Tokeneer in Isabelle/UTP

Simon Foster, Mario Gleirscher, and Yakoub Nemouchi

June 26, 2019

Contents

1 Tokeneer in Isabelle/UTP

```
theory Tokeneer
  imports
    ZedLite.zedlite
```

UTP.utp-easy-parser
begin recall-syntax

2 Introduction

hide-const *dom*

named-theorems *tis-defs*

2.1 TIS Basic Types

type-synonym *TIME* = *nat*

abbreviation *zeroTime* ≡ *0*

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

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

record *Clearance* =
class :: *CLASS*

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

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

typedecl *USER*

consts *ISSUER* :: *USER set*

typedecl *FINGERPRINT*

typedecl *FINGERPRINTTEMPLATE*

alphabet *FingerprintTemplate* =
template :: *FINGERPRINTTEMPLATE*

2.2 Keys and Encryption

typedecl *KEYPART*

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

2.3 Certificates, Tokens, and Enrolment Data

2.3.1 Certificates

typedecl *TOKENID*

record *CertificateId* =
  issuer :: *USER*

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

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

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

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

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

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

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

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

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

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

type-synonym *AuthCert* = *PrivCert*

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

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

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

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

2.3.2 Tokens

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

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

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

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

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

2.3.3 Enrolment Data

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

We had to add two extra clauses to Enrol here that we're specified in the Tokeneer Z-schema, namely that (1) all issuer certificates correspond to 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. idStationCert e ∈ issuerCerts e ∧
            subject ‘ issuerCerts e ⊆ ISSUER ∧
            (∀ c ∈ issuerCerts e. ∀ d ∈ issuerCerts e. subject c = subject d ⟶
c = d)}

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

2.4 World Outside the ID Station

2.4.1 Real World Types and Entities (1)

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

datatype FINGERPRINTTRY = noFP | badFP | goodFP FINGERPRINT

alphabet Finger =
  currentFinger :: FINGERPRINTTRY
  fingerPresence :: PRESENCE

abbreviation Finger :: Finger upred where Finger ≡ true

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

definition DoorLatchAlarm :: DoorLatchAlarm upred where
[upred-defs, tis-defs]:
DoorLatchAlarm = (
  (currentLatch = «locked» ⇔ currentTime ≥ latchTimeout) ∧
  (doorAlarm = «alarming» ⇔
    (currentDoor = «dopen»
      ∧ currentLatch = «locked»
      ∧ currentTime ≥ alarmTimeout))
  )ₑ
```

3 The Token ID Station

3.1 Configuration Data

```
consts maxSupportedLogSize :: nat

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

definition Config :: Config upred where
[upred-defs, tis-defs]:
Config = (alarmThresholdSize < minPreservedLogSize ∧
          minPreservedLogSize ≤ «maxSupportedLogSize» ∧
          latchUnlockDuration > 0 ∧
          alarmSilentDuration > 0)ₑ
```

3.2 AuditLog

```
typedecl AuditEvent
typedecl AuditUser

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

3.2.1 Real World Types and Entities (2)

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

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

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

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

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

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

datatype $KEYBOARD = noKB \mid badKB \mid keyedOps\ (ofKeyedOps: ADMINOP)$

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

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

3.3 System Statistics

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

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

3.4 Key Store

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

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

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

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

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

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

definition $\mathit{AuthCertOK} :: {}'a\ \mathit{Certificate\text{-}scheme} \Rightarrow \mathit{KeyStore}\ \mathit{upred}$ **where**
$[\mathit{upred\text{-}defs},\ \mathit{tis\text{-}defs}]$: $\mathit{AuthCertOK}\ c = (\mathit{CertIssuerIsThisTIS}\ c \wedge \mathit{CertOK}\ c)$

definition $\mathit{oldestLogTime} :: \mathit{Audit}\ \mathit{set} \Rightarrow \mathit{TIME}$ **where**
$[\mathit{upred\text{-}defs},\ \mathit{tis\text{-}defs}]$:
$\mathit{oldestLogTime}\ \mathit{lg} = (\mathit{Min}\ (\mathit{get}_{\mathit{auditTime}}\ `\ \mathit{lg}))$

definition $\mathit{newestLogTime} :: \mathit{Audit}\ \mathit{set} \Rightarrow \mathit{TIME}$ **where**
$[\mathit{upred\text{-}defs},\ \mathit{tis\text{-}defs}]$:
$\mathit{newestLogTime}\ \mathit{lg} = (\mathit{Max}\ (\mathit{get}_{\mathit{auditTime}}\ `\ \mathit{lg}))$

lemma $\mathit{newestLogTime\text{-}union}$: $[\![\ \mathit{finite}\ A;\ A \neq \{\};\ \mathit{finite}\ B;\ B \neq \{\}\ ]\!] \Longrightarrow \mathit{newestLogTime}\ (A \cup B) \geq \mathit{newestLogTime}\ A$
by $(\mathit{simp}\ \mathit{add}\colon \mathit{newestLogTime\text{-}def})$

lemma $\mathit{oldestLogTime\text{-}union}$: $[\![\ \mathit{finite}\ A;\ A \neq \{\};\ \mathit{finite}\ B;\ B \neq \{\}\ ]\!] \Longrightarrow \mathit{oldestLogTime}\ (A \cup B) \leq \mathit{oldestLogTime}\ A$
by $(\mathit{simp}\ \mathit{add}\colon \mathit{oldestLogTime\text{-}def})$

3.5 Administration

alphabet $\mathit{Admin} =$
$\mathit{rolePresent} :: \mathit{PRIVILEGE}\ \mathit{option}$
$\mathit{availableOps} :: \mathit{ADMINOP}\ \mathit{set}$
$\mathit{currentAdminOp} :: \mathit{ADMINOP}\ \mathit{option}$

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

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

3.6 AuditLog (2)

alphabet $\mathit{AuditLog} =$
$\mathit{auditLog} :: \mathit{Audit\ set}$
$\mathit{auditAlarm} :: \mathit{ALARM}$

abbreviation $\mathit{AuditLog} :: \mathit{AuditLog\ upred}$ **where**
$\mathit{AuditLog} \equiv \mathit{true}$

3.6.1 Real World Types and Entities (3)

datatype $\mathit{SCREENTEXT} = \mathit{clear} \mid \mathit{welcomeAdmin} \mid \mathit{busy} \mid \mathit{removeAdminToken}$
$\mid \mathit{closeDoor} \mid$
$\mathit{requestAdminOp} \mid \mathit{doingOp} \mid \mathit{invalidRequest} \mid \mathit{invalidData} \mid$
$\mathit{insertEnrolmentData} \mid \mathit{validatingEnrolmentData} \mid \mathit{enrolmentFailed} \mid$
$\mathit{archiveFailed} \mid \mathit{insertBlankFloppy} \mid \mathit{insertConfigData} \mid$
$\mathit{displayStats\ Stats} \mid \mathit{displayConfigData\ Config}$

alphabet $\mathit{Screen} =$
$\mathit{screenStats} :: \mathit{SCREENTEXT}$
$\mathit{screenMsg} :: \mathit{SCREENTEXT}$
$\mathit{screenConfig} :: \mathit{SCREENTEXT}$

datatype $\mathit{TOKENTRY} = \mathit{noT} \mid \mathit{badT} \mid \mathit{goodT}\ (\mathit{ofGoodT}\colon \mathit{Token})$

alphabet $\mathit{UserToken} =$
$\mathit{currentUserToken} :: \mathit{TOKENTRY}$
$\mathit{userTokenPresence} :: \mathit{PRESENCE}$

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

alphabet $\mathit{AdminToken} =$
$\mathit{currentAdminToken} :: \mathit{TOKENTRY}$
$\mathit{adminTokenPresence} :: \mathit{PRESENCE}$

definition $\mathit{AdminToken} :: \mathit{AdminToken\ upred}$ **where**
$[\mathit{upred\text{-}defs},\ \mathit{tis\text{-}defs}]$:

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

3.7 Internal State

datatype *STATUS* = *quiescent* | *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*]:
$UserTokenWithOKAuthCert =$
$\quad (\&iuserToken{:}currentUserToken \in_u \ll range(goodT) \gg \wedge$
$\quad\ (\exists\ t {\in} \ll TokenWithValidAuth \gg \cdot$
$\qquad (\ll goodT(t) \gg =_u \&iuserToken{:}currentUserToken$

(* ∧ &doorLatchAlarm:currentTime $\in_u$ ≪validityPeriod (the(authCert t))≫ *)
∧ (∃ c ∈ ≪IDCert≫ · ≪c = idCert t≫ ∧ CertOK c) $\oplus_p$ keyStore
∧ (∃ c ∈ ≪AuthCert≫ · ≪c = the (authCert t)≫ ∧ AuthCertOK c) $\oplus_p$ keyStore))
)

definition *UserTokenOK* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
UserTokenOK =
(&iuserToken:currentUserToken $\in_u$ ≪range(goodT)≫ ∧
(∃ t ·
(≪goodT(t)≫ $=_u$ &iuserToken:currentUserToken
∧ ≪t ∈ CurrentToken ti≫⟦ti → &doorLatchAlarm:currentTime⟧
∧ (∃ c ∈ ≪IDCert≫ · ≪c = idCert t≫ ∧ CertOK c) $\oplus_p$ keyStore
∧ (∃ c ∈ ≪PrivCert≫ · ≪c = privCert t≫ ∧ CertOK c) $\oplus_p$ keyStore
∧ (∃ c ∈ ≪IandACert≫ · ≪c = iandACert t≫ ∧ CertOK c) $\oplus_p$ keyStore))
)

definition *AdminTokenOK* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
AdminTokenOK =
(&iadminToken:currentAdminToken $\in_u$ ≪range(goodT)≫ ∧
(∃ t∈≪TokenWithValidAuth≫ ·
(≪goodT(t)≫ $=_u$ &iadminToken:currentAdminToken
∧ ≪t ∈ CurrentToken ti≫⟦ti → &doorLatchAlarm:currentTime⟧
∧ (∃ c ∈ ≪IDCert≫ · ≪c = idCert t≫ ∧ CertOK c) $\oplus_p$ keyStore
∧ (∃ c ∈ ≪AuthCert≫ · ≪Some c = authCert t≫ ∧ AuthCertOK c
∧ ≪role c ∈ ADMINPRIVILEGE≫) $\oplus_p$ keyStore
))
)

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

definition *IDStation-inv1* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv1 =
(internal:status ∈
{≪gotFinger≫, ≪waitingFinger≫, ≪waitingUpdateToken≫, ≪waitingEntry≫, ≪waitingRemoveTokenSuccess≫}
⇒ (@UserTokenWithOKAuthCert ∨ @UserTokenOK)$)_e$

definition *IDStation-inv2* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv2 =

(*admin*:*rolePresent* ≠ ≪*None*≫ ⇒ @*AdminTokenOK*)$_e$

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

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

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

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

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

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

Extra Invariant (1):

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

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

definition *IDStation-inv10* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
IDStation-inv10 =
(*iadminToken*:*adminTokenPresence* = ≪*present*≫ ∧ *admin*:*rolePresent* ≠ ≪*None*≫

$\Rightarrow \mathit{admin}{:}\mathit{rolePresent} = \mathit{Some}(\mathit{role}(\mathit{the}(\mathit{authCert}(\mathit{ofGoodT}(\mathit{iadminToken}{:}\mathit{currentAdminToken}))))))_e$

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

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

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

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

have *f1*: (*IDStation-inv* ∧ *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*$⦄_u$
    by (*metis* (*no-types*) *assms*(*1*) *hoare-r-weaken-pre*(*2*))
  have ⦃*IDStation*⦄ *P* ⦃*IDStation-inv*$⦄_u$
   using *f1* **by** (*metis* (*no-types*) *assms*(*2*) *hoare-r-weaken-pre*(*2*) *utp-pred-laws.inf-commute*)
  then show *?thesis*
using *f2 f1*
  using *hoare-r-conj* **by** *fastforce*
qed

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

4 Operations Interfacing to the ID Station (1)

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

abbreviation *TISControlledRealWorld* :: *TISControlledRealWorld upred* **where**
TISControlledRealWorld ≡ *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 =
($\bigvee$ *t* · *monitored*:*now* := *monitored*:*now* + «*t*» ;;
monitored:*door* := ∗ ;; *monitored*:*finger* := ∗ ;;
monitored:*userToken* := ∗ ;; *monitored*:*adminToken* := ∗ ;;
monitored:*floppy* := ∗ ;; *monitored*:*keyboard* := ∗ ;;
controlled:*latch* := ∗ ;; *controlled*:*alarm* := ∗ ;;
controlled:*display* := ∗ ;; *controlled*:*screen* := ∗)

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

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

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

5 Internal Operations

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

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

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

definition *LogChange* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$\mathit{LogChange} = (\mathit{AuditAlarm} \lor \mathit{AuditLatch} \lor \mathit{AuditDoor} \lor \mathit{AuditLogAlarm} \lor \mathit{AuditScreen} \lor \mathit{AuditDisplay} \lor \mathit{NoChange})$

5.1 Updating System Statistics

definition *AddSuccessfulEntryToStats* :: *Stats hrel* **where**
[*upred-defs*, *tis-defs*]:
$\mathit{AddSuccessfulEntryToStats} =$
$(\Delta[\mathit{Stats}] \land$
$\$\mathit{failEntry}' =_u \$\mathit{failEntry} \land$
$\$\mathit{successEntry}' =_u \$\mathit{successEntry} + 1 \land$
$\$\mathit{failBio}' =_u \$\mathit{failBio} \land$
$\$\mathit{successBio}' =_u \$\mathit{successBio})$

lemma *AddSuccessfulEntryToStats-prog-def*:
$\mathit{AddSuccessfulEntryToStats} = (\mathit{successEntry} := \mathit{successEntry} + 1)$
by (*rel-auto*)

definition *AddFailedEntryToStats* :: *Stats hrel* **where**
[*upred-defs*, *tis-defs*]:
$\mathit{AddFailedEntryToStats} =$
$(\Delta[\mathit{Stats}] \land$
$\$\mathit{failEntry}' =_u \$\mathit{failEntry} + 1 \land$
$\$\mathit{successEntry}' =_u \$\mathit{successEntry} \land$
$\$\mathit{failBio}' =_u \$\mathit{failBio} \land$
$\$\mathit{successBio}' =_u \$\mathit{successBio})$

lemma *AddFailedEntryToStats-prog-def*:
$\mathit{AddFailedEntryToStats} = (\mathit{failEntry} := \mathit{failEntry} + 1)$
by (*rel-auto*)

definition *AddSuccessfulBioEntryToStats* :: *Stats hrel* **where**
[*upred-defs*, *tis-defs*]:
$\mathit{AddSuccessfulBioEntryToStats} =$
$(\Delta[\mathit{Stats}] \land$
$\$\mathit{failEntry}' =_u \$\mathit{failEntry} \land$
$\$\mathit{successEntry}' =_u \$\mathit{successEntry} \land$
$\$\mathit{failBio}' =_u \$\mathit{failBio} \land$

$successBio´ $=_u$ $successBio + 1)

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

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

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

5.2 Operating the Door

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

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

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

5.3 Certificate Operations

5.3.1 Generating Authorisation Certificates

definition $\mathit{NewAuthCert} :: \text{-} \Rightarrow \text{-} \Rightarrow \mathit{TIME} \Rightarrow \mathit{IDStation}\ \mathit{upred}$ **where**
$[\mathit{upred\text{-}defs},\ \mathit{tis\text{-}defs}]$:
$\mathit{NewAuthCert}\ \mathit{token}\ \mathit{newAuthCert}\ \mathit{curTime} = ($
$\ll \mathit{token} \in \mathit{ValidToken} \gg \wedge$
$\mathit{KeyStore} \oplus_p \mathit{keyStore} \wedge$
$\mathit{Config} \oplus_p \mathit{config} \wedge$

$\&\mathit{keyStore}{:}\mathit{ownName} \neq_u \mathit{None}_u \wedge$

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

5.3.2 Adding Authorisation Certificates to User Token

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

6 Operations Interfacing to the ID Station (2)

6.1 Obtaining inputs from the real world

6.1.1 Polling the Real World

definition $\mathit{PollTime} :: \mathit{SystemState}\ \mathit{hrel}$ **where**
$[\mathit{upred\text{-}defs}]$:
$\mathit{PollTime} =$
$(\Delta[\mathit{idStation}{:}\mathit{doorLatchAlarm},\mathit{DoorLatchAlarm}] \wedge$
$\$\mathit{idStation}{:}\mathit{doorLatchAlarm}{:}\mathit{currentTime}' =_u \$\mathit{realWorld}{:}\mathit{monitored}{:}\mathit{now})$

definition $\mathit{PollDoor} :: \mathit{SystemState}\ \mathit{hrel}$ **where**
$[\mathit{upred\text{-}defs}]$:

$\mathit{PollDoor} =$
$(\Delta[\mathit{idStation}{:}\mathit{doorLatchAlarm},\mathit{DoorLatchAlarm}] \wedge$
$\$\mathit{idStation}{:}\mathit{doorLatchAlarm}{:}\mathit{currentDoor}' =_u \$\mathit{realWorld}{:}\mathit{monitored}{:}\mathit{door} \wedge$
$\$\mathit{idStation}{:}\mathit{doorLatchAlarm}{:}\mathit{latchTimeout}' =_u \$\mathit{idStation}{:}\mathit{doorLatchAlarm}{:}\mathit{latchTimeout}$
$\wedge$
$\$\mathit{idStation}{:}\mathit{doorLatchAlarm}{:}\mathit{alarmTimeout}' =_u \$\mathit{idStation}{:}\mathit{doorLatchAlarm}{:}\mathit{alarmTimeout})$

definition *PollUserToken* :: *SystemState hrel* **where**
[*upred-defs*]:
$\mathit{PollUserToken} =$
$(\Delta[\mathit{idStation}{:}\mathit{iuserToken}, \mathit{UserToken}] \wedge$
$\$\mathit{idStation}{:}\mathit{iuserToken}{:}\mathit{userTokenPresence}' =_u \ll\mathit{present}\gg \Leftrightarrow \$\mathit{realWorld}{:}\mathit{monitored}{:}\mathit{userToken}$
$\neq_u \ll\mathit{noT}\gg \wedge$
$\$\mathit{idStation}{:}\mathit{iuserToken}{:}\mathit{currentUserToken}' =_u$
$(\$\mathit{realWorld}{:}\mathit{monitored}{:}\mathit{userToken} \triangleleft \$\mathit{realWorld}{:}\mathit{monitored}{:}\mathit{userToken} \neq_u \ll\mathit{noT}\gg \triangleright$
$\$\mathit{idStation}{:}\mathit{iuserToken}{:}\mathit{currentUserToken}))$

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

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

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

definition *PollKeyboard* :: *SystemState hrel* **where**
[*upred-defs*]:
PollKeyboard =
  (Δ[*idStation*:*ikeyboard*,*Keyboard*] $\wedge$
  \$*idStation*:*ikeyboard*:*keyedDataPresence*´ $=_u$ ≪*present*≫ $\Leftrightarrow$ \$*realWorld*:*monitored*:*keyboard* $\neq_u$ ≪*noKB*≫ $\wedge$
   \$*idStation*:*ikeyboard*:*currentKeyedData*´ $=_u$
    (\$*realWorld*:*monitored*:*keyboard* ◁ \$*realWorld*:*monitored*:*keyboard* $\neq_u$ ≪*noKB*≫ ▷ \$*idStation*:*ikeyboard*:*currentKeyedData*))

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

$(\mathit{idStation}{:}\mathit{ikeyboard}{:}\mathit{currentKeyedData}$
$\quad \triangleleft\ (\mathit{realWorld}{:}\mathit{monitored}{:}\mathit{keyboard} = \ll\mathit{noKB}\gg)\ \triangleright$
$\mathit{realWorld}{:}\mathit{monitored}{:}\mathit{keyboard})$
$)$

6.2 The ID Station Changes the World

6.2.1 Periodic Updates

definition $\mathit{UpdateLatch} :: \mathit{SystemState\ hrel}$ **where**
$[\mathit{upred\text{-}defs}]$:
$\mathit{UpdateLatch} =$
$(\Xi[\mathit{idStation}{:}\mathit{doorLatchAlarm},\mathit{DoorLatchAlarm}] \wedge$
$\mathit{RealWorldChanges} \oplus_r \mathit{realWorld} \wedge$
$\$\mathit{realWorld}{:}\mathit{controlled}{:}\mathit{latch}' =_u \$\mathit{idStation}{:}\mathit{doorLatchAlarm}{:}\mathit{currentLatch})$

definition $\mathit{UpdateAlarm} :: \mathit{SystemState\ hrel}$ **where**
$[\mathit{upred\text{-}defs}]$:
$\mathit{UpdateAlarm} =$
$(\Xi[\mathit{idStation}{:}\mathit{doorLatchAlarm},\mathit{DoorLatchAlarm}] \wedge$
$\mathit{RealWorldChanges} \oplus_r \mathit{realWorld} \wedge$
$\lceil \mathit{AuditLog} \rceil_< \oplus_r \mathit{idStation}{:}\mathit{audit} \wedge$
$\$\mathit{realWorld}{:}\mathit{controlled}{:}\mathit{alarm}' =_u \ll\mathit{alarming}\gg \Leftrightarrow (\$\mathit{idStation}{:}\mathit{doorLatchAlarm}{:}\mathit{doorAlarm}$
$=_u \ll\mathit{alarming}\gg$
$\vee\ \$\mathit{idStation}{:}\mathit{audit}{:}\mathit{auditAlarm} =_u$
$\ll\mathit{alarming}\gg))$

definition $\mathit{UpdateDisplay} :: \mathit{SystemState\ hrel}$ **where**
$[\mathit{upred\text{-}defs}]$:
$\mathit{UpdateDisplay} =$
$(\Delta[\mathit{idStation},\mathit{IDStation}] \wedge$
$\mathit{RealWorldChanges} \oplus_r \mathit{realWorld} \wedge$
$\$\mathit{realWorld}{:}\mathit{controlled}{:}\mathit{display}' =_u \$\mathit{idStation}{:}\mathit{currentDisplay} \wedge$
$\$\mathit{idStation}{:}\mathit{currentDisplay}' =_u \$\mathit{idStation}{:}\mathit{currentDisplay})$

definition $\mathit{UpdateScreen} :: \mathit{SystemState\ hrel}$ **where**
$[\mathit{upred\text{-}defs}]$:
$\mathit{UpdateScreen} =$
$(\Delta[\mathit{idStation},\mathit{IDStation}] \wedge$
$\Xi[\mathit{idStation}{:}\mathit{admin},\mathit{Admin}] \wedge$
$\mathit{RealWorldChanges} \oplus_r \mathit{realWorld} \wedge$
$\$\mathit{realWorld}{:}\mathit{controlled}{:}\mathit{screen}{:}\mathit{screenMsg}' =_u \$\mathit{idStation}{:}\mathit{currentScreen}{:}\mathit{screenMsg}$
$\wedge$
$\$\mathit{realWorld}{:}\mathit{controlled}{:}\mathit{screen}{:}\mathit{screenConfig}' =_u$
$(\$\mathit{idStation}{:}\mathit{currentScreen}{:}\mathit{screenConfig}$
$\quad \triangleleft\ \$\mathit{idStation}{:}\mathit{admin}{:}\mathit{rolePresent} =_u \ll\mathit{Some}(\mathit{securityOfficer})\gg\ \triangleright$
$\ll\mathit{clear}\gg) \wedge$
$\$\mathit{realWorld}{:}\mathit{controlled}{:}\mathit{screen}{:}\mathit{screenStats}' =_u$
$(\$\mathit{idStation}{:}\mathit{currentScreen}{:}\mathit{screenStats}$
$\quad \triangleleft\ \$\mathit{idStation}{:}\mathit{admin}{:}\mathit{rolePresent} \neq_u \ll\mathit{None}\gg\ \triangleright$

$\ll$*clear*$\gg$))

definition *TISUpdate* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISUpdate =
  (*realWorld*:[*RealWorldChanges*]$^+$;;
   realWorld:*controlled*:*latch* := *idStation*:*doorLatchAlarm*:*currentLatch* ;;
   realWorld:*controlled*:*alarm* := ($\ll$*alarming*$\gg$
                        $\triangleleft$ (*idStation*:*doorLatchAlarm*:*doorAlarm* = $\ll$*alarming*$\gg$
                          $\vee$ *idStation*:*audit*:*auditAlarm* = $\ll$*alarming*$\gg$)
                        $\triangleright$ $\ll$*silent*$\gg$) ;;
   realWorld:*controlled*:*display* := *idStation*:*currentDisplay*)

6.2.2 Updating the User Token

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

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

7 The User Entry Operation (1)

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

lemma *mark-alpha-ResetScreenMessage* [*mark-alpha*]:
  $\Sigma \triangleleft_\alpha$ *ResetScreenMessage* = {&*admin*,&*currentScreen*,&*internal*} $\triangleleft_\alpha$ *ResetScreenMessage*

by (*rel-auto*)

definition *UserEntryContext* :: *SystemState hrel* **where**
[*upred-defs*]:
UserEntryContext =
$((\mathit{RealWorldChanges} \wedge \Xi[\mathit{controlled},\ \mathit{TISControlledRealWorld}]) \oplus_r \mathit{realWorld} \wedge$
$(\Delta[\mathit{iuserToken},\mathit{UserToken}] \wedge$
$\Delta[\mathit{doorLatchAlarm},\mathit{DoorLatchAlarm}] \wedge$
$\Delta[\mathit{audit},\mathit{AuditLog}] \wedge$
$\Xi[\mathit{config},\ \mathit{Config}] \wedge$
$\Xi[\mathit{iadminToken},\ \mathit{AdminToken}] \wedge$
$\Xi[\mathit{keyStore},\ \mathit{KeyStore}] \wedge$
$\Xi[\mathit{admin},\ \mathit{Admin}] \wedge$
$\Xi[\mathit{ikeyboard},\ \mathit{Keyboard}] \wedge$
$\Xi[\mathit{ifloppy},\ \mathit{Floppy}] \wedge$
$\Xi[\mathit{ifinger},\ \mathit{Finger}] \wedge$
$\Delta[\mathit{IDStation\text{-}inv}] \wedge$
$\mathit{ResetScreenMessage} \wedge$
$(\$\mathit{enclaveStatus}' =_u \$\mathit{enclaveStatus} \wedge$
$(\$\mathit{status} \neq_u \ll\mathit{waitingEntry}\gg \Rightarrow \$\mathit{tokenRemovalTimeout}' =_u \$\mathit{tokenRemovalTimeout})$
$) \oplus_r \mathit{internal}) \oplus_r \mathit{idStation}$
$)$

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

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

lemma $\mathit{pre}((\mathit{RealWorldChanges} \wedge \Xi[\mathit{controlled},\ \mathit{TISControlledRealWorld}]) \oplus_r$ *re-*

$alWorld) = true$
by (*rel-auto*)

7.1 User Token Tears

definition *UserTokenTorn* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$UserTokenTorn =$
$((internal{:}status \in \{\ll gotUserToken\gg, \ll waitingUpdateToken\gg, \ll waitingFinger\gg,$
$\ll gotFinger\gg, \ll waitingEntry\gg\}$
$\wedge\ iuserToken{:}userTokenPresence = \ll absent\gg$
$) \longrightarrow_r currentDisplay := \ll welcom\gg\ ;;\ internal{:}status := \ll quiescent\gg)$

lemma *UserTokenTorn-correct* [*hoare-safe*]: $\{\!|IDStation|\!\}\, UserTokenTorn\, \{\!|IDStation|\!\}_u$
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*]:
$EnclaveContext =$
$(\Delta[idStation, IDStation] \wedge$
$RealWorldChanges \oplus_r realWorld \wedge$
$\Xi[realWorld{:}controlled, TISControlledRealWorld] \wedge$
$\Xi[idStation{:}iuserToken, UserToken] \wedge$
$\Xi[idStation{:}iadminToken, AdminToken] \wedge$
$\Xi[idStation{:}ifinger, Finger] \wedge$
$\Xi[idStation{:}stats, Stats] \wedge$
$(\$tokenRemovalTimeout\acute{} =_u \$tokenRemovalTimeout) \oplus_r idStation{:}internal$
$)$

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

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

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

8.1 Updating the Key Store

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

lemma *rel-typed-Collect* [*rclos*]: $[\![\bigwedge x\ y.\ P\ (x,\ y) \Longrightarrow x \in A \wedge y \in B\ ]\!] \Longrightarrow \mathit{Collect}\ P \in A \leftrightarrow_r B$
 by (*auto simp add*: *rel-typed-def*)

lemma *rel-pfun-Collect* [*rclos*]: $[\![\bigwedge x\ y.\ P\ (x,\ y) \Longrightarrow x \in A \wedge y \in B;\ \bigwedge x\ y\ z.\ [\![P\ (x,\ y);\ P\ (x,\ z)\ ]\!] \Longrightarrow y = z\ ]\!] \Longrightarrow \mathit{Collect}\ P \in A \rightharpoonup_r B$
 by (*auto simp add*: *rel-pfun-def rel-typed-def functional-algebraic*)

lemma *UpdateKeyStore-prog-def*:
 UpdateKeyStore e =
  $?[@\mathit{KeyStore} \wedge \ll e \in \mathit{ValidEnrol} \gg]$;;
  $\mathit{ownName} :=\ \ll \mathit{Some}\ (\mathit{subject}\ (\mathit{idStationCert}\ e)) \gg$;;
  $\mathit{issuerKey} := \mathit{issuerKey} \oplus \ll \{(\mathit{subject}\ c,\ \mathit{subjectPubK}\ c) \mid c.\ c \in \mathit{issuerCerts}\ e\} \gg \oplus \{(\mathit{the}(\mathit{ownName}),\ \ll \mathit{subjectPubK}\ (\mathit{idStationCert}\ e) \gg)\}$
 (**is** *?P* = *?Q*)
proof (*rule antisym*)
 show $?P \sqsubseteq ?Q$
  by (*rel-auto*, *auto intro*: *rclos intro*!: *rel-pfun-override rel-pfun-Collect*)
 show $?Q \sqsubseteq ?P$
  by (*rel-auto*)
qed

lemma *pre-KeyStore*:
 $e \in \mathit{ValidEnrol} \Longrightarrow \mathit{Dom}(\mathit{UpdateKeyStore}\ e) = \mathit{KeyStore}$
 apply (*rel-auto*)
  apply (*auto intro*: *rclos intro*!: *rel-pfun-override*)
 done

definition *UpdateKeyStoreFromFloppy* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
UpdateKeyStoreFromFloppy =
 $(\Delta[\mathit{keyStore},\mathit{KeyStore}] \wedge$
 $\lceil \mathit{Floppy} \oplus_p \mathit{ifloppy} \rceil_< \wedge$
 $\$\mathit{ifloppy}{:}\mathit{currentFloppy} \in_u \ll \mathit{range}(\mathit{enrolmentFile}) \gg \wedge$
 $(\exists\ e \cdot \ll e \gg =_u \ll \mathit{enrolmentFile\text{-}of} \gg (\$\mathit{ifloppy}{:}\mathit{currentFloppy})_a$
  $\wedge\ \mathit{UpdateKeyStore}\ e \oplus_r \mathit{keyStore}))$

9 The User Entry Operation (2)

9.1 Reading the User Token

definition *ReadUserToken* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
$ReadUserToken =$
$\quad ((internal{:}enclaveStatus \in \{\ll enclaveQuiescent\gg, \ll waitingRemoveAdminToken\text{-}Fail\gg\}$
$\qquad \wedge\ internal{:}status = \ll quiescent\gg$
$\qquad \wedge\ iuserToken{:}userTokenPresence = \ll present\gg$
$\quad) \longrightarrow_r currentDisplay := \ll wait\gg\ ;;\ internal{:}status := \ll gotUserToken\gg)$

9.2 Validating the User Token

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

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

lemma *UEC-correct*: $\{\!|I|\!\}P\{\!|I|\!\}_u \Longrightarrow \{\!|I \oplus_p idStation|\!\}UEC(P)\{\!|I \oplus_p idStation|\!\}_u$
apply (*simp add*: *wlp-hoare-link wp UEC-def alpha unrest usubst*)
apply (*rel-simp*)
done

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

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

lemma *TISReadUserToken-correct*: $\{\!|IDStation \oplus_p idStation|\!\}TISReadUserToken\{\!|IDStation \oplus_p idStation|\!\}_u$
by (*simp add*: *ReadUserToken-correct TISReadUserToken-def UEC-correct*)

lemma ‘$UserTokenOK \Rightarrow (\exists\ e {\in} \ll ValidToken\gg \cdot \ll goodT(e)\gg =_u \&iuserToken{:}currentUserToken)$‘
by (*rel-auto*)

lemma ‘$UserTokenWithOKAuthCert \Rightarrow (\exists\ e {\in} \ll TokenWithValidAuth\gg \cdot \ll goodT(e)\gg =_u$

&*iuserToken*:*currentUserToken*)'
  by (*rel-auto*)

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

lemma *BioCheckNotRequired-correct*: ⦃*IDStation*⦄*BioCheckNotRequired*⦃*IDStation*⦄$_u$
  apply (*rule IDStation-correct-intro*)
   apply (*simp add*: *tis-defs*, *hoare-auto*)
  apply (*simp add*: *tis-defs*, *hoare-auto*)
  done

definition *BioCheckRequired* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
BioCheckRequired =
  ((*internal*:*status* = ≪*gotUserToken*≫
  ∧ *iuserToken*:*userTokenPresence* = ≪*present*≫
  ∧ (¬ @*UserTokenWithOKAuthCert*) ∧ @*UserTokenOK*
) $\longrightarrow_r$ *internal*:*status* := ≪*waitingFinger*≫ ;; *currentDisplay* := ≪*insertFinger*≫)

lemma *BioCheckRequired-correct*: ⦃*IDStation*⦄*BioCheckRequired*⦃*IDStation*⦄$_u$
  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*]: *ValidateUserTokenOK* = (*BioCheckRequired* ∨
BioCheckNotRequired)

lemma *ValidateUserTokenOK-correct*: ⦃*IDStation*⦄ *ValidateUserTokenOK* ⦃*IDStation*⦄$_u$
 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*)
) $\longrightarrow_r$ *internal*:*status* := ≪*waitingRemoveTokenFail*≫ ;; *currentDisplay* := ≪*removeToken*≫)

lemma *ValidateUserTokenFail-correct*: ⦃*IDStation*⦄ *ValidateUserTokenFail*⦃*IDStation*⦄$_u$
  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*⦄$_u$
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* $\oplus_p$ *idStation*⦄*TISValidateUserToken*⦃*IDStation* $\oplus_p$ *idStation*⦄$_u$
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*≫
) $\longrightarrow_r$ *internal*:*status* := ≪*gotFinger*≫ ;; *currentDisplay* := ≪*wait*≫)

lemma *ReadFingerOK-correct*: ⦃*IDStation*⦄*ReadFingerOK*⦃*IDStation*⦄$_u$
apply (*rule IDStation-correct-intro*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
apply (*simp add*: *tis-defs*, *hoare-auto*)
done

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

definition *FingerTimeout* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
FingerTimeout =
 ((*internal*:*status* = $\ll$*waitingFinger*$\gg$
  $\wedge$ *ifinger*:*fingerPresence* = $\ll$*absent*$\gg$
  $\wedge$ *iuserToken*:*userTokenPresence* = $\ll$*present*$\gg$
) $\longrightarrow_r$ *currentDisplay* := $\ll$*removeToken*$\gg$;; *internal*:*status* := $\ll$*waitingRemoveTokenFail*$\gg$)

lemma *FingerTimeout-correct*: $\{\!|$*IDStation*$|\!\}$*FingerTimeout*$\{\!|$*IDStation*$|\!\}_u$
 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*) $\vee$ *UEC*(*FingerTimeout*) $\vee$ *UEC*(*NoFinger*)
  $\vee$ *UEC*(*UserTokenTorn* ;; *?*[*internal*:*status* = $\ll$*waitingFinger*$\gg$]))

lemma *TISReadFinger-correct*: $\{\!|$*IDStation* $\oplus_p$ *idStation*$|\!\}$*TISReadFinger*$\{\!|$*IDStation* $\oplus_p$ *idStation*$|\!\}_u$
 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* = $\ll$*gotFinger*$\gg$
  $\wedge$ *iuserToken*:*userTokenPresence* = $\ll$*present*$\gg$
  $\wedge$ @*FingerOK*
) $\longrightarrow_r$ *currentDisplay* := $\ll$*wait*$\gg$;; *internal*:*status* := $\ll$*waitingUpdateToken*$\gg$)

lemma *ValidateFingerOK-correct*: $\{\!|$*IDStation*$|\!\}$*ValidateFingerOK*$\{\!|$*IDStation*$|\!\}_u$
 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* = $\ll$*gotFinger*$\gg$
  $\wedge$ *iuserToken*:*userTokenPresence* = $\ll$*present*$\gg$
  $\wedge$ @*FingerOK*
) $\longrightarrow_r$ *currentDisplay* := $\ll$*removeToken*$\gg$;; *internal*:*status* := $\ll$*waitingRemoveTokenFail*$\gg$)

lemma *ValidateFingerFail-correct*: $\{\!|$*IDStation*$|\!\}$ *ValidateFingerFail* $\{\!|$*IDStation*$|\!\}_u$
 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*) $\vee$ *UEC*(*ValidateFingerFail*)
  $\vee$ *UEC*(*UserTokenTorn* ;; *?*[*internal*:*status* = «*gotFinger*»]))

lemma *TISValidateFinger-correct*: $\{\!|$*IDStation* $\oplus_p$ *idStation*$|\!\}$ *TISValidateFinger* $\{\!|$*IDStation* $\oplus_p$ *idStation*$|\!\}_u$
 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*»
  $\wedge$ *iuserToken*:*userTokenPresence* = «*present*»
) $\longrightarrow_r$ *AddAuthCertToUserToken* ;;
  currentDisplay := «*wait*» ;;
  internal:*status* := «*waitingEntry*»)

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

lemma *WriteUserTokenOK-correct*: $\{\!|$*IDStation*$|\!\}$ *WriteUserTokenOK* $\{\!|$*IDStation*$|\!\}_u$
proof –
 have *inv*: $\{\!|$*IDStation-inv*$|\!\}$ *WriteUserTokenOK* $\{\!|$*IDStation-inv*$|\!\}_u$
 proof –
  have *a*:$\{\!|$*IDStation-inv1* $\wedge$ *IDStation-inv9*$|\!\}$ *WriteUserTokenOK* $\{\!|$*IDStation-inv9*$|\!\}_u$
   by (*hoare-wlp-auto defs*: *tis-defs*)
  have *1*:$\{\!|$*IDStation-inv*$|\!\}$ *WriteUserTokenOK* $\{\!|$*IDStation-inv9*$|\!\}_u$
   by (*rule-tac pre-str-hoare-r*[*OF* - *a*], *rel-auto*)
  have *b*: $\{\!|$*IDStation-inv1*$|\!\}$ *WriteUserTokenOK* $\{\!|$*IDStation-inv1*$|\!\}_u$
   by (*hoare-wlp-auto defs*: *tis-defs*)
  have *2*:$\{\!|$*IDStation-inv*$|\!\}$ *WriteUserTokenOK* $\{\!|$*IDStation-inv1*$|\!\}_u$
   by (*rule-tac pre-str-hoare-r*[*OF* - *b*], *rel-auto*)
  have *3*:
   $\{\!|$*IDStation-inv*$|\!\}$ *WriteUserTokenOK* $\{\!|$*IDStation-inv2*$|\!\}_u$
   $\{\!|$*IDStation-inv*$|\!\}$ *WriteUserTokenOK* $\{\!|$*IDStation-inv3*$|\!\}_u$
   $\{\!|$*IDStation-inv*$|\!\}$ *WriteUserTokenOK* $\{\!|$*IDStation-inv4*$|\!\}_u$
   $\{\!|$*IDStation-inv*$|\!\}$ *WriteUserTokenOK* $\{\!|$*IDStation-inv5*$|\!\}_u$
   $\{\!|$*IDStation-inv*$|\!\}$ *WriteUserTokenOK* $\{\!|$*IDStation-inv6*$|\!\}_u$

```
      ⦃IDStation-inv⦄ WriteUserTokenOK ⦃IDStation-inv7⦄_u
      ⦃IDStation-inv⦄ WriteUserTokenOK ⦃IDStation-inv8⦄_u
      ⦃IDStation-inv⦄ WriteUserTokenOK ⦃IDStation-inv10⦄_u
      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⦄_u
    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⦄_u
proof −
  have inv: ⦃IDStation-inv⦄ WriteUserTokenFail ⦃IDStation-inv⦄_u
  proof −
   have a:⦃IDStation-inv1 ∧ IDStation-inv9⦄ WriteUserTokenFail ⦃IDStation-inv9⦄_u
      by (hoare-wlp-auto defs: tis-defs)
    have 1:⦃IDStation-inv⦄ WriteUserTokenFail ⦃IDStation-inv9⦄_u
      by (rule-tac pre-str-hoare-r[OF - a], rel-auto)
    have b: ⦃IDStation-inv1⦄ WriteUserTokenFail ⦃IDStation-inv1⦄_u
      by (hoare-wlp-auto defs: tis-defs)
    have 2:⦃IDStation-inv⦄ WriteUserTokenFail ⦃IDStation-inv1⦄_u
      by (rule-tac pre-str-hoare-r[OF - b], rel-auto)
    have 3:
      ⦃IDStation-inv⦄ WriteUserTokenFail ⦃IDStation-inv2⦄_u
      ⦃IDStation-inv⦄ WriteUserTokenFail ⦃IDStation-inv3⦄ᵤ
      ⦃IDStation-inv⦄ WriteUserTokenFail ⦃IDStation-inv4⦄ᵤ
      ⦃IDStation-inv⦄ WriteUserTokenFail ⦃IDStation-inv5⦄ᵤ
      ⦃IDStation-inv⦄ WriteUserTokenFail ⦃IDStation-inv6⦄ᵤ
      ⦃IDStation-inv⦄ WriteUserTokenFail ⦃IDStation-inv7⦄ᵤ
      ⦃IDStation-inv⦄ WriteUserTokenFail ⦃IDStation-inv8⦄ᵤ
      ⦃IDStation-inv⦄ WriteUserTokenFail ⦃IDStation-inv10⦄ᵤ
      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⦄ᵤ
    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 idStation⦄TISWriteUserToken⦃IDStation ⊕ₚ idStation⦄ᵤ
proof −
  have 1: ⦃IDStation ⊕ₚ idStation⦄UEC(WriteUserToken) ;; UpdateUserToken⦃IDStation ⊕ₚ idStation⦄ᵤ
    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**
[*upred-defs*]:
UserAllowedEntry =
(((∃ *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*)))≫]))$_e$

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*⦄$_u$
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*⦄$_u$
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* $\oplus_p$ *idStation*⦄*TISValidateEntry*⦃*IDStation* $\oplus_p$ *idStation*⦄$_u$
  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*≫)
  $\longrightarrow_r$ *UnlockDoor* ;; *currentDisplay* := ≪*doorUnlocked*≫ ;; *internal*:*status* := ≪*quiescent*≫

lemma *UnlockDoorOK-correct*: ⦃*IDStation*⦄*UnlockDoorOK*⦃*IDStation*⦄$_u$
  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*≫) =
    (*internal*:*status* = ≪*waitingRemoveTokenSuccess*≫ ∧ *iuserToken*:*userTokenPresence* = ≪*absent*≫)$_e$
  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*⦄$_u$
  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*≫) $\longrightarrow_r$
internal:*status* := ≪*waitingRemoveTokenFail*≫ ;;
currentDisplay := ≪*removeToken*≫)

lemma *TokenRemovalTimeout-correct*: ⦃*IDStation*⦄*TokenRemovalTimeout*⦃*IDStation*⦄$_u$
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* $\oplus_p$ *idStation*⦄*TISUnlockDoor*⦃*IDStation* $\oplus_p$ *idStation*⦄$_u$
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*≫) $\longrightarrow_r$
internal:*status* := ≪*quiescent*≫ ;;
currentDisplay := ≪*welcom*≫)

lemma *FailedAccessTokenRemoved-correct*: ⦃*IDStation*⦄*FailedAccessTokenRemoved*⦃*IDStation*⦄$_u$
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* = ≪*waitingRemoveTokenFail*≫]))

lemma *TISCompleteFailedAccess-correct*:
⦃*IDStation* $\oplus_p$ *idStation*⦄*TISCompleteFailedAccess*⦃*IDStation* $\oplus_p$ *idStation*⦄$_u$
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* $\vee$ *TISValidateUserToken* $\vee$ *TISReadFinger* $\vee$ *TISValidateFinger*
$\vee$ *TISWriteUserToken* $\vee$ *TISValidateEntry* $\vee$ *TISUnlockDoor* $\vee$ *TISCompleteFailedAccess*)

lemma *hoare-disj* [*hoare-safe*]:
assumes $\{\!|pr|\!\}P\{\!|post|\!\}_u$ $\{\!|pr|\!\}Q\{\!|post|\!\}_u$
shows $\{\!|pr|\!\}(P \vee Q)\{\!|post|\!\}_u$
using *assms* **by** (*rel-auto*)

lemma *TISUserEntryOp-inv*: $\{\!|$*IDStation* $\oplus_p$ *idStation*$|\!\}$*TISUserEntryOp*$\{\!|$*IDStation* $\oplus_p$ *idStation*$|\!\}_u$
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* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
RequestEnrolment = (*EnrolContext* $\wedge$
Ξ[*idStation*:*keyStore*, *KeyStore*] $\wedge$
Ξ[*idStation*:*audit*, *AuditLog*] $\wedge$
Ξ[*idStation*:*internal*, *Internal*] $\wedge$
(\$*enclaveStatus* $=_u$ $\ll$*notEnrolled*$\gg$) $\oplus_r$ *idStation*:*internal* $\wedge$
(\$*floppyPresence* $=_u$ $\ll$*absent*$\gg$) $\oplus_r$ *idStation*:*ifloppy* $\wedge$
(\$*currentScreen*:*screenMsg*´ $=_u$ $\ll$*insertEnrolmentData*$\gg$ $\wedge$
\$*currentDisplay*´ $=_u$ $\ll$*blank*$\gg$) $\oplus_r$ *idStation*
)

definition *ReadEnrolmentFloppy* :: *SystemState hrel* **where**
ReadEnrolmentFloppy = (*EnrolContext* $\wedge$
Ξ[*idStation*:*keyStore*, *KeyStore*] $\wedge$
(\$*enclaveStatus* $=_u$ $\ll$*notEnrolled*$\gg$) $\oplus_r$ *idStation*:*internal* $\wedge$
(\$*floppyPresence* $=_u$ $\ll$*present*$\gg$) $\oplus_r$ *idStation*:*ifloppy* $\wedge$
(\$*currentScreen*:*screenMsg*´ $=_u$ $\ll$*validatingEnrolmentData*$\gg$ $\wedge$
\$*internal*:*status*´ $=_u$ \$*internal*:*status* $\wedge$
\$*currentDisplay*´ $=_u$ $\ll$*blank*$\gg$) $\oplus_r$ *idStation*
)

definition *ReadEnrolmentData* = (*ReadEnrolmentFloppy* $\vee$ *RequestEnrolment*)

10.1.2 Validating Enrolment data from Floppy

definition *EnrolmentDataOK* :: *IDStation upred* **where**
$\mathit{EnrolmentDataOK} = (\mathit{Floppy} \oplus_p \mathit{ifloppy} \wedge$
$\quad \mathit{KeyStore} \oplus_p \mathit{keyStore} \wedge$
$\quad (\mathit{ifloppy}{:}\mathit{currentFloppy} \in \ll \mathit{range}\ \mathit{enrolmentFile} \gg \wedge$
$\quad\quad \ll \mathit{enrolmentFile\text{-}of} \gg [\mathit{ifloppy}{:}\mathit{currentFloppy}] \in \ll \mathit{ValidEnrol} \gg)_e)$

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

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

definition $\mathit{ValidateEnrolmentData} = (\mathit{ValidateEnrolmentDataOK} \vee \mathit{ValidateEnrolmentDataFail})$

10.1.3 Completing a Failed Enrolment

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

definition *WaitingFloppyRemoval* :: *SystemState hrel* **where**
$WaitingFloppyRemoval =$
$(EnrolContext \wedge$
$\Xi[idStation,IDStation] \wedge$
$(\$internal{:}enclaveStatus =_u \ll waitingEndEnrol \gg \wedge$
$\$ifloppy{:}floppyPresence =_u \ll present \gg$
$) \oplus_r idStation)$

definition $CompleteFailedEnrolment = (FailedEnrolFloppyRemoved \vee WaitingFloppyRemoval)$

10.1.4 The Complete Enrolment

definition *TISEnrolOp* :: *SystemState hrel* **where**
$[upred\text{-}defs, tis\text{-}defs]$:
$TISEnrolOp = false$

10.2 Further Administrator Operations

definition *AdminLogon* :: *IDStation hrel* **where**
$[upred\text{-}defs, tis\text{-}defs]$:
$AdminLogon =$
$((admin{:}rolePresent = \ll None \gg \wedge$
$(\exists\ t {\in} \ll ValidToken \gg \cdot (\ll goodT(t) \gg = iadminToken{:}currentAdminToken))$
$) \longrightarrow_r admin{:}rolePresent := Some(role(the(authCert(ofGoodT(iadminToken{:}currentAdminToken)))))$
$;;$
$admin{:}currentAdminOp := \ll None \gg\ ;;$
— The assignments below were added to ensure the invariant *Admin* is satisfied
$if\ admin{:}rolePresent = \ll Some(guard) \gg$
$then\ admin{:}availableOps := \{\ll overrideLock \gg\}$
$else\ if\ admin{:}rolePresent = \ll Some(auditManager) \gg$
$then\ admin{:}availableOps := \{\ll archiveLog \gg\}$
$else$
$admin{:}availableOps := \{\ll updateConfigData \gg, \ll shutdownOp \gg\}$
$fi\ fi)$

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

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

) $\longrightarrow_r$ *admin*:*currentAdminOp* := *Some*(*ofKeyedOps*(*ikeyboard*:*currentKeyedData*)))

definition *AdminFinishOp* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
AdminFinishOp =
((*admin*:*rolePresent* ≠ ≪*None*≫
∧ *admin*:*currentAdminOp* ≠ ≪*None*≫
) $\longrightarrow_r$ *admin*:*currentAdminOp* := ≪*None*≫)

definition *AdminTokenTear* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
AdminTokenTear =
((*iadminToken*:*adminTokenPresence* = ≪*absent*≫
) $\longrightarrow_r$ *internal*:*enclaveStatus* := ≪*enclaveQuiescent*≫)

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

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

definition *LoginAborted* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
LoginAborted = ((*internal*:*enclaveStatus* = ≪*gotAdminToken*≫) $\longrightarrow_r$ *BadAdminTokenTear*)

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

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

definition *ValidateAdminTokenOK* :: *IDStation hrel* **where**
[*upred-defs*, *tis-defs*]:
ValidateAdminTokenOK =
((*internal*:*enclaveStatus* = ≪*gotAdminToken*≫

```
   ∧ iadminToken:adminTokenPresence = ≪present≫
   ∧ @AdminTokenOK
  ) ⟶_r AdminLogon ;;
        currentScreen:screenMsg := ≪requestAdminOp≫ ;;
        internal:enclaveStatus := ≪enclaveQuiescent≫)

lemma ValidateAdminTokenOK-correct:
 ⦃IDStation⦄ ValidateAdminTokenOK ⦃IDStation⦄_u
 apply (rule IDStation-correct-intro)
  apply (hoare-wlp-auto defs: tis-defs)
 apply (hoare-wlp-auto defs: tis-defs)
 done

definition ValidateAdminTokenFail :: IDStation hrel where
[upred-defs, tis-defs]:
ValidateAdminTokenFail =
 ((internal:enclaveStatus = ≪gotAdminToken≫
   ∧ iadminToken:adminTokenPresence = ≪present≫
   ∧ (¬ @AdminTokenOK)
  ) ⟶_r currentScreen:screenMsg := ≪removeAdminToken≫ ;;
        internal:enclaveStatus := ≪waitingRemoveAdminTokenFail≫)

lemma ValidateAdminTokenFail-correct:
 ⦃IDStation⦄ ValidateAdminTokenFail ⦃IDStation⦄_u
 apply (rule IDStation-correct-intro)
  apply (hoare-wlp-auto defs: tis-defs)
 apply (simp add: IDStation-inv-def)
 apply (auto simp add: hoare-post-conj-split)
        apply (hoare-wlp-auto defs: tis-defs)
 apply (hoare-wlp-auto defs: tis-defs)
 apply (hoare-wlp-auto defs: tis-defs)
 apply (hoare-wlp-auto defs: tis-defs)
    apply (hoare-wlp-auto defs: tis-defs)
 apply (hoare-wlp-auto defs: tis-defs)
 apply (hoare-wlp-auto defs: tis-defs)
  apply (hoare-wlp-auto defs: tis-defs)
 apply (hoare-wlp-auto defs: tis-defs)
 apply (hoare-wlp-auto defs: tis-defs)
 done

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

definition FailedAdminTokenRemove :: IDStation hrel where
[upred-defs, tis-defs]:
FailedAdminTokenRemove =
 ((internal:enclaveStatus = ≪waitingRemoveAdminTokenFail≫
```

$\quad \land\ \mathit{iadminToken}{:}\mathit{adminTokenPresence} = \ll\mathit{absent}\gg$
$\quad)\ \longrightarrow_r\ \mathit{currentScreen}{:}\mathit{screenMsg} := \ll\mathit{welcomeAdmin}\gg\ ;;$
$\qquad\quad \mathit{internal}{:}\mathit{enclaveStatus} := \ll\mathit{enclaveQuiescent}\gg)$

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

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

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

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

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

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

definition $\mathit{NoOpRequest} :: \mathit{IDStation\ hrel}$ **where**
$[\mathit{upred\text{-}defs},\ \mathit{tis\text{-}defs}]$:
$\mathit{NoOpRequest} =$
$\quad ((\mathit{ikeyboard}{:}\mathit{keyedDataPresence} = \ll\mathit{absent}\gg)\ \longrightarrow_r\ \mathit{StartOpContext})$

```
definition [upred-defs, tis-defs]:
ValidateOpRequest = (ValidateOpRequestOK ∨ ValidateOpRequestFail ∨ NoOpRequest)

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

definition AdminOpStartedContext :: IDStation hrel where
[upred-defs, tis-defs]:
AdminOpStartedContext =
  ((internal:enclaveStatus = «waitingStartAdminOp»
   ∧ iadminToken:adminTokenPresence = «present»
   ) ⟶_r II)

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»
  )

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

definition TISShutdownOp :: SystemState hrel where
[upred-defs, tis-defs]:
TISShutdownOp = (UEC(ShutdownOK) ∨ UEC(ShutdownWaitingDoor))

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

definition *TISOverrideDoorLockOp* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISOverrideDoorLockOp =
(*UEC*(*OverrideDoorLockOK*)
$\vee$ *UEC*((*internal*:*enclaveStatus* = $\ll$*waitingStartAdminOp*$\gg$
$\wedge$ *admin*:*currentAdminOp* = $\ll$*Some*(*overrideLock*)$\gg$) $\longrightarrow_r$ *BadAdminLogout*))

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

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

definition *TISAdminOp* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISAdminOp = (*TISOverrideDoorLockOp* $\vee$ *TISShutdownOp* $\vee$ *TISUpdateConfigDataOp* $\vee$ *TISArchiveLog*)
definition *TISAdminLogout* :: *SystemState hrel* **where** [*upred-defs*, *tis-defs*]: *TISAdminLogout* = *false*
definition *TISIdle* :: *SystemState hrel* **where**
[*upred-defs*, *tis-defs*]:
TISIdle = *UEC*((*internal*:*status* = $\ll$*quiescent*$\gg$
$\wedge$ *internal*:*enclaveStatus* = $\ll$*enclaveQuiescent*$\gg$
$\wedge$ *iuserToken*:*userTokenPresence* = $\ll$*absent*$\gg$
$\wedge$ *iadminToken*:*adminTokenPresence* = $\ll$*absent*$\gg$
$\wedge$ *admin*:*rolePresent* = $\ll$*None*$\gg$) $\longrightarrow_r$ *II*)

11 The Whole ID Station

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

definition *InitDoorLatchAlarm* **where**
[*upred-defs*]:
InitDoorLatchAlarm =
(*DoorLatchAlarm* $\wedge$
&*currentTime* $=_u$ $\ll$*zeroTime*$\gg$ $\wedge$

&*currentDoor* $=_u$ ≪*closed*≫ ∧
&*latchTimeout* $=_u$ ≪*zeroTime*≫ ∧
&*alarmTimeout* $=_u$ ≪*zeroTime*≫)

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

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

12 Proving Security Properties

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

lemma
([&*idStation*:*doorLatchAlarm*:*currentLatch* $\mapsto_s$ ≪*locked*≫] †
(*TISReadUserToken wp* (*idStation*:*doorLatchAlarm*:*currentLatch* = ≪*unlocked*≫)))
= *false*
by (*simp add*: *tis-defs wp usubst unrest alpha*)

12.1 Proving Security Functional Requirement 1

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

definition *AdminTokenGuardOK* :: *IDStation upred* **where**
[*upred-defs*, *tis-defs*]:
AdminTokenGuardOK =
(&*iadminToken*:*currentAdminToken* $\in_u$ ≪*range*(*goodT*)≫ ∧
(∃ *t*∈≪*TokenWithValidAuth*≫ •
(≪*goodT*(*t*)≫ $=_u$ &*iadminToken*:*currentAdminToken*
∧ (∃ *c* ∈ ≪*AuthCert*≫ • ≪*Some c* = *authCert t*≫
∧ ≪*role c* = *guard*≫) $\oplus_p$ *keyStore*
))
)

lemma *admin-unlock*:
[&*idStation*:*doorLatchAlarm*:*currentLatch* $\mapsto_s$ ≪*locked*≫]
† ((*TISAdminOp* ;; *TISUpdate*) *wp* (*realWorld*:*controlled*:*latch* = ≪*unlocked*≫)) =
((&*idStation*:*internal*:*enclaveStatus* $=_u$ ≪*waitingStartAdminOp*≫ ∧ &*idStation*:*iadminToken*:*adminToken*
$=_u$ ≪*present*≫) ∧
&*idStation*:*admin*:*currentAdminOp* $=_u$ ≪*Some overrideLock*≫ ∧ &*idStation*:*admin*:*rolePresent*
$\neq_u$ $None_u$ ∧ &*idStation*:*admin*:*currentAdminOp* $\neq_u$ $None_u$)
by (*simp add*: *tis-defs wp usubst unrest alpha*)

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

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

abbreviation *FSFR1* ≡ ‘(*IDStation-inv*) $\oplus_p$ *idStation* ∧
[&*idStation*:*doorLatchAlarm*:*currentLatch* $\mapsto_s$ ≪*locked*≫]
† ((*TISUserEntryOp* ;; *TISUpdate*) *wp* (*realWorld*:*controlled*:*latch* = ≪*unlocked*≫))
⇒ ((*UserTokenOK* ∧ *FingerOK*) ∨ (*UserTokenWithOKAuthCert*)) $\oplus_p$ *idStation*‘

lemma *FSFR1-proof*:
‘(*IDStation-inv*) $\oplus_p$ *idStation* ∧
[&*idStation*:*doorLatchAlarm*:*currentLatch* $\mapsto_s$ ≪*locked*≫]
† ((*TISUserEntryOp* ;; *TISUpdate*) *wp* (*realWorld*:*controlled*:*latch* = ≪*unlocked*≫))
⇒ ((*UserTokenOK* ∧ *FingerOK*) ∨ (*UserTokenWithOKAuthCert*)) $\oplus_p$ *idStation*‘
apply (*simp add*: *user-unlock*)
apply (*rel-auto*)
done

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.

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

definition *AlarmInv* :: *SystemState upred* **where**
[*upred-defs*, *tis-defs*]:
AlarmInv = (*realWorld*:*controlled*:*latch* = ≪*locked*≫ ∧

$idStation{:}doorLatchAlarm{:}currentDoor = \ll dopen \gg \wedge$
$idStation{:}doorLatchAlarm{:}currentTime \geq idStation{:}doorLatchAlarm{:}alarmTimeout$
$\Rightarrow realWorld{:}controlled{:}alarm = \ll alarming \gg)_e$

lemma $\{\{realWorld{:}controlled{:}latch = \ll locked \gg \wedge$
$idStation{:}doorLatchAlarm{:}currentDoor = \ll dopen \gg \wedge$
$idStation{:}doorLatchAlarm{:}currentTime \geq idStation{:}doorLatchAlarm{:}alarmTimeout$
$\wedge$
$(@DoorLatchAlarm \oplus_p idStation{:}doorLatchAlarm)\}\}$ $TISUpdate\{\{realWorld{:}controlled{:}alarm$
$= \ll alarming \gg\}\}$
oops

end