Tokeneer in Isabelle/UTP

Simon Foster, Mario Gleirscher, and Yakoub Nemouchi

June 24, 2019

Contents

1 Tokeneer in Isabelle/UTP

theory *Tokeneer*
imports
ZedLite.zedlite

UTP.utp-easy-parser
begin recall-syntax

2 Introduction

hide-const *dom*

named-theorems *tis-defs*

2.1 TIS Basic Types

type-synonym *TIME* = *nat*

abbreviation *zeroTime* ≡ *0*

datatype *PRESENCE* = *present* | *absent*

datatype *CLASS* = *unmarked* | *unclassified* | *restricted* | *confidential* | *secret* | *topsecret*

record *Clearance* =
class :: *CLASS*

consts *minClearance* :: *Clearance* × *Clearance* ⇒ *Clearance*

datatype *PRIVILEGE* = *userOnly* | *guard* | *securityOfficer* | *auditManager*

typedecl *USER*

consts *ISSUER* :: *USER set*

typedecl *FINGERPRINT*

typedecl *FINGERPRINTTEMPLATE*

alphabet *FingerprintTemplate* =
template :: *FINGERPRINTTEMPLATE*

2.2 Keys and Encryption

typedecl *KEYPART*

abbreviation *KEYPART* :: *KEYPART set* **where** *KEYPART* ≡ *UNIV*

2.3 Certificates, Tokens, and Enrolment Data

2.3.1 Certificates

typedecl *TOKENID*

record *CertificateId* =
  issuer :: *USER*

definition *CertificateId* :: *CertificateId set* **where**
[*upred-defs*, *tis-defs*]: *CertificateId* = {*c*. *issuer c* ∈ *ISSUER*}

record *Certificate* =
  cid :: *CertificateId*
  validityPeriod :: *TIME set*
  isValidatedBy :: *KEYPART option*

definition *Certificate* :: *'a Certificate-scheme set* **where**
[*upred-defs*, *tis-defs*]: *Certificate* = {*c*. *cid c* ∈ *CertificateId*}

record *IDCert* = *Certificate* +
  subject :: *USER*
  subjectPubK :: *KEYPART*

definition *IDCert* :: *'a IDCert-scheme set* **where**
[*upred-defs*, *tis-defs*]: *IDCert* = *Certificate*

definition *CAIdCert* :: *IDCert set* **where**
[*upred-defs*, *tis-defs*]: *CAIdCert* = {*c* ∈ *IDCert*. *isValidatedBy c* = *Some*(*subjectPubK c*)}

record *AttCertificate* = *Certificate* +
  baseCertId :: *CertificateId*
  atokenID :: *TOKENID*

definition *AttCertificate* :: *'a AttCertificate-scheme set* **where**
[*upred-defs*, *tis-defs*]: *AttCertificate* = *Certificate*

record *PrivCert* = *AttCertificate* +
  role :: *PRIVILEGE*
  clearance :: *Clearance*

definition *PrivCert* :: *PrivCert set* **where**
[*upred-defs*, *tis-defs*]: *PrivCert* = *AttCertificate*

type-synonym *AuthCert* = *PrivCert*

abbreviation *AuthCert* :: *AuthCert set* **where** *AuthCert* ≡ *PrivCert*

record *IandACert* = *AttCertificate* +
  template :: *FingerprintTemplate*

definition *IandACert* :: *IandACert set* **where**

[*upred-defs*, *tis-defs*]: *IandACert* = *AttCertificate*

2.3.2 Tokens

record *Token* =
  tokenID :: *TOKENID*
  idCert :: *IDCert*
  privCert :: *PrivCert*
  iandACert :: *IandACert*
  authCert :: *AuthCert option*

definition *Token* :: *Token set* **where**
[*upred-defs*, *tis-defs*]:
Token = {*c*. *idCert c* ∈ *IDCert* ∧
  privCert c ∈ *PrivCert* ∧
  iandACert c ∈ *IandACert* ∧
  (∀ *x*. *authCert c* = *Some*(*x*) ⟶ *x* ∈ *AuthCert*)
  }

definition *ValidToken* :: *Token set* **where**
[*upred-defs*, *tis-defs*]:
ValidToken =
  {*t*∈*Token*. *baseCertId* (*privCert t*) = *cid* (*idCert t*)
    ∧ *baseCertId* (*iandACert t*) = *cid* (*idCert t*)
    ∧ *atokenID* (*privCert t*) = *tokenID t*
    ∧ *atokenID* (*iandACert t*) = *tokenID t*}

definition *TokenWithValidAuth* :: *Token set* **where**
[*upred-defs*, *tis-defs*]:
TokenWithValidAuth =
  {*t*. *authCert t* ≠ *None* ∧
    atokenID (*the* (*authCert t*)) = *tokenID t* ∧
    baseCertId (*the* (*authCert t*)) = *cid* (*idCert t*)}

definition *CurrentToken* :: *TIME* ⇒ *Token set* **where**
[*upred-defs*, *tis-defs*]:
CurrentToken now =
  (*ValidToken* ∩
    {*t*. *now* ∈ *validityPeriod* (*idCert t*)
      ∩ *validityPeriod* (*privCert t*)
      ∩ *validityPeriod* (*iandACert t*)})

2.3.3 Enrolment Data

record *Enrol* =
  idStationCert :: *IDCert*
  issuerCerts :: *IDCert set*

We had to add two extra clauses to Enrol here that we're specified in the Tokeneer Z-schema, namely that (1) all issuer certificates correspond to ele-

ments of *ISSUER* and (2) the subjects uniquely identify one issue certificate. Without these, it is not possible to update the key store and maintain the partial function there.

definition *Enrol* :: *Enrol set* **where**
[*upred-defs*, *tis-defs*]:
Enrol = {*e*. *idStationCert e* ∈ *issuerCerts e* ∧
subject ‘ *issuerCerts e* ⊆ *ISSUER* ∧
(∀ *c* ∈ *issuerCerts e*. ∀ *d* ∈ *issuerCerts e*. *subject c* = *subject d* ⟶ *c* = *d*)}

definition *ValidEnrol* :: *Enrol set* **where**
[*upred-defs*, *tis-defs*]:
ValidEnrol = (*Enrol* ∩
{*e*. *issuerCerts e* ∩ *CAIdCert* ≠ {} ∧
(∀ *cert* ∈ *issuerCerts e*. *isValidatedBy cert* ≠ *None* ∧
(∃ *issuerCert* ∈ *issuerCerts e*.
issuerCert ∈ *CAIdCert* ∧
the(*isValidatedBy cert*) = *subjectPubK issuerCert* ∧
issuer (*cid cert*) = *subject issuerCert*))})

2.4 World Outside the ID Station

2.4.1 Real World Types and Entities (1)

datatype *DOOR* = *dopen* | *closed*
datatype *LATCH* = *unlocked* | *locked*
datatype *ALARM* = *silent* | *alarming*
datatype *DISPLAYMESSAGE* = *blank* | *welcom* | *insertFinger* | *openDoor* | *wait* | *removeToken* | *tokenUpdateFailed* | *doorUnlocked*

datatype *FINGERPRINTTRY* = *noFP* | *badFP* | *goodFP FINGERPRINT*

alphabet *Finger* =
currentFinger :: *FINGERPRINTTRY*
fingerPresence :: *PRESENCE*

abbreviation *Finger* :: *Finger upred* **where** *Finger* ≡ *true*

alphabet *DoorLatchAlarm* =
currentTime :: *TIME*
currentDoor :: *DOOR*
currentLatch :: *LATCH*
doorAlarm :: *ALARM*
latchTimeout :: *TIME*
alarmTimeout :: *TIME*

definition *DoorLatchAlarm* :: *DoorLatchAlarm upred* **where**
[*upred-defs*, *tis-defs*]:
DoorLatchAlarm = (

$(currentLatch = \ll locked \gg \Leftrightarrow currentTime \geq latchTimeout) \wedge$
$(doorAlarm = \ll alarming \gg \Leftrightarrow$
$(currentDoor = \ll dopen \gg$
$\wedge\ currentLatch = \ll locked \gg$
$\wedge\ currentTime \geq alarmTimeout))$
$)_e$

3 The Token ID Station

3.1 Configuration Data

consts *maxSupportedLogSize* :: *nat*

alphabet *Config* =
alarmSilentDuration :: *TIME*
latchUnlockDuration :: *TIME*
tokenRemovalDuration :: *TIME*
enclaveClearance :: *Clearance*
authPeriod :: *PRIVILEGE* $\Rightarrow$ *TIME* $\Rightarrow$ *TIME set*
entryPeriod :: *PRIVILEGE* $\Rightarrow$ *CLASS* $\Rightarrow$ *TIME set*
minPreservedLogSize :: *nat*
alarmThresholdSize :: *nat*

definition *Config* :: *Config upred* **where**
[*upred-defs*, *tis-defs*]:
Config = (*alarmThresholdSize* $<$ *minPreservedLogSize* $\wedge$
minPreservedLogSize $\leq \ll maxSupportedLogSize \gg \wedge$
latchUnlockDuration $> 0 \wedge$
alarmSilentDuration $> 0)_e$

3.2 AuditLog

typedecl *AuditEvent*
typedecl *AuditUser*

alphabet *Audit* =
auditTime :: *TIME*
auditEvent :: *AuditEvent*
auditUser :: *AuditUser*
sizeElement :: *nat*

3.2.1 Real World Types and Entities (2)

datatype *FLOPPY* = *noFloppy* | *emptyFloppy* | *badFloppy* | *enrolmentFile* (*enrolmentFile-of*: *Enrol*) |
auditFile Audit set | *configFile Config*

definition *FLOPPY* :: *FLOPPY upred* **where**
[*upred-defs*, *tis-defs*]:

$FLOPPY = (\forall\ e \cdot \mathbf{v} = \ll enrolmentFile\ e\gg \Rightarrow \ll e \in ValidEnrol\gg)_e$

alphabet *Floppy* =
  currentFloppy :: *FLOPPY*
  writtenFloppy :: *FLOPPY*
  floppyPresence :: *PRESENCE*

definition *Floppy* :: *Floppy upred* **where**
[*upred-defs*, *tis-defs*]:
Floppy = (*FLOPPY* $\oplus_p$ *currentFloppy* $\wedge$ *FLOPPY* $\oplus_p$ *writtenFloppy*)

definition [*upred-defs*, *tis-defs*]: *ADMINPRIVILEGE* = {*guard*, *auditManager*, *securityOfficer*}
datatype *ADMINOP* = *archiveLog* | *updateConfigData* | *overrideLock* | *shutdownOp*

datatype *KEYBOARD* = *noKB* | *badKB* | *keyedOps* (*ofKeyedOps*: *ADMINOP*)

alphabet *Keyboard* =
  currentKeyedData :: *KEYBOARD*
  keyedDataPresence :: *PRESENCE*

abbreviation *Keyboard* :: *Keyboard upred* **where** *Keyboard* $\equiv$ *true*

3.3 System Statistics

alphabet *Stats* =
  successEntry :: *nat*
  failEntry :: *nat*
  successBio :: *nat*
  failBio :: *nat*

abbreviation *Stats* :: *Stats upred* **where** *Stats* $\equiv$ *true*

3.4 Key Store

alphabet *KeyStore* =
  issuerKey :: *USER* $\leftrightarrow$ *KEYPART*
  ownName :: *USER option*

definition *KeyStore* :: *KeyStore upred* **where**
[*upred-defs*, *tis-defs*]:
KeyStore =
  (*issuerKey* $\in \ll ISSUER \rightarrow_r KEYPART\gg \wedge$
   $udom(issuerKey) \subseteq \ll ISSUER\gg \wedge$
   $(ownName \neq \ll None\gg \Rightarrow the(ownName) \in udom(issuerKey)))_e$

definition *CertIssuerKnown* :: *'a Certificate-scheme* $\Rightarrow$ *KeyStore upred* **where**
[*upred-defs*, *tis-defs*]:
CertIssuerKnown c =
  (*KeyStore* $\wedge$

$(\ll c \in Certificate\gg \wedge$
$\ll issuer\ (cid\ c)\gg \in udom(issuerKey))_e)$

definition *CertOK* :: *'a Certificate-scheme* ⇒ *KeyStore upred* **where**
[*upred-defs*, *tis-defs*]:
CertOK c =
$(CertIssuerKnown\ c \wedge$
$(Some(issuerKey[\ll issuer\ (cid\ c)\gg]) = \ll isValidatedBy\ c\gg)_e)$

definition *CertIssuerIsThisTIS* :: *'a Certificate-scheme* ⇒ *KeyStore upred* **where**
[*upred-defs*, *tis-defs*]:
CertIssuerIsThisTIS c =
$(KeyStore \wedge$
$\ll c \in Certificate\gg \wedge$
$(ownName \neq \ll None\gg \wedge$
$\ll issuer\ (cid\ c)\gg = the(ownName))_e)$

definition *AuthCertOK* :: *'a Certificate-scheme* ⇒ *KeyStore upred* **where**
[*upred-defs*, *tis-defs*]: *AuthCertOK c* = (*CertIssuerIsThisTIS c* ∧ *CertOK c*)

definition *oldestLogTime* :: *Audit set* ⇒ *TIME* **where**
[*upred-defs*, *tis-defs*]:
oldestLogTime lg = (*Min* ($get_{auditTime}$ ‘ *lg*))

definition *newestLogTime* :: *Audit set* ⇒ *TIME* **where**
[*upred-defs*, *tis-defs*]:
newestLogTime lg = (*Max* ($get_{auditTime}$ ‘ *lg*))

lemma *newestLogTime-union*: ⟦ *finite A*; *A* ≠ {}; *finite B*; *B* ≠ {} ⟧ ⟹ *newestLogTime* (*A* ∪ *B*) ≥ *newestLogTime A*
by (*simp add*: *newestLogTime-def*)

lemma *oldestLogTime-union*: ⟦ *finite A*; *A* ≠ {}; *finite B*; *B* ≠ {} ⟧ ⟹ *oldestLogTime* (*A* ∪ *B*) ≤ *oldestLogTime A*
by (*simp add*: *oldestLogTime-def*)

3.5 Administration

alphabet *Admin* =
rolePresent :: *PRIVILEGE option*
availableOps :: *ADMINOP set*
currentAdminOp :: *ADMINOP option*

definition *Admin* :: *Admin upred* **where**
[*upred-defs*, *tis-defs*]:
Admin =
$((rolePresent \neq \ll None\gg \Rightarrow the(rolePresent) \in \ll ADMINPRIVILEGE\gg) \wedge$
$(rolePresent = \ll None\gg \Rightarrow availableOps = \{\}) \wedge$
$(rolePresent \neq \ll None\gg \wedge the(rolePresent) = \ll guard\gg \Rightarrow availableOps =$

$\{\ll overrideLock\gg\}) \wedge$
$(rolePresent \neq \ll None\gg \wedge the(rolePresent) = \ll auditManager\gg \Rightarrow availableOps$
$= \{\ll archiveLog\gg\}) \wedge$
$(rolePresent \neq \ll None\gg \wedge the(rolePresent) = \ll securityOfficer\gg$
$\Rightarrow availableOps = \{\ll updateConfigData\gg,\ll shutdownOp\gg\}) \wedge$
$(currentAdminOp \neq \ll None\gg \Rightarrow$
$the(currentAdminOp) \in availableOps \wedge rolePresent \neq \ll None\gg)$
$)_e$

3.6 AuditLog (2)

alphabet *AuditLog* =
auditLog :: *Audit set*
auditAlarm :: *ALARM*

abbreviation *AuditLog* :: *AuditLog upred* **where**
AuditLog ≡ *true*

3.6.1 Real World Types and Entities (3)

datatype *SCREENTEXT* = *clear* | *welcomeAdmin* | *busy* | *removeAdminToken*
| *closeDoor* |
requestAdminOp | *doingOp* | *invalidRequest* | *invalidData* |
insertEnrolmentData | *validatingEnrolmentData* | *enrolmentFailed* |
archiveFailed | *insertBlankFloppy* | *insertConfigData* |
displayStats Stats | *displayConfigData Config*

alphabet *Screen* =
screenStats :: *SCREENTEXT*
screenMsg :: *SCREENTEXT*
screenConfig :: *SCREENTEXT*

datatype *TOKENTRY* = *noT* | *badT* | *goodT* (*ofGoodT*: *Token*)

alphabet *UserToken* =
currentUserToken :: *TOKENTRY*
userTokenPresence :: *PRESENCE*

definition *UserToken* :: *UserToken upred* **where**
[*upred-defs*, *tis-defs*]:
UserToken = $((\exists\ t \cdot currentUserToken = goodT(\ll t\gg)) \Rightarrow ofGoodT(currentUserToken)$
$\in \ll Token\gg)_e$

alphabet *AdminToken* =
currentAdminToken :: *TOKENTRY*
adminTokenPresence :: *PRESENCE*

definition *AdminToken* :: *AdminToken upred* **where**
[*upred-defs*, *tis-defs*]:

$AdminToken = ((\exists\ t \cdot currentAdminToken = goodT(\ll t \gg)) \Rightarrow ofGoodT(currentAdminToken) \in \ll Token \gg)_e$

3.7 Internal State

datatype $STATUS = quiescent \mid gotUserToken \mid waitingFinger \mid gotFinger \mid waitingUpdateToken \mid$
$waitingEntry \mid waitingRemoveTokenSuccess \mid waitingRemoveTokenFail$

datatype $ENCLAVESTATUS = notEnrolled \mid waitingEnrol \mid waitingEndEnrol \mid enclaveQuiescent \mid$
$gotAdminToken \mid waitingRemoveAdminTokenFail \mid waitingStartAdminOp \mid waitingFinishAdminOp \mid$
$shutdown$

alphabet $Internal =$
$status :: STATUS$
$enclaveStatus :: ENCLAVESTATUS$
$tokenRemovalTimeout :: TIME$

definition $Internal :: Internal\ upred$ **where**
[*upred-defs*, *tis-defs*]:
$Internal = true$

3.8 The Whole Token ID Station

alphabet $IDStation =$
$iuserToken :: UserToken$
$iadminToken :: AdminToken$
$ifinger :: Finger$
$doorLatchAlarm :: DoorLatchAlarm$
$ifloppy :: Floppy$
$ikeyboard :: Keyboard$
$config :: Config$
$stats :: Stats$
$keyStore :: KeyStore$
$admin :: Admin$
$audit :: AuditLog$
$internal :: Internal$
$currentDisplay :: DISPLAYMESSAGE$
$currentScreen :: Screen$

definition $UserTokenWithOKAuthCert :: IDStation\ upred$ **where**
[*upred-defs*, *tis-defs*]:
$UserTokenWithOKAuthCert =$
$(\&iuserToken{:}currentUserToken \in_u \ll range(goodT) \gg \wedge$
$(\exists\ t{\in}\ll TokenWithValidAuth \gg \cdot$
$(\ll goodT(t) \gg =_u \&iuserToken{:}currentUserToken$
$\wedge\ \&doorLatchAlarm{:}currentTime \in_u \ll validityPeriod\ (the(authCert\ t)) \gg$
$\wedge\ (\exists\ c \in \ll IDCert \gg \cdot \ll c = idCert\ t \gg \wedge\ CertOK\ c) \oplus_p keyStore$

$\wedge$ ($\exists$ $c \in \ll AuthCert \gg \cdot \ll c = the\ (authCert\ t) \gg \wedge\ AuthCertOK\ c) \oplus_p$ *keyStore*))
)

definition *UserTokenOK* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
UserTokenOK =
(&*iuserToken*:*currentUserToken* $\in_u \ll range(goodT) \gg \wedge$
($\exists$ $t \cdot$
($\ll goodT(t) \gg =_u$ &*iuserToken*:*currentUserToken*
$\wedge \ll t \in CurrentToken\ ti \gg [\![ti \rightarrow$ &*doorLatchAlarm*:*currentTime*$]\!]$
$\wedge$ ($\exists$ $c \in \ll IDCert \gg \cdot \ll c = idCert\ t \gg \wedge\ CertOK\ c) \oplus_p\ keyStore$
$\wedge$ ($\exists$ $c \in \ll PrivCert \gg \cdot \ll c = privCert\ t \gg \wedge\ CertOK\ c) \oplus_p\ keyStore$
$\wedge$ ($\exists$ $c \in \ll IandACert \gg \cdot \ll c = iandACert\ t \gg \wedge\ CertOK\ c) \oplus_p\ keyStore$))
)

definition *AdminTokenOK* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
AdminTokenOK =
(&*iadminToken*:*currentAdminToken* $\in_u \ll range(goodT) \gg \wedge$
($\exists$ $t \in \ll TokenWithValidAuth \gg \cdot$
($\ll goodT(t) \gg =_u$ &*iadminToken*:*currentAdminToken*
$\wedge \ll t \in CurrentToken\ ti \gg [\![ti \rightarrow$ &*doorLatchAlarm*:*currentTime*$]\!]$
$\wedge$ ($\exists$ $c \in \ll IDCert \gg \cdot \ll c = idCert\ t \gg \wedge\ CertOK\ c) \oplus_p\ keyStore$
$\wedge$ ($\exists$ $c \in \ll AuthCert \gg \cdot \ll Some\ c = authCert\ t \gg \wedge\ AuthCertOK\ c$
$\wedge \ll role\ c \in ADMINPRIVILEGE \gg) \oplus_p\ keyStore$
))
)

definition *FingerOK* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
FingerOK = (
Finger $\oplus_p$ *ifinger* $\wedge$
UserToken $\oplus_p$ *iuserToken* $\wedge$
&*ifinger*:*currentFinger* $\in_u \ll range(goodFP) \gg$)

definition *IDStation-inv1* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv1 =
(*internal*:*status* $\in$
{$\ll gotFinger \gg, \ll waitingFinger \gg, \ll waitingUpdateToken \gg, \ll waitingEntry \gg, \ll waitingRemoveTokenSuccess \gg$}
$\Rightarrow$ (@*UserTokenWithOKAuthCert* $\vee$ @*UserTokenOK*))$_e$

definition *IDStation-inv2* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv2 =
(*admin*:*rolePresent* $\neq \ll None \gg \Rightarrow$ @*AdminTokenOK*)$_e$

definition *IDStation-inv3* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv3 =
(*internal*:*enclaveStatus* ∉ {≪*notEnrolled*≫, ≪*waitingEnrol*≫, ≪*waitingEndEnrol*≫} ⇒
keyStore:*ownName* ≠ ≪*None*≫)$_e$

definition *IDStation-inv4* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv4 =
(*internal*:*enclaveStatus* ∈ {≪*waitingStartAdminOp*≫, ≪*waitingFinishAdminOp*≫}
⇔ *admin*:*currentAdminOp* ≠ ≪*None*≫)$_e$

definition *IDStation-inv5* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv5 =
(*admin*:*currentAdminOp* ≠ ≪*None*≫ ∧ *the*(*admin*:*currentAdminOp*) ∈ {≪*shutdownOp*≫, ≪*overrideLock*≫}
⇒ *internal*:*enclaveStatus* = ≪*waitingStartAdminOp*≫)$_e$

definition *IDStation-inv6* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv6 = (*internal*:*enclaveStatus* = ≪*gotAdminToken*≫ ⇒ *admin*:*rolePresent* = ≪*None*≫)$_e$

definition *IDStation-inv7* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv7 = (*currentScreen*:*screenStats* = ≪*displayStats*≫[*stats*])$_e$

definition *IDStation-inv8* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv8 = (*currentScreen*:*screenConfig* = ≪*displayConfigData*≫[*config*])$_e$

Extra Invariant (1):

definition *IDStation-inv9* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv9 =
(*internal*:*status* ∈
{≪*waitingEntry*≫, ≪*waitingRemoveTokenSuccess*≫}
⇒ (@*UserTokenWithOKAuthCert* ∨ @*FingerOK*))$_e$

Extra Invariant (2): If an admin token is present, and a role has been validated then the role matches the one present on the authorisation certificate.

definition *IDStation-inv10* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv10 =
(*iadminToken*:*adminTokenPresence* = ≪*present*≫ ∧ *admin*:*rolePresent* ≠ ≪*None*≫
⇒ *admin*:*rolePresent* = *Some*(*role*(*the*(*authCert*(*ofGoodT*(*iadminToken*:*currentAdminToken*))))))$_e$

definition
[*upred-defs*, *tis-defs*]:
IDStation-wf =
(*DoorLatchAlarm* $\oplus_p$ *doorLatchAlarm* $\wedge$
Floppy $\oplus_p$ *ifloppy* $\wedge$
KeyStore $\oplus_p$ *keyStore* $\wedge$
Admin $\oplus_p$ *admin* $\wedge$
Config $\oplus_p$ *config* $\wedge$
AdminToken $\oplus_p$ *iadminToken* $\wedge$
UserToken $\oplus_p$ *iuserToken*)

definition
[*upred-defs*, *tis-defs*]:
IDStation-inv = (
IDStation-inv1 $\wedge$
IDStation-inv2 $\wedge$
IDStation-inv3 $\wedge$
IDStation-inv4 $\wedge$
IDStation-inv5 $\wedge$
IDStation-inv6 $\wedge$
IDStation-inv7 $\wedge$
IDStation-inv8 $\wedge$
IDStation-inv9 $\wedge$
IDStation-inv10)

definition *IDStation* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation =
(
IDStation-wf $\wedge$
IDStation-inv
)

lemma *IDStation-correct-intro*:
assumes $\{\!|$*DoorLatchAlarm* $\oplus_p$ *doorLatchAlarm* $\wedge$ *Floppy* $\oplus_p$ *ifloppy* $\wedge$ *KeyStore* $\oplus_p$ *keyStore* $\wedge$ *Admin* $\oplus_p$ *admin* $\wedge$
Config $\oplus_p$ *config* $\wedge$ *AdminToken* $\oplus_p$ *iadminToken* $\wedge$ *UserToken* $\oplus_p$ *iuserToken*$|\!\}$
P
$\{\!|$*DoorLatchAlarm* $\oplus_p$ *doorLatchAlarm* $\wedge$ *Floppy* $\oplus_p$ *ifloppy* $\wedge$ *KeyStore* $\oplus_p$ *keyStore* $\wedge$ *Admin* $\oplus_p$ *admin* $\wedge$
Config $\oplus_p$ *config* $\wedge$ *AdminToken* $\oplus_p$ *iadminToken* $\wedge$ *UserToken* $\oplus_p$ *iuserToken*$|\!\}_u$
$\{\!|$*IDStation-inv*$|\!\}$ *P* $\{\!|$*IDStation-inv*$|\!\}_u$
shows $\{\!|$*IDStation*$|\!\}$ *P* $\{\!|$*IDStation*$|\!\}_u$
using *assms*
proof −
have *f1*: (*IDStation-inv* $\wedge$ *DoorLatchAlarm* $\oplus_p$ *doorLatchAlarm* $\wedge$ *Floppy* $\oplus_p$ *ifloppy* $\wedge$ *KeyStore* $\oplus_p$ *keyStore* $\wedge$ *Admin* $\oplus_p$ *admin* $\wedge$ *Config* $\oplus_p$ *config* $\wedge$ *Ad-*

minToken $\oplus_p$ *iadminToken* $\wedge$ *UserToken* $\oplus_p$ *iuserToken*) = *IDStation*
by (*simp add*: *IDStation-def IDStation-wf-def utp-pred-laws.inf-commute utp-pred-laws.inf-left-commute*)
then have *f2*: $\{\!|IDStation|\!\}$ *P* $\{\!|DoorLatchAlarm \oplus_p doorLatchAlarm \wedge Floppy \oplus_p ifloppy \wedge KeyStore \oplus_p keyStore \wedge Admin \oplus_p admin \wedge Config \oplus_p config \wedge AdminToken \oplus_p iadminToken \wedge UserToken \oplus_p iuserToken|\!\}_u$
by (*metis* (*no-types*) *assms*(*1*) *hoare-r-weaken-pre*(*2*))
have $\{\!|IDStation|\!\}$ *P* $\{\!|IDStation\text{-}inv|\!\}_u$
using *f1* **by** (*metis* (*no-types*) *assms*(*2*) *hoare-r-weaken-pre*(*2*) *utp-pred-laws.inf-commute*)
then show *?thesis*
using *f2 f1*
using *hoare-r-conj* **by** *fastforce*
qed

lemma *IDStation-inv-intro*:
assumes
$\{\!|IDStation\text{-}inv1|\!\}P\{\!|IDStation\text{-}inv1|\!\}_u$
$\{\!|IDStation\text{-}inv2|\!\}P\{\!|IDStation\text{-}inv2|\!\}_u$
$\{\!|IDStation\text{-}inv3|\!\}P\{\!|IDStation\text{-}inv3|\!\}_u$
$\{\!|IDStation\text{-}inv4|\!\}P\{\!|IDStation\text{-}inv4|\!\}_u$
$\{\!|IDStation\text{-}inv5|\!\}P\{\!|IDStation\text{-}inv5|\!\}_u$
$\{\!|IDStation\text{-}inv6|\!\}P\{\!|IDStation\text{-}inv6|\!\}_u$
$\{\!|IDStation\text{-}inv7|\!\}P\{\!|IDStation\text{-}inv7|\!\}_u$
$\{\!|IDStation\text{-}inv8|\!\}P\{\!|IDStation\text{-}inv8|\!\}_u$
$\{\!|IDStation\text{-}inv9|\!\}P\{\!|IDStation\text{-}inv9|\!\}_u$
$\{\!|IDStation\text{-}inv10|\!\}P\{\!|IDStation\text{-}inv10|\!\}_u$
shows $\{\!|IDStation\text{-}inv|\!\}P\{\!|IDStation\text{-}inv|\!\}_u$
by (*simp add*: *IDStation-inv-def assms hoare-r-conj hoare-r-weaken-pre*(*1*) *hoare-r-weaken-pre*(*2*))

4 Operations Interfacing to the ID Station (1)

alphabet *TISControlledRealWorld* =
latch :: *LATCH*
alarm :: *ALARM*
display :: *DISPLAYMESSAGE*
screen :: *Screen*

abbreviation *TISControlledRealWorld* :: *TISControlledRealWorld upred* **where**
TISControlledRealWorld $\equiv$ *true*

alphabet *TISMonitoredRealWorld* =
now :: *TIME*
door :: *DOOR*
finger :: *FINGERPRINTTRY*
userToken :: *TOKENTRY*
adminToken :: *TOKENTRY*
floppy :: *FLOPPY*
keyboard :: *KEYBOARD*

alphabet *RealWorld* =

controlled :: *TISControlledRealWorld*
monitored :: *TISMonitoredRealWorld*

definition *RealWorld* :: *RealWorld upred* **where**
[*upred-defs*, *tis-defs*]:
RealWorld = *true*

4.1 Real World Changes

We permit any part of the real-world to change without constraint, except time must monotonically increase.

definition *RealWorldChanges* :: *RealWorld hrel* **where**
[*upred-defs*, *tis-defs*]:
RealWorldChanges =
(⋁ *t* · *monitored*:*now* := *monitored*:*now* + ≪*t*≫ ;;
monitored:*door* := ∗ ;; *monitored*:*finger* := ∗ ;;
monitored:*userToken* := ∗ ;; *monitored*:*adminToken* := ∗ ;;
monitored:*floppy* := ∗ ;; *monitored*:*keyboard* := ∗ ;;
controlled:*latch* := ∗ ;; *controlled*:*alarm* := ∗ ;;
controlled:*display* := ∗ ;; *controlled*:*screen* := ∗)

lemma *RealWorldChanges-original*: *RealWorldChanges* = ($*monitored*:*now*´ $\geq_u$ $*monitored*:*now*)
by (*rel-auto*, *simp add*: *nat-le-iff-add*)

lemma *pre-RealWorldChanges*: *Dom*(*RealWorldChanges*) = *true*
by (*rel-auto*)

alphabet *SystemState* =
idStation :: *IDStation*
realWorld :: *RealWorld*

5 Internal Operations

definition *AddElementsToLog* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]: *AddElementsToLog* = *true*

definition *AuditAlarm* :: *IDStation hrel* **where** [*upred-defs*, *tis-defs*]: *AuditAlarm* = *true*
definition *AuditLatch* :: *IDStation hrel* **where** [*upred-defs*, *tis-defs*]: *AuditLatch* = *true*
definition *AuditDoor* :: *IDStation hrel* **where** [*upred-defs*, *tis-defs*]: *AuditDoor* = *true*
definition *AuditLogAlarm* :: *IDStation hrel* **where** [*upred-defs*, *tis-defs*]: *AuditLogAlarm* = *true*

definition *AuditScreen* :: *IDStation hrel* **where** [*upred-defs*, *tis-defs*]: *AuditScreen* = *true*
definition *AuditDisplay* :: *IDStation hrel* **where** [*upred-defs*, *tis-defs*]: *AuditDisplay* = *true*
definition *NoChange* :: *IDStation hrel* **where** [*upred-defs*, *tis-defs*]: *NoChange* = *true*

definition *LogChange* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
LogChange = (*AuditAlarm* ∨ *AuditLatch* ∨ *AuditDoor* ∨ *AuditLogAlarm* ∨ *AuditScreen* ∨ *AuditDisplay* ∨ *NoChange*)

5.1 Updating System Statistics

definition *AddSuccessfulEntryToStats* :: *Stats hrel* **where**
[*upred-defs*, *tis-defs*]:
AddSuccessfulEntryToStats =
(Δ[*Stats*] ∧
$*failEntry*´ =$_u$ $*failEntry* ∧
$*successEntry*´ =$_u$ $*successEntry* + *1* ∧
$*failBio*´ =$_u$ $*failBio* ∧
$*successBio*´ =$_u$ $*successBio*)

lemma *AddSuccessfulEntryToStats-prog-def*:
AddSuccessfulEntryToStats = (*successEntry* := *successEntry* + *1*)
by (*rel-auto*)

definition *AddFailedEntryToStats* :: *Stats hrel* **where**
[*upred-defs*, *tis-defs*]:
AddFailedEntryToStats =
(Δ[*Stats*] ∧
$*failEntry*´ =$_u$ $*failEntry* + *1* ∧
$*successEntry*´ =$_u$ $*successEntry* ∧
$*failBio*´ =$_u$ $*failBio* ∧
$*successBio*´ =$_u$ $*successBio*)

lemma *AddFailedEntryToStats-prog-def*:
AddFailedEntryToStats = (*failEntry* := *failEntry* + *1*)
by (*rel-auto*)

definition *AddSuccessfulBioEntryToStats* :: *Stats hrel* **where**
[*upred-defs*, *tis-defs*]:
AddSuccessfulBioEntryToStats =
(Δ[*Stats*] ∧
$*failEntry*´ =$_u$ $*failEntry* ∧
$*successEntry*´ =$_u$ $*successEntry* ∧
$*failBio*´ =$_u$ $*failBio* ∧
$*successBio*´ =$_u$ $*successBio* + *1*)

lemma *AddSuccessfulBioEntryToStats-prog-def*:
AddSuccessfulBioEntryToStats = (*successBio* := *successBio* + *1*)
by (*rel-auto*)

definition *AddFailedBioEntryToStats* :: *Stats hrel* **where**
[*upred-defs*, *tis-defs*]:
AddFailedBioEntryToStats =
(Δ[*Stats*] ∧
$*failEntry*´ =$_u$ $*failEntry* ∧
$*successEntry*´ =$_u$ $*successEntry* ∧
$*failBio*´ =$_u$ $*failBio* + *1* ∧
$*successBio*´ =$_u$ $*successBio*)

lemma *AddFailedBioEntryToStats-prog-def*:
AddFailedBioEntryToStats = (*failBio* := *failBio* + *1*)
by (*rel-auto*)

5.2 Operating the Door

definition *UnlockDoor* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
UnlockDoor =
doorLatchAlarm:*latchTimeout* := *doorLatchAlarm*:*currentTime* + *config*:*latchUnlockDuration*
;;
doorLatchAlarm:*alarmTimeout* := *doorLatchAlarm*:*currentTime* + *config*:*latchUnlockDuration*
+ *config*:*alarmSilentDuration* ;;
doorLatchAlarm:*currentLatch* := «*unlocked*» ;;
doorLatchAlarm:*doorAlarm* := «*silent*»

lemma *UnlockDoor-correct*:
{|*IDStation*|}*UnlockDoor*{|*IDStation*|}$_u$
apply (*rule IDStation-correct-intro*)
apply (*simp-all add*: *tis-defs*)
apply (*hoare-auto*)
apply (*hoare-auto*)
done

definition *LockDoor* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
LockDoor =
doorLatchAlarm:*latchTimeout* := *doorLatchAlarm*:*currentTime* ;;
doorLatchAlarm:*alarmTimeout* := *doorLatchAlarm*:*currentTime* ;;
doorLatchAlarm:*currentLatch* := «*locked*» ;;
doorLatchAlarm:*doorAlarm* := «*silent*»

5.3 Certificate Operations

5.3.1 Generating Authorisation Certificates

definition *NewAuthCert* :: - ⇒ - ⇒ *TIME* ⇒ *IDStation upred* **where**

[*upred-defs*, *tis-defs*]:
NewAuthCert token newAuthCert curTime = (
  $\ll$*token* $\in$ *ValidToken*$\gg$ $\wedge$
  KeyStore $\oplus_p$ *keyStore* $\wedge$
  Config $\oplus_p$ *config* $\wedge$

  &*keyStore*:*ownName* $\neq_u$ *None*$_u$ $\wedge$

  $\ll$*issuer* (*cid newAuthCert*)$\gg$ $=_u$ *the*$_u$(&*keyStore*:*ownName*) $\wedge$
  $\ll$*validityPeriod newAuthCert*$\gg$ $=_u$ &*config*:*authPeriod*($\ll$*role* (*privCert token*)$\gg$)$_a$($\ll$*curTime*$\gg$)$_a$ $\wedge$
  $\ll$*baseCertId newAuthCert* = *cid* (*idCert token*)$\gg$ $\wedge$
  $\ll$*atokenID newAuthCert* = *tokenID token*$\gg$ $\wedge$
  $\ll$*role newAuthCert* = *role* (*privCert token*)$\gg$ $\wedge$
  $\ll$*clearance newAuthCert*$\gg$ $=_u$ $\ll$*minClearance*$\gg$(&*config*:*enclaveClearance*, $\ll$*clearance* (*privCert token*)$\gg$)$_a$ $\wedge$
  $\ll$*isValidatedBy newAuthCert*$\gg$ $=_u$ *Some*$_u$(&*keyStore*:*issuerKey*(*the*$_u$(&*keyStore*:*ownName*))$_a$)
)

5.3.2 Adding Authorisation Certificates to User Token

definition *AddAuthCertToUserToken* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
AddAuthCertToUserToken =
  ($\prod$ (*t*, *newAuthCert*) $\cdot$
  (*iuserToken*:*userTokenPresence* = $\ll$*present*$\gg$ $\wedge$
   $\ll$*goodT*(*t*)$\gg$ = *iuserToken*:*currentUserToken* $\wedge$
   $\ll$*t* $\in$ *ValidToken*$\gg$ $\wedge$
  @(*NewAuthCert t newAuthCert curTime*$[\![$*curTime*$\rightarrow$&*doorLatchAlarm*:*currentTime*$]\!]$)
) $\longrightarrow_r$ *iuserToken*:*currentUserToken* := $\ll$*goodT*(*t*$(\!|$*authCert* := *Some*(*newAuthCert*)$|\!)$)$\gg$)

6 Operations Interfacing to the ID Station (2)

6.1 Obtaining inputs from the real world

6.1.1 Polling the Real World

definition *PollTime* :: *SystemState hrel* **where**
[*upred-defs*]:
PollTime =
  (Δ[*idStation*:*doorLatchAlarm*,*DoorLatchAlarm*] $\wedge$
   $*idStation*:*doorLatchAlarm*:*currentTime*´ $=_u$ $*realWorld*:*monitored*:*now*)

definition *PollDoor* :: *SystemState hrel* **where**
[*upred-defs*]:
PollDoor =
  (Δ[*idStation*:*doorLatchAlarm*,*DoorLatchAlarm*] $\wedge$
   $*idStation*:*doorLatchAlarm*:*currentDoor*´ $=_u$ $*realWorld*:*monitored*:*door* $\wedge$
  $*idStation*:*doorLatchAlarm*:*latchTimeout*´ $=_u$ $*idStation*:*doorLatchAlarm*:*latchTimeout*
$\wedge$

$idStation:doorLatchAlarm:alarmTimeout´ =_u $idStation:doorLatchAlarm:alarmTimeout)

definition *PollUserToken* :: *SystemState hrel* **where**
[*upred-defs*]:
PollUserToken =
(Δ[*idStation:iuserToken*,*UserToken*] ∧
$idStation:iuserToken:userTokenPresence´ =_u «present» ⇔ $realWorld:monitored:userToken
≠_u «noT» ∧
$idStation:iuserToken:currentUserToken´ =_u
($realWorld:monitored:userToken ◁ $realWorld:monitored:userToken ≠_u «noT» ▷
$idStation:iuserToken:currentUserToken))

definition *PollAdminToken* :: *SystemState hrel* **where**
[*upred-defs*]:
PollAdminToken =
(Δ[*idStation:iadminToken*,*AdminToken*] ∧
$idStation:iadminToken:adminTokenPresence´ =_u «present» ⇔ $realWorld:monitored:adminToken
≠_u «noT» ∧
$idStation:iadminToken:currentAdminToken´ =_u
($realWorld:monitored:adminToken ◁ $realWorld:monitored:adminToken ≠_u
«noT» ▷ $idStation:iadminToken:currentAdminToken))

definition *PollFinger* :: *SystemState hrel* **where**
[*upred-defs*]:
PollFinger =
(Δ[*idStation:ifinger*,*Finger*] ∧
$idStation:ifinger:fingerPresence´ =_u «present» ⇔ $realWorld:monitored:finger
≠_u «noFP» ∧
$idStation:ifinger:currentFinger´ =_u
($realWorld:monitored:finger ◁ $realWorld:monitored:finger ≠_u «noFP» ▷
$idStation:ifinger:currentFinger))

definition *PollFloppy* :: *SystemState hrel* **where**
[*upred-defs*]:
PollFloppy =
(Δ[*idStation:ifloppy*,*Floppy*] ∧
$idStation:ifloppy:floppyPresence´ =_u «present» ⇔ $realWorld:monitored:floppy
≠_u «noFloppy» ∧
$idStation:ifloppy:currentFloppy´ =_u
($realWorld:monitored:floppy ◁ $realWorld:monitored:floppy ≠_u «noFloppy» ▷
$idStation:ifloppy:currentFloppy) ∧
$idStation:ifloppy:writtenFloppy´ =_u $idStation:ifloppy:writtenFloppy
)

definition *PollKeyboard* :: *SystemState hrel* **where**
[*upred-defs*]:
PollKeyboard =
(Δ[*idStation:ikeyboard*,*Keyboard*] ∧
$idStation:ikeyboard:keyedDataPresence´ =_u «present» ⇔ $realWorld:monitored:keyboard

$\neq_u \ll noKB \gg \wedge$
$\$idStation{:}ikeyboard{:}currentKeyedData' =_u$
$(\$realWorld{:}monitored{:}keyboard \triangleleft \$realWorld{:}monitored{:}keyboard \neq_u \ll noKB \gg \triangleright$
$\$idStation{:}ikeyboard{:}currentKeyedData))$

definition *TISPoll* :: *SystemState hrel* **where**
[*upred-defs*]:
TISPoll =
(— PollTime
idStation:*doorLatchAlarm*:*currentTime* := *realWorld*:*monitored*:*now* ;;
— PollDoor
idStation:*doorLatchAlarm*:*currentDoor* := *realWorld*:*monitored*:*door* ;;
— PollUserToken
idStation:*iuserToken*:*userTokenPresence* :=
(≪*absent*≫ ◁ (*realWorld*:*monitored*:*userToken* = ≪*noT*≫) ▷ ≪*absent*≫) ;;
idStation:*iuserToken*:*currentUserToken* :=
(*idStation*:*iuserToken*:*currentUserToken*
◁ (*realWorld*:*monitored*:*userToken* = ≪*noT*≫) ▷
realWorld:*monitored*:*userToken*) ;;
— PollAdminToken
idStation:*iadminToken*:*adminTokenPresence* :=
(≪*absent*≫ ◁ (*realWorld*:*monitored*:*adminToken* = ≪*noT*≫) ▷ ≪*absent*≫) ;;
idStation:*iadminToken*:*currentAdminToken* :=
(*idStation*:*iadminToken*:*currentAdminToken*
◁ (*realWorld*:*monitored*:*adminToken* = ≪*noT*≫) ▷
realWorld:*monitored*:*adminToken*) ;;
— PollFinger
idStation:*ifinger*:*fingerPresence* :=
(≪*absent*≫ ◁ (*realWorld*:*monitored*:*finger* = ≪*noFP*≫) ▷ ≪*absent*≫) ;;
idStation:*ifinger*:*currentFinger* :=
(*idStation*:*ifinger*:*currentFinger*
◁ (*realWorld*:*monitored*:*finger* = ≪*noFP*≫) ▷
realWorld:*monitored*:*finger*) ;;
— PollFloppy
idStation:*ifloppy*:*floppyPresence* :=
(≪*absent*≫ ◁ (*realWorld*:*monitored*:*floppy* = ≪*noFloppy*≫) ▷ ≪*absent*≫) ;;
idStation:*ifloppy*:*currentFloppy* :=
(*idStation*:*ifloppy*:*currentFloppy*
◁ (*realWorld*:*monitored*:*floppy* = ≪*noFloppy*≫) ▷
realWorld:*monitored*:*floppy*) ;;
— PollKeyboard
idStation:*ikeyboard*:*keyedDataPresence* :=
(≪*absent*≫ ◁ (*realWorld*:*monitored*:*keyboard* = ≪*noKB*≫) ▷ ≪*absent*≫) ;;
idStation:*ikeyboard*:*currentKeyedData* :=
(*idStation*:*ikeyboard*:*currentKeyedData*
◁ (*realWorld*:*monitored*:*keyboard* = ≪*noKB*≫) ▷
realWorld:*monitored*:*keyboard*)
)

6.2 The ID Station Changes the World

6.2.1 Periodic Updates

definition *UpdateLatch* :: *SystemState hrel* **where**
[*upred-defs*]:
UpdateLatch =
(Ξ[*idStation*:*doorLatchAlarm*,*DoorLatchAlarm*] ∧
RealWorldChanges ⊕$_r$ *realWorld* ∧
$*realWorld*:*controlled*:*latch*´ =$_u$ $*idStation*:*doorLatchAlarm*:*currentLatch*)

definition *UpdateAlarm* :: *SystemState hrel* **where**
[*upred-defs*]:
UpdateAlarm =
(Ξ[*idStation*:*doorLatchAlarm*,*DoorLatchAlarm*] ∧
RealWorldChanges ⊕$_r$ *realWorld* ∧
⌈*AuditLog*⌉$_<$ ⊕$_r$ *idStation*:*audit* ∧
$*realWorld*:*controlled*:*alarm*´ =$_u$ «*alarming*» ⇔ ($*idStation*:*doorLatchAlarm*:*doorAlarm* =$_u$ «*alarming*» ∨ $*idStation*:*audit*:*auditAlarm* =$_u$ «*alarming*»))

definition *UpdateDisplay* :: *SystemState hrel* **where**
[*upred-defs*]:
UpdateDisplay =
(Δ[*idStation*,*IDStation*] ∧
RealWorldChanges ⊕$_r$ *realWorld* ∧
$*realWorld*:*controlled*:*display*´ =$_u$ $*idStation*:*currentDisplay* ∧
$*idStation*:*currentDisplay*´ =$_u$ $*idStation*:*currentDisplay*)

definition *UpdateScreen* :: *SystemState hrel* **where**
[*upred-defs*]:
UpdateScreen =
(Δ[*idStation*,*IDStation*] ∧
Ξ[*idStation*:*admin*,*Admin*] ∧
RealWorldChanges ⊕$_r$ *realWorld* ∧
$*realWorld*:*controlled*:*screen*:*screenMsg*´ =$_u$ $*idStation*:*currentScreen*:*screenMsg* ∧
$*realWorld*:*controlled*:*screen*:*screenConfig*´ =$_u$
($*idStation*:*currentScreen*:*screenConfig*
◁ $*idStation*:*admin*:*rolePresent* =$_u$ «*Some*(*securityOfficer*)» ▷
«*clear*») ∧
$*realWorld*:*controlled*:*screen*:*screenStats*´ =$_u$
($*idStation*:*currentScreen*:*screenStats*
◁ $*idStation*:*admin*:*rolePresent* ≠$_u$ «*None*» ▷
«*clear*»))

definition *TISUpdate* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISUpdate =

$(realWorld{:}[RealWorldChanges]^+$;;
$realWorld{:}controlled{:}latch := idStation{:}doorLatchAlarm{:}currentLatch$;;
$realWorld{:}controlled{:}alarm := (\ll alarming \gg$
$\triangleleft\ (idStation{:}doorLatchAlarm{:}doorAlarm = \ll alarming \gg$
$\vee\ idStation{:}audit{:}auditAlarm = \ll alarming \gg)$
$\triangleright \ll silent \gg)$;;
$realWorld{:}controlled{:}display := idStation{:}currentDisplay)$

6.2.2 Updating the User Token

definition *UpdateUserToken* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
$UpdateUserToken = realWorld{:}monitored{:}userToken := idStation{:}iuserToken{:}currentUserToken$

7 The User Entry Operation (1)

definition *ResetScreenMessage* :: *IDStation hrel* **where**
[*upred-defs*]:
$ResetScreenMessage =$
$(\Delta[admin,Admin]$
$\wedge ((\$internal{:}status' \notin_u \{\ll quiescent \gg,\ll waitingRemoveTokenFail \gg\}_u \wedge \$currentScreen{:}screenMsg'$
$=_u \ll busy \gg) \vee$
$(\$internal{:}status' \in_u \{\ll quiescent \gg,\ll waitingRemoveTokenFail \gg\}_u \wedge$
$(\ \$internal{:}enclaveStatus' =_u \ll enclaveQuiescent \gg \wedge \$admin{:}rolePresent' =_u$
$\ll None \gg \wedge \$currentScreen{:}screenMsg' =_u \ll welcomeAdmin \gg$
$\vee\ \$internal{:}enclaveStatus' =_u \ll enclaveQuiescent \gg \wedge \$admin{:}rolePresent' \neq_u$
$\ll None \gg \wedge \$currentScreen{:}screenMsg' =_u \ll requestAdminOp \gg$
$\vee\ \$internal{:}enclaveStatus' =_u \ll waitingRemoveAdminTokenFail \gg \wedge \$currentScreen{:}screenMsg'$
$=_u \ll removeAdminToken \gg$
$\vee\ \$internal{:}enclaveStatus' \notin_u \{\ll enclaveQuiescent \gg,\ll waitingRemoveAdminTo-$
$kenFail \gg\}_u \wedge \$currentScreen{:}screenMsg' =_u \$currentScreen{:}screenMsg$
))))

lemma *mark-alpha-ResetScreenMessage* [*mark-alpha*]:
$\Sigma \lhd_\alpha ResetScreenMessage = \{\&admin,\&currentScreen,\&internal\} \lhd_\alpha ResetScreenMessage$
by (*rel-auto*)

definition *UserEntryContext* :: *SystemState hrel* **where**
[*upred-defs*]:
$UserEntryContext =$
$((RealWorldChanges \wedge \Xi[controlled,\ TISControlledRealWorld]) \oplus_r realWorld \wedge$
$(\Delta[iuserToken,UserToken] \wedge$
$\Delta[doorLatchAlarm,DoorLatchAlarm] \wedge$

$\Delta[audit,AuditLog] \wedge$
$\Xi[config,\ Config] \wedge$
$\Xi[iadminToken,\ AdminToken] \wedge$
$\Xi[keyStore,\ KeyStore] \wedge$
$\Xi[admin,\ Admin] \wedge$
$\Xi[ikeyboard,\ Keyboard] \wedge$
$\Xi[ifloppy,\ Floppy] \wedge$
$\Xi[ifinger,\ Finger] \wedge$
$\Delta[IDStation\text{-}inv] \wedge$
$ResetScreenMessage \wedge$
$(\$enclaveStatus' =_u \$enclaveStatus \wedge$
$(\$status \neq_u \ll waitingEntry\gg \Rightarrow \$tokenRemovalTimeout' =_u \$tokenRemovalTimeout)$
$) \oplus_r internal) \oplus_r idStation$
$)$

lemma *pre UserEntryContext = IDStation* $\oplus_p$ *idStation*
apply (*unfold UserEntryContext-def*)
apply (*simp*)
apply (*zcalcpre*)
oops

lemma *UserEntryContext-alt-def* [*upred-defs*]:
UserEntryContext =
$((RealWorldChanges \wedge \Xi[controlled,\ TISControlledRealWorld]) \oplus_r realWorld \wedge$
$(\Delta[IDStation] \wedge$
$\$config' =_u \$config \wedge$
$\$iadminToken' =_u \$iadminToken \wedge$
$\$keyStore' =_u \$keyStore \wedge$
$\$admin' =_u \$admin \wedge$
$\$ikeyboard' =_u \$ikeyboard \wedge$
$\$ifloppy' =_u \$ifloppy \wedge$
$\$ifinger' =_u \$ifinger \wedge$
$ResetScreenMessage \wedge$
$(\$enclaveStatus' =_u \$enclaveStatus \wedge$
$\$status \neq_u \ll waitingEntry\gg \Rightarrow \$tokenRemovalTimeout' =_u \$tokenRemovalTimeout$
$) \oplus_r internal) \oplus_r idStation$
$)$
oops

lemma *pre*$((RealWorldChanges \wedge \Xi[controlled,\ TISControlledRealWorld]) \oplus_r$ *realWorld*$) = true$
by (*rel-auto*)

7.1 User Token Tears

definition *UserTokenTorn* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
UserTokenTorn =
$((internal{:}status \in \{\ll gotUserToken\gg, \ll waitingUpdateToken\gg, \ll waitingFinger\gg,$

$\ll gotFinger\gg$, $\ll waitingEntry\gg\}$
  $\wedge$ *iuserToken*:*userTokenPresence* = $\ll absent\gg$
) $\longrightarrow_r$ *currentDisplay* := $\ll welcom\gg$;; *internal*:*status* := $\ll quiescent\gg$)

lemma $\{\!|IDStation\text{-}inv|\!\}\, UserTokenTorn \{\!|IDStation\text{-}inv|\!\}_u$
 by (*simp add*: *UserTokenTorn-def*, *hoare-auto*)

8 Operations within the Enclave (1)

definition *EnclaveContext* :: *SystemState hrel* **where**
[*upred-defs*]:
EnclaveContext =
 (Δ[*idStation*, *IDStation*] $\wedge$
 RealWorldChanges $\oplus_r$ *realWorld* $\wedge$
 Ξ[*realWorld*:*controlled*, *TISControlledRealWorld*] $\wedge$
 Ξ[*idStation*:*iuserToken*, *UserToken*] $\wedge$
 Ξ[*idStation*:*iadminToken*, *AdminToken*] $\wedge$
 Ξ[*idStation*:*ifinger*, *Finger*] $\wedge$
 Ξ[*idStation*:*stats*, *Stats*] $\wedge$
 (\$*tokenRemovalTimeout*´ $=_u$ \$*tokenRemovalTimeout*) $\oplus_r$ *idStation*:*internal*
)

definition *EnrolContext* :: *SystemState hrel* **where**
EnrolContext = (*EnclaveContext* $\wedge$
 Ξ[*idStation*:*ikeyboard*, *Keyboard*] $\wedge$
 Ξ[*idStation*:*admin*, *Admin*] $\wedge$
 Ξ[*idStation*:*doorLatchAlarm*, *DoorLatchAlarm*] $\wedge$
 Ξ[*idStation*:*config*, *Config*] $\wedge$
 Ξ[*idStation*:*ifloppy*, *Floppy*])

We depart from the Z specification for this operation, as to precisely implement the Z behaviour we need a state space containing both a *ValidEnrol* and a *KeyStore*. Since the former is static rather than dynamic, it seems to make sense to treat it as a parameter here.

FIX: We had to change ownName (as it was in Tokeneer Z) to ownName' in the function addition.

8.1 Updating the Key Store

definition *UpdateKeyStore* :: *Enrol* $\Rightarrow$ *KeyStore hrel* **where**
[*upred-defs*]:
UpdateKeyStore e =
 (Δ[*KeyStore*] $\wedge$
  $\ll e \in ValidEnrol\gg \wedge$
  \$*ownName*´ $=_u$ $\ll Some\ (subject\ (idStationCert\ e))\gg \wedge$
  \$*issuerKey*´ $=_u$ \$*issuerKey* $\oplus$ $\ll\{(subject\ c, subjectPubK\ c) \mid c.\ c \in issuerCerts\ e\}\gg \oplus \{(the_u(\$ownName´), \ll subjectPubK\ (idStationCert\ e)\gg)_u\}_u$
)

lemma *rel-typed-Collect* [*rclos*]: ⟦ ⋀ *x y*. *P* (*x*, *y*) ⟹ *x* ∈ *A* ∧ *y* ∈ *B* ⟧ ⟹ *Collect* *P* ∈ *A* ↔$_r$ *B*
by (*auto simp add*: *rel-typed-def*)

lemma *rel-pfun-Collect* [*rclos*]: ⟦ ⋀ *x y*. *P* (*x*, *y*) ⟹ *x* ∈ *A* ∧ *y* ∈ *B*; ⋀ *x y z*. ⟦ *P* (*x*, *y*); *P* (*x*, *z*) ⟧ ⟹ *y* = *z* ⟧ ⟹ *Collect* *P* ∈ *A* ⇀$_r$ *B*
by (*auto simp add*: *rel-pfun-def rel-typed-def functional-algebraic*)

lemma *UpdateKeyStore-prog-def*:
UpdateKeyStore e =
?[@*KeyStore* ∧ «*e* ∈ *ValidEnrol*»] ;;
ownName := «*Some* (*subject* (*idStationCert e*))» ;;
issuerKey := *issuerKey* ⊕ «{(*subject c*, *subjectPubK c*) | *c*. *c* ∈ *issuerCerts* *e*}» ⊕ {(*the*(*ownName*), «*subjectPubK* (*idStationCert e*)»)}
(**is** *?P* = *?Q*)
proof (*rule antisym*)
show *?P* ⊑ *?Q*
by (*rel-auto*, *auto intro*: *rclos intro*!: *rel-pfun-override rel-pfun-Collect*)
show *?Q* ⊑ *?P*
by (*rel-auto*)
qed

lemma *pre-KeyStore*:
e ∈ *ValidEnrol* ⟹ *Dom*(*UpdateKeyStore e*) = *KeyStore*
apply (*rel-auto*)
apply (*auto intro*: *rclos intro*!: *rel-pfun-override*)
done

definition *UpdateKeyStoreFromFloppy* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
UpdateKeyStoreFromFloppy =
(Δ[*keyStore*,*KeyStore*] ∧
⌈*Floppy* ⊕$_p$ *ifloppy*⌉$_<$ ∧
$*ifloppy*:*currentFloppy* ∈$_u$ «*range*(*enrolmentFile*)» ∧
(∃ *e* · «*e*» =$_u$ «*enrolmentFile-of*»($*ifloppy*:*currentFloppy*)$_a$
∧ *UpdateKeyStore e* ⊕$_r$ *keyStore*))

9 The User Entry Operation (2)

9.1 Reading the User Token

definition *ReadUserToken* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ReadUserToken =
((*internal*:*enclaveStatus* ∈ {«*enclaveQuiescent*», «*waitingRemoveAdminToken-Fail*»}
∧ *internal*:*status* = «*quiescent*»
∧ *iuserToken*:*userTokenPresence* = «*present*»

$) \longrightarrow_r$ *currentDisplay* := $\ll$*wait*$\gg$;; *internal*:*status* := $\ll$*gotUserToken*$\gg$)

9.2 Validating the User Token

definition *UEC* :: *IDStation hrel* $\Rightarrow$ *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
UEC(*Op*) =
($\bigsqcap$ *t* · *idStation*:[*Op*]$^+$;;
realWorld:[
monitored:*now* := *monitored*:*now* + $\ll$*t*$\gg$;;
monitored:*door* := $*$;; *monitored*:*finger* := $*$;;
monitored:*userToken* := $*$;; *monitored*:*adminToken* := $*$;;
monitored:*floppy* := $*$;; *monitored*:*keyboard* := $*$]$^+$)

lemma *UEC-refines-RealWorldChanges*:
(*RealWorldChanges* $\oplus_r$ *realWorld*) $\sqsubseteq$ *UEC*(*Op*)
by (*rel-auto*)

lemma *ReadUserToken-correct*: $\{\!|$*IDStation*$|\!\}$ *ReadUserToken* $\{\!|$*IDStation*$|\!\}_u$
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-wlp-auto*)
apply (*simp add*: *tis-defs*, *hoare-wlp-auto*)
done

definition [*upred-defs*, *tis-defs*]: *TISReadUserToken* = *UEC*(*ReadUserToken*)
lemma ‘*UserTokenOK* $\Rightarrow$ ($\exists$ *e*$\in\ll$*ValidToken*$\gg$ · $\ll$*goodT*(*e*)$\gg$ $=_u$ &*iuserToken*:*currentUserToken*)‘
by (*rel-auto*)

lemma ‘*UserTokenWithOKAuthCert* $\Rightarrow$ ($\exists$ *e*$\in\ll$*TokenWithValidAuth*$\gg$ · $\ll$*goodT*(*e*)$\gg$ $=_u$
&*iuserToken*:*currentUserToken*)‘
by (*rel-auto*)

definition *BioCheckNotRequired* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
BioCheckNotRequired =
((*internal*:*status* = $\ll$*gotUserToken*$\gg$
$\wedge$ *iuserToken*:*userTokenPresence* = $\ll$*present*$\gg$
$\wedge$ @*UserTokenWithOKAuthCert*
) $\longrightarrow_r$ *internal*:*status* := $\ll$*waitingEntry*$\gg$;; *currentDisplay* := $\ll$*wait*$\gg$)

lemma *BioCheckNotRequired-correct*: $\{\!|$*IDStation*$|\!\}$ *BioCheckNotRequired* $\{\!|$*IDStation*$|\!\}_u$
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-wlp-auto*)
apply (*simp add*: *tis-defs*, *hoare-wlp-auto*)
done

definition *BioCheckRequired* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
BioCheckRequired =

((*internal*:*status* = ≪*gotUserToken*≫
∧ *iuserToken*:*userTokenPresence* = ≪*present*≫
∧ (¬ @*UserTokenWithOKAuthCert*) ∧ @*UserTokenOK*
) ⟶$_r$ *internal*:*status* := ≪*waitingFinger*≫ ;; *currentDisplay* := ≪*insertFinger*≫)

lemma *BioCheckRequired-correct*: ⦃*IDStation-inv*⦄*BioCheckRequired*⦃*IDStation-inv*⦄$_u$
by (*simp add*: *BioCheckRequired-def*, *hoare-auto*)

definition [*upred-defs*, *tis-defs*]: *ValidateUserTokenOK* = (*BioCheckRequired* ∨ *BioCheckNotRequired*)

definition *ValidateUserTokenFail* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ValidateUserTokenFail =
((*internal*:*status* = ≪*gotUserToken*≫
∧ *iuserToken*:*userTokenPresence* = ≪*present*≫
∧ (¬ @*UserTokenWithOKAuthCert*) ∧ (¬ @*UserTokenOK*)
) ⟶$_r$ *internal*:*status* := ≪*waitingRemoveTokenFail*≫ ;; *currentDisplay* := ≪*removeToken*≫)

lemma *ValidateUserTokeFail-correct*: ⦃*IDStation-inv*⦄*ValidateUserTokenFail*⦃*IDStation-inv*⦄$_u$
by (*simp add*: *ValidateUserTokenFail-def*, *hoare-auto*)

definition [*upred-defs*, *tis-defs*]:
TISValidateUserToken = (*UEC*(*ValidateUserTokenOK*) ∨ *UEC*(*ValidateUserTokenFail*)
∨ *UEC*(*UserTokenTorn* ;; *?*[*internal*:*status* = ≪*gotUserToken*≫]))

9.3 Reading a Fingerprint

definition *ReadFingerOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ReadFingerOK =
((*internal*:*status* = ≪*waitingFinger*≫
∧ *ifinger*:*fingerPresence* = ≪*present*≫
∧ *iuserToken*:*userTokenPresence* = ≪*present*≫
) ⟶$_r$ *internal*:*status* := ≪*gotFinger*≫ ;; *currentDisplay* := ≪*wait*≫)

definition *NoFinger* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
NoFinger =
?[*internal*:*status* = ≪*waitingFinger*≫
∧ *ifinger*:*fingerPresence* = ≪*absent*≫
∧ *iuserToken*:*userTokenPresence* = ≪*present*≫
]

definition *FingerTimeout* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
FingerTimeout =

$((internal{:}status = \ll waitingFinger\gg$
$\quad \wedge\ ifinger{:}fingerPresence = \ll absent\gg$
$\quad \wedge\ iuserToken{:}userTokenPresence = \ll present\gg$
$)\longrightarrow_r currentDisplay := \ll removeToken\gg\ ;;\ internal{:}status := \ll waitingRemove\text{-}TokenFail\gg)$

definition [*upred-defs*, *tis-defs*]:
$TISReadFinger = (UEC(ReadFingerOK) \vee UEC(FingerTimeout) \vee UEC(NoFinger)$
$\qquad \vee\ UEC(UserTokenTorn\ ;;\ ?[internal{:}status = \ll waitingFinger\gg]))$

9.4 Validating a Fingerprint

definition *ValidateFingerOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$ValidateFingerOK =$
$((internal{:}status = \ll gotFinger\gg$
$\quad \wedge\ iuserToken{:}userTokenPresence = \ll present\gg$
$\quad \wedge\ @FingerOK$
$)\longrightarrow_r currentDisplay := \ll wait\gg\ ;;\ internal{:}status := \ll waitingUpdateToken\gg)$

definition *ValidateFingerFail* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$ValidateFingerFail =$
$((internal{:}status = \ll gotFinger\gg$
$\quad \wedge\ iuserToken{:}userTokenPresence = \ll present\gg$
$\quad \wedge\ @FingerOK$
$)\longrightarrow_r currentDisplay := \ll removeToken\gg\ ;;\ internal{:}status := \ll waitingRemove\text{-}TokenFail\gg)$

definition [*upred-defs*, *tis-defs*]:
$TISValidateFinger = (UEC(ValidateFingerOK) \vee UEC(ValidateFingerFail)$
$\qquad \vee\ UEC(UserTokenTorn\ ;;\ ?[internal{:}status = \ll gotFinger\gg]))$

9.5 Writing the User Token

definition *WriteUserTokenOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$WriteUserTokenOK =$
$((internal{:}status = \ll waitingUpdateToken\gg$
$\quad \wedge\ iuserToken{:}userTokenPresence = \ll present\gg$
$)\longrightarrow_r AddAuthCertToUserToken\ ;;$
$\qquad currentDisplay := \ll wait\gg\ ;;$
$\qquad internal{:}status := \ll waitingEntry\gg)$

definition *WriteUserTokenFail* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$WriteUserTokenFail =$
$((internal{:}status = \ll waitingUpdateToken\gg$
$\quad \wedge\ iuserToken{:}userTokenPresence = \ll present\gg$
$)\longrightarrow_r AddAuthCertToUserToken\ ;;$

$currentDisplay := \ll tokenUpdateFailed \gg$;;
$internal{:}status := \ll waitingEntry \gg$)

definition [*upred-defs*, *tis-defs*]:
$WriteUserToken = (WriteUserTokenOK \vee WriteUserTokenFail)$

definition [*upred-defs*, *tis-defs*]:
$TISWriteUserToken =$
$((UEC(WriteUserToken)\ ;;\ UpdateUserToken)$
$\vee\ UEC(UserTokenTorn\ ;;\ ?[internal{:}status = \ll waitingUpdateToken \gg]))$

term $(config{:}entryPeriod[\ll role\ (privCert\ t) \gg][\ll class\ (clearance\ (privCert\ t)) \gg])_e$

9.6 Validating Entry

definition *UserAllowedEntry* :: *IDStation upred* **where**
[*upred-defs*]:
UserAllowedEntry =
$(((\exists\ t \in \ll ValidToken \gg \cdot$
$\ll goodT(t) \gg = iuserToken{:}currentUserToken$
$\wedge\ doorLatchAlarm{:}currentTime \in config{:}entryPeriod[\ll role\ (privCert\ t) \gg][\ll class\ (clearance\ (privCert\ t)) \gg]))$
$\vee\ (\exists\ t \in \ll TokenWithValidAuth \gg \cdot$
$\ll goodT(t) \gg = iuserToken{:}currentUserToken$
$\wedge\ doorLatchAlarm{:}currentTime \in config{:}entryPeriod[\ll role\ (the\ (authCert\ t)) \gg][\ll class\ (clearance\ (the\ (authCert\ t))) \gg])_e)$

definition *EntryOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
EntryOK =
$((internal{:}status = \ll waitingEntry \gg \wedge$
$iuserToken{:}userTokenPresence = \ll present \gg \wedge$
$@UserAllowedEntry)$
$\longrightarrow_r currentDisplay := \ll openDoor \gg$;;
$internal{:}status := \ll waitingRemoveTokenSuccess \gg$;;
$internal{:}tokenRemovalTimeout := doorLatchAlarm{:}currentTime + config{:}tokenRemovalDuration)$

definition *EntryNotAllowed* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
EntryNotAllowed =
$((internal{:}status = \ll waitingEntry \gg \wedge$
$iuserToken{:}userTokenPresence = \ll present \gg \wedge$
$(\neg\ @UserAllowedEntry))$
$\longrightarrow_r currentDisplay := \ll removeToken \gg$;;
$internal{:}status := \ll waitingRemoveTokenFail \gg)$

definition [*upred-defs*, *tis-defs*]:
TISValidateEntry =

$(UEC(EntryOK) \vee UEC(EntryNotAllowed) \vee UEC(UserTokenTorn ;; ?[internal{:}status = \ll waitingEntry \gg]))$

9.7 Unlocking the Door

definition *UnlockDoorOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$UnlockDoorOK =$
$(internal{:}status = \ll waitingRemoveTokenSuccess \gg \wedge$
$iuserToken{:}userTokenPresence = \ll absent \gg)$
$\longrightarrow_r UnlockDoor ;; currentDisplay := \ll doorUnlocked \gg ;; internal{:}status := \ll quiescent \gg$

lemma *wp-UnlockDoorOK*:
$UnlockDoorOK\ wp\ (doorLatchAlarm{:}currentLatch = \ll unlocked \gg) =$
$(internal{:}status = \ll waitingRemoveTokenSuccess \gg \wedge iuserToken{:}userTokenPresence = \ll absent \gg)_e$
by (*simp add*: *tis-defs wp usubst unrest*)

definition *WaitingTokenRemoval* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$WaitingTokenRemoval =$
$?[internal{:}status \in \{\ll waitingRemoveTokenSuccess \gg, \ll waitingRemoveTokenFail \gg\} \wedge$
$internal{:}status = \ll waitingRemoveTokenSuccess \gg \Rightarrow doorLatchAlarm{:}currentTime < internal{:}tokenRemovalTimeout \wedge$
$iuserToken{:}userTokenPresence = \ll present \gg]$

definition *TokenRemovalTimeout* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$TokenRemovalTimeout =$
$((internal{:}status = \ll waitingRemoveTokenSuccess \gg \wedge$
$doorLatchAlarm{:}currentTime \geq internal{:}tokenRemovalTimeout \wedge$
$iuserToken{:}userTokenPresence = \ll present \gg) \longrightarrow_r$
$internal{:}status := \ll waitingRemoveTokenFail \gg ;;$
$currentDisplay := \ll removeToken \gg)$

definition [*upred-defs*, *tis-defs*]:
$TISUnlockDoor = (UEC(UnlockDoorOK)$
$\vee UEC(WaitingTokenRemoval ;; ?[internal{:}status = \ll waitingRemoveTokenSuccess \gg])$
$\vee UEC(TokenRemovalTimeout))$

9.8 Terminating a Failed Access

definition *FailedAccessTokenRemoved* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$FailedAccessTokenRemoved =$
$((internal{:}status = \ll waitingRemoveTokenFail \gg \wedge$
$iuserToken{:}userTokenPresence = \ll absent \gg) \longrightarrow_r$

internal:*status* := ≪*quiescent*≫ ;;
currentDisplay := ≪*welcom*≫)

definition [*upred-defs*, *tis-defs*]:
TISCompleteFailedAccess = (*UEC*(*FailedAccessTokenRemoved*)
∨ *UEC*(*WaitingTokenRemoval* ;; ?[*internal*:*status* = ≪*waitingRemoveTokenFail*≫]))

9.9 The Complete User Entry

definition [*upred-defs*, *tis-defs*]:
TISUserEntryOp = (*TISReadUserToken* ∨ *TISValidateUserToken* ∨ *TISReadFinger* ∨ *TISValidateFinger*
∨ *TISWriteUserToken* ∨ *TISValidateEntry* ∨ *TISUnlockDoor* ∨ *TISCompleteFailedAccess*)

10 Operations Within the Enclave (2)

10.1 Enrolment of an ID Station

10.1.1 Requesting Enrolment

definition *RequestEnrolment* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
RequestEnrolment = (*EnrolContext* ∧
Ξ[*idStation*:*keyStore*, *KeyStore*] ∧
Ξ[*idStation*:*audit*, *AuditLog*] ∧
Ξ[*idStation*:*internal*, *Internal*] ∧
($*enclaveStatus* =$_u$ ≪*notEnrolled*≫) ⊕$_r$ *idStation*:*internal* ∧
($*floppyPresence* =$_u$ ≪*absent*≫) ⊕$_r$ *idStation*:*ifloppy* ∧
($*currentScreen*:*screenMsg*´ =$_u$ ≪*insertEnrolmentData*≫ ∧
$*currentDisplay*´ =$_u$ ≪*blank*≫) ⊕$_r$ *idStation*
)

definition *ReadEnrolmentFloppy* :: *SystemState hrel* **where**
ReadEnrolmentFloppy = (*EnrolContext* ∧
Ξ[*idStation*:*keyStore*, *KeyStore*] ∧
($*enclaveStatus* =$_u$ ≪*notEnrolled*≫) ⊕$_r$ *idStation*:*internal* ∧
($*floppyPresence* =$_u$ ≪*present*≫) ⊕$_r$ *idStation*:*ifloppy* ∧
($*currentScreen*:*screenMsg*´ =$_u$ ≪*validatingEnrolmentData*≫ ∧
$*internal*:*status*´ =$_u$ $*internal*:*status* ∧
$*currentDisplay*´ =$_u$ ≪*blank*≫) ⊕$_r$ *idStation*
)

definition *ReadEnrolmentData* = (*ReadEnrolmentFloppy* ∨ *RequestEnrolment*)

10.1.2 Validating Enrolment data from Floppy

definition *EnrolmentDataOK* :: *IDStation upred* **where**
EnrolmentDataOK = (*Floppy* ⊕$_p$ *ifloppy* ∧

$KeyStore \oplus_p keyStore \wedge$
$(ifloppy{:}currentFloppy \in \ll range\ enrolmentFile \gg \wedge$
$\ll enrolmentFile\text{-}of \gg [ifloppy{:}currentFloppy] \in \ll ValidEnrol \gg)_e)$

definition *ValidateEnrolmentDataOK* :: *SystemState hrel* **where**
ValidateEnrolmentDataOK =
$(EnrolContext \wedge$
$(UpdateKeyStoreFromFloppy \wedge$
$AddElementsToLog \wedge$
$\$internal{:}enclaveStatus =_u \ll waitingEnrol \gg \wedge$
$\lceil EnrolmentDataOK \rceil_< \wedge$
$\$currentScreen{:}screenMsg' =_u \ll welcomeAdmin \gg \wedge$
$\$internal{:}enclaveStatus' =_u \ll enclaveQuiescent \gg \wedge$
$\$internal{:}status' =_u \ll quiescent \gg \wedge$
$\$currentDisplay' =_u \ll welcom \gg$
$) \oplus_r idStation)$

definition *ValidateEnrolmentDataFail* :: *SystemState hrel* **where**
ValidateEnrolmentDataFail =
$(EnrolContext \wedge$
$(\Xi[keyStore, KeyStore] \wedge$
$AddElementsToLog \wedge$
$\$internal{:}enclaveStatus =_u \ll waitingEnrol \gg \wedge$
$\lceil \neg\ EnrolmentDataOK \rceil_< \wedge$
$\$currentScreen{:}screenMsg' =_u \ll enrolmentFailed \gg \wedge$
$\$internal{:}enclaveStatus' =_u \ll waitingEndEnrol \gg \wedge$
$\$internal{:}status' =_u \$internal{:}status \wedge$
$\$currentDisplay' =_u \ll blank \gg$
$) \oplus_r idStation)$

definition *ValidateEnrolmentData* = $(ValidateEnrolmentDataOK \vee ValidateEnrolmentDataFail)$

10.1.3 Completing a Failed Enrolment

definition *FailedEnrolFloppyRemoved* :: *SystemState hrel* **where**
FailedEnrolFloppyRemoved =
$(EnrolContext \wedge$
$(\Xi[keyStore, KeyStore] \wedge$
$\$internal{:}enclaveStatus =_u \ll waitingEndEnrol \gg \wedge$
$\$ifloppy{:}floppyPresence =_u \ll absent \gg \wedge$
$\$currentScreen{:}screenMsg' =_u \ll insertEnrolmentData \gg \wedge$
$\$internal{:}enclaveStatus' =_u \ll notEnrolled \gg \wedge$
$\$internal{:}status' =_u \$internal{:}status \wedge$
$\$currentDisplay' =_u \ll blank \gg$
$) \oplus_r idStation)$

definition *WaitingFloppyRemoval* :: *SystemState hrel* **where**
WaitingFloppyRemoval =

(*EnrolContext* ∧
Ξ[*idStation*,*IDStation*] ∧
($*internal*:*enclaveStatus* =$_u$ ≪*waitingEndEnrol*≫ ∧
$*ifloppy*:*floppyPresence* =$_u$ ≪*present*≫
) ⊕$_r$ *idStation*)

definition *CompleteFailedEnrolment* = (*FailedEnrolFloppyRemoved* ∨ *WaitingFloppyRemoval*)

10.1.4 The Complete Enrolment

definition *TISEnrolOp* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISEnrolOp = *false*

10.2 Further Administrator Operations

definition *AdminLogon* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
AdminLogon =
((*admin*:*rolePresent* = ≪*None*≫ ∧
(∃ *t*∈≪*ValidToken*≫ • (≪*goodT*(*t*)≫ = *iadminToken*:*currentAdminToken*))
) ⟶$_r$ *admin*:*rolePresent* := *Some*(*role*(*the*(*authCert*(*ofGoodT*(*iadminToken*:*currentAdminToken*))))))
;;
admin:*currentAdminOp* := ≪*None*≫ ;;
— The assignments below were added to ensure the invariant *Admin* is satisfied
if admin:*rolePresent* = ≪*Some*(*guard*)≫
then admin:*availableOps* := {≪*overrideLock*≫}
else if admin:*rolePresent* = ≪*Some*(*auditManager*)≫
then admin:*availableOps* := {≪*archiveLog*≫}
else
admin:*availableOps* := {≪*updateConfigData*≫,≪*shutdownOp*≫}
fi fi)

definition *AdminLogout* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
AdminLogout =
((*admin*:*rolePresent* ≠ ≪*None*≫
) ⟶$_r$ *admin*:*rolePresent* := ≪*None*≫ ;; *admin*:*currentAdminOp* := ≪*None*≫)

definition *AdminStartOp* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
AdminStartOp =
((*admin*:*rolePresent* ≠ ≪*None*≫
∧ *admin*:*currentAdminOp* = ≪*None*≫
∧ *ikeyboard*:*currentKeyedData* ∈ ≪*keyedOps*≫(|*admin*:*availableOps*|)
) ⟶$_r$ *admin*:*currentAdminOp* := *Some*(*ofKeyedOps*(*ikeyboard*:*currentKeyedData*)))

definition *AdminFinishOp* :: *IDStation hrel* **where**

[*upred-defs*, *tis-defs*]:
AdminFinishOp =
((*admin*:*rolePresent* ≠ «*None*»
∧ *admin*:*currentAdminOp* ≠ «*None*»
) ⟶$_r$ *admin*:*currentAdminOp* := «*None*»)

definition *AdminTokenTear* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
AdminTokenTear =
((*iadminToken*:*adminTokenPresence* = «*absent*»
) ⟶$_r$ *internal*:*enclaveStatus* := «*enclaveQuiescent*»)

definition *BadAdminTokenTear* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
BadAdminTokenTear =
((*internal*:*enclaveStatus* ∈ {«*gotAdminToken*», «*waitingStartAdminOp*», «*waitingFinishAdminOp*»})
⟶$_r$ *AdminTokenTear*)

definition *BadAdminLogout* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
BadAdminLogout =
((*internal*:*enclaveStatus* ∈ {«*waitingStartAdminOp*», «*waitingFinishAdminOp*»}
) ⟶$_r$ (*BadAdminTokenTear* ;; *AdminLogout*))

definition *LoginAborted* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
LoginAborted = ((*internal*:*enclaveStatus* = «*gotAdminToken*») ⟶$_r$ *BadAdminTokenTear*)

definition *ReadAdminToken* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ReadAdminToken =
((*internal*:*enclaveStatus* = «*enclaveQuiescent*»
∧ *internal*:*status* ∈ {«*quiescent*», «*waitingRemoveTokenFail*»}
∧ *admin*:*rolePresent* = «*None*»
∧ *iadminToken*:*adminTokenPresence* = «*present*»
) ⟶$_r$ *internal*:*enclaveStatus* := «*gotAdminToken*»)

definition *TISReadAdminToken* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]: *TISReadAdminToken* = *UEC*(*ReadAdminToken*)

definition *ValidateAdminTokenOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ValidateAdminTokenOK =
((*internal*:*enclaveStatus* = «*gotAdminToken*»
∧ *iadminToken*:*adminTokenPresence* = «*present*»
∧ @*AdminTokenOK*
) ⟶$_r$ *AdminLogon* ;;

$currentScreen{:}screenMsg := \ll requestAdminOp \gg$;;
$internal{:}enclaveStatus := \ll enclaveQuiescent \gg$)

lemma *hoare-post-conj-split*: $\{\!|b|\!\} P \{\!|c \wedge d|\!\}_u \longleftrightarrow (\{\!|b|\!\} P \{\!|c|\!\}_u \wedge \{\!|b|\!\} P \{\!|d|\!\}_u)$
by (*rel-auto*)

lemma *ValidateAdminTokenOK-correct*:
$\{\!|IDStation|\!\}\, ValidateAdminTokenOK \{\!|IDStation|\!\}_u$
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
done

definition *ValidateAdminTokenFail* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ValidateAdminTokenFail =
(($internal{:}enclaveStatus = \ll gotAdminToken \gg$
$\wedge$ $iadminToken{:}adminTokenPresence = \ll present \gg$
$\wedge$ ($\neg$ @*AdminTokenOK*)
) $\longrightarrow_r$ $currentScreen{:}screenMsg := \ll removeAdminToken \gg$;;
$internal{:}enclaveStatus := \ll waitingRemoveAdminTokenFail \gg$)

lemma *ValidateAdminTokenFail-correct*:
$\{\!|IDStation|\!\}\, ValidateAdminTokenFail \{\!|IDStation|\!\}_u$
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*simp add*: *IDStation-inv-def*)
apply (*auto simp add*: *hoare-post-conj-split*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
done

definition *TISValidateAdminToken* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISValidateAdminToken =
(*UEC*(*ValidateAdminTokenOK*) $\vee$ *UEC*(*ValidateAdminTokenFail*) $\vee$ *UEC*(*LoginAborted*))

definition *FailedAdminTokenRemove* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:

$FailedAdminTokenRemove =$
$((internal{:}enclaveStatus = \ll waitingRemoveAdminTokenFail\gg$
$\wedge\ iadminToken{:}adminTokenPresence = \ll absent\gg$
$) \longrightarrow_r currentScreen{:}screenMsg := \ll welcomeAdmin\gg\ ;;$
$internal{:}enclaveStatus := \ll enclaveQuiescent\gg)$

definition $WaitingAdminTokenRemoval :: IDStation\ hrel$ **where**
$[upred\text{-}defs, tis\text{-}defs]:$
$WaitingAdminTokenRemoval =$
$((internal{:}enclaveStatus = \ll waitingRemoveAdminTokenFail\gg$
$\wedge\ iadminToken{:}adminTokenPresence = \ll present\gg) \longrightarrow_r II)$

definition $TISCompleteFailedAdminLogon :: SystemState\ hrel$ **where**
$[upred\text{-}defs, tis\text{-}defs]:$
$TISCompleteFailedAdminLogon = (UEC(FailedAdminTokenRemove) \vee UEC(WaitingAdminTokenRemoval))$

definition $[upred\text{-}defs, tis\text{-}defs]:$
$TISAdminLogon = (TISReadAdminToken \vee TISValidateAdminToken \vee TISCompleteFailedAdminLogon)$

definition $StartOpContext :: IDStation\ hrel$ **where**
$[upred\text{-}defs, tis\text{-}defs]:$
$StartOpContext =$
$((internal{:}enclaveStatus = \ll enclaveQuiescent\gg$
$\wedge\ iadminToken{:}adminTokenPresence = \ll present\gg$
$\wedge\ admin{:}rolePresent \neq \ll None\gg$
$\wedge\ internal{:}status \in \{\ll quiescent\gg, \ll waitingRemoveTokenFail\gg\}) \longrightarrow_r II)$

definition $ValidateOpRequestOK :: IDStation\ hrel$ **where**
$[upred\text{-}defs, tis\text{-}defs]:$
$ValidateOpRequestOK =$
$((ikeyboard{:}keyedDataPresence = \ll present\gg \wedge$
$ikeyboard{:}currentKeyedData \in \ll keyedOps\gg(\!|admin{:}availableOps|\!))$
$\longrightarrow_r StartOpContext\ ;;$
$AdminStartOp\ ;;$
$currentScreen{:}screenMsg := \ll doingOp\gg\ ;;$
$internal{:}enclaveStatus := \ll waitingStartAdminOp\gg)$

definition $ValidateOpRequestFail :: IDStation\ hrel$ **where**
$[upred\text{-}defs, tis\text{-}defs]:$
$ValidateOpRequestFail =$
$((ikeyboard{:}keyedDataPresence = \ll present\gg \wedge$
$ikeyboard{:}currentKeyedData \notin \ll keyedOps\gg(\!|admin{:}availableOps|\!))$
$\longrightarrow_r StartOpContext\ ;;$
$currentScreen{:}screenMsg := \ll invalidRequest\gg)$

definition $NoOpRequest :: IDStation\ hrel$ **where**
$[upred\text{-}defs, tis\text{-}defs]:$
$NoOpRequest =$

$((ikeyboard{:}keyedDataPresence = \ll absent \gg) \longrightarrow_r StartOpContext)$

definition [*upred-defs*, *tis-defs*]:
$ValidateOpRequest = (ValidateOpRequestOK \vee ValidateOpRequestFail \vee NoOpRequest)$

definition [*upred-defs*, *tis-defs*]: $TISStartAdminOp = UEC(ValidateOpRequest)$

definition *AdminOpStartedContext* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$AdminOpStartedContext =$
$((internal{:}enclaveStatus = \ll waitingStartAdminOp \gg$
$\wedge\ iadminToken{:}adminTokenPresence = \ll present \gg$
$) \longrightarrow_r II)$

definition *ShutdownOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$ShutdownOK =$
$((internal{:}enclaveStatus = \ll waitingStartAdminOp \gg$
$\wedge\ admin{:}currentAdminOp = \ll Some(shutdownOp) \gg$
$\wedge\ doorLatchAlarm{:}currentDoor = \ll closed \gg$
$) \longrightarrow_r LockDoor\ ;;$
$AdminLogout\ ;;$
$currentScreen{:}screenMsg := \ll clear \gg\ ;;$
$internal{:}enclaveStatus := \ll shutdown \gg\ ;;$
$currentDisplay := \ll blank \gg$
$)$

definition *ShutdownWaitingDoor* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$ShutdownWaitingDoor =$
$((internal{:}enclaveStatus = \ll waitingStartAdminOp \gg$
$\wedge\ admin{:}currentAdminOp = \ll Some(shutdownOp) \gg$
$\wedge\ doorLatchAlarm{:}currentDoor = \ll dopen \gg$
$) \longrightarrow_r currentScreen{:}screenMsg := \ll closeDoor \gg$
$)$

definition *TISShutdownOp* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
$TISShutdownOp = (UEC(ShutdownOK) \vee UEC(ShutdownWaitingDoor))$

definition *OverrideDoorLockOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$OverrideDoorLockOK =$
$AdminOpStartedContext\ ;;$
$((admin{:}currentAdminOp = \ll Some(overrideLock) \gg$
$) \longrightarrow_r currentScreen{:}screenMsg := \ll requestAdminOp \gg\ ;;$
$currentDisplay := \ll doorUnlocked \gg\ ;;$
$internal{:}enclaveStatus := \ll enclaveQuiescent \gg\ ;;$

UnlockDoor ;;
AdminFinishOp)

lemma $\{\!|IDStation\text{-}inv|\!\}\, OverrideDoorLockOK\, \{\!|IDStation\text{-}inv|\!\}_u$
apply (*rule IDStation-inv-intro*)
oops

definition *TISOverrideDoorLockOp* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISOverrideDoorLockOp =
(*UEC*(*OverrideDoorLockOK*)
∨ *UEC*((*internal*:*enclaveStatus* = «*waitingStartAdminOp*»
∧ *admin*:*currentAdminOp* = «*Some*(*overrideLock*)») ⟶$_r$ *BadAdminLogout*))

definition *TISUpdateConfigDataOp* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]: *TISUpdateConfigDataOp* = *false*

definition *TISArchiveLog* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]: *TISArchiveLog* = *false*

definition *TISAdminOp* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISAdminOp = (*TISOverrideDoorLockOp* ∨ *TISShutdownOp* ∨ *TISUpdateConfigDataOp* ∨ *TISArchiveLog*)
definition *TISAdminLogout* :: *SystemState hrel* **where** [*upred-defs*, *tis-defs*]: *TISAdminLogout* = *false*
definition *TISIdle* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISIdle = *UEC*((*internal*:*status* = «*quiescent*»
∧ *internal*:*enclaveStatus* = «*enclaveQuiescent*»
∧ *iuserToken*:*userTokenPresence* = «*absent*»
∧ *iadminToken*:*adminTokenPresence* = «*absent*»
∧ *admin*:*rolePresent* = «*None*») ⟶$_r$ *II*)

11 The Whole ID Station

definition *TISOp* :: *SystemState hrel* **where**
TISOp = ((*TISEnrolOp*
∨ *TISUserEntryOp*
∨ *TISAdminLogon*
∨ *TISStartAdminOp*
∨ *TISAdminOp*
∨ *TISAdminLogout*
∨ *TISIdle*))

definition *InitDoorLatchAlarm* **where**
[*upred-defs*]:
InitDoorLatchAlarm =

(*DoorLatchAlarm* ∧
&*currentTime* =$_u$ «*zeroTime*» ∧
&*currentDoor* =$_u$ «*closed*» ∧
&*latchTimeout* =$_u$ «*zeroTime*» ∧
&*alarmTimeout* =$_u$ «*zeroTime*»)

lemma *InitDoorLatchAlarm* ≠ *false*
by (*rel-auto*)

abbreviation *TISOpThenUpdate* ≡ (*TISOp* ;; *TISUpdate*)

12 Proving Security Properties

lemma *RealWorld-wp* [*wp*]: ⟦*controlled* ♯ *b*; *monitored* ♯ *b*⟧ ⟹ (*RealWorldChanges wp* @*b*) = *b*
by (*simp add*: *tis-defs wp usubst unrest*)

lemma
([&*idStation*:*doorLatchAlarm*:*currentLatch* ↦$_s$ «*locked*»] †
(*TISReadUserToken wp* (*idStation*:*doorLatchAlarm*:*currentLatch* = «*unlocked*»)))
= *false*
by (*simp add*: *tis-defs wp usubst unrest alpha*)

12.1 Proving Security Functional Requirement 1

lemma [*wp*]: (*RealWorldChanges wlp false*) = *false*
by (*rel-auto*)

definition *AdminTokenGuardOK* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
AdminTokenGuardOK =
(&*iadminToken*:*currentAdminToken* ∈$_u$ «*range*(*goodT*)» ∧
(∃ *t*∈«*TokenWithValidAuth*» ·
(«*goodT*(*t*)» =$_u$ &*iadminToken*:*currentAdminToken*
∧ (∃ *c* ∈ «*AuthCert*» · «*Some c* = *authCert t*»
∧ «*role c* = *guard*») ⊕$_p$ *keyStore*
))
)

lemma *admin-unlock*:
[&*idStation*:*doorLatchAlarm*:*currentLatch* ↦$_s$ «*locked*»]
† ((*TISAdminOp* ;; *TISUpdate*) *wp* (*realWorld*:*controlled*:*latch* = «*unlocked*»)) =
((&*idStation*:*internal*:*enclaveStatus* =$_u$ «*waitingStartAdminOp*» ∧ &*idStation*:*iadminToken*:*adminToken*
=$_u$ «*present*») ∧
&*idStation*:*admin*:*currentAdminOp* =$_u$ «*Some overrideLock*» ∧ &*idStation*:*admin*:*rolePresent*

$\neq_u$ $None_u$ $\wedge$ &$idStation$:$admin$:$currentAdminOp$ $\neq_u$ $None_u$)
by (*simp add*: *tis-defs wp usubst unrest alpha*)

lemma *user-unlock*:
[&$idStation$:$doorLatchAlarm$:$currentLatch$ $\mapsto_s$ $\ll locked \gg$]
† ((*TISUserEntryOp* ;; *TISUpdate*) *wp* (*realWorld*:*controlled*:*latch* = $\ll$*unlocked*$\gg$)) =
(&$idStation$:*internal*:*status* $=_u$ $\ll$*waitingRemoveTokenSuccess*$\gg$ $\wedge$ &$idStation$:*iuserToken*:*userTokenPresence* $=_u$ $\ll$*absent*$\gg$)
by (*simp add*: *tis-defs alpha unrest usubst wp*)

SFR1(a): If the system invariants hold, the door is initially locked, and a *TISUserEntryOp* transition is enabled that unlocks the door, then (1) a valid user token is present and (2) either a valid finger print or a valid authorisation certificate is also present.

lemma *FSFR1a*:
‘(*IDStation-inv*) $\oplus_p$ *idStation* $\wedge$
[&$idStation$:$doorLatchAlarm$:$currentLatch$ $\mapsto_s$ $\ll locked \gg$]
† ((*TISUserEntryOp* ;; *TISUpdate*) *wp* (*realWorld*:*controlled*:*latch* = $\ll$*unlocked*$\gg$))
$\Rightarrow$ ((*UserTokenOK* $\wedge$ *FingerOK*) $\vee$ (*UserTokenWithOKAuthCert*)) $\oplus_p$ *idStation*‘
apply (*simp add*: *user-unlock*)
apply (*rel-auto*)
done

SFR1(a): If the system invariants hold, the door is initially locked, and a *TISAdminOp* transition is enabled that unlocks the door, then an admin token is present with the role “guard” attached.

lemma *FSFR1b*:
‘((*IDStation-inv2* $\wedge$ (*Admin* $\oplus_p$ *admin*) $\wedge$ *IDStation-inv10*) $\oplus_p$ *idStation* $\wedge$
[&$idStation$:$doorLatchAlarm$:$currentLatch$ $\mapsto_s$ $\ll locked \gg$]
† ((*TISAdminOp* ;; *TISUpdate*) *wp* (*realWorld*:*controlled*:*latch* = $\ll$*unlocked*$\gg$)))
$\Rightarrow$ *AdminTokenGuardOK*
$\oplus_p$ *idStation*‘
apply (*simp add*: *admin-unlock*)
apply (*simp add*: *Admin-def alpha*)
apply (*rel-auto*)
done

definition *AlarmInv* :: *SystemState upred* **where**
[*upred-defs*, *tis-defs*]:
AlarmInv = (*realWorld*:*controlled*:*latch* = $\ll$*locked*$\gg$ $\wedge$
idStation:*doorLatchAlarm*:*currentDoor* = $\ll$*dopen*$\gg$ $\wedge$
idStation:*doorLatchAlarm*:*currentTime* $\geq$ *idStation*:*doorLatchAlarm*:*alarmTimeout*
$\Rightarrow$ *realWorld*:*controlled*:*alarm* = $\ll$*alarming*$\gg$)$_e$

lemma {{*realWorld*:*controlled*:*latch* = ≪*locked*≫ ∧
idStation:*doorLatchAlarm*:*currentDoor* = ≪*dopen*≫ ∧
idStation:*doorLatchAlarm*:*currentTime* ≥ *idStation*:*doorLatchAlarm*:*alarmTimeout*
∧
(@*DoorLatchAlarm* ⊕$_p$ *idStation*:*doorLatchAlarm*)}} *TISUpdate*{{*realWorld*:*controlled*:*alarm*
= ≪*alarming*≫}}
oops

end