** *Crypt\_safeness*: "[ $\text{evs} \in p2$ ;  $A \notin \text{bad}$ ]  $\implies \text{Crypt } (\text{priK } A) \ X \in \text{parts } (\text{spies evs})$   
 $\longrightarrow (\exists B \ Y. \text{Says } A \ B \ Y \in \text{set evs} \ \& \ \text{Crypt } (\text{priK } A) \ X \in \text{parts } \{Y\})$ "  
*<proof>*

**lemma** *Crypt\_Hash\_imp\_sign*: "[ $\text{evs} \in p2$ ;  $A \notin \text{bad}$ ]  $\implies \text{Crypt } (\text{priK } A) \ (\text{Hash } X) \in \text{parts } (\text{spies evs})$   
 $\longrightarrow (\exists B \ Y. \text{Says } A \ B \ Y \in \text{set evs} \ \wedge \ \text{sign } A \ X \in \text{parts } \{Y\})$ "  
*<proof>*

**lemma** *sign\_safeness*: "[ $\text{evs} \in p2$ ;  $A \notin \text{bad}$ ]  $\implies \text{sign } A \ X \in \text{parts } (\text{spies evs})$   
 $\longrightarrow (\exists B \ Y. \text{Says } A \ B \ Y \in \text{set evs} \ \wedge \ \text{sign } A \ X \in \text{parts } \{Y\})$ "  
*<proof>*

end

## 40 Needham-Schroeder-Lowe Public-Key Protocol

theory Guard\_NS\_Public imports Guard\_Public begin

### 40.1 messages used in the protocol

abbreviation (input)

ns1 :: "agent => agent => nat => event" where  
 "ns1 A B NA == Says A B (Crypt (pubK B) {Nonce NA, Agent A})"

abbreviation (input)

ns1' :: "agent => agent => nat => event" where  
 "ns1' A' A B NA == Says A' B (Crypt (pubK B) {Nonce NA, Agent A})"

abbreviation (input)

ns2 :: "agent => agent => nat => nat => event" where  
 "ns2 B A NA NB == Says B A (Crypt (pubK A) {Nonce NA, Nonce NB, Agent B})"

abbreviation (input)

ns2' :: "agent => agent => agent => nat => nat => event" where  
 "ns2' B' B A NA NB == Says B' A (Crypt (pubK A) {Nonce NA, Nonce NB, Agent B})"

abbreviation (input)

ns3 :: "agent => agent => nat => event" where  
 "ns3 A B NB == Says A B (Crypt (pubK B) (Nonce NB))"

### 40.2 definition of the protocol

inductive\_set nsp :: "event list set"

where

Nil: "[ ] ∈ nsp"

/ Fake: "[ evs ∈ nsp; X ∈ synth (analz (spies evs)) ] ⇒ Says Spy B X # evs ∈ nsp"

/ NS1: "[ evs1 ∈ nsp; Nonce NA ∉ used evs1 ] ⇒ ns1 A B NA # evs1 ∈ nsp"

/ NS2: "[ evs2 ∈ nsp; Nonce NB ∉ used evs2; ns1' A' A B NA ∈ set evs2 ] ⇒ ns2 B A NA NB # evs2 ∈ nsp"

/ NS3: "[ A B B' NA NB evs3. [ evs3 ∈ nsp; ns1 A B NA ∈ set evs3; ns2' B' B A NA NB ∈ set evs3 ] ⇒ ns3 A B NB # evs3 ∈ nsp"

### 40.3 declarations for tactics

declare knows\_Spy\_partsEs [elim]

```
declare Fake_parts_insert [THEN subsetD, dest]
declare initState.simps [simp del]
```

#### 40.4 general properties of nsp

```
lemma nsp_has_no_Gets: "evs ∈ nsp ⇒ ∀ A X. Gets A X ∉ set evs"
⟨proof⟩
```

```
lemma nsp_is_Gets_correct [iff]: "Gets_correct nsp"
⟨proof⟩
```

```
lemma nsp_is_one_step [iff]: "one_step nsp"
⟨proof⟩
```

```
lemma nsp_has_only_Says' [rule_format]: "evs ∈ nsp ⇒
ev ∈ set evs → (∃ A B X. ev=Says A B X)"
⟨proof⟩
```

```
lemma nsp_has_only_Says [iff]: "has_only_Says nsp"
⟨proof⟩
```

```
lemma nsp_is_regular [iff]: "regular nsp"
⟨proof⟩
```

#### 40.5 nonce are used only once

```
lemma NA_is_uniq [rule_format]: "evs ∈ nsp ⇒
Crypt (pubK B) {Nonce NA, Agent A} ∈ parts (spies evs)
→ Crypt (pubK B') {Nonce NA, Agent A'} ∈ parts (spies evs)
→ Nonce NA ∉ analz (spies evs) → A=A' ∧ B=B'"
⟨proof⟩
```

```
lemma no_Nonce_NS1_NS2 [rule_format]: "evs ∈ nsp ⇒
Crypt (pubK B') {Nonce NA', Nonce NA, Agent A'} ∈ parts (spies evs)
→ Crypt (pubK B) {Nonce NA, Agent A} ∈ parts (spies evs)
→ Nonce NA ∈ analz (spies evs)"
⟨proof⟩
```

```
lemma no_Nonce_NS1_NS2' [rule_format]:
"[[Crypt (pubK B') {Nonce NA', Nonce NA, Agent A'} ∈ parts (spies evs);
Crypt (pubK B) {Nonce NA, Agent A} ∈ parts (spies evs); evs ∈ nsp]]
⇒ Nonce NA ∈ analz (spies evs)"
⟨proof⟩
```

```
lemma NB_is_uniq [rule_format]: "evs ∈ nsp ⇒
Crypt (pubK A) {Nonce NA, Nonce NB, Agent B} ∈ parts (spies evs)
→ Crypt (pubK A') {Nonce NA', Nonce NB, Agent B'} ∈ parts (spies evs)
→ Nonce NB ∉ analz (spies evs) → A=A' ∧ B=B' ∧ NA=NA'"
⟨proof⟩
```

#### 40.6 guardedness of NA

```
lemma ns1_imp_Guard [rule_format]: "[[evs ∈ nsp; A ∉ bad; B ∉ bad]] ⇒
ns1 A B NA ∈ set evs → Guard NA {priK A, priK B} (spies evs)"
```

*<proof>*

### 40.7 guardedness of NB

**lemma** *ns2\_imp\_Guard* [rule\_format]: " $\llbracket \text{evs} \in \text{nsp}; A \notin \text{bad}; B \notin \text{bad} \rrbracket \implies$   
 $\text{ns2 } B \ A \ NA \ NB \in \text{set evs} \longrightarrow \text{Guard } NB \ \{\text{priK } A, \text{priK } B\} \ (\text{spies evs})$ "  
*<proof>*

### 40.8 Agents' Authentication

**lemma** *B\_trusts\_NS1*: " $\llbracket \text{evs} \in \text{nsp}; A \notin \text{bad}; B \notin \text{bad} \rrbracket \implies$   
 $\text{Crypt } (\text{pubK } B) \ \{\text{Nonce } NA, \text{Agent } A\} \in \text{parts } (\text{spies evs})$   
 $\longrightarrow \text{Nonce } NA \notin \text{analz } (\text{spies evs}) \longrightarrow \text{ns1 } A \ B \ NA \in \text{set evs}$ "  
*<proof>*

**lemma** *A\_trusts\_NS2*: " $\llbracket \text{evs} \in \text{nsp}; A \notin \text{bad}; B \notin \text{bad} \rrbracket \implies \text{ns1 } A \ B \ NA \in \text{set evs}$   
 $\longrightarrow \text{Crypt } (\text{pubK } A) \ \{\text{Nonce } NA, \text{Nonce } NB, \text{Agent } B\} \in \text{parts } (\text{spies evs})$   
 $\longrightarrow \text{ns2 } B \ A \ NA \ NB \in \text{set evs}$ "  
*<proof>*

**lemma** *B\_trusts\_NS3*: " $\llbracket \text{evs} \in \text{nsp}; A \notin \text{bad}; B \notin \text{bad} \rrbracket \implies \text{ns2 } B \ A \ NA \ NB \in \text{set evs}$   
 $\longrightarrow \text{Crypt } (\text{pubK } B) \ (\text{Nonce } NB) \in \text{parts } (\text{spies evs}) \longrightarrow \text{ns3 } A \ B \ NB \in \text{set evs}$ "  
*<proof>*

**end**

## 41 Other Protocol-Independent Results

**theory** *Proto* imports *Guard\_Public* **begin**

### 41.1 protocols

**type\_synonym** *rule* = "event set \* event"

**abbreviation**

*msg'* :: "rule => msg" **where**  
*"msg' R == msg (snd R)"*

**type\_synonym** *proto* = "rule set"

**definition** *wdef* :: "proto => bool" **where**  
 $\text{"wdef } p \equiv \forall R \ k. R \in p \longrightarrow \text{Number } k \in \text{parts } \{\text{msg'} R\}$   
 $\longrightarrow \text{Number } k \in \text{parts } (\text{msg'} (\text{fst } R))$

### 41.2 substitutions

**record** *subs* =  
*agent* :: "agent => agent"  
*nonce* :: "nat => nat"  
*nb* :: "nat => msg"  
*key* :: "key => key"



```

primrec apm :: "subs => msg => msg" where
  "apm s (Agent A) = Agent (agent s A)"
| "apm s (Nonce n) = Nonce (nonce s n)"
| "apm s (Number n) = nb s n"
| "apm s (Key K) = Key (key s K)"
| "apm s (Hash X) = Hash (apm s X)"
| "apm s (Crypt K X) = (
  if (∃ A. K = pubK A) then Crypt (pubK (agent s (agt K))) (apm s X)
  else if (∃ A. K = priK A) then Crypt (priK (agent s (agt K))) (apm s X)
  else Crypt (key s K) (apm s X))"
| "apm s {X,Y} = {apm s X, apm s Y}"

```

**lemma** apm\_parts: " $X \in \text{parts } \{Y\} \implies \text{apm } s \ X \in \text{parts } \{\text{apm } s \ Y\}$ "  
 <proof>

**lemma** Nonce\_apm [rule\_format]: " $\text{Nonce } n \in \text{parts } \{\text{apm } s \ X\} \implies$   
 $(\forall k. \text{Number } k \in \text{parts } \{X\} \longrightarrow \text{Nonce } n \notin \text{parts } \{\text{nb } s \ k\}) \longrightarrow$   
 $(\exists k. \text{Nonce } k \in \text{parts } \{X\} \wedge \text{nonce } s \ k = n)$ "  
 <proof>

**lemma** wdef\_Nonce: " $\llbracket \text{Nonce } n \in \text{parts } \{\text{apm } s \ X\}; R \in p; \text{msg}' R = X; \text{wdef } p; \text{Nonce } n \notin \text{parts } (\text{apm } s \ '(\text{msg}' (fst R))) \rrbracket \implies$   
 $(\exists k. \text{Nonce } k \in \text{parts } \{X\} \wedge \text{nonce } s \ k = n)$ "  
 <proof>

```

primrec ap :: "subs => event => event" where
  "ap s (Says A B X) = Says (agent s A) (agent s B) (apm s X)"
| "ap s (Gets A X) = Gets (agent s A) (apm s X)"
| "ap s (Notes A X) = Notes (agent s A) (apm s X)"

```

#### abbreviation

```

ap' :: "subs => rule => event" where
  "ap' s R ≡ ap s (snd R)"

```

#### abbreviation

```

apm' :: "subs => rule => msg" where
  "apm' s R ≡ apm s (msg' R)"

```

#### abbreviation

```

priK' :: "subs => agent => key" where
  "priK' s A ≡ priK (agent s A)"

```

#### abbreviation

```

pubK' :: "subs => agent => key" where
  "pubK' s A ≡ pubK (agent s A)"

```

### 41.3 nonces generated by a rule

**definition** newn :: "rule => nat set" **where**  
 "newn R ≡ {n. Nonce n ∈ parts {msg (snd R)} ∧ Nonce n ∉ parts (msg' (fst R))}"

**lemma** newn\_parts: " $n \in \text{newn } R \implies \text{Nonce } (\text{nonce } s \ n) \in \text{parts } \{\text{apm}' s \ R\}$ "  
 <proof>

#### 41.4 traces generated by a protocol

**definition** `ok` :: "event list  $\Rightarrow$  rule  $\Rightarrow$  subs  $\Rightarrow$  bool" where  
`"ok evs R s  $\equiv$  ( $\forall x. x \in \text{fst } R \longrightarrow \text{ap } s \ x \in \text{set evs}$ )`  
 `$\wedge (\forall n. n \in \text{newn } R \longrightarrow \text{Nonce } (\text{nonce } s \ n) \notin \text{used evs})"$`

**inductive\_set**

`tr` :: "proto  $\Rightarrow$  event list set"  
 for `p` :: proto

where

`Nil [intro]: "[ ]  $\in$  tr p"`

`/ Fake [intro]: "[[evsf  $\in$  tr p; X  $\in$  synth (analz (spies evsf))]]`  
 `$\implies$  Says Spy B X # evsf  $\in$  tr p"`

`/ Proto [intro]: "[[evs  $\in$  tr p; R  $\in$  p; ok evs R s]]  $\implies$  ap' s R # evs  $\in$  tr`  
`p"`

#### 41.5 general properties

**lemma** `one_step_tr [iff]: "one_step (tr p)"`  
 `$\langle$ proof $\rangle$`

**definition** `has_only_Says'` :: "proto  $\Rightarrow$  bool" where  
`"has_only_Says' p  $\equiv$   $\forall R. R \in p \longrightarrow \text{is\_Says } (\text{snd } R)"$`

**lemma** `has_only_Says'D: "[R  $\in$  p; has_only_Says' p]`  
 `$\implies$  ( $\exists A \ B \ X. \text{snd } R = \text{Says } A \ B \ X)"$`   
 `$\langle$ proof $\rangle$`

**lemma** `has_only_Says_tr [simp]: "has_only_Says' p  $\implies$  has_only_Says (tr p)"`  
 `$\langle$ proof $\rangle$`

**lemma** `has_only_Says'_in_trD: "[[has_only_Says' p; list @ ev # evs1  $\in$  tr p]]`  
 `$\implies$  ( $\exists A \ B \ X. \text{ev} = \text{Says } A \ B \ X)"$`   
 `$\langle$ proof $\rangle$`

**lemma** `ok_not_used: "[[Nonce n  $\notin$  used evs; ok evs R s;`  
 `$\forall x. x \in \text{fst } R \longrightarrow \text{is\_Says } x]] \implies \text{Nonce } n \notin \text{parts } (\text{apm } s \ '(\text{msg } '(\text{fst } R)))"$`   
 `$\langle$ proof $\rangle$`

**lemma** `ok_is_Says: "[[evs' @ ev # evs  $\in$  tr p; ok evs R s; has_only_Says' p;`  
`R  $\in$  p; x  $\in$  fst R]]  $\implies$  is_Says x"`  
 `$\langle$ proof $\rangle$`

#### 41.6 types

`type_synonym keyfun = "rule  $\Rightarrow$  subs  $\Rightarrow$  nat  $\Rightarrow$  event list  $\Rightarrow$  key set"`

`type_synonym secfun = "rule  $\Rightarrow$  nat  $\Rightarrow$  subs  $\Rightarrow$  key set  $\Rightarrow$  msg"`

#### 41.7 introduction of a fresh guarded nonce

**definition** `fresh` :: "proto  $\Rightarrow$  rule  $\Rightarrow$  subs  $\Rightarrow$  nat  $\Rightarrow$  key set  $\Rightarrow$  event list

$\Rightarrow \text{bool}$ " where  
 "fresh p R s n Ks evs  $\equiv (\exists \text{ evs1 evs2. evs} = \text{evs2} @ \text{ap}' s R \# \text{evs1}$   
 $\wedge \text{Nonce } n \notin \text{used evs1} \wedge R \in p \wedge \text{ok evs1 } R s \wedge \text{Nonce } n \in \text{parts } \{\text{apm}' s R\}$   
 $\wedge \text{apm}' s R \in \text{guard } n \text{ Ks})$ "

**lemma freshD:** "fresh p R s n Ks evs  $\implies (\exists \text{ evs1 evs2.}$   
 $\text{evs} = \text{evs2} @ \text{ap}' s R \# \text{evs1} \wedge \text{Nonce } n \notin \text{used evs1} \wedge R \in p \wedge \text{ok evs1 } R s$   
 $\wedge \text{Nonce } n \in \text{parts } \{\text{apm}' s R\} \wedge \text{apm}' s R \in \text{guard } n \text{ Ks})$ "  
 <proof>

**lemma freshI [intro]:** "[Nonce n  $\notin$  used evs1; R  $\in$  p; Nonce n  $\in$  parts {apm' s R};  
 ok evs1 R s; apm' s R  $\in$  guard n Ks]  
 $\implies \text{fresh p R s n Ks (list @ ap}' s R \# \text{evs1})$ "  
 <proof>

**lemma freshI':** "[Nonce n  $\notin$  used evs1; (l,r)  $\in$  p;  
 Nonce n  $\in$  parts {apm s (msg r)}; ok evs1 (l,r) s; apm s (msg r)  $\in$  guard n  
 Ks]  
 $\implies \text{fresh p (l,r) s n Ks (evs2 @ ap s r \# evs1)}$ "  
 <proof>

**lemma fresh\_used:** "[fresh p R' s' n Ks evs; has\_only\_Says' p]  
 $\implies \text{Nonce } n \in \text{used evs}$ "  
 <proof>

**lemma fresh\_newn:** "[evs' @ ap' s R # evs  $\in$  tr p; wdef p; has\_only\_Says'  
 p;  
 Nonce n  $\notin$  used evs; R  $\in$  p; ok evs R s; Nonce n  $\in$  parts {apm' s R}]  
 $\implies \exists k. k \in \text{newn } R \wedge \text{nonce } s k = n$ "  
 <proof>

**lemma fresh\_rule:** "[evs' @ ev # evs  $\in$  tr p; wdef p; Nonce n  $\notin$  used evs;  
 Nonce n  $\in$  parts {msg ev}]  $\implies \exists R s. R \in p \wedge \text{ap}' s R = \text{ev}$ "  
 <proof>

**lemma fresh\_ruleD:** "[fresh p R' s' n Ks evs; keys R' s' n evs  $\subseteq$  Ks; wdef  
 p;  
 has\_only\_Says' p; evs  $\in$  tr p;  $\forall R k s. \text{nonce } s k = n \longrightarrow \text{Nonce } n \in \text{used evs}$   
 $\longrightarrow$   
 $R \in p \longrightarrow k \in \text{newn } R \longrightarrow \text{Nonce } n \in \text{parts } \{\text{apm}' s R\} \longrightarrow \text{apm}' s R \in \text{guard}$   
 $n \text{ Ks} \longrightarrow$   
 $\text{apm}' s R \in \text{parts (spies evs)} \longrightarrow \text{keys } R s n \text{ evs} \subseteq \text{Ks} \longrightarrow P] \implies P$ "  
 <proof>

## 41.8 safe keys

**definition safe :: "key set  $\Rightarrow$  msg set  $\Rightarrow$  bool"** where  
 "safe Ks G  $\equiv \forall K. K \in \text{Ks} \longrightarrow \text{Key } K \notin \text{analz } G$ "

**lemma safeD [dest]:** "[safe Ks G; K  $\in$  Ks]  $\implies \text{Key } K \notin \text{analz } G$ "  
 <proof>

**lemma safe\_insert:** "safe Ks (insert X G)  $\implies \text{safe Ks } G$ "

*<proof>*

**lemma** Guard\_safe: "[[Guard n Ks G; safe Ks G]]  $\implies$  Nonce n  $\notin$  analz G"  
*<proof>*

### 41.9 guardedness preservation

**definition** preserv :: "proto  $\Rightarrow$  keyfun  $\Rightarrow$  nat  $\Rightarrow$  key set  $\Rightarrow$  bool" where  
 "preserv p keys n Ks  $\equiv$  ( $\forall$  evs R' s' R s. evs  $\in$  tr p  $\longrightarrow$   
 Guard n Ks (spies evs)  $\longrightarrow$  safe Ks (spies evs)  $\longrightarrow$  fresh p R' s' n Ks evs  
 $\longrightarrow$   
 keys R' s' n evs  $\subseteq$  Ks  $\longrightarrow$  R  $\in$  p  $\longrightarrow$  ok evs R s  $\longrightarrow$  apm' s R  $\in$  guard n Ks)"

**lemma** preservD: "[[preserv p keys n Ks; evs  $\in$  tr p; Guard n Ks (spies evs);  
 safe Ks (spies evs); fresh p R' s' n Ks evs; R  $\in$  p; ok evs R s;  
 keys R' s' n evs  $\subseteq$  Ks]]  $\implies$  apm' s R  $\in$  guard n Ks"  
*<proof>*

**lemma** preservD': "[[preserv p keys n Ks; evs  $\in$  tr p; Guard n Ks (spies evs);  
 safe Ks (spies evs); fresh p R' s' n Ks evs; (l, Says A B X)  $\in$  p;  
 ok evs (l, Says A B X) s; keys R' s' n evs  $\subseteq$  Ks]]  $\implies$  apm s X  $\in$  guard n Ks"  
*<proof>*

### 41.10 monotonic keyfun

**definition** monoton :: "proto  $\Rightarrow$  keyfun  $\Rightarrow$  bool" where  
 "monoton p keys  $\equiv$   $\forall$  R' s' n ev evs. ev  $\#$  evs  $\in$  tr p  $\longrightarrow$   
 keys R' s' n evs  $\subseteq$  keys R' s' n (ev  $\#$  evs)"

**lemma** monotonD [dest]: "[[keys R' s' n (ev  $\#$  evs)  $\subseteq$  Ks; monoton p keys;  
 ev  $\#$  evs  $\in$  tr p]]  $\implies$  keys R' s' n evs  $\subseteq$  Ks"  
*<proof>*

### 41.11 guardedness theorem

**lemma** Guard\_tr [rule\_format]: "[[evs  $\in$  tr p; has\_only\_Says' p;  
 preserv p keys n Ks; monoton p keys; Guard n Ks (initState Spy)]]  $\implies$   
 safe Ks (spies evs)  $\longrightarrow$  fresh p R' s' n Ks evs  $\longrightarrow$  keys R' s' n evs  $\subseteq$  Ks  
 $\longrightarrow$   
 Guard n Ks (spies evs)"  
*<proof>*

### 41.12 useful properties for guardedness

**lemma** newn\_neq\_used: "[[Nonce n  $\in$  used evs; ok evs R s; k  $\in$  newn R]]  
 $\implies$  n  $\neq$  nonce s k"  
*<proof>*

**lemma** ok\_Guard: "[[ok evs R s; Guard n Ks (spies evs); x  $\in$  fst R; is\_Says  
 x]]  
 $\implies$  apm s (msg x)  $\in$  parts (spies evs)  $\wedge$  apm s (msg x)  $\in$  guard n Ks"  
*<proof>*

**lemma** *ok\_parts\_not\_new*: "[Y ∈ parts (spies evs); Nonce (nonce s n) ∈ parts {Y}];  
*ok evs R s* ⟹ n ∉ newn R"  
 ⟨proof⟩

### 41.13 uniqueness

**definition** *uniq* :: "proto ⇒ secfun ⇒ bool" **where**  
 "uniq p secret ≡ ∀ evs R R' n n' Ks s s'. R ∈ p ⟶ R' ∈ p ⟶  
 n ∈ newn R ⟶ n' ∈ newn R' ⟶ nonce s n = nonce s' n' ⟶  
 Nonce (nonce s n) ∈ parts {apm' s R} ⟶ Nonce (nonce s n) ∈ parts {apm' s' R'} ⟶  
 apm' s R ∈ guard (nonce s n) Ks ⟶ apm' s' R' ∈ guard (nonce s n) Ks ⟶  
 evs ∈ tr p ⟶ Nonce (nonce s n) ∉ analz (spies evs) ⟶  
 secret R n s Ks ∈ parts (spies evs) ⟶ secret R' n' s' Ks ∈ parts (spies evs) ⟶  
 secret R n s Ks = secret R' n' s' Ks"

**lemma** *uniqD*: "[uniq p secret; evs ∈ tr p; R ∈ p; R' ∈ p; n ∈ newn R; n' ∈ newn R';  
 nonce s n = nonce s' n'; Nonce (nonce s n) ∉ analz (spies evs);  
 Nonce (nonce s n) ∈ parts {apm' s R}; Nonce (nonce s n) ∈ parts {apm' s' R'};  
 secret R n s Ks ∈ parts (spies evs); secret R' n' s' Ks ∈ parts (spies evs);  
 apm' s R ∈ guard (nonce s n) Ks; apm' s' R' ∈ guard (nonce s n) Ks] ⟹  
 secret R n s Ks = secret R' n' s' Ks"  
 ⟨proof⟩

**definition** *ord* :: "proto ⇒ (rule ⇒ rule ⇒ bool) ⇒ bool" **where**  
 "ord p inff ≡ ∀ R R'. R ∈ p ⟶ R' ∈ p ⟶ ¬ inff R R' ⟶ inff R' R"

**lemma** *ordD*: "[ord p inff; ¬ inff R R'; R ∈ p; R' ∈ p] ⟹ inff R' R"  
 ⟨proof⟩

**definition** *uniq'* :: "proto ⇒ (rule ⇒ rule ⇒ bool) ⇒ secfun ⇒ bool" **where**  
 "uniq' p inff secret ≡ ∀ evs R R' n n' Ks s s'. R ∈ p ⟶ R' ∈ p ⟶  
 inff R R' ⟶ n ∈ newn R ⟶ n' ∈ newn R' ⟶ nonce s n = nonce s' n' ⟶  
 Nonce (nonce s n) ∈ parts {apm' s R} ⟶ Nonce (nonce s n) ∈ parts {apm' s' R'} ⟶  
 apm' s R ∈ guard (nonce s n) Ks ⟶ apm' s' R' ∈ guard (nonce s n) Ks ⟶  
 evs ∈ tr p ⟶ Nonce (nonce s n) ∉ analz (spies evs) ⟶  
 secret R n s Ks ∈ parts (spies evs) ⟶ secret R' n' s' Ks ∈ parts (spies evs) ⟶  
 secret R n s Ks = secret R' n' s' Ks"

**lemma** *uniq'D*: "[uniq' p inff secret; evs ∈ tr p; inff R R'; R ∈ p; R' ∈ p; n ∈ newn R;  
 n' ∈ newn R'; nonce s n = nonce s' n'; Nonce (nonce s n) ∉ analz (spies evs);  
 Nonce (nonce s n) ∈ parts {apm' s R}; Nonce (nonce s n) ∈ parts {apm' s' R'};  
 secret R n s Ks ∈ parts (spies evs); secret R' n' s' Ks ∈ parts (spies evs);  
 apm' s R ∈ guard (nonce s n) Ks; apm' s' R' ∈ guard (nonce s n) Ks] ⟹  
 secret R n s Ks = secret R' n' s' Ks"  
 ⟨proof⟩

**lemma** *uniq'\_imp\_uniq*: "[uniq' p inff secret; ord p inff] ⟹ uniq p secret"

*<proof>*

#### 41.14 Needham-Schroeder-Lowe

```

definition a :: agent where "a == Friend 0"
definition b :: agent where "b == Friend 1"
definition a' :: agent where "a' == Friend 2"
definition b' :: agent where "b' == Friend 3"
definition Na :: nat where "Na == 0"
definition Nb :: nat where "Nb == 1"

```

abbreviation

```

ns1 :: rule where
  "ns1 == ({}, Says a b (Crypt (pubK b) {Nonce Na, Agent a}))"

```

abbreviation

```

ns2 :: rule where
  "ns2 == ({Says a' b (Crypt (pubK b) {Nonce Na, Agent a})},
    Says b a (Crypt (pubK a) {Nonce Na, Nonce Nb, Agent b}))"

```

abbreviation

```

ns3 :: rule where
  "ns3 == ({Says a b (Crypt (pubK b) {Nonce Na, Agent a}),
    Says b' a (Crypt (pubK a) {Nonce Na, Nonce Nb, Agent b})},
    Says a b (Crypt (pubK b) (Nonce Nb)))"

```

inductive\_set ns :: proto where

```

  [iff]: "ns1 ∈ ns"
| [iff]: "ns2 ∈ ns"
| [iff]: "ns3 ∈ ns"

```

abbreviation (input)

```

ns3a :: event where
  "ns3a == Says a b (Crypt (pubK b) {Nonce Na, Agent a})"

```

abbreviation (input)

```

ns3b :: event where
  "ns3b == Says b' a (Crypt (pubK a) {Nonce Na, Nonce Nb, Agent b})"

```

definition keys :: "keyfun" where

```

"keys R' s' n evs == {priK' s' a, priK' s' b}"

```

lemma "monoton ns keys"

*<proof>*

definition secret :: "secfun" where

```

"secret R n s Ks ==
  (if R=ns1 then apm s (Crypt (pubK b) {Nonce Na, Agent a})
  else if R=ns2 then apm s (Crypt (pubK a) {Nonce Na, Nonce Nb, Agent b})
  else Number 0)"

```

definition inf :: "rule => rule => bool" where

```

"inf R R' == (R=ns1 | (R=ns2 & R'~ns1) | (R=ns3 & R'=ns3))"

```

```
lemma inf_is_ord [iff]: "ord ns inf"
<proof>
```

### 41.15 general properties

```
lemma ns_has_only_Says' [iff]: "has_only_Says' ns"
<proof>
```

```
lemma newn_ns1 [iff]: "newn ns1 = {Na}"
<proof>
```

```
lemma newn_ns2 [iff]: "newn ns2 = {Nb}"
<proof>
```

```
lemma newn_ns3 [iff]: "newn ns3 = {}"
<proof>
```

```
lemma ns_wdef [iff]: "wdef ns"
<proof>
```

### 41.16 guardedness for NSL

```
lemma "uniq ns secret  $\implies$  preserv ns keys n Ks"
<proof>
```

### 41.17 unicity for NSL

```
lemma "uniq' ns inf secret"
<proof>
```

```
end
```

## 42 Blanqui's "guard" concept: protocol-independent secrecy

```
theory Auth_Guard_Public
imports
  P1
  P2
  Guard_NS_Public
  Proto
begin

end
```