Tokeneer in Isabelle/UTP

Simon Foster, Mario Gleirscher, and Yakoub Nemouchi

July 14, 2020

Contents

1 Tokeneer in Isabelle/UTP

theory *Tokeneer*
imports
ZedLite.zedlite
UTP.utp
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* =

tisCert :: *IDCert*
issuerCerts :: *IDCert set*

We had to add two extra clauses to Enrol here that were specified in the Tokeneer Z-schema, namely that (1) all issuer certificates correspond to elements 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*. *tisCert 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*))})

lemma *ValidEnrol-functional*:
e ∈ *ValidEnrol* ⟹ *functional* {(*subject c*, *subjectPubK c*) | *c*. *c* ∈ *issuerCerts e*}
apply (*simp add*: *functional-def*)
apply (*simp add*: *ValidEnrol-def Enrol-def*)
apply (*rule inj-onI*)
apply (*force*)
done

lemma *Enrol-function*:
e ∈ *ValidEnrol* ⟹ {(*subject c*, *subjectPubK c*) | *c*. *c* ∈ *issuerCerts e*} ∈ *ISSUER* →$_r$ *KEYPART*
apply (*rule rel-pfun-intro*)
apply (*simp add*: *rel-typed-def Enrol-def ValidEnrol-def*)
apply *blast*
apply (*simp add*: *ValidEnrol-functional*)
done

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 = *U*(
  (*currentLatch* = ≪*locked*≫ ⇔ *currentTime* ≥ *latchTimeout*) ∧
  (*doorAlarm* = ≪*alarming*≫ ⇔
    (*currentDoor* = ≪*dopen*≫
      ∧ *currentLatch* = ≪*locked*≫
      ∧ *currentTime* ≥ *alarmTimeout*))
)

3 The Token ID Station

3.1 Configuration Data

consts *maxSupportedLogSize* :: *nat*

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

definition *Config* :: *Config upred* **where**
[*upred-defs*, *tis-defs*]:
Config = *U*(*alarmThresholdSize* < *minPreservedLogSize* ∧

$minPreservedLogSize \leq \ll maxSupportedLogSize\gg \wedge$
$latchUnlockDuration > 0 \wedge$
$alarmSilentDuration > 0)$

3.2 AuditLog

typedecl *AuditEvent*
typedecl *AuditUser*

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

Rather than axiomatising this, we explicitly define it and prove the two axioms as theorems.

definition *sizeLog* :: *Audit set* ⇒ *nat* **where**
sizeLog A = $(\sum\ a \in A.\ sizeElement\ a)$

lemma *sizeLog-empty* [*simp*]: *sizeLog* {} = *0*
  by (*simp add*: *sizeLog-def*)

lemma *sizeLog-calc*:
  assumes *finite L*
  shows *entry* ∈ *L* ⟹ *sizeLog L* = *sizeLog* (*L* − {*entry*}) + *sizeElement entry*
  using *assms*
  by (*simp add*: *sizeLog-def sum.remove*)

3.2.1 Real World Types and Entities (2)

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

definition *FLOPPY* :: *FLOPPY upred* **where**
[*upred-defs*, *tis-defs*]:
FLOPPY = *U*(∀ *e*. &**v** = ≪*enrolmentFile e*≫ ⇒ ≪*e* ∈ *ValidEnrol*≫)

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

definition *Floppy* :: *Floppy upred* **where**
[*upred-defs*, *tis-defs*]:
Floppy = (*FLOPPY* $\oplus_p$ *currentFloppy* ∧ *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* ≡ *true*

3.3 System Statistics

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

abbreviation *Stats* :: *Stats upred* **where** *Stats* ≡ *true*

3.4 Key Store

alphabet *KeyStore* =
  issuerKey :: *USER* ↔ *KEYPART*
  ownName :: *USER option*

definition *KeyStore* :: *KeyStore upred* **where**
[*upred-defs*, *tis-defs*]:
KeyStore =
  U(*issuerKey* ∈ ≪*ISSUER* ⇀$_r$ *KEYPART*≫ ∧
    udom(*issuerKey*) ⊆ ≪*ISSUER*≫ ∧
    (*ownName* ≠ ≪*None*≫ ⇒ *the*(*ownName*) ∈ *udom*(*issuerKey*)))

definition *CertIssuerKnown* :: *'a Certificate-scheme* ⇒ *KeyStore upred* **where**
[*upred-defs*, *tis-defs*]:
CertIssuerKnown c =
  U(*KeyStore* ∧
   (≪*c* ∈ *Certificate*≫ ∧
   ≪*issuer* (*cid c*)≫ ∈ *udom*(*issuerKey*)))

declare [[*coercion rel-apply*]]

term *U*(*issuerKey*(*x*))

definition *CertOK* :: *'a Certificate-scheme* ⇒ *KeyStore upred* **where**
[*upred-defs*, *tis-defs*]:
CertOK c =
  (*CertIssuerKnown c* ∧
   U(*Some*(*issuerKey*(≪*issuer* (*cid c*)≫)) = ≪*isValidatedBy c*≫))

definition *CertIssuerIsThisTIS* :: *'a Certificate-scheme* ⇒ *KeyStore upred* **where**
[*upred-defs*, *tis-defs*]:
CertIssuerIsThisTIS c =
(*KeyStore* ∧
U(≪*c* ∈ *Certificate*≫ ∧
(*ownName* ≠ ≪*None*≫ ∧
≪*issuer* (*cid c*)≫ = *the*(*ownName*))))

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* (*auditTime* ‘ *lg*))

definition *newestLogTime* :: *Audit set* ⇒ *TIME* **where**
[*upred-defs*, *tis-defs*]:
newestLogTime lg = (*Max* (*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 =
U((*rolePresent* ≠ ≪*None*≫ ⇒ *the*(*rolePresent*) ∈ ≪*ADMINPRIVILEGE*≫) ∧
(*rolePresent* = ≪*None*≫ ⇒ *availableOps* = {}) ∧
(*rolePresent* = ≪*Some guard*≫ ⇒ *availableOps* = {≪*overrideLock*≫}) ∧
(*rolePresent* = ≪*Some auditManager*≫ ⇒ *availableOps* = {≪*archiveLog*≫}) ∧
(*rolePresent* = ≪*Some securityOfficer*≫
⇒ *availableOps* = {≪*updateConfigData*≫,≪*shutdownOp*≫}) ∧
(*currentAdminOp* ≠ ≪*None*≫ ⇒
the(*currentAdminOp*) ∈ *availableOps* ∧ *rolePresent* ≠ ≪*None*≫)
)

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 = *U*((∃ *t*. *currentUserToken* = *goodT*(≪*t*≫)) ⇒ *ofGoodT*(*currentUserToken*)
∈ ≪*Token*≫)

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

definition *AdminToken* :: *AdminToken upred* **where**
[*upred-defs*, *tis-defs*]:
AdminToken = *U*((∃ *t*. *currentAdminToken* = *goodT*(≪*t*≫)) ⇒ *ofGoodT*(*currentAdminToken*)
∈ ≪*Token*≫)

3.7 Internal State

datatype *STATUS* = *quiescent* | *gotUserToken* | *waitingFinger* | *gotFinger* | *waitingUpdateToken* |
waitingEntry | *waitingRemoveTokenSuccess* | *waitingRemoveTokenFail*

datatype *ENCLAVESTATUS* = *notEnrolled* | *waitingEnrol* | *waitingEndEnrol* |
enclaveQuiescent |
gotAdminToken | *waitingRemoveAdminTokenFail* | *waitingStartAdminOp* | *waitingFinishAdminOp* |

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*)$\gg$ $\wedge$
(∃ *t*$\in\ll$*TokenWithValidAuth*$\gg$ $\cdot$
($\ll$*goodT*(*t*)$\gg$ $=_u$ &*iuserToken*:*currentUserToken*
$\wedge$ (∃ *c* $\in$ $\ll$*IDCert*$\gg$ $\cdot$ $\ll$*c* = *idCert t*$\gg$ $\wedge$ *CertOK c*) $\oplus_p$ *keyStore*
$\wedge$ (∃ *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*)$\gg$ $\wedge$
(∃ *t* $\cdot$
($\ll$*goodT*(*t*)$\gg$ $=_u$ &*iuserToken*:*currentUserToken*
$\wedge$ $\ll$*t* $\in$ *CurrentToken ti*$\gg$[*ti* $\rightarrow$ &*doorLatchAlarm*:*currentTime*]

```
    ∧ (∃ c ∈ «IDCert» • «c = idCert t» ∧ CertOK c) ⊕p keyStore
    ∧ (∃ c ∈ «PrivCert» • «c = privCert t» ∧ CertOK c) ⊕p keyStore
    ∧ (∃ c ∈ «IandACert» • «c = iandACert t» ∧ CertOK c) ⊕p keyStore))
  )

definition AdminTokenOK :: IDStation upred where
[upred-defs, tis-defs]:
AdminTokenOK =
  (&iadminToken:currentAdminToken ∈u «range(goodT)» ∧
   (∃ t∈«TokenWithValidAuth» •
     («goodT(t)» =u &iadminToken:currentAdminToken
     ∧ «t ∈ CurrentToken ti»⟦ti → &doorLatchAlarm:currentTime⟧
     ∧ (∃ c ∈ «IDCert» • «c = idCert t» ∧ CertOK c) ⊕p keyStore
     ∧ (∃ c ∈ «AuthCert» • «Some c = authCert t» ∧ AuthCertOK c
       ∧ «role c ∈ ADMINPRIVILEGE») ⊕p keyStore
   ))
  )

definition FingerOK :: IDStation upred where
[upred-defs, tis-defs]:
FingerOK = (
  Finger ⊕p ifinger ∧
  UserToken ⊕p iuserToken ∧
  &ifinger:currentFinger ∈u «range(goodFP)»)

definition IDStation-inv1 :: IDStation upred where
  [upred-defs, tis-defs]:
  IDStation-inv1 =
  U(&internal:status ∈
  {«gotFinger», «waitingFinger», «waitingUpdateToken», «waitingEntry», «waitingRemoveTokenSuccess»}
   ⇒ (@UserTokenWithOKAuthCert ∨ @UserTokenOK))

definition IDStation-inv2 :: IDStation upred where
  [upred-defs, tis-defs]:
  IDStation-inv2 =
   U(&admin:rolePresent ≠ «None» ⇒ @AdminTokenOK)

definition IDStation-inv3 :: IDStation upred where
  [upred-defs, tis-defs]:
  IDStation-inv3 =
  U(&internal:enclaveStatus ∉ {«notEnrolled», «waitingEnrol», «waitingEndEnrol»} ⇒
       &keyStore:ownName ≠ «None»)

definition IDStation-inv4 :: IDStation upred where
  [upred-defs, tis-defs]:
  IDStation-inv4 =
  U(&internal:enclaveStatus ∈ {«waitingStartAdminOp», «waitingFinishAdminOp»}
```

$\Leftrightarrow$ &*admin*:*currentAdminOp* $\neq$ «*None*»)

definition *IDStation-inv5* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv5 =
U(&*admin*:*currentAdminOp* $\neq$ «*None*» $\wedge$ *the*(&*admin*:*currentAdminOp*) $\in$ {«*shutdownOp*», «*overrideLock*»}
$\Rightarrow$ &*internal*:*enclaveStatus* = «*waitingStartAdminOp*»)

definition *IDStation-inv6* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv6 = *U*(&*internal*:*enclaveStatus* = «*gotAdminToken*» $\Rightarrow$ &*admin*:*rolePresent* = «*None*»)

definition *IDStation-inv7* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv7 = *U*(&*currentScreen*:*screenStats* = *displayStats stats*)

definition *IDStation-inv8* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv8 = *U*(&*currentScreen*:*screenConfig* = *displayConfigData config*)

Extra Invariant (1):

definition *IDStation-inv9* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv9 =
U(&*internal*:*status* $\in$
{«*waitingEntry*», «*waitingRemoveTokenSuccess*»}
$\Rightarrow$ (@*UserTokenWithOKAuthCert* $\vee$ @*FingerOK*))

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 =
U(&*iadminToken*:*adminTokenPresence* = «*present*» $\wedge$ &*admin*:*rolePresent* $\neq$ «*None*»
$\Rightarrow$ &*admin*:*rolePresent* = *Some*(*role*(*the*(*authCert*(*ofGoodT*(&*iadminToken*:*currentAdminToken*))))))

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 ∧
IDStation-inv2 ∧
IDStation-inv3 ∧
IDStation-inv4 ∧
IDStation-inv5 ∧
IDStation-inv6 ∧
IDStation-inv7 ∧
IDStation-inv8 ∧
IDStation-inv9 ∧
IDStation-inv10)

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

lemma *IDStation-correct-intro*:
assumes {*DoorLatchAlarm* $\oplus_p$ *doorLatchAlarm* ∧ *Floppy* $\oplus_p$ *ifloppy* ∧ *KeyStore* $\oplus_p$ *keyStore* ∧ *Admin* $\oplus_p$ *admin* ∧
Config $\oplus_p$ *config* ∧ *AdminToken* $\oplus_p$ *iadminToken* ∧ *UserToken* $\oplus_p$ *iuserToken*}
P
{*DoorLatchAlarm* $\oplus_p$ *doorLatchAlarm* ∧ *Floppy* $\oplus_p$ *ifloppy* ∧ *KeyStore* $\oplus_p$ *keyStore* ∧ *Admin* $\oplus_p$ *admin* ∧
Config $\oplus_p$ *config* ∧ *AdminToken* $\oplus_p$ *iadminToken* ∧ *UserToken* $\oplus_p$ *iuserToken*}
{*IDStation-inv*}*P*{*IDStation-inv*}
shows {*IDStation*}*P*{*IDStation*}
using *assms*
proof −
have *f1*: (*IDStation-inv* ∧ *DoorLatchAlarm* $\oplus_p$ *doorLatchAlarm* ∧ *Floppy* $\oplus_p$ *ifloppy* ∧ *KeyStore* $\oplus_p$ *keyStore* ∧ *Admin* $\oplus_p$ *admin* ∧ *Config* $\oplus_p$ *config* ∧ *AdminToken* $\oplus_p$ *iadminToken* ∧ *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* ∧ *Floppy* $\oplus_p$ *ifloppy* ∧ *KeyStore* $\oplus_p$ *keyStore* ∧ *Admin* $\oplus_p$ *admin* ∧ *Config* $\oplus_p$ *config* ∧ *AdminToken* $\oplus_p$ *iadminToken* ∧ *UserToken* $\oplus_p$ *iuserToken*}
by (*metis* (*no-types*) *assms*(*1*) *hoare-r-weaken-pre*(*2*))
have {*IDStation*} *P* {*IDStation-inv*}
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-inv1*}*P*{*IDStation-inv1*}
{*IDStation-inv2*}*P*{*IDStation-inv2*}
{*IDStation-inv3*}*P*{*IDStation-inv3*}
{*IDStation-inv4*}*P*{*IDStation-inv4*}
{*IDStation-inv5*}*P*{*IDStation-inv5*}
{*IDStation-inv6*}*P*{*IDStation-inv6*}
{*IDStation-inv7*}*P*{*IDStation-inv7*}
{*IDStation-inv8*}*P*{*IDStation-inv8*}
{*IDStation-inv9*}*P*{*IDStation-inv9*}
{*IDStation-inv10*}*P*{*IDStation-inv10*}
shows {*IDStation-inv*}*P*{*IDStation-inv*}
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 ≡ *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* = ($*monitored*:*now*´ ≥$_u$ $*monitored*:*now*)
  by (*rel-auto*, *simp add*: *nat-le-iff-add*)

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

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

5 Internal Operations

The Z-Schema for AddElementsToLog seems to allow, in some cases, the audit log to grow beyond its maximum size. As a I understand, it can nondeterminstically chose to extend the log anyway, or else remove old log entries.

definition *AddElementsToLog* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
  AddElementsToLog =
    (⨅ *newElements* :: *Audit set* ·
      ?[«*finite*(*newElements*)»
       ∧ «*newElements* ≠ {}»
       ∧ «*oldestLogTime newElements*» ≥ *newestLogTime* &*audit*:*auditLog*] ;;
     ((*audit*:*auditLog* := &*audit*:*auditLog* ∪ «*newElements*» ;;
       if (*sizeLog* &*audit*:*auditLog* < &*config*:*alarmThresholdSize*)
         then II
         else audit:*auditAlarm* := *alarming*
       fi)
      ⊓
     (?[*sizeLog* &*audit*:*auditLog* + «*sizeLog newElements*» > &*config*:*minPreservedLogSize*]
;;
       (⨅ *oldElements* :: *Audit set* ·

$$
\begin{array}{l}
\quad ?[\ll oldElements\gg \subseteq \&audit{:}auditLog \\
\quad\ \wedge\ oldestLogTime\ \&audit{:}auditLog \geq \ll oldestLogTime\ oldElements\gg] \ ;; \\
\quad audit{:}auditLog := (\&audit{:}auditLog - \ll oldElements\gg) \cup \ll newElements\gg \ ;; \\
\quad\ ?[sizeLog\ \&audit{:}auditLog \geq \&config{:}minPreservedLogSize] \ ;; \\
\quad\ audit{:}auditAlarm := alarming \\
\quad) \\
\quad) \\
\quad) \\
\quad)
\end{array}
$$

I believe this achieves the same effect as the Z-Schema – add some elements to the log (audit elements presumably) and choose some continuous subset of the result log for archiving.

definition *ArchiveLog* :: *Audit set* ⇒ *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ArchiveLog archive =
$$
\begin{array}{l}
\quad AddElementsToLog \ ;; \\
\quad (\bigsqcap\ (notArchived :: Audit\ set) \cdot \\
\quad\ ?[\ll archive\gg \subseteq \&audit{:}auditLog \\
\quad\ \wedge \ll newestLogTime\ archive\gg \leq newestLogTime\ (\&audit{:}auditLog - \ll archive\gg)] \\
\quad)
\end{array}
$$

definition *ClearLog* :: *Audit set* ⇒ *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ClearLog archive =
$$
\begin{array}{l}
\ (\bigsqcap\ (sinceArchive :: Audit\ set, lostSinceArchive :: Audit\ set) \cdot \\
\quad\ ?[\ll archive \cup sinceArchive\gg = \ll lostSinceArchive\gg \cup \&audit{:}auditLog \\
\quad\ \wedge \ll oldestLogTime\ sinceArchive\gg \geq \ll newestLogTime\ archive\gg \\
\quad\ \wedge \ll newestLogTime\ sinceArchive\gg \leq oldestLogTime\ \&audit{:}auditLog] \ ;; \\
\quad\ audit{:}auditLog := \ll sinceArchive\gg) \ ;; \\
\ if\ (sizeLog\ \&audit{:}auditLog < \&config{:}alarmThresholdSize) \\
\quad then\ audit{:}auditAlarm := silent \\
\quad else\ audit{:}auditAlarm := alarming \\
\ fi
\end{array}
$$

abbreviation *ClearLogThenAddElements archive* ≡ *ClearLog archive* ;; *AddElementsToLog*

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*}
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 = (
«*token* ∈ *ValidToken*» ∧
KeyStore ⊕$_p$ *keyStore* ∧

$Config \oplus_p config \wedge$

$\&keyStore{:}ownName \neq_u None_u \wedge$

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

5.3.2 Adding Authorisation Certificates to User Token

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

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 =
$(\Delta[tis{:}doorLatchAlarm, DoorLatchAlarm] \wedge$
$\$tis{:}doorLatchAlarm{:}currentTime' =_u \$realWorld{:}monitored{:}now)$

definition *PollDoor* :: *SystemState hrel* **where**
[*upred-defs*]:
PollDoor =
$(\Delta[tis{:}doorLatchAlarm, DoorLatchAlarm] \wedge$
$\$tis{:}doorLatchAlarm{:}currentDoor' =_u \$realWorld{:}monitored{:}door \wedge$
$\$tis{:}doorLatchAlarm{:}latchTimeout' =_u \$tis{:}doorLatchAlarm{:}latchTimeout \wedge$
$\$tis{:}doorLatchAlarm{:}alarmTimeout' =_u \$tis{:}doorLatchAlarm{:}alarmTimeout)$

definition *PollUserToken* :: *SystemState hrel* **where**
[*upred-defs*]:
PollUserToken =

$(\Delta[tis{:}iuserToken, UserToken] \wedge$
$\$tis{:}iuserToken{:}userTokenPresence' =_u \ll present \gg \Leftrightarrow \$realWorld{:}monitored{:}userToken$
$\neq_u \ll noT \gg \wedge$
$\$tis{:}iuserToken{:}currentUserToken' =_u$
$(\$realWorld{:}monitored{:}userToken \triangleleft \$realWorld{:}monitored{:}userToken \neq_u \ll noT \gg \triangleright$
$\$tis{:}iuserToken{:}currentUserToken))$

definition *PollAdminToken* :: *SystemState hrel* **where**
[*upred-defs*]:
PollAdminToken =
$(\Delta[tis{:}iadminToken, AdminToken] \wedge$
$\$tis{:}iadminToken{:}adminTokenPresence' =_u \ll present \gg \Leftrightarrow \$realWorld{:}monitored{:}adminToken$
$\neq_u \ll noT \gg \wedge$
$\$tis{:}iadminToken{:}currentAdminToken' =_u$
$(\$realWorld{:}monitored{:}adminToken \triangleleft \$realWorld{:}monitored{:}adminToken \neq_u$
$\ll noT \gg \triangleright \$tis{:}iadminToken{:}currentAdminToken))$

definition *PollFinger* :: *SystemState hrel* **where**
[*upred-defs*]:
PollFinger =
$(\Delta[tis{:}ifinger, Finger] \wedge$
$\$tis{:}ifinger{:}fingerPresence' =_u \ll present \gg \Leftrightarrow \$realWorld{:}monitored{:}finger \neq_u$
$\ll noFP \gg \wedge$
$\$tis{:}ifinger{:}currentFinger' =_u$
$(\$realWorld{:}monitored{:}finger \triangleleft \$realWorld{:}monitored{:}finger \neq_u \ll noFP \gg \triangleright$
$\$tis{:}ifinger{:}currentFinger))$

definition *PollFloppy* :: *SystemState hrel* **where**
[*upred-defs*]:
PollFloppy =
$(\Delta[tis{:}ifloppy, Floppy] \wedge$
$\$tis{:}ifloppy{:}floppyPresence' =_u \ll present \gg \Leftrightarrow \$realWorld{:}monitored{:}floppy \neq_u$
$\ll noFloppy \gg \wedge$
$\$tis{:}ifloppy{:}currentFloppy' =_u$
$(\$realWorld{:}monitored{:}floppy \triangleleft \$realWorld{:}monitored{:}floppy \neq_u \ll noFloppy \gg \triangleright$
$\$tis{:}ifloppy{:}currentFloppy) \wedge$
$\$tis{:}ifloppy{:}writtenFloppy' =_u \$tis{:}ifloppy{:}writtenFloppy$
$)$

definition *PollKeyboard* :: *SystemState hrel* **where**
[*upred-defs*]:
PollKeyboard =
$(\Delta[tis{:}ikeyboard, Keyboard] \wedge$
$\$tis{:}ikeyboard{:}keyedDataPresence' =_u \ll present \gg \Leftrightarrow \$realWorld{:}monitored{:}keyboard$
$\neq_u \ll noKB \gg \wedge$
$\$tis{:}ikeyboard{:}currentKeyedData' =_u$
$(\$realWorld{:}monitored{:}keyboard \triangleleft \$realWorld{:}monitored{:}keyboard \neq_u \ll noKB \gg \triangleright$
$\$tis{:}ikeyboard{:}currentKeyedData))$

definition *TISPoll* :: *SystemState hrel* **where**
[*tis-defs*, *upred-defs*]:
TISPoll =
(— PollTime
tis:*doorLatchAlarm*:*currentTime* := &*realWorld*:*monitored*:*now* ;;
— The following behaviour locks the door after a timeout and activates the alarm when necessary. This behaviour is implicit in the Z specification through the DoorLatchAlarm schema invariants.
tis:*doorLatchAlarm*:[
if (*currentTime* ≥ *latchTimeout*) *then currentLatch* := *locked else currentLatch* := *unlocked fi* ;;
if (*currentDoor* = ≪*dopen*≫
∧ *currentLatch* = ≪*locked*≫ ∧ *currentTime* ≥ *alarmTimeout*) *then doorAlarm* := *alarming else doorAlarm* := *silent fi*
]$^+$;;
— PollDoor
tis:*doorLatchAlarm*:*currentDoor* := &*realWorld*:*monitored*:*door* ;;
— PollUserToken
tis:*iuserToken*:*userTokenPresence* :=
(≪*absent*≫ ◁ (&*realWorld*:*monitored*:*userToken* = ≪*noT*≫) ▷ ≪*absent*≫) ;;
tis:*iuserToken*:*currentUserToken* :=
(&*tis*:*iuserToken*:*currentUserToken*
◁ (&*realWorld*:*monitored*:*userToken* = ≪*noT*≫) ▷
&*realWorld*:*monitored*:*userToken*) ;;
— PollAdminToken
tis:*iadminToken*:*adminTokenPresence* :=
(≪*absent*≫ ◁ (&*realWorld*:*monitored*:*adminToken* = ≪*noT*≫) ▷ ≪*absent*≫) ;;
tis:*iadminToken*:*currentAdminToken* :=
(&*tis*:*iadminToken*:*currentAdminToken*
◁ (&*realWorld*:*monitored*:*adminToken* = ≪*noT*≫) ▷
&*realWorld*:*monitored*:*adminToken*) ;;
— PollFinger
tis:*ifinger*:*fingerPresence* :=
(≪*absent*≫ ◁ (&*realWorld*:*monitored*:*finger* = ≪*noFP*≫) ▷ ≪*absent*≫) ;;
tis:*ifinger*:*currentFinger* :=
(&*tis*:*ifinger*:*currentFinger*
◁ (&*realWorld*:*monitored*:*finger* = ≪*noFP*≫) ▷
&*realWorld*:*monitored*:*finger*) ;;
— PollFloppy
tis:*ifloppy*:*floppyPresence* :=
(≪*absent*≫ ◁ (&*realWorld*:*monitored*:*floppy* = ≪*noFloppy*≫) ▷ ≪*absent*≫) ;;
tis:*ifloppy*:*currentFloppy* :=
(&*tis*:*ifloppy*:*currentFloppy*
◁ (&*realWorld*:*monitored*:*floppy* = ≪*noFloppy*≫) ▷
&*realWorld*:*monitored*:*floppy*) ;;
— PollKeyboard
tis:*ikeyboard*:*keyedDataPresence* :=
(≪*absent*≫ ◁ (&*realWorld*:*monitored*:*keyboard* = ≪*noKB*≫) ▷ ≪*absent*≫) ;;
tis:*ikeyboard*:*currentKeyedData* :=

$$
\begin{array}{l}
\quad (\&tis{:}ikeyboard{:}currentKeyedData \\
\quad\quad \triangleleft\ (\&realWorld{:}monitored{:}keyboard = \ll noKB \gg)\ \triangleright \\
\quad\ \&realWorld{:}monitored{:}keyboard) \\
)
\end{array}
$$

6.2 The ID Station Changes the World

6.2.1 Periodic Updates

abbreviation *UpdateLatch* :: *SystemState hrel* **where**

$$UpdateLatch \equiv realWorld{:}controlled{:}latch := \&tis{:}doorLatchAlarm{:}currentLatch$$

abbreviation *UpdateAlarm* :: *SystemState hrel* **where**

$$
\begin{array}{l}
UpdateAlarm \equiv \\
\quad realWorld{:}controlled{:}alarm := (\ll alarming \gg \\
\qquad\qquad\qquad\qquad \triangleleft\ (\&tis{:}doorLatchAlarm{:}doorAlarm = \ll alarming \gg \\
\qquad\qquad\qquad\qquad\quad \vee\ \&tis{:}audit{:}auditAlarm = \ll alarming \gg) \\
\qquad\qquad\qquad\qquad \triangleright\ \ll silent \gg)
\end{array}
$$

definition *UpdateDisplay* :: *SystemState hrel* **where**

[*upred-defs*]:

$$
\begin{array}{l}
UpdateDisplay = \\
\quad (\Delta[tis,IDStation] \wedge \\
\quad\ RealWorldChanges \oplus_r realWorld \wedge \\
\quad\ \$realWorld{:}controlled{:}display' =_u \$tis{:}currentDisplay \wedge \\
\quad\ \$tis{:}currentDisplay' =_u \$tis{:}currentDisplay)
\end{array}
$$

definition *UpdateScreen* :: *SystemState hrel* **where**

[*upred-defs*]:

$$
\begin{array}{l}
UpdateScreen = \\
\quad (\Delta[tis,IDStation] \wedge \\
\quad\ \Xi[tis{:}admin,Admin] \wedge \\
\quad\ RealWorldChanges \oplus_r realWorld \wedge \\
\quad\ \$realWorld{:}controlled{:}screen{:}screenMsg' =_u \$tis{:}currentScreen{:}screenMsg \wedge \\
\quad\ \$realWorld{:}controlled{:}screen{:}screenConfig' =_u \\
\qquad (\$tis{:}currentScreen{:}screenConfig \\
\qquad\quad \triangleleft\ \$tis{:}admin{:}rolePresent =_u \ll Some(securityOfficer) \gg\ \triangleright \\
\qquad\ \ll clear \gg) \wedge \\
\quad\ \$realWorld{:}controlled{:}screen{:}screenStats' =_u \\
\qquad (\$tis{:}currentScreen{:}screenStats \\
\qquad\quad \triangleleft\ \$tis{:}admin{:}rolePresent \neq_u \ll None \gg\ \triangleright \\
\qquad\ \ll clear \gg))
\end{array}
$$

definition *TISUpdate* :: *SystemState hrel* **where**

[*upred-defs*, *tis-defs*]:

$$
\begin{array}{l}
TISUpdate = \\
\quad (realWorld{:}[RealWorldChanges]^+ \ ;; \\
\quad\ UpdateLatch \ ;;
\end{array}
$$

UpdateAlarm ;;
realWorld:*controlled*:*display* := &*tis*:*currentDisplay*)

definition *UpdateFloppy* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
UpdateFloppy =
(*realWorld*:[*RealWorldChanges*]$^+$;;
realWorld:*monitored*:*floppy* := &*tis*:*ifloppy*:*writtenFloppy* ;;
tis:*ifloppy*:*currentFloppy* := *badFloppy*)

definition *TISEarlyUpdate* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISEarlyUpdate = *UpdateLatch* ;; *UpdateAlarm*

6.2.2 Updating the User Token

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

lemma *UpdateUserToken-correct*:
{*IDStation* $\oplus_p$ *tis***}** *UpdateUserToken***{***IDStation* $\oplus_p$ *tis***}**
by (*simp add*: *tis-defs*, *hoare-auto*)

7 The User Entry Operation (1)

definition *ResetScreenMessage* :: *IDStation hrel* **where**
[*upred-defs*]:
ResetScreenMessage =
(Δ[*admin*,*Admin*]
∧ (($*internal*:*status*´ $\notin_u$ {«*quiescent*»,«*waitingRemoveTokenFail*»}$_u$ ∧ $*currentScreen*:*screenMsg*´ $=_u$ «*busy*») ∨
($*internal*:*status*´ $\in_u$ {«*quiescent*»,«*waitingRemoveTokenFail*»}$_u$ ∧
($*internal*:*enclaveStatus*´ $=_u$ «*enclaveQuiescent*» ∧ $*admin*:*rolePresent*´ $=_u$ «*None*» ∧ $*currentScreen*:*screenMsg*´ $=_u$ «*welcomeAdmin*»
∨ $*internal*:*enclaveStatus*´ $=_u$ «*enclaveQuiescent*» ∧ $*admin*:*rolePresent*´ $\neq_u$ «*None*» ∧ $*currentScreen*:*screenMsg*´ $=_u$ «*requestAdminOp*»
∨ $*internal*:*enclaveStatus*´ $=_u$ «*waitingRemoveAdminTokenFail*» ∧ $*currentScreen*:*screenMsg*´ $=_u$ «*removeAdminToken*»
∨ $*internal*:*enclaveStatus*´ $\notin_u$ {«*enclaveQuiescent*»,«*waitingRemoveAdminTokenFail*»}$_u$ ∧ $*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* ∧ Ξ[*controlled*, *TISControlledRealWorld*]) ⊕$_r$ *realWorld* ∧
(Δ[*iuserToken*,*UserToken*] ∧
Δ[*doorLatchAlarm*,*DoorLatchAlarm*] ∧
Δ[*audit*,*AuditLog*] ∧
Ξ[*config*, *Config*] ∧
Ξ[*iadminToken*, *AdminToken*] ∧
Ξ[*keyStore*, *KeyStore*] ∧
Ξ[*admin*, *Admin*] ∧
Ξ[*ikeyboard*, *Keyboard*] ∧
Ξ[*ifloppy*, *Floppy*] ∧
Ξ[*ifinger*, *Finger*] ∧
Δ[*IDStation-inv*] ∧
ResetScreenMessage ∧
($*enclaveStatus*´ =$_u$ $*enclaveStatus* ∧
($*status* ≠$_u$ ≪*waitingEntry*≫ ⇒ $*tokenRemovalTimeout*´ =$_u$ $*tokenRemovalTimeout*)
) ⊕$_r$ *internal*) ⊕$_r$ *tis*
)

lemma *Pre UserEntryContext* = *IDStation* ⊕$_p$ *tis*
apply (*unfold UserEntryContext-def*)
apply (*simp*)
apply (*zcalcpre*)
oops

lemma *UserEntryContext-alt-def* [*upred-defs*]:
UserEntryContext =
((*RealWorldChanges* ∧ Ξ[*controlled*, *TISControlledRealWorld*]) ⊕$_r$ *realWorld* ∧
(Δ[*IDStation*] ∧
$*config*´ =$_u$ $*config* ∧
$*iadminToken*´ =$_u$ $*iadminToken* ∧
$*keyStore*´ =$_u$ $*keyStore* ∧
$*admin*´ =$_u$ $*admin* ∧
$*ikeyboard*´ =$_u$ $*ikeyboard* ∧
$*ifloppy*´ =$_u$ $*ifloppy* ∧
$*ifinger*´ =$_u$ $*ifinger* ∧
ResetScreenMessage ∧
($*enclaveStatus*´ =$_u$ $*enclaveStatus* ∧
$*status* ≠$_u$ ≪*waitingEntry*≫ ⇒ $*tokenRemovalTimeout*´ =$_u$ $*tokenRemovalTimeout*
) ⊕$_r$ *internal*) ⊕$_r$ *tis*
)
oops

lemma *Pre*((*RealWorldChanges* ∧ Ξ[*controlled*, *TISControlledRealWorld*]) ⊕$_r$ *realWorld*) = *true*
by (*rel-auto*)

7.1 User Token Tears

definition *UserTokenTorn* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
UserTokenTorn =
((&*internal*:*status* ∈ {«*gotUserToken*», «*waitingUpdateToken*», «*waitingFinger*», «*gotFinger*», «*waitingEntry*»}
∧ &*iuserToken*:*userTokenPresence* = «*absent*»
) ⟶$_r$ *currentDisplay* := «*welcom*» ;; *internal*:*status* := «*quiescent*»)

lemma *UserTokenTorn-correct* [*hoare-safe*]: {*IDStation*}*UserTokenTorn*{*IDStation*}
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

8 Operations within the Enclave (1)

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

definition *EnrolContext* :: *SystemState hrel* **where**
EnrolContext = (*EnclaveContext* ∧
Ξ[*tis*:*ikeyboard*, *Keyboard*] ∧
Ξ[*tis*:*admin*, *Admin*] ∧
Ξ[*tis*:*doorLatchAlarm*, *DoorLatchAlarm*] ∧
Ξ[*tis*:*config*, *Config*] ∧
Ξ[*tis*:*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* ⇒ *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
UpdateKeyStore e =
(≪*e* ∈ *ValidEnrol*≫)
⟶$_r$ *keyStore*:*ownName* := ≪*Some* (*subject* (*tisCert e*))≫
;; *keyStore*:*issuerKey* := &*keyStore*:*issuerKey*
+$_r$ ≪{(*subject c*, *subjectPubK c*) | *c*. *c* ∈ *issuerCerts e*}≫
+$_r$ {(*the*(&*keyStore*:*ownName*), ≪*subjectPubK* (*tisCert e*)≫)}

definition *call* :: ('*a* ⇒ '*s hrel*) ⇒ ('*a*, '*s*) *uexpr* ⇒ '*s hrel* (*call*) **where**
[*upred-defs*]: *call F e* = (⨅ *x* ⋅ *?*[≪*x*≫ = *e*] ;; *F x*)

lemma *nmods-call* [*closure*]: ⟦ ⋀ *v*. (*P v*) *nmods x* ⟧ ⟹ (*call P e*) *nmods x*
by (*rel-auto*)

lemma *hoare-call* [*hoare-safe*]: ⟦ ⋀ *x*. {≪*x*≫ = *e* ∧ *P*}*F x*{*Q*} ⟧ ⟹ {*P*}*call F e*{*Q*}
by (*rel-auto*)

lemma *wlp-call* [*wp*]: (*call F e*) *wlp b* = *U*(∀ *x*. ≪*x*≫ = *e* ⇒ *F x wlp b*)
by (*simp add*: *call-def wp*)

utp-lift-notation *call* (*0*)

definition *UpdateKeyStoreFromFloppy* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
UpdateKeyStoreFromFloppy =
(&*ifloppy*:*currentFloppy* ∈ *range*(*enrolmentFile*))
⟶$_r$ *call UpdateKeyStore* (*enrolmentFile-of* &*ifloppy*:*currentFloppy*)

declare *ValidEnrol-def* [*upred-defs del*, *tis-defs del*]

lemma *rel-pfun-pair*: ⟦ *x* ∈ *A*; *y* ∈ *B* ⟧ ⟹ {(*x*, *y*)} ∈ *A* ⇸$_r$ *B*
by (*simp add*: *rel-pfun-def rel-typed-def*)

lemma *UpdateKeyStore-KeyStore-inv*: {*KeyStore* ⊕$_p$ *keyStore*}*UpdateKeyStore x*{*KeyStore* ⊕$_p$ *keyStore*}
apply (*simp add*: *tis-defs del*:*Enrol-def*, *hoare-auto*)
apply (*rule rel-pfun-override*)
apply (*rule rel-pfun-override*)
apply (*auto*)
using *Enrol-function* **apply** *blast*
apply (*rule rel-pfun-pair*)

apply (*simp add*: *Enrol-def ValidEnrol-def subset-eq*)
apply *blast*
apply (*simp add*: *Enrol-def ValidEnrol-def subset-eq*)
apply (*simp add*: *Enrol-def ValidEnrol-def subset-eq*)
apply (*rule rel-pfun-override*)
apply (*rule rel-pfun-override*)
apply (*auto*)
using *Enrol-function* **apply** *blast*
apply (*rule rel-pfun-pair*)
apply (*simp add*: *Enrol-def ValidEnrol-def subset-eq*)
apply *blast*
apply (*simp add*: *Enrol-def ValidEnrol-def subset-eq*)
apply (*simp add*: *Enrol-def ValidEnrol-def subset-eq*)
done

lemma *hoare-sep-inv*: ⟦ {*p*}*S*{*p*}; {*q*}*S*{*q*} ⟧ ⟹ {*p* ∧ *q*}*S*{*p* ∧ *q*}
by (*rel-auto*)

lemma *UpdateKeyStore-IDStation-wf*: {*IDStation-wf*} *UpdateKeyStore x* {*IDStation-wf*}
apply (*simp add*: *IDStation-wf-def*)
apply (*auto intro*!:*hoare-sep-inv*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *UpdateKeyStore-KeyStore-inv*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

lemma *UpdateKeyStoreFromFloppy-IDStation-wf*: {*IDStation-wf*} *UpdateKeyStoreFromFloppy* {*IDStation-wf*}
apply (*simp add*: *UpdateKeyStoreFromFloppy-def*)
apply (*hoare-split*)
apply (*metis UpdateKeyStore-IDStation-wf conj-comm hoare-r-weaken-pre*(*2*))
done

9 The User Entry Operation (2)

definition *UEC* :: *IDStation hrel* ⇒ *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
UEC(*Op*) =
(⨅ *t* · *tis*:[*Op*]$^+$;;
realWorld:[
monitored:*now* := &*monitored*:*now* + *t* ;;
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 *UEC-correct*: $\{I\}P\{I\} \Longrightarrow \{I \oplus_p tis\}UEC(P)\{I \oplus_p tis\}$
apply (*simp add*: *wlp-hoare-link wp UEC-def alpha unrest usubst*)
apply (*rel-simp*)
done

9.1 Reading the User Token

definition *ReadUserToken* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]: *ReadUserToken* =
((&*internal*:*enclaveStatus* ∈ {*enclaveQuiescent*, *waitingRemoveAdminTokenFail*}
∧ &*internal*:*status* = *quiescent* ∧ &*iuserToken*:*userTokenPresence* = *present*
) ⟶$_r$ *currentDisplay* := *wait* ;; *internal*:*status* := *gotUserToken*)

lemma *ReadUserToken-correct*: {*IDStation*}*ReadUserToken*{*IDStation*}
by (*rule IDStation-correct-intro*; *hoare-wlp-auto defs*: *tis-defs*)

definition *BioCheckNotRequired* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]: *BioCheckNotRequired* =
((&*internal*:*status* = *gotUserToken*
∧ &*iuserToken*:*userTokenPresence* = *present* ∧ @*UserTokenWithOKAuthCert*
) ⟶$_r$ *internal*:*status* := *waitingEntry* ;; *currentDisplay* := *wait*)

lemma *BioCheckNotRequired-correct*: {*IDStation*}*BioCheckNotRequired*{*IDStation*}
by (*rule IDStation-correct-intro*; *hoare-wlp-auto defs*: *tis-defs*)

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*}*BioCheckRequired*{*IDStation*}
by (*rule IDStation-correct-intro*; (*simp add*: *tis-defs*, *hoare-auto*))

definition [*upred-defs*, *tis-defs*]: *TISReadUserToken* = *UEC*(*ReadUserToken*)

lemma *TISReadUserToken-correct*: {*IDStation* ⊕$_p$ *tis*}*TISReadUserToken*{*IDStation* ⊕$_p$ *tis*}
by (*simp add*: *ReadUserToken-correct TISReadUserToken-def UEC-correct*)

9.2 Validating the User Token

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

lemma *ValidateUserTokenOK-correct*: {*IDStation*} *ValidateUserTokenOK*{*IDStation*}
by (*simp add*: *BioCheckNotRequired-correct BioCheckRequired-correct ValidateUserTokenOK-def disj-upred-def hoare-ndet*)

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 *ValidateUserTokenFail-correct*: {*IDStation*} *ValidateUserTokenFail*{*IDStation*}
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

definition [*upred-defs*, *tis-defs*]:
TISValidateUserToken = (*UEC*(*ValidateUserTokenOK*) ∨ *UEC*(*ValidateUserTokenFail*)

∨ *UEC*(*UserTokenTorn* ;; *?*[&*internal*:*status* = «*gotUserToken*»]))

lemma *UserTokenTorn-test-correct*:
{*IDStation*}(*UserTokenTorn* ;; *?*[@*b*]){*IDStation*}
by (*rule seq-hoare-inv-r-2*, *simp add*: *hoare-safe*, *rule hoare-test*, *simp add*: *impl-alt-def utp-pred-laws.sup-commute*)

lemma *TISValidateUserToken-correct*: {*IDStation* ⊕$_p$ *tis*} *TISValidateUserToken*{*IDStation* ⊕$_p$ *tis*}
by (*simp add*: *TISValidateUserToken-def UEC-correct UserTokenTorn-test-correct ValidateUserTokenFail-correct ValidateUserTokenOK-correct disj-upred-def hoare-ndet*)

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*»)

lemma *ReadFingerOK-correct*: {*IDStation*} *ReadFingerOK*{*IDStation*}
by (*rule IDStation-correct-intro*, (*simp add*: *tis-defs*, *hoare-auto*)+)

definition *NoFinger* :: *IDStation hrel* **where**

[*upred-defs*, *tis-defs*]:
NoFinger =
  ?[&*internal*:*status* = «*waitingFinger*»
    ∧ &*ifinger*:*fingerPresence* = «*absent*»
    ∧ &*iuserToken*:*userTokenPresence* = «*present*»
  ]

lemma *NoFinger-correct*: {*IDStation*}*NoFinger*{*IDStation*}
  by (*rule IDStation-correct-intro*, (*simp add*: *tis-defs*, *hoare-auto*)+)

definition *FingerTimeout* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
FingerTimeout =
 ((&*internal*:*status* = «*waitingFinger*»
    ∧ &*ifinger*:*fingerPresence* = «*absent*»
    ∧ &*iuserToken*:*userTokenPresence* = «*present*»
) ⟶$_r$ *currentDisplay* := «*removeToken*» ;; *internal*:*status* := «*waitingRemove-TokenFail*»)

lemma *FingerTimeout-correct*: {*IDStation*}*FingerTimeout*{*IDStation*}
  apply (*rule IDStation-correct-intro*)
   apply (*simp add*: *tis-defs*, *hoare-auto*)
  apply (*simp add*: *tis-defs*, *hoare-auto*)
  done

definition [*upred-defs*, *tis-defs*]:
TISReadFinger = (*UEC*(*ReadFingerOK*) ∨ *UEC*(*FingerTimeout*) ∨ *UEC*(*NoFinger*)
                ∨ *UEC*(*UserTokenTorn* ;; *?*[&*internal*:*status* = «*waitingFinger*»]))

lemma *TISReadFinger-correct*: {*IDStation* ⊕$_p$ *tis*}*TISReadFinger*{*IDStation* ⊕$_p$ *tis*}
  by (*simp add*: *FingerTimeout-correct NoFinger-correct ReadFingerOK-correct TISReadFinger-def UEC-correct UserTokenTorn-test-correct disj-upred-def hoare-ndet*)

9.4 Validating a Fingerprint

definition *ValidateFingerOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ValidateFingerOK =
 ((&*internal*:*status* = «*gotFinger*»
    ∧ &*iuserToken*:*userTokenPresence* = «*present*»
    ∧ @*FingerOK*
) ⟶$_r$ *currentDisplay* := «*wait*» ;; *internal*:*status* := «*waitingUpdateToken*»)

lemma *ValidateFingerOK-correct*: {*IDStation*}*ValidateFingerOK*{*IDStation*}
  apply (*rule IDStation-correct-intro*)
   apply (*simp add*: *tis-defs*, *hoare-auto*)
  apply (*simp add*: *tis-defs*, *hoare-auto*)
  done

definition *ValidateFingerFail* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ValidateFingerFail =
((&*internal*:*status* = «*gotFinger*»
∧ &*iuserToken*:*userTokenPresence* = «*present*»
∧ @*FingerOK*
) ⟶$_r$ *currentDisplay* := «*removeToken*» ;; *internal*:*status* := «*waitingRemove-TokenFail*»)

lemma *ValidateFingerFail-correct*: {*IDStation*} *ValidateFingerFail*{*IDStation*}
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

definition [*upred-defs*, *tis-defs*]:
TISValidateFinger = (*UEC*(*ValidateFingerOK*) ∨ *UEC*(*ValidateFingerFail*)
∨ *UEC*(*UserTokenTorn* ;; *?*[&*internal*:*status* = «*gotFinger*»]))

lemma *TISValidateFinger-correct*: {*IDStation* ⊕$_p$ *tis*} *TISValidateFinger*{*IDStation* ⊕$_p$ *tis*}
by (*simp add*: *TISValidateFinger-def UEC-correct UserTokenTorn-test-correct ValidateFingerFail-correct ValidateFingerOK-correct disj-upred-def hoare-ndet*)

9.5 Writing the User Token

definition *WriteUserTokenOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
WriteUserTokenOK =
((&*internal*:*status* = «*waitingUpdateToken*»
∧ &*iuserToken*:*userTokenPresence* = «*present*»
) ⟶$_r$ *AddAuthCertToUserToken* ;;
currentDisplay := «*wait*» ;;
internal:*status* := «*waitingEntry*»)

lemma *hoare-post-conj-split*: {*b*}*P*{*c* ∧ *d*} ⟷ ({*b*}*P*{*c*} ∧ {*b*}*P*{*d*})
by (*rel-auto*)

lemma *WriteUserTokenOK-correct*: {*IDStation*} *WriteUserTokenOK*{*IDStation*}
proof −
have *inv*: {*IDStation-inv*} *WriteUserTokenOK* {*IDStation-inv*}
proof −
have *a*:{*IDStation-inv1* ∧ *IDStation-inv9*} *WriteUserTokenOK* {*IDStation-inv9*}
by (*hoare-wlp-auto defs*: *tis-defs*)
have *1*:{*IDStation-inv*} *WriteUserTokenOK* {*IDStation-inv9*}
by (*rule-tac pre-str-hoare-r*[*OF* - *a*], *rel-auto*)

```
    have b: {IDStation-inv1} WriteUserTokenOK {IDStation-inv1}
      by (hoare-wlp-auto defs: tis-defs)
    have 2:{IDStation-inv} WriteUserTokenOK {IDStation-inv1}
      by (rule-tac pre-str-hoare-r[OF - b], rel-auto)
    have 3:
      {IDStation-inv} WriteUserTokenOK {IDStation-inv2}
      {IDStation-inv} WriteUserTokenOK {IDStation-inv3}
      {IDStation-inv} WriteUserTokenOK {IDStation-inv4}
      {IDStation-inv} WriteUserTokenOK {IDStation-inv5}
      {IDStation-inv} WriteUserTokenOK {IDStation-inv6}
      {IDStation-inv} WriteUserTokenOK {IDStation-inv7}
      {IDStation-inv} WriteUserTokenOK {IDStation-inv8}
      {IDStation-inv} WriteUserTokenOK {IDStation-inv10}
      by (hoare-wlp-auto defs: tis-defs)+
    from 1 2 3 show ?thesis
      by (auto simp add: IDStation-inv-def hoare-post-conj-split)
  qed

  have ut: {UserToken ⊕p iuserToken} WriteUserTokenOK {UserToken ⊕p iuserToken}
    by (hoare-wlp-auto defs: tis-defs)

  show ?thesis
    apply (rule-tac IDStation-correct-intro)
    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 (simp add: ut hoare-r-weaken-pre(1) hoare-r-weaken-pre(2))
    apply (simp add: inv)
    done
qed

definition WriteUserTokenFail :: IDStation hrel where
[upred-defs, tis-defs]:
WriteUserTokenFail =
 ((&internal:status = «waitingUpdateToken»
     ∧ &iuserToken:userTokenPresence = «present»
  ) ⟶r AddAuthCertToUserToken ;;
        currentDisplay := «tokenUpdateFailed» ;;
        internal:status := «waitingEntry»)

lemma WriteUserTokenFail-correct: {IDStation} WriteUserTokenFail {IDStation}
proof −
  have inv: {IDStation-inv} WriteUserTokenFail {IDStation-inv}
  proof −
   have a:{IDStation-inv1 ∧ IDStation-inv9} WriteUserTokenFail {IDStation-inv9}
```

```
    by (hoare-wlp-auto defs: tis-defs)
  have 1:{IDStation-inv} WriteUserTokenFail {IDStation-inv9}
    by (rule-tac pre-str-hoare-r[OF - a], rel-auto)
  have b: {IDStation-inv1} WriteUserTokenFail {IDStation-inv1}
    by (hoare-wlp-auto defs: tis-defs)
  have 2:{IDStation-inv} WriteUserTokenFail {IDStation-inv1}
    by (rule-tac pre-str-hoare-r[OF - b], rel-auto)
  have 3:
    {IDStation-inv} WriteUserTokenFail {IDStation-inv2}
    {IDStation-inv} WriteUserTokenFail {IDStation-inv3}
    {IDStation-inv} WriteUserTokenFail {IDStation-inv4}
    {IDStation-inv} WriteUserTokenFail {IDStation-inv5}
    {IDStation-inv} WriteUserTokenFail {IDStation-inv6}
    {IDStation-inv} WriteUserTokenFail {IDStation-inv7}
    {IDStation-inv} WriteUserTokenFail {IDStation-inv8}
    {IDStation-inv} WriteUserTokenFail {IDStation-inv10}
    by (hoare-wlp-auto defs: tis-defs)+
  from 1 2 3 show ?thesis
    by (auto simp add: IDStation-inv-def hoare-post-conj-split)
 qed

 have ut: {UserToken ⊕p iuserToken} WriteUserTokenFail {UserToken ⊕p iuserToken}
   by (hoare-wlp-auto defs: tis-defs)

 show ?thesis
   apply (rule-tac IDStation-correct-intro)
   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 (simp add: ut hoare-r-weaken-pre(1) hoare-r-weaken-pre(2))
   apply (simp add: inv)
   done
qed

definition [upred-defs, tis-defs]:
  WriteUserToken = (WriteUserTokenOK ∨ WriteUserTokenFail)

definition [upred-defs, tis-defs]:
  TISWriteUserToken =
  ((UEC(WriteUserToken) ;; UpdateUserToken)
   ∨ UEC(UserTokenTorn ;; ?[&internal:status = ≪waitingUpdateToken≫]))

lemma TISWriteUserToken-correct:
  {IDStation ⊕p tis}TISWriteUserToken{IDStation ⊕p tis}
proof −
```

have *1*: **{***IDStation* $\oplus_p$ *tis***}** *UEC*(*WriteUserToken*) ;; *UpdateUserToken***{***IDStation* $\oplus_p$ *tis***}**
by (*simp add*: *UEC-correct UpdateUserToken-correct WriteUserTokenFail-correct WriteUserTokenOK-correct WriteUserToken-def disj-upred-def hoare-ndet seq-hoare-inv-r-2*)
thus *?thesis*
by (*simp add*: *TISWriteUserToken-def UEC-correct UserTokenTorn-test-correct disj-upred-def hoare-ndet*)
qed

9.6 Validating Entry

definition *UserAllowedEntry* :: *IDStation upred* **where**
[*tis-defs*, *upred-defs*]:
UserAllowedEntry =
U(((∃ *t*∈≪*ValidToken*≫.
≪*goodT*(*t*)≫ = &*iuserToken*:*currentUserToken*
∧ &*doorLatchAlarm*:*currentTime* ∈ &*config*:*entryPeriod* ≪*role* (*privCert t*)≫ ≪*class* (*clearance* (*privCert t*))≫))
∨ (∃ *t*∈≪*TokenWithValidAuth*≫.
≪*goodT*(*t*)≫ = &*iuserToken*:*currentUserToken*
∧ &*doorLatchAlarm*:*currentTime* ∈ &*config*:*entryPeriod* ≪*role* (*the* (*authCert t*))≫ ≪*class* (*clearance* (*the* (*authCert t*)))≫))

definition *EntryOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
EntryOK =
((&*internal*:*status* = ≪*waitingEntry*≫ ∧
&*iuserToken*:*userTokenPresence* = ≪*present*≫ ∧
@*UserAllowedEntry*)
⟶$_r$ *currentDisplay* := ≪*openDoor*≫ ;;
internal:*status* := ≪*waitingRemoveTokenSuccess*≫ ;;
internal:*tokenRemovalTimeout* := &*doorLatchAlarm*:*currentTime* + &*config*:*tokenRemovalDuration*)

lemma *EntryOK-correct*: **{***IDStation***}***EntryOK***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

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

lemma *EntryNotAllowed-correct*: **{***IDStation***}***EntryNotAllowed***{***IDStation***}**

apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

definition [*upred-defs*, *tis-defs*]:
TISValidateEntry =
(*UEC*(*EntryOK*) ∨ *UEC*(*EntryNotAllowed*) ∨ *UEC*(*UserTokenTorn* ;; *?*[&*internal*:*status* = ≪*waitingEntry*≫]))

lemma *TISValidateEntry-correct*: **{***IDStation* ⊕$_p$ *tis***}** *TISValidateEntry***{***IDStation* ⊕$_p$ *tis***}**
by (*simp add*: *EntryNotAllowed-correct EntryOK-correct TISValidateEntry-def UEC-correct UserTokenTorn-test-correct disj-upred-def hoare-ndet*)

9.7 Unlocking the Door

definition *UnlockDoorOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
UnlockDoorOK =
(&*internal*:*status* = ≪*waitingRemoveTokenSuccess*≫ ∧
&*iuserToken*:*userTokenPresence* = ≪*absent*≫)
⟶$_r$ *UnlockDoor* ;; *currentDisplay* := ≪*doorUnlocked*≫ ;; *internal*:*status* := ≪*quiescent*≫

lemma *UnlockDoorOK-correct*: **{***IDStation***}** *UnlockDoorOK***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

lemma *wp-UnlockDoorOK*:
UnlockDoorOK wp (&*doorLatchAlarm*:*currentLatch* = ≪*unlocked*≫) =
U(&*internal*:*status* = ≪*waitingRemoveTokenSuccess*≫ ∧ &*iuserToken*:*userTokenPresence* = ≪*absent*≫)
by (*simp add*: *tis-defs wp usubst unrest*)

definition *WaitingTokenRemoval* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
WaitingTokenRemoval =
?[&*internal*:*status* ∈ {≪*waitingRemoveTokenSuccess*≫, ≪*waitingRemoveTokenFail*≫} ∧
&*internal*:*status* = ≪*waitingRemoveTokenSuccess*≫ ⇒ &*doorLatchAlarm*:*currentTime* < &*internal*:*tokenRemovalTimeout* ∧
&*iuserToken*:*userTokenPresence* = ≪*present*≫]

lemma *WaitingTokenRemoval-correct*:
{*IDStation***}** *WaitingTokenRemoval* ;; *?*[@*b*]**{***IDStation***}**
apply (*rule IDStation-correct-intro*)

apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

definition *TokenRemovalTimeout* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
TokenRemovalTimeout =
((&*internal*:*status* = «*waitingRemoveTokenSuccess*» ∧
&*doorLatchAlarm*:*currentTime* ≥ &*internal*:*tokenRemovalTimeout* ∧
&*iuserToken*:*userTokenPresence* = «*present*») ⟶$_r$
internal:*status* := «*waitingRemoveTokenFail*» ;;
currentDisplay := «*removeToken*»)

lemma *TokenRemovalTimeout-correct*: {*IDStation*} *TokenRemovalTimeout*{*IDStation*}
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

definition [*upred-defs*, *tis-defs*]:
TISUnlockDoor = (*UEC*(*UnlockDoorOK*)
∨ *UEC*(*WaitingTokenRemoval* ;; *?*[&*internal*:*status* = «*waitingRemoveTokenSuccess*»])
∨ *UEC*(*TokenRemovalTimeout*))

lemma *TISUnlockDoor-correct*:
{*IDStation* ⊕$_p$ *tis*} *TISUnlockDoor*{*IDStation* ⊕$_p$ *tis*}
by (*simp add*: *TISUnlockDoor-def TokenRemovalTimeout-correct UEC-correct UnlockDoorOK-correct WaitingTokenRemoval-correct disj-upred-def hoare-ndet*)

9.8 Terminating a Failed Access

definition *FailedAccessTokenRemoved* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
FailedAccessTokenRemoved =
((&*internal*:*status* = «*waitingRemoveTokenFail*» ∧
&*iuserToken*:*userTokenPresence* = «*absent*») ⟶$_r$
internal:*status* := «*quiescent*» ;;
currentDisplay := «*welcom*»)

lemma *FailedAccessTokenRemoved-correct*: {*IDStation*} *FailedAccessTokenRemoved*{*IDStation*}
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

definition [*upred-defs*, *tis-defs*]:
TISCompleteFailedAccess = (*UEC*(*FailedAccessTokenRemoved*)
∨ *UEC*(*WaitingTokenRemoval* ;; *?*[&*internal*:*status* = «*waitingRemoveTo-*

kenFail≫]))

lemma *TISCompleteFailedAccess-correct*:
{*IDStation* $\oplus_p$ *tis***}** *TISCompleteFailedAccess***{***IDStation* $\oplus_p$ *tis***}**
by (*simp add*: *FailedAccessTokenRemoved-correct TISCompleteFailedAccess-def UEC-correct WaitingTokenRemoval-correct disj-upred-def hoare-ndet*)

9.9 The Complete User Entry

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

lemma *TISUserEntryOp-inv*: **{***IDStation* $\oplus_p$ *tis***}** *TISUserEntryOp***{***IDStation* $\oplus_p$ *tis***}**
apply (*auto simp add*: *TISUserEntryOp-def intro*!:*hoare-disj*)
apply (*simp-all add*: *TISReadUserToken-correct TISValidateUserToken-correct*

TISReadFinger-correct TISValidateFinger-correct TISWriteUserToken-correct
TISValidateEntry-correct TISUnlockDoor-correct TISCompleteFailedAccess-correct)
done

10 Operations Within the Enclave (2)

10.1 Enrolment of an ID Station

10.1.1 Requesting Enrolment

definition *RequestEnrolment* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
RequestEnrolment =
(&*internal*:*enclaveStatus* = *notEnrolled* ∧
&*ifloppy*:*floppyPresence* = *absent*)
⟶$_r$ *currentScreen*:*screenMsg* := *insertEnrolmentData* ;; *currentDisplay* := *blank*

lemma *RequestEnrolment-correct*: **{***IDStation***}***RequestEnrolment***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

definition *ReadEnrolmentFloppy* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ReadEnrolmentFloppy =
(&*internal*:*enclaveStatus* = *notEnrolled* ∧
&*ifloppy*:*floppyPresence* = *present*)

$\longrightarrow_r$ *currentScreen*:*screenMsg* := *validatingEnrolmentData* ;; *currentDisplay* := *blank*

lemma *ReadEnrolmentFloppy-correct*: **{***IDStation***}***ReadEnrolmentFloppy***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

definition [*tis-defs*]: *ReadEnrolmentData* = (*ReadEnrolmentFloppy* ∨ *RequestEnrolment*)

10.1.2 Validating Enrolment data from Floppy

definition *EnrolmentDataOK* :: *IDStation upred* **where**
[*tis-defs*]:
EnrolmentDataOK = (*U*(&*ifloppy*:*currentFloppy* ∈ ≪*range enrolmentFile*≫ ∧
enrolmentFile-of &*ifloppy*:*currentFloppy* ∈ ≪*ValidEnrol*≫))

definition *ValidateEnrolmentDataOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ValidateEnrolmentDataOK =
(&*internal*:*enclaveStatus* = *waitingEnrol* ∧
EnrolmentDataOK)
$\longrightarrow_r$ *UpdateKeyStoreFromFloppy* ;;
currentScreen:*screenMsg* := *welcomeAdmin* ;;
internal:*enclaveStatus* := *enclaveQuiescent* ;;
internal:*status* := *quiescent* ;;
currentDisplay := *welcom*

lemma [*simp*]:
⟦ *k* ∈ *ISSUER* $\rightharpoonup_r$ *KEYPART*; *e* ∈ *ValidEnrol* ⟧ ⟹
k +$_r$ {(*subject c*, *subjectPubK c*) | *c*. *c* ∈ *issuerCerts e*} +$_r$ {(*subject* (*tisCert e*), *subjectPubK* (*tisCert e*))}
∈ *ISSUER* $\rightharpoonup_r$ *KEYPART*
apply (*rule rel-pfun-override*)
apply (*rule rel-pfun-override*)
apply (*auto*)
using *Enrol-function* **apply** *blast*
apply (*rule rel-pfun-pair*)
apply (*simp add*: *Enrol-def ValidEnrol-def subset-eq*)
apply *blast*
done

lemma *ValidEnrol-props* [*simp*]:
assumes *e* ∈ *ValidEnrol*
shows *subject* (*tisCert e*) ∈ *ISSUER*

using *assms* **by** (*auto simp add*: *ValidEnrol-def Enrol-def*)

declare *ValidEnrol-def* [*tis-defs del*, *upred-defs del*]

lemma *ValidateEnrolmentDataOK-wf*: **{***IDStation-wf***}** *ValidateEnrolmentDataOK***{***IDStation-wf***}**
apply (*simp add*: *IDStation-wf-def*)
apply (*simp add*: *ValidateEnrolmentDataOK-def*, *hoare-split*)
apply (*simp add*: *tis-defs call-def*, *hoare-auto*)
apply (*simp add*: *tis-defs call-def*, *hoare-auto*)
apply (*rule hoare-r-conseq*[*OF UpdateKeyStoreFromFloppy-IDStation-wf*])
apply (*rel-auto*)
apply (*rel-auto*)
apply (*simp add*: *tis-defs call-def*, *hoare-auto*)
apply (*simp add*: *tis-defs call-def*, *hoare-auto*)
apply (*simp add*: *tis-defs call-def*, *hoare-auto*)
apply (*simp add*: *tis-defs call-def*, *hoare-auto*)
done

lemma *ValidateEnrolmentDataOK-inv*: **{***IDStation-inv***}** *ValidateEnrolmentDataOK***{***IDStation-inv***}**
apply (*rule IDStation-inv-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*simp add*: *IDStation-inv2-def*)
oops

definition *ValidateEnrolmentDataFail* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ValidateEnrolmentDataFail =
(&*internal*:*enclaveStatus* = *waitingEnrol* ∧
¬ *EnrolmentDataOK*)
⟶$_r$ *AddElementsToLog* ;;
currentScreen:*screenMsg* := *enrolmentFailed* ;;
internal:*enclaveStatus* := *waitingEndEnrol* ;;
currentDisplay := *blank*

lemma *ValidateEnrolmentDataFail-correct*: **{***IDStation***}** *ValidateEnrolmentDataFail***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
oops

definition [*tis-defs*]: *ValidateEnrolmentData* = (*ValidateEnrolmentDataOK* ∨ *ValidateEnrolmentDataFail*)

10.1.3 Completing a Failed Enrolment

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

FailedEnrolFloppyRemoved =
(&*internal*:*enclaveStatus* = *waitingEndEnrol* ∧
&*ifloppy*:*floppyPresence* = *absent*)
⟶$_r$ *currentScreen*:*screenMsg* := *insertEnrolmentData* ;;
internal:*enclaveStatus* := *notEnrolled* ;;
currentDisplay := *blank*

definition *WaitingFloppyRemoval* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
WaitingFloppyRemoval =
(&*internal*:*enclaveStatus* = *waitingEndEnrol* ∧
&*ifloppy*:*floppyPresence* = *present*)
⟶$_r$ *II*

definition [*tis-defs*]: *CompleteFailedEnrolment* = (*FailedEnrolFloppyRemoved* ∨ *WaitingFloppyRemoval*)

10.1.4 The Complete Enrolment

definition *TISEnrolOp* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISEnrolOp = *UEC*(*ReadEnrolmentData* ∨ *ValidateEnrolmentData* ∨ *Complete-FailedEnrolment*)

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*≫

$) \longrightarrow_r admin{:}rolePresent := \ll None \gg ;; admin{:}currentAdminOp := \ll None \gg ;;$
$admin{:}availableOps := \{\})$

definition *AdminStartOp* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$AdminStartOp =$
$((\&admin{:}rolePresent \neq \ll None \gg$
$\wedge\ \&admin{:}currentAdminOp = \ll None \gg$
$\wedge\ \&ikeyboard{:}currentKeyedData \in \ll keyedOps \gg\ `\ \&admin{:}availableOps$
$) \longrightarrow_r admin{:}currentAdminOp := Some(ofKeyedOps(\&ikeyboard{:}currentKeyedData)))$

definition *AdminFinishOp* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$AdminFinishOp =$
$((\&admin{:}rolePresent \neq \ll None \gg$
$\wedge\ \&admin{:}currentAdminOp \neq \ll None \gg$
$) \longrightarrow_r admin{:}currentAdminOp := \ll None \gg)$

definition *AdminTokenTear* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$AdminTokenTear =$
$((\&iadminToken{:}adminTokenPresence = \ll absent \gg$
$) \longrightarrow_r internal{:}enclaveStatus := \ll enclaveQuiescent \gg)$

definition *BadAdminTokenTear* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$BadAdminTokenTear =$
$((\&internal{:}enclaveStatus \in \{\ll gotAdminToken \gg, \ll waitingStartAdminOp \gg, \ll waitingFinishAdminOp \gg\})$
$\longrightarrow_r AdminTokenTear)$

definition *BadAdminLogout* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$BadAdminLogout =$
$((\&internal{:}enclaveStatus \in \{\ll waitingStartAdminOp \gg, \ll waitingFinishAdminOp \gg\}$
$) \longrightarrow_r (BadAdminTokenTear\ ;;\ AdminLogout))$

definition *LoginAborted* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$LoginAborted = ((\&internal{:}enclaveStatus = \ll gotAdminToken \gg) \longrightarrow_r BadAdminTokenTear)$

lemma *LoginAborted-correct*:
{*IDStation***}***LoginAborted***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*rule IDStation-inv-intro*)
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 *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*»)

lemma *ReadAdminToken-correct*:
{*IDStation***}***ReadAdminToken***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*rule IDStation-inv-intro*)
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 *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* := «*requestAdminOp*» ;;
internal:*enclaveStatus* := «*enclaveQuiescent*»)

lemma *ValidateAdminTokenOK-correct*:
{*IDStation***}** *ValidateAdminTokenOK***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*rule IDStation-inv-intro*)
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 *ValidateAdminTokenFail* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ValidateAdminTokenFail =
((&*internal*:*enclaveStatus* = «*gotAdminToken*»
∧ &*iadminToken*:*adminTokenPresence* = «*present*»
∧ (¬ @*AdminTokenOK*)
) ⟶$_r$ *currentScreen*:*screenMsg* := «*removeAdminToken*» ;;
internal:*enclaveStatus* := «*waitingRemoveAdminTokenFail*»)

lemma *ValidateAdminTokenFail-correct*:
{*IDStation***}** *ValidateAdminTokenFail***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
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*)
done

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

lemma *TISValidateAdminToken-correct*:
{*IDStation* $\oplus_p$ *tis***}** *TISValidateAdminToken***{***IDStation* $\oplus_p$ *tis***}**
by (*simp add*: *LoginAborted-correct TISValidateAdminToken-def UEC-correct ValidateAdminTokenFail-correct ValidateAdminTokenOK-correct disj-upred-def hoare-ndet*)

definition *FailedAdminTokenRemove* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
FailedAdminTokenRemove =
((&*internal*:*enclaveStatus* = «*waitingRemoveAdminTokenFail*»
∧ &*iadminToken*:*adminTokenPresence* = «*absent*»
) $\longrightarrow_r$ *currentScreen*:*screenMsg* := «*welcomeAdmin*» ;;
internal:*enclaveStatus* := «*enclaveQuiescent*»)

lemma *FailedAdminTokenRemove-correct*:
{*IDStation***}***FailedAdminTokenRemove***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*rule IDStation-inv-intro*)
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 *WaitingAdminTokenRemoval* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
WaitingAdminTokenRemoval =
((&*internal*:*enclaveStatus* = «*waitingRemoveAdminTokenFail*»
∧ &*iadminToken*:*adminTokenPresence* = «*present*») $\longrightarrow_r$ *II*)

lemma *WaitingAdminTokenRemoval-correct*:
{*IDStation***}***WaitingAdminTokenRemoval***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*rule IDStation-inv-intro*)
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 *TISCompleteFailedAdminLogon* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISCompleteFailedAdminLogon = (*UEC*(*FailedAdminTokenRemove*) ∨ *UEC*(*WaitingAdminTokenRemoval*))

lemma *TISCompleteFailedAdminLogon-correct*:
{*IDStation* ⊕$_p$ *tis***}** *TISCompleteFailedAdminLogon***{***IDStation* ⊕$_p$ *tis***}**
by (*simp add*: *FailedAdminTokenRemove-correct TISCompleteFailedAdminLogon-def UEC-correct WaitingAdminTokenRemoval-correct disj-upred-def hoare-ndet*)

definition [*upred-defs*, *tis-defs*]:
TISAdminLogon = (*TISReadAdminToken* ∨ *TISValidateAdminToken* ∨ *TISCompleteFailedAdminLogon*)

lemma *TISAdminLogon-correct*:
{*IDStation* ⊕$_p$ *tis***}** *TISAdminLogon***{***IDStation* ⊕$_p$ *tis***}**
by (*simp add*: *ReadAdminToken-correct TISAdminLogon-def TISCompleteFailedAdminLogon-correct TISReadAdminToken-def TISValidateAdminToken-correct UEC-correct disj-upred-def hoare-ndet*)

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

definition *ValidateOpRequestOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ValidateOpRequestOK =
((&*ikeyboard*:*keyedDataPresence* = «*present*» ∧
&*ikeyboard*:*currentKeyedData* ∈ «*keyedOps*» ‘ &*admin*:*availableOps*)
⟶$_r$ *StartOpContext* ;;
AdminStartOp ;;
currentScreen:*screenMsg* := «*doingOp*» ;;
internal:*enclaveStatus* := «*waitingStartAdminOp*»)

lemma *ValidateOpRequestOK-correct*:
{*IDStation***}** *ValidateOpRequestOK***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*rule IDStation-inv-intro*)
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 ValidateOpRequestFail :: IDStation hrel where
[upred-defs, tis-defs]:
ValidateOpRequestFail =
 ((&ikeyboard:keyedDataPresence = «present» ∧
   &ikeyboard:currentKeyedData ∉ «keyedOps»‘&admin:availableOps)
   ⟶r StartOpContext ;;
       currentScreen:screenMsg := «invalidRequest»)

lemma ValidateOpRequestFail-correct:
 {IDStation}ValidateOpRequestFail{IDStation}
 apply (rule IDStation-correct-intro)
  apply (hoare-wlp-auto defs: tis-defs)
 apply (rule IDStation-inv-intro)
         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 NoOpRequest :: IDStation hrel where
[upred-defs, tis-defs]:
NoOpRequest =
 ((&ikeyboard:keyedDataPresence = «absent») ⟶r StartOpContext)

lemma NoOpRequest-correct:
 {IDStation}NoOpRequest{IDStation}
 apply (rule IDStation-correct-intro)
  apply (hoare-wlp-auto defs: tis-defs)
 apply (rule IDStation-inv-intro)
         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 [*upred-defs*, *tis-defs*]:
ValidateOpRequest = (*ValidateOpRequestOK* ∨ *ValidateOpRequestFail* ∨ *NoOpRequest*)

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

lemma *TISStartAdminOp-correct*:
{*IDStation* ⊕$_p$ *tis***}** *TISStartAdminOp***{***IDStation* ⊕$_p$ *tis***}**
by (*simp add*: *NoOpRequest-correct TISStartAdminOp-def UEC-correct ValidateOpRequestFail-correct ValidateOpRequestOK-correct ValidateOpRequest-def disj-upred-def hoare-ndet*)

definition *AdminOpStartedContext* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
AdminOpStartedContext =
U(&*internal*:*enclaveStatus* = *waitingStartAdminOp* ∧ &*iadminToken*:*adminTokenPresence* = *present*)

definition *AdminOpFinishContext* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
AdminOpFinishContext =
(&*internal*:*enclaveStatus* = *waitingFinishAdminOp* ∧ &*iadminToken*:*adminTokenPresence* = *present*)
⟶$_r$ *internal*:*enclaveStatus* := *enclaveQuiescent* ;; *admin*:*currentAdminOp* := *None*

definition *ShutdownOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ShutdownOK =
((&*internal*:*enclaveStatus* = ≪*waitingStartAdminOp*≫
∧ &*admin*:*currentAdminOp* = ≪*Some*(*shutdownOp*)≫
∧ &*doorLatchAlarm*:*currentDoor* = ≪*closed*≫
) ⟶$_r$ *LockDoor* ;;
AdminLogout ;;
currentScreen:*screenMsg* := ≪*clear*≫ ;;
internal:*enclaveStatus* := ≪*shutdown*≫ ;;
currentDisplay := ≪*blank*≫
)

lemma *ShutdownOK-correct*:
{*IDStation***}***ShutdownOK***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)

apply (*rule IDStation-inv-intro*)
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 *ShutdownWaitingDoor* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ShutdownWaitingDoor =
((&*internal*:*enclaveStatus* = «*waitingStartAdminOp*»
∧ &*admin*:*currentAdminOp* = «*Some*(*shutdownOp*)»
∧ &*doorLatchAlarm*:*currentDoor* = «*dopen*»
) ⟶$_r$ *currentScreen*:*screenMsg* := «*closeDoor*»
)

lemma *ShutdownWaitingDoor-correct*:
{*IDStation***}***ShutdownWaitingDoor***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*rule IDStation-inv-intro*)
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 *TISShutdownOp* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISShutdownOp = (*UEC*(*ShutdownOK*) ∨ *UEC*(*ShutdownWaitingDoor*))

lemma *TISShutdownOp-correct*:
{*IDStation* ⊕$_p$ *tis***}** *TISShutdownOp***{***IDStation* ⊕$_p$ *tis***}**
by (*simp add*: *ShutdownOK-correct ShutdownWaitingDoor-correct TISShutdownOp-def UEC-correct disj-upred-def hoare-ndet*)

definition *OverrideDoorLockOK* :: *IDStation hrel* **where**

[*upred-defs*, *tis-defs*]:
OverrideDoorLockOK =
((*AdminOpStartedContext* ∧ &*admin*:*currentAdminOp* = ≪*Some*(*overrideLock*)≫
) ⟶$_r$ *currentScreen*:*screenMsg* := *requestAdminOp* ;;
currentDisplay := *doorUnlocked* ;;
internal:*enclaveStatus* := *enclaveQuiescent* ;;
UnlockDoor ;;
AdminFinishOp)

lemma *OverrideDoorLockOK-correct*:
{*IDStation***}***OverrideDoorLockOK***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*rule IDStation-inv-intro*)
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 *TISOverrideDoorLockOp* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISOverrideDoorLockOp =
(*UEC*(*OverrideDoorLockOK*)
∨ *UEC*((&*internal*:*enclaveStatus* = ≪*waitingStartAdminOp*≫
∧ &*admin*:*currentAdminOp* = ≪*Some*(*overrideLock*)≫) ⟶$_r$ *BadAdminLogout*))

lemma *TISOverrideDoorLockOp-correct*:
{*IDStation* ⊕$_p$ *tis***}***TISOverrideDoorLockOp***{***IDStation* ⊕$_p$ *tis***}**
apply (*simp add*: *TISOverrideDoorLockOp-def*)
apply (*rule hoare-disj*)
using *OverrideDoorLockOK-correct UEC-correct* **apply** *blast*
apply (*rule UEC-correct*)
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*rule IDStation-inv-intro*)
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 StartArchiveLogOK :: IDStation hrel where
[upred-defs, tis-defs]:
StartArchiveLogOK =
  (AdminOpStartedContext ∧ &admin:currentAdminOp = «Some(archiveLog)»
  ∧ &ifloppy:floppyPresence = present) ⟶r
  currentScreen:screenMsg := doingOp ;;
  internal:enclaveStatus := waitingFinishAdminOp ;;
  (⨅ archive :: Audit set ·
    ArchiveLog archive ;;
    ifloppy:writtenFloppy := «auditFile archive»)

definition StartArchiveLogWaitingFloppy :: IDStation hrel where
[upred-defs, tis-defs]:
StartArchiveLogWaitingFloppy =
  (AdminOpStartedContext ∧ &admin:currentAdminOp = «Some(archiveLog)»
  ∧ &ifloppy:floppyPresence = absent) ⟶r
    currentScreen:screenMsg := insertBlankFloppy

definition StartArchiveLog :: SystemState hrel where
[upred-defs, tis-defs]:
  StartArchiveLog = ((tis:[StartArchiveLogOK]+ ;; UpdateFloppy)
                   ∨ tis:[StartArchiveLogWaitingFloppy]+
                   ∨ tis:[BadAdminLogout ;;
                         ?[&internal:enclaveStatus = waitingStartAdminOp
                          ∧ &admin:currentAdminOp = «Some(archiveLog)»]]+)

definition FinishArchiveLogOK :: IDStation hrel where
[upred-defs, tis-defs]:
FinishArchiveLogOK =
  AdminOpFinishContext ;;
  ((&admin:currentAdminOp = «Some(archiveLog)»
  ∧ &ifloppy:floppyPresence = present
  ∧ &ifloppy:writtenFloppy = &ifloppy:currentFloppy)
    ⟶r (⨅ archive :: Audit set ·
          ClearLogThenAddElements archive ;;
          ifloppy:writtenFloppy := «auditFile archive»
        ) ;;
        currentScreen:screenMsg := requestAdminOp)

definition FinishArchiveLogNoFloppy :: IDStation hrel where
[upred-defs, tis-defs]:
FinishArchiveLogNoFloppy =
  AdminOpFinishContext ;;
```

```
  ((&admin:currentAdminOp = «Some(archiveLog)»
  ∧ &ifloppy:floppyPresence = absent)
    ⟶r AddElementsToLog ;; currentScreen:screenMsg := archiveFailed)

definition FinishArchiveLogBadMatch :: IDStation hrel where
[upred-defs, tis-defs]:
FinishArchiveLogBadMatch =
  AdminOpFinishContext ;;
  ((&admin:currentAdminOp = «Some(archiveLog)»
  ∧ &ifloppy:floppyPresence = present
  ∧ &ifloppy:writtenFloppy ≠ &ifloppy:currentFloppy)
    ⟶r AddElementsToLog ;; currentScreen:screenMsg := archiveFailed)

abbreviation FinishArchiveLogFail ≡ FinishArchiveLogBadMatch ∨ FinishArchiveLogNoFloppy

definition FinishArchiveLog :: IDStation hrel where
[upred-defs, tis-defs]:
FinishArchiveLog = (FinishArchiveLogOK ∨ FinishArchiveLogFail
            ∨ BadAdminLogout ;; ?[&internal:enclaveStatus = waitingFinishAdminOp
                                 ∧ &admin:currentAdminOp = «Some(archiveLog)»])

abbreviation TISArchiveLogOp ≡ StartArchiveLog ∨ UEC(FinishArchiveLog)

definition StartUpdateConfigOK :: IDStation hrel where
[upred-defs, tis-defs]:
StartUpdateConfigOK =
 (AdminOpStartedContext ∧ &admin:currentAdminOp = «Some(updateConfigData)»
 ∧ &ifloppy:floppyPresence = present) ⟶r
   currentScreen:screenMsg := doingOp ;;
   internal:enclaveStatus := waitingFinishAdminOp

lemma StartUpdateConfigOK-correct:
  {IDStation}StartUpdateConfigOK{IDStation}
  apply (rule IDStation-correct-intro)
   apply (hoare-wlp-auto defs: tis-defs)
  apply (rule IDStation-inv-intro)
          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 StartUpdateConfigWaitingFloppy :: IDStation hrel where
```

[*upred-defs*, *tis-defs*]:
StartUpdateConfigWaitingFloppy =
(*AdminOpStartedContext* ∧ &*admin*:*currentAdminOp* = «*Some*(*updateConfigData*)»
∧ &*ifloppy*:*floppyPresence* = *absent*) ⟶$_r$
currentScreen:*screenMsg* := *insertConfigData*

lemma *StartUpdateConfigWaitingFloppy-correct*:
{*IDStation***}***StartUpdateConfigWaitingFloppy***{***IDStation***}**
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*rule IDStation-inv-intro*)
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 *StartUpdateConfigData* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
StartUpdateConfigData ≡ *StartUpdateConfigOK* ∨ *StartUpdateConfigWaitingFloppy*
∨ *?*[&*internal*:*enclaveStatus* = *waitingStartAdminOp* ∧
&*admin*:*currentAdminOp* = «*Some*(*updateConfigData*)»] ;;
BadAdminLogout

lemma *StartUpdateConfigData-correct*:
{*IDStation***}***StartUpdateConfigData***{***IDStation***}**
apply (*simp add*: *StartUpdateConfigData-def*)
apply (*rule hoare-disj*)
apply (*simp add*: *StartUpdateConfigOK-correct*)
apply (*rule hoare-disj*)
apply (*simp add*: *StartUpdateConfigWaitingFloppy-correct*)
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*rule IDStation-inv-intro*)
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 FinishUpdateConfigDataOK :: IDStation hrel where
[upred-defs, tis-defs]:
FinishUpdateConfigDataOK =
  AdminOpFinishContext ;;
  ((&admin:currentAdminOp = «Some(updateConfigData)»
  ∧ &ifloppy:floppyPresence = present — Can this we secured via an invariant?
  ∧ &ifloppy:currentFloppy ∈ «range(configFile)»
  ∧ (Config ⊕p config)⟦U(configFile-of &ifloppy:currentFloppy)/&config⟧
  ) ⟶r
    config := configFile-of &ifloppy:currentFloppy ;;
    currentScreen:screenMsg := requestAdminOp ;;
    — The lines are added to preserve the invariants
    currentScreen:screenConfig := displayConfigData config
    )

lemma FinishUpdateConfigDataOK-correct:
  {IDStation}FinishUpdateConfigDataOK{IDStation}
  apply (rule IDStation-correct-intro)
        apply (hoare-wlp-auto defs: tis-defs)
  apply (rule IDStation-inv-intro)
          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 FinishUpdateConfigDataFail :: IDStation hrel where
[upred-defs, tis-defs]:
FinishUpdateConfigDataFail =
  AdminOpFinishContext ;;
  ((&admin:currentAdminOp = «Some(updateConfigData)»
  ∧ &ifloppy:currentFloppy ∉ «range(configFile)»
  ) ⟶r
    currentScreen:screenMsg := invalidData ;;
    AddElementsToLog)

lemma FinishUpdateConfigDataFail-correct:
  {IDStation}FinishUpdateConfigDataFail{IDStation}
  apply (rule IDStation-correct-intro)
   apply (hoare-wlp-auto defs: tis-defs)
```

apply (*rule IDStation-inv-intro*)
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
[*upred-defs*, *tis-defs*]:
FinishUpdateConfigData ≡ *FinishUpdateConfigDataOK* ∨ *FinishUpdateConfigDataFail*
∨ *?*[&*internal*:*enclaveStatus* = *waitingFinishAdminOp*
∧ &*admin*:*currentAdminOp* = ≪*Some*(*updateConfigData*)≫]
;; *BadAdminLogout*

lemma *FinishUpdateConfigData-correct*:
{*IDStation***}***FinishUpdateConfigData***{***IDStation***}**
apply (*simp add*: *FinishUpdateConfigData-def*)
apply (*rule hoare-disj*)
apply (*simp add*: *FinishUpdateConfigDataOK-correct*)
apply (*rule hoare-disj*)
using *FinishUpdateConfigDataFail-correct* **apply** *blast*
apply (*rule IDStation-correct-intro*)
apply (*hoare-wlp-auto defs*: *tis-defs*)
apply (*rule IDStation-inv-intro*)
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 *TISUpdateConfigDataOp* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]: *TISUpdateConfigDataOp* ≡ *UEC*(*StartUpdateConfigData* ∨
FinishUpdateConfigData)

lemma *TISUpdateConfigDataOp-correct*:
{*IDStation* ⊕$_p$ *tis***}***TISUpdateConfigDataOp***{***IDStation* ⊕$_p$ *tis***}**
by (*simp add*: *FinishUpdateConfigData-correct StartUpdateConfigData-correct*

TISUpdateConfigDataOp-def UEC-correct disj-upred-def hoare-ndet)

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

definition *TISAdminOp* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISAdminOp = (*TISOverrideDoorLockOp* ∨ *TISShutdownOp* ∨ *TISUpdateConfigDataOp* ∨ *TISArchiveLog*)

lemma *TISAdminOp-correct*:
{*IDStation* ⊕$_p$ *tis***}** *TISAdminOp***{***IDStation* ⊕$_p$ *tis***}**
by (*simp add*: *TISAdminOp-def TISArchiveLog-def TISOverrideDoorLockOp-correct TISShutdownOp-correct TISUpdateConfigDataOp-correct disj-upred-def hoare-ndet*)

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 Initial System and Startup

definition *InitDoorLatchAlarm* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
InitDoorLatchAlarm =
doorLatchAlarm:[
currentTime := *zeroTime* ;;
currentDoor := *closed* ;;
currentLatch := *locked* ;;
doorAlarm := *silent* ;;
latchTimeout := *zeroTime* ;;
alarmTimeout := *zeroTime*]$^+$

definition *InitKeyStore* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
InitKeyStore =
keyStore:[
issuerKey := {} ;;
ownName := *None*
]$^+$

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

```
  config:[
    alarmSilentDuration := 10 ;;
    latchUnlockDuration := 150 ;;
    tokenRemovalDuration := 100 ;;
    enclaveClearance := (| class = unmarked |) ;;
    authPeriod := «λ p t. {t..t + 72000}» ;;
    entryPeriod := «λ p c. UNIV» ;;
    minPreservedLogSize := * ;;
    alarmThresholdSize := * ;;
    ?[Config]
  ]+

definition InitAdmin :: IDStation hrel where
[upred-defs, tis-defs]:
InitAdmin =
  admin:[
    rolePresent := None ;;
    currentAdminOp := None ;;
    availableOps := {}
  ]+

definition InitStats :: IDStation hrel where
[upred-defs, tis-defs]:
InitStats =
  stats:[
    successEntry := 0 ;;
    failEntry := 0 ;;
    successBio := 0 ;;
    failBio := 0
  ]+

definition InitAuditLog :: IDStation hrel where
[upred-defs, tis-defs]:
InitAuditLog =
  audit:[
    auditLog := {} ;;
    auditAlarm := silent
  ]+

definition InitIDStation :: IDStation hrel where
[upred-defs, tis-defs]:
InitIDStation =
  InitDoorLatchAlarm ;;
  InitConfig ;;
  InitKeyStore ;;
  InitStats ;;
  InitAuditLog ;;
  InitAdmin ;;
  currentScreen := * ;;
```

currentScreen:*screenMsg* := *clear* ;;
currentDisplay := *blank* ;;
internal:*enclaveStatus* := *notEnrolled* ;;
internal:*status* := *quiescent* ;;
— We select arbitrary values for the monitored variables
iuserToken := ∗ ;;
?[*UserToken* $\oplus_p$ *iuserToken*] ;;
iadminToken := ∗ ;;
?[*AdminToken* $\oplus_p$ *iadminToken*] ;;
ifinger := ∗ ;;
?[*Finger* $\oplus_p$ *ifinger*] ;;
ifloppy := ∗ ;;
?[*Floppy* $\oplus_p$ *ifloppy*] ;;
ikeyboard := ∗ ;;
?[*Keyboard* $\oplus_p$ *ikeyboard*]

lemma {*true*}*InitIDStation*{*IDStation*}
apply (*simp add*: *tis-defs frame assigns-comp usubst alpha*)
oops

abbreviation *TISInit* ≡ *tis*:[*InitIDStation*]$^+$

12 The Whole ID Station

definition *TISOp* :: *SystemState hrel* **where**
[*tis-defs*]:
TISOp = ((*TISEnrolOp*
∨ *TISUserEntryOp*
∨ *TISAdminLogon*
∨ *TISStartAdminOp*
∨ *TISAdminOp*
∨ *TISAdminLogout*
∨ *TISIdle*))

abbreviation *TISOpThenUpdate* ≡ *TISOp* ;; *TISUpdate*

abbreviation *TISBody* ≡ *TISPoll* ;; *TISEarlyUpdate* ;; *TISOp* ;; *TISUpdate*

abbreviation *TIS* ≡ *TISInit* ;; *TISBody**

13 Proving Security Properties

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

lemma
([&*tis*:*doorLatchAlarm*:*currentLatch* $\mapsto_s$ ≪*locked*≫] †

(*TISReadUserToken wp* (&*tis*:*doorLatchAlarm*:*currentLatch* = «*unlocked*»)))
= *false*
by (*simp add*: *tis-defs wp usubst unrest alpha*)

13.1 Proving Security Functional Requirement 1

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

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

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.

SFR1(b): 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.

abbreviation (*input*) *FSFR1* ≡
&*tis*:*doorLatchAlarm*:*currentLatch* = *locked* ∧ *IDStation* ⊕$_p$ *tis* ⊩
(*TISOp* ;; *TISUpdate*) *wp* (&*realWorld*:*controlled*:*latch* = *unlocked*) ⇒
(((*UserTokenOK* ∧ *FingerOK*) ∨ *UserTokenWithOKAuthCert*) ∨ *AdminTokenGuardOK*) ⊕$_p$ *tis*

declare *subst-lit-aext* [*usubst del*]

lemma *admin-unlock*:
[&*tis*:*doorLatchAlarm*:*currentLatch* ↦$_s$ «*locked*»]
† ((*TISAdminOp* ;; *TISUpdate*) *wp* (&*realWorld*:*controlled*:*latch* = «*unlocked*»)) =
U((&*tis*:*internal*:*enclaveStatus* = «*waitingStartAdminOp*» ∧ &*tis*:*iadminToken*:*adminTokenPresence* = «*present*») ∧
&*tis*:*admin*:*currentAdminOp* = «*Some overrideLock*» ∧ &*tis*:*admin*:*rolePresent* ≠ *None* ∧ &*tis*:*admin*:*currentAdminOp* ≠ *None*)
by (*simp add*: *tis-defs wp usubst unrest alpha*, *rel-auto*)

declare *subst-lit-aext* [*usubst*]

lemma *user-unlock*:
[&*tis*:*doorLatchAlarm*:*currentLatch* ↦$_s$ «*locked*»]

```
        † ((TISUserEntryOp ;; TISUpdate) wp (&realWorld:controlled:latch =
«unlocked»)) =
  U(&tis:internal:status = «waitingRemoveTokenSuccess» ∧ &tis:iuserToken:userTokenPresence
= «absent»)
  by (simp add: tis-defs alpha unrest usubst wp)

lemma unlock:
  [&tis:doorLatchAlarm:currentLatch ↦s «locked»]
      † ((TISOpThenUpdate) wp (&realWorld:controlled:latch = «unlocked»)) =
U(&tis:internal:status = waitingRemoveTokenSuccess ∧ &tis:iuserToken:userTokenPresence
= absent ∨
      ((&tis:internal:enclaveStatus = waitingStartAdminOp ∧ &tis:iadminToken:adminTokenPresence
= present) ∧
        &tis:admin:currentAdminOp = Some overrideLock ∧
          (&tis:admin:rolePresent) ≠ None ∧ (&tis:admin:currentAdminOp) ≠
None))
 apply (simp add: TISOp-def seqr-or-distl wp-disj user-unlock usubst admin-unlock)
  apply (simp add: tis-defs alpha unrest usubst wp call-def)
  apply (rel-auto)
  done

lemma FSFR1-proof: FSFR1
  apply (rule sVarEqI)
   apply (simp-all add: usubst unrest unlock)
  apply (rule sImplI)
  apply (simp add: distrib(4))
  apply (rule sAsmDisj)
   apply (simp add: aext-or)
   apply (rule sDisjI1)
   apply (rule sWk[where P=U(&tis:internal:status = waitingRemoveTokenSuc-
cess ∧ &tis:iuserToken:userTokenPresence = absent) ∧
   U(&tis:doorLatchAlarm:currentLatch = locked) ∧ (IDStation-inv1 ∧ IDStation-inv9)
⊕p tis])
    apply (rel-auto)
   apply (rel-auto)
  apply (rule sWk[where P=(U((&tis:internal:enclaveStatus = waitingStartAd-
minOp ∧ &tis:iadminToken:adminTokenPresence = present) ∧
        &tis:admin:currentAdminOp = Some overrideLock ∧
          (&tis:admin:rolePresent) ≠ None ∧ (&tis:admin:currentAdminOp) ≠
None) ∧ (IDStation-inv2 ∧ (Admin ⊕p admin) ∧ IDStation-inv10) ⊕p tis)])
   apply (rel-simp')
  apply (force)
  apply (rel-auto)
  done

abbreviation IDStation-inv11 ≡
  U(&internal:status = «waitingRemoveTokenSuccess» ⇒
    (∃ t. «goodT(t)» = &iuserToken:currentUserToken
        ∧ (&doorLatchAlarm:currentTime ∈ &config:entryPeriod «role (privCert
```

$t) \gg \ll class\ (clearance\ (privCert\ t)) \gg)$
$\quad \vee\ \&doorLatchAlarm{:}currentTime \in \&config{:}entryPeriod \ll role\ (the$
$(authCert\ t)) \gg \ll class\ (clearance\ (the\ (authCert\ t))) \gg))$
lemma $\{IDStation\text{-}inv11\}\, UnlockDoorOK \{IDStation\text{-}inv11\}$
by (*hoare-wlp-auto*)

lemma $\{IDStation\text{-}inv11\}\, WaitingTokenRemoval \{IDStation\text{-}inv11\}$
by (*hoare-wlp-auto*)

lemma $\{IDStation\text{-}inv11\}\, TokenRemovalTimeout \{IDStation\text{-}inv11\}$
by (*hoare-wlp-auto*)

lemma $\{IDStation\text{-}inv11\}\, EntryOK \{IDStation\text{-}inv11\}$
by (*hoare-wlp-auto*)

lemma $\{IDStation\text{-}inv11 \oplus_p tis\}\, TISPoll\ ;;\ TISUnlockDoor \{IDStation\text{-}inv11 \oplus_p tis\}$
apply (*simp add*: *wlp-hoare-link wp unrest usubst tis-defs*)
oops

lemma $(TISReadUserToken\ wp\ (\&tis{:}audit{:}auditAlarm = alarming))[\![\ll silent\gg/\&tis{:}audit{:}auditAlarm]\!]$
$= false$
by (*simp add*: *tis-defs alpha unrest usubst wp*)

lemma $(AddElementsToLog\ wp\ (\&audit{:}auditAlarm = alarming))[\![\ll silent\gg/\&audit{:}auditAlarm]\!]$
$= undefined$
apply (*simp add*: *tis-defs alpha unrest usubst wp*)
oops

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

abbreviation $FSFR3 \equiv$
$\quad `IDStation \Rightarrow$
$\quad\quad U(\&doorLatchAlarm{:}currentLatch = locked$
$\quad\quad \wedge\ \&doorLatchAlarm{:}currentDoor = dopen$
$\quad\quad \wedge\ \&doorLatchAlarm{:}currentTime \geq \&doorLatchAlarm{:}alarmTimeout$
$\quad\quad \Rightarrow \&doorLatchAlarm{:}doorAlarm = alarming)`$

lemma *FSFR3-proof*: *FSFR3*
by *rel-auto*

lemma *nmods-UEC* [*closure*]: $[\![\ vwb\text{-}lens\ a;\ P\ nmods\ a\ ]\!] \Longrightarrow UEC(P)\ nmods$
$\&tis{:}a$
by (*simp add*: *UEC-def closure*)

declare *nmods-assigns* [*closure del*]

lemma *TISUserEntryOp-nmods-config*: *TISUserEntryOp nmods &tis:config*
by (*simp add*: *tis-defs closure del*: *UEC-def*)

lemma *TISEnrolOp-nmods-config*: *TISEnrolOp nmods &tis:config*
by (*simp add*: *tis-defs closure del*: *UEC-def*)

lemma *TISAdminLogon-nmods-config*: *TISAdminLogon nmods &tis:config*
by (*simp add*: *tis-defs closure del*: *UEC-def*)

lemma *TISStartAdminOp-nmods-config*: *TISStartAdminOp nmods &tis:config*
by (*simp add*: *tis-defs closure del*: *UEC-def*)

lemma *StartUpdateConfigData-nmods-config*:
StartUpdateConfigData nmods config
by (*simp add*: *tis-defs closure*)

lemma *FinishUpdateConfigOK-absent-nmods-config*:
?[*&iadminToken:adminTokenPresence = absent*] ;; *FinishUpdateConfigDataOK nmods config*
proof −
have *?*[*&iadminToken:adminTokenPresence = absent*] ;; *AdminOpFinishContext = false*
by *rel-simp′*
hence *?*[*&iadminToken:adminTokenPresence = absent*] ;; *FinishUpdateConfigDataOK = false*
by (*simp add*: *FinishUpdateConfigDataOK-def RA1*)
thus *?thesis*
by (*metis assume-false config-vwb-lens nmods-guard*)
qed

lemma *FinishUpdateConfigDataFail-nmods-config*:
FinishUpdateConfigDataFail nmods config
by (*simp add*: *tis-defs closure*)

lemma *BadAdminLogout-nmods-config*:
BadAdminLogout nmods config
by (*simp add*: *tis-defs closure*)

lemma *FinishUpdateConfigData-absent-nmods-config*:
(*?*[*&iadminToken:adminTokenPresence = absent*] ;; *FinishUpdateConfigData*) *nmods config*
by (*simp add*: *FinishUpdateConfigData-def seqr-or-distr closure BadAdminLogout-nmods-config FinishUpdateConfigDataFail-nmods-config FinishUpdateConfigOK-absent-nmods-config*)

lemma *frext-guard* [*frame*]: *vwb-lens a* ⟹ *a*:[*?*[*b*]]$^+$ = *?*[*b* ⊕$_p$ *a*]
by (*rel-auto*)

lemma *TISUpdateConfigDataOp-absent-nmods-config*:
?[&*tis*:*iadminToken*:*adminTokenPresence* = *absent*] ;; *TISUpdateConfigDataOp nmods* &*tis*:*config*
proof −
have ⋀ *P*. *?*[&*tis*:*iadminToken*:*adminTokenPresence* = *absent*] ;; *UEC*(*P*) = *UEC*(*?*[&*iadminToken*:*adminTokenPresence* = *absent*] ;; *P*)
by (*simp add*: *UEC-def seq-UINF-distl*′ *frame alpha seqr-assoc*)
thus *?thesis*
by (*simp add*: *TISUpdateConfigDataOp-def seqr-or-distr closure FinishUpdateConfigData-absent-nmods-config StartUpdateConfigData-nmods-config*)
qed

lemma *TISOverrideDoorLockOp-nmods-config*: *TISOverrideDoorLockOp nmods* &*tis*:*config*
by (*simp add*: *tis-defs closure del*: *UEC-def*)

lemma *TISShutdownOp-nmods-config*: *TISShutdownOp nmods* &*tis*:*config*
by (*simp add*: *tis-defs closure del*: *UEC-def*)

lemma *TISArchiveLog-nmods-config*: *TISArchiveLog nmods* &*tis*:*config*
by (*simp add*: *tis-defs closure del*: *UEC-def*)

lemma *TISAdminOp-absent-nmods-config*:
?[&*tis*:*iadminToken*:*adminTokenPresence* = *absent*] ;; *TISAdminOp nmods* &*tis*:*config*
by (*simp add*: *TISAdminOp-def seqr-or-distr TISUpdateConfigDataOp-absent-nmods-config closure TISOverrideDoorLockOp-nmods-config TISShutdownOp-nmods-config TISArchiveLog-nmods-config*)

lemma *TISAdminLogout-nmods-config*: *TISAdminLogout nmods* &*tis*:*config*
by (*simp add*: *tis-defs closure del*: *UEC-def*)

lemma *TISIdle-nmods-config*: *TISIdle nmods* &*tis*:*config*
by (*simp add*: *tis-defs closure del*: *UEC-def*)

lemma *TISOp-nmods-floppy*: *TISOp nmods* &*tis*:*ifloppy*
by (*simp add*: *tis-defs closure del*: *UEC-def*)

term *in-var*

lemma *nmods-union* [*closure*]: ⟦ *P nmods x*; *P nmods y* ⟧ ⟹ *P nmods* (*x* ; *y*)
by (*rel-auto*, *force*)

abbreviation *FSFR6* == *?*[&*tis*:*iadminToken*:*adminTokenPresence* = *absent*] ;; *TISOp nmods* {&*tis*:*config*, &*tis*:*ifloppy*}

lemma *FSFR6-proof*: *FSFR6*
apply (*rule nmods-union*)
apply (*simp add*: *TISOp-def seqr-or-distr closure TISUserEntryOp-nmods-config TISEnrolOp-nmods-config*

TISAdminLogon-nmods-config TISStartAdminOp-nmods-config TISAdminLogout-nmods-config
TISIdle-nmods-config TISAdminOp-absent-nmods-config)
apply (*simp add*: *TISOp-nmods-floppy closure*)
done

end