

CoSMedis: A confidentiality-verified distributed social media platform

Thomas Bauereiss Andrei Popescu

February 6, 2026

Abstract

This entry contains the confidentiality verification of the (functional kernel of) the CoSMedis distributed social media platform presented in [3]. CoSMedis is a multi-node extension the CoSMed prototype social media platform [2, 4, 6]. The confidentiality properties are formalized as instances of BD Security [7, 8]. The lifting of confidentiality properties from single nodes to the entire CoSMedis network is performed using compositionality and transport theorems for BD Security, which are described in [3] and formalized in the AFP entry [5].

Contents

1	Introduction	3
2	Preliminaries	5
2.1	The basic types	5
2.2	Identifiers	7
3	The CoSMedis single node specification	8
3.1	The state	9
3.2	The actions	10
3.2.1	Initialization of the system	10
3.2.2	Starting action	11
3.2.3	Creation actions	11
3.2.4	Deletion (removal) actions	13
3.2.5	Updating actions	13
3.2.6	Reading actions	14
3.2.7	Listing actions	17
3.2.8	Actions of communication with other APIs	20
3.3	The step function	23
3.4	Code generation	33
4	The CoSMedis network of communicating nodes	34

5	Safety properties	38
6	Post confidentiality	41
6.1	Confidentiality for a secret issuer node	42
6.1.1	Observation setup	42
6.1.2	Unwinding helper lemmas and definitions	45
6.1.3	Value setup	52
6.1.4	Issuer declassification bound	54
6.1.5	Unwinding proof	55
6.2	Confidentiality for a secret receiver node	57
6.2.1	Observation setup	57
6.2.2	Unwinding helper definitions and lemmas	60
6.2.3	Value setup	65
6.2.4	Declassification bound	67
6.2.5	Unwinding proof	67
6.3	Confidentiality for the (binary) issuer-receiver composition . .	69
6.4	Confidentiality for the N-ary composition	73
6.5	Variation with dynamic declassification trigger	77
6.5.1	Issuer value setup	77
6.5.2	Issuer declassification bound	81
6.5.3	Issuer unwinding proof	83
6.5.4	Confidentiality for the (binary) issuer-receiver compo- sition	86
6.5.5	Confidentiality for the N-ary composition	90
6.6	Variation with multiple independent secret posts	94
6.6.1	Issuer observation setup	94
6.6.2	Issuer value setup	97
6.6.3	Issuer declassification bound	102
6.6.4	Issuer unwinding proof	103
6.6.5	Receiver observation setup	106
6.6.6	Receiver value setup	109
6.6.7	Receiver declassification bound	111
6.6.8	Receiver unwinding proof	111
6.6.9	Confidentiality for the N-ary composition	113
6.6.10	Composition of confidentiality guarantees for different posts	116
7	Friendship status confidentiality	121
7.1	Observation setup	122
7.2	Unwinding helper definitions and lemmas	123
7.3	Dynamic declassification trigger	131
7.4	Value Setup	133
7.5	Declassification bound	135
7.6	Unwinding proof	136

7.7	Confidentiality for the N-ary composition	138
8	Friendship request confidentiality	140
8.1	Value setup	140
8.2	Declassification bound	144
8.3	Unwinding proof	145
8.4	Confidentiality for the N-ary composition	150
9	Remote (outer) friendship status confidentiality	151
9.1	Issuer node	152
9.1.1	Observation setup	152
9.1.2	Unwinding helper definitions and lemmas	155
9.1.3	Dynamic declassification trigger	161
9.1.4	Value setup	162
9.1.5	Declassification bound	164
9.1.6	Unwinding proof	165
9.2	Receiver nodes	167
9.2.1	Observation setup	167
9.2.2	Unwinding helper definitions and lemmas	169
9.2.3	Value Setup	174
9.2.4	Declassification bound	175
9.2.5	Unwinding proof	176
9.3	Confidentiality for the N-ary composition	178

1 Introduction

This entry contains the confidentiality verification of the (functional kernel of) the CoSMedis distributed social media platform presented in [3].

CoSMedis [2, 4] (whose formalization is described in a separate AFP entry, [6]) is a simple Facebook-style social media platform where users can register, create posts and establish friendship relationships. CoSMedis is a multi-node distributed extension of CoSMedis that follows a Diaspora-style scheme [1]: Different nodes can be deployed independently at different internet locations. The admins of any two nodes can initiate a protocol to connect these nodes, after which the users of one node can establish friendship relationships and share data with users of the other. Thus, a node of CoSMedis consists of CoSMedis plus actions for connecting nodes and cross-node post sharing and friending.

After this introduction and a section on technical preliminaries/prerequisites, this document presents the specification of a single CoSMedis node, followed by a specification of the entire CoSMedis network, consisting of a finite but unbounded number of mutually communicating nodes.

Next is a section on proved safety properties about the system—essentially, some system invariants that are needed in the proofs of confidentiality.

Next come the main sections, those dealing with confidentiality. The confidentiality properties of CoSMedis (like those of CoSMed) are formalized as instances of BD Security [7], a general confidentiality verification framework that has been formalized in the AFP entry [8]. They cover confidentiality aspects about:

- posts
- friendship status (whether or not two users are friends)
- friendship request status (whether or not a user has submitted a friendship request to another user)

Each of these types of confidentiality properties have dedicated sections (and corresponding folders in the formalization) with self-explanatory names.

In addition to the properties lifted from CoSMed, we also prove the confidentiality of remote friendships (i.e., friendship relations established between users at different nodes), which is a new feature of CoSMedis compared to CoSMed. This has a dedicated section/folder as well.

The properties are first proved for individual nodes, and then they are lifted to the entire CoSMedis network using compositionality and transport theorems for BD Security, which are described in [3] and formalized in the AFP entry [5].

All the sections on confidentiality follow a similar structure (with some variations), as can be seen in the names of their subsections. There are subsections for:

- defining the observation and secrecy infrastructures¹
- defining the declassification bounds and triggers²
- the main results, namely:
 - the BD Security instance proved by unwinding for an individual node
 - the lifting of this result from a CoSMedis node to an entire network using the n -ary compositionality theorem for BD security

In the case of post confidentiality and outer friend confidentiality, the secret may be communicated from the issuer to other nodes. For this purpose, we formalize corresponding local security properties for the issuer and the receiver nodes, contained in separate subsections with names containing “Issuer” and “Receiver”, respectively.

¹NB: The secrets are called “values” in the formalization.

²In many cases, the CoSMed and CoSMedis bounds incorporate the triggers as well—see [3, Appendix C] and [4, Section 3.3].

In the case of post confidentiality, we have a version with static declassification trigger and one with dynamic trigger. (The dynamic version is described in [3, Appendix C].) Moreover, in the section on “independent posts”, we formalize the lifting of the confidentiality of one given (arbitrary but fixed) post to the confidentiality of two posts of arbitrary nodes of the network (as described in [3, Appendix E]).

As a matter of notation, this formalization (similarly to all our AFP formalizations involving BD security) differs from the paper [3] (and on most papers on CoSMed, CoSMedis or CoCon) in that the secrets are called “values” (and consequently the type of secrets is denoted by “value”), and are ranged over by v rather than s . On the other hand, we use s (rather than σ) to range over states. Moreover, the formalization uses the following notations for the various BD security components:

- φ for the secret discriminator for isSec
- f for the secret selector getSec
- γ for the observation discriminator isObs
- g for the observation selector getObs

Finally, what the paper [3] refers to as “nodes” are referred in the formalization as “APIs”. (The “API” terminology is justified by the fact that nodes behave similarly to a form communicating APIs.)

2 Preliminaries

```
theory Prelim
  imports
    Fresh-Identifiers.Fresh-String
    Bounded-Deducibility-Security.Trivia
begin
```

2.1 The basic types

```
definition emptyStr = STR ""
```

```
datatype name = Nam String.literal
definition emptyName  $\equiv$  Nam emptyStr
datatype inform = Info String.literal
definition emptyInfo  $\equiv$  Info emptyStr
```

```
datatype user = Ustr name inform
fun nameUser where nameUser (Ustr name info) = name
```

```

fun infoUser where infoUser (Usr name info) = info
definition emptyUser ≡ Usr emptyName emptyInfo

typedecl raw-data
code-printing type-constructor raw-data → (Scala) java.io.File

```

```

datatype img = emptyImg | Imag raw-data

```

```

datatype vis = Vsb String.literal

```

```

abbreviation FriendV ≡ Vsb (STR "friend")

```

```

abbreviation PublicV ≡ Vsb (STR "public")
fun stringOfVis where stringOfVis (Vsb str) = str

```

```

datatype title = Tit String.literal
definition emptyTitle ≡ Tit emptyStr
datatype text = Txt String.literal
definition emptyText ≡ Txt emptyStr

```

```

datatype post = Pst title text img

```

```

fun titlePost where titlePost (Pst title text img) = title
fun textPost where textPost (Pst title text img) = text
fun imgPost where imgPost (Pst title text img) = img

```

```

fun setTitlePost where setTitlePost (Pst title text img) title' = Pst title' text img
fun setTextPost where setTextPost (Pst title text img) text' = Pst title text' img
fun setImgPost where setImgPost (Pst title text img) img' = Pst title text img'

```

```

definition emptyPost :: post where
emptyPost ≡ Pst emptyTitle emptyText emptyImg

```

```

lemma titlePost-emptyPost[simp]: titlePost emptyPost = emptyTitle
and textPost-emptyPost[simp]: textPost emptyPost = emptyText
and imgPost-emptyPost[simp]: imgPost emptyPost = emptyImg

```

⟨proof⟩

```

lemma set-get-post[simp]:
titlePost (setTitlePost ntc title) = title
titlePost (setTextPost ntc text) = titlePost ntc
titlePost (setImgPost ntc img) = titlePost ntc

```

textPost (*setTitlePost ntc title*) = *textPost ntc*
textPost (*setTextPost ntc text*) = *text*
textPost (*setImgPost ntc img*) = *textPost ntc*

imgPost (*setTitlePost ntc title*) = *imgPost ntc*
imgPost (*setTextPost ntc text*) = *imgPost ntc*
imgPost (*setImgPost ntc img*) = *img*

<proof>

lemma *setTextPost-absorb[simp]*:
setTitlePost (*setTitlePost pst tit*) *tit1* = *setTitlePost pst tit1*
setTextPost (*setTextPost pst txt*) *txt1* = *setTextPost pst txt1*
setImgPost (*setImgPost pst img*) *img1* = *setImgPost pst img1*

<proof>

datatype *password* = *Psw String.literal*
definition *emptyPass* ≡ *Psw emptyStr*

datatype *salt* = *Slt String.literal*
definition *emptySalt* ≡ *Slt emptyStr*

datatype *requestInfo* = *ReqInfo String.literal*
definition *emptyRequestInfo* ≡ *ReqInfo emptyStr*

2.2 Identifiers

datatype *apiID* = *Aid String.literal*
datatype *userID* = *Uid String.literal*
datatype *postID* = *Pid String.literal*

definition *emptyApiID* ≡ *Aid emptyStr*
definition *emptyUserID* ≡ *Uid emptyStr*
definition *emptyPostID* ≡ *Pid emptyStr*

fun *apiIDAsStr* **where** *apiIDAsStr* (*Aid str*) = *str*

definition *getFreshApiID* *apiIDs* ≡ *Aid (fresh (set (map apiIDAsStr apiIDs)) (STR "1"))*

lemma *ApiID-apiIDAsStr[simp]*: $Aid (apiIDAsStr\ apiID) = apiID$
{proof}

lemma *member-apiIDAsStr-iff[simp]*: $str \in apiIDAsStr \text{ ' } apiIDs \longleftrightarrow Aid\ str \in apiIDs$
{proof}

lemma *getFreshApiID*: $\neg getFreshApiID\ apiIDs \in \in apiIDs$
{proof}

fun *userIDAsStr* **where** *userIDAsStr* (*Uid* *str*) = *str*

definition *getFreshUserID* *userIDs* $\equiv Uid (fresh (set (map\ userIDAsStr\ userIDs)))$
(*STR* "2")

lemma *UserID-userIDAsStr[simp]*: $Uid (userIDAsStr\ userID) = userID$
{proof}

lemma *member-userIDAsStr-iff[simp]*: $str \in userIDAsStr \text{ ' } (set\ userIDs) \longleftrightarrow Uid\ str \in \in userIDs$
{proof}

lemma *getFreshUserID*: $\neg getFreshUserID\ userIDs \in \in userIDs$
{proof}

fun *postIDAsStr* **where** *postIDAsStr* (*Pid* *str*) = *str*

definition *getFreshPostID* *postIDs* $\equiv Pid (fresh (set (map\ postIDAsStr\ postIDs)))$
(*STR* "3")

lemma *PostID-postIDAsStr[simp]*: $Pid (postIDAsStr\ postID) = postID$
{proof}

lemma *member-postIDAsStr-iff[simp]*: $str \in postIDAsStr \text{ ' } (set\ postIDs) \longleftrightarrow Pid\ str \in \in postIDs$
{proof}

lemma *getFreshPostID*: $\neg getFreshPostID\ postIDs \in \in postIDs$
{proof}

end

3 The CoSMedis single node specification

This is the specification of a CoSMedis node, as described in Sections II and IV.B of [3]. NB: What that paper refers to as "nodes" are referred in this

formalization as "APIs".

A CoSMedis node extends CoSMed [2, 4, 6] with inter-node communication actions.

```
theory System-Specification
  imports
    Prelim
    Bounded-Deducibility-Security.IO-Automaton
begin
```

An aspect not handled in this specification is the uniqueness of the node IDs. These are assumed to be handled externally as follows: a node ID is an URI, and therefore is unique.

```
declare List.insert[simp]
```

3.1 The state

```
record state =
  admin :: userID

  pendingUReqs :: userID list
  userReq :: userID  $\Rightarrow$  requestInfo
  userIDs :: userID list
  user :: userID  $\Rightarrow$  user
  pass :: userID  $\Rightarrow$  password

  pendingFReqs :: userID  $\Rightarrow$  userID list
  friendReq :: userID  $\Rightarrow$  userID  $\Rightarrow$  requestInfo
  friendIDs :: userID  $\Rightarrow$  userID list

  sentOuterFriendIDs :: userID  $\Rightarrow$  (apiID  $\times$  userID) list
  recvOuterFriendIDs :: userID  $\Rightarrow$  (apiID  $\times$  userID) list

  postIDs :: postID list
  post :: postID  $\Rightarrow$  post
  owner :: postID  $\Rightarrow$  userID
  vis :: postID  $\Rightarrow$  vis

  pendingSApiReqs :: apiID list
  sApiReq :: apiID  $\Rightarrow$  requestInfo
  serverApiIDs :: apiID list

  serverPass :: apiID  $\Rightarrow$  password
  outerPostIDs :: apiID  $\Rightarrow$  postID list
  outerPost :: apiID  $\Rightarrow$  postID  $\Rightarrow$  post
  outerOwner :: apiID  $\Rightarrow$  postID  $\Rightarrow$  userID
  outerVis :: apiID  $\Rightarrow$  postID  $\Rightarrow$  vis
```

pendingCApiReqs :: *apiID list*
cApiReq :: *apiID* ⇒ *requestInfo*
clientApiIDs :: *apiID list*

clientPass :: *apiID* ⇒ *password*
sharedWith :: *postID* ⇒ (*apiID* × *bool*) *list*

definition *IDsOK* :: *state* ⇒ *userID list* ⇒ *postID list* ⇒ (*apiID* × *postID list*)
list ⇒ *apiID list* ⇒ *bool*

where

IDsOK s uIDs pIDs saID-pIDs-s caIDs ≡
list-all (λ *uID*. *uID* ∈∈ *userIDs s*) *uIDs* ∧
list-all (λ *pID*. *pID* ∈∈ *postIDs s*) *pIDs* ∧
list-all (λ (*aID*,*pIDs*). *aID* ∈∈ *serverApiIDs s* ∧
list-all (λ *pID*. *pID* ∈∈ *outerPostIDs s aID*) *pIDs*) *saID-pIDs-s* ∧
list-all (λ *aID*. *aID* ∈∈ *clientApiIDs s*) *caIDs*

3.2 The actions

3.2.1 Initialization of the system

definition *istate* :: *state*

where

istate ≡

(
admin = *emptyUserID*,

pendingUReqs = [],
userReq = (λ *uID*. *emptyRequestInfo*),
userIDs = [],
user = (λ *uID*. *emptyUser*),
pass = (λ *uID*. *emptyPass*),

pendingFReqs = (λ *uID*. []),
friendReq = (λ *uID uID'*. *emptyRequestInfo*),
friendIDs = (λ *uID*. []),

sentOuterFriendIDs = (λ *uID*. []),
recvOuterFriendIDs = (λ *uID*. []),

postIDs = [],
post = (λ *papID*. *emptyPost*),

```

owner = (λ pID. emptyUserID),
vis = (λ pID. FriendV),

pendingSApiReqs = [],
sApiReq = (λ aID. emptyRequestInfo),
serverApiIDs = [],
serverPass = (λ aID. emptyPass),
outerPostIDs = (λ aID. []),
outerPost = (λ aID papID. emptyPost),
outerOwner = (λ aID papID. emptyUserID),
outerVis = (λ aID pID. FriendV),

pendingCApiReqs = [],
cApiReq = (λ aID. emptyRequestInfo),
clientApiIDs = [],
clientPass = (λ aID. emptyPass),
sharedWith = (λ pID. [])

```

3.2.2 Starting action

definition *startSys* ::

state ⇒ *userID* ⇒ *password* ⇒ *state*

where

```

startSys s uID p ≡
s (|admin := uID,
   |userIDs := [uID],
   |user := (user s) (uID := emptyUser),
   |pass := (pass s) (uID := p))

```

definition *e-startSys* :: *state* ⇒ *userID* ⇒ *password* ⇒ *bool*

where

e-startSys s uID p ≡ *userIDs* s = []

3.2.3 Creation actions

definition *createNUReq* :: *state* ⇒ *userID* ⇒ *requestInfo* ⇒ *state*

where

```

createNUReq s uID reqInfo ≡
s (|pendingUReqs := pendingUReqs s @ [uID],
   |userReq := (userReq s)(uID := reqInfo)

```

)

definition *e-createNUReq* :: *state* ⇒ *userID* ⇒ *requestInfo* ⇒ *bool*

where

```

e-createNUReq s uID requestInfo ≡
admin s ∈∈ userIDs s ∧ ¬ uID ∈∈ userIDs s ∧ ¬ uID ∈∈ pendingUReqs s

```

definition $createUser :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow password \Rightarrow state$

where

$createUser\ s\ uID\ p\ uID'\ p' \equiv$
 $s\ (\backslash userIDs := uID' \# (userIDs\ s),$
 $\quad user := (user\ s)\ (uID' := emptyUser),$
 $\quad pass := (pass\ s)\ (uID' := p'),$
 $\quad pendingUReqs := remove1\ uID'\ (pendingUReqs\ s),$
 $\quad userReq := (userReq\ s)(uID := emptyRequestInfo)\)$

definition $e-createUser :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow password \Rightarrow bool$

where

$e-createUser\ s\ uID\ p\ uID'\ p' \equiv$
 $IDsOK\ s\ [uID]\ []\ []\ [] \wedge pass\ s\ uID = p \wedge uID = admin\ s \wedge uID' \in \in pendingUReqs\ s$

definition $createPost :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow state$

where

$createPost\ s\ uID\ p\ pID \equiv$
 $s\ (\backslash postIDs := pID \# postIDs\ s,$
 $\quad post := (post\ s)\ (pID := emptyPost),$
 $\quad owner := (owner\ s)\ (pID := uID)\)$

definition $e-createPost :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow bool$

where

$e-createPost\ s\ uID\ p\ pID \equiv$
 $IDsOK\ s\ [uID]\ []\ []\ [] \wedge pass\ s\ uID = p \wedge$
 $\neg pID \in \in postIDs\ s$

definition $createFriendReq :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow requestInfo \Rightarrow state$

where

$createFriendReq\ s\ uID\ p\ uID'\ req \equiv$
 $let\ pfr = pendingFReqs\ s\ in$
 $s\ (\backslash pendingFReqs := pfr\ (uID' := pfr\ uID'\ @\ [uID]),$
 $\quad friendReq := fun-upd2\ (friendReq\ s)\ uID\ uID'\ req)$

definition $e-createFriendReq :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow requestInfo \Rightarrow bool$

where

$e-createFriendReq\ s\ uID\ p\ uID'\ req \equiv$
 $IDsOK\ s\ [uID, uID']\ []\ []\ [] \wedge pass\ s\ uID = p \wedge$
 $\neg uID \in \in pendingFReqs\ s\ uID' \wedge \neg uID \in \in friendIDs\ s\ uID'$

definition $createFriend :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow state$
where
 $createFriend\ s\ uID\ p\ uID' \equiv$
 $let\ fr = friendIDs\ s;\ pfr = pendingFReqs\ s\ in$
 $s\ (\{friendIDs := fr\ (uID := fr\ uID\ @\ [uID']),\ uID' := fr\ uID'\ @\ [uID]),$
 $pendingFReqs := pfr\ (uID := remove1\ uID'\ (pfr\ uID),\ uID' := remove1\ uID$
 $(pfr\ uID')),$
 $friendReq := fun-upd2\ (fun-upd2\ (friendReq\ s)\ uID'\ uID\ emptyRequestInfo)$
 $uID\ uID'\ emptyRequestInfo)$

definition $e-createFriend :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$
where
 $e-createFriend\ s\ uID\ p\ uID' \equiv$
 $IDsOK\ s\ [uID, uID']\ []\ [] \wedge pass\ s\ uID = p \wedge$
 $uID' \in \in\ pendingFReqs\ s\ uID$

3.2.4 Deletion (removal) actions

definition $deleteFriend :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow state$
where
 $deleteFriend\ s\ uID\ p\ uID' \equiv$
 $let\ fr = friendIDs\ s\ in$
 $s\ (\{friendIDs := fr\ (uID := removeAll\ uID'\ (fr\ uID),\ uID' := removeAll\ uID\ (fr$
 $uID'))\})$

definition $e-deleteFriend :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$
where
 $e-deleteFriend\ s\ uID\ p\ uID' \equiv$
 $IDsOK\ s\ [uID, uID']\ []\ [] \wedge pass\ s\ uID = p \wedge$
 $uID' \in \in\ friendIDs\ s\ uID$

3.2.5 Updating actions

definition $updateUser :: state \Rightarrow userID \Rightarrow password \Rightarrow password \Rightarrow name \Rightarrow$
 $inform \Rightarrow state$
where
 $updateUser\ s\ uID\ p\ p'\ name\ info \equiv$
 $s\ (\{user := (user\ s)\ (uID := Usr\ name\ info),$
 $pass := (pass\ s)\ (uID := p')\})$

definition $e-updateUser :: state \Rightarrow userID \Rightarrow password \Rightarrow password \Rightarrow name \Rightarrow$
 $inform \Rightarrow bool$
where
 $e-updateUser\ s\ uID\ p\ p'\ name\ info \equiv$
 $IDsOK\ s\ [uID]\ []\ [] \wedge pass\ s\ uID = p$

definition $updatePost :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow post \Rightarrow state$

where

$updatePost\ s\ uID\ p\ pID\ pst \equiv$
 $let\ sW = sharedWith\ s\ in$
 $s\ (\!post := (post\ s)\ (pID := pst),$
 $sharedWith := sW\ (pID := map\ (\lambda\ (aID,-).\ (aID,False))\ (sW\ pID))\!)$

definition $e-updatePost :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow post \Rightarrow bool$

where

$e-updatePost\ s\ uID\ p\ pID\ pst \equiv$
 $IDsOK\ s\ [uID]\ [pID]\ []\ [] \wedge pass\ s\ uID = p \wedge$
 $owner\ s\ pID = uID$

definition $updateVisPost :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow vis \Rightarrow state$

where

$updateVisPost\ s\ uID\ p\ pID\ vs \equiv$
 $s\ (\!vis := (vis\ s)\ (pID := vs)\!)$

definition $e-updateVisPost :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow vis \Rightarrow bool$

where

$e-updateVisPost\ s\ uID\ p\ pID\ vs \equiv$
 $IDsOK\ s\ [uID]\ [pID]\ []\ [] \wedge pass\ s\ uID = p \wedge$
 $owner\ s\ pID = uID \wedge vs \in \{FriendV, PublicV\}$

3.2.6 Reading actions

definition $readNUReq :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow requestInfo$

where

$readNUReq\ s\ uID\ p\ uID' \equiv userReq\ s\ uID'$

definition $e-readNUReq :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$

where

$e-readNUReq\ s\ uID\ p\ uID' \equiv$
 $IDsOK\ s\ [uID]\ []\ [] \wedge pass\ s\ uID = p \wedge$
 $uID = admin\ s \wedge uID' \in \in pendingUReqs\ s$

definition $readUser :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow name$

where

$readUser\ s\ uID\ p\ uID' \equiv nameUser\ (user\ s\ uID')$

definition $e-readUser :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$

where

$e-readUser\ s\ uID\ p\ uID' \equiv$
 $IDsOK\ s\ [uID,uID']\ []\ [] \wedge pass\ s\ uID = p$

definition $readAmIAdmin :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

where

$readAmIAdmin\ s\ uID\ p \equiv uID = admin\ s$

definition $e\text{-readAmIAdmin} :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

where

$e\text{-readAmIAdmin } s \ uID \ p \equiv$
 $IDsOK \ s \ [uID] \ [] \ [] \wedge \text{pass } s \ uID = p$

definition $readPost :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow post$

where

$readPost \ s \ uID \ p \ pID \equiv post \ s \ pID$

definition $e\text{-readPost} :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow bool$

where

$e\text{-readPost } s \ uID \ p \ pID \equiv$
 $IDsOK \ s \ [uID] \ [pID] \ [] \ [] \wedge \text{pass } s \ uID = p \wedge$
 $(owner \ s \ pID = uID \vee uID \in\in friendIDs \ s \ (owner \ s \ pID) \vee vis \ s \ pID = PublicV)$

definition $readOwnerPost :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow userID$

where

$readOwnerPost \ s \ uID \ p \ pID \equiv owner \ s \ pID$

definition $e\text{-readOwnerPost} :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow bool$

where

$e\text{-readOwnerPost } s \ uID \ p \ pID \equiv$
 $IDsOK \ s \ [uID] \ [pID] \ [] \ [] \wedge \text{pass } s \ uID = p \wedge$
 $(admin \ s = uID \vee owner \ s \ pID = uID \vee uID \in\in friendIDs \ s \ (owner \ s \ pID) \vee$
 $vis \ s \ pID = PublicV)$

definition $readVisPost :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow vis$

where

$readVisPost \ s \ uID \ p \ pID \equiv vis \ s \ pID$

definition $e\text{-readVisPost} :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow bool$

where

$e\text{-readVisPost } s \ uID \ p \ pID \equiv$
 $IDsOK \ s \ [uID] \ [pID] \ [] \ [] \wedge \text{pass } s \ uID = p \wedge$
 $(admin \ s = uID \vee owner \ s \ pID = uID \vee uID \in\in friendIDs \ s \ (owner \ s \ pID) \vee$
 $vis \ s \ pID = PublicV)$

definition $readOPost :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow postID \Rightarrow post$

where

$readOPost \ s \ uID \ p \ aID \ pID \equiv outerPost \ s \ aID \ pID$

definition $e\text{-readOPost} :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow postID \Rightarrow bool$

where

$e\text{-readOPost } s \ uID \ p \ aID \ pID \equiv$

$IDsOK\ s\ [uID]\ []\ [(aID,[pID])]\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
 $(admin\ s = uID\ \vee\ (aID,outerOwner\ s\ aID\ pID) \in\in\ recvOuterFriendIDs\ s\ uID\ \vee$
 $outerVis\ s\ aID\ pID = PublicV)$

definition $readOwnerOPost :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow postID \Rightarrow userID$

where

$readOwnerOPost\ s\ uID\ p\ aID\ pID \equiv outerOwner\ s\ aID\ pID$

definition $e-readOwnerOPost :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow postID \Rightarrow bool$

where

$e-readOwnerOPost\ s\ uID\ p\ aID\ pID \equiv$

$IDsOK\ s\ [uID]\ []\ [(aID,[pID])]\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
 $(admin\ s = uID\ \vee\ (aID,outerOwner\ s\ aID\ pID) \in\in\ recvOuterFriendIDs\ s\ uID\ \vee$
 $outerVis\ s\ aID\ pID = PublicV)$

definition $readVisOPost :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow postID \Rightarrow vis$
where

$readVisOPost\ s\ uID\ p\ aID\ pID \equiv outerVis\ s\ aID\ pID$

definition $e-readVisOPost :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow postID \Rightarrow bool$

where

$e-readVisOPost\ s\ uID\ p\ aID\ pID \equiv$

$let\ post = outerPost\ s\ aID\ pID\ in$
 $IDsOK\ s\ [uID]\ []\ [(aID,[pID])]\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
 $(admin\ s = uID\ \vee\ (aID,outerOwner\ s\ aID\ pID) \in\in\ recvOuterFriendIDs\ s\ uID\ \vee$
 $outerVis\ s\ aID\ pID = PublicV)$

definition $readFriendReqToMe :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow requestInfo$

where

$readFriendReqToMe\ s\ uID\ p\ uID' \equiv friendReq\ s\ uID'\ uID$

definition $e-readFriendReqToMe :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$
where

$e-readFriendReqToMe\ s\ uID\ p\ uID' \equiv$

$IDsOK\ s\ [uID,uID']\ []\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
 $uID' \in\in\ pendingFReqs\ s\ uID$

definition $readFriendReqFromMe :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow requestInfo$

where

$readFriendReqFromMe\ s\ uID\ p\ uID' \equiv friendReq\ s\ uID\ uID'$

definition $e-readFriendReqFromMe :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$

where

$e-readFriendReqFromMe\ s\ uID\ p\ uID' \equiv$
 $IDsOK\ s\ [uID, uID']\ []\ [] \wedge pass\ s\ uID = p \wedge$
 $uID \in \in pendingFReqs\ s\ uID'$

definition $readSApiReq :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow requestInfo$

where

$readSApiReq\ s\ uID\ p\ uID' \equiv sApiReq\ s\ uID'$

definition $e-readSApiReq :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow bool$

where

$e-readSApiReq\ s\ uID\ p\ uID' \equiv$
 $IDsOK\ s\ [uID]\ []\ [] \wedge pass\ s\ uID = p \wedge$
 $uID = admin\ s \wedge uID' \in \in pendingSApiReqs\ s$

definition $readCApiReq :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow requestInfo$

where

$readCApiReq\ s\ uID\ p\ uID' \equiv cApiReq\ s\ uID'$

definition $e-readCApiReq :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow bool$

where

$e-readCApiReq\ s\ uID\ p\ uID' \equiv$
 $IDsOK\ s\ [uID]\ []\ [] \wedge pass\ s\ uID = p \wedge$
 $uID = admin\ s \wedge uID' \in \in pendingCApiReqs\ s$

3.2.7 Listing actions

definition $listPendingUReqs :: state \Rightarrow userID \Rightarrow password \Rightarrow userID\ list$

where

$listPendingUReqs\ s\ uID\ p \equiv pendingUReqs\ s$

definition $e-listPendingUReqs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

where

$e-listPendingUReqs\ s\ uID\ p \equiv$
 $IDsOK\ s\ [uID]\ []\ [] \wedge pass\ s\ uID = p \wedge uID = admin\ s$

definition $listAllUsers :: state \Rightarrow userID \Rightarrow password \Rightarrow userID\ list$

where

$listAllUsers\ s\ uID\ p \equiv userIDs\ s$

definition $e-listAllUsers :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$

where

$e\text{-listAllUsers } s \text{ } uID \text{ } p \equiv IDsOK \ s \ [uID] \ [] \ [] \wedge \text{pass } s \ uID = p$

definition $listFriends :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow userID \text{ list}$
where
 $listFriends \ s \ uID \ p \ uID' \equiv friendIDs \ s \ uID'$

definition $e\text{-listFriends} :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$
where
 $e\text{-listFriends } s \ uID \ p \ uID' \equiv$
 $IDsOK \ s \ [uID, uID'] \ [] \ [] \wedge \text{pass } s \ uID = p \wedge$
 $(uID = uID' \vee uID \in\in friendIDs \ s \ uID')$

definition $listSentOuterFriends :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow (apiID \times userID) \text{ list}$
where
 $listSentOuterFriends \ s \ uID \ p \ uID' \equiv sentOuterFriendIDs \ s \ uID'$

definition $e\text{-listSentOuterFriends} :: state \Rightarrow userID \Rightarrow password \Rightarrow userID \Rightarrow bool$
where
 $e\text{-listSentOuterFriends } s \ uID \ p \ uID' \equiv$
 $IDsOK \ s \ [uID, uID'] \ [] \ [] \wedge \text{pass } s \ uID = p \wedge$
 $(uID = uID' \vee uID \in\in friendIDs \ s \ uID')$

definition $listRecvOuterFriends :: state \Rightarrow userID \Rightarrow password \Rightarrow (apiID \times userID) \text{ list}$
where
 $listRecvOuterFriends \ s \ uID \ p \equiv recvOuterFriendIDs \ s \ uID$

definition $e\text{-listRecvOuterFriends} :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
 $e\text{-listRecvOuterFriends } s \ uID \ p \equiv$
 $IDsOK \ s \ [uID] \ [] \ [] \wedge \text{pass } s \ uID = p$

definition $listInnerPosts :: state \Rightarrow userID \Rightarrow password \Rightarrow (userID \times postID) \text{ list}$
where
 $listInnerPosts \ s \ uID \ p \equiv$
 $[(owner \ s \ pID, pID).$
 $\quad pID \leftarrow postIDs \ s,$
 $\quad vis \ s \ pID \neq FriendV \vee uID \in\in friendIDs \ s \ (owner \ s \ pID) \vee uID = owner \ s$
 $\quad pID$
 $]$

definition $e\text{-listInnerPosts} :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
 $e\text{-listInnerPosts } s \ uID \ p \equiv IDsOK \ s \ [uID] \ [] \ [] \wedge \text{pass } s \ uID = p$

definition $listOuterPosts :: state \Rightarrow userID \Rightarrow password \Rightarrow (apiID \times postID) list$
where
 $listOuterPosts s uID p \equiv$
 $[(saID, pID).$
 $saID \leftarrow serverApiIDs s,$
 $pID \leftarrow outerPostIDs s saID,$
 $outerVis s saID pID = PublicV \vee (saID, outerOwner s saID pID) \in \in recvOuter-$
 $FriendIDs s uID$
 $]$

definition $e-listOuterPosts :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
 $e-listOuterPosts s uID p \equiv IDsOK s [uID] [] [] \wedge pass s uID = p$

definition $listClientsPost :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow (apiID \times bool) list$
where
 $listClientsPost s uID p pID \equiv sharedWith s pID$

definition $e-listClientsPost :: state \Rightarrow userID \Rightarrow password \Rightarrow postID \Rightarrow bool$
where
 $e-listClientsPost s uID p pID \equiv$
 $IDsOK s [uID] [pID] [] [] \wedge pass s uID = p \wedge uID = admin s$

definition $listPendingSApiReqs :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID list$
where
 $listPendingSApiReqs s uID p \equiv pendingSApiReqs s$

definition $e-listPendingSApiReqs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
 $e-listPendingSApiReqs s uID p \equiv$
 $IDsOK s [uID] [] [] \wedge pass s uID = p \wedge uID = admin s$

definition $listServerAPIs :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID list$
where
 $listServerAPIs s uID p \equiv serverApiIDs s$

definition $e-listServerAPIs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
 $e-listServerAPIs s uID p \equiv$
 $IDsOK s [uID] [] [] \wedge pass s uID = p \wedge uID = admin s$

definition $listPendingCApiReqs :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow list$
where
 $listPendingCApiReqs s uID p \equiv pendingCApiReqs s$

definition $e-listPendingCApiReqs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
 $e-listPendingCApiReqs s uID p \equiv$
 $IDsOK s [uID] [] [] \wedge pass s uID = p \wedge uID = admin s$

definition $listClientAPIs :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow list$
where
 $listClientAPIs s uID p \equiv clientApiIDs s$

definition $e-listClientAPIs :: state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
 $e-listClientAPIs s uID p \equiv$
 $IDsOK s [uID] [] [] \wedge pass s uID = p \wedge uID = admin s$

3.2.8 Actions of communication with other APIs

definition $sendServerReq :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow requestInfo$
 $\Rightarrow (apiID \times requestInfo) \times state$
where
 $sendServerReq s uID p aID reqInfo \equiv$
 $((aID, reqInfo),$
 $s (\downarrow pendingSApiReqs := pendingSApiReqs s @ [aID],$
 $sApiReq := (sApiReq s) (aID := reqInfo)))$

definition $e-sendServerReq :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow requestInfo$
 $\Rightarrow bool$
where
 $e-sendServerReq s uID p aID reqInfo \equiv$
 $IDsOK s [uID] [] [] \wedge pass s uID = p \wedge$
 $uID = admin s \wedge \neg aID \in pendingSApiReqs s$

definition $receiveClientReq :: state \Rightarrow apiID \Rightarrow requestInfo \Rightarrow state$
where
 $receiveClientReq s aID reqInfo \equiv$
 $s (\downarrow pendingCApiReqs := pendingCApiReqs s @ [aID],$
 $cApiReq := (cApiReq s) (aID := reqInfo))$

definition $e-receiveClientReq :: state \Rightarrow apiID \Rightarrow requestInfo \Rightarrow bool$
where
 $e-receiveClientReq s aID reqInfo \equiv$

$\neg aID \in \in pendingCApiReqs\ s \wedge admin\ s \in \in userIDs\ s$

definition $connectClient :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow password \Rightarrow (apiID \times password) \times state$

where

$connectClient\ s\ uID\ p\ aID\ cp \equiv$
 $((aID, cp),$
 $s\ (\!|clientApiIDs := (aID \# clientApiIDs\ s),$
 $clientPass := (clientPass\ s)\ (aID := cp),$
 $pendingCApiReqs := remove1\ aID\ (pendingCApiReqs\ s),$
 $cApiReq := (cApiReq\ s)(aID := emptyRequestInfo)\!|)$
 $)$

definition $e-connectClient :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow password \Rightarrow bool$

where

$e-connectClient\ s\ uID\ p\ aID\ cp \equiv$
 $IDsOK\ s\ [uID]\ []\ [] \wedge pass\ s\ uID = p \wedge$
 $uID = admin\ s \wedge$
 $aID \in \in pendingCApiReqs\ s \wedge \neg aID \in \in clientApiIDs\ s$

definition $connectServer :: state \Rightarrow apiID \Rightarrow password \Rightarrow state$

where

$connectServer\ s\ aID\ sp \equiv$
 $s\ (\!|serverApiIDs := (aID \# serverApiIDs\ s),$
 $serverPass := (serverPass\ s)\ (aID := sp),$
 $pendingSApiReqs := remove1\ aID\ (pendingSApiReqs\ s),$
 $sApiReq := (sApiReq\ s)(aID := emptyRequestInfo)\!|)$

definition $e-connectServer :: state \Rightarrow apiID \Rightarrow password \Rightarrow bool$

where

$e-connectServer\ s\ aID\ sp \equiv$
 $aID \in \in pendingSApiReqs\ s \wedge \neg aID \in \in serverApiIDs\ s$

definition $sendPost ::$

$state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow postID \Rightarrow (apiID \times password \times postID \times post \times userID \times vis) \times state$

where

$sendPost\ s\ uID\ p\ aID\ pID \equiv$
 $((aID, clientPass\ s\ aID, pID, post\ s\ pID, owner\ s\ pID, vis\ s\ pID),$
 $s\ (\!|sharedWith := (sharedWith\ s)\ (pID := insert2\ aID\ True\ (sharedWith\ s\ pID))\!|))$

definition $e-sendPost :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow postID \Rightarrow bool$

where

$e\text{-sendPost } s \ uID \ p \ aID \ pID \equiv$
 $IDsOK \ s \ [uID] \ [pID] \ [] \ [aID] \ \wedge \ pass \ s \ uID = p \ \wedge$
 $uID = admin \ s \ \wedge \ aID \in\in \ clientApiIDs \ s$

definition $receivePost :: state \Rightarrow apiID \Rightarrow password \Rightarrow postID \Rightarrow post \Rightarrow userID$
 $\Rightarrow vis \Rightarrow state$

where

$receivePost \ s \ aID \ sp \ pID \ pst \ uID \ vs \equiv$
 $let \ opIDs = outerPostIDs \ s \ in$
 $s \ (\setminus outerPostIDs := opIDs \ (aID := List.insert \ pID \ (opIDs \ aID)),$
 $outerPost := fun\text{-upd2} \ (outerPost \ s) \ aID \ pID \ pst,$
 $outerOwner := fun\text{-upd2} \ (outerOwner \ s) \ aID \ pID \ uID,$
 $outerVis := fun\text{-upd2} \ (outerVis \ s) \ aID \ pID \ vs)$

definition $e\text{-receivePost} :: state \Rightarrow apiID \Rightarrow password \Rightarrow postID \Rightarrow post \Rightarrow userID$
 $\Rightarrow vis \Rightarrow bool$

where

$e\text{-receivePost } s \ aID \ sp \ pID \ nt \ uID \ vs \equiv$
 $IDsOK \ s \ [] \ [] \ [(aID,[])] \ [] \ \wedge \ serverPass \ s \ aID = sp$

definition $sendCreateOFriend ::$

$state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow userID \Rightarrow (apiID \times password \times userID$
 $\times userID) \times state$

where

$sendCreateOFriend \ s \ uID \ p \ aID \ uID' \equiv$
 $let \ ofr = sentOuterFriendIDs \ s \ in$
 $((aID, clientPass \ s \ aID, uID, uID'),$
 $s \ (\setminus sentOuterFriendIDs := ofr \ (uID := ofr \ uID \ @ \ [(aID, uID')]))$

definition $e\text{-sendCreateOFriend} :: state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow userID$
 $\Rightarrow bool$

where

$e\text{-sendCreateOFriend } s \ uID \ p \ caID \ uID' \equiv$
 $IDsOK \ s \ [uID] \ [] \ [] \ [caID] \ \wedge \ pass \ s \ uID = p \ \wedge$
 $\neg \ (caID, uID') \in\in \ sentOuterFriendIDs \ s \ uID$

definition $receiveCreateOFriend :: state \Rightarrow apiID \Rightarrow password \Rightarrow userID \Rightarrow userID$
 $\Rightarrow state$

where

$receiveCreateOFriend \ s \ saID \ sp \ uID \ uID' \equiv$

let $ofr = \text{recvOuterFriendIDs } s \text{ in}$
 $s (\text{recvOuterFriendIDs} := ofr (uID' := ofr uID' @ [(saID, uID)]))$

definition $e\text{-receiveCreateOFriend} :: \text{state} \Rightarrow \text{apiID} \Rightarrow \text{password} \Rightarrow \text{userID} \Rightarrow \text{userID} \Rightarrow \text{bool}$

where

$e\text{-receiveCreateOFriend } s \text{ saID } sp \text{ uID } uID' \equiv$
 $IDsOK \ s \ [] \ [] \ [(saID, [])] \ [] \wedge \text{serverPass } s \ saID = sp \wedge$
 $\neg (saID, uID) \in \in \text{recvOuterFriendIDs } s \ uID'$

definition $\text{sendDeleteOFriend} ::$

$\text{state} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{apiID} \Rightarrow \text{userID} \Rightarrow (\text{apiID} \times \text{password} \times \text{userID}$
 $\times \text{userID}) \times \text{state}$

where

$\text{sendDeleteOFriend } s \ uID \ p \ aID \ uID' \equiv$
 let $ofr = \text{sentOuterFriendIDs } s \text{ in}$
 $((aID, \text{clientPass } s \ aID, \ uID, \ uID'),$
 $s (\text{sentOuterFriendIDs} := ofr (uID := \text{remove1 } (aID, uID') (ofr uID))))$

definition $e\text{-sendDeleteOFriend} :: \text{state} \Rightarrow \text{userID} \Rightarrow \text{password} \Rightarrow \text{apiID} \Rightarrow \text{userID} \Rightarrow \text{bool}$

where

$e\text{-sendDeleteOFriend } s \ uID \ p \ caID \ uID' \equiv$
 $IDsOK \ s \ [uID] \ [] \ [] \ [caID] \wedge \text{pass } s \ uID = p \wedge$
 $(caID, uID') \in \in \text{sentOuterFriendIDs } s \ uID$

definition $\text{receiveDeleteOFriend} :: \text{state} \Rightarrow \text{apiID} \Rightarrow \text{password} \Rightarrow \text{userID} \Rightarrow \text{userID} \Rightarrow \text{state}$

where

$\text{receiveDeleteOFriend } s \ saID \ sp \ uID \ uID' \equiv$
 let $ofr = \text{recvOuterFriendIDs } s \text{ in}$
 $s (\text{recvOuterFriendIDs} := ofr (uID' := \text{remove1 } (saID, uID) (ofr uID')))$

definition $e\text{-receiveDeleteOFriend} :: \text{state} \Rightarrow \text{apiID} \Rightarrow \text{password} \Rightarrow \text{userID} \Rightarrow \text{userID} \Rightarrow \text{bool}$

where

$e\text{-receiveDeleteOFriend } s \ saID \ sp \ uID \ uID' \equiv$
 $IDsOK \ s \ [] \ [] \ [(saID, [])] \ [] \wedge \text{serverPass } s \ saID = sp \wedge$
 $(saID, uID) \in \in \text{recvOuterFriendIDs } s \ uID'$

3.3 The step function

datatype $\text{out} =$

$\text{outOK} \mid \text{outErr} \mid$

outBool *bool* | *outName* *name* |
outPost *post* | *outVis* *vis* |
outReq *requestInfo* |

outUID *userID* | *outUIDL* *userID list* |
outAIDL *apiID list* | *outAIDBL* (*apiID* × *bool*) *list* |
outUIDPIDL (*userID* × *postID*)*list* | *outAIDPIDL* (*apiID* × *postID*)*list* |
outAIDUIDL (*apiID* × *userID*) *list* |

O-sendServerReq *apiID* × *requestInfo* | *O-connectClient* *apiID* × *password* |
O-sendPost *apiID* × *password* × *postID* × *post* × *userID* × *vis* |
O-sendCreateOFriend *apiID* × *password* × *userID* × *userID* |
O-sendDeleteOFriend *apiID* × *password* × *userID* × *userID*

fun *from-O-sendPost* **where**
from-O-sendPost (*O-sendPost* *antt*) = *antt*
|*from-O-sendPost* - = *undefined*

datatype *sActt* =
sSys *userID* *password*

lemmas *s-defs* =
e-startSys-def *startSys-def*

fun *sStep* :: *state* ⇒ *sActt* ⇒ *out* * *state* **where**
sStep *s* (*sSys* *uID* *p*) =
(if *e-startSys* *s* *uID* *p*
then (*outOK*, *startSys* *s* *uID* *p*)
else (*outErr*, *s*))

fun *sUserOfA* :: *sActt* ⇒ *userID* **where**
sUserOfA (*sSys* *uID* *p*) = *uID*

datatype *cActt* =
cNUReq *userID* *requestInfo*
|*cUser* *userID* *password* *userID* *password*
|*cPost* *userID* *password* *postID*
|*cFriendReq* *userID* *password* *userID* *requestInfo*
|*cFriend* *userID* *password* *userID*

lemmas *c-defs* =
e-createNUReq-def *createNUReq-def*
e-createUser-def *createUser-def*
e-createPost-def *createPost-def*

e-createFriendReq-def createFriendReq-def
e-createFriend-def createFriend-def

```
fun cStep :: state ⇒ cActt ⇒ out * state where
cStep s (cNUReq uID req) =
  (if e-createNUReq s uID req
   then (outOK, createNUReq s uID req)
   else (outErr, s))
|
cStep s (cUser uID p uID' p') =
  (if e-createUser s uID p uID' p'
   then (outOK, createUser s uID p uID' p')
   else (outErr, s))
|
cStep s (cPost uID p pID) =
  (if e-createPost s uID p pID
   then (outOK, createPost s uID p pID)
   else (outErr, s))
|
cStep s (cFriendReq uID p uID' req) =
  (if e-createFriendReq s uID p uID' req
   then (outOK, createFriendReq s uID p uID' req)
   else (outErr, s))
|
cStep s (cFriend uID p uID') =
  (if e-createFriend s uID p uID'
   then (outOK, createFriend s uID p uID')
   else (outErr, s))
```

```
fun cUserOfA :: cActt ⇒ userID where
cUserOfA (cNUReq uID req) = uID
| cUserOfA (cUser uID p uID' p') = uID
| cUserOfA (cPost uID p pID) = uID
| cUserOfA (cFriendReq uID p uID' req) = uID
| cUserOfA (cFriend uID p uID') = uID
```

```
datatype dActt =
  dFriend userID password userID
```

lemmas d-defs =
e-deleteFriend-def deleteFriend-def

```
fun dStep :: state ⇒ dActt ⇒ out * state where
dStep s (dFriend uID p uID') =
  (if e-deleteFriend s uID p uID'
   then (outOK, deleteFriend s uID p uID')
   else (outErr, s))
```

```
fun dUserOfA :: dActt ⇒ userID where
  dUserOfA (dFriend uID p uID') = uID
```

```
datatype uActt =
  |isuUser: uUser userID password password name inform
  |isuPost: uPost userID password postID post
  |isuVisPost: uVisPost userID password postID vis
```

```
lemmas u-defs =
  e-updateUser-def updateUser-def
  e-updatePost-def updatePost-def
  e-updateVisPost-def updateVisPost-def
```

```
fun uStep :: state ⇒ uActt ⇒ out * state where
  uStep s (uUser uID p p' name info) =
    (if e-updateUser s uID p p' name info
     then (outOK, updateUser s uID p p' name info)
     else (outErr, s))
  |
  uStep s (uPost uID p pID pst) =
    (if e-updatePost s uID p pID pst
     then (outOK, updatePost s uID p pID pst)
     else (outErr, s))
  |
  uStep s (uVisPost uID p pID visStr) =
    (if e-updateVisPost s uID p pID visStr
     then (outOK, updateVisPost s uID p pID visStr)
     else (outErr, s))
```

```
fun uUserOfA :: uActt ⇒ userID where
  uUserOfA (uUser uID p p' name info) = uID
  |uUserOfA (uPost uID p pID pst) = uID
  |uUserOfA (uVisPost uID p pID visStr) = uID
```

```
datatype rActt =
  |rNUReq userID password userID
  |rUser userID password userID
  |rAmIAdmin userID password

  |rPost userID password postID

  |rOwnerPost userID password postID
  |rVisPost userID password postID

  |rOPost userID password apiID postID
```

|*rOwnerOPost* *userID password apiID postID*
|*rVisOPost* *userID password apiID postID*

|*rFriendReqToMe* *userID password userID*
|*rFriendReqFromMe* *userID password userID*
|*rSApiReq* *userID password apiID*
|*rCApiReq* *userID password apiID*

lemmas *r-defs* =

readNUReq-def e-readNUReq-def
readUser-def e-readUser-def
readAmIAAdmin-def e-readAmIAAdmin-def

readPost-def e-readPost-def

readOwnerPost-def e-readOwnerPost-def
readVisPost-def e-readVisPost-def

readOPost-def e-readOPost-def

readOwnerOPost-def e-readOwnerOPost-def
readVisOPost-def e-readVisOPost-def

readFriendReqToMe-def e-readFriendReqToMe-def
readFriendReqFromMe-def e-readFriendReqFromMe-def
readSApiReq-def e-readSApiReq-def
readCApiReq-def e-readCApiReq-def

fun *rObs* :: *state* \Rightarrow *rActt* \Rightarrow *out* **where**

rObs *s* (*rNUReq* *uID* *p* *uID'*) =
 (*if* *e-readNUReq* *s* *uID* *p* *uID'* *then outReq* (*readNUReq* *s* *uID* *p* *uID'*) *else outErr*)
|
rObs *s* (*rUser* *uID* *p* *uID'*) =
 (*if* *e-readUser* *s* *uID* *p* *uID'* *then outName* (*readUser* *s* *uID* *p* *uID'*) *else outErr*)
|
rObs *s* (*rAmIAAdmin* *uID* *p*) =
 (*if* *e-readAmIAAdmin* *s* *uID* *p* *then outBool* (*readAmIAAdmin* *s* *uID* *p*) *else outErr*)
|
rObs *s* (*rPost* *uID* *p* *pID*) =
 (*if* *e-readPost* *s* *uID* *p* *pID* *then outPost* (*readPost* *s* *uID* *p* *pID*) *else outErr*)
|
rObs *s* (*rOwnerPost* *uID* *p* *pID*) =
 (*if* *e-readOwnerPost* *s* *uID* *p* *pID* *then outUID* (*readOwnerPost* *s* *uID* *p* *pID*) *else outErr*)
|
rObs *s* (*rVisPost* *uID* *p* *pID*) =
 (*if* *e-readVisPost* *s* *uID* *p* *pID* *then outVis* (*readVisPost* *s* *uID* *p* *pID*) *else outErr*)
|

```

rObs s (rOPost uID p aID pID) =
  (if e-readOPost s uID p aID pID then outPost (readOPost s uID p aID pID) else
outErr)
|
rObs s (rOwnerOPost uID p aID pID) =
  (if e-readOwnerOPost s uID p aID pID then outUID (readOwnerOPost s uID p
aID pID) else outErr)
|
rObs s (rVisOPost uID p aID pID) =
  (if e-readVisOPost s uID p aID pID then outVis (readVisOPost s uID p aID pID)
else outErr)
|

```

```

rObs s (rFriendReqToMe uID p uID') =
  (if e-readFriendReqToMe s uID p uID' then outReq (readFriendReqToMe s uID p
uID') else outErr)
|
rObs s (rFriendReqFromMe uID p uID') =
  (if e-readFriendReqFromMe s uID p uID' then outReq (readFriendReqFromMe s
uID p uID') else outErr)
|
rObs s (rSApiReq uID p aID) =
  (if e-readSApiReq s uID p aID then outReq (readSApiReq s uID p aID) else outErr)
|
rObs s (rCApiReq uID p aID) =
  (if e-readCApiReq s uID p aID then outReq (readCApiReq s uID p aID) else outErr)

```

```

fun rUserOfA :: rActt ⇒ userID where
  rUserOfA (rNUReq uID p uID') = uID
| rUserOfA (rUser uID p uID') = uID
| rUserOfA (rAmIAdmin uID p) = uID

| rUserOfA (rPost uID p pID) = uID
| rUserOfA (rOwnerPost uID p pID) = uID
| rUserOfA (rVisPost uID p pID) = uID

| rUserOfA (rOPost uID p aID pID) = uID
| rUserOfA (rOwnerOPost uID p aID pID) = uID
| rUserOfA (rVisOPost uID p aID pID) = uID

| rUserOfA (rFriendReqToMe uID p uID') = uID
| rUserOfA (rFriendReqFromMe uID p uID') = uID
| rUserOfA (rSApiReq uID p aID) = uID
| rUserOfA (rCApiReq uID p aID) = uID

```

```

datatype lActt =

```

```

lPendingUReqs userID password
|lAllUsers userID password
|lFriends userID password userID
|lSentOuterFriends userID password userID
|lRecvOuterFriends userID password
|lInnerPosts userID password
|lOuterPosts userID password
|lClientsPost userID password postID
|lPendingSApiReqs userID password
|lServerAPIs userID password
|lPendingCApiReqs userID password
|lClientAPIs userID password

```

lemmas *l-defs* =

```

listPendingUReqs-def e-listPendingUReqs-def
listAllUsers-def e-listAllUsers-def
listFriends-def e-listFriends-def
listSentOuterFriends-def e-listSentOuterFriends-def
listRecvOuterFriends-def e-listRecvOuterFriends-def
listInnerPosts-def e-listInnerPosts-def
listOuterPosts-def e-listOuterPosts-def
listClientsPost-def e-listClientsPost-def
listPendingSApiReqs-def e-listPendingSApiReqs-def
listServerAPIs-def e-listServerAPIs-def
listPendingCApiReqs-def e-listPendingCApiReqs-def
listClientAPIs-def e-listClientAPIs-def

```

fun *lObs* :: *state* ⇒ *lActt* ⇒ *out* **where**

```

lObs s (lPendingUReqs uID p) =
  (if e-listPendingUReqs s uID p then outUIDL (listPendingUReqs s uID p) else outErr)
|
lObs s (lAllUsers uID p) =
  (if e-listAllUsers s uID p then outUIDL (listAllUsers s uID p) else outErr)
|
lObs s (lFriends uID p uID') =
  (if e-listFriends s uID p uID' then outUIDL (listFriends s uID p uID') else outErr)
|
lObs s (lSentOuterFriends uID p uID') =
  (if e-listSentOuterFriends s uID p uID' then outAIDUIDL (listSentOuterFriends
s uID p uID') else outErr)
|
lObs s (lRecvOuterFriends uID p) =
  (if e-listRecvOuterFriends s uID p then outAIDUIDL (listRecvOuterFriends s uID
p) else outErr)
|
lObs s (lInnerPosts uID p) =
  (if e-listInnerPosts s uID p then outUIDPIDL (listInnerPosts s uID p) else outErr)

```

```

|
lObs s (lOuterPosts uID p) =
  (if e-listOuterPosts s uID p then outAIDPIDL (listOuterPosts s uID p) else out-
  Err)
|
lObs s (lClientsPost uID p pID) =
  (if e-listClientsPost s uID p pID then outAIDBL (listClientsPost s uID p pID) else
  outErr)
|
lObs s (lPendingSApiReqs uID p) =
  (if e-listPendingSApiReqs s uID p then outAIDL (listPendingSApiReqs s uID p)
  else outErr)
|
lObs s (lServerAPIs uID p) =
  (if e-listServerAPIs s uID p then outAIDL (listServerAPIs s uID p) else outErr)
|
lObs s (lClientAPIs uID p) =
  (if e-listClientAPIs s uID p then outAIDL (listClientAPIs s uID p) else outErr)
|
lObs s (lPendingCApiReqs uID p) =
  (if e-listPendingCApiReqs s uID p then outAIDL (listPendingCApiReqs s uID p)
  else outErr)

```

```

fun lUserOfA :: lActt ⇒ userID where
  lUserOfA (lPendingUReqs uID p) = uID
|lUserOfA (lAllUsers uID p) = uID
|lUserOfA (lFriends uID p uID') = uID
|lUserOfA (lSentOuterFriends uID p uID') = uID
|lUserOfA (lRecvOuterFriends uID p) = uID
|lUserOfA (lInnerPosts uID p) = uID
|lUserOfA (lOuterPosts uID p) = uID
|lUserOfA (lClientsPost uID p pID) = uID
|lUserOfA (lPendingSApiReqs uID p) = uID
|lUserOfA (lServerAPIs uID p) = uID
|lUserOfA (lClientAPIs uID p) = uID
|lUserOfA (lPendingCApiReqs uID p) = uID

```

```

datatype comActt =
  comSendServerReq userID password apiID requestInfo
|comReceiveClientReq apiID requestInfo
|comConnectClient userID password apiID password
|comConnectServer apiID password
|comReceivePost apiID password postID post userID vis
|comSendPost userID password apiID postID
|comReceiveCreateOfriend apiID password userID userID
|comSendCreateOfriend userID password apiID userID

```

```
|comReceiveDeleteOFriend apiID password userID userID
|comSendDeleteOFriend userID password apiID userID
```

lemmas *com-defs* =

```
sendServerReq-def e-sendServerReq-def
receiveClientReq-def e-receiveClientReq-def
connectClient-def e-connectClient-def
connectServer-def e-connectServer-def
receivePost-def e-receivePost-def
sendPost-def e-sendPost-def
receiveCreateOFriend-def e-receiveCreateOFriend-def
sendCreateOFriend-def e-sendCreateOFriend-def
receiveDeleteOFriend-def e-receiveDeleteOFriend-def
sendDeleteOFriend-def e-sendDeleteOFriend-def
```

fun *comStep* :: *state* ⇒ *comActt* ⇒ *out* × *state* **where**

```
comStep s (comSendServerReq uID p aID reqInfo) =
  (if e-sendServerReq s uID p aID reqInfo
   then let (x,s) = sendServerReq s uID p aID reqInfo in (O-sendServerReq x, s)
   else (outErr, s))
|
comStep s (comReceiveClientReq aID reqInfo) =
  (if e-receiveClientReq s aID reqInfo then (outOK, receiveClientReq s aID reqInfo)
  else (outErr, s))
|
comStep s (comConnectClient uID p aID cp) =
  (if e-connectClient s uID p aID cp
   then let (aID-cp,s) = connectClient s uID p aID cp in (O-connectClient aID-cp,
   s)
   else (outErr, s))
|
comStep s (comConnectServer aID sp) =
  (if e-connectServer s aID sp then (outOK, connectServer s aID sp) else (outErr,
  s))
|
comStep s (comReceivePost aID sp pID nt uID vs) =
  (if e-receivePost s aID sp pID nt uID vs
   then (outOK, receivePost s aID sp pID nt uID vs)
   else (outErr, s))
|
comStep s (comSendPost uID p aID pID) =
  (if e-sendPost s uID p aID pID
   then let (x,s) = sendPost s uID p aID pID in (O-sendPost x, s)
   else (outErr, s))
|
comStep s (comReceiveCreateOFriend aID cp uID uID') =
  (if e-receiveCreateOFriend s aID cp uID uID'
   then (outOK, receiveCreateOFriend s aID cp uID uID')
   else (outErr, s))
```

```

|
comStep s (comSendCreateOFriend uID p aID uID') =
  (if e-sendCreateOFriend s uID p aID uID'
   then let (apuu,s) = sendCreateOFriend s uID p aID uID' in (O-sendCreateOFriend
apuu, s)
   else (outErr, s))
|
comStep s (comReceiveDeleteOFriend aID cp uID uID') =
  (if e-receiveDeleteOFriend s aID cp uID uID'
   then (outOK, receiveDeleteOFriend s aID cp uID uID')
   else (outErr, s))
|
comStep s (comSendDeleteOFriend uID p aID uID') =
  (if e-sendDeleteOFriend s uID p aID uID'
   then let (apuu,s) = sendDeleteOFriend s uID p aID uID' in (O-sendDeleteOFriend
apuu, s)
   else (outErr, s))

```

```

fun comUserOfA :: comActt ⇒ userID option where
  comUserOfA (comSendServerReq uID p aID reqInfo) = Some uID
| comUserOfA (comReceiveClientReq aID reqInfo) = None
| comUserOfA (comConnectClient uID p aID sp) = Some uID
| comUserOfA (comConnectServer aID sp) = None
| comUserOfA (comReceivePost aID sp pID nt uID vs) = None
| comUserOfA (comSendPost uID p aID pID) = Some uID
| comUserOfA (comReceiveCreateOFriend aID cp uID uID') = None
| comUserOfA (comSendCreateOFriend uID p aID uID') = Some uID
| comUserOfA (comReceiveDeleteOFriend aID cp uID uID') = None
| comUserOfA (comSendDeleteOFriend uID p aID uID') = Some uID

```

```

fun comApiOfA :: comActt ⇒ apiID where
  comApiOfA (comSendServerReq uID p aID reqInfo) = aID
| comApiOfA (comReceiveClientReq aID reqInfo) = aID
| comApiOfA (comConnectClient uID p aID sp) = aID
| comApiOfA (comConnectServer aID sp) = aID
| comApiOfA (comReceivePost aID sp pID nt uID vs) = aID
| comApiOfA (comSendPost uID p aID pID) = aID
| comApiOfA (comReceiveCreateOFriend aID cp uID uID') = aID
| comApiOfA (comSendCreateOFriend uID p aID uID') = aID
| comApiOfA (comReceiveDeleteOFriend aID cp uID uID') = aID
| comApiOfA (comSendDeleteOFriend uID p aID uID') = aID

```

```

datatype act =
  isSact: Sact sActt |

  isCact: Cact cActt | isDact: Dact dActt | isUact: Uact uActt |

```

```

    isRact: Ract rActt | isLact: Lact lActt |

    isCOMact: COMact comActt

fun step :: state ⇒ act ⇒ out * state where
step s (Sact sa) = sStep s sa
|
step s (Cact ca) = cStep s ca
|
step s (Dact da) = dStep s da
|
step s (Uact ua) = uStep s ua
|
step s (Ract ra) = (rObs s ra, s)
|
step s (Lact la) = (lObs s la, s)
|
step s (COMact ca) = comStep s ca

fun userOfA :: act ⇒ userID option where
userOfA (Sact sa) = Some (sUserOfA sa)
|
userOfA (Cact ca) = Some (cUserOfA ca)
|
userOfA (Dact da) = Some (dUserOfA da)
|
userOfA (Uact ua) = Some (uUserOfA ua)
|
userOfA (Ract ra) = Some (rUserOfA ra)
|
userOfA (Lact la) = Some (lUserOfA la)
|
userOfA (COMact ca) = comUserOfA ca

```

```

interpretation IO-Automaton where
istate = istate and step = step
⟨proof⟩

```

3.4 Code generation

```

export-code step istate getFreshPostID in Scala

end

```

4 The CoSMeDis network of communicating nodes

This is the specification of an entire CoSMeDis network of communicating nodes, as described in Section IV.B of [3] NB: What that paper refers to as "nodes" are referred in this formalization as "APIs".

```

theory API-Network
imports
  BD-Security-Compositional.Composing-Security-Network
  System-Specification
begin

locale Network =
fixes AIDs :: apiID set
assumes finite-AIDs: finite AIDs
begin

fun comOfO :: apiID  $\Rightarrow$  (act  $\times$  out)  $\Rightarrow$  com where
  comOfO aid (COMact (comSendServerReq uid password aID req), ou) =
    (if aid  $\neq$  aID  $\wedge$  ou  $\neq$  outErr then Send else Internal)
| comOfO aid (COMact (comConnectClient uid p aID sp), ou) =
  (if aid  $\neq$  aID  $\wedge$  ou  $\neq$  outErr then Send else Internal)
| comOfO aid (COMact (comSendPost uid p aID nID), ou) =
  (if aid  $\neq$  aID  $\wedge$  ou  $\neq$  outErr then Send else Internal)
| comOfO aid (COMact (comSendCreateOFriend uid p aID uid'), ou) =
  (if aid  $\neq$  aID  $\wedge$  ou  $\neq$  outErr then Send else Internal)
| comOfO aid (COMact (comSendDeleteOFriend uid p aID uid'), ou) =
  (if aid  $\neq$  aID  $\wedge$  ou  $\neq$  outErr then Send else Internal)
| comOfO aid (COMact (comReceiveClientReq aID req), ou) =
  (if aid  $\neq$  aID  $\wedge$  ou  $\neq$  outErr then Recv else Internal)
| comOfO aid (COMact (comConnectServer aID sp), ou) =
  (if aid  $\neq$  aID  $\wedge$  ou  $\neq$  outErr then Recv else Internal)
| comOfO aid (COMact (comReceivePost aID sp nID ntc uid v), ou) =
  (if aid  $\neq$  aID  $\wedge$  ou  $\neq$  outErr then Recv else Internal)
| comOfO aid (COMact (comReceiveCreateOFriend aID sp uid uid'), ou) =
  (if aid  $\neq$  aID  $\wedge$  ou  $\neq$  outErr then Recv else Internal)
| comOfO aid (COMact (comReceiveDeleteOFriend aID sp uid uid'), ou) =
  (if aid  $\neq$  aID  $\wedge$  ou  $\neq$  outErr then Recv else Internal)
| comOfO - - = Internal

fun comOf :: apiID  $\Rightarrow$  (state, act, out) trans  $\Rightarrow$  com where
  comOf aid (Trans - a ou -) = comOfO aid (a, ou)

fun syncO :: apiID  $\Rightarrow$  (act  $\times$  out)  $\Rightarrow$  apiID  $\Rightarrow$  (act  $\times$  out)  $\Rightarrow$  bool where
  syncO aid1 (COMact (comSendServerReq uid p aid req), ou1) aid2 (a2, ou2) =
    ( $\exists$  req2. a2 = (COMact (comReceiveClientReq aid1 req2))  $\wedge$  ou1 = O-sendServerReq
(aid2, req2)  $\wedge$  ou2 = outOK)
| syncO aid1 (COMact (comConnectClient uid p aid sp), ou1) aid2 (a2, ou2) =
  ( $\exists$  sp2. a2 = (COMact (comConnectServer aid1 sp2))  $\wedge$  ou1 = O-connectClient

```

$(aid2, sp2) \wedge ou2 = outOK$
 $| \text{syncO } aid1 \text{ (COMact (comSendPost } uid \text{ } p \text{ } aid \text{ } nid), ou1) } aid2 \text{ (} a2, ou2) =$
 $(\exists sp2 \text{ } nid2 \text{ } ntc2 \text{ } uid2 \text{ } v. a2 = (COMact (comReceivePost } aid1 \text{ } sp2 \text{ } nid2 \text{ } ntc2$
 $uid2 \text{ } v)) \wedge ou1 = O\text{-sendPost (} aid2, sp2, nid2, ntc2, uid2, v) \wedge ou2 = outOK$
 $| \text{syncO } aid1 \text{ (COMact (comSendCreateOFriend } uid \text{ } p \text{ } aid \text{ } uid'), ou1) } aid2 \text{ (} a2,$
 $ou2) =$
 $(\exists sp2 \text{ } uid2 \text{ } uid2'. a2 = (COMact (comReceiveCreateOFriend } aid1 \text{ } sp2 \text{ } uid2$
 $uid2')$ $\wedge ou1 = O\text{-sendCreateOFriend (} aid2, sp2, uid2, uid2')$ $\wedge ou2 = outOK$
 $| \text{syncO } aid1 \text{ (COMact (comSendDeleteOFriend } uid \text{ } p \text{ } aid \text{ } uid'), ou1) } aid2 \text{ (} a2,$
 $ou2) =$
 $(\exists sp2 \text{ } uid2 \text{ } uid2'. a2 = (COMact (comReceiveDeleteOFriend } aid1 \text{ } sp2 \text{ } uid2$
 $uid2')$ $\wedge ou1 = O\text{-sendDeleteOFriend (} aid2, sp2, uid2, uid2')$ $\wedge ou2 = outOK$
 $| \text{syncO } - - - = False$

fun *cmpO* :: *apiID* \Rightarrow (*act* \times *out*) \Rightarrow *apiID* \Rightarrow (*act* \times *out*) \Rightarrow (*apiID* \times *act* \times *out*
 \times *apiID* \times *act* \times *out*) **where**
cmpO *aid1* *obs1* *aid2* *obs2* = (*aid1*, *fst* *obs1*, *snd* *obs1*, *aid2*, *fst* *obs2*, *snd* *obs2*)

fun *sync* :: *apiID* \Rightarrow (*state*, *act*, *out*) *trans* \Rightarrow *apiID* \Rightarrow (*state*, *act*, *out*) *trans* \Rightarrow
bool **where**
sync *aid1* (*Trans* *s1* *a1* *ou1* *s1'*) *aid2* (*Trans* *s2* *a2* *ou2* *s2'*) = *syncO* *aid1* (*a1*,
ou1) *aid2* (*a2*, *ou2*)

lemma *syncO-cases*:

assumes *syncO* *aid1* *obs1* *aid2* *obs2*

obtains

(*Req*) *uid* *p* *aid* *req1* *req2*
where *obs1* = (*COMact* (*comSendServerReq* *uid* *p* *aid* *req1*), *O-sendServerReq*
(*aid2*, *req2*))
and *obs2* = (*COMact* (*comReceiveClientReq* *aid1* *req2*), *outOK*)
| (*Connect*) *uid* *p* *aid* *sp* *sp2*
where *obs1* = (*COMact* (*comConnectClient* *uid* *p* *aid* *sp*), *O-connectClient*
(*aid2*, *sp2*))
and *obs2* = (*COMact* (*comConnectServer* *aid1* *sp2*), *outOK*)
| (*Notice*) *uid* *p* *aid* *nid* *sp2* *nid2* *ntc2* *own2* *v*
where *obs1* = (*COMact* (*comSendPost* *uid* *p* *aid* *nid*), *O-sendPost* (*aid2*, *sp2*,
nid2, *ntc2*, *own2*, *v*))
and *obs2* = (*COMact* (*comReceivePost* *aid1* *sp2* *nid2* *ntc2* *own2* *v*), *outOK*)
| (*CFriend*) *uid* *p* *aid* *uid'* *sp2* *uid2* *uid2'*
where *obs1* = (*COMact* (*comSendCreateOFriend* *uid* *p* *aid* *uid'*), *O-sendCreateOFriend*
(*aid2*, *sp2*, *uid2*, *uid2'*))
and *obs2* = (*COMact* (*comReceiveCreateOFriend* *aid1* *sp2* *uid2* *uid2'*), *outOK*)
| (*DFriend*) *uid* *p* *aid* *uid'* *sp2* *uid2* *uid2'*
where *obs1* = (*COMact* (*comSendDeleteOFriend* *uid* *p* *aid* *uid'*), *O-sendDeleteOFriend*
(*aid2*, *sp2*, *uid2*, *uid2'*))
and *obs2* = (*COMact* (*comReceiveDeleteOFriend* *aid1* *sp2* *uid2* *uid2'*), *outOK*)
(*proof*)

lemma *sync-cases*:

assumes *sync aid1 trn1 aid2 trn2*

and *validTrans trn1*

obtains

(*Req*) *uid p aid req s1 s1' s2 s2'*
where *trn1 = Trans s1 (COMact (comSendServerReq uid p aid req)) (O-sendServerReq (aid2, req)) s1'*
and *trn2 = Trans s2 (COMact (comReceiveClientReq aid1 req)) outOK s2'*
| (*Connect*) *uid p aid sp s1 s1' s2 s2'*
where *trn1 = Trans s1 (COMact (comConnectClient uid p aid sp)) (O-connectClient (aid2, sp)) s1'*
and *trn2 = Trans s2 (COMact (comConnectServer aid1 sp)) outOK s2'*
| (*Notice*) *uid p aid nid sp2 nid2 ntc2 own2 v s1 s1' s2 s2'*
where *trn1 = Trans s1 (COMact (comSendPost uid p aid nid)) (O-sendPost (aid2, sp2, nid2, ntc2, own2, v)) s1'*
and *trn2 = Trans s2 (COMact (comReceivePost aid1 sp2 nid2 ntc2 own2 v)) outOK s2'*
| (*CFriend*) *uid p uid' sp s1 s1' s2 s2'*
where *trn1 = Trans s1 (COMact (comSendCreateOFriend uid p aid2 uid')) (O-sendCreateOFriend (aid2, sp, uid, uid')) s1'*
and *trn2 = Trans s2 (COMact (comReceiveCreateOFriend aid1 sp uid uid')) outOK s2'*
| (*DFriend*) *uid p aid uid' sp s1 s1' s2 s2'*
where *trn1 = Trans s1 (COMact (comSendDeleteOFriend uid p aid2 uid')) (O-sendDeleteOFriend (aid2, sp, uid, uid')) s1'*
and *trn2 = Trans s2 (COMact (comReceiveDeleteOFriend aid1 sp uid uid')) outOK s2'*
<proof>

fun *tgtNodeOfO* :: *apiID* \Rightarrow (*act* \times *out*) \Rightarrow *apiID* **where**

tgtNodeOfO - (*COMact (comSendServerReq uID p aID reqInfo), ou*) = *aID*
| *tgtNodeOfO* - (*COMact (comReceiveClientReq aID reqInfo), ou*) = *aID*
| *tgtNodeOfO* - (*COMact (comConnectClient uID p aID sp), ou*) = *aID*
| *tgtNodeOfO* - (*COMact (comConnectServer aID sp), ou*) = *aID*
| *tgtNodeOfO* - (*COMact (comSendPost uID p aID nID), ou*) = *aID*
| *tgtNodeOfO* - (*COMact (comReceivePost aID sp nID title text v), ou*) = *aID*
| *tgtNodeOfO* - (*COMact (comSendCreateOFriend uID p aID uID'), ou*) = *aID*
| *tgtNodeOfO* - (*COMact (comReceiveCreateOFriend aID sp uid uid'), ou*) = *aID*
| *tgtNodeOfO* - (*COMact (comSendDeleteOFriend uID p aID uID'), ou*) = *aID*
| *tgtNodeOfO* - (*COMact (comReceiveDeleteOFriend aID sp uid uid'), ou*) = *aID*
| *tgtNodeOfO* - - = *undefined*

fun *tgtNodeOf* :: *apiID* \Rightarrow (*state*, *act*, *out*) *trans* \Rightarrow *apiID* **where**

tgtNodeOf - (*Trans s (COMact (comSendServerReq uID p aID reqInfo)) ou s'*) = *aID*
| *tgtNodeOf* - (*Trans s (COMact (comReceiveClientReq aID reqInfo)) ou s'*) = *aID*
| *tgtNodeOf* - (*Trans s (COMact (comConnectClient uID p aID sp)) ou s'*) = *aID*
| *tgtNodeOf* - (*Trans s (COMact (comConnectServer aID sp)) ou s'*) = *aID*
| *tgtNodeOf* - (*Trans s (COMact (comSendPost uID p aID nID)) ou s'*) = *aID*

```

| tgtNodeOf - (Trans s (COMact (comReceivePost aID sp nID title text v)) ou s')
= aID
| tgtNodeOf - (Trans s (COMact (comSendCreateOFriend uID p aID uID')) ou s')
= aID
| tgtNodeOf - (Trans s (COMact (comReceiveCreateOFriend aID sp uid uid')) ou
s') = aID
| tgtNodeOf - (Trans s (COMact (comSendDeleteOFriend uID p aID uID')) ou s')
= aID
| tgtNodeOf - (Trans s (COMact (comReceiveDeleteOFriend aID sp uid uid')) ou
s') = aID
| tgtNodeOf - - = undefined

```

abbreviation *validTrans* :: *apiID* ⇒ (*state*, *act*, *out*) *trans* ⇒ *bool* **where**
validTrans *aid* ≡ *System-Specification.validTrans*

sublocale *TS-Network*

where *istate* = λ-. *istate* **and** *validTrans* = *validTrans* **and** *srcOf* = λ-. *srcOf*
and *tgtOf* = λ-. *tgtOf*
and *nodes* = *AIDs* **and** *comOf* = *comOf* **and** *tgtNodeOf* = *tgtNodeOf*
and *sync* = *sync*
⟨*proof*⟩

end

end

theory *Automation-Setup*

imports *System-Specification*

begin

lemma *add-prop*:

assumes *PROP* (*T*)

shows *A* ==> *PROP* (*T*)

⟨*proof*⟩

lemmas *exhaust-elim* =

sActt.exhaust[*of* *x*, *THEN add-prop*[**where** *A=a=Sact* *x*], *rotated* -1]

cActt.exhaust[*of* *x*, *THEN add-prop*[**where** *A=a=Cact* *x*], *rotated* -1]

uActt.exhaust[*of* *x*, *THEN add-prop*[**where** *A=a=Uact* *x*], *rotated* -1]

rActt.exhaust[*of* *x*, *THEN add-prop*[**where** *A=a=Ract* *x*], *rotated* -1]

lActt.exhaust[*of* *x*, *THEN add-prop*[**where** *A=a=Lact* *x*], *rotated* -1]

comActt.exhaust[*of* *x*, *THEN add-prop*[**where** *A=a=COMact* *x*], *rotated* -1]

for *x a*

lemma *state-cong*:

fixes *s::state*

assumes

pendingUReqs *s* = *pendingUReqs* *s1* ∧ *userReq* *s* = *userReq* *s1* ∧ *userIDs* *s* =

```

userIDs s1  $\wedge$ 
postIDs s = postIDs s1  $\wedge$  admin s = admin s1  $\wedge$ 
user s = user s1  $\wedge$  pass s = pass s1  $\wedge$  pendingFReqs s = pendingFReqs s1  $\wedge$ 
friendReq s = friendReq s1  $\wedge$  friendIDs s = friendIDs s1  $\wedge$ 
sentOuterFriendIDs s = sentOuterFriendIDs s1  $\wedge$ 
recvOuterFriendIDs s = recvOuterFriendIDs s1  $\wedge$ 
post s = post s1  $\wedge$ 
owner s = owner s1  $\wedge$  vis s = vis s1  $\wedge$ 
pendingSApiReqs s = pendingSApiReqs s1  $\wedge$  sApiReq s = sApiReq s1  $\wedge$  server-
ApiIDs s = serverApiIDs s1  $\wedge$  serverPass s = serverPass s1  $\wedge$ 
outerPostIDs s = outerPostIDs s1  $\wedge$  outerPost s = outerPost s1  $\wedge$ 
outerOwner s = outerOwner s1  $\wedge$  outerVis s = outerVis s1  $\wedge$ 
pendingCApiReqs s = pendingCApiReqs s1  $\wedge$  cApiReq s = cApiReq s1  $\wedge$  clien-
tApiIDs s = clientApiIDs s1  $\wedge$  clientPass s = clientPass s1  $\wedge$ 
sharedWith s = sharedWith s1
shows s = s1
<proof>

```

end

5 Safety properties

Here we prove some safety properties (state invariants) for a CoSMedis node that are needed in the proof of BD Security properties.

theory *Safety-Properties*

imports

Automation-Setup

begin

declare *Let-def*[simp]

declare *if-splits*[split]

declare *IDsOK-def*[simp]

lemmas *eff-defs* = *s-defs* *c-defs* *d-defs* *u-defs*

lemmas *obs-defs* = *r-defs* *l-defs*

lemmas *effc-defs* = *eff-defs* *com-defs*

lemmas *all-defs* = *effc-defs* *obs-defs*

declare *sstep-Cons*[simp]

lemma *Lact-Ract-noStateChange*[simp]:

assumes $a \in \text{Lact} \cup \text{UNIV} \cup \text{Ract} \cup \text{UNIV}$

shows $\text{snd}(\text{step } s \ a) = s$

<proof>

lemma *Lact-Ract-noStateChange-set*:

assumes $set\ al \subseteq Lact \text{ ' } UNIV \cup Ract \text{ ' } UNIV$

shows $snd\ (sstep\ s\ al) = s$

$\langle proof \rangle$

lemma *reach-postIDs-persist*:

$pID \in\in\ postIDs\ s \implies step\ s\ a = (ou, s') \implies pID \in\in\ postIDs\ s'$

$\langle proof \rangle$

lemma *userOfA-not-userIDs-outErr*:

$\exists\ uid.\ userOfA\ a = Some\ uid \wedge \neg\ uid \in\in\ userIDs\ s \implies$

$\forall\ aID\ uID\ p\ name.\ a \neq Sact\ (sSys\ uID\ p) \implies$

$\forall\ uID\ name.\ a \neq Cact\ (cNUReq\ uID\ name) \implies$

$fst\ (step\ s\ a) = outErr$

$\langle proof \rangle$

lemma *reach-vis*: $reach\ s \implies vis\ s\ pID \in\ \{FriendV, PublicV\}$

$\langle proof \rangle$

lemma *reach-not-postIDs-emptyPost*:

$reach\ s \implies PID \notin\ set\ (postIDs\ s) \implies post\ s\ PID = emptyPost$

$\langle proof \rangle$

lemma *reach-not-postIDs-friendV*:

$reach\ s \implies PID \notin\ set\ (postIDs\ s) \implies vis\ s\ PID = FriendV$

$\langle proof \rangle$

lemma *reach-owner-userIDs*: $reach\ s \implies pID \in\in\ postIDs\ s \implies owner\ s\ pID \in\in\ userIDs\ s$

$\langle proof \rangle$

lemma *reach-admin-userIDs*: $reach\ s \implies uID \in\in\ userIDs\ s \implies admin\ s \in\in\ userIDs\ s$

$\langle proof \rangle$

lemma *reach-pendingUReqs-distinct*: $reach\ s \implies distinct\ (pendingUReqs\ s)$

$\langle proof \rangle$

lemma *reach-pendingUReqs*:

$reach\ s \implies uid \in\in\ pendingUReqs\ s \implies uid \notin\ set\ (userIDs\ s) \wedge admin\ s \in\in\ userIDs\ s$

$\langle proof \rangle$

lemma *reach-friendIDs-symmetric*:

$reach\ s \implies uID1 \in\in\ friendIDs\ s\ uID2 \iff uID2 \in\in\ friendIDs\ s\ uID1$

$\langle proof \rangle$

lemma *reach-distinct-friends-reqs*:

assumes *reach s*

shows *distinct (friendIDs s uid) and distinct (pendingFReqs s uid)*
and *distinct (sentOuterFriendIDs s uid) and distinct (recvOuterFriendIDs s uid)*
and $uid' \in \text{pendingFReqs } s \text{ uid} \implies uid' \notin \text{set (friendIDs } s \text{ uid)}$
and $uid' \in \text{pendingFReqs } s \text{ uid} \implies uid' \notin \text{set (friendIDs } s \text{ uid')}$
{proof}

lemma *remove1-in-set*: $x \in \text{remove1 } y \text{ xs} \implies x \in \text{xs}$

{proof}

lemma *reach-IDs-used-IDsOK*[rule-format]:

assumes *reach s*

shows $uid \in \text{pendingFReqs } s \text{ uid}' \longrightarrow \text{IDsOK } s \text{ [uid, uid'] [] [] (is ?p)}$

and $uid \in \text{friendIDs } s \text{ uid}' \longrightarrow \text{IDsOK } s \text{ [uid, uid'] [] [] (is ?f)}$

{proof}

lemma *reach-AID-used-valid*:

assumes *reach s*

and $aid \in \text{serverApiIDs } s \vee aid \in \text{clientApiIDs } s \vee aid \in \text{pendingSApiReqs } s$
 $\vee aid \in \text{pendingCApiReqs } s$

shows $\text{admin } s \in \text{userIDs } s$

{proof}

lemma *IDs-mono*[rule-format]:

assumes *step s a = (ou, s')*

shows $uid \in \text{userIDs } s \longrightarrow uid \in \text{userIDs } s' \text{ (is ?u)}$

and $nid \in \text{postIDs } s \longrightarrow nid \in \text{postIDs } s' \text{ (is ?n)}$

and $aid \in \text{clientApiIDs } s \longrightarrow aid \in \text{clientApiIDs } s' \text{ (is ?c)}$

and $sid \in \text{serverApiIDs } s \longrightarrow sid \in \text{serverApiIDs } s' \text{ (is ?s)}$

and $nid \in \text{outerPostIDs } s \text{ aid} \longrightarrow nid \in \text{outerPostIDs } s' \text{ aid (is ?o)}$

{proof}

lemma *IDsOK-mono*:

assumes *step s a = (ou, s')*

and $\text{IDsOK } s \text{ uIDs pIDs saID-pIDs-s caIDs}$

shows $\text{IDsOK } s' \text{ uIDs pIDs saID-pIDs-s caIDs}$

{proof}

lemma *step-outerFriendIDs-idem*:

assumes *step s a = (ou, s')*

and $\forall uID \text{ p aID uID}'. a \neq \text{COMact (comSendCreateOFriend uID p aID uID')} \wedge$
 $a \neq \text{COMact (comReceiveCreateOFriend aID p uID uID')} \wedge$
 $a \neq \text{COMact (comSendDeleteOFriend uID p aID uID')} \wedge$
 $a \neq \text{COMact (comReceiveDeleteOFriend aID p uID uID')}$

shows $\text{sentOuterFriendIDs } s' = \text{sentOuterFriendIDs } s \text{ (is ?sent)}$

and $recvOuterFriendIDs\ s' = recvOuterFriendIDs\ s$ (**is** $?recv$)
 $\langle proof \rangle$

lemma $istate-sSys$:
assumes $step\ istate\ a = (ou, s')$
obtains $uid\ p$ **where** $a = Sact\ (sSys\ uid\ p)$
 $\quad | s' = istate$
 $\langle proof \rangle$

end
theory $Post-Intro$
imports $../Safety-Properties$
begin

6 Post confidentiality

We verify the following BD Security property of the CoSMedis network:

Given a coalition consisting of groups of users $UIDs\ j$ from multiple nodes j and given a post PID at node i ,

the coalition cannot learn anything about the updates to this post beyond those updates performed while or last before one of the following holds:

- (1) Some user in $UIDs\ i$ is the admin at node i , is the owner of PID or is friends with the owner of PID
- (2) PID is marked as public

unless some user in $UIDs\ j$ for a node j different than i is admin of node j or is remote friend with the owner of PID .³

As explained in [3], in order to prove this property for the CoSMedis network, we compose BD security properties of individual CoSMedis nodes. When formulating the individual node properties, we will distinguish between the *secret issuer* node i and the (potential) *secret receiver* nodes: all nodes different from i . Consequently, we will have two BD security properties – for issuers and for receivers – proved in their corresponding subsections. Then we prove BD Security for the (binary) composition of an issuer and a receiver node, and finally we prove BD Security for the n-ary composition (of an entire CoSMedis network of nodes).

Described above is the property in a form that employs a dynamic trigger

³So $UIDs$ is a function from node identifiers (called API IDs in this formalization) to sets of user IDs. We will write AID instead of i (which will be fixed in our locales) and aid instead of j .

(i.e., an inductive bound that incorporates an iterated trigger) for the secret issuer node. However, the first subsections of this section cover the static version of this (multi-node) property, corresponding to a static BD security property for the secret issuer. The dynamic version is covered after that, in a dedicated subsection.

Finally, we lift the above BD security property, which refers to a single secret source, i.e., a post at some node, to simultaneous BD Security for two independent secret sources, i.e., two different posts at two (possibly different) nodes. For this, we use the BD Security system compositionality and transport theorems formalized in the AFP entry [5]. More details about this approach can be found in [3]; in particular, Appendix A from that paper discusses the transport theorem.

end

theory *Post-Observation-Setup-ISSUER*
imports *Post-Intro*
begin

6.1 Confidentiality for a secret issuer node

We verify that a group of users of a given node i can learn nothing about the updates to the content of a post PID located at that node beyond the existence of an update unless one of them is the admin or the owner of PID , or becomes friends with the owner, or PID is marked as public. This is formulated as a BD Security property and is proved by unwinding.

See [3] for more details.

6.1.1 Observation setup

type-synonym $obs = act * out$

locale *Fixed-UIDs* = **fixes** $UIDs :: userID\ set$

locale *Fixed-PID* = **fixes** $PID :: postID$

locale *ObservationSetup-ISSUER* = $Fixed-UIDs + Fixed-PID$
begin

fun $\gamma :: (state, act, out) trans \Rightarrow bool$ **where**
 $\gamma (Trans - a - -) \longleftrightarrow$
 $(\exists uid. userOfA\ a = Some\ uid \wedge uid \in UIDs)$
 \vee
 $(\exists ca. a = COMact\ ca)$
 \vee

$(\exists uid p. a = Sact (sSys uid p))$

fun *sPurge* :: *sActt* \Rightarrow *sActt* **where**
sPurge (*sSys uid pwd*) = *sSys uid emptyPass*

fun *comPurge* :: *comActt* \Rightarrow *comActt* **where**
comPurge (*comSendServerReq uID p aID reqInfo*) = *comSendServerReq uID emptyPass aID reqInfo*
|i*comPurge* (*comConnectClient uID p aID sp*) = *comConnectClient uID emptyPass aID sp*
|i*comPurge* (*comConnectServer aID sp*) = *comConnectServer aID sp*
|i*comPurge* (*comSendPost uID p aID pID*) = *comSendPost uID emptyPass aID pID*
|i*comPurge* (*comSendCreateOFriend uID p aID uID'*) = *comSendCreateOFriend uID emptyPass aID uID'*
|i*comPurge* (*comSendDeleteOFriend uID p aID uID'*) = *comSendDeleteOFriend uID emptyPass aID uID'*
|i*comPurge* *ca* = *ca*

fun *outPurge* :: *out* \Rightarrow *out* **where**
outPurge (*O-sendPost (aID, sp, pID, pst, uID, vs)*) =
(*let* *pst'* = (*if* *pID* = *PID* *then* *emptyPost* *else* *pst*)
in *O-sendPost (aID, sp, pID, pst', uID, vs)*)
|i*outPurge* *ou* = *ou*

fun *g* :: (*state,act,out*)*trans* \Rightarrow *obs* **where**
g (*Trans* - (*Sact sa*) *ou* -) = (*Sact (sPurge sa)*, *outPurge ou*)
|i*g* (*Trans* - (*COMact ca*) *ou* -) = (*COMact (comPurge ca)*, *outPurge ou*)
|i*g* (*Trans* - *a* *ou* -) = (*a,ou*)

lemma *comPurge-simps*:

comPurge ca = comSendServerReq uID p aID reqInfo \longleftrightarrow ($\exists p'$. *ca = comSendServerReq uID p' aID reqInfo* \wedge *p = emptyPass*)
comPurge ca = comReceiveClientReq aID reqInfo \longleftrightarrow *ca = comReceiveClientReq aID reqInfo*
comPurge ca = comConnectClient uID p aID sp \longleftrightarrow ($\exists p'$. *ca = comConnectClient uID p' aID sp* \wedge *p = emptyPass*)
comPurge ca = comConnectServer aID sp \longleftrightarrow *ca = comConnectServer aID sp*
comPurge ca = comReceivePost aID sp nID nt uID v \longleftrightarrow *ca = comReceivePost aID sp nID nt uID v*
comPurge ca = comSendPost uID p aID nID \longleftrightarrow ($\exists p'$. *ca = comSendPost uID p' aID nID* \wedge *p = emptyPass*)
comPurge ca = comSendCreateOFriend uID p aID uID' \longleftrightarrow ($\exists p'$. *ca = comSendCreateOFriend uID p' aID uID'* \wedge *p = emptyPass*)
comPurge ca = comReceiveCreateOFriend aID cp uID uID' \longleftrightarrow *ca = comRe-*

$ceiveCreateOFriend\ aID\ cp\ uID\ uID'$
 $comPurge\ ca = comSendDeleteOFriend\ uID\ p\ aID\ uID' \longleftrightarrow (\exists p'. ca = comSendDeleteOFriend\ uID\ p'\ aID\ uID' \wedge p = emptyPass)$
 $comPurge\ ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'$
 <proof>

lemma *outPurge-simps*[simp]:

$outPurge\ ou = outErr \longleftrightarrow ou = outErr$
 $outPurge\ ou = outOK \longleftrightarrow ou = outOK$
 $outPurge\ ou = O-sendServerReq\ ossr \longleftrightarrow ou = O-sendServerReq\ ossr$
 $outPurge\ ou = O-connectClient\ occ \longleftrightarrow ou = O-connectClient\ occ$
 $outPurge\ ou = O-sendPost\ (aid,\ sp,\ pid,\ pst',\ uid,\ vs) \longleftrightarrow (\exists pst.\ ou = O-sendPost\ (aid,\ sp,\ pid,\ pst,\ uid,\ vs) \wedge pst' = (if\ pid = PID\ then\ emptyPost\ else\ pst))$
 $outPurge\ ou = O-sendCreateOFriend\ oscf \longleftrightarrow ou = O-sendCreateOFriend\ oscf$
 $outPurge\ ou = O-sendDeleteOFriend\ osdf \longleftrightarrow ou = O-sendDeleteOFriend\ osdf$
 <proof>

lemma *g-simps*:

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendServerReq\ uID\ p\ aID\ reqInfo),\ O-sendServerReq\ ossr)$
 $\longleftrightarrow (\exists p'. a = COMact\ (comSendServerReq\ uID\ p'\ aID\ reqInfo) \wedge p = emptyPass \wedge ou = O-sendServerReq\ ossr)$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveClientReq\ aID\ reqInfo),\ outOK)$
 $\longleftrightarrow a = COMact\ (comReceiveClientReq\ aID\ reqInfo) \wedge ou = outOK$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectClient\ uID\ p\ aID\ sp),\ O-connectClient\ occ)$
 $\longleftrightarrow (\exists p'. a = COMact\ (comConnectClient\ uID\ p'\ aID\ sp) \wedge p = emptyPass \wedge ou = O-connectClient\ occ)$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectServer\ aID\ sp),\ outOK)$
 $\longleftrightarrow a = COMact\ (comConnectServer\ aID\ sp) \wedge ou = outOK$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceivePost\ aID\ sp\ nID\ nt\ uID\ v),\ outOK)$
 $\longleftrightarrow a = COMact\ (comReceivePost\ aID\ sp\ nID\ nt\ uID\ v) \wedge ou = outOK$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendPost\ uID\ p\ aID\ nID),\ O-sendPost\ (aid,\ sp,\ pid,\ pst',\ uid,\ vs))$
 $\longleftrightarrow (\exists pst\ p'. a = COMact\ (comSendPost\ uID\ p'\ aID\ nID) \wedge p = emptyPass \wedge ou = O-sendPost\ (aid,\ sp,\ pid,\ pst,\ uid,\ vs) \wedge pst' = (if\ pid = PID\ then\ emptyPost\ else\ pst))$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendCreateOFriend\ uID\ p\ aID\ uID'),\ O-sendCreateOFriend\ (aid,\ sp,\ uid,\ uid'))$
 $\longleftrightarrow (\exists p'. a = (COMact\ (comSendCreateOFriend\ uID\ p'\ aID\ uID')) \wedge p = emptyPass \wedge ou = O-sendCreateOFriend\ (aid,\ sp,\ uid,\ uid'))$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID'),\ outOK)$
 $\longleftrightarrow a = COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID') \wedge ou = outOK$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendDeleteOFriend\ uID\ p\ aID\ uID'),\ O-sendDeleteOFriend\ (aid,\ sp,\ uid,\ uid'))$

$\longleftrightarrow (\exists p'. a = \text{COMact} (\text{comSendDeleteOFriend } uID \ p' \ aID \ uID') \wedge p = \text{empty-Pass} \wedge ou = \text{O-sendDeleteOFriend} (aid, sp, uid, uid'))$
 $g (\text{Trans } s \ a \ ou \ s') = (\text{COMact} (\text{comReceiveDeleteOFriend } aID \ cp \ uID \ uID'), \text{outOK})$
 $\longleftrightarrow a = \text{COMact} (\text{comReceiveDeleteOFriend } aID \ cp \ uID \ uID') \wedge ou = \text{outOK}$
 <proof>

end

end

theory *Post-Unwinding-Helper-ISSUER*

imports *Post-Observation-Setup-ISSUER*

begin

locale *Issuer-State-Equivalence-Up-To-PID = Fixed-PID*

begin

6.1.2 Unwinding helper lemmas and definitions

definition *eqButPID* **where**

eqButPID *psts psts1* \equiv

$\forall pid. \text{if } pid = PID \text{ then True else } psts \ pid = psts1 \ pid$

lemmas *eqButPID-intro* = *eqButPID-def*[*THEN meta-eq-to-obj-eq, THEN iffD2*]

lemma *eqButPID-eq*[*simp,intro*]: *eqButPID* *psts psts*

<proof>

lemma *eqButPID-sym*:

assumes *eqButPID* *psts psts1* **shows** *eqButPID* *psts1 psts*

<proof>

lemma *eqButPID-trans*:

assumes *eqButPID* *psts psts1* **and** *eqButPID* *psts1 psts2* **shows** *eqButPID* *psts psts2*

<proof>

lemma *eqButPID-cong*:

assumes *eqButPID* *psts psts1*

and $pid = PID \implies uu = uu1$

and $pid \neq PID \implies uu = uu1$

shows *eqButPID* (*psts* ($pid := uu$)) (*psts1* ($pid := uu1$))

<proof>

lemma *eqButPID-not-PID*:

$\llbracket \text{eqButPID } psts \ psts1; \ pid \neq PID \rrbracket \implies psts \ pid = psts1 \ pid$

<proof>

lemma *eeqButPID-toEq*:
assumes *eeqButPID psts psts1*
shows $psts (PID := pid) =$
 $psts1 (PID := pid)$
 $\langle proof \rangle$

lemma *eeqButPID-update-post*:
assumes *eeqButPID psts psts1*
shows $eeqButPID (psts (pid := pst)) (psts1 (pid := pst))$
 $\langle proof \rangle$

fun *eqButF* :: $(apiID \times bool) list \Rightarrow (apiID \times bool) list \Rightarrow bool$ **where**
 $eqButF aID-bl aID-bl1 = (map fst aID-bl = map fst aID-bl1)$

lemma *eqButF-eq[simp,intro!]*: $eqButF aID-bl aID-bl$
 $\langle proof \rangle$

lemma *eqButF-sym*:
assumes $eqButF aID-bl aID-bl1$
shows $eqButF aID-bl1 aID-bl$
 $\langle proof \rangle$

lemma *eqButF-trans*:
assumes $eqButF aID-bl aID-bl1$ **and** $eqButF aID-bl1 aID-bl2$
shows $eqButF aID-bl aID-bl2$
 $\langle proof \rangle$

lemma *eqButF-insert2*:
 $eqButF aID-bl aID-bl1 \implies$
 $eqButF (insert2 aID b aID-bl) (insert2 aID b aID-bl1)$
 $\langle proof \rangle$

definition *eeqButPID-F* **where**
 $eeqButPID-F sw sw1 \equiv$
 $\forall pid. \text{if } pid = PID \text{ then } eqButF (sw PID) (sw1 PID) \text{ else } sw pid = sw1 pid$

lemmas $eeqButPID-F-intro = eeqButPID-F-def[THEN meta-eq-to-obj-eq, THEN iffD2]$

lemma *eeqButPID-F-eeq[simp,intro!]*: $eeqButPID-F sw sw$
 $\langle proof \rangle$

lemma *eeqButPID-F-sym*:
assumes *eeqButPID-F sw sw1* **shows** *eeqButPID-F sw1 sw*
 ⟨*proof*⟩

lemma *eeqButPID-F-trans*:
assumes *eeqButPID-F sw sw1* **and** *eeqButPID-F sw1 sw2* **shows** *eeqButPID-F sw sw2*
 ⟨*proof*⟩

lemma *eeqButPID-F-cong*:
assumes *eeqButPID-F sw sw1*
and $PID = PID \implies eqButF uu uu1$
and $pid \neq PID \implies uu = uu1$
shows *eeqButPID-F (sw (pid := uu)) (sw1 (pid := uu1))*
 ⟨*proof*⟩

lemma *eeqButPID-F-eqButF*:
eeqButPID-F sw sw1 $\implies eqButF (sw PID) (sw1 PID)$
 ⟨*proof*⟩

lemma *eeqButPID-F-not-PID*:
 $\llbracket eeqButPID-F sw sw1; pid \neq PID \rrbracket \implies sw pid = sw1 pid$
 ⟨*proof*⟩

lemma *eeqButPID-F-postSelectors*:
eeqButPID-F sw sw1 $\implies map fst (sw pid) = map fst (sw1 pid)$
 ⟨*proof*⟩

lemma *eeqButPID-F-insert2*:
eeqButPID-F sw sw1 \implies
 $eqButF (insert2 aID b (sw PID)) (insert2 aID b (sw1 PID))$
 ⟨*proof*⟩

lemma *eeqButPID-F-toEq*:
assumes *eeqButPID-F sw sw1*
shows $sw (PID := map (\lambda (aID,-). (aID,b)) (sw PID)) =$
 $sw1 (PID := map (\lambda (aID,-). (aID,b)) (sw1 PID))$
 ⟨*proof*⟩

lemma *eeqButPID-F-updateShared*:
assumes *eeqButPID-F sw sw1*
shows *eeqButPID-F (sw (pid := aID-b)) (sw1 (pid := aID-b))*
 ⟨*proof*⟩

definition *eqButPID* :: *state* \Rightarrow *state* \Rightarrow *bool* **where**
eqButPID s s1 \equiv
admin s = admin s1 \wedge

$pendingUReqs\ s = pendingUReqs\ s1 \wedge userReq\ s = userReq\ s1 \wedge$
 $userIDs\ s = userIDs\ s1 \wedge user\ s = user\ s1 \wedge pass\ s = pass\ s1 \wedge$

$pendingFReqs\ s = pendingFReqs\ s1 \wedge friendReq\ s = friendReq\ s1 \wedge friendIDs\ s$
 $= friendIDs\ s1 \wedge$
 $sentOuterFriendIDs\ s = sentOuterFriendIDs\ s1 \wedge recvOuterFriendIDs\ s = recvOuter-$
 $FriendIDs\ s1 \wedge$

$postIDs\ s = postIDs\ s1 \wedge admin\ s = admin\ s1 \wedge$
 $eqButPID\ (post\ s)\ (post\ s1) \wedge$
 $owner\ s = owner\ s1 \wedge$
 $vis\ s = vis\ s1 \wedge$

$pendingSApiReqs\ s = pendingSApiReqs\ s1 \wedge sApiReq\ s = sApiReq\ s1 \wedge$
 $serverApiIDs\ s = serverApiIDs\ s1 \wedge serverPass\ s = serverPass\ s1 \wedge$
 $outerPostIDs\ s = outerPostIDs\ s1 \wedge outerPost\ s = outerPost\ s1 \wedge$
 $outerOwner\ s = outerOwner\ s1 \wedge$
 $outerVis\ s = outerVis\ s1 \wedge$

$pendingCApiReqs\ s = pendingCApiReqs\ s1 \wedge cApiReq\ s = cApiReq\ s1 \wedge$
 $clientApiIDs\ s = clientApiIDs\ s1 \wedge clientPass\ s = clientPass\ s1 \wedge$
 $eqButPID-F\ (sharedWith\ s)\ (sharedWith\ s1)$

lemmas $eqButPID-intro = eqButPID-def[THEN\ meta-eq-to-obj-eq,\ THEN\ iffD2]$

lemma $eqButPID-refl[simp,intro!]$: $eqButPID\ s\ s$
 $\langle proof \rangle$

lemma $eqButPID-sym$:
assumes $eqButPID\ s\ s1$ **shows** $eqButPID\ s1\ s$
 $\langle proof \rangle$

lemma $eqButPID-trans$:
assumes $eqButPID\ s\ s1$ **and** $eqButPID\ s1\ s2$ **shows** $eqButPID\ s\ s2$
 $\langle proof \rangle$

lemma $eqButPID-stateSelectors$:
 $eqButPID\ s\ s1 \implies$
 $admin\ s = admin\ s1 \wedge$

$pendingUReqs\ s = pendingUReqs\ s1 \wedge userReq\ s = userReq\ s1 \wedge$
 $userIDs\ s = userIDs\ s1 \wedge user\ s = user\ s1 \wedge pass\ s = pass\ s1 \wedge$

$pendingFReqs\ s = pendingFReqs\ s1 \wedge friendReq\ s = friendReq\ s1 \wedge friendIDs\ s$
 $= friendIDs\ s1 \wedge$
 $sentOuterFriendIDs\ s = sentOuterFriendIDs\ s1 \wedge recvOuterFriendIDs\ s = recvOuter-$
 $FriendIDs\ s1 \wedge$

$postIDs\ s = postIDs\ s1 \wedge admin\ s = admin\ s1 \wedge$
 $eqButPID\ (post\ s)\ (post\ s1) \wedge$
 $owner\ s = owner\ s1 \wedge$
 $vis\ s = vis\ s1 \wedge$

$pendingSApiReqs\ s = pendingSApiReqs\ s1 \wedge sApiReq\ s = sApiReq\ s1 \wedge$
 $serverApiIDs\ s = serverApiIDs\ s1 \wedge serverPass\ s = serverPass\ s1 \wedge$
 $outerPostIDs\ s = outerPostIDs\ s1 \wedge outerPost\ s = outerPost\ s1 \wedge$
 $outerOwner\ s = outerOwner\ s1 \wedge$
 $outerVis\ s = outerVis\ s1 \wedge$

$pendingCApiReqs\ s = pendingCApiReqs\ s1 \wedge cApiReq\ s = cApiReq\ s1 \wedge$
 $clientApiIDs\ s = clientApiIDs\ s1 \wedge clientPass\ s = clientPass\ s1 \wedge$
 $eqButPID-F\ (sharedWith\ s)\ (sharedWith\ s1) \wedge$

$IDsOK\ s = IDsOK\ s1$
 $\langle proof \rangle$

lemma $eqButPID-not-PID$:
 $eqButPID\ s\ s1 \implies pid \neq PID \implies post\ s\ pid = post\ s1\ pid$
 $\langle proof \rangle$

lemma $eqButPID-eqButF$:
 $eqButPID\ s\ s1 \implies eqButF\ (sharedWith\ s\ PID)\ (sharedWith\ s1\ PID)$
 $\langle proof \rangle$

lemma $eqButPID-not-PID-sharedWith$:
 $eqButPID\ s\ s1 \implies pid \neq PID \implies sharedWith\ s\ pid = sharedWith\ s1\ pid$
 $\langle proof \rangle$

lemma $eqButPID-insert2$:
 $eqButPID\ s\ s1 \implies$
 $eqButF\ (insert2\ aID\ b\ (sharedWith\ s\ PID))\ (insert2\ aID\ b\ (sharedWith\ s1\ PID))$
 $\langle proof \rangle$

lemma $eqButPID-actions$:
assumes $eqButPID\ s\ s1$
shows $listInnerPosts\ s\ uid\ p = listInnerPosts\ s1\ uid\ p$
 $\langle proof \rangle$

lemma *eqButPID-update*:
assumes *eqButPID s s1*
shows $(\text{post } s)(\text{PID} := \text{txt}) = (\text{post } s1)(\text{PID} := \text{txt})$
 $\langle \text{proof} \rangle$

lemma *eqButPID-update-post*:
assumes *eqButPID s s1*
shows $\text{eeqButPID } ((\text{post } s) (\text{pid} := \text{pst})) ((\text{post } s1) (\text{pid} := \text{pst}))$
 $\langle \text{proof} \rangle$

lemma *eqButPID-setShared*:
assumes *eqButPID s s1*
shows $(\text{sharedWith } s) (\text{PID} := \text{map } (\lambda (aID,-). (aID,b)) (\text{sharedWith } s \text{PID})) =$
 $(\text{sharedWith } s1) (\text{PID} := \text{map } (\lambda (aID,-). (aID,b)) (\text{sharedWith } s1 \text{PID}))$
 $\langle \text{proof} \rangle$

lemma *eqButPID-updateShared*:
assumes *eqButPID s s1*
shows $\text{eeqButPID-F } ((\text{sharedWith } s) (\text{pid} := aID-b)) ((\text{sharedWith } s1) (\text{pid} := aID-b))$
 $\langle \text{proof} \rangle$

lemma *eqButPID-cong[simp]*:
 $\bigwedge uu1 uu2. \text{eqButPID } s s1 \implies uu1 = uu2 \implies \text{eqButPID } (s (\text{admin} := uu1)) (s1 (\text{admin} := uu2))$

$\bigwedge uu1 uu2. \text{eqButPID } s s1 \implies uu1 = uu2 \implies \text{eqButPID } (s (\text{pendingUReqs} := uu1)) (s1 (\text{pendingUReqs} := uu2))$

$\bigwedge uu1 uu2. \text{eqButPID } s s1 \implies uu1 = uu2 \implies \text{eqButPID } (s (\text{userReq} := uu1)) (s1 (\text{userReq} := uu2))$

$\bigwedge uu1 uu2. \text{eqButPID } s s1 \implies uu1 = uu2 \implies \text{eqButPID } (s (\text{userIDs} := uu1)) (s1 (\text{userIDs} := uu2))$

$\bigwedge uu1 uu2. \text{eqButPID } s s1 \implies uu1 = uu2 \implies \text{eqButPID } (s (\text{user} := uu1)) (s1 (\text{user} := uu2))$

$\bigwedge uu1 uu2. \text{eqButPID } s s1 \implies uu1 = uu2 \implies \text{eqButPID } (s (\text{pass} := uu1)) (s1 (\text{pass} := uu2))$

$\bigwedge uu1 uu2. \text{eqButPID } s s1 \implies uu1 = uu2 \implies \text{eqButPID } (s (\text{postIDs} := uu1)) (s1 (\text{postIDs} := uu2))$

$\bigwedge uu1 uu2. \text{eqButPID } s s1 \implies \text{eeqButPID } uu1 uu2 \implies \text{eqButPID } (s (\text{post} := uu1)) (s1 (\text{post} := uu2))$

$\bigwedge uu1 uu2. \text{eqButPID } s s1 \implies uu1 = uu2 \implies \text{eqButPID } (s (\text{owner} := uu1)) (s1 (\text{owner} := uu2))$

$\bigwedge uu1 uu2. \text{eqButPID } s s1 \implies uu1 = uu2 \implies \text{eqButPID } (s (\text{vis} := uu1)) (s1 (\text{vis} := uu2))$

$\bigwedge uu1 uu2. \text{eqButPID } s s1 \implies uu1 = uu2 \implies \text{eqButPID } (s (\text{pendingFReqs} := uu1)) (s1 (\text{pendingFReqs} := uu2))$

$\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash friendReq := uu1))$
 $(s1 (\backslash friendReq := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash friendIDs := uu1))$
 $(s1 (\backslash friendIDs := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash sentOuterFriendIDs$
 $:= uu1)) (s1 (\backslash sentOuterFriendIDs := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash recvOuterFriendIDs$
 $:= uu1)) (s1 (\backslash recvOuterFriendIDs := uu2))$

$\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash pendingSApiReqs$
 $:= uu1)) (s1 (\backslash pendingSApiReqs := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash sApiReq := uu1))$
 $(s1 (\backslash sApiReq := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash serverApiIDs :=$
 $uu1)) (s1 (\backslash serverApiIDs := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash serverPass := uu1))$
 $(s1 (\backslash serverPass := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash outerPostIDs :=$
 $uu1)) (s1 (\backslash outerPostIDs := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash outerPost := uu1))$
 $(s1 (\backslash outerPost := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash outerOwner :=$
 $uu1)) (s1 (\backslash outerOwner := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash outerVis := uu1))$
 $(s1 (\backslash outerVis := uu2))$

$\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash pendingCApiReqs$
 $:= uu1)) (s1 (\backslash pendingCApiReqs := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash cApiReq := uu1))$
 $(s1 (\backslash cApiReq := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash clientApiIDs :=$
 $uu1)) (s1 (\backslash clientApiIDs := uu2))$
 $\wedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\backslash clientPass := uu1))$
 $(s1 (\backslash clientPass := uu2))$

$\wedge uu1 uu2. eqButPID s s1 \implies eqButPID-F uu1 uu2 \implies eqButPID (s (\backslash shared-$
 $With := uu1)) (s1 (\backslash sharedWith := uu2))$
 $\langle proof \rangle$

lemma *eqButPID-step*:
assumes *ss1*: *eqButPID s s1*
and *step*: *step s a = (ou, s')*
and *step1*: *step s1 a = (ou1, s1')*
shows *eqButPID s' s1'*
 $\langle proof \rangle$

end

end

theory *Post-Value-Setup-ISSUER*

imports

../Safety-Properties

Post-Observation-Setup-ISSUER

Post-Unwinding-Helper-ISSUER

begin

locale *Post-ISSUER = ObservationSetup-ISSUER*

begin

6.1.3 Value setup

datatype *value =*

isPVal: PVal post — updating the post content locally

| *isPValS: PValS (PValS-tgtAPI: apiID) post* — sending the post to another node

lemma *filter-isPValS-Nil: filter isPValS vl = [] \longleftrightarrow list-all isPVal vl*

<proof>

fun $\varphi :: (state, act, out) trans \Rightarrow bool$ **where**

$\varphi (Trans - (Uact (uPost uid p pid pst)) ou -) = (pid = PID \wedge ou = outOK)$

|

$\varphi (Trans - (COMact (comSendPost uid p aid pid)) ou -) = (pid = PID \wedge ou \neq outErr)$

|

$\varphi (Trans s - - s') = False$

lemma φ -def2:

shows

$\varphi (Trans s a ou s') \longleftrightarrow$

$(\exists uid p pst. a = Uact (uPost uid p PID pst) \wedge ou = outOK) \vee$

$(\exists uid p aid. a = COMact (comSendPost uid p aid PID) \wedge ou \neq outErr)$

<proof>

lemma *uPost-out:*

assumes 1: *step s a = (ou, s')* **and** *a: a = Uact (uPost uid p PID pst)* **and** 2: *ou = outOK*

shows *uid = owner s PID \wedge p = pass s uid*

<proof>

lemma *comSendPost-out:*

assumes 1: *step s a = (ou, s')* **and** *a: a = COMact (comSendPost uid p aid PID)*

and 2: *ou \neq outErr*

shows *ou = O-sendPost (aid, clientPass s aid, PID, post s PID, owner s PID, vis s PID)*

$\wedge uid = admin s \wedge p = pass s (admin s)$

$\langle proof \rangle$

lemma φ -def3:

assumes $step\ s\ a = (ou, s')$

shows

$\varphi (Trans\ s\ a\ ou\ s') \longleftrightarrow$

$(\exists\ pst. a = Uact\ (uPost\ (owner\ s\ PID)\ (pass\ s\ (owner\ s\ PID))\ PID\ pst) \wedge ou = outOK) \vee$

$(\exists\ aid. a = COMact\ (comSendPost\ (admin\ s)\ (pass\ s\ (admin\ s))\ aid\ PID) \wedge ou = O-sendPost\ (aid, clientPass\ s\ aid, PID, post\ s\ PID, owner\ s\ PID, vis\ s\ PID))$

$\langle proof \rangle$

lemma φ -cases:

assumes $\varphi (Trans\ s\ a\ ou\ s')$

and $step\ s\ a = (ou, s')$

and $reach\ s$

obtains

$(UpdateT)\ uid\ p\ PID\ pst$ **where** $a = Uact\ (uPost\ uid\ p\ PID\ pst)\ ou = outOK\ p = pass\ s\ uid$

$uid = owner\ s\ PID$

$| (Send)\ uid\ p\ aid$ **where** $a = COMact\ (comSendPost\ uid\ p\ aid\ PID)\ ou \neq outErr\ p = pass\ s\ uid$

$uid = admin\ s$

$\langle proof \rangle$

fun $f :: (state, act, out)\ trans \Rightarrow value$ **where**

$f (Trans\ s\ (Uact\ (uPost\ uid\ p\ pid\ pst)) - s') =$

$(if\ pid = PID\ then\ PVal\ pst\ else\ undefined)$

$|$

$f (Trans\ s\ (COMact\ (comSendPost\ uid\ p\ aid\ pid)) (O-sendPost\ (-, -, -, pst, -, -)) s') =$

$(if\ pid = PID\ then\ PValS\ aid\ pst\ else\ undefined)$

$|$

$f (Trans\ s\ -\ -\ s') = undefined$

sublocale *Issuer-State-Equivalence-Up-To-PID* $\langle proof \rangle$

lemma *Uact-uPaperC-step-eqButPID*:

assumes $a = Uact\ (uPost\ uid\ p\ PID\ pst)$

and $step\ s\ a = (ou, s')$

shows $eqButPID\ s\ s'$

$\langle proof \rangle$

lemma *eqButPID-step- φ -imp*:

assumes $ss1: eqButPID\ s\ s1$

and $step: step\ s\ a = (ou, s')$ **and** $step1: step\ s1\ a = (ou1, s1')$

```

and  $\varphi: \varphi (Trans\ s\ a\ ou\ s')$ 
shows  $\varphi (Trans\ s1\ a\ ou1\ s1')$ 
 $\langle proof \rangle$ 

lemma eqButPID-step- $\varphi$ :
assumes  $s's1'$ : eqButPID  $s\ s1$ 
and step:  $step\ s\ a = (ou, s')$  and step1:  $step\ s1\ a = (ou1, s1')$ 
shows  $\varphi (Trans\ s\ a\ ou\ s') = \varphi (Trans\ s1\ a\ ou1\ s1')$ 
 $\langle proof \rangle$ 

end

end
theory Post-ISSUER
  imports
    Bounded-Deducibility-Security.Compositional-Reasoning
    Post-Observation-Setup-ISSUER
    Post-Value-Setup-ISSUER
begin

```

6.1.4 Issuer declassification bound

We verify that a group of users of some node i , allowed to take normal actions to the system and observe their outputs *and additionally allowed to observe communication*, can learn nothing about the updates to a post located at node i and the sends of that post to other nodes beyond

- (1) the presence of the sends (i.e., the number of the sending actions)
- (2) the public knowledge that what is being sent is always the last version (modeled as the correlation predicate)

unless:

- either a user in the group is the post's owner or the administrator
- or a user in the group becomes a friend of the owner
- or the group has at least one registered user and the post is being marked as "public"

See [3] for more details.

```

context Post-ISSUER
begin

fun  $T :: (state, act, out)\ trans \Rightarrow bool$  where
 $T (Trans\ s\ a\ ou\ s') \longleftrightarrow$ 
 $(\exists\ uid \in UIDs.$ 
   $uid \in userIDs\ s' \wedge PID \in postIDs\ s' \wedge$ 

```

$(uid = admin\ s' \vee$
 $uid = owner\ s'\ PID \vee$
 $uid \in\ friendIDs\ s'\ (owner\ s'\ PID) \vee$
 $vis\ s'\ PID = PublicV))$

fun *corrFrom* :: *post* \Rightarrow *value list* \Rightarrow *bool* **where**
 $corrFrom\ pst\ [] = True$
 $|corrFrom\ pst\ (PVal\ pstt\ \#\ vl) = corrFrom\ pstt\ vl$
 $|corrFrom\ pst\ (PValS\ aid\ pstt\ \#\ vl) = (pst = pstt \wedge corrFrom\ pst\ vl)$

abbreviation *corr* :: *value list* \Rightarrow *bool* **where** $corr \equiv corrFrom\ emptyPost$

definition *B* :: *value list* \Rightarrow *value list* \Rightarrow *bool* **where**
 $B\ vl\ vl1 \equiv$
 $corr\ vl1 \wedge$
 $(vl = [] \longrightarrow vl1 = []) \wedge$
 $map\ PValS\ tgtAPI\ (filter\ isPValS\ vl) = map\ PValS\ tgtAPI\ (filter\ isPValS\ vl1)$

sublocale *BD-Security-IO* **where**
 $istate = istate$ **and** $step = step$ **and**
 $\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
 $\langle proof \rangle$

6.1.5 Unwinding proof

lemma *reach-PublicV-imples-FriendV[simp]*:
assumes $reach\ s$
and $vis\ s\ pID \neq PublicV$
shows $vis\ s\ pID = FriendV$
 $\langle proof \rangle$

lemma *reachNT-state*:
assumes $reachNT\ s$
shows $\neg (\exists\ uid \in\ UIDs.$
 $uid \in\ userIDs\ s \wedge PID \in\ postIDs\ s \wedge$
 $(uid = admin\ s \vee uid = owner\ s\ PID \vee uid \in\ friendIDs\ s\ (owner\ s\ PID) \vee$
 $vis\ s\ PID = PublicV))$
 $\langle proof \rangle$

lemma *T- φ - γ* :
assumes $1: reachNT\ s$ **and** $2: step\ s\ a = (ou, s')$
and $3: \varphi\ (Trans\ s\ a\ ou\ s')$ **and**
 $4: \forall\ ca. a \neq COMact\ ca$
shows $\neg \gamma\ (Trans\ s\ a\ ou\ s')$
 $\langle proof \rangle$

lemma *eqButPID-step- γ -out*:
assumes *ss1*: *eqButPID s s1*
and *step*: *step s a = (ou, s')* **and** *step1*: *step s1 a = (ou1, s1')*
and *sT*: *reachNT s* **and** *T*: $\neg T$ (*Trans s a ou s'*)
and *s1*: *reach s1*
and γ : γ (*Trans s a ou s'*)
shows $(\exists \text{uid p aid pid. } a = \text{COMact} (\text{comSendPost uid p aid pid}) \wedge \text{outPurge ou} = \text{outPurge ou1}) \vee \text{ou} = \text{ou1}$
<proof>

lemma *eqButPID-step-eq*:
assumes *ss1*: *eqButPID s s1*
and *a*: *a = Uact (uPost uid p PID pst) ou = outOK*
and *step*: *step s a = (ou, s')* **and** *step1*: *step s1 a = (ou', s1')*
shows $s' = s1'$
<proof>

definition $\Delta 0 :: \text{state} \Rightarrow \text{value list} \Rightarrow \text{state} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**
 $\Delta 0 s vl s1 vl1 \equiv$
 $\neg \text{PID} \in \text{postIDs } s \wedge \text{post } s \text{ PID} = \text{emptyPost} \wedge$
 $s = s1 \wedge$
 $\text{corrFrom} (\text{post } s1 \text{ PID}) vl1 \wedge$
 $(vl = [] \longrightarrow vl1 = []) \wedge$
 $\text{map } \text{PValS-tgtAPI} (\text{filter isPValS } vl) = \text{map } \text{PValS-tgtAPI} (\text{filter isPValS } vl1)$

definition $\Delta 1 :: \text{state} \Rightarrow \text{value list} \Rightarrow \text{state} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**
 $\Delta 1 s vl s1 vl1 \equiv$
 $\text{PID} \in \text{postIDs } s \wedge$
 $\text{eqButPID } s s1 \wedge$
 $\text{corrFrom} (\text{post } s1 \text{ PID}) vl1 \wedge$
 $(vl = [] \longrightarrow vl1 = []) \wedge$
 $\text{map } \text{PValS-tgtAPI} (\text{filter isPValS } vl) = \text{map } \text{PValS-tgtAPI} (\text{filter isPValS } vl1)$

definition $\Delta 2 :: \text{state} \Rightarrow \text{value list} \Rightarrow \text{state} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**
 $\Delta 2 s vl s1 vl1 \equiv$
 $\text{PID} \in \text{postIDs } s \wedge$
 $\text{eqButPID } s s1 \wedge$
 $vl = [] \wedge \text{list-all isPVal } vl1$

lemma *istate- $\Delta 0$* :
assumes *B*: *B vl vl1*
shows $\Delta 0 \text{ istate } vl \text{ istate } vl1$
<proof>

lemma *unwind-cont- $\Delta 0$* : *unwind-cont $\Delta 0$ { $\Delta 0, \Delta 1$ }*
<proof>

lemma *unwind-cont- $\Delta 1$* : *unwind-cont $\Delta 1$ { $\Delta 1, \Delta 2$ }*

<proof>

lemma *unwind-cont- $\Delta 2$* : *unwind-cont $\Delta 2$ { $\Delta 2$ }*
<proof>

definition *Gr* where

Gr =
{
 ($\Delta 0$, { $\Delta 0, \Delta 1$ }),
 ($\Delta 1$, { $\Delta 1, \Delta 2$ }),
 ($\Delta 2$, { $\Delta 2$ })
}

theorem *Post-secure: secure*
<proof>

end

end

theory *Post-Observation-Setup-RECEIVER*

imports *../Safety-Properties*

begin

6.2 Confidentiality for a secret receiver node

We verify that a group of users of a given node j can learn nothing about the updates to the content of a post PID located at a different node i beyond the existence of an update unless PID is being shared between the two nodes and one of the users is the admin at node j or becomes a remote friend of PID 's owner, or PID is marked as public. This is formulated as a BD Security property and is proved by unwinding.

See [3] for more details.

6.2.1 Observation setup

type-synonym *obs* = *act * out*

locale *Fixed-UIDs* = **fixes** *UIDs* :: *userID set*

locale *Fixed-PID* = **fixes** *PID* :: *postID*

locale *Fixed-AID* = **fixes** *AID* :: *apiID*

locale *ObservationSetup-RECEIVER* = *Fixed-UIDs* + *Fixed-PID* + *Fixed-AID*

begin

fun $\gamma :: (state, act, out) trans \Rightarrow bool$ **where**
 $\gamma (Trans - a - -) \longleftrightarrow$
 $(\exists uid. userOfA a = Some uid \wedge uid \in UIDs)$
 \vee
 $(\exists ca. a = COMact ca)$
 \vee
 $(\exists uid p. a = Sact (sSys uid p))$

fun $sPurge :: sActt \Rightarrow sActt$ **where**
 $sPurge (sSys uid pwd) = sSys uid emptyPass$

fun $comPurge :: comActt \Rightarrow comActt$ **where**
 $comPurge (comSendServerReq uID p aID reqInfo) = comSendServerReq uID emptyPass aID reqInfo$
 $| comPurge (comConnectClient uID p aID sp) = comConnectClient uID emptyPass aID sp$
 $| comPurge (comReceivePost aID sp pID pst uID vs) =$
 $(let pst' = (if aID = AID \wedge pID = PID then emptyPost else pst)$
 $in comReceivePost aID sp pID pst' uID vs)$
 $| comPurge (comSendPost uID p aID pID) = comSendPost uID emptyPass aID pID$
 $| comPurge (comSendCreateOFriend uID p aID uID') = comSendCreateOFriend uID emptyPass aID uID'$
 $| comPurge (comSendDeleteOFriend uID p aID uID') = comSendDeleteOFriend uID emptyPass aID uID'$
 $| comPurge ca = ca$

fun $g :: (state, act, out)trans \Rightarrow obs$ **where**
 $g (Trans - (Sact sa) ou -) = (Sact (sPurge sa), ou)$
 $| g (Trans - (COMact ca) ou -) = (COMact (comPurge ca), ou)$
 $| g (Trans - a ou -) = (a, ou)$

lemma $comPurge-simps:$

$comPurge ca = comSendServerReq uID p aID reqInfo \longleftrightarrow (\exists p'. ca = comSendServerReq uID p' aID reqInfo \wedge p = emptyPass)$
 $comPurge ca = comReceiveClientReq aID reqInfo \longleftrightarrow ca = comReceiveClientReq aID reqInfo$
 $comPurge ca = comConnectClient uID p aID sp \longleftrightarrow (\exists p'. ca = comConnectClient uID p' aID sp \wedge p = emptyPass)$
 $comPurge ca = comConnectServer aID sp \longleftrightarrow ca = comConnectServer aID sp$

$comPurge\ ca = comReceivePost\ aID\ sp\ pID\ pst'\ uID\ v \longleftrightarrow (\exists\ pst.\ ca = comReceivePost\ aID\ sp\ pID\ pst\ uID\ v \wedge pst' = (if\ pID = PID \wedge aID = AID\ then\ emptyPost\ else\ pst))$

$comPurge\ ca = comSendPost\ uID\ p\ aID\ pID \longleftrightarrow (\exists\ p'.\ ca = comSendPost\ uID\ p'\ aID\ pID \wedge p = emptyPass)$

$comPurge\ ca = comSendCreateOFriend\ uID\ p\ aID\ uID' \longleftrightarrow (\exists\ p'.\ ca = comSendCreateOFriend\ uID\ p'\ aID\ uID' \wedge p = emptyPass)$

$comPurge\ ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uID'$

$comPurge\ ca = comSendDeleteOFriend\ uID\ p\ aID\ uID' \longleftrightarrow (\exists\ p'.\ ca = comSendDeleteOFriend\ uID\ p'\ aID\ uID' \wedge p = emptyPass)$

$comPurge\ ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'$

$\langle proof \rangle$

lemma g -simps:

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendServerReq\ uID\ p\ aID\ reqInfo),\ ou')$
 $\longleftrightarrow (\exists\ p'.\ a = COMact\ (comSendServerReq\ uID\ p'\ aID\ reqInfo) \wedge p = emptyPass \wedge ou = ou')$

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveClientReq\ aID\ reqInfo),\ ou')$
 $\longleftrightarrow a = COMact\ (comReceiveClientReq\ aID\ reqInfo) \wedge ou = ou'$

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectClient\ uID\ p\ aID\ sp),\ ou')$
 $\longleftrightarrow (\exists\ p'.\ a = COMact\ (comConnectClient\ uID\ p'\ aID\ sp) \wedge p = emptyPass \wedge ou = ou')$

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectServer\ aID\ sp),\ ou')$
 $\longleftrightarrow a = COMact\ (comConnectServer\ aID\ sp) \wedge ou = ou'$

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceivePost\ aID\ sp\ pID\ pst'\ uID\ v),\ ou')$
 $\longleftrightarrow (\exists\ pst.\ a = COMact\ (comReceivePost\ aID\ sp\ pID\ pst\ uID\ v) \wedge pst' = (if\ pID = PID \wedge aID = AID\ then\ emptyPost\ else\ pst) \wedge ou = ou')$

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendPost\ uID\ p\ aID\ nID),\ O-sendPost\ (aid,\ sp,\ pid,\ pst,\ own,\ v))$
 $\longleftrightarrow (\exists\ p'.\ a = COMact\ (comSendPost\ uID\ p'\ aID\ nID) \wedge p = emptyPass \wedge ou = O-sendPost\ (aid,\ sp,\ pid,\ pst,\ own,\ v))$

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendCreateOFriend\ uID\ p\ aID\ uID'),\ ou')$
 $\longleftrightarrow (\exists\ p'.\ a = (COMact\ (comSendCreateOFriend\ uID\ p'\ aID\ uID')) \wedge p = emptyPass \wedge ou = ou')$

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID'),\ ou')$
 $\longleftrightarrow a = COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID') \wedge ou = ou'$

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendDeleteOFriend\ uID\ p\ aID\ uID'),\ ou')$
 $\longleftrightarrow (\exists\ p'.\ a = COMact\ (comSendDeleteOFriend\ uID\ p'\ aID\ uID') \wedge p = emptyPass \wedge ou = ou')$

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'),\ ou')$
 $\longleftrightarrow a = COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID') \wedge ou = ou'$

$\langle proof \rangle$

end

end
theory *Post-Unwinding-Helper-RECEIVER*
imports *Post-Observation-Setup-RECEIVER*
begin

6.2.2 Unwinding helper definitions and lemmas

locale *Receiver-State-Equivalence-Up-To-PID = Fixed-PID + Fixed-AID*
begin

definition *eeqButPID* **where**
eeqButPID psts psts1 \equiv
 \forall *aid pid*. *if* $aid = AID \wedge pid = PID$ *then* *True*
 else $psts\ aid\ pid = psts1\ aid\ pid$

lemmas *eeqButPID-intro = eeqButPID-def*[*THEN meta-eq-to-obj-eq, THEN iffD2*]

lemma *eeqButPID-eeq*[*simp,intro!*]: *eeqButPID psts psts*
 $\langle proof \rangle$

lemma *eeqButPID-sym*:
assumes *eeqButPID psts psts1* **shows** *eeqButPID psts1 psts*
 $\langle proof \rangle$

lemma *eeqButPID-trans*:
assumes *eeqButPID psts psts1* **and** *eeqButPID psts1 psts2* **shows** *eeqButPID psts*
psts2
 $\langle proof \rangle$

lemma *eeqButPID-cong*:
assumes *eeqButPID psts psts1*
and $aid = AID \implies pid = PID \implies eqButT\ uu\ uu1$
and $aid \neq AID \vee pid \neq PID \implies uu = uu1$
shows *eeqButPID (fun-upd2 psts aid pid uu) (fun-upd2 psts1 aid pid uu1)*
 $\langle proof \rangle$

lemma *eeqButPID-not-PID*:
 $\llbracket eeqButPID\ psts\ psts1; aid \neq AID \vee pid \neq PID \rrbracket \implies psts\ aid\ pid = psts1\ aid\ pid$
 $\langle proof \rangle$

lemma *eeqButPID-toEq*:
assumes *eeqButPID psts psts1*
shows $fun-upd2\ psts\ AID\ PID\ pst =$
 $fun-upd2\ psts1\ AID\ PID\ pst$
 $\langle proof \rangle$

lemma *eqButPID-update-post*:
assumes *eqButPID psts psts1*
shows *eqButPID (fun-upd2 psts aid pid pst) (fun-upd2 psts1 aid pid pst)*
 \langle *proof* \rangle

fun *eqButF* :: $(\text{apiID} \times \text{bool}) \text{ list} \Rightarrow (\text{apiID} \times \text{bool}) \text{ list} \Rightarrow \text{bool}$ **where**
eqButF aID-bl aID-bl1 = $(\text{map fst aID-bl} = \text{map fst aID-bl1})$

lemma *eqButF-eq[simp,intro!]*: *eqButF aID-bl aID-bl*
 \langle *proof* \rangle

lemma *eqButF-sym*:
assumes *eqButF aID-bl aID-bl1*
shows *eqButF aID-bl1 aID-bl*
 \langle *proof* \rangle

lemma *eqButF-trans*:
assumes *eqButF aID-bl aID-bl1* **and** *eqButF aID-bl1 aID-bl2*
shows *eqButF aID-bl aID-bl2*
 \langle *proof* \rangle

lemma *eqButF-insert2*:
eqButF aID-bl aID-bl1 \implies
eqButF (insert2 aID b aID-bl) (insert2 aID b aID-bl1)
 \langle *proof* \rangle

definition *eqButPID* :: $\text{state} \Rightarrow \text{state} \Rightarrow \text{bool}$ **where**
eqButPID s s1 \equiv
admin s = admin s1 \wedge

pendingUReqs s = pendingUReqs s1 \wedge *userReq s = userReq s1* \wedge
userIDs s = userIDs s1 \wedge *user s = user s1* \wedge *pass s = pass s1* \wedge

pendingFReqs s = pendingFReqs s1 \wedge *friendReq s = friendReq s1* \wedge *friendIDs s*
 $=$ *friendIDs s1* \wedge
sentOuterFriendIDs s = sentOuterFriendIDs s1 \wedge *recvOuterFriendIDs s = recvOuter-*
FriendIDs s1 \wedge

postIDs s = postIDs s1 \wedge *admin s = admin s1* \wedge
post s = post s1 \wedge
owner s = owner s1 \wedge
vis s = vis s1 \wedge

pendingSApiReqs s = pendingSApiReqs s1 \wedge *sApiReq s = sApiReq s1* \wedge
serverApiIDs s = serverApiIDs s1 \wedge *serverPass s = serverPass s1* \wedge

$outerPostIDs\ s = outerPostIDs\ s1 \wedge$
 $eqButPID\ (outerPost\ s)\ (outerPost\ s1) \wedge$
 $outerOwner\ s = outerOwner\ s1 \wedge$
 $outerVis\ s = outerVis\ s1 \wedge$

$pendingCApiReqs\ s = pendingCApiReqs\ s1 \wedge cApiReq\ s = cApiReq\ s1 \wedge$
 $clientApiIDs\ s = clientApiIDs\ s1 \wedge clientPass\ s = clientPass\ s1 \wedge$
 $sharedWith\ s = sharedWith\ s1$

lemmas $eqButPID-intro = eqButPID-def[THEN\ meta-eq-to-obj-eq,\ THEN\ iffD2]$

lemma $eqButPID-refl[simp,intro!]$: $eqButPID\ s\ s$
 $\langle proof \rangle$

lemma $eqButPID-sym$:
assumes $eqButPID\ s\ s1$ **shows** $eqButPID\ s1\ s$
 $\langle proof \rangle$

lemma $eqButPID-trans$:
assumes $eqButPID\ s\ s1$ **and** $eqButPID\ s1\ s2$ **shows** $eqButPID\ s\ s2$
 $\langle proof \rangle$

lemma $eqButPID-stateSelectors$:
 $eqButPID\ s\ s1 \implies$
 $admin\ s = admin\ s1 \wedge$

$pendingUReqs\ s = pendingUReqs\ s1 \wedge userReq\ s = userReq\ s1 \wedge$
 $userIDs\ s = userIDs\ s1 \wedge user\ s = user\ s1 \wedge pass\ s = pass\ s1 \wedge$

$pendingFReqs\ s = pendingFReqs\ s1 \wedge friendReq\ s = friendReq\ s1 \wedge friendIDs\ s$
 $= friendIDs\ s1 \wedge$
 $sentOuterFriendIDs\ s = sentOuterFriendIDs\ s1 \wedge recvOuterFriendIDs\ s = recvOuter-$
 $FriendIDs\ s1 \wedge$

$postIDs\ s = postIDs\ s1 \wedge admin\ s = admin\ s1 \wedge$
 $post\ s = post\ s1 \wedge$
 $owner\ s = owner\ s1 \wedge$
 $vis\ s = vis\ s1 \wedge$

$pendingSApiReqs\ s = pendingSApiReqs\ s1 \wedge sApiReq\ s = sApiReq\ s1 \wedge$
 $serverApiIDs\ s = serverApiIDs\ s1 \wedge serverPass\ s = serverPass\ s1 \wedge$
 $outerPostIDs\ s = outerPostIDs\ s1 \wedge$
 $eqButPID\ (outerPost\ s)\ (outerPost\ s1) \wedge$
 $outerOwner\ s = outerOwner\ s1 \wedge$
 $outerVis\ s = outerVis\ s1 \wedge$

$pendingCApiReqs\ s = pendingCApiReqs\ s1 \wedge cApiReq\ s = cApiReq\ s1 \wedge$
 $clientApiIDs\ s = clientApiIDs\ s1 \wedge clientPass\ s = clientPass\ s1 \wedge$

$sharedWith\ s = sharedWith\ s1 \wedge$

$IDsOK\ s = IDsOK\ s1$

$\langle proof \rangle$

lemma *eqButPID-not-PID*:

$eqButPID\ s\ s1 \implies aid \neq AID \vee pid \neq PID \implies outerPost\ s\ aid\ pid = outerPost\ s1\ aid\ pid$

$\langle proof \rangle$

lemma *eqButPID-actions*:

assumes $eqButPID\ s\ s1$

shows $listInnerPosts\ s\ uid\ p = listInnerPosts\ s1\ uid\ p$

and $listOuterPosts\ s\ uid\ p = listOuterPosts\ s1\ uid\ p$

$\langle proof \rangle$

lemma *eqButPID-update*:

assumes $eqButPID\ s\ s1$

shows $fun-upd2\ (outerPost\ s)\ AID\ PID\ pst = fun-upd2\ (outerPost\ s1)\ AID\ PID\ pst$

$\langle proof \rangle$

lemma *eqButPID-update-post*:

assumes $eqButPID\ s\ s1$

shows $eqButPID\ (fun-upd2\ (outerPost\ s)\ aid\ pid\ pst)\ (fun-upd2\ (outerPost\ s1)\ aid\ pid\ pst)$

$\langle proof \rangle$

lemma *eqButPID-cong[simp, intro]*:

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|admin := uu1|))\ (s1\ (\!|admin := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|pendingUReqs := uu1|))\ (s1\ (\!|pendingUReqs := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|userReq := uu1|))\ (s1\ (\!|userReq := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|userIDs := uu1|))\ (s1\ (\!|userIDs := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|user := uu1|))\ (s1\ (\!|user := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|pass := uu1|))\ (s1\ (\!|pass := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|postIDs := uu1|))\ (s1\ (\!|postIDs := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|post := uu1|))\ (s1\ (\!|post := uu2|))$

$\bigwedge uu1\ uu2. eqButPID\ s\ s1 \implies uu1 = uu2 \implies eqButPID\ (s\ (\!|owner := uu1|))\ (s1\ (\!|owner := uu2|))$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|vis := uu1\!) (s1 (\!|vis := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|pendingFReqs := uu1\!) (s1 (\!|pendingFReqs := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|friendReq := uu1\!) (s1 (\!|friendReq := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|friendIDs := uu1\!) (s1 (\!|friendIDs := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|sentOuterFriendIDs := uu1\!) (s1 (\!|sentOuterFriendIDs := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|recvOuterFriendIDs := uu1\!) (s1 (\!|recvOuterFriendIDs := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|pendingSApiReqs := uu1\!) (s1 (\!|pendingSApiReqs := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|sApiReq := uu1\!) (s1 (\!|sApiReq := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|serverApiIDs := uu1\!) (s1 (\!|serverApiIDs := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|serverPass := uu1\!) (s1 (\!|serverPass := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|outerPostIDs := uu1\!) (s1 (\!|outerPostIDs := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies eqButPID uu1 uu2 \implies eqButPID (s (\!|outerPost := uu1\!) (s1 (\!|outerPost := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|outerVis := uu1\!) (s1 (\!|outerVis := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|outerOwner := uu1\!) (s1 (\!|outerOwner := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|pendingCApiReqs := uu1\!) (s1 (\!|pendingCApiReqs := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|cApiReq := uu1\!) (s1 (\!|cApiReq := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|clientApiIDs := uu1\!) (s1 (\!|clientApiIDs := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|clientPass := uu1\!) (s1 (\!|clientPass := uu2\!)))$$

$$\bigwedge uu1 uu2. eqButPID s s1 \implies uu1 = uu2 \implies eqButPID (s (\!|sharedWith := uu1\!) (s1 (\!|sharedWith := uu2\!)))$$

<proof>

lemma *comReceivePost-step-eqButPID:*
assumes *a: a = COMact (comReceivePost AID sp PID pst uid vs)*

and $a1: a1 = COMact (comReceivePost AID sp PID pst1 uid vs)$
and $step\ s\ a = (ou, s')$ **and** $step\ s1\ a1 = (ou1, s1')$
and $eqButPID\ s\ s1$
shows $eqButPID\ s'\ s1'$
 $\langle proof \rangle$

lemma $eqButPID\text{-}step$:
assumes $ss1: eqButPID\ s\ s1$
and $step: step\ s\ a = (ou, s')$
and $step1: step\ s1\ a = (ou1, s1')$
shows $eqButPID\ s'\ s1'$
 $\langle proof \rangle$

end

end

theory $Post\text{-}Value\text{-}Setup\text{-}RECEIVER$
imports
 $../Safety\text{-}Properties$
 $Post\text{-}Observation\text{-}Setup\text{-}RECEIVER$
 $Post\text{-}Unwinding\text{-}Helper\text{-}RECEIVER$
begin

6.2.3 Value setup

locale $Post\text{-}RECEIVER = ObservationSetup\text{-}RECEIVER$
begin

datatype $value = PValR\ post$ — post content received from the issuer node

fun $\varphi :: (state, act, out)\ trans \Rightarrow bool$ **where**
 $\varphi (Trans\ -\ (COMact\ (comReceivePost\ aid\ sp\ pid\ pst\ uid\ vs))\ ou\ -) =$
 $(aid = AID \wedge pid = PID \wedge ou = outOK)$
 $|$
 $\varphi (Trans\ s\ -\ s') = False$

lemma $\varphi\text{-}def2$:
 $\varphi (Trans\ s\ a\ ou\ s') \longleftrightarrow$
 $(\exists\ uid\ p\ pst\ vs.\ a = COMact\ (comReceivePost\ AID\ p\ PID\ pst\ uid\ vs) \wedge ou =$
 $outOK)$
 $\langle proof \rangle$

lemma $comReceivePost\text{-}out$:
assumes $1: step\ s\ a = (ou, s')$ **and** $a: a = COMact (comReceivePost AID p PID pst uid vs)$ **and** $2: ou = outOK$
shows $p = serverPass\ s\ AID$
 $\langle proof \rangle$

lemma φ -def3:
assumes $step\ s\ a = (ou, s')$
shows
 $\varphi\ (Trans\ s\ a\ ou\ s') \longleftrightarrow$
 $(\exists\ uid\ pst\ vs.\ a = COMact\ (comReceivePost\ AID\ (serverPass\ s\ AID)\ PID\ pst\ uid$
 $vs) \wedge ou = outOK)$
 $\langle proof \rangle$

lemma φ -cases:
assumes $\varphi\ (Trans\ s\ a\ ou\ s')$
and $step\ s\ a = (ou, s')$
and $reach\ s$
obtains
 $(Recv)\ uid\ sp\ aID\ pID\ pst\ vs$ **where** $a = COMact\ (comReceivePost\ aID\ sp\ pID$
 $pst\ uid\ vs)\ ou = outOK$
 $sp = serverPass\ s\ AID$
 $aID = AID\ pID = PID$
 $\langle proof \rangle$

fun $f :: (state, act, out)\ trans \Rightarrow value$ **where**
 $f\ (Trans\ s\ (COMact\ (comReceivePost\ aid\ sp\ pid\ pst\ uid\ vs)) - s') =$
 $(if\ aid = AID \wedge pid = PID\ then\ PValR\ pst\ else\ undefined)$
 $|$
 $f\ (Trans\ s\ - -\ s') = undefined$

sublocale *Receiver-State-Equivalence-Up-To-PID* $\langle proof \rangle$

lemma *eqButPID-step- φ -imp*:
assumes $ss1: eqButPID\ s\ s1$
and $step: step\ s\ a = (ou, s')$ **and** $step1: step\ s1\ a = (ou1, s1')$
and $\varphi: \varphi\ (Trans\ s\ a\ ou\ s')$
shows $\varphi\ (Trans\ s1\ a\ ou1\ s1')$
 $\langle proof \rangle$

lemma *eqButPID-step- φ* :
assumes $s's1': eqButPID\ s\ s1$
and $step: step\ s\ a = (ou, s')$ **and** $step1: step\ s1\ a = (ou1, s1')$
shows $\varphi\ (Trans\ s\ a\ ou\ s') = \varphi\ (Trans\ s1\ a\ ou1\ s1')$
 $\langle proof \rangle$

end

end
theory *Post-RECEIVER*
imports

Bounded-Deducibility-Security.Compositional-Reasoning
Post-Observation-Setup-RECEIVER
Post-Value-Setup-RECEIVER

begin

6.2.4 Declassification bound

We verify that a group of users of some node i , allowed to take normal actions to the system and observe their outputs *and additionally allowed to observe communication*, can learn nothing about the updates to a post received from a remote node j beyond the number of updates unless:

- either a user in the group is the administrator
- or a user in the group becomes a remote friend of the post's owner
- or the group has at least one registered user and the post is being marked as "public"

See [3] for more details.

context *Post-RECEIVER*
begin

fun $T :: (state, act, out) \text{ trans} \Rightarrow \text{bool}$ **where**
 $T (Trans\ s\ a\ ou\ s') \longleftrightarrow$
 $(\exists\ uid \in UIDs.$
 $uid \in\in userIDs\ s' \wedge PID \in\in outerPostIDs\ s'\ AID \wedge$
 $(uid = admin\ s' \vee$
 $(AID, outerOwner\ s'\ AID\ PID) \in\in recvOuterFriendIDs\ s'\ uid \vee$
 $outerVis\ s'\ AID\ PID = PublicV))$

definition $B :: \text{value list} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**
 $B\ vl\ vl1 \equiv length\ vl = length\ vl1$

sublocale *BD-Security-IO* **where**
 $istate = istate$ **and** $step = step$ **and**
 $\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
 $\langle proof \rangle$

6.2.5 Unwinding proof

lemma *reach-PublicV-imples-FriendV[simp]*:
assumes $reach\ s$
and $vis\ s\ pID \neq PublicV$
shows $vis\ s\ pID = FriendV$
 $\langle proof \rangle$

lemma *reachNT-state*:
assumes *reachNT s*
shows
 $\neg (\exists \text{uid} \in \text{UIDs}.$
 $\text{uid} \in \text{userIDs } s \wedge \text{PID} \in \text{outerPostIDs } s \text{ AID} \wedge$
 $(\text{uid} = \text{admin } s \vee$
 $(\text{AID}, \text{outerOwner } s \text{ AID } \text{PID}) \in \text{recvOuterFriendIDs } s \text{ uid} \vee$
 $\text{outerVis } s \text{ AID } \text{PID} = \text{PublicV}))$
 $\langle \text{proof} \rangle$

lemma *eqButPID-step- γ -out*:
assumes *ss1: eqButPID s s1*
and *step: step s a = (ou, s[^])* **and** *step1: step s1 a = (ou1, s1[^])*
and *sT: reachNT s* **and** *T: $\neg T$ (Trans s a ou s[^])*
and *s1: reach s1*
and *γ : γ (Trans s a ou s[^])*
shows *ou = ou1*
 $\langle \text{proof} \rangle$

definition $\Delta 0 :: \text{state} \Rightarrow \text{value list} \Rightarrow \text{state} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**
 $\Delta 0 \text{ s vl s1 vl1} \equiv$
 $\neg \text{AID} \in \text{serverApiIDs } s \wedge$
 $s = s1 \wedge$
 $\text{length vl} = \text{length vl1}$

definition $\Delta 1 :: \text{state} \Rightarrow \text{value list} \Rightarrow \text{state} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**
 $\Delta 1 \text{ s vl s1 vl1} \equiv$
 $\text{AID} \in \text{serverApiIDs } s \wedge$
 $\text{eqButPID } s \text{ s1} \wedge$
 $\text{length vl} = \text{length vl1}$

lemma *istate- $\Delta 0$* :
assumes *B: B vl vl1*
shows $\Delta 0 \text{ istate vl istate vl1}$
 $\langle \text{proof} \rangle$

lemma *unwind-cont- $\Delta 0$* : *unwind-cont $\Delta 0$ { $\Delta 0, \Delta 1$ }*
 $\langle \text{proof} \rangle$

lemma *unwind-cont- $\Delta 1$* : *unwind-cont $\Delta 1$ { $\Delta 1$ }*
 $\langle \text{proof} \rangle$

definition *Gr* **where**
 $Gr =$

```

{
( $\Delta 0$ , { $\Delta 0, \Delta 1$ }),
( $\Delta 1$ , { $\Delta 1$ })
}

```

theorem *Post-secure: secure*
 \langle *proof* \rangle

end

end

theory *Post-COMPOSE2*

imports

Post-ISSUER

Post-RECEIVER

BD-Security-Compositional.Composing-Security

begin

6.3 Confidentiality for the (binary) issuer-receiver composition

type-synonym *ttrans* = (*state, act, out*) *trans*

type-synonym *value1* = *Post-ISSUER.value* **type-synonym** *value2* = *Post-RECEIVER.value*

type-synonym *obs1* = *Post-Observation-Setup-ISSUER.obs*

type-synonym *obs2* = *Post-Observation-Setup-RECEIVER.obs*

datatype *cval* = *PValC post*

type-synonym *cobs* = *obs1* \times *obs2*

locale *Post-COMPOSE2* =

Iss: Post-ISSUER UIDs PID +

Rcv: Post-RECEIVER UIDs2 PID AID1

for *UIDs* :: *userID set* **and** *UIDs2* :: *userID set* **and**

AID1 :: *apiID* **and** *PID* :: *postID*

+ **fixes** *AID2* :: *apiID*

begin

abbreviation $\varphi 1 \equiv$ *Iss. φ* **abbreviation** *f1* \equiv *Iss.f* **abbreviation** $\gamma 1 \equiv$ *Iss. γ*

abbreviation *g1* \equiv *Iss.g*

abbreviation *T1* \equiv *Iss.T* **abbreviation** *B1* \equiv *Iss.B*

abbreviation $\varphi 2 \equiv$ *Rcv. φ* **abbreviation** *f2* \equiv *Rcv.f* **abbreviation** $\gamma 2 \equiv$ *Rcv. γ*

abbreviation *g2* \equiv *Rcv.g*

abbreviation *T2* \equiv *Rcv.T* **abbreviation** *B2* \equiv *Rcv.B*

fun *isCom1* :: *ttrans* \Rightarrow *bool* **where**

```

isCom1 (Trans s (COMact ca1) ou1 s') = (ou1 ≠ outErr)
|isCom1 - = False

```

```

fun isCom2 :: ttrans ⇒ bool where
isCom2 (Trans s (COMact ca2) ou2 s') = (ou2 ≠ outErr)
|isCom2 - = False

```

```

fun isComV1 :: value1 ⇒ bool where
isComV1 (Iss.PValS aid1 txt1) = True
|isComV1 - = False

```

```

fun isComV2 :: value2 ⇒ bool where
isComV2 (Rcv.PValR txt2) = True

```

```

fun syncV :: value1 ⇒ value2 ⇒ bool where
syncV (Iss.PValS aid1 txt1) (Rcv.PValR txt2) = (txt1 = txt2)
|syncV - - = False

```

```

fun cmpV :: value1 ⇒ value2 ⇒ cval where
cmpV (Iss.PValS aid1 txt1) (Rcv.PValR txt2) = PValC txt1
|cmpV - - = undefined

```

```

fun isComO1 :: obs1 ⇒ bool where
isComO1 (COMact ca1, ou1) = (ou1 ≠ outErr)
|isComO1 - = False

```

```

fun isComO2 :: obs2 ⇒ bool where
isComO2 (COMact ca2, ou2) = (ou2 ≠ outErr)
|isComO2 - = False

```

```

fun comSyncOA :: out ⇒ comActt ⇒ bool where
comSyncOA (O-sendServerReq (aid2, reqInfo1)) (comReceiveClientReq aid1 re-
qInfo2) =
  (aid1 = AID1 ∧ aid2 = AID2 ∧ reqInfo1 = reqInfo2)
|comSyncOA (O-connectClient (aid2, sp1)) (comConnectServer aid1 sp2) =
  (aid1 = AID1 ∧ aid2 = AID2 ∧ sp1 = sp2)
|comSyncOA (O-sendPost (aid2, sp1, pid1, pst1, uid1, vis1)) (comReceivePost
aid1 sp2 pid2 pst2 uid2 vis2) =
  (aid1 = AID1 ∧ aid2 = AID2 ∧ (pid1, pst1, uid1, vis1) = (pid2, pst2, uid2,
vis2))
|comSyncOA (O-sendCreateOFriend (aid2, sp1, uid1, uid1')) (comReceiveCreateOFriend
aid1 sp2 uid2 uid2') =
  (aid1 = AID1 ∧ aid2 = AID2 ∧ (uid1, uid1') = (uid2, uid2'))
|comSyncOA (O-sendDeleteOFriend (aid2, sp1, uid1, uid1')) (comReceiveDeleteOFriend
aid1 sp2 uid2 uid2') =
  (aid1 = AID1 ∧ aid2 = AID2 ∧ (uid1, uid1') = (uid2, uid2'))
|comSyncOA - - = False

```

fun *syncO* :: *obs1* \Rightarrow *obs2* \Rightarrow *bool* **where**
syncO (*COMact ca1, ou1*) (*COMact ca2, ou2*) =
 (*ou1* \neq *outErr* \wedge *ou2* \neq *outErr* \wedge
 (*comSyncOA ou1 ca2* \vee *comSyncOA ou2 ca1*)
)
 | *syncO* - - = *False*

fun *sync* :: *ttrans* \Rightarrow *ttrans* \Rightarrow *bool* **where**
sync (*Trans s1 a1 ou1 s1'*) (*Trans s2 a2 ou2 s2'*) = *syncO* (*a1, ou1*) (*a2, ou2*)

definition *cmpO* :: *obs1* \Rightarrow *obs2* \Rightarrow *cobs* **where**
cmpO *o1 o2* \equiv (*o1, o2*)

lemma *isCom1-isComV1*:
assumes *validTrans trn1* **and** *reach (srcOf trn1)* **and** φ_1 *trn1*
shows *isCom1 trn1* \longleftrightarrow *isComV1 (f1 trn1)*
<proof>

lemma *isCom1-isComO1*:
assumes *validTrans trn1* **and** *reach (srcOf trn1)* **and** γ_1 *trn1*
shows *isCom1 trn1* \longleftrightarrow *isComO1 (g1 trn1)*
<proof>

lemma *isCom2-isComV2*:
assumes *validTrans trn2* **and** *reach (srcOf trn2)* **and** φ_2 *trn2*
shows *isCom2 trn2* \longleftrightarrow *isComV2 (f2 trn2)*
<proof>

lemma *isCom2-isComO2*:
assumes *validTrans trn2* **and** *reach (srcOf trn2)* **and** γ_2 *trn2*
shows *isCom2 trn2* \longleftrightarrow *isComO2 (g2 trn2)*
<proof>

lemma *sync-syncV*:
assumes *validTrans trn1* **and** *reach (srcOf trn1)*
and *validTrans trn2* **and** *reach (srcOf trn2)*
and *isCom1 trn1* **and** *isCom2 trn2* **and** φ_1 *trn1* **and** φ_2 *trn2*
and *sync trn1 trn2*
shows *syncV (f1 trn1) (f2 trn2)*
<proof>

lemma *sync-syncO*:
assumes *validTrans trn1* **and** *reach (srcOf trn1)*
and *validTrans trn2* **and** *reach (srcOf trn2)*

and *isCom1 trn1 and isCom2 trn2 and γ_1 trn1 and γ_2 trn2*
and *sync trn1 trn2*
shows *syncO (g1 trn1) (g2 trn2)*
<proof>

lemma *sync- φ_1 - φ_2 :*
assumes *v1: validTrans trn1 and r1: reach (srcOf trn1)*
and *v2: validTrans trn2 and s2: reach (srcOf trn2)*
and *c1: isCom1 trn1 and c2: isCom2 trn2*
and *sn: sync trn1 trn2*
shows *φ_1 trn1 \longleftrightarrow φ_2 trn2 (is ?A \longleftrightarrow ?B)*
<proof>

lemma *textPost-textPost-cong[intro]:*
assumes *textPost pst1 = textPost pst2*
and *setTextPost pst1 emptyText = setTextPost pst2 emptyText*
shows *pst1 = pst2*
<proof>

lemma *sync- φ - γ :*
assumes *validTrans trn1 and reach (srcOf trn1)*
and *validTrans trn2 and reach (srcOf trn2)*
and *isCom1 trn1 and isCom2 trn2*
and *γ_1 trn1 and γ_2 trn2*
and *so: syncO (g1 trn1) (g2 trn2)*
and *φ_1 trn1 \implies φ_2 trn2 \implies syncV (f1 trn1) (f2 trn2)*
shows *sync trn1 trn2*
<proof>

lemma *isCom1- γ_1 :*
assumes *validTrans trn1 and reach (srcOf trn1) and isCom1 trn1*
shows *γ_1 trn1*
<proof>

lemma *isCom2- γ_2 :*
assumes *validTrans trn2 and reach (srcOf trn2) and isCom2 trn2*
shows *γ_2 trn2*
<proof>

lemma *isCom2-V2:*
assumes *validTrans trn2 and reach (srcOf trn2) and φ_2 trn2*
shows *isCom2 trn2*
<proof>

end

sublocale *Post-COMPOSE2 < BD-Security-TS-Comp where*
istate1 = istate and validTrans1 = validTrans and srcOf1 = srcOf and tgtOf1

```

= tgtOf
  and  $\varphi1 = \varphi1$  and  $f1 = f1$  and  $\gamma1 = \gamma1$  and  $g1 = g1$  and  $T1 = T1$  and
   $B1 = B1$ 
  and
     $istate2 = istate$  and  $validTrans2 = validTrans$  and  $srcOf2 = srcOf$  and  $tgtOf2$ 
= tgtOf
  and  $\varphi2 = \varphi2$  and  $f2 = f2$  and  $\gamma2 = \gamma2$  and  $g2 = g2$  and  $T2 = T2$  and
   $B2 = B2$ 
  and  $isCom1 = isCom1$  and  $isCom2 = isCom2$  and  $sync = sync$ 
  and  $isComV1 = isComV1$  and  $isComV2 = isComV2$  and  $syncV = syncV$ 
  and  $isComO1 = isComO1$  and  $isComO2 = isComO2$  and  $syncO = syncO$ 

```

<proof>

```

context Post-COMPOSE2
begin

```

```

theorem secure: secure
  <proof>

```

end

end

```

theory Post-Network

```

```

imports

```

```

  ../API-Network

```

```

  Post-ISSUER

```

```

  Post-RECEIVER

```

```

  BD-Security-Compositional.Composing-Security-Network

```

```

begin

```

6.4 Confidentiality for the N-ary composition

```

type-synonym ttrans = (state, act, out) trans

```

```

type-synonym obs = Post-Observation-Setup-ISSUER.obs

```

```

type-synonym value = Post-ISSUER.value + Post-RECEIVER.value

```

```

lemma value-cases:

```

```

fixes  $v :: value$ 

```

```

obtains (PVal)  $pst$  where  $v = Inl (Post-ISSUER.PVal pst)$ 

```

```

  | (PValS)  $aid pst$  where  $v = Inl (Post-ISSUER.PValS aid pst)$ 

```

```

  | (PValR)  $pst$  where  $v = Inr (Post-RECEIVER.PValR pst)$ 

```

<proof>

```

locale Post-Network = Network

```

```

+ fixes  $UIDs :: apiID \Rightarrow userID set$ 

```

```

  and  $AID :: apiID$  and  $PID :: postID$ 

```

```

  assumes AID-in-AIDs: AID  $\in$  AIDs

```

begin

sublocale *Iss*: *Post-ISSUER* *UIDs* *AID* *PID* \langle *proof* \rangle

abbreviation $\varphi :: \text{apiID} \Rightarrow (\text{state}, \text{act}, \text{out}) \text{trans} \Rightarrow \text{bool}$

where $\varphi \text{ aid trn} \equiv (\text{if } \text{aid} = \text{AID} \text{ then } \text{Iss}.\varphi \text{ trn} \text{ else } \text{Post-RECEIVER}.\varphi \text{ PID AID trn})$

abbreviation $f :: \text{apiID} \Rightarrow (\text{state}, \text{act}, \text{out}) \text{trans} \Rightarrow \text{value}$

where $f \text{ aid trn} \equiv (\text{if } \text{aid} = \text{AID} \text{ then } \text{Inl} (\text{Iss}.f \text{ trn}) \text{ else } \text{Inr} (\text{Post-RECEIVER}.f \text{ PID AID trn}))$

abbreviation $\gamma :: \text{apiID} \Rightarrow (\text{state}, \text{act}, \text{out}) \text{trans} \Rightarrow \text{bool}$

where $\gamma \text{ aid trn} \equiv (\text{if } \text{aid} = \text{AID} \text{ then } \text{Iss}.\gamma \text{ trn} \text{ else } \text{ObservationSetup-RECEIVER}.\gamma (\text{UIDs } \text{aid}) \text{ trn})$

abbreviation $g :: \text{apiID} \Rightarrow (\text{state}, \text{act}, \text{out}) \text{trans} \Rightarrow \text{obs}$

where $g \text{ aid trn} \equiv (\text{if } \text{aid} = \text{AID} \text{ then } \text{Iss}.g \text{ trn} \text{ else } \text{ObservationSetup-RECEIVER}.g \text{ PID AID trn})$

abbreviation $T :: \text{apiID} \Rightarrow (\text{state}, \text{act}, \text{out}) \text{trans} \Rightarrow \text{bool}$

where $T \text{ aid trn} \equiv (\text{if } \text{aid} = \text{AID} \text{ then } \text{Iss}.T \text{ trn} \text{ else } \text{Post-RECEIVER}.T (\text{UIDs } \text{aid}) \text{ PID AID trn})$

abbreviation $B :: \text{apiID} \Rightarrow \text{value list} \Rightarrow \text{value list} \Rightarrow \text{bool}$

where $B \text{ aid vl vl1} \equiv$

$(\text{if } \text{aid} = \text{AID} \text{ then } \text{list-all } \text{isl } \text{vl} \wedge \text{list-all } \text{isl } \text{vl1} \wedge \text{Iss}.B (\text{map } \text{projl } \text{vl}) (\text{map } \text{projl } \text{vl1})$

$\text{else } \text{list-all } (\text{Not } \circ \text{isl}) \text{vl} \wedge \text{list-all } (\text{Not } \circ \text{isl}) \text{vl1} \wedge \text{Post-RECEIVER}.B (\text{map } \text{projr } \text{vl}) (\text{map } \text{projr } \text{vl1}))$

fun *comOfV* :: *apiID* \Rightarrow *value* \Rightarrow *com* **where**

$\text{comOfV } \text{aid} (\text{Inl} (\text{Post-ISSUER}.\text{PValS } \text{aid}' \text{ pst})) = (\text{if } \text{aid}' \neq \text{aid} \text{ then } \text{Send} \text{ else } \text{Internal})$

$| \text{comOfV } \text{aid} (\text{Inl} (\text{Post-ISSUER}.\text{PVal } \text{pst})) = \text{Internal}$

$| \text{comOfV } \text{aid} (\text{Inr } \text{v}) = \text{Recv}$

fun *tgtNodeOfV* :: *apiID* \Rightarrow *value* \Rightarrow *apiID* **where**

$\text{tgtNodeOfV } \text{aid} (\text{Inl} (\text{Post-ISSUER}.\text{PValS } \text{aid}' \text{ pst})) = \text{aid}'$

$| \text{tgtNodeOfV } \text{aid} (\text{Inl} (\text{Post-ISSUER}.\text{PVal } \text{pst})) = \text{undefined}$

$| \text{tgtNodeOfV } \text{aid} (\text{Inr } \text{v}) = \text{AID}$

definition *syncV* :: *apiID* \Rightarrow *value* \Rightarrow *apiID* \Rightarrow *value* \Rightarrow *bool* **where**

$\text{syncV } \text{aid1 } \text{v1 } \text{aid2 } \text{v2} =$

$(\exists \text{pst}. \text{aid1} = \text{AID} \wedge \text{v1} = \text{Inl} (\text{Post-ISSUER}.\text{PValS } \text{aid2 } \text{pst}) \wedge \text{v2} = \text{Inr} (\text{Post-RECEIVER}.\text{PValR } \text{pst}))$

lemma *syncVI*: $\text{syncV } \text{AID} (\text{Inl} (\text{Post-ISSUER}.\text{PValS } \text{aid}' \text{ pst})) \text{aid}' (\text{Inr} (\text{Post-RECEIVER}.\text{PValR } \text{pst}))$

<proof>

lemma *syncVE*:

assumes *syncV aid1 v1 aid2 v2*

obtains *pst* **where** *aid1 = AID v1 = Inl (Post-ISSUER.PValS aid2 pst) v2 = Inr (Post-RECEIVER.PValR pst)*

<proof>

fun *getTgtV* **where**

getTgtV (Inl (Post-ISSUER.PValS aid pst)) = Inr (Post-RECEIVER.PValR pst)
| getTgtV v = v

lemma *comOfV-AID*:

comOfV AID v = Send \longleftrightarrow isl v \wedge Iss.isPValS (projl v) \wedge Iss.PValS-tgtAPI (projl v) \neq AID

comOfV AID v = Recv \longleftrightarrow Not (isl v)

<proof>

lemmas *φ -defs = Post-RECEIVER. φ -def2 Iss. φ -def2*

sublocale *Net: BD-Security-TS-Network-getTgtV*

where *istate = λ -. istate* **and** *validTrans = validTrans* **and** *srcOf = λ -. srcOf*
and *tgtOf = λ -. tgtOf*

and *nodes = AIDs* **and** *comOf = comOf* **and** *tgtNodeOf = tgtNodeOf*

and *sync = sync* **and** *$\varphi = \varphi$* **and** *f = f* **and** *$\gamma = \gamma$* **and** *g = g* **and** *T = T* **and**
B = B

and *comOfV = comOfV* **and** *tgtNodeOfV = tgtNodeOfV* **and** *syncV = syncV*

and *comOfO = comOfO* **and** *tgtNodeOfO = tgtNodeOfO* **and** *syncO = syncO*

and *source = AID* **and** *getTgtV = getTgtV*

<proof>

lemma *list-all-Not-isl-projectSrcV*: *list-all (Not o isl) (Net.projectSrcV aid vlSrc)*

<proof>

context

fixes *AID' :: apiID*

assumes *AID': AID' \in AIDs - {AID}*

begin

interpretation *Sink: Post-RECEIVER UIDs AID' PID AID* *<proof>*

lemma *Source-B-Sink-B-aux*:

assumes *list-all isl vl*

and *list-all isl vl1*

and *map Iss.PValS-tgtAPI (filter Iss.isPValS (map projl vl)) =*

map Iss.PValS-tgtAPI (filter Iss.isPValS (map projl vl1))

shows *length (map projr (Net.projectSrcV AID' vl)) = length (map projr (Net.projectSrcV AID' vl1))*

<proof>

```

lemma Source-B-Sink-B:
assumes B AID vl vl1
shows Sink.B (map projr (Net.projectSrcV AID' vl)) (map projr (Net.projectSrcV AID' vl1))
  <proof>

end

lemma map-projl-Inl: map (projl o Inl) vl = vl
  <proof>

lemma these-map-Inl-projl: list-all isl vl  $\implies$  these (map (Some o Inl o projl) vl) = vl
  <proof>

lemma map-projr-Inr: map (projr o Inr) vl = vl
  <proof>

lemma these-map-Inr-projr: list-all (Not o isl) vl  $\implies$  these (map (Some o Inr o projr) vl) = vl
  <proof>

sublocale BD-Security-TS-Network-Preserve-Source-Security-getTgtV
where istate =  $\lambda$ -. istate and validTrans = validTrans and srcOf =  $\lambda$ -. srcOf
and tgtOf =  $\lambda$ -. tgtOf
  and nodes = AIDs and comOf = comOf and tgtNodeOf = tgtNodeOf
  and sync = sync and  $\varphi = \varphi$  and f = f and  $\gamma = \gamma$  and g = g and T = T and
  B = B
  and comOfV = comOfV and tgtNodeOfV = tgtNodeOfV and syncV = syncV
  and comOfO = comOfO and tgtNodeOfO = tgtNodeOfO and syncO = syncO
  and source = AID and getTgtV = getTgtV
  <proof>

theorem secure: secure
  <proof>

end

end

theory DYNAMIC-Post-Value-Setup-ISSUER
imports
  ../Safety-Properties
  Post-Observation-Setup-ISSUER
  Post-Unwinding-Helper-ISSUER
begin

```

6.5 Variation with dynamic declassification trigger

This section formalizes the “dynamic” variation of one post confidentiality properties, as described in [3, Appendix C].

locale *Post* = *ObservationSetup-ISSUER*
begin

6.5.1 Issuer value setup

datatype *value* =
isPVal: *PVal post* — updating the post content locally
| *isPValS*: *PValS (tgtAPI: apiID) post* — sending the post to another node
| *isOVal*: *OVal bool* — change in the dynamic declassification trigger condition

The dynamic declassification trigger condition holds, i.e. the access window to the confidential information is open, when the post is public or one of the observers is the administrator, the post’s owner, or a friend of the post’s owner.

definition *open where*

open s \equiv
 $\exists uid \in UIDs.$
 $uid \in \in userIDs\ s \wedge PID \in \in postIDs\ s \wedge$
 $(uid = admin\ s \vee uid = owner\ s\ PID \vee uid \in \in friendIDs\ s\ (owner\ s\ PID) \vee$
 $vis\ s\ PID = PublicV)$

sublocale *Issuer-State-Equivalence-Up-To-PID* $\langle proof \rangle$

lemma *eqButPID-open:*

assumes *eqButPID s s1*

shows *open s* \longleftrightarrow *open s1*

$\langle proof \rangle$

lemma *not-open-eqButPID:*

assumes *1: $\neg open\ s$ and 2: eqButPID s s1*

shows $\neg open\ s1$

$\langle proof \rangle$

lemma *step-isCOMact-open:*

assumes *step s a = (ou, s')*

and *isCOMact a*

shows *open s' = open s*

$\langle proof \rangle$

lemma *validTrans-isCOMact-open:*

assumes *validTrans trn*

and *isCOMact (actOf trn)*

shows *open (tgtOf trn) = open (srcOf trn)*

$\langle proof \rangle$

lemma *list-all-isOVal-filter-isPValS*:
list-all isOVal vl \implies *filter (Not o isPValS) vl = vl*
 ⟨proof⟩

lemma *list-all-Not-isOVal-OVal-True*:
assumes *list-all (Not o isOVal) ul*
and *ul @ vll = OVal True # vll'*
shows *ul = []*
 ⟨proof⟩

lemma *list-all-filter-isOVal-isPVal-isPValS*:
assumes *list-all (Not o isOVal) ul*
and *filter isPValS ul = []* **and** *filter isPVal ul = []*
shows *ul = []*
 ⟨proof⟩

lemma *filter-list-all-isPValS-isOVal*:
assumes *list-all (Not o isOVal) ul* **and** *filter isPVal ul = []*
shows *list-all isPValS ul*
 ⟨proof⟩

lemma *filter-list-all-isPVal-isOVal*:
assumes *list-all (Not o isOVal) ul* **and** *filter isPValS ul = []*
shows *list-all isPVal ul*
 ⟨proof⟩

lemma *list-all-isPValS-Not-isOVal-filter*:
assumes *list-all isPValS ul* **shows** *list-all (Not o isOVal) ul \wedge filter isPVal ul = []*
 ⟨proof⟩

lemma *filter-isTValS-Nil*:
filter isPValS vl = [] \longleftrightarrow
list-all ($\lambda v. isPVal v \vee isOVal v$) vl
 ⟨proof⟩

fun $\varphi :: (state, act, out) trans \Rightarrow bool$ **where**
 $\varphi (Trans - (Uact (uPost uid p pid pst)) ou -) = (pid = PID \wedge ou = outOK)$
 |
 $\varphi (Trans - (COMact (comSendPost uid p aid pid)) ou -) = (pid = PID \wedge ou \neq outErr)$
 |
 $\varphi (Trans s - - s') = (open s \neq open s')$

lemma φ -def2:

assumes $step\ s\ a = (ou, s')$

shows

$\varphi (Trans\ s\ a\ ou\ s') \longleftrightarrow$
 $(\exists\ uid\ p\ pst. a = Uact\ (uPost\ uid\ p\ PID\ pst) \wedge ou = outOK) \vee$
 $(\exists\ uid\ p\ aid. a = COMact\ (comSendPost\ uid\ p\ aid\ PID) \wedge ou \neq outErr) \vee$
 $open\ s \neq open\ s'$
 $\langle proof \rangle$

lemma $uTextPost-out$:

assumes $1: step\ s\ a = (ou, s')$ **and** $a: a = Uact\ (uPost\ uid\ p\ PID\ pst)$ **and** $2: ou = outOK$

shows $uid = owner\ s\ PID \wedge p = pass\ s\ uid$

$\langle proof \rangle$

lemma $comSendPost-out$:

assumes $1: step\ s\ a = (ou, s')$ **and** $a: a = COMact\ (comSendPost\ uid\ p\ aid\ PID)$
and $2: ou \neq outErr$

shows $ou = O-sendPost\ (aid, clientPass\ s\ aid, PID, post\ s\ PID, owner\ s\ PID, vis\ s\ PID)$

$\wedge uid = admin\ s \wedge p = pass\ s\ (admin\ s)$

$\langle proof \rangle$

lemma $step-open-isCOMact$:

assumes $step\ s\ a = (ou, s')$

and $open\ s \neq open\ s'$

shows $\neg isCOMact\ a \wedge \neg (\exists\ ua. isuPost\ ua \wedge a = Uact\ ua)$

$\langle proof \rangle$

lemma $\varphi-def3$:

assumes $step\ s\ a = (ou, s')$

shows

$\varphi (Trans\ s\ a\ ou\ s') \longleftrightarrow$
 $(\exists\ pst. a = Uact\ (uPost\ (owner\ s\ PID)\ (pass\ s\ (owner\ s\ PID))\ PID\ pst) \wedge ou = outOK)$
 \vee
 $(\exists\ aid. a = COMact\ (comSendPost\ (admin\ s)\ (pass\ s\ (admin\ s))\ aid\ PID) \wedge$
 $ou = O-sendPost\ (aid, clientPass\ s\ aid, PID, post\ s\ PID, owner\ s\ PID, vis\ s\ PID))$
 \vee
 $open\ s \neq open\ s' \wedge \neg isCOMact\ a \wedge \neg (\exists\ ua. isuPost\ ua \wedge a = Uact\ ua)$
 $\langle proof \rangle$

fun $f :: (state, act, out)\ trans \Rightarrow value$ **where**

$f (Trans\ s\ (Uact\ (uPost\ uid\ p\ pid\ pst)) - s') =$

$(if\ pid = PID\ then\ PVal\ pst\ else\ OVal\ (open\ s'))$

$|$

$f (Trans\ s\ (COMact\ (comSendPost\ uid\ p\ aid\ pid)) (O-sendPost\ (-, -, -, pst, -)) s')$

$=$

$(if\ pid = PID\ then\ PValS\ aid\ pst\ else\ OVal\ (open\ s'))$

|
 $f (Trans s - - s') = OVal (open s')$

lemma *f-open-OVal*:

assumes $step s a = (ou, s')$

and $open s \neq open s' \wedge \neg isCOMact a \wedge \neg (\exists ua. isuPost ua \wedge a = Uact ua)$

shows $f (Trans s a ou s') = OVal (open s')$

<proof>

lemma *f-eq-PVal*:

assumes $step s a = (ou, s')$ **and** $\varphi (Trans s a ou s')$

and $f (Trans s a ou s') = PVal pst$

shows $a = Uact (uPost (owner s PID) (pass s (owner s PID))) PID pst$

<proof>

lemma *f-eq-PValS*:

assumes $step s a = (ou, s')$ **and** $\varphi (Trans s a ou s')$

and $f (Trans s a ou s') = PValS aid pst$

shows $a = COMact (comSendPost (admin s) (pass s (admin s))) aid PID$

<proof>

lemma *f-eq-OVal*:

assumes $step s a = (ou, s')$ **and** $\varphi (Trans s a ou s')$

and $f (Trans s a ou s') = OVal b$

shows $open s' \neq open s$

<proof>

lemma *uPost-comSendPost-open-eq*:

assumes $step: step s a = (ou, s')$

and $a: a = Uact (uPost uid p pid pst) \vee a = COMact (comSendPost uid p aid pid)$

shows $open s' = open s$

<proof>

lemma *step-open-φ-f-PVal-γ*:

assumes $s: reach s$

and $step: step s a = (ou, s')$

and $PID: PID \in set (postIDs s)$

and $op: \neg open s$ **and** $fi: \varphi (Trans s a ou s')$ **and** $f: f (Trans s a ou s') = PVal pst$

shows $\neg \gamma (Trans s a ou s')$

<proof>

lemma *Uact-uPaperC-step-eqButPID*:

assumes $a: a = Uact (uPost uid p PID pst)$

and $step s a = (ou, s')$

shows $eqButPID s s'$

<proof>

lemma *eqButPID-step- φ -imp*:
assumes *ss1*: *eqButPID s s1*
and *step*: *step s a = (ou,s[^])* **and** *step1*: *step s1 a = (ou1,s1[^])*
and φ : φ (*Trans s a ou s[^]*)
shows φ (*Trans s1 a ou1 s1[^]*)
<proof>

lemma *eqButPID-step- φ* :
assumes *s's1'*: *eqButPID s s1*
and *step*: *step s a = (ou,s[^])* **and** *step1*: *step s1 a = (ou1,s1[^])*
shows φ (*Trans s a ou s[^]*) = φ (*Trans s1 a ou1 s1[^]*)
<proof>

end

end
theory *DYNAMIC-Post-ISSUER*
imports
Post-Observation-Setup-ISSUER
DYNAMIC-Post-Value-Setup-ISSUER
Bounded-Deducibility-Security.Compositional-Reasoning
begin

6.5.2 Issuer declassification bound

We verify that a group of users of some node i , allowed to take normal actions to the system and observe their outputs *and additionally allowed to observe communication*, can learn nothing about the updates to a post located at node i and the sends of that post to other nodes beyond:

(1) the updates that occur during the times when one of the following holds, and the *last* update *before* one of the following holds (because, for example, observers can see the current version of the post when it is made public):

- either a user in the group is the post's owner or the administrator
- or a user in the group is a friend of the owner
- or the group has at least one registered user and the post is marked "public"

(2) the presence of the sends (i.e., the number of the sending actions)

(3) the public knowledge that what is being sent is always the last version (modeled as the correlation predicate)

See [3, Appendix C] for more details. This is the dynamic-trigger (i.e., trigger-incorporating inductive bound) version of the property proved in

Section 6.1. For a discussion of this “while-or-last-before” style of formalizing bounds, see [4, Section 3.4] about the the corresponding property of CoSMed.

context *Post*
begin

fun *T* :: (*state,act,out*) *trans* \Rightarrow *bool* **where** *T* - = *False*

inductive *BC* :: *value list* \Rightarrow *value list* \Rightarrow *bool*

and *BO* :: *value list* \Rightarrow *value list* \Rightarrow *bool*

where

BC-PVal[*simp,intro!*]:
 $list_all (Not \circ isOVal) ul \Longrightarrow list_all (Not \circ isOVal) ul1 \Longrightarrow$
 $map\ tgtAPI (filter\ isPValS\ ul) = map\ tgtAPI (filter\ isPValS\ ul1) \Longrightarrow$
 $(ul = [] \longrightarrow ul1 = [])$
 $\Longrightarrow BC\ ul\ ul1$
BC-BO[*intro!*]:
 $BO\ vl\ vl1 \Longrightarrow$
 $list_all (Not \circ isOVal) ul \Longrightarrow list_all (Not \circ isOVal) ul1 \Longrightarrow$
 $map\ tgtAPI (filter\ isPValS\ ul) = map\ tgtAPI (filter\ isPValS\ ul1) \Longrightarrow$
 $(ul = [] \longleftrightarrow ul1 = []) \Longrightarrow$
 $(ul \neq [] \Longrightarrow isPVal (last\ ul) \wedge last\ ul = last\ ul1) \Longrightarrow$
 $list_all\ isPValS\ sul$
 \Longrightarrow
 $BC (ul @ sul @ OVal\ True \# vl)$
 $(ul1 @ sul @ OVal\ True \# vl1)$

BO-PVal[*simp,intro!*]:
 $list_all (Not \circ isOVal) ul \Longrightarrow BO\ ul\ ul$

BO-BC[*intro!*]:
 $BC\ vl\ vl1 \Longrightarrow$
 $list_all (Not \circ isOVal) ul$
 \Longrightarrow
 $BO (ul @ OVal\ False \# vl) (ul @ OVal\ False \# vl1)$

lemma *list-all-filter-Not-isOVal*:

assumes $list_all (Not \circ isOVal) ul$

and $filter\ isPValS\ ul = []$ **and** $filter\ isPVal\ ul = []$

shows $ul = []$

<proof>

lemma *BC-not-Nil*: $BC\ vl\ vl1 \Longrightarrow vl = [] \Longrightarrow vl1 = []$

<proof>

lemma *BC-OVal-True*:

assumes $BC (OVal\ True \# vl')$ *vl1*

shows $\exists\ vl1'. BO\ vl'\ vl1' \wedge vl1 = OVal\ True \# vl1'$

<proof>

fun *corrFrom* :: *post* \Rightarrow *value list* \Rightarrow *bool* **where**
corrFrom *pst* [] = *True*
| *corrFrom* *pst* (*PVal* *psst* # *vl*) = *corrFrom* *psst* *vl*
| *corrFrom* *pst* (*PValS* *aid* *psst* # *vl*) = (*pst* = *psst* \wedge *corrFrom* *pst* *vl*)
| *corrFrom* *pst* (*OVal* *b* # *vl*) = (*corrFrom* *pst* *vl*)

abbreviation *corr* :: *value list* \Rightarrow *bool* **where** *corr* \equiv *corrFrom* *emptyPost*

definition *B* **where**
B *vl vl1* \equiv *BC* *vl vl1* \wedge *corr* *vl1*

lemma *B-not-Nil*:
assumes *B*: *B* *vl vl1* **and** *vl*: *vl* = []
shows *vl1* = []
<proof>

sublocale *BD-Security-IO* **where**
istate = *istate* **and** *step* = *step* **and**
 $\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
<proof>

6.5.3 Issuer unwinding proof

lemma *reach-PublicV-imples-FriendV[simp]*:
assumes *reach* *s*
and *vis* *s* *pid* \neq *PublicV*
shows *vis* *s* *pid* = *FriendV*
<proof>

lemma *eqButPID-step- γ -out*:
assumes *ss1*: *eqButPID* *s* *s1*
and *step*: *step* *s* *a* = (*ou*, *s'*) **and** *step1*: *step* *s1* *a* = (*ou1*, *s1'*)
and *op*: \neg *open* *s*
and *sT*: *reachNT* *s* **and** *s1*: *reach* *s1*
and γ : γ (*Trans* *s* *a* *ou* *s'*)
shows (\exists *uid* *p* *aid* *pid*. *a* = *COMact* (*comSendPost* *uid* *p* *aid* *pid*) \wedge *outPurge* *ou*
= *outPurge* *ou1*) \vee
 $ou = ou1$
<proof>

lemma *eqButPID-step-eq*:
assumes *ss1*: *eqButPID* *s* *s1*
and *a*: *a* = *Uact* (*uPost* *uid* *p* *PID* *pst*) *ou* = *outOK*

and *step*: $step\ s\ a = (ou, s')$ **and** *step1*: $step\ s1\ a = (ou', s1')$
shows $s' = s1'$
 $\langle proof \rangle$

definition $\Delta 0 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta 0\ s\ vl\ s1\ vl1 \equiv$
 $\neg PID \in \in postIDs\ s \wedge$
 $s = s1 \wedge BC\ vl\ vl1 \wedge$
 $corr\ vl1$

definition $\Delta 1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta 1\ s\ vl\ s1\ vl1 \equiv$
 $PID \in \in postIDs\ s \wedge$
 $list\ all\ (Not\ o\ isOVal)\ vl \wedge list\ all\ (Not\ o\ isOVal)\ vl1 \wedge$
 $map\ tgtAPI\ (filter\ isPValS\ vl) = map\ tgtAPI\ (filter\ isPValS\ vl1) \wedge$
 $(vl = [] \longrightarrow vl1 = []) \wedge$
 $eqButPID\ s\ s1 \wedge \neg open\ s \wedge$
 $corrFrom\ (post\ s1\ PID)\ vl1$

definition $\Delta 11 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta 11\ s\ vl\ s1\ vl1 \equiv$
 $PID \in \in postIDs\ s \wedge$
 $vl = [] \wedge list\ all\ isPVal\ vl1 \wedge$
 $eqButPID\ s\ s1 \wedge \neg open\ s \wedge$
 $corrFrom\ (post\ s1\ PID)\ vl1$

definition $\Delta 2 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta 2\ s\ vl\ s1\ vl1 \equiv$
 $PID \in \in postIDs\ s \wedge$
 $list\ all\ (Not\ o\ isOVal)\ vl \wedge$
 $vl = vl1 \wedge$
 $s = s1 \wedge open\ s \wedge$
 $corrFrom\ (post\ s1\ PID)\ vl1$

definition $\Delta 31 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta 31\ s\ vl\ s1\ vl1 \equiv$
 $PID \in \in postIDs\ s \wedge$
 $(\exists\ ul\ ul1\ sul\ vll\ vll1.$
 $\quad BO\ vll\ vll1 \wedge$
 $\quad list\ all\ (Not\ o\ isOVal)\ ul \wedge list\ all\ (Not\ o\ isOVal)\ ul1 \wedge$
 $\quad map\ tgtAPI\ (filter\ isPValS\ ul) = map\ tgtAPI\ (filter\ isPValS\ ul1) \wedge$
 $\quad ul \neq [] \wedge ul1 \neq [] \wedge$
 $\quad isPVal\ (last\ ul) \wedge last\ ul = last\ ul1 \wedge$
 $\quad list\ all\ isPValS\ sul \wedge$
 $\quad vl = ul @ sul @ OVal\ True \# vll \wedge vl1 = ul1 @ sul @ OVal\ True \# vll1) \wedge$
 $eqButPID\ s\ s1 \wedge \neg open\ s \wedge$
 $corrFrom\ (post\ s1\ PID)\ vl1$

definition $\Delta_{32} :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta_{32}\ s\ vl\ s1\ vl1 \equiv$
 $PID \in\in\ postIDs\ s \wedge$
 $(\exists\ sul\ vll\ vll1.$
 $\quad BO\ vll\ vll1 \wedge$
 $\quad list\ all\ isPValS\ sul \wedge$
 $\quad vl = sul\ @\ OVal\ True\ \# \ vll \wedge\ vl1 = sul\ @\ OVal\ True\ \# \ vll1) \wedge$
 $s = s1 \wedge \neg\ open\ s \wedge$
 $corrFrom\ (post\ s1\ PID)\ vl1$

definition $\Delta_4 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta_4\ s\ vl\ s1\ vl1 \equiv$
 $PID \in\in\ postIDs\ s \wedge$
 $(\exists\ ul\ vll\ vll1.$
 $\quad BC\ vll\ vll1 \wedge$
 $\quad list\ all\ (Not\ o\ isOVal)\ ul \wedge$
 $\quad vl = ul\ @\ OVal\ False\ \# \ vll \wedge\ vl1 = ul\ @\ OVal\ False\ \# \ vll1) \wedge$
 $s = s1 \wedge open\ s \wedge$
 $corrFrom\ (post\ s1\ PID)\ vl1$

lemma *istate- Δ_0* :
assumes $B: B\ vl\ vl1$
shows $\Delta_0\ istate\ vl\ istate\ vl1$
 $\langle proof \rangle$

lemma *list-all-filter[simp]*:
assumes $list\ all\ PP\ xs$
shows $filter\ PP\ xs = xs$
 $\langle proof \rangle$

lemma *unwind-cont- Δ_0* : $unwind\ cont\ \Delta_0\ \{\Delta_0, \Delta_1, \Delta_2, \Delta_{31}, \Delta_{32}, \Delta_4\}$
 $\langle proof \rangle$

lemma *unwind-cont- Δ_1* : $unwind\ cont\ \Delta_1\ \{\Delta_1, \Delta_{11}\}$
 $\langle proof \rangle$

lemma *unwind-cont- Δ_{11}* : $unwind\ cont\ \Delta_{11}\ \{\Delta_{11}\}$
 $\langle proof \rangle$

lemma *unwind-cont- Δ_{31}* : $unwind\ cont\ \Delta_{31}\ \{\Delta_{31}, \Delta_{32}\}$
 $\langle proof \rangle$

lemma *unwind-cont- Δ_{32}* : $unwind\ cont\ \Delta_{32}\ \{\Delta_2, \Delta_{32}, \Delta_4\}$
 $\langle proof \rangle$

lemma *unwind-cont- Δ_2* : $unwind\ cont\ \Delta_2\ \{\Delta_2\}$
 $\langle proof \rangle$

lemma *unwind-cont- Δ_4* : *unwind-cont Δ_4 $\{\Delta_1, \Delta_{31}, \Delta_{32}, \Delta_4\}$*
 ⟨*proof*⟩

definition *Gr* where

Gr =
 {
 (Δ_0 , $\{\Delta_0, \Delta_1, \Delta_2, \Delta_{31}, \Delta_{32}, \Delta_4\}$),
 (Δ_1 , $\{\Delta_1, \Delta_{11}\}$),
 (Δ_{11} , $\{\Delta_{11}\}$),
 (Δ_2 , $\{\Delta_2\}$),
 (Δ_{31} , $\{\Delta_{31}, \Delta_{32}\}$),
 (Δ_{32} , $\{\Delta_2, \Delta_{32}, \Delta_4\}$),
 (Δ_4 , $\{\Delta_1, \Delta_{31}, \Delta_{32}, \Delta_4\}$)
 }

theorem *secure*: *secure*
 ⟨*proof*⟩

end

end

theory *DYNAMIC-Post-COMPOSE2*

imports

DYNAMIC-Post-ISSUER

Post-RECEIVER

BD-Security-Compositional.Composing-Security

begin

6.5.4 Confidentiality for the (binary) issuer-receiver composition

type-synonym *ttrans* = (*state*, *act*, *out*) *trans*

type-synonym *value1* = *Post.value* **type-synonym** *value2* = *Post-RECEIVER.value*

type-synonym *obs1* = *Post-Observation-Setup-ISSUER.obs*

type-synonym *obs2* = *Post-Observation-Setup-RECEIVER.obs*

datatype *cval* = *PValC post*

type-synonym *cobs* = *obs1* × *obs2*

locale *Post-COMPOSE2* =

Iss: *Post UIDs PID* +

Rcv: *Post-RECEIVER UIDs2 PID AID1*

for *UIDs* :: *userID set* **and** *UIDs2* :: *userID set* **and**

AID1 :: *apiID* **and** *PID* :: *postID*

+ **fixes** *AID2* :: *apiID*

begin

abbreviation $\varphi1 \equiv Iss.\varphi$ **abbreviation** $f1 \equiv Iss.f$ **abbreviation** $\gamma1 \equiv Iss.\gamma$
abbreviation $g1 \equiv Iss.g$
 abbreviation $T1 \equiv Iss.T$ **abbreviation** $B1 \equiv Iss.B$
abbreviation $\varphi2 \equiv Rcv.\varphi$ **abbreviation** $f2 \equiv Rcv.f$ **abbreviation** $\gamma2 \equiv Rcv.\gamma$
abbreviation $g2 \equiv Rcv.g$
 abbreviation $T2 \equiv Rcv.T$ **abbreviation** $B2 \equiv Rcv.B$

fun $isCom1 :: ttrans \Rightarrow bool$ **where**
 $isCom1 (Trans\ s\ (COMact\ ca1)\ ou1\ s') = (ou1 \neq outErr)$
 $| isCom1 - = False$

fun $isCom2 :: ttrans \Rightarrow bool$ **where**
 $isCom2 (Trans\ s\ (COMact\ ca2)\ ou2\ s') = (ou2 \neq outErr)$
 $| isCom2 - = False$

fun $isComV1 :: value1 \Rightarrow bool$ **where**
 $isComV1 (Iss.PValS\ aid1\ pst1) = True$
 $| isComV1 - = False$

fun $isComV2 :: value2 \Rightarrow bool$ **where**
 $isComV2 (Rcv.PValR\ pst2) = True$

fun $syncV :: value1 \Rightarrow value2 \Rightarrow bool$ **where**
 $syncV (Iss.PValS\ aid1\ pst1)\ (Rcv.PValR\ pst2) = (pst1 = pst2)$
 $| syncV - - = False$

fun $cmpV :: value1 \Rightarrow value2 \Rightarrow cval$ **where**
 $cmpV (Iss.PValS\ aid1\ pst1)\ (Rcv.PValR\ pst2) = PValC\ pst1$
 $| cmpV - - = undefined$

fun $isComO1 :: obs1 \Rightarrow bool$ **where**
 $isComO1 (COMact\ ca1,\ ou1) = (ou1 \neq outErr)$
 $| isComO1 - = False$

fun $isComO2 :: obs2 \Rightarrow bool$ **where**
 $isComO2 (COMact\ ca2,\ ou2) = (ou2 \neq outErr)$
 $| isComO2 - = False$

fun $comSyncOA :: out \Rightarrow comActt \Rightarrow bool$ **where**
 $comSyncOA (O-sendServerReq\ (aid2,\ reqInfo1))\ (comReceiveClientReq\ aid1\ reqInfo2) =$
 $(aid1 = AID1 \wedge aid2 = AID2 \wedge reqInfo1 = reqInfo2)$
 $| comSyncOA (O-connectClient\ (aid2,\ sp1))\ (comConnectServer\ aid1\ sp2) =$
 $(aid1 = AID1 \wedge aid2 = AID2 \wedge sp1 = sp2)$

```

|comSyncOA (O-sendPost (aid2, sp1, pid1, pst1, uid1, vs1)) (comReceivePost aid1
sp2 pid2 pst2 uid2 vs2) =
  (aid1 = AID1  $\wedge$  aid2 = AID2  $\wedge$  (pid1, pst1, uid1, vs1) = (pid2, pst2, uid2,
vs2))
|comSyncOA (O-sendCreateOFriend (aid2, sp1, uid1, uid1')) (comReceiveCreateOFriend
aid1 sp2 uid2 uid2') =
  (aid1 = AID1  $\wedge$  aid2 = AID2  $\wedge$  (uid1, uid1') = (uid2, uid2'))
|comSyncOA (O-sendDeleteOFriend (aid2, sp1, uid1, uid1')) (comReceiveDeleteOFriend
aid1 sp2 uid2 uid2') =
  (aid1 = AID1  $\wedge$  aid2 = AID2  $\wedge$  (uid1, uid1') = (uid2, uid2'))
|comSyncOA - - = False

```

```

fun syncO :: obs1  $\Rightarrow$  obs2  $\Rightarrow$  bool where
  syncO (COMact ca1, ou1) (COMact ca2, ou2) =
    (ou1  $\neq$  outErr  $\wedge$  ou2  $\neq$  outErr  $\wedge$ 
      (comSyncOA ou1 ca2  $\vee$  comSyncOA ou2 ca1)
    )
|syncO - - = False

```

```

fun sync :: ttrans  $\Rightarrow$  ttrans  $\Rightarrow$  bool where
  sync (Trans s1 a1 ou1 s1') (Trans s2 a2 ou2 s2') = syncO (a1, ou1) (a2, ou2)

```

```

definition cmpO :: obs1  $\Rightarrow$  obs2  $\Rightarrow$  cobs where
  cmpO o1 o2  $\equiv$  (o1,o2)

```

```

lemma isCom1-isComV1:
assumes v: validTrans trn1 and r: reach (srcOf trn1) and  $\varphi$ 1:  $\varphi$ 1 trn1
shows isCom1 trn1  $\longleftrightarrow$  isComV1 (f1 trn1)
<proof>

```

```

lemma isCom1-isComO1:
assumes validTrans trn1 and reach (srcOf trn1) and  $\gamma$ 1 trn1
shows isCom1 trn1  $\longleftrightarrow$  isComO1 (g1 trn1)
<proof>

```

```

lemma isCom2-isComV2:
assumes validTrans trn2 and reach (srcOf trn2) and  $\varphi$ 2 trn2
shows isCom2 trn2  $\longleftrightarrow$  isComV2 (f2 trn2)
<proof>

```

```

lemma isCom2-isComO2:
assumes validTrans trn2 and reach (srcOf trn2) and  $\gamma$ 2 trn2
shows isCom2 trn2  $\longleftrightarrow$  isComO2 (g2 trn2)

```

<proof>

lemma *sync-syncV*:

assumes *v1: validTrans trn1 and reach (srcOf trn1)*
and *v2: validTrans trn2 and reach (srcOf trn2)*
and *c1: isCom1 trn1 and c2: isCom2 trn2 and $\varphi_1: \varphi_1 trn1$ and $\varphi_2: \varphi_2 trn2$*
and *sn: sync trn1 trn2*
shows *syncV (f1 trn1) (f2 trn2)*
<proof>

lemma *sync-syncO*:

assumes *validTrans trn1 and reach (srcOf trn1)*
and *validTrans trn2 and reach (srcOf trn2)*
and *isCom1 trn1 and isCom2 trn2 and $\gamma_1 trn1$ and $\gamma_2 trn2$*
and *sync trn1 trn2*
shows *syncO (g1 trn1) (g2 trn2)*
<proof>

lemma *sync- φ_1 - φ_2* :

assumes *v1: validTrans trn1 and r1: reach (srcOf trn1)*
and *v2: validTrans trn2 and s2: reach (srcOf trn2)*
and *c1: isCom1 trn1 and c2: isCom2 trn2*
and *sn: sync trn1 trn2*
shows *$\varphi_1 trn1 \longleftrightarrow \varphi_2 trn2$ (is ?A \longleftrightarrow ?B)*
<proof>

lemma *textPost-textPost-cong[intro]*:

assumes *textPost pst1 = textPost pst2*
and *setTextPost pst1 emptyText = setTextPost pst2 emptyText*
shows *pst1 = pst2*
<proof>

lemma *sync- φ - γ* :

assumes *validTrans trn1 and reach (srcOf trn1)*
and *validTrans trn2 and reach (srcOf trn2)*
and *isCom1 trn1 and isCom2 trn2*
and *$\gamma_1 trn1$ and $\gamma_2 trn2$*
and *so: syncO (g1 trn1) (g2 trn2)*
and *$\varphi_1 trn1 \implies \varphi_2 trn2 \implies syncV (f1 trn1) (f2 trn2)$*
shows *sync trn1 trn2*
<proof>

lemma *isCom1- γ_1* :

assumes *validTrans trn1 and reach (srcOf trn1) and isCom1 trn1*
shows *$\gamma_1 trn1$*
<proof>

lemma *isCom2- γ_2* :

assumes *validTrans trn2 and reach (srcOf trn2) and isCom2 trn2*

shows $\gamma^2 \text{ trn}^2$
<proof>

lemma *isCom2-V2*:
assumes *validTrans trn2* **and** *reach (srcOf trn2)* **and** $\varphi^2 \text{ trn}^2$
shows *isCom2 trn2*
<proof>

end

sublocale *Post-COMPOSE2 < BD-Security-TS-Comp* **where**
istate1 = istate **and** *validTrans1 = validTrans* **and** *srcOf1 = srcOf* **and** *tgtOf1 = tgtOf*
and $\varphi^1 = \varphi^1$ **and** $f^1 = f^1$ **and** $\gamma^1 = \gamma^1$ **and** $g^1 = g^1$ **and** $T^1 = T^1$ **and** $B^1 = B^1$
and
istate2 = istate **and** *validTrans2 = validTrans* **and** *srcOf2 = srcOf* **and** *tgtOf2 = tgtOf*
and $\varphi^2 = \varphi^2$ **and** $f^2 = f^2$ **and** $\gamma^2 = \gamma^2$ **and** $g^2 = g^2$ **and** $T^2 = T^2$ **and** $B^2 = B^2$
and *isCom1 = isCom1* **and** *isCom2 = isCom2* **and** *sync = sync*
and *isComV1 = isComV1* **and** *isComV2 = isComV2* **and** *syncV = syncV*
and *isComO1 = isComO1* **and** *isComO2 = isComO2* **and** *syncO = syncO*
<proof>

context *Post-COMPOSE2*
begin

theorem *secure: secure*
<proof>

end

end
theory *DYNAMIC-Post-Network*
imports
DYNAMIC-Post-ISSUER
Post-RECEIVER
../API-Network
BD-Security-Compositional.Composing-Security-Network
begin

6.5.5 Confidentiality for the N-ary composition

type-synonym *ttrans = (state, act, out) trans*

type-synonym *obs* = *Post-Observation-Setup-ISSUER.obs*
type-synonym *value* = *Post.value* + *Post-RECEIVER.value*

lemma *value-cases*:

fixes *v* :: *value*

obtains (*PVal*) *pst* **where** *v* = *Inl (Post.PVal pst)*
| (*PValS*) *aid pst* **where** *v* = *Inl (Post.PValS aid pst)*
| (*OVal*) *ov* **where** *v* = *Inl (Post.OVal ov)*
| (*PValR*) *pst* **where** *v* = *Inr (Post-RECEIVER.PValR pst)*
⟨*proof*⟩

locale *Post-Network* = *Network*

+ **fixes** *UIDs* :: *apiID* ⇒ *userID* *set*

and *AID* :: *apiID* **and** *PID* :: *postID*

assumes *AID-in-AIDs*: *AID* ∈ *AIDs*

begin

sublocale *Iss*: *Post UIDs AID PID* ⟨*proof*⟩

abbreviation φ :: *apiID* ⇒ (*state, act, out*) *trans* ⇒ *bool*

where φ *aid trn* ≡ (*if aid = AID then Iss.φ trn else Post-RECEIVER.φ PID AID trn*)

abbreviation *f* :: *apiID* ⇒ (*state, act, out*) *trans* ⇒ *value*

where *f aid trn* ≡ (*if aid = AID then Inl (Iss.f trn) else Inr (Post-RECEIVER.f PID AID trn)*)

abbreviation γ :: *apiID* ⇒ (*state, act, out*) *trans* ⇒ *bool*

where γ *aid trn* ≡ (*if aid = AID then Iss.γ trn else ObservationSetup-RECEIVER.γ (UIDs aid) trn*)

abbreviation *g* :: *apiID* ⇒ (*state, act, out*) *trans* ⇒ *obs*

where *g aid trn* ≡ (*if aid = AID then Iss.g trn else ObservationSetup-RECEIVER.g PID AID trn*)

abbreviation *T* :: *apiID* ⇒ (*state, act, out*) *trans* ⇒ *bool*

where *T aid trn* ≡ (*if aid = AID then Iss.T trn else Post-RECEIVER.T (UIDs aid) PID AID trn*)

lemma *T-def*:

T aid trn ⇔ *aid ≠ AID* ∧ *Post-RECEIVER.T (UIDs aid) PID AID trn*
⟨*proof*⟩

abbreviation *B* :: *apiID* ⇒ *value list* ⇒ *value list* ⇒ *bool*

where *B aid vl vl1* ≡

(*if aid = AID then list-all isl vl* ∧ *list-all isl vl1* ∧ *Iss.B (map projl vl) (map projl vl1)*)

else list-all (Not o isl) vl ∧ *list-all (Not o isl) vl1* ∧ *Post-RECEIVER.B (map*

projr vl) (*map projr vl1*)

fun *comOfV* :: *apiID* ⇒ *value* ⇒ *com* **where**
comOfV aid (Inl (Post.PValS aid' pst)) = (*if aid' ≠ aid then Send else Internal*)
| *comOfV aid (Inl (Post.PVal pst))* = *Internal*
| *comOfV aid (Inl (Post.OVal ov))* = *Internal*
| *comOfV aid (Inr v)* = *Recv*

fun *tgtNodeOfV* :: *apiID* ⇒ *value* ⇒ *apiID* **where**
tgtNodeOfV aid (Inl (Post.PValS aid' pst)) = *aid'*
| *tgtNodeOfV aid (Inl (Post.PVal pst))* = *undefined*
| *tgtNodeOfV aid (Inl (Post.OVal ov))* = *undefined*
| *tgtNodeOfV aid (Inr v)* = *AID*

definition *syncV* :: *apiID* ⇒ *value* ⇒ *apiID* ⇒ *value* ⇒ *bool* **where**
syncV aid1 v1 aid2 v2 =
(\exists *pst. aid1 = AID* ∧ *v1 = Inl (Post.PValS aid2 pst)* ∧ *v2 = Inr (Post-RECEIVER.PValR pst)*)

lemma *syncVI*: *syncV AID (Inl (Post.PValS aid' pst)) aid' (Inr (Post-RECEIVER.PValR pst))*
⟨*proof*⟩

lemma *syncVE*:

assumes *syncV aid1 v1 aid2 v2*

obtains *pst* **where** *aid1 = AID* *v1 = Inl (Post.PValS aid2 pst)* *v2 = Inr (Post-RECEIVER.PValR pst)*

⟨*proof*⟩

fun *getTgtV* **where**

getTgtV (Inl (Post.PValS aid pst)) = *Inr (Post-RECEIVER.PValR pst)*
| *getTgtV v* = *v*

lemma *comOfV-AID*:

comOfV AID v = Send ⇔ *isl v* ∧ *Iss.isPValS (projl v)* ∧ *Iss.tgtAPI (projl v)*
≠ *AID*

comOfV AID v = Recv ⇔ *Not (isl v)*

⟨*proof*⟩

lemmas φ -*defs* = *Post-RECEIVER.φ-def2* *Iss.φ-def3*

sublocale *Net*: *BD-Security-TS-Network-getTgtV*

where *istate* = λ-. *istate* **and** *validTrans* = *validTrans* **and** *srcOf* = λ-. *srcOf*
and *tgtOf* = λ-. *tgtOf*

and *nodes* = *AIDs* **and** *comOf* = *comOf* **and** *tgtNodeOf* = *tgtNodeOf*

and *sync* = *sync* **and** φ = φ **and** *f* = *f* **and** γ = γ **and** *g* = *g* **and** *T* = *T* **and**
B = *B*

and *comOfV* = *comOfV* **and** *tgtNodeOfV* = *tgtNodeOfV* **and** *syncV* = *syncV*

and *comOfO* = *comOfO* **and** *tgtNodeOfO* = *tgtNodeOfO* **and** *syncO* = *syncO*

and $source = AID$ **and** $getTgtV = getTgtV$
 $\langle proof \rangle$

lemma *list-all-Not-isl-projectSrcV*: $list-all (Not\ o\ isl) (Net.projectSrcV\ aid\ vlSrc)$
 $\langle proof \rangle$

context
fixes $AID' :: apiID$
assumes $AID' : AID' \in AIDs - \{AID\}$
begin

interpretation *Recv*: *Post-RECEIVER UIDs AID' PID AID* $\langle proof \rangle$

lemma *Iss-BC-BO-tgtAPI*:
shows $(Iss.BC\ vl\ vl1 \longrightarrow map\ Iss.tgtAPI\ (filter\ Iss.isPValS\ vl) =$
 $map\ Iss.tgtAPI\ (filter\ Iss.isPValS\ vl1)) \wedge$
 $(Iss.BO\ vl\ vl1 \longrightarrow map\ Iss.tgtAPI\ (filter\ Iss.isPValS\ vl) =$
 $map\ Iss.tgtAPI\ (filter\ Iss.isPValS\ vl1))$
 $\langle proof \rangle$

lemma *Iss-B-Recv-B-aux*:
assumes $list-all\ isl\ vl$
and $list-all\ isl\ vl1$
and $map\ Iss.tgtAPI\ (filter\ Iss.isPValS\ (map\ projl\ vl)) =$
 $map\ Iss.tgtAPI\ (filter\ Iss.isPValS\ (map\ projl\ vl1))$
shows $length\ (map\ projr\ (Net.projectSrcV\ AID'\ vl)) = length\ (map\ projr\ (Net.projectSrcV$
 $AID'\ vl1))$
 $\langle proof \rangle$

lemma *Iss-B-Recv-B*:
assumes $B\ AID\ vl\ vl1$
shows $Recv.B\ (map\ projr\ (Net.projectSrcV\ AID'\ vl))\ (map\ projr\ (Net.projectSrcV$
 $AID'\ vl1))$
 $\langle proof \rangle$

end

lemma *map-projl-Inl*: $map\ (projl\ o\ Inl)\ vl = vl$
 $\langle proof \rangle$

lemma *these-map-Inl-projl*: $list-all\ isl\ vl \implies these\ (map\ (Some\ o\ Inl\ o\ projl)\ vl)$
 $=\ vl$
 $\langle proof \rangle$

lemma *map-projr-Inr*: $map\ (projr\ o\ Inr)\ vl = vl$
 $\langle proof \rangle$

lemma *these-map-Inr-projr*: $list-all\ (Not\ o\ isl)\ vl \implies these\ (map\ (Some\ o\ Inr\ o\$
 $projr)\ vl) = vl$

<proof>

```
sublocale BD-Security-TS-Network-Preserve-Source-Security-getTgtV
where istate =  $\lambda$ -. istate and validTrans = validTrans and srcOf =  $\lambda$ -. srcOf
and tgtOf =  $\lambda$ -. tgtOf
  and nodes = AIDs and comOf = comOf and tgtNodeOf = tgtNodeOf
  and sync = sync and  $\varphi$  =  $\varphi$  and f = f and  $\gamma$  =  $\gamma$  and g = g and T = T and
  B = B
  and comOfV = comOfV and tgtNodeOfV = tgtNodeOfV and syncV = syncV
  and comOfO = comOfO and tgtNodeOfO = tgtNodeOfO and syncO = syncO
  and source = AID and getTgtV = getTgtV
<proof>
```

theorem *secure: secure*

<proof>

end

end

theory *Independent-Post-Observation-Setup-ISSUER*

imports

../Safety-Properties

../Post-Observation-Setup-ISSUER

begin

6.6 Variation with multiple independent secret posts

This section formalizes the lifting of the confidentiality of one given (arbitrary but fixed) post to the confidentiality of two posts of arbitrary nodes of the network, as described in [3, Appendix E].

6.6.1 Issuer observation setup

locale *Strong-ObservationSetup-ISSUER* = *Fixed-UIDs* + *Fixed-PID*

begin

fun γ :: $(state, act, out) trans \Rightarrow bool$ **where**

$\gamma (Trans - a - -) \longleftrightarrow$

$(\exists uid. userOfA a = Some\ uid \wedge uid \in UIDs)$

\vee

— Communication actions are considered to be observable in order to make the security properties compositional

$(\exists ca. a = COMact\ ca)$

\vee

— The following actions are added to strengthen the observers in order to show that all posts *other than PID* are completely independent of *PID*; the confidentiality

of PID is protected even if the observers can see all updates to other posts (and actions contributing to the declassification triggers of those posts).

$$\begin{aligned}
& (\exists uid\ p\ pid\ pst. a = Uact\ (uPost\ uid\ p\ pid\ pst) \wedge pid \neq PID) \\
& \vee \\
& (\exists uid\ p. a = Sact\ (sSys\ uid\ p)) \\
& \vee \\
& (\exists uid\ p\ uid'\ p'. a = Cact\ (cUser\ uid\ p\ uid'\ p')) \\
& \vee \\
& (\exists uid\ p\ pid. a = Cact\ (cPost\ uid\ p\ pid)) \\
& \vee \\
& (\exists uid\ p\ uid'. a = Cact\ (cFriend\ uid\ p\ uid')) \\
& \vee \\
& (\exists uid\ p\ uid'. a = Dact\ (dFriend\ uid\ p\ uid')) \\
& \vee \\
& (\exists uid\ p\ pid\ v. a = Uact\ (uVisPost\ uid\ p\ pid\ v))
\end{aligned}$$

fun $sPurge :: sActt \Rightarrow sActt$ **where**
 $sPurge\ (sSys\ uid\ pwd) = sSys\ uid\ emptyPass$

fun $comPurge :: comActt \Rightarrow comActt$ **where**
 $comPurge\ (comSendServerReq\ uID\ p\ aID\ reqInfo) = comSendServerReq\ uID\ emptyPass\ aID\ reqInfo$
 $| comPurge\ (comConnectClient\ uID\ p\ aID\ sp) = comConnectClient\ uID\ emptyPass\ aID\ sp$
 $| comPurge\ (comConnectServer\ aID\ sp) = comConnectServer\ aID\ sp$
 $| comPurge\ (comSendPost\ uID\ p\ aID\ pID) = comSendPost\ uID\ emptyPass\ aID\ pID$
 $| comPurge\ (comSendCreateOFriend\ uID\ p\ aID\ uID') = comSendCreateOFriend\ uID\ emptyPass\ aID\ uID'$
 $| comPurge\ (comSendDeleteOFriend\ uID\ p\ aID\ uID') = comSendDeleteOFriend\ uID\ emptyPass\ aID\ uID'$
 $| comPurge\ ca = ca$

fun $outPurge :: out \Rightarrow out$ **where**
 $outPurge\ (O-sendPost\ (aID,\ sp,\ pID,\ pst,\ uID,\ vs)) =$
 $\quad (let\ pst' = (if\ pID = PID\ then\ emptyPost\ else\ pst)$
 $\quad \quad in\ O-sendPost\ (aID,\ sp,\ pID,\ pst',\ uID,\ vs))$
 $| outPurge\ ou = ou$

fun $g :: (state,act,out)trans \Rightarrow obs$ **where**
 $g\ (Trans - (Sact\ sa)\ ou\ -) = (Sact\ (sPurge\ sa),\ outPurge\ ou)$
 $| g\ (Trans - (COMact\ ca)\ ou\ -) = (COMact\ (comPurge\ ca),\ outPurge\ ou)$
 $| g\ (Trans - a\ ou\ -) = (a,ou)$

lemma $comPurge-simps$:

$comPurge\ ca = comSendServerReq\ uID\ p\ aID\ reqInfo \longleftrightarrow (\exists p'.\ ca = comSendServerReq\ uID\ p'\ aID\ reqInfo \wedge p = emptyPass)$
 $comPurge\ ca = comReceiveClientReq\ aID\ reqInfo \longleftrightarrow ca = comReceiveClientReq\ aID\ reqInfo$
 $comPurge\ ca = comConnectClient\ uID\ p\ aID\ sp \longleftrightarrow (\exists p'.\ ca = comConnectClient\ uID\ p'\ aID\ sp \wedge p = emptyPass)$
 $comPurge\ ca = comConnectServer\ aID\ sp \longleftrightarrow ca = comConnectServer\ aID\ sp$
 $comPurge\ ca = comReceivePost\ aID\ sp\ nID\ nt\ uID\ v \longleftrightarrow ca = comReceivePost\ aID\ sp\ nID\ nt\ uID\ v$
 $comPurge\ ca = comSendPost\ uID\ p\ aID\ nID \longleftrightarrow (\exists p'.\ ca = comSendPost\ uID\ p'\ aID\ nID \wedge p = emptyPass)$
 $comPurge\ ca = comSendCreateOFriend\ uID\ p\ aID\ uID' \longleftrightarrow (\exists p'.\ ca = comSendCreateOFriend\ uID\ p'\ aID\ uID' \wedge p = emptyPass)$
 $comPurge\ ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uID'$
 $comPurge\ ca = comSendDeleteOFriend\ uID\ p\ aID\ uID' \longleftrightarrow (\exists p'.\ ca = comSendDeleteOFriend\ uID\ p'\ aID\ uID' \wedge p = emptyPass)$
 $comPurge\ ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'$
 {proof}

lemma *outPurge-simps[simp]*:

$outPurge\ ou = outErr \longleftrightarrow ou = outErr$
 $outPurge\ ou = outOK \longleftrightarrow ou = outOK$
 $outPurge\ ou = O-sendServerReq\ ossr \longleftrightarrow ou = O-sendServerReq\ ossr$
 $outPurge\ ou = O-connectClient\ occ \longleftrightarrow ou = O-connectClient\ occ$
 $outPurge\ ou = O-sendPost\ (aid,\ sp,\ pid,\ pst',\ uid,\ vs) \longleftrightarrow (\exists pst.\ ou = O-sendPost\ (aid,\ sp,\ pid,\ pst,\ uid,\ vs) \wedge pst' = (if\ pid = PID\ then\ emptyPost\ else\ pst))$
 $outPurge\ ou = O-sendCreateOFriend\ oscf \longleftrightarrow ou = O-sendCreateOFriend\ oscf$
 $outPurge\ ou = O-sendDeleteOFriend\ osdf \longleftrightarrow ou = O-sendDeleteOFriend\ osdf$
 {proof}

lemma *g-simps*:

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendServerReq\ uID\ p\ aID\ reqInfo),\ O-sendServerReq\ ossr)$
 $\longleftrightarrow (\exists p'.\ a = COMact\ (comSendServerReq\ uID\ p'\ aID\ reqInfo) \wedge p = emptyPass \wedge ou = O-sendServerReq\ ossr)$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveClientReq\ aID\ reqInfo),\ outOK)$
 $\longleftrightarrow a = COMact\ (comReceiveClientReq\ aID\ reqInfo) \wedge ou = outOK$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectClient\ uID\ p\ aID\ sp),\ O-connectClient\ occ)$
 $\longleftrightarrow (\exists p'.\ a = COMact\ (comConnectClient\ uID\ p'\ aID\ sp) \wedge p = emptyPass \wedge ou = O-connectClient\ occ)$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectServer\ aID\ sp),\ outOK)$
 $\longleftrightarrow a = COMact\ (comConnectServer\ aID\ sp) \wedge ou = outOK$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceivePost\ aID\ sp\ nID\ nt\ uID\ v),\ outOK)$
 $\longleftrightarrow a = COMact\ (comReceivePost\ aID\ sp\ nID\ nt\ uID\ v) \wedge ou = outOK$

$g (\text{Trans } s \ a \ ou \ s') = (\text{COMact } (\text{comSendPost } uID \ p \ aID \ nID), \text{O-sendPost } (aid, sp, pid, pst', uid, vs))$
 $\longleftrightarrow (\exists pst \ p'. a = \text{COMact } (\text{comSendPost } uID \ p' \ aID \ nID) \wedge p = \text{emptyPass} \wedge ou = \text{O-sendPost } (aid, sp, pid, pst, uid, vs) \wedge pst' = (\text{if } pid = PID \text{ then } \text{emptyPost} \text{ else } pst))$
 $g (\text{Trans } s \ a \ ou \ s') = (\text{COMact } (\text{comSendCreateOFriend } uID \ p \ aID \ uID'), \text{O-sendCreateOFriend } (aid, sp, uid, uid'))$
 $\longleftrightarrow (\exists p'. a = (\text{COMact } (\text{comSendCreateOFriend } uID \ p' \ aID \ uID')) \wedge p = \text{emptyPass} \wedge ou = \text{O-sendCreateOFriend } (aid, sp, uid, uid'))$
 $g (\text{Trans } s \ a \ ou \ s') = (\text{COMact } (\text{comReceiveCreateOFriend } aID \ cp \ uID \ uID'), \text{outOK})$
 $\longleftrightarrow a = \text{COMact } (\text{comReceiveCreateOFriend } aID \ cp \ uID \ uID') \wedge ou = \text{outOK}$
 $g (\text{Trans } s \ a \ ou \ s') = (\text{COMact } (\text{comSendDeleteOFriend } uID \ p \ aID \ uID'), \text{O-sendDeleteOFriend } (aid, sp, uid, uid'))$
 $\longleftrightarrow (\exists p'. a = \text{COMact } (\text{comSendDeleteOFriend } uID \ p' \ aID \ uID') \wedge p = \text{emptyPass} \wedge ou = \text{O-sendDeleteOFriend } (aid, sp, uid, uid'))$
 $g (\text{Trans } s \ a \ ou \ s') = (\text{COMact } (\text{comReceiveDeleteOFriend } aID \ cp \ uID \ uID'), \text{outOK})$
 $\longleftrightarrow a = \text{COMact } (\text{comReceiveDeleteOFriend } aID \ cp \ uID \ uID') \wedge ou = \text{outOK}$
<proof>

end

end

theory *Independent-DYNAMIC-Post-Value-Setup-ISSUER*

imports

../Safety-Properties

Independent-Post-Observation-Setup-ISSUER

../Post-Unwinding-Helper-ISSUER

begin

6.6.2 Issuer value setup

locale *Post = Strong-ObservationSetup-ISSUER*

begin

datatype *value* =

isPVal: *PVal post* — updating the post content locally

| *isPValS*: *PValS (tgtAPI: apiID) post* — sending the post to another node

| *isOVal*: *OVal bool* — change in the dynamic declassification trigger condition

definition *open where*

open s \equiv

$\exists uid \in \text{UIDs.}$

$uid \in \in \text{userIDs } s \wedge PID \in \in \text{postIDs } s \wedge$

$(uid = \text{admin } s \vee uid = \text{owner } s \text{ PID} \vee uid \in \in \text{friendIDs } s \text{ (owner } s \text{ PID)}) \vee$

$\text{vis } s \text{ PID} = \text{Public } V)$

sublocale *Issuer-State-Equivalence-Up-To-PID* $\langle proof \rangle$

lemma *eqButPID-open*:
assumes *eqButPID s s1*
shows $open\ s \longleftrightarrow open\ s1$
 $\langle proof \rangle$

lemma *not-open-eqButPID*:
assumes *1: $\neg open\ s$ and 2: *eqButPID s s1**
shows $\neg open\ s1$
 $\langle proof \rangle$

lemma *step-isCOMact-open*:
assumes *step s a = (ou, s')*
and *isCOMact a*
shows $open\ s' = open\ s$
 $\langle proof \rangle$

lemma *validTrans-isCOMact-open*:
assumes *validTrans trn*
and *isCOMact (actOf trn)*
shows $open\ (tgtOf\ trn) = open\ (srcOf\ trn)$
 $\langle proof \rangle$

lemma *list-all-isOVal-filter-isPValS*:
list-all isOVal vl \implies filter (Not o isPValS) vl = vl
 $\langle proof \rangle$

lemma *list-all-Not-isOVal-OVal-True*:
assumes *list-all (Not o isOVal) ul*
and *ul @ vll = OVal True # vll'*
shows $ul = []$
 $\langle proof \rangle$

lemma *list-all-filter-isOVal-isPVal-isPValS*:
assumes *list-all (Not o isOVal) ul*
and *filter isPValS ul = [] and filter isPVal ul = []*
shows $ul = []$
 $\langle proof \rangle$

lemma *filter-list-all-isPValS-isOVal*:
assumes *list-all (Not o isOVal) ul and filter isPVal ul = []*
shows *list-all isPValS ul*
 $\langle proof \rangle$

lemma *filter-list-all-isPVal-isOVal*:
assumes *list-all (Not o isOVal) ul and filter isPValS ul = []*
shows *list-all isPVal ul*
 $\langle proof \rangle$

lemma *list-all-isPValS-Not-isOVal-filter*:

assumes *list-all isPValS ul* **shows** *list-all (Not o isOVal) ul* \wedge *filter isPVal ul = []*
 \langle *proof* \rangle

lemma *filter-isTValS-Nil*:

filter isPValS vl = [] \longleftrightarrow
list-all $(\lambda v. \text{isPVal } v \vee \text{isOVal } v)$ *vl*
 \langle *proof* \rangle

fun $\varphi :: (\text{state}, \text{act}, \text{out}) \text{ trans} \Rightarrow \text{bool}$ **where**

$\varphi (\text{Trans } - (\text{Uact } (\text{uPost } \text{uid } p \text{ pid } \text{pst})) \text{ ou } -) = (\text{pid} = \text{PID} \wedge \text{ou} = \text{outOK})$

|

$\varphi (\text{Trans } - (\text{COMact } (\text{comSendPost } \text{uid } p \text{ aid } \text{pid})) \text{ ou } -) = (\text{pid} = \text{PID} \wedge \text{ou} \neq \text{outErr})$

|

$\varphi (\text{Trans } s \text{ - - } s') = (\text{open } s \neq \text{open } s')$

lemma φ -def1:

$\varphi \text{ trn} \longleftrightarrow$
 $(\exists \text{uid } p \text{ pst. actOf trn} = \text{Uact } (\text{uPost } \text{uid } p \text{ PID } \text{pst}) \wedge \text{outOf trn} = \text{outOK}) \vee$
 $(\exists \text{uid } p \text{ aid. actOf trn} = \text{COMact } (\text{comSendPost } \text{uid } p \text{ aid } \text{PID}) \wedge \text{outOf trn} \neq \text{outErr}) \vee$
 $(\forall \text{uid } p \text{ pid } \text{pst. actOf trn} \neq \text{Uact } (\text{uPost } \text{uid } p \text{ pid } \text{pst})) \wedge$
 $(\forall \text{uid } p \text{ aid } \text{pid. actOf trn} \neq \text{COMact } (\text{comSendPost } \text{uid } p \text{ aid } \text{pid})) \wedge$
 $\text{open } (\text{srcOf trn}) \neq \text{open } (\text{tgtOf trn})$
 \langle *proof* \rangle

lemma φ -def2:

assumes *step s a = (ou, s')*

shows

$\varphi (\text{Trans } s \text{ a } \text{ou } s') \longleftrightarrow$
 $(\exists \text{uid } p \text{ pst. a} = \text{Uact } (\text{uPost } \text{uid } p \text{ PID } \text{pst}) \wedge \text{ou} = \text{outOK}) \vee$
 $(\exists \text{uid } p \text{ aid. a} = \text{COMact } (\text{comSendPost } \text{uid } p \text{ aid } \text{PID}) \wedge \text{ou} \neq \text{outErr}) \vee$
 $\text{open } s \neq \text{open } s'$
 \langle *proof* \rangle

lemma *uTextPost-out*:

assumes 1: *step s a = (ou, s')* **and** *a: a = Uact (uPost uid p PID pst)* **and** 2: *ou = outOK*

shows *uid = owner s PID* \wedge *p = pass s uid*

\langle *proof* \rangle

lemma *comSendPost-out*:

assumes 1: *step s a = (ou, s')* **and** *a: a = COMact (comSendPost uid p aid PID)*
and 2: *ou* \neq *outErr*

shows $ou = O\text{-sendPost} (aid, clientPass\ s\ aid, PID, post\ s\ PID, owner\ s\ PID, vis\ s\ PID)$
 $\wedge uid = admin\ s \wedge p = pass\ s\ (admin\ s)$
 $\langle proof \rangle$

lemma *step-open-isCOMact*:
assumes $step\ s\ a = (ou, s')$
and $open\ s \neq open\ s'$
shows $\neg isCOMact\ a \wedge \neg (\exists ua. isuPost\ ua \wedge a = Uact\ ua)$
 $\langle proof \rangle$

lemma $\varphi\text{-def3}$:
assumes $step\ s\ a = (ou, s')$
shows
 $\varphi (Trans\ s\ a\ ou\ s') \longleftrightarrow$
 $(\exists pst. a = Uact (uPost (owner\ s\ PID) (pass\ s\ (owner\ s\ PID))) PID\ pst) \wedge ou = outOK)$
 \vee
 $(\exists aid. a = COMact (comSendPost (admin\ s) (pass\ s\ (admin\ s)) aid\ PID) \wedge$
 $ou = O\text{-sendPost} (aid, clientPass\ s\ aid, PID, post\ s\ PID, owner\ s\ PID, vis\ s\ PID))$
 \vee
 $open\ s \neq open\ s' \wedge \neg isCOMact\ a \wedge \neg (\exists ua. isuPost\ ua \wedge a = Uact\ ua)$
 $\langle proof \rangle$

fun $f :: (state, act, out) trans \Rightarrow value$ **where**
 $f (Trans\ s (Uact (uPost\ uid\ p\ pid\ pst)) - s') =$
 $(if\ pid = PID\ then\ PVal\ pst\ else\ OVal (open\ s'))$
 $|$
 $f (Trans\ s (COMact (comSendPost\ uid\ p\ aid\ pid)) (O\text{-sendPost} (-, -, -, pst, -)) s')$
 $=$
 $(if\ pid = PID\ then\ PValS\ aid\ pst\ else\ OVal (open\ s'))$
 $|$
 $f (Trans\ s - - s') = OVal (open\ s')$

lemma *f-open-OVal*:
assumes $step\ s\ a = (ou, s')$
and $open\ s \neq open\ s' \wedge \neg isCOMact\ a \wedge \neg (\exists ua. isuPost\ ua \wedge a = Uact\ ua)$
shows $f (Trans\ s\ a\ ou\ s') = OVal (open\ s')$
 $\langle proof \rangle$

lemma *f-eq-PVal*:
assumes $step\ s\ a = (ou, s')$ **and** $\varphi (Trans\ s\ a\ ou\ s')$
and $f (Trans\ s\ a\ ou\ s') = PVal\ pst$
shows $a = Uact (uPost (owner\ s\ PID) (pass\ s\ (owner\ s\ PID))) PID\ pst$
 $\langle proof \rangle$

lemma *f-eq-PValS*:
assumes $step\ s\ a = (ou, s')$ **and** $\varphi (Trans\ s\ a\ ou\ s')$

and $f (Trans\ s\ a\ ou\ s') = PValS\ aid\ pst$
shows $a = COMact\ (comSendPost\ (admin\ s)\ (pass\ s\ (admin\ s))\ aid\ PID)$
 $\langle proof \rangle$

lemma *f-eq-OVal*:
assumes $step\ s\ a = (ou, s')$ **and** $\varphi (Trans\ s\ a\ ou\ s')$
and $f (Trans\ s\ a\ ou\ s') = OVal\ b$
shows $open\ s' \neq open\ s$
 $\langle proof \rangle$

lemma *uPost-comSendPost-open-eq*:
assumes $step\ s\ a = (ou, s')$
and $a: a = Uact\ (uPost\ uid\ p\ pid\ pst) \vee a = COMact\ (comSendPost\ uid\ p\ aid\ pid)$
shows $open\ s' = open\ s$
 $\langle proof \rangle$

lemma *step-open-φ-f-PVal-γ*:
assumes $s: reach\ s$
and $step: step\ s\ a = (ou, s')$
and $PID: PID \in set\ (postIDs\ s)$
and $op: \neg open\ s$ **and** $fi: \varphi (Trans\ s\ a\ ou\ s')$ **and** $f: f (Trans\ s\ a\ ou\ s') = PVal\ pst$
shows $\neg \gamma (Trans\ s\ a\ ou\ s')$
 $\langle proof \rangle$

lemma *Uact-uPaperC-step-eqButPID*:
assumes $a: a = Uact\ (uPost\ uid\ p\ PID\ pst)$
and $step\ s\ a = (ou, s')$
shows $eqButPID\ s\ s'$
 $\langle proof \rangle$

lemma *eqButPID-step-φ-imp*:
assumes $ss1: eqButPID\ s\ s1$
and $step: step\ s\ a = (ou, s')$ **and** $step1: step\ s1\ a = (ou1, s1')$
and $\varphi: \varphi (Trans\ s\ a\ ou\ s')$
shows $\varphi (Trans\ s1\ a\ ou1\ s1')$
 $\langle proof \rangle$

lemma *eqButPID-step-φ*:
assumes $s's1': eqButPID\ s\ s1$
and $step: step\ s\ a = (ou, s')$ **and** $step1: step\ s1\ a = (ou1, s1')$
shows $\varphi (Trans\ s\ a\ ou\ s') = \varphi (Trans\ s1\ a\ ou1\ s1')$
 $\langle proof \rangle$

end

```

end
theory Independent-DYNAMIC-Post-ISSUER
  imports
    Independent-Post-Observation-Setup-ISSUER
    Independent-DYNAMIC-Post-Value-Setup-ISSUER
    Bounded-Deducibility-Security.Compositional-Reasoning
begin

```

6.6.3 Issuer declassification bound

```

context Post
begin

```

```

fun T :: (state,act,out) trans  $\Rightarrow$  bool where T - = False

```

We again use the dynamic declassification bound for the issuer node (Section 6.5.2).

```

inductive BC :: value list  $\Rightarrow$  value list  $\Rightarrow$  bool
and BO :: value list  $\Rightarrow$  value list  $\Rightarrow$  bool
where

```

```

  BC-PVal[simp,intro!]:
    list-all (Not o isOVal) ul  $\Longrightarrow$  list-all (Not o isOVal) ul1  $\Longrightarrow$ 
      map tgtAPI (filter isPValS ul) = map tgtAPI (filter isPValS ul1)  $\Longrightarrow$ 
      (ul = []  $\longrightarrow$  ul1 = [])
       $\Longrightarrow$  BC ul ul1
|BC-BO[intro!]:
  BO vl vl1  $\Longrightarrow$ 
    list-all (Not o isOVal) ul  $\Longrightarrow$  list-all (Not o isOVal) ul1  $\Longrightarrow$ 
      map tgtAPI (filter isPValS ul) = map tgtAPI (filter isPValS ul1)  $\Longrightarrow$ 
      (ul = []  $\longleftrightarrow$  ul1 = [])  $\Longrightarrow$ 
      (ul  $\neq$  []  $\Longrightarrow$  isPVal (last ul)  $\wedge$  last ul = last ul1)  $\Longrightarrow$ 
      list-all isPValS sul
       $\Longrightarrow$ 
      BC (ul @ sul @ OVal True # vl)
      (ul1 @ sul @ OVal True # vl1)

```

```

|BO-PVal[simp,intro!]:
  list-all (Not o isOVal) ul  $\Longrightarrow$  BO ul ul
|BO-BC[intro!]:
  BC vl vl1  $\Longrightarrow$ 
    list-all (Not o isOVal) ul
     $\Longrightarrow$ 
    BO (ul @ OVal False # vl) (ul @ OVal False # vl1)

```

```

lemma list-all-filter-Not-isOVal:
assumes list-all (Not o isOVal) ul
and filter isPValS ul = [] and filter isPVal ul = []
shows ul = []

```

<proof>

lemma *BC-not-Nil*: $BC\ vl\ vl1 \implies vl = [] \implies vl1 = []$
<proof>

lemma *BC-OVal-True*:

assumes $BC\ (OVal\ True\ \#\ vl')\ vl1$

shows $\exists\ vl1'.\ BO\ vl'\ vl1' \wedge vl1 = OVal\ True\ \#\ vl1'$
<proof>

fun *corrFrom* :: $post \Rightarrow value\ list \Rightarrow bool$ **where**

$corrFrom\ pst\ [] = True$

$| corrFrom\ pst\ (PVal\ pstt\ \#\ vl) = corrFrom\ pstt\ vl$

$| corrFrom\ pst\ (PValS\ aid\ pstt\ \#\ vl) = (pst = pstt \wedge corrFrom\ pst\ vl)$

$| corrFrom\ pst\ (OVal\ b\ \#\ vl) = (corrFrom\ pst\ vl)$

abbreviation *corr* :: $value\ list \Rightarrow bool$ **where** $corr \equiv corrFrom\ emptyPost$

definition *B* **where**

$B\ vl\ vl1 \equiv BC\ vl\ vl1 \wedge corr\ vl1$

lemma *B-not-Nil*:

assumes $B\ vl\ vl1$ **and** $vl: vl = []$

shows $vl1 = []$

<proof>

sublocale *BD-Security-IO* **where**

$istate = istate$ **and** $step = step$ **and**

$\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$

<proof>

6.6.4 Issuer unwinding proof

lemma *reach-PublicV-imples-FriendV[simp]*:

assumes $reach\ s$

and $vis\ s\ pid \neq PublicV$

shows $vis\ s\ pid = FriendV$

<proof>

lemma *eqButPID-step- γ -out*:

assumes $ss1: eqButPID\ s\ s1$

and $step: step\ s\ a = (ou, s')$ **and** $step1: step\ s1\ a = (ou1, s1')$

and $op: \neg open\ s$

and sT : $reachNT\ s$ **and** $s1$: $reach\ s1$
and γ : $\gamma\ (Trans\ s\ a\ ou\ s')$
shows $(\exists\ uid\ p\ aid\ pid.\ a = COMact\ (comSendPost\ uid\ p\ aid\ pid) \wedge outPurge\ ou = outPurge\ ou1) \vee ou = ou1$
 $\langle proof \rangle$

lemma $eqButPID$ -step-eq:
assumes $ss1$: $eqButPID\ s\ s1$
and a : $a = Uact\ (uPost\ uid\ p\ PID\ pst)\ ou = outOK$
and $step$: $step\ s\ a = (ou,\ s')$ **and** $step1$: $step\ s1\ a = (ou',\ s1')$
shows $s' = s1'$
 $\langle proof \rangle$

definition $\Delta0$:: $state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta0\ s\ vl\ s1\ vl1 \equiv$
 $\neg\ PID \in \in\ postIDs\ s \wedge$
 $s = s1 \wedge BC\ vl\ vl1 \wedge$
 $corr\ vl1$

definition $\Delta1$:: $state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta1\ s\ vl\ s1\ vl1 \equiv$
 $PID \in \in\ postIDs\ s \wedge$
 $list\ all\ (Not\ o\ isOVal)\ vl \wedge list\ all\ (Not\ o\ isOVal)\ vl1 \wedge$
 $map\ tgtAPI\ (filter\ isPValS\ vl) = map\ tgtAPI\ (filter\ isPValS\ vl1) \wedge$
 $(vl = [] \longrightarrow vl1 = []) \wedge$
 $eqButPID\ s\ s1 \wedge \neg\ open\ s \wedge$
 $corrFrom\ (post\ s1\ PID)\ vl1$

definition $\Delta11$:: $state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta11\ s\ vl\ s1\ vl1 \equiv$
 $PID \in \in\ postIDs\ s \wedge$
 $vl = [] \wedge list\ all\ isPVal\ vl1 \wedge$
 $eqButPID\ s\ s1 \wedge \neg\ open\ s \wedge$
 $corrFrom\ (post\ s1\ PID)\ vl1$

definition $\Delta2$:: $state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta2\ s\ vl\ s1\ vl1 \equiv$
 $PID \in \in\ postIDs\ s \wedge$
 $list\ all\ (Not\ o\ isOVal)\ vl \wedge$
 $vl = vl1 \wedge$
 $s = s1 \wedge open\ s \wedge$
 $corrFrom\ (post\ s1\ PID)\ vl1$

definition $\Delta31$:: $state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta31\ s\ vl\ s1\ vl1 \equiv$
 $PID \in \in\ postIDs\ s \wedge$
 $(\exists\ ul\ ul1\ sul\ vll\ vll1.$

$BO\ vll\ vll1 \wedge$
 $list\text{-}all\ (Not\ o\ isOVal)\ ul \wedge list\text{-}all\ (Not\ o\ isOVal)\ ul1 \wedge$
 $map\ tgtAPI\ (filter\ isPValS\ ul) = map\ tgtAPI\ (filter\ isPValS\ ul1) \wedge$
 $ul \neq [] \wedge ul1 \neq [] \wedge$
 $isPVal\ (last\ ul) \wedge last\ ul = last\ ul1 \wedge$
 $list\text{-}all\ isPValS\ sul \wedge$
 $vl = ul @ sul @ OVal\ True \# vll \wedge vl1 = ul1 @ sul @ OVal\ True \# vll1) \wedge$
 $eqButPID\ s\ s1 \wedge \neg\ open\ s \wedge$
 $corrFrom\ (post\ s1\ PID)\ vl1$

definition $\Delta32 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**

$\Delta32\ s\ vl\ s1\ vl1 \equiv$

$PID \in \in postIDs\ s \wedge$

$(\exists\ sul\ vll\ vll1.$

$BO\ vll\ vll1 \wedge$

$list\text{-}all\ isPValS\ sul \wedge$

$vl = sul @ OVal\ True \# vll \wedge vl1 = sul @ OVal\ True \# vll1) \wedge$

$s = s1 \wedge \neg\ open\ s \wedge$

$corrFrom\ (post\ s1\ PID)\ vl1$

definition $\Delta4 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**

$\Delta4\ s\ vl\ s1\ vl1 \equiv$

$PID \in \in postIDs\ s \wedge$

$(\exists\ ul\ vll\ vll1.$

$BC\ vll\ vll1 \wedge$

$list\text{-}all\ (Not\ o\ isOVal)\ ul \wedge$

$vl = ul @ OVal\ False \# vll \wedge vl1 = ul @ OVal\ False \# vll1) \wedge$

$s = s1 \wedge open\ s \wedge$

$corrFrom\ (post\ s1\ PID)\ vl1$

lemma *istate- $\Delta0$* :

assumes $B: B\ vl\ vl1$

shows $\Delta0\ istate\ vl\ istate\ vl1$

<proof>

lemma *list-all-filter[simp]*:

assumes $list\text{-}all\ PP\ xs$

shows $filter\ PP\ xs = xs$

<proof>

lemma *unwind-cont- $\Delta0$* : $unwind\text{-}cont\ \Delta0\ \{\Delta0, \Delta1, \Delta2, \Delta31, \Delta32, \Delta4\}$

<proof>

lemma *unwind-cont- $\Delta1$* : $unwind\text{-}cont\ \Delta1\ \{\Delta1, \Delta11\}$

<proof>

lemma *unwind-cont- $\Delta11$* : $unwind\text{-}cont\ \Delta11\ \{\Delta11\}$

<proof>

lemma *unwind-cont- Δ_{31}* : *unwind-cont* Δ_{31} $\{\Delta_{31}, \Delta_{32}\}$
<proof>

lemma *unwind-cont- Δ_{32}* : *unwind-cont* Δ_{32} $\{\Delta_2, \Delta_{32}, \Delta_4\}$
<proof>

lemma *unwind-cont- Δ_2* : *unwind-cont* Δ_2 $\{\Delta_2\}$
<proof>

lemma *unwind-cont- Δ_4* : *unwind-cont* Δ_4 $\{\Delta_1, \Delta_{31}, \Delta_{32}, \Delta_4\}$
<proof>

definition *Gr* where

Gr =
{
 (Δ_0 , $\{\Delta_0, \Delta_1, \Delta_2, \Delta_{31}, \Delta_{32}, \Delta_4\}$),
 (Δ_1 , $\{\Delta_1, \Delta_{11}\}$),
 (Δ_{11} , $\{\Delta_{11}\}$),
 (Δ_2 , $\{\Delta_2\}$),
 (Δ_{31} , $\{\Delta_{31}, \Delta_{32}\}$),
 (Δ_{32} , $\{\Delta_2, \Delta_{32}, \Delta_4\}$),
 (Δ_4 , $\{\Delta_1, \Delta_{31}, \Delta_{32}, \Delta_4\}$)
}

theorem *secure*: *secure*
<proof>

end

end

theory *Independent-Post-Observation-Setup-RECEIVER*

imports

../Safety-Properties

../Post-Observation-Setup-RECEIVER

begin

6.6.5 Receiver observation setup

locale *Strong-ObservationSetup-RECEIVER* = *Fixed-UIDs* + *Fixed-PID* + *Fixed-AID*

begin

fun γ :: (*state, act, out*) *trans* \Rightarrow *bool* **where**

γ (*Trans* - *a* - -) \longleftrightarrow
 $(\exists \text{ uid. userOfA } a = \text{Some uid} \wedge \text{uid} \in \text{UIDs})$
 \vee
 — Communication actions are considered to be observable in order to make the security properties compositional
 $(\exists \text{ ca. } a = \text{COMAct ca})$
 \vee
 — The following actions are added to strengthen the observers in order to show that all posts *other than PID of AID* are completely independent of that post; the confidentiality of the latter is protected even if the observers can see all updates to other posts (and actions contributing to the declassification triggers of those posts).
 $(\exists \text{ uid } p \text{ pid } \text{pst. } a = \text{Uact (uPost uid } p \text{ pid } \text{pst)})$
 \vee
 $(\exists \text{ uid } p. a = \text{Sact (sSys uid } p))$
 \vee
 $(\exists \text{ uid } p \text{ uid}' \text{ p}'. a = \text{Cact (cUser uid } p \text{ uid}' \text{ p}'))$
 \vee
 $(\exists \text{ uid } p \text{ pid. } a = \text{Cact (cPost uid } p \text{ pid)})$
 \vee
 $(\exists \text{ uid } p \text{ uid}'. a = \text{Cact (cFriend uid } p \text{ uid}'))$
 \vee
 $(\exists \text{ uid } p \text{ uid}'. a = \text{Dact (dFriend uid } p \text{ uid}'))$
 \vee
 $(\exists \text{ uid } p \text{ pid } v. a = \text{Uact (uVisPost uid } p \text{ pid } v))$

fun *sPurge* :: *sActt* \Rightarrow *sActt* **where**
sPurge (*sSys uid pwd*) = *sSys uid emptyPass*

fun *comPurge* :: *comActt* \Rightarrow *comActt* **where**
comPurge (*comSendServerReq uID p aID reqInfo*) = *comSendServerReq uID emptyPass aID reqInfo*
| comPurge (*comConnectClient uID p aID sp*) = *comConnectClient uID emptyPass aID sp*

| comPurge (*comReceivePost aID sp pID pst uID vs*) =
 (let *pst'* = (if *aID* = *AID* \wedge *pID* = *PID* then *emptyPost* else *pst*)
 in *comReceivePost aID sp pID pst' uID vs*)

| comPurge (*comSendPost uID p aID pID*) = *comSendPost uID emptyPass aID pID*
| comPurge (*comSendCreateOFriend uID p aID uID'*) = *comSendCreateOFriend uID emptyPass aID uID'*
| comPurge (*comSendDeleteOFriend uID p aID uID'*) = *comSendDeleteOFriend uID emptyPass aID uID'*
| comPurge *ca* = *ca*

fun $g :: (state, act, out)trans \Rightarrow obs$ **where**
 $g (Trans - (Sact\ sa)\ ou\ -) = (Sact\ (sPurge\ sa),\ ou)$
 $|g (Trans - (COMact\ ca)\ ou\ -) = (COMact\ (comPurge\ ca),\ ou)$
 $|g (Trans - a\ ou\ -) = (a, ou)$

lemma *comPurge-simps*:

$comPurge\ ca = comSendServerReq\ uID\ p\ aID\ reqInfo \longleftrightarrow (\exists p'. ca = comSendServerReq\ uID\ p'\ aID\ reqInfo \wedge p = emptyPass)$
 $comPurge\ ca = comReceiveClientReq\ aID\ reqInfo \longleftrightarrow ca = comReceiveClientReq\ aID\ reqInfo$
 $comPurge\ ca = comConnectClient\ uID\ p\ aID\ sp \longleftrightarrow (\exists p'. ca = comConnectClient\ uID\ p'\ aID\ sp \wedge p = emptyPass)$
 $comPurge\ ca = comConnectServer\ aID\ sp \longleftrightarrow ca = comConnectServer\ aID\ sp$
 $comPurge\ ca = comReceivePost\ aID\ sp\ pID\ pst'\ uID\ v \longleftrightarrow (\exists pst. ca = comReceivePost\ aID\ sp\ pID\ pst\ uID\ v \wedge pst' = (if\ pID = PID \wedge aID = AID\ then\ emptyPost\ else\ pst))$
 $comPurge\ ca = comSendPost\ uID\ p\ aID\ pID \longleftrightarrow (\exists p'. ca = comSendPost\ uID\ p'\ aID\ pID \wedge p = emptyPass)$
 $comPurge\ ca = comSendCreateOFriend\ uID\ p\ aID\ uID' \longleftrightarrow (\exists p'. ca = comSendCreateOFriend\ uID\ p'\ aID\ uID' \wedge p = emptyPass)$
 $comPurge\ ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uID'$
 $comPurge\ ca = comSendDeleteOFriend\ uID\ p\ aID\ uID' \longleftrightarrow (\exists p'. ca = comSendDeleteOFriend\ uID\ p'\ aID\ uID' \wedge p = emptyPass)$
 $comPurge\ ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'$
 $\langle proof \rangle$

lemma *g-simps*:

$g (Trans\ s\ a\ ou\ s') = (COMact\ (comSendServerReq\ uID\ p\ aID\ reqInfo),\ ou')$
 $\longleftrightarrow (\exists p'. a = COMact\ (comSendServerReq\ uID\ p'\ aID\ reqInfo) \wedge p = emptyPass \wedge ou = ou')$
 $g (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveClientReq\ aID\ reqInfo),\ ou')$
 $\longleftrightarrow a = COMact\ (comReceiveClientReq\ aID\ reqInfo) \wedge ou = ou'$
 $g (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectClient\ uID\ p\ aID\ sp),\ ou')$
 $\longleftrightarrow (\exists p'. a = COMact\ (comConnectClient\ uID\ p'\ aID\ sp) \wedge p = emptyPass \wedge ou = ou')$
 $g (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectServer\ aID\ sp),\ ou')$
 $\longleftrightarrow a = COMact\ (comConnectServer\ aID\ sp) \wedge ou = ou'$
 $g (Trans\ s\ a\ ou\ s') = (COMact\ (comReceivePost\ aID\ sp\ pID\ pst'\ uID\ v),\ ou')$
 $\longleftrightarrow (\exists pst. a = COMact\ (comReceivePost\ aID\ sp\ pID\ pst\ uID\ v) \wedge pst' = (if\ pID = PID \wedge aID = AID\ then\ emptyPost\ else\ pst) \wedge ou = ou')$
 $g (Trans\ s\ a\ ou\ s') = (COMact\ (comSendPost\ uID\ p\ aID\ nID),\ O-sendPost\ (aid,\ sp,\ pid,\ pst,\ own,\ v))$
 $\longleftrightarrow (\exists p'. a = COMact\ (comSendPost\ uID\ p'\ aID\ nID) \wedge p = emptyPass \wedge ou = O-sendPost\ (aid,\ sp,\ pid,\ pst,\ own,\ v))$
 $g (Trans\ s\ a\ ou\ s') = (COMact\ (comSendCreateOFriend\ uID\ p\ aID\ uID'),\ ou')$

```

 $\longleftrightarrow (\exists p'. a = (COMact (comSendCreateOFriend uID p' aID uID')) \wedge p = emptyPass \wedge ou = ou')$ 
  g (Trans s a ou s') = (COMact (comReceiveCreateOFriend aID cp uID uID'), ou')
 $\longleftrightarrow a = COMact (comReceiveCreateOFriend aID cp uID uID') \wedge ou = ou'$ 
  g (Trans s a ou s') = (COMact (comSendDeleteOFriend uID p aID uID'), ou')
 $\longleftrightarrow (\exists p'. a = COMact (comSendDeleteOFriend uID p' aID uID') \wedge p = emptyPass \wedge ou = ou')$ 
  g (Trans s a ou s') = (COMact (comReceiveDeleteOFriend aID cp uID uID'), ou')
 $\longleftrightarrow a = COMact (comReceiveDeleteOFriend aID cp uID uID') \wedge ou = ou'$ 
  <proof>

```

end

end

theory *Independent-Post-Value-Setup-RECEIVER*

imports

../Safety-Properties

Independent-Post-Observation-Setup-RECEIVER

../Post-Unwinding-Helper-RECEIVER

begin

6.6.6 Receiver value setup

locale *Post-RECEIVER* = *Strong-ObservationSetup-RECEIVER*

begin

datatype *value* = *PValR post*

fun $\varphi :: (state, act, out) trans \Rightarrow bool$ **where**

$\varphi (Trans - (COMact (comReceivePost aid sp pid pst uid vs)) ou -) =$
 $(aid = AID \wedge pid = PID \wedge ou = outOK)$

|
 $\varphi (Trans s - - s') = False$

lemma φ -def2:

shows

$\varphi (Trans s a ou s') \longleftrightarrow$
 $(\exists uid p pst vs. a = COMact (comReceivePost AID p PID pst uid vs) \wedge ou = outOK)$

<proof>

lemma *comReceivePost-out*:

assumes 1: $step\ s\ a = (ou, s')$ **and** $a = COMact\ (comReceivePost\ AID\ p\ PID\ pst\ uid\ vs)$ **and** 2: $ou = outOK$
shows $p = serverPass\ s\ AID$
 $\langle proof \rangle$

lemma φ -def3:

assumes $step\ s\ a = (ou, s')$

shows

$\varphi\ (Trans\ s\ a\ ou\ s') \longleftrightarrow$

$(\exists\ uid\ pst\ vs.\ a = COMact\ (comReceivePost\ AID\ (serverPass\ s\ AID)\ PID\ pst\ uid\ vs) \wedge ou = outOK)$

$\langle proof \rangle$

lemma φ -cases:

assumes $\varphi\ (Trans\ s\ a\ ou\ s')$

and $step\ s\ a = (ou, s')$

and $reach\ s$

obtains

$(Recv)\ uid\ sp\ aID\ pID\ pst\ vs$ **where** $a = COMact\ (comReceivePost\ aID\ sp\ pID\ pst\ uid\ vs)\ ou = outOK$

$sp = serverPass\ s\ AID$

$aID = AID\ pID = PID$

$\langle proof \rangle$

fun $f :: (state, act, out)\ trans \Rightarrow value$ **where**

$f\ (Trans\ s\ (COMact\ (comReceivePost\ aid\ sp\ pid\ pst\ uid\ vs)) - s') =$

$(if\ aid = AID \wedge pid = PID\ then\ PValR\ pst\ else\ undefined)$

$|$

$f\ (Trans\ s\ - -\ s') = undefined$

sublocale *Receiver-State-Equivalence-Up-To-PID* $\langle proof \rangle$

lemma *eqButPID-step- φ -imp*:

assumes $ss1: eqButPID\ s\ s1$

and $step: step\ s\ a = (ou, s')$ **and** $step1: step\ s1\ a = (ou1, s1')$

and $\varphi: \varphi\ (Trans\ s\ a\ ou\ s')$

shows $\varphi\ (Trans\ s1\ a\ ou1\ s1')$

$\langle proof \rangle$

lemma *eqButPID-step- φ* :

assumes $s's1': eqButPID\ s\ s1$

and $step: step\ s\ a = (ou, s')$ **and** $step1: step\ s1\ a = (ou1, s1')$

shows $\varphi\ (Trans\ s\ a\ ou\ s') = \varphi\ (Trans\ s1\ a\ ou1\ s1')$

$\langle proof \rangle$

end

```

end
theory Independent-Post-RECEIVER
  imports
    Independent-Post-Observation-Setup-RECEIVER
    Independent-Post-Value-Setup-RECEIVER
    Bounded-Deducibility-Security.Compositional-Reasoning
begin

```

6.6.7 Receiver declassification bound

```

context Post-RECEIVER
begin

```

```

fun T :: (state,act,out) trans  $\Rightarrow$  bool where
T (Trans s a ou s')  $\longleftrightarrow$ 
  ( $\exists$  uid  $\in$  UIDs.
    uid  $\in\in$  userIDs s'  $\wedge$  PID  $\in\in$  outerPostIDs s' AID  $\wedge$ 
    (uid = admin s'  $\vee$ 
      (AID,outerOwner s' AID PID)  $\in\in$  recvOuterFriendIDs s' uid  $\vee$ 
        outerVis s' AID PID = PublicV))

```

```

definition B :: value list  $\Rightarrow$  value list  $\Rightarrow$  bool where
B vl vl1  $\equiv$  length vl = length vl1

```

```

sublocale BD-Security-IO where
  istate = istate and step = step and
   $\varphi = \varphi$  and  $f = f$  and  $\gamma = \gamma$  and  $g = g$  and  $T = T$  and  $B = B$ 
   $\langle$ proof $\rangle$ 

```

6.6.8 Receiver unwinding proof

```

lemma reach-PublicV-imples-FriendV[simp]:

```

```

assumes reach s
and vis s pID  $\neq$  PublicV
shows vis s pID = FriendV
   $\langle$ proof $\rangle$ 

```

```

lemma reachNT-state:

```

```

assumes reachNT s
shows
   $\neg$  ( $\exists$  uid  $\in$  UIDs.
    uid  $\in\in$  userIDs s  $\wedge$  PID  $\in\in$  outerPostIDs s AID  $\wedge$ 
    (uid = admin s  $\vee$ 
      (AID,outerOwner s AID PID)  $\in\in$  recvOuterFriendIDs s uid  $\vee$ 
        outerVis s AID PID = PublicV))
   $\langle$ proof $\rangle$ 

```

lemma *eqButPID-step- γ -out*:
assumes *ss1: eqButPID s s1*
and *step: step s a = (ou,s')* **and** *step1: step s1 a = (ou1,s1')*
and *sT: reachNT s* **and** *T: $\neg T$ (Trans s a ou s')*
and *s1: reach s1*
and *γ : γ (Trans s a ou s')*
shows *ou = ou1*
\langle proof \rangle

definition $\Delta 0 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta 0\ s\ vl\ s1\ vl1 \equiv$
 $\neg AID \in \in serverApiIDs\ s \wedge$
 $s = s1 \wedge$
 $length\ vl = length\ vl1$

definition $\Delta 1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta 1\ s\ vl\ s1\ vl1 \equiv$
 $AID \in \in serverApiIDs\ s \wedge$
 $eqButPID\ s\ s1 \wedge$
 $length\ vl = length\ vl1$

lemma *istate- $\Delta 0$* :
assumes *B: B vl vl1*
shows $\Delta 0\ istate\ vl\ istate\ vl1$
\langle proof \rangle

lemma *unwind-cont- $\Delta 0$* : *unwind-cont $\Delta 0$ { $\Delta 0, \Delta 1$ }*
\langle proof \rangle

lemma *unwind-cont- $\Delta 1$* : *unwind-cont $\Delta 1$ { $\Delta 1$ }*
\langle proof \rangle

definition *Gr* **where**
 $Gr =$
 $\{$
 $(\Delta 0, \{\Delta 0, \Delta 1\}),$
 $(\Delta 1, \{\Delta 1\})$
 $\}$

theorem *Post-secure: secure*
\langle proof \rangle

end

end

theory *Independent-DYNAMIC-Post-Network*

```

imports
  Independent-DYNAMIC-Post-ISSUER
  Independent-Post-RECEIVER
  ../../API-Network
  BD-Security-Compositional.Composing-Security-Network
begin

```

6.6.9 Confidentiality for the N-ary composition

```

type-synonym ttrans = (state, act, out) trans
type-synonym obs = Post-Observation-Setup-ISSUER.obs
type-synonym value = Post.value + Post-RECEIVER.value

```

lemma *value-cases*:

```

fixes v :: value
obtains (PVal) pst where v = Inl (Post.PVal pst)
  | (PValS) aid pst where v = Inl (Post.PValS aid pst)
  | (OVal) ov where v = Inl (Post.OVal ov)
  | (PValR) pst where v = Inr (Post-RECEIVER.PValR pst)
⟨proof⟩

```

```

locale Post-Network = Network
+ fixes UIDs :: apiID ⇒ userID set
  and AID :: apiID and PID :: postID
  assumes AID-in-AIDs: AID ∈ AIDs
begin

```

```

sublocale Iss: Post UIDs AID PID ⟨proof⟩

```

```

abbreviation  $\varphi$  :: apiID ⇒ (state, act, out) trans ⇒ bool
where  $\varphi$  aid trn ≡ (if aid = AID then Iss. $\varphi$  trn else Post-RECEIVER. $\varphi$  PID AID trn)

```

```

abbreviation f :: apiID ⇒ (state, act, out) trans ⇒ value
where f aid trn ≡ (if aid = AID then Inl (Iss.f trn) else Inr (Post-RECEIVER.f PID AID trn))

```

```

abbreviation  $\gamma$  :: apiID ⇒ (state, act, out) trans ⇒ bool
where  $\gamma$  aid trn ≡ (if aid = AID then Iss. $\gamma$  trn else Strong-ObservationSetup-RECEIVER. $\gamma$  (UIDs aid) trn)

```

```

abbreviation g :: apiID ⇒ (state, act, out) trans ⇒ obs
where g aid trn ≡ (if aid = AID then Iss.g trn else Strong-ObservationSetup-RECEIVER.g PID AID trn)

```

```

abbreviation T :: apiID ⇒ (state, act, out) trans ⇒ bool
where T aid trn ≡ (if aid = AID then Iss.T trn else Post-RECEIVER.T (UIDs aid) PID AID trn)

```

abbreviation $B :: \text{apiID} \Rightarrow \text{value list} \Rightarrow \text{value list} \Rightarrow \text{bool}$

where $B \text{ aid vl vl1} \equiv$

$(\text{if } \text{aid} = \text{AID} \text{ then } \text{list-all } \text{isl } \text{vl} \wedge \text{list-all } \text{isl } \text{vl1} \wedge \text{Iss.B } (\text{map } \text{projl } \text{vl}) (\text{map } \text{projl } \text{vl1}))$

$\text{else } \text{list-all } (\text{Not } o \text{ isl}) \text{ vl} \wedge \text{list-all } (\text{Not } o \text{ isl}) \text{ vl1} \wedge \text{Post-RECEIVER.B } (\text{map } \text{projr } \text{vl}) (\text{map } \text{projr } \text{vl1}))$

fun $\text{comOfV} :: \text{apiID} \Rightarrow \text{value} \Rightarrow \text{com}$ **where**

$\text{comOfV } \text{aid } (\text{Inl } (\text{Post.PValS } \text{aid}' \text{ pst})) = (\text{if } \text{aid}' \neq \text{aid} \text{ then } \text{Send} \text{ else } \text{Internal})$
| $\text{comOfV } \text{aid } (\text{Inl } (\text{Post.PVal } \text{pst})) = \text{Internal}$
| $\text{comOfV } \text{aid } (\text{Inl } (\text{Post.OVal } \text{ov})) = \text{Internal}$
| $\text{comOfV } \text{aid } (\text{Inr } v) = \text{Recv}$

fun $\text{tgtNodeOfV} :: \text{apiID} \Rightarrow \text{value} \Rightarrow \text{apiID}$ **where**

$\text{tgtNodeOfV } \text{aid } (\text{Inl } (\text{Post.PValS } \text{aid}' \text{ pst})) = \text{aid}'$
| $\text{tgtNodeOfV } \text{aid } (\text{Inl } (\text{Post.PVal } \text{pst})) = \text{undefined}$
| $\text{tgtNodeOfV } \text{aid } (\text{Inl } (\text{Post.OVal } \text{ov})) = \text{undefined}$
| $\text{tgtNodeOfV } \text{aid } (\text{Inr } v) = \text{AID}$

definition $\text{syncV} :: \text{apiID} \Rightarrow \text{value} \Rightarrow \text{apiID} \Rightarrow \text{value} \Rightarrow \text{bool}$ **where**

$\text{syncV } \text{aid1 } v1 \text{ aid2 } v2 =$
 $(\exists \text{pst. } \text{aid1} = \text{AID} \wedge v1 = \text{Inl } (\text{Post.PValS } \text{aid2 } \text{pst}) \wedge v2 = \text{Inr } (\text{Post-RECEIVER.PValR } \text{pst}))$

lemma syncVI : $\text{syncV } \text{AID } (\text{Inl } (\text{Post.PValS } \text{aid}' \text{ pst})) \text{ aid}' (\text{Inr } (\text{Post-RECEIVER.PValR } \text{pst}))$
 $\langle \text{proof} \rangle$

lemma syncVE :

assumes $\text{syncV } \text{aid1 } v1 \text{ aid2 } v2$

obtains pst **where** $\text{aid1} = \text{AID}$ $v1 = \text{Inl } (\text{Post.PValS } \text{aid2 } \text{pst})$ $v2 = \text{Inr } (\text{Post-RECEIVER.PValR } \text{pst})$
 $\langle \text{proof} \rangle$

fun getTgtV **where**

$\text{getTgtV } (\text{Inl } (\text{Post.PValS } \text{aid } \text{pst})) = \text{Inr } (\text{Post-RECEIVER.PValR } \text{pst})$
| $\text{getTgtV } v = v$

lemma comOfV-AID :

$\text{comOfV } \text{AID } v = \text{Send} \iff \text{isl } v \wedge \text{Iss.isPValS } (\text{projl } v) \wedge \text{Iss.tgtAPI } (\text{projl } v) \neq \text{AID}$

$\text{comOfV } \text{AID } v = \text{Recv} \iff \text{Not } (\text{isl } v)$
 $\langle \text{proof} \rangle$

lemmas $\varphi\text{-defs} = \text{Post-RECEIVER.}\varphi\text{-def2 } \text{Iss.}\varphi\text{-def3}$

sublocale Net : $\text{BD-Security-TS-Network-getTgtV}$

where $\text{istate} = \lambda\cdot. \text{istate}$ **and** $\text{validTrans} = \text{validTrans}$ **and** $\text{srcOf} = \lambda\cdot. \text{srcOf}$
and $\text{tgtOf} = \lambda\cdot. \text{tgtOf}$

and $nodes = AIDs$ **and** $comOf = comOf$ **and** $tgtNodeOf = tgtNodeOf$
and $sync = sync$ **and** $\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and**
 $B = B$
and $comOfV = comOfV$ **and** $tgtNodeOfV = tgtNodeOfV$ **and** $syncV = syncV$
and $comOfO = comOfO$ **and** $tgtNodeOfO = tgtNodeOfO$ **and** $syncO = syncO$
and $source = AID$ **and** $getTgtV = getTgtV$
 $\langle proof \rangle$

lemma *list-all-Not-isl-projectSrcV*: $list\text{-}all\ (Not\ o\ isl)\ (Net.\text{projectSrcV}\ aid\ vlSrc)$
 $\langle proof \rangle$

context
fixes $AID' :: apiID$
assumes $AID' : AID' \in AIDs - \{AID\}$
begin

interpretation *Recv*: *Post-RECEIVER UIDs* AID' PID AID $\langle proof \rangle$

lemma *Iss-BC-BO-tgtAPI*:
shows $(Iss.BC\ vl\ vl1 \longrightarrow map\ Iss.tgtAPI\ (filter\ Iss.isPValS\ vl) =$
 $map\ Iss.tgtAPI\ (filter\ Iss.isPValS\ vl1)) \wedge$
 $(Iss.BO\ vl\ vl1 \longrightarrow map\ Iss.tgtAPI\ (filter\ Iss.isPValS\ vl) =$
 $map\ Iss.tgtAPI\ (filter\ Iss.isPValS\ vl1))$
 $\langle proof \rangle$

lemma *Iss-B-Recv-B-aux*:
assumes $list\text{-}all\ isl\ vl$
and $list\text{-}all\ isl\ vl1$
and $map\ Iss.tgtAPI\ (filter\ Iss.isPValS\ (map\ projl\ vl)) =$
 $map\ Iss.tgtAPI\ (filter\ Iss.isPValS\ (map\ projl\ vl1))$
shows $length\ (map\ projr\ (Net.\text{projectSrcV}\ AID'\ vl)) = length\ (map\ projr\ (Net.\text{projectSrcV}\ AID'\ vl1))$
 $\langle proof \rangle$

lemma *Iss-B-Recv-B*:
assumes $B\ AID\ vl\ vl1$
shows $Recv.B\ (map\ projr\ (Net.\text{projectSrcV}\ AID'\ vl))\ (map\ projr\ (Net.\text{projectSrcV}\ AID'\ vl1))$
 $\langle proof \rangle$

end

lemma *map-projl-Inl*: $map\ (projl\ o\ Inl)\ vl = vl$
 $\langle proof \rangle$

lemma *these-map-Inl-projl*: $list\text{-}all\ isl\ vl \implies these\ (map\ (Some\ o\ Inl\ o\ projl)\ vl)$
 $=\ vl$
 $\langle proof \rangle$

lemma *map-projr-Inr*: $\text{map } (\text{projr } o \text{ Inr}) \text{ vl} = \text{vl}$
 ⟨proof⟩

lemma *these-map-Inr-projr*: $\text{list-all } (\text{Not } o \text{ isl}) \text{ vl} \implies \text{these } (\text{map } (\text{Some } o \text{ Inr } o \text{ projr}) \text{ vl}) = \text{vl}$
 ⟨proof⟩

sublocale *BD-Security-TS-Network-Preserve-Source-Security-getTgtV*
where $\text{istate} = \lambda\cdot. \text{istate}$ **and** $\text{validTrans} = \text{validTrans}$ **and** $\text{srcOf} = \lambda\cdot. \text{srcOf}$
and $\text{tgtOf} = \lambda\cdot. \text{tgtOf}$
and $\text{nodes} = \text{AIDs}$ **and** $\text{comOf} = \text{comOf}$ **and** $\text{tgtNodeOf} = \text{tgtNodeOf}$
and $\text{sync} = \text{sync}$ **and** $\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and**
 $B = B$
and $\text{comOfV} = \text{comOfV}$ **and** $\text{tgtNodeOfV} = \text{tgtNodeOfV}$ **and** $\text{syncV} = \text{syncV}$
and $\text{comOfO} = \text{comOfO}$ **and** $\text{tgtNodeOfO} = \text{tgtNodeOfO}$ **and** $\text{syncO} = \text{syncO}$
and $\text{source} = \text{AID}$ **and** $\text{getTgtV} = \text{getTgtV}$
 ⟨proof⟩

theorem *secure*: *secure*
 ⟨proof⟩

end

end

theory *Independent-Posts-Network*

imports

Independent-DYNAMIC-Post-Network

BD-Security-Compositional.Independent-Secrets

begin

6.6.10 Composition of confidentiality guarantees for different posts

We combine *two* confidentiality guarantees for two different posts in arbitrary nodes of the network.

For this purpose, we have strengthened the observation power of the security property for individual posts to make all transitions that update *other* posts observable, as well as all transitions that contribute to the state of the trigger (see the observation setup theories). This guarantees that the confidentiality of one post is independent of actions affecting other posts, which will allow us to combine security guarantees for different posts.

We now prove a few helper lemmas establishing that now the observable transitions indeed fully determine the state of the trigger.

fun *obsEffect* :: $\text{state} \Rightarrow \text{obs} \Rightarrow \text{state}$ **where**
 $\text{obsEffect } s \ (\text{Uact } (\text{uPost } \text{uid } p \ \text{pid } \ \text{pst}), \ \text{ou}) = \text{updatePost } s \ \text{uid } p \ \text{pid } \ \text{pst}$
 $\text{obsEffect } s \ (\text{Uact } (\text{uVisPost } \text{uid } p \ \text{pid } \ v), \ \text{ou}) = \text{updateVisPost } s \ \text{uid } p \ \text{pid } \ v$
 $\text{obsEffect } s \ (\text{Sact } (\text{sSys } \text{uid } p), \ \text{ou}) = \text{startSys } s \ \text{uid } p$
 $\text{obsEffect } s \ (\text{Cact } (\text{cUser } \text{uid } p \ \text{uid}' \ p'), \ \text{ou}) = \text{createUser } s \ \text{uid } p \ \text{uid}' \ p'$

$| \text{obsEffect } s \text{ (Cact (cPost uid p pid), ou) = createPost } s \text{ uid p pid}$
 $| \text{obsEffect } s \text{ (Cact (cFriend uid p uid'), ou) = createFriend } s \text{ uid p uid'}$
 $| \text{obsEffect } s \text{ (Dact (dFriend uid p uid'), ou) = deleteFriend } s \text{ uid p uid'}$
 $| \text{obsEffect } s \text{ (COMact (comSendPost uid p aid pid), ou) = snd (sendPost } s \text{ uid p aid pid)}$
 $| \text{obsEffect } s \text{ (COMact (comReceivePost aid p pid pst uid v), ou) = receivePost } s \text{ aid p pid pst uid v}$
 $| \text{obsEffect } s \text{ (COMact (comReceiveCreateOFriend aid p uid uid'), ou) = receiveCreateOFriend } s \text{ aid p uid uid'}$
 $| \text{obsEffect } s \text{ (COMact (comReceiveDeleteOFriend aid p uid uid'), ou) = receiveDeleteOFriend } s \text{ aid p uid uid'}$
 $| \text{obsEffect } s \text{ - = } s$

fun *obsStep* :: *state* \Rightarrow *obs* \Rightarrow *state* **where**
obsStep *s* (*a*,*ou*) = (if *ou* \neq *outErr* then *obsEffect* *s* (*a*,*ou*) else *s*)

fun *obsSteps* :: *state* \Rightarrow *obs list* \Rightarrow *state* **where**
obsSteps *s* *obsl* = *foldl obsStep s obsl*

definition *triggerEq* :: *state* \Rightarrow *state* \Rightarrow *bool* **where**
triggerEq *s* *s'* \longleftrightarrow *userIDs* *s* = *userIDs* *s'* \wedge *postIDs* *s* = *postIDs* *s'* \wedge *admin* *s* = *admin* *s'* \wedge
 $\qquad \qquad \qquad$ *owner* *s* = *owner* *s'* \wedge *friendIDs* *s* = *friendIDs* *s'* \wedge *vis* *s* = *vis* *s'*
 \wedge
 $\qquad \qquad \qquad$ *outerPostIDs* *s* = *outerPostIDs* *s'* \wedge *outerOwner* *s* = *outerOwner* *s'* \wedge
 $\qquad \qquad \qquad$ *recvOuterFriendIDs* *s* = *recvOuterFriendIDs* *s'* \wedge *outerVis* *s* = *outerVis* *s'*

lemma *triggerEq-refl[simp]*: *triggerEq* *s* *s*
and *triggerEq-sym*: *triggerEq* *s* *s'* \Longrightarrow *triggerEq* *s'* *s*
and *triggerEq-trans*: *triggerEq* *s* *s'* \Longrightarrow *triggerEq* *s'* *s''* \Longrightarrow *triggerEq* *s* *s''*
 $\langle \text{proof} \rangle$

unbundle *no relcomp-syntax*

context *Post*
begin

lemma [*simp*]: *outOK* = *outPurge* *ou* \longleftrightarrow *ou* = *outOK* $\langle \text{proof} \rangle$
lemma [*simp*]: *sPurge* *sa* = *sSys* (*sUserOfA* *sa*) *emptyPass* $\langle \text{proof} \rangle$
lemma *sStep-unfold*: *sStep* *s* *sa* = (if *userIDs* *s* = []
 $\qquad \qquad \qquad$ then (case *sa* of *sSys* *uid* *p* \Rightarrow (*outOK*, *startSys* *s* *uid* *p*))
 $\qquad \qquad \qquad$ else (*outErr*, *s*))
 $\langle \text{proof} \rangle$

lemma *triggerEq-open*:
assumes *triggerEq* *s* *s'*
shows *open* *s* \longleftrightarrow *open* *s'*

<proof>

lemma *triggerEq-not- γ* :

assumes *validTrans* (*Trans s a ou s'*) **and** $\neg\gamma$ (*Trans s a ou s'*)

shows *triggerEq s s'*

<proof>

lemma *triggerEq-obsStep*:

assumes *validTrans* (*Trans s a ou s'*) **and** γ (*Trans s a ou s'*) **and** *triggerEq s s1*

shows *triggerEq s' (obsStep s1 (g (Trans s a ou s')))*

<proof>

lemma *triggerEq-obsSteps*:

assumes *validFrom s tr* **and** *triggerEq s s'*

shows *triggerEq (tgtOfTrFrom s tr) (obsSteps s' (O tr))*

<proof>

end

context *Post-RECEIVER*

begin

lemma *sPurge-simp[simp]*: *sPurge sa = sSys (sUserOfA sa) emptyPass* *<proof>*

definition *T-state s' \equiv*

$(\exists$ *uid* \in *UIDs*.

uid $\in\in$ *userIDs s' \wedge PID $\in\in$ outerPostIDs s' AID \wedge*

(uid = admin s' \vee

(AID,outerOwner s' AID PID) $\in\in$ recvOuterFriendIDs s' uid \vee

outerVis s' AID PID = PublicV))

lemma *T-T-state*: *T trn \longleftrightarrow T-state (tgtOf trn)*

<proof>

lemma *triggerEq-T*:

assumes *triggerEq s s'*

shows *T-state s \longleftrightarrow T-state s'*

<proof>

lemma *never-T-not-T-state*:

assumes *validFrom s tr* **and** *never T tr* **and** \neg *T-state s*

shows \neg *T-state (tgtOfTrFrom s tr)*

<proof>

lemma *triggerEq-not- γ* :

assumes *validTrans* (*Trans s a ou s'*) **and** $\neg\gamma$ (*Trans s a ou s'*)

shows *triggerEq s s'*

<proof>

lemma *triggerEq-obsStep*:
assumes *validTrans* (*Trans s a ou s'*) **and** γ (*Trans s a ou s'*) **and** *triggerEq s s1*
shows *triggerEq s' (obsStep s1 (g (Trans s a ou s')))*
 \langle *proof* \rangle

lemma *triggerEq-obsSteps*:
assumes *validFrom s tr* **and** *triggerEq s s'*
shows *triggerEq (tgtOfTrFrom s tr) (obsSteps s' (O tr))*
 \langle *proof* \rangle

end

context *Post-Network*
begin

fun *nObsStep* :: (*apiID* \Rightarrow *state*) \Rightarrow (*apiID*, *act* \times *out*) *nobs* \Rightarrow (*apiID* \Rightarrow *state*)
where
nObsStep s (LObs aid obs) = s(aid := obsStep (s aid) obs)
 $|$ *nObsStep s (CObs aid1 obs1 aid2 obs2) = s(aid1 := obsStep (s aid1) obs1, aid2 := obsStep (s aid2) obs2)*

fun *nObsSteps* :: (*apiID* \Rightarrow *state*) \Rightarrow (*apiID*, *act* \times *out*) *nobs list* \Rightarrow (*apiID* \Rightarrow *state*) **where**
nObsSteps s obsl = foldl nObsStep s obsl

definition *nTriggerEq* :: (*apiID* \Rightarrow *state*) \Rightarrow (*apiID* \Rightarrow *state*) \Rightarrow *bool* **where**
*nTriggerEq s s' \longleftrightarrow (\forall aid. *triggerEq (s aid) (s' aid)*)*

lemma *nTriggerEq-refl[simp]*: *nTriggerEq s s*
and *nTriggerEq-sym*: *nTriggerEq s s' \Longrightarrow nTriggerEq s' s*
and *nTriggerEq-trans*: *nTriggerEq s s' \Longrightarrow nTriggerEq s' s'' \Longrightarrow nTriggerEq s s''*
 \langle *proof* \rangle

lemma *nTriggerEq-open*:
assumes *nTriggerEq s s'*
shows \forall aid. *Iss.open (s aid) \longleftrightarrow Iss.open (s' aid)*
 \langle *proof* \rangle

lemma *nTriggerEq-not- γ* :
assumes *nValidTrans trn* **and** \neg *Net.n γ trn*
shows *nTriggerEq (nSrcOf trn) (nTgtOf trn)*
 \langle *proof* \rangle

lemma *nTriggerEq-obsStep*:
assumes *nValidTrans trn* **and** *Net.n γ trn* **and** *nTriggerEq (nSrcOf trn) s1*
shows *nTriggerEq (nTgtOf trn) (nObsStep s1 (Net.ng trn))*
 \langle *proof* \rangle

lemma *triggerEq-obsSteps*:
assumes *validFrom s tr* **and** *nTriggerEq s s'*
shows *nTriggerEq (nTgtOfTrFrom s tr) (nObsSteps s' (O tr))*
<proof>

lemma *O-eq-nTriggerEq*:
assumes *O: O tr = O tr'* **and** *tr: validFrom s (tr ## trn)* **and** *tr': validFrom s' (tr' ## trn')*
and $\gamma: \text{Net.n}\gamma \text{trn}$ **and** $\gamma': \text{Net.n}\gamma \text{trn}'$ **and** $g: \text{Net.ng trn} = \text{Net.ng trn}'$
and *s-s': nTriggerEq s s'*
shows *nTriggerEq (nSrcOf trn) (nSrcOf trn')* **and** *nTriggerEq (nTgtOf trn) (nTgtOf trn')*
<proof>

end

We are now ready to combine two confidentiality properties for different posts in different nodes.

locale *Posts-Network* =
Post1: Post-Network AIDs UIDs AID1 PID1
+ *Post2: Post-Network AIDs UIDs AID2 PID2*
for *AIDs :: apiID set*
and *UIDs :: apiID \Rightarrow userID set*
and *AID1 :: apiID and AID2 :: apiID*
and *PID1 :: postID and PID2 :: postID*
+
assumes *AID1-neq-AID2: AID1 \neq AID2*
begin

The combined observations consist of the local actions of observing users and their outputs, as usual. We do not consider communication actions here for simplicity, because this would require us to combine the *purgings* of observations of the two properties. This is straightforward, but tedious.

fun *n γ* :: *(apiID, state, (state, act, out) trans) ntrans \Rightarrow bool* **where**
n γ (LTrans s aid (Trans - a -)) = (\exists uid. userOfA a = Some uid \wedge uid \in UIDs aid \wedge (\neg isCOMact a))
| *n γ (CTrans s aid1 trn1 aid2 trn2) = False*

fun *g* :: *(state,act,out) trans \Rightarrow obs* **where**
g (Trans - (Sact sa) ou -) = (Sact (Post1.Iss.sPurge sa), ou)
| *g (Trans - a ou -) = (a,ou)*

fun *ng* :: *(apiID, state, (state, act, out) trans) ntrans \Rightarrow (apiID, act \times out) nob*
where
ng (LTrans s aid trn) = LObs aid (g trn)
| *ng (CTrans s aid1 trn1 aid2 trn2) = undefined*

abbreviation *validSystemTrace* \equiv *Post1.validFrom* (λ -. *istate*)

We now instantiate the generic technique for combining security properties with independent secret sources.

```
sublocale BD-Security-TS-Two-Secrets  $\lambda$ -. istate Post1.nValidTrans Post1.nSrcOf
Post1.nTgtOf
  Post1.Net.n $\varphi$  Post1.nf' Post1.Net.n $\gamma$  Post1.Net.n $\eta$  Post1.Net.nT Post1.B AID1
  Post2.Net.n $\varphi$  Post2.nf' Post2.Net.n $\gamma$  Post2.Net.n $\eta$  Post2.Net.nT Post2.B AID2
  n $\gamma$  n $\eta$ 
   $\langle$ proof $\rangle$ 
```

```
theorem two-posts-secure:
  secure
   $\langle$ proof $\rangle$ 
```

end

end

theory *Post-All*

imports

Post-COMPOSE2

Post-Network

DYNAMIC-Post-COMPOSE2

DYNAMIC-Post-Network

Independent-Posts/Independent-Posts-Network

begin

end

theory *Friend-Intro*

imports *../Safety-Properties*

begin

7 Friendship status confidentiality

We verify the following property:

Given a coalition consisting of groups of users *UIDs* *j* from multiple nodes *j* and given two users *UID1* and *UID2* at some node *i* who are not in these groups,

the coalition cannot learn anything about the changes in the status of friendship between *UID1* and *UID2*

beyond what everybody knows, namely that

- there is no friendship between them before those users have been created, and
- the updates form an alternating sequence of friending and unfriending,

and beyond those updates performed while or last before a user in the group $UIDs$ i is friends with $UID1$ or $UID2$.

The approach to proving this is similar to that for post confidentiality (explained in the introduction of the post confidentiality section 6), but conceptually simpler since here secret information is not communicated between different nodes (so we don't need to distinguish between an issuer node and the other, receiver nodes).

Moreover, here we do not consider static versions of the bounds, but go directly for the dynamic ones. Also, we prove directly the BD security for a network of n nodes, omitting the case of two nodes.

Note that, unlike for post confidentiality, here remote friendship plays no role in the statement of the property. This is because, in CoSMedis, the listing of a user's friends is only available to local (same-node) friends of that user, and not to the remote (outer) friends.

```

end
theory Friend-Observation-Setup
  imports Friend-Intro
begin

```

7.1 Observation setup

```

type-synonym obs = act * out

```

```

locale FriendObservationSetup =
  fixes UIDs :: userID set — local group of observers at a given node
begin

```

```

fun  $\gamma$  :: (state,act,out) trans  $\Rightarrow$  bool where
 $\gamma$  (Trans - a - -) = ( $\exists$  uid. userOfA a = Some uid  $\wedge$  uid  $\in$  UIDs  $\vee$  ( $\exists$  ca. a =
```

```

COMact ca))

```

```

fun  $g$  :: (state,act,out)trans  $\Rightarrow$  obs where
 $g$  (Trans - a ou -) = (a,ou)

```

```

end

```

```

locale FriendNetworkObservationSetup =
  fixes UIDs :: apiID  $\Rightarrow$  userID set — groups of observers at different nodes
begin

```

```

abbreviation  $\gamma$  :: apiID  $\Rightarrow$  (state,act,out) trans  $\Rightarrow$  bool where
 $\gamma$  aid trn  $\equiv$  FriendObservationSetup. $\gamma$  (UIDs aid) trn

```

```

abbreviation  $g$  :: apiID  $\Rightarrow$  (state,act,out)trans  $\Rightarrow$  obs where

```

g aid trn \equiv *FriendObservationSetup.g trn*

end

end

theory *Friend-State-Indistinguishability*
imports *Friend-Observation-Setup*
begin

7.2 Unwinding helper definitions and lemmas

locale *Friend* = *FriendObservationSetup* +

fixes

UID1 :: *userID*

and

UID2 :: *userID*

assumes

UID1-UID2-UIDs: $\{UID1, UID2\} \cap UIDs = \{\}$

and

UID1-UID2: $UID1 \neq UID2$

begin

fun *eqButUIDl* :: *userID* \Rightarrow *userID list* \Rightarrow *userID list* \Rightarrow *bool* **where**
eqButUIDl uid uidl uidl1 = (*remove1 uid uidl* = *remove1 uid uidl1*)

lemma *eqButUIDl-eq[simp,intro!]*: *eqButUIDl uid uidl uidl*
(*proof*)

lemma *eqButUIDl-sym*:

assumes *eqButUIDl uid uidl uidl1*

shows *eqButUIDl uid uidl1 uidl*

(*proof*)

lemma *eqButUIDl-trans*:

assumes *eqButUIDl uid uidl uidl1* **and** *eqButUIDl uid uidl1 uidl2*

shows *eqButUIDl uid uidl uidl2*

(*proof*)

lemma *eqButUIDl-remove1-cong*:

assumes *eqButUIDl uid uidl uidl1*

shows *eqButUIDl uid (remove1 uid' uidl) (remove1 uid' uidl1)*

(*proof*)

lemma *eqButUIDl-snoc-cong*:

assumes *eqButUIDl uid uidl uidl1*

and *uid' \in uidl \longleftrightarrow uid' \in uidl1*

shows *eqButUIDl uid (uidl ## uid') (uidl1 ## uid')*

<proof>

definition *eqButUIDf* **where**

eqButUIDf frds frds1 \equiv
eqButUIDl UID2 (frds UID1) (frds1 UID1)
 \wedge *eqButUIDl UID1 (frds UID2) (frds1 UID2)*
 $\wedge (\forall uid. uid \neq UID1 \wedge uid \neq UID2 \longrightarrow frds uid = frds1 uid)$

lemmas *eqButUIDf-intro* = *eqButUIDf-def*[*THEN meta-eq-to-obj-eq, THEN iffD2*]

lemma *eqButUIDf-eeq*[*simp,intro!*]: *eqButUIDf frds frds*
<proof>

lemma *eqButUIDf-sym*:
assumes *eqButUIDf frds frds1* **shows** *eqButUIDf frds1 frds*
<proof>

lemma *eqButUIDf-trans*:
assumes *eqButUIDf frds frds1* **and** *eqButUIDf frds1 frds2*
shows *eqButUIDf frds frds2*
<proof>

lemma *eqButUIDf-cong*:
assumes *eqButUIDf frds frds1*
and *uid = UID1* \implies *eqButUIDl UID2 uu uu1*
and *uid = UID2* \implies *eqButUIDl UID1 uu uu1*
and *uid* \neq *UID1* \implies *uid* \neq *UID2* \implies *uu = uu1*
shows *eqButUIDf (frds (uid := uu)) (frds1 (uid := uu1))*
<proof>

lemma *eqButUIDf-eqButUIDl*:
assumes *eqButUIDf frds frds1*
shows *eqButUIDl UID2 (frds UID1) (frds1 UID1)*
and *eqButUIDl UID1 (frds UID2) (frds1 UID2)*
<proof>

lemma *eqButUIDf-not-UID*:
 $\llbracket eqButUIDf frds frds1; uid \neq UID1; uid \neq UID2 \rrbracket \implies frds uid = frds1 uid$
<proof>

lemma *eqButUIDf-not-UID'*:
assumes *eq1: eqButUIDf frds frds1*
and *uid: (uid,uid')* \notin $\{(UID1,UID2), (UID2,UID1)\}$
shows *uid* \in *frds uid'* \longleftrightarrow *uid* \in *frds1 uid'*
<proof>

definition *eqButUID12* **where**

$eqButUID12\ freq\ freq1 \equiv$
 $\forall\ uid\ uid'.\ if\ (uid,uid') \in \{(UID1,UID2), (UID2,UID1)\}$ then True else $freq\ uid$
 $uid' = freq1\ uid\ uid'$

lemmas $eqButUID12-intro = eqButUID12-def[THEN\ meta-eq-to-obj-eq,\ THEN$
 $iffD2]$

lemma $eqButUID12-eeq[simp,intro!]: eqButUID12\ freq\ freq$
 $\langle proof \rangle$

lemma $eqButUID12-sym:$
assumes $eqButUID12\ freq\ freq1$ **shows** $eqButUID12\ freq1\ freq$
 $\langle proof \rangle$

lemma $eqButUID12-trans:$
assumes $eqButUID12\ freq\ freq1$ **and** $eqButUID12\ freq1\ freq2$
shows $eqButUID12\ freq\ freq2$
 $\langle proof \rangle$

lemma $eqButUID12-cong:$
assumes $eqButUID12\ freq\ freq1$

and $\neg (uid,uid') \in \{(UID1,UID2), (UID2,UID1)\} \implies uu = uu1$
shows $eqButUID12\ (fun-upd2\ freq\ uid\ uid'\ uu)\ (fun-upd2\ freq1\ uid\ uid'\ uu1)$
 $\langle proof \rangle$

lemma $eqButUID12-not-UID:$
 $\llbracket eqButUID12\ freq\ freq1; \neg (uid,uid') \in \{(UID1,UID2), (UID2,UID1)\} \rrbracket \implies freq$
 $uid\ uid' = freq1\ uid\ uid'$
 $\langle proof \rangle$

definition $eqButUID :: state \Rightarrow state \Rightarrow bool$ **where**
 $eqButUID\ s\ s1 \equiv$
 $admin\ s = admin\ s1 \wedge$

$pendingUReqs\ s = pendingUReqs\ s1 \wedge userReq\ s = userReq\ s1 \wedge$
 $userIDs\ s = userIDs\ s1 \wedge user\ s = user\ s1 \wedge pass\ s = pass\ s1 \wedge$

$eqButUIDf\ (pendingFReqs\ s)\ (pendingFReqs\ s1) \wedge$
 $eqButUID12\ (friendReq\ s)\ (friendReq\ s1) \wedge$
 $eqButUIDf\ (friendIDs\ s)\ (friendIDs\ s1) \wedge$

$postIDs\ s = postIDs\ s1 \wedge admin\ s = admin\ s1 \wedge$
 $post\ s = post\ s1 \wedge vis\ s = vis\ s1 \wedge$
 $owner\ s = owner\ s1 \wedge$

$pendingSApiReqs\ s = pendingSApiReqs\ s1 \wedge sApiReq\ s = sApiReq\ s1 \wedge$

$serverApiIDs\ s = serverApiIDs\ s1 \wedge serverPass\ s = serverPass\ s1 \wedge$
 $outerPostIDs\ s = outerPostIDs\ s1 \wedge outerPost\ s = outerPost\ s1 \wedge outerVis\ s =$
 $outerVis\ s1 \wedge$
 $outerOwner\ s = outerOwner\ s1 \wedge$
 $sentOuterFriendIDs\ s = sentOuterFriendIDs\ s1 \wedge$
 $recvOuterFriendIDs\ s = recvOuterFriendIDs\ s1 \wedge$

$pendingCApiReqs\ s = pendingCApiReqs\ s1 \wedge cApiReq\ s = cApiReq\ s1 \wedge$
 $clientApiIDs\ s = clientApiIDs\ s1 \wedge clientPass\ s = clientPass\ s1 \wedge$
 $sharedWith\ s = sharedWith\ s1$

lemmas $eqButUID-intro = eqButUID-def[THEN\ meta-eq-to-obj-eq,\ THEN\ iffD2]$

lemma $eqButUID-refl[simp,intro!]$: $eqButUID\ s\ s$
 $\langle proof \rangle$

lemma $eqButUID-sym[sym]$:
assumes $eqButUID\ s\ s1$ **shows** $eqButUID\ s1\ s$
 $\langle proof \rangle$

lemma $eqButUID-trans[trans]$:
assumes $eqButUID\ s\ s1$ **and** $eqButUID\ s1\ s2$ **shows** $eqButUID\ s\ s2$
 $\langle proof \rangle$

lemma $eqButUID-stateSelectors$:
assumes $eqButUID\ s\ s1$
shows $admin\ s = admin\ s1$
 $pendingUReqs\ s = pendingUReqs\ s1\ userReq\ s = userReq\ s1$
 $userIDs\ s = userIDs\ s1\ user\ s = user\ s1\ pass\ s = pass\ s1$
 $eqButUIDf\ (pendingFReqs\ s)\ (pendingFReqs\ s1)$
 $eqButUID12\ (friendReq\ s)\ (friendReq\ s1)$
 $eqButUIDf\ (friendIDs\ s)\ (friendIDs\ s1)$

$postIDs\ s = postIDs\ s1$
 $post\ s = post\ s1\ vis\ s = vis\ s1$
 $owner\ s = owner\ s1$

$pendingSApiReqs\ s = pendingSApiReqs\ s1\ sApiReq\ s = sApiReq\ s1$
 $serverApiIDs\ s = serverApiIDs\ s1\ serverPass\ s = serverPass\ s1$
 $outerPostIDs\ s = outerPostIDs\ s1\ outerPost\ s = outerPost\ s1\ outerVis\ s = outer-$
 $Vis\ s1$
 $outerOwner\ s = outerOwner\ s1$
 $sentOuterFriendIDs\ s = sentOuterFriendIDs\ s1$
 $recvOuterFriendIDs\ s = recvOuterFriendIDs\ s1$

$pendingCApiReqs\ s = pendingCApiReqs\ s1\ cApiReq\ s = cApiReq\ s1$
 $clientApiIDs\ s = clientApiIDs\ s1\ clientPass\ s = clientPass\ s1$
 $sharedWith\ s = sharedWith\ s1$

$IDsOK\ s = IDsOK\ s1$

$\langle proof \rangle$

lemma $eqButUID$ - $eqButUID2$:

$eqButUID\ s\ s1 \implies eqButUIDl\ UID2\ (friendIDs\ s\ UID1)\ (friendIDs\ s1\ UID1)$

$\langle proof \rangle$

lemma $eqButUID$ - not - UID :

$eqButUID\ s\ s1 \implies uid \neq UID \implies post\ s\ uid = post\ s1\ uid$

$\langle proof \rangle$

lemma $eqButUID$ - $cong[simp, intro]$:

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!admin := uu1\!))\ (s1\ (\!admin := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!pendingUReqs := uu1\!))\ (s1\ (\!pendingUReqs := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!userReq := uu1\!))\ (s1\ (\!userReq := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!userIDs := uu1\!))\ (s1\ (\!userIDs := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!user := uu1\!))\ (s1\ (\!user := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!pass := uu1\!))\ (s1\ (\!pass := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!postIDs := uu1\!))\ (s1\ (\!postIDs := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!owner := uu1\!))\ (s1\ (\!owner := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!post := uu1\!))\ (s1\ (\!post := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!vis := uu1\!))\ (s1\ (\!vis := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies eqButUIDf\ uu1\ uu2 \implies eqButUID\ (s\ (\!pendingFReqs := uu1\!))\ (s1\ (\!pendingFReqs := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies eqButUID12\ uu1\ uu2 \implies eqButUID\ (s\ (\!friendReq := uu1\!))\ (s1\ (\!friendReq := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies eqButUIDf\ uu1\ uu2 \implies eqButUID\ (s\ (\!friendIDs := uu1\!))\ (s1\ (\!friendIDs := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!pendingSApiReqs := uu1\!))\ (s1\ (\!pendingSApiReqs := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!sApiReq := uu1\!))\ (s1\ (\!sApiReq := uu2\!))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!serverApiIDs :=$

$uu1)) (s1 \langle serverApiIDs := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle serverPass := uu1 \rangle)$
 $(s1 \langle serverPass := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle outerPostIDs := uu1 \rangle)$
 $(s1 \langle outerPostIDs := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle outerPost := uu1 \rangle)$
 $(s1 \langle outerPost := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle outerVis := uu1 \rangle)$
 $(s1 \langle outerVis := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle outerOwner := uu1 \rangle)$
 $(s1 \langle outerOwner := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle sentOuterFriendIDs := uu1 \rangle)$
 $(s1 \langle sentOuterFriendIDs := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle recvOuterFriendIDs := uu1 \rangle)$
 $(s1 \langle recvOuterFriendIDs := uu2 \rangle)$

$\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle pendingCApiReqs := uu1 \rangle)$
 $(s1 \langle pendingCApiReqs := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle cApiReq := uu1 \rangle)$
 $(s1 \langle cApiReq := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle clientApiIDs := uu1 \rangle)$
 $(s1 \langle clientApiIDs := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle clientPass := uu1 \rangle)$
 $(s1 \langle clientPass := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle sharedWith := uu1 \rangle)$
 $(s1 \langle sharedWith := uu2 \rangle)$
 $\langle proof \rangle$

definition $friends12 :: state \Rightarrow bool$

where $friends12 s \equiv UID1 \in \in friendIDs s \wedge UID2 \in \in friendIDs s \wedge UID1$

lemma $step-friendIDs:$

assumes $step s a = (ou, s')$

and $a \neq Cact (cFriend uid (pass s uid) uid') \wedge a \neq Cact (cFriend uid' (pass s uid') uid) \wedge$

$a \neq Dact (dFriend uid (pass s uid) uid') \wedge a \neq Dact (dFriend uid' (pass s uid') uid)$

shows $uid \in \in friendIDs s' \iff uid \in \in friendIDs s \wedge uid' \in \in friendIDs s$ (**is** ?uid)

and $uid' \in \in friendIDs s' \iff uid' \in \in friendIDs s \wedge uid \in \in friendIDs s$ (**is** ?uid')

$\langle proof \rangle$

lemma $step-friends12:$

assumes $step s a = (ou, s')$

and $a \neq Cact (cFriend UID1 (pass s UID1) UID2) \wedge a \neq Cact (cFriend UID2 (pass s UID2) UID1) \wedge$

$a \neq Dact (dFriend UID1 (pass s UID1) UID2) \wedge a \neq Dact (dFriend UID2 (pass s UID2) UID1)$

shows $friends12 s' \iff friends12 s$

$\langle proof \rangle$

lemma *step-pendingFReqs*:
assumes *step*: $step\ s\ a = (ou, s')$
and $\forall req. a \neq Cact\ (cFriend\ uid\ (pass\ s\ uid)\ uid') \wedge a \neq Cact\ (cFriend\ uid'\ (pass\ s\ uid')\ uid) \wedge$
 $a \neq Dact\ (dFriend\ uid\ (pass\ s\ uid)\ uid') \wedge a \neq Dact\ (dFriend\ uid'\ (pass\ s\ uid')\ uid) \wedge$
 $a \neq Cact\ (cFriendReq\ uid\ (pass\ s\ uid)\ uid'\ req) \wedge$
 $a \neq Cact\ (cFriendReq\ uid'\ (pass\ s\ uid')\ uid\ req)$
shows $uid \in \in pendingFReqs\ s'\ uid' \longleftrightarrow uid \in \in pendingFReqs\ s\ uid' \text{ (is ?uid)}$
and $uid' \in \in pendingFReqs\ s'\ uid \longleftrightarrow uid' \in \in pendingFReqs\ s\ uid \text{ (is ?uid')}$
 $\langle proof \rangle$

lemma *eqButUID-friends12-set-friendIDs-eq*:
assumes *ss1*: $eqButUID\ s\ s1$
and *f12*: $friends12\ s = friends12\ s1$
and *rs*: $reach\ s$ **and** *rs1*: $reach\ s1$
shows $set\ (friendIDs\ s\ uid) = set\ (friendIDs\ s1\ uid)$
 $\langle proof \rangle$

lemma *distinct-remove1-idem*: $distinct\ xs \implies remove1\ y\ (remove1\ y\ xs) = remove1\ y\ xs$
 $\langle proof \rangle$

lemma *Cact-cFriend-step-eqButUID*:
assumes *step*: $step\ s\ (Cact\ (cFriend\ uid\ p\ uid')) = (ou, s')$
and *s*: $reach\ s$
and *uids*: $(uid = UID1 \wedge uid' = UID2) \vee (uid = UID2 \wedge uid' = UID1) \text{ (is ?u12} \vee ?u21)$
shows $eqButUID\ s\ s'$
 $\langle proof \rangle$

lemma *Cact-cFriendReq-step-eqButUID*:
assumes *step*: $step\ s\ (Cact\ (cFriendReq\ uid\ p\ uid'\ req)) = (ou, s')$
and *uids*: $(uid = UID1 \wedge uid' = UID2) \vee (uid = UID2 \wedge uid' = UID1) \text{ (is ?u12} \vee ?u21)$
shows $eqButUID\ s\ s'$
 $\langle proof \rangle$

lemma *Dact-dFriend-step-eqButUID*:
assumes *step*: $step\ s\ (Dact\ (dFriend\ uid\ p\ uid')) = (ou, s')$
and *s*: $reach\ s$
and *uids*: $(uid = UID1 \wedge uid' = UID2) \vee (uid = UID2 \wedge uid' = UID1) \text{ (is ?u12} \vee ?u21)$
shows $eqButUID\ s\ s'$
 $\langle proof \rangle$

lemma *eqButUID-step*:
assumes *ss1*: *eqButUID s s1*
and *step*: *step s a = (ou,s[^])*
and *step1*: *step s1 a = (ou1,s1[^])*
and *rs*: *reach s*
and *rs1*: *reach s1*
shows *eqButUID s' s1'*
<proof>

lemma *eqButUID-step-friendIDs-eq*:
assumes *ss1*: *eqButUID s s1*
and *rs*: *reach s* **and** *rs1*: *reach s1*
and *step*: *step s a = (ou,s[^])* **and** *step1*: *step s1 a = (ou1,s1[^])*
and *a*: *a ≠ Cact (cFriend UID1 (pass s UID1) UID2) ∧ a ≠ Cact (cFriend UID2 (pass s UID2) UID1) ∧*
a ≠ Dact (dFriend UID1 (pass s UID1) UID2) ∧ a ≠ Dact (dFriend UID2 (pass s UID2) UID1)
and *friendIDs s = friendIDs s1*
shows *friendIDs s' = friendIDs s1'*
<proof>

lemma *createFriend-sym*: *createFriend s uid p uid' = createFriend s uid' p' uid*
<proof>

lemma *deleteFriend-sym*: *deleteFriend s uid p uid' = deleteFriend s uid' p' uid*
<proof>

lemma *createFriendReq-createFriend-absorb*:
assumes *e-createFriendReq s uid' p uid req*
shows *createFriend (createFriendReq s uid' p1 uid req) uid p2 uid' = createFriend s uid p3 uid'*
<proof>

lemma *eqButUID-deleteFriend12-friendIDs-eq*:
assumes *ss1*: *eqButUID s s1*
and *rs*: *reach s* **and** *rs1*: *reach s1*
shows *friendIDs (deleteFriend s UID1 p UID2) = friendIDs (deleteFriend s1 UID1 p' UID2)*
<proof>

lemma *eqButUID-createFriend12-friendIDs-eq*:
assumes *ss1*: *eqButUID s s1*
and *rs*: *reach s* **and** *rs1*: *reach s1*
and *f12*: *¬friends12 s ¬friends12 s1*
shows *friendIDs (createFriend s UID1 p UID2) = friendIDs (createFriend s1 UID1 p' UID2)*
<proof>

$s \text{ uid}$
 $\wedge \text{uid}' \in \text{UIDs}$
 $\langle \text{proof} \rangle$

$$\text{uid} \in \text{UIDs} \wedge \text{uid}' \in \{\text{UID1}, \text{UID2}\} \vee \text{uid} \in \{\text{UID1}, \text{UID2}\}$$

$$\text{openByF } s \neg \text{openByF } s'$$

lemma *step-open-cases:*

assumes *step:* $\text{step } s \ a = (\text{ou}, s')$

and *op:* $\text{open } s \neq \text{open } s'$

obtains

$(\text{CloseA}) \text{ uid } p \ \text{uid}' \ p'$ **where** $a = \text{Cact } (\text{cUser } \text{uid } p \ \text{uid}' \ p')$
 $\text{uid}' = \text{UID1} \vee \text{uid}' = \text{UID2}$ $\text{ou} = \text{outOK}$ $p = \text{pass } s \ \text{uid}$
 $\text{openByA } s \neg \text{openByA } s' \neg \text{openByF } s \neg \text{openByF } s'$
 $| (\text{OpenF}) \text{ uid } p \ \text{uid}'$ **where** $a = \text{Cact } (\text{cFriend } \text{uid } p \ \text{uid}')$ $\text{ou} = \text{outOK}$ $p = \text{pass}$
 $s \ \text{uid}$

$\text{uid} \in \text{UIDs} \wedge \text{uid}' \in \{\text{UID1}, \text{UID2}\} \vee \text{uid} \in \{\text{UID1}, \text{UID2}\}$
 $\wedge \text{uid}' \in \text{UIDs}$
 $\text{openByF } s' \neg \text{openByF } s \neg \text{openByA } s \neg \text{openByA } s'$
 $| (\text{CloseF}) \text{ uid } p \ \text{uid}'$ **where** $a = \text{Dact } (\text{dFriend } \text{uid } p \ \text{uid}')$ $\text{ou} = \text{outOK}$ $p = \text{pass}$
 $s \ \text{uid}$

$\text{uid} \in \text{UIDs} \wedge \text{uid}' \in \{\text{UID1}, \text{UID2}\} \vee \text{uid} \in \{\text{UID1}, \text{UID2}\}$
 $\wedge \text{uid}' \in \text{UIDs}$
 $\text{openByF } s \neg \text{openByF } s' \neg \text{openByA } s \neg \text{openByA } s'$
 $\langle \text{proof} \rangle$

lemma *eqButUID-openByA-eq:*

assumes $\text{eqButUID } s \ s1$

shows $\text{openByA } s = \text{openByA } s1$

$\langle \text{proof} \rangle$

lemma *eqButUID-openByF-eq:*

assumes $ss1: \text{eqButUID } s \ s1$

shows $\text{openByF } s = \text{openByF } s1$

$\langle \text{proof} \rangle$

lemma *eqButUID-open-eq:* $\text{eqButUID } s \ s1 \implies \text{open } s = \text{open } s1$

$\langle \text{proof} \rangle$

lemma *eqButUID-step- γ -out:*

assumes $ss1: \text{eqButUID } s \ s1$

and *step:* $\text{step } s \ a = (\text{ou}, s')$ **and** *step1:* $\text{step } s1 \ a = (\text{ou1}, s1')$

and $\gamma: \gamma (\text{Trans } s \ a \ \text{ou} \ s')$

and $\text{os}: \text{open } s \longrightarrow \text{friendIDs } s = \text{friendIDs } s1$

shows $\text{ou} = \text{ou1}$

$\langle \text{proof} \rangle$

end

end

theory *Friend-Value-Setup*
 imports *Friend-Openness*
begin

7.4 Value Setup

context *Friend*
begin

datatype *value* =
 FrVal bool — updated friendship status between *UID1* and *UID2*
 | *OVal bool* — updated dynamic declassification trigger condition

fun $\varphi :: (state, act, out) trans \Rightarrow bool$ **where**
 $\varphi (Trans\ s\ (Cact\ (cFriend\ uid\ p\ uid'))\ ou\ s') =$
 $((uid, uid') \in \{(UID1, UID2), (UID2, UID1)\} \wedge ou = outOK \vee$
 $open\ s \neq open\ s')$
 |
 $\varphi (Trans\ s\ (Dact\ (dFriend\ uid\ p\ uid'))\ ou\ s') =$
 $((uid, uid') \in \{(UID1, UID2), (UID2, UID1)\} \wedge ou = outOK \vee$
 $open\ s \neq open\ s')$
 |
 $\varphi (Trans\ s\ (Cact\ (cUser\ uid\ p\ uid'\ p'))\ ou\ s') =$
 $(open\ s \neq open\ s')$
 |
 $\varphi - = False$

fun $f :: (state, act, out) trans \Rightarrow value$ **where**
 $f (Trans\ s\ (Cact\ (cFriend\ uid\ p\ uid'))\ ou\ s') =$
 $(if\ (uid, uid') \in \{(UID1, UID2), (UID2, UID1)\}$ *then FrVal True*
 else OVal True)
 |
 $f (Trans\ s\ (Dact\ (dFriend\ uid\ p\ uid'))\ ou\ s') =$
 $(if\ (uid, uid') \in \{(UID1, UID2), (UID2, UID1)\}$ *then FrVal False*
 else OVal False)
 |
 $f (Trans\ s\ (Cact\ (cUser\ uid\ p\ uid'\ p'))\ ou\ s') = OVal\ False$
 |
 $f - = undefined$

lemma φE :
assumes $\varphi (Trans\ s\ a\ ou\ s')$ (**is** $\varphi\ ?trn$)
and *step*: $step\ s\ a = (ou, s')$

and $rs: reach\ s$
obtains $(Friend)\ uid\ p\ uid'$ **where** $a = Cact\ (cFriend\ uid\ p\ uid')$ $ou = outOK\ f$
 $?trn = FrVal\ True$
 $uid = UID1 \wedge uid' = UID2 \vee uid = UID2 \wedge uid' =$
 $UID1$
 $IDsOK\ s\ [UID1, UID2]\ \square\ \square\ \square$
 $\neg friends12\ s\ friends12\ s'$
 $| (Unfriend)\ uid\ p\ uid'$ **where** $a = Dact\ (dFriend\ uid\ p\ uid')$ $ou = outOK\ f$
 $?trn = FrVal\ False$
 $uid = UID1 \wedge uid' = UID2 \vee uid = UID2 \wedge uid' =$
 $UID1$
 $IDsOK\ s\ [UID1, UID2]\ \square\ \square\ \square$
 $friends12\ s\ \neg friends12\ s'$
 $| (OpenF)\ uid\ p\ uid'$ **where** $a = Cact\ (cFriend\ uid\ p\ uid')$
 $(uid \in UIDs \wedge uid' \in \{UID1, UID2\}) \vee (uid' \in UIDs \wedge$
 $uid \in \{UID1, UID2\})$
 $ou = outOK\ f\ ?trn = OVal\ True\ \neg openByF\ s\ openByF\ s'$
 $\neg openByA\ s\ \neg openByA\ s'$
 $| (CloseF)\ uid\ p\ uid'$ **where** $a = Dact\ (dFriend\ uid\ p\ uid')$
 $(uid \in UIDs \wedge uid' \in \{UID1, UID2\}) \vee (uid' \in UIDs$
 $\wedge uid \in \{UID1, UID2\})$
 $ou = outOK\ f\ ?trn = OVal\ False\ openByF\ s\ \neg openByF$
 s'
 $\neg openByA\ s\ \neg openByA\ s'$
 $| (CloseA)\ uid\ p\ uid'\ p'$ **where** $a = Cact\ (cUser\ uid\ p\ uid'\ p')$
 $uid' \in \{UID1, UID2\}\ openByA\ s\ \neg openByA\ s'$
 $\neg openByF\ s\ \neg openByF\ s'$
 $ou = outOK\ f\ ?trn = OVal\ False$
 $\langle proof \rangle$

lemma $step-open-\varphi$:
assumes $step\ s\ a = (ou, s')$
and $open\ s \neq open\ s'$
shows $\varphi\ (Trans\ s\ a\ ou\ s')$
 $\langle proof \rangle$

lemma $step-friends12-\varphi$:
assumes $step\ s\ a = (ou, s')$
and $friends12\ s \neq friends12\ s'$
shows $\varphi\ (Trans\ s\ a\ ou\ s')$
 $\langle proof \rangle$

lemma $eqButUID-step-\varphi-imp$:
assumes $ss1: eqButUID\ s\ s1$
and $rs: reach\ s$ **and** $rs1: reach\ s1$
and $step: step\ s\ a = (ou, s')$ **and** $step1: step\ s1\ a = (ou1, s1')$
and $a \neq Cact\ (cFriend\ UID1\ (pass\ s\ UID1)\ UID2) \wedge a \neq Cact\ (cFriend\ UID2$
 $(pass\ s\ UID2)\ UID1) \wedge$
 $a \neq Dact\ (dFriend\ UID1\ (pass\ s\ UID1)\ UID2) \wedge a \neq Dact\ (dFriend\ UID2$

```

(pass s UID2) UID1)
and  $\varphi$ :  $\varphi$  (Trans s a ou s')
shows  $\varphi$  (Trans s1 a ou1 s1')
<proof>

```

```

lemma eqButUID-step- $\varphi$ :
assumes ss1: eqButUID s s1
and rs: reach s and rs1: reach s1
and step: step s a = (ou,s^) and step1: step s1 a = (ou1,s1^)
and a:  $a \neq \text{Cact}(\text{cFriend UID1 } (\text{pass s UID1}) \text{ UID2}) \wedge a \neq \text{Cact}(\text{cFriend UID2 } (\text{pass s UID2}) \text{ UID1}) \wedge$ 
 $a \neq \text{Dact}(\text{dFriend UID1 } (\text{pass s UID1}) \text{ UID2}) \wedge a \neq \text{Dact}(\text{dFriend UID2 } (\text{pass s UID2}) \text{ UID1})$ 
shows  $\varphi$  (Trans s a ou s') =  $\varphi$  (Trans s1 a ou1 s1')
<proof>

```

end

end

theory *Friend*

imports

Friend-Value-Setup

Bounded-Deducibility-Security.Compositional-Reasoning

begin

7.5 Declassification bound

context *Friend*

begin

fun *T* :: (*state,act,out*) *trans* \Rightarrow *bool*

where *T trn* = *False*

The bound has the same “while-or-last-before” shape as the dynamic version of the issuer bound for post confidentiality (Section 6.5.2), alternating between phases with open (*BO*) or closed (*BC*) access to the confidential information.

The access window is initially open, because the two users are known not to exist when the system is initialized, so there cannot be friendship between them.

The bound also incorporates the static knowledge that the friendship status alternates between *False* and *True*.

fun *alternatingFriends* :: *value list* \Rightarrow *bool* \Rightarrow *bool* **where**

alternatingFriends [] = *True*

| *alternatingFriends* (*FrVal st # vl*) *st'* \longleftrightarrow *st'* = (\neg *st*) \wedge *alternatingFriends vl st*

| *alternatingFriends* (*OVal - # vl*) *st* = *alternatingFriends vl st*

inductive *BO* :: *value list* \Rightarrow *value list* \Rightarrow *bool*

and $BC :: \text{value list} \Rightarrow \text{value list} \Rightarrow \text{bool}$
where
 $BO\text{-}FrVal[simp,intro!]:$
 $BO (\text{map } FrVal fs) (\text{map } FrVal fs)$
 $|BO\text{-}BC[intro]:$
 $BC vl vl1 \Longrightarrow$
 $BO (\text{map } FrVal fs @ OVal False \# vl) (\text{map } FrVal fs @ OVal False \# vl1)$

 $|BC\text{-}FrVal[simp,intro!]:$
 $BC (\text{map } FrVal fs) (\text{map } FrVal fs1)$
 $|BC\text{-}BO[intro]:$
 $BO vl vl1 \Longrightarrow (fs = [] \longleftrightarrow fs1 = []) \Longrightarrow (fs \neq [] \Longrightarrow \text{last } fs = \text{last } fs1) \Longrightarrow$
 $BC (\text{map } FrVal fs @ OVal True \# vl)$
 $(\text{map } FrVal fs1 @ OVal True \# vl1)$

definition $B vl vl1 \equiv BO vl vl1 \wedge \text{alternatingFriends } vl1 \text{ False}$

lemma $BO\text{-}Nil\text{-}Nil: BO vl vl1 \Longrightarrow vl = [] \Longrightarrow vl1 = []$
 $\langle \text{proof} \rangle$

unbundle $no \text{ relcomp-syntax}$

sublocale $BD\text{-}Security\text{-}IO$ **where**
 $istate = istate$ **and** $step = step$ **and**
 $\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
 $\langle \text{proof} \rangle$

7.6 Unwinding proof

lemma $toggle\text{-}friends12\text{-}True:$
assumes $rs: \text{reach } s$
and $IDs: IDsOK s [UID1, UID2] [] [] []$
and $nf12: \neg \text{friends12 } s$
obtains $al \text{ out}$
where $sstep s al = (out, \text{createFriend } s \text{ UID1 } (\text{pass } s \text{ UID1}) \text{ UID2})$
and $al \neq []$ **and** $eqButUID s (\text{createFriend } s \text{ UID1 } (\text{pass } s \text{ UID1}) \text{ UID2})$
and $\text{friends12 } (\text{createFriend } s \text{ UID1 } (\text{pass } s \text{ UID1}) \text{ UID2})$
and $O (\text{traceOf } s al) = []$ **and** $V (\text{traceOf } s al) = [FrVal True]$
 $\langle \text{proof} \rangle$

lemma $toggle\text{-}friends12\text{-}False:$
assumes $rs: \text{reach } s$
and $IDs: IDsOK s [UID1, UID2] [] [] []$
and $f12: \text{friends12 } s$
obtains $al \text{ out}$
where $sstep s al = (out, \text{deleteFriend } s \text{ UID1 } (\text{pass } s \text{ UID1}) \text{ UID2})$
and $al \neq []$ **and** $eqButUID s (\text{deleteFriend } s \text{ UID1 } (\text{pass } s \text{ UID1}) \text{ UID2})$
and $\neg \text{friends12 } (\text{deleteFriend } s \text{ UID1 } (\text{pass } s \text{ UID1}) \text{ UID2})$

and $O(\text{traceOf } s \text{ al}) = []$ **and** $V(\text{traceOf } s \text{ al}) = [\text{FrVal False}]$
 ⟨proof⟩

definition $\Delta 0 :: \text{state} \Rightarrow \text{value list} \Rightarrow \text{state} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**
 $\Delta 0 \ s \ vl \ s1 \ vl1 \equiv$
 $\text{eqButUID } s \ s1 \wedge \text{friendIDs } s = \text{friendIDs } s1 \wedge \text{open } s \wedge$
 $\text{BO } vl \ vl1 \wedge \text{alternatingFriends } vl1 \ (\text{friends12 } s1)$

definition $\Delta 1 :: \text{state} \Rightarrow \text{value list} \Rightarrow \text{state} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**
 $\Delta 1 \ s \ vl \ s1 \ vl1 \equiv (\exists fs \ fs1.$
 $\text{eqButUID } s \ s1 \wedge \neg \text{open } s \wedge$
 $\text{alternatingFriends } vl1 \ (\text{friends12 } s1) \wedge$
 $vl = \text{map FrVal } fs \wedge vl1 = \text{map FrVal } fs1)$

definition $\Delta 2 :: \text{state} \Rightarrow \text{value list} \Rightarrow \text{state} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**
 $\Delta 2 \ s \ vl \ s1 \ vl1 \equiv (\exists fs \ fs1 \ vlr \ vlr1.$
 $\text{eqButUID } s \ s1 \wedge \neg \text{open } s \wedge \text{BO } vlr \ vlr1 \wedge$
 $\text{alternatingFriends } vl1 \ (\text{friends12 } s1) \wedge$
 $(fs = [] \longleftrightarrow fs1 = []) \wedge$
 $(fs \neq [] \longrightarrow \text{last } fs = \text{last } fs1) \wedge$
 $(fs = [] \longrightarrow \text{friendIDs } s = \text{friendIDs } s1) \wedge$
 $vl = \text{map FrVal } fs \ @ \ \text{OVal True} \ \# \ vlr \wedge$
 $vl1 = \text{map FrVal } fs1 \ @ \ \text{OVal True} \ \# \ vlr1)$

lemma $\Delta 2\text{-I}$:

assumes $\text{eqButUID } s \ s1 \ \neg \text{open } s \ \text{BO } vlr \ vlr1 \ \text{alternatingFriends } vl1 \ (\text{friends12 } s1)$
 $fs = [] \longleftrightarrow fs1 = [] \ fs \neq [] \longrightarrow \text{last } fs = \text{last } fs1$
 $fs = [] \longrightarrow \text{friendIDs } s = \text{friendIDs } s1$
 $vl = \text{map FrVal } fs \ @ \ \text{OVal True} \ \# \ vlr$
 $vl1 = \text{map FrVal } fs1 \ @ \ \text{OVal True} \ \# \ vlr1$

shows $\Delta 2 \ s \ vl \ s1 \ vl1$
 ⟨proof⟩

lemma $\text{istate-}\Delta 0$:

assumes $B: B \ vl \ vl1$
shows $\Delta 0 \ \text{istate } vl \ \text{istate } vl1$
 ⟨proof⟩

lemma $\text{unwind-cont-}\Delta 0$: $\text{unwind-cont } \Delta 0 \ \{\Delta 0, \Delta 1, \Delta 2\}$
 ⟨proof⟩

lemma $\text{unwind-cont-}\Delta 1$: $\text{unwind-cont } \Delta 1 \ \{\Delta 1, \Delta 0\}$
 ⟨proof⟩

lemma $\text{unwind-cont-}\Delta 2$: $\text{unwind-cont } \Delta 2 \ \{\Delta 2, \Delta 0\}$
 ⟨proof⟩

definition *Gr* where

```
Gr =  
{  
  ( $\Delta 0$ , { $\Delta 0, \Delta 1, \Delta 2$ }),  
  ( $\Delta 1$ , { $\Delta 1, \Delta 0$ }),  
  ( $\Delta 2$ , { $\Delta 2, \Delta 0$ })  
}
```

theorem *secure: secure*

<proof>

end

end

theory *Friend-Network*

imports

../API-Network

Friend

BD-Security-Compositional.Composing-Security-Network

begin

7.7 Confidentiality for the N-ary composition

locale *FriendNetwork* = *Network* + *FriendNetworkObservationSetup* +

fixes

AID :: *apiID*

and

UID1 :: *userID*

and

UID2 :: *userID*

assumes

UID1-UID2-UIDs: {*UID1, UID2*} \cap (*UIDs AID*) = {}

and

UID1-UID2: *UID1* \neq *UID2*

and

AID-AIDs: *AID* \in *AIDs*

begin

sublocale *Issuer: Friend* *UIDs AID UID1 UID2* *<proof>*

abbreviation φ :: *apiID* \Rightarrow (*state, act, out*) *trans* \Rightarrow *bool*

where φ *aid trn* \equiv (*Issuer.* φ *trn* \wedge *aid* = *AID*)

abbreviation *f* :: *apiID* \Rightarrow (*state, act, out*) *trans* \Rightarrow *Friend.value*

where *f* *aid trn* \equiv *Friend.f* *UID1 UID2 trn*

abbreviation *T* :: *apiID* \Rightarrow (*state, act, out*) *trans* \Rightarrow *bool*

where $T \text{ aid } trn \equiv \text{False}$

abbreviation $B :: \text{apiID} \Rightarrow \text{Friend.value list} \Rightarrow \text{Friend.value list} \Rightarrow \text{bool}$
where $B \text{ aid } vl \text{ vl1} \equiv (\text{if } \text{aid} = \text{AID} \text{ then } \text{Issuer.B } vl \text{ vl1} \text{ else } (vl = [] \wedge vl1 = []))$

abbreviation $\text{comOfV } \text{aid } vl \equiv \text{Internal}$

abbreviation $\text{tgtNodeOfV } \text{aid } vl \equiv \text{undefined}$

abbreviation $\text{syncV } \text{aid1 } vl1 \text{ aid2 } vl2 \equiv \text{False}$

lemma $[\text{simp}]$: $\text{validTrans } \text{aid } trn \implies \text{lreach } \text{aid } (\text{srcOf } trn) \implies \varphi \text{ aid } trn \implies$
 $\text{comOf } \text{aid } trn = \text{Internal}$
 $\langle \text{proof} \rangle$

sublocale $\text{Net: BD-Security-TS-Network-getTgtV}$

where $\text{istate} = \lambda\cdot. \text{istate}$ **and** $\text{validTrans} = \text{validTrans}$ **and** $\text{srcOf} = \lambda\cdot. \text{srcOf}$
and $\text{tgtOf} = \lambda\cdot. \text{tgtOf}$
and $\text{nodes} = \text{AIDs}$ **and** $\text{comOf} = \text{comOf}$ **and** $\text{tgtNodeOf} = \text{tgtNodeOf}$
and $\text{sync} = \text{sync}$ **and** $\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and**
 $B = B$

and $\text{comOfV} = \text{comOfV}$ **and** $\text{tgtNodeOfV} = \text{tgtNodeOfV}$ **and** $\text{syncV} = \text{syncV}$
and $\text{comOfO} = \text{comOfO}$ **and** $\text{tgtNodeOfO} = \text{tgtNodeOfO}$ **and** $\text{syncO} = \text{syncO}$
and $\text{source} = \text{AID}$ **and** $\text{getTgtV} = \text{id}$
 $\langle \text{proof} \rangle$

sublocale $\text{BD-Security-TS-Network-Preserve-Source-Security-getTgtV}$

where $\text{istate} = \lambda\cdot. \text{istate}$ **and** $\text{validTrans} = \text{validTrans}$ **and** $\text{srcOf} = \lambda\cdot. \text{srcOf}$
and $\text{tgtOf} = \lambda\cdot. \text{tgtOf}$
and $\text{nodes} = \text{AIDs}$ **and** $\text{comOf} = \text{comOf}$ **and** $\text{tgtNodeOf} = \text{tgtNodeOf}$
and $\text{sync} = \text{sync}$ **and** $\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and**
 $B = B$

and $\text{comOfV} = \text{comOfV}$ **and** $\text{tgtNodeOfV} = \text{tgtNodeOfV}$ **and** $\text{syncV} = \text{syncV}$
and $\text{comOfO} = \text{comOfO}$ **and** $\text{tgtNodeOfO} = \text{tgtNodeOfO}$ **and** $\text{syncO} = \text{syncO}$
and $\text{source} = \text{AID}$ **and** $\text{getTgtV} = \text{id}$
 $\langle \text{proof} \rangle$

theorem secure: secure

$\langle \text{proof} \rangle$

end

end

theory Friend-All

imports Friend-Network

begin

end

theory $\text{Friend-Request-Intro}$

imports

```

    ../Friend-Confidentiality/Friend-Openness
    ../Friend-Confidentiality/Friend-State-Indistinguishability
begin

```

8 Friendship request confidentiality

We verify the following property:

Given a coalition consisting of groups of users *UIDs* j from multiple nodes j and given two users *UID1* and *UID2* at some node i who are not in these groups,

the coalition cannot learn anything about the the friendship requests issued between *UID1* and *UID2*

beyond what everybody knows, namely that

- every successful friend creation is preceded by at least one and at most two requests, and
- friendship status updates form an alternating sequence of friending and unfriending,

and beyond the existence of requests issued while or last before a user in the group *UIDs* i is a local friend of *UID1* or *UID2*.

The approach here is similar to that for friendship status confidentiality (explained in the introduction of Section 7). Like in the case of friendship status, here secret information is not communicated between different nodes (so again we don't need to distinguish between an issuer node and the other, receiver nodes).

end

```

theory Friend-Request-Value-Setup
  imports Friend-Request-Intro
begin

```

8.1 Value setup

```

context Friend
begin

```

```

datatype fUser = U1 | U2

```

```

datatype value =

```

```

  isFRVal: FRVal fUser requestInfo — friendship requests from UID1 to UID2 (or
vice versa)

```

```

| isFVal: FVal bool — updated friendship status between UID1 and UID2

```

| *isOVal*: *OVal bool* — updated dynamic declassification trigger condition

fun $\varphi :: (state, act, out) \text{ trans} \Rightarrow \text{bool}$ **where**
 $\varphi (Trans\ s\ (Cact\ (cFriendReq\ uid\ p\ uid'\ req))\ ou\ s') =$
 $((uid, uid') \in \{(UID1, UID2), (UID2, UID1)\} \wedge ou = outOK)$
|
 $\varphi (Trans\ s\ (Cact\ (cFriend\ uid\ p\ uid'))\ ou\ s') =$
 $((uid, uid') \in \{(UID1, UID2), (UID2, UID1)\} \wedge ou = outOK \vee$
 $open\ s \neq open\ s')$
|
 $\varphi (Trans\ s\ (Dact\ (dFriend\ uid\ p\ uid'))\ ou\ s') =$
 $((uid, uid') \in \{(UID1, UID2), (UID2, UID1)\} \wedge ou = outOK \vee$
 $open\ s \neq open\ s')$
|
 $\varphi (Trans\ s\ (Cact\ (cUser\ uid\ p\ uid'\ p'))\ ou\ s') =$
 $(open\ s \neq open\ s')$
|
 $\varphi - = False$

fun $f :: (state, act, out) \text{ trans} \Rightarrow \text{value}$ **where**
 $f (Trans\ s\ (Cact\ (cFriendReq\ uid\ p\ uid'\ req))\ ou\ s') =$
 $(if\ uid = UID1 \wedge uid' = UID2\ then\ FRVal\ U1\ req$
 $else\ if\ uid = UID2 \wedge uid' = UID1\ then\ FRVal\ U2\ req$
 $else\ OVal\ True)$
|
 $f (Trans\ s\ (Cact\ (cFriend\ uid\ p\ uid'))\ ou\ s') =$
 $(if\ (uid, uid') \in \{(UID1, UID2), (UID2, UID1)\}\ then\ FVal\ True$
 $else\ OVal\ True)$
|
 $f (Trans\ s\ (Dact\ (dFriend\ uid\ p\ uid'))\ ou\ s') =$
 $(if\ (uid, uid') \in \{(UID1, UID2), (UID2, UID1)\}\ then\ FVal\ False$
 $else\ OVal\ False)$
|
 $f (Trans\ s\ (Cact\ (cUser\ uid\ p\ uid'\ p'))\ ou\ s') = OVal\ False$
|
 $f - = undefined$

lemma φE :

assumes $\varphi: \varphi (Trans\ s\ a\ ou\ s')\ (\text{is}\ \varphi\ ?trn)$

and step: $step\ s\ a = (ou, s')$

and rs: $reach\ s$

obtains $(FReq1)\ u\ p\ req$ **where** $a = Cact\ (cFriendReq\ UID1\ p\ UID2\ req)\ ou = outOK$

$$f\ ?trn = FRVal\ u\ req\ u = U1\ IDsOK\ s\ [UID1,\ UID2]\ \square\ \square\ \square$$

$$\neg friends12\ s\ \neg friends12\ s'\ open\ s' = open\ s$$

$UID1 \in \in pendingFReqs\ s'\ UID2\ UID1 \notin set\ (pendingFReqs$
 $s\ UID2)$

$s\ UID1$

$$UID2 \in \in pendingFReqs\ s'\ UID1 \longleftrightarrow UID2 \in \in pendingFReqs$$

$$\begin{aligned}
& | (FReq2) \ u \ p \ req \ \mathbf{where} \ a = Cact \ (cFriendReq \ UID2 \ p \ UID1 \ req) \ ou = \\
& outOK \\
& \quad f \ ?trn = FVal \ u \ req \ u = U2 \ IDsOK \ s \ [UID1, \ UID2] \ [] \ [] \ [] \\
& \quad \neg friends12 \ s \ \neg friends12 \ s' \ open \ s' = open \ s \\
& \quad UID2 \ \in \in \ pendingFReqs \ s' \ UID1 \ UID2 \notin set \ (pendingFReqs \\
& s \ UID1) \\
& \quad UID1 \ \in \in \ pendingFReqs \ s' \ UID2 \longleftrightarrow UID1 \ \in \in \ pendingFReqs \\
& s \ UID2 \\
& | (Friend) \ uid \ p \ uid' \ \mathbf{where} \ a = Cact \ (cFriend \ uid \ p \ uid') \ ou = outOK \ f \ ?trn \\
& = FVal \ True \\
& \quad uid = UID1 \wedge uid' = UID2 \vee uid = UID2 \wedge uid' = \\
& UID1 \\
& \quad IDsOK \ s \ [UID1, \ UID2] \ [] \ [] \ [] \\
& \quad \neg friends12 \ s \ friends12 \ s' \ uid' \ \in \in \ pendingFReqs \ s \ uid \\
& \quad UID1 \notin set \ (pendingFReqs \ s' \ UID2) \\
& \quad UID2 \notin set \ (pendingFReqs \ s' \ UID1) \\
& | (Unfriend) \ uid \ p \ uid' \ \mathbf{where} \ a = Dact \ (dFriend \ uid \ p \ uid') \ ou = outOK \ f \\
& ?trn = FVal \ False \\
& \quad uid = UID1 \wedge uid' = UID2 \vee uid = UID2 \wedge uid' = \\
& UID1 \\
& \quad IDsOK \ s \ [UID1, \ UID2] \ [] \ [] \ [] \\
& \quad friends12 \ s \ \neg friends12 \ s' \\
& \quad UID1 \notin set \ (pendingFReqs \ s' \ UID2) \\
& \quad UID1 \notin set \ (pendingFReqs \ s \ UID2) \\
& \quad UID2 \notin set \ (pendingFReqs \ s' \ UID1) \\
& \quad UID2 \notin set \ (pendingFReqs \ s \ UID1) \\
& | (OpenF) \ uid \ p \ uid' \ \mathbf{where} \ a = Cact \ (cFriend \ uid \ p \ uid') \\
& \quad (uid \ \in \ IDs \wedge uid' \ \in \ \{UID1, \ UID2\}) \vee (uid' \ \in \ IDs \wedge \\
& uid \ \in \ \{UID1, \ UID2\}) \\
& \quad ou = outOK \ f \ ?trn = OVal \ True \ \neg openByF \ s \ openByF \ s' \\
& \quad \neg openByA \ s \ \neg openByA \ s' \\
& \quad friends12 \ s' = friends12 \ s \\
& \quad UID1 \ \in \in \ pendingFReqs \ s' \ UID2 \longleftrightarrow UID1 \ \in \in \\
& pendingFReqs \ s \ UID2 \\
& \quad UID2 \ \in \in \ pendingFReqs \ s' \ UID1 \longleftrightarrow UID2 \ \in \in \\
& pendingFReqs \ s \ UID1 \\
& | (CloseF) \ uid \ p \ uid' \ \mathbf{where} \ a = Dact \ (dFriend \ uid \ p \ uid') \\
& \quad (uid \ \in \ IDs \wedge uid' \ \in \ \{UID1, \ UID2\}) \vee (uid' \ \in \ IDs \\
& \wedge uid \ \in \ \{UID1, \ UID2\}) \\
& \quad ou = outOK \ f \ ?trn = OVal \ False \ openByF \ s \ \neg openByF \\
& s' \\
& \quad \neg openByA \ s \ \neg openByA \ s' \\
& \quad friends12 \ s' = friends12 \ s \\
& \quad UID1 \ \in \in \ pendingFReqs \ s' \ UID2 \longleftrightarrow UID1 \ \in \in \\
& pendingFReqs \ s \ UID2 \\
& \quad UID2 \ \in \in \ pendingFReqs \ s' \ UID1 \longleftrightarrow UID2 \ \in \in \\
& pendingFReqs \ s \ UID1 \\
& | (CloseA) \ uid \ p \ uid' \ p' \ \mathbf{where} \ a = Cact \ (cUser \ uid \ p \ uid' \ p') \\
& \quad uid' \ \in \ \{UID1, \ UID2\} \ openByA \ s \ \neg openByA \ s'
\end{aligned}$$

$\neg \text{openByF } s \ \neg \text{openByF } s'$
 $ou = \text{outOK } f \ ?trn = \text{OVal False}$
 $\text{friends12 } s' = \text{friends12 } s$
 $UID1 \in \in \text{pendingFReqs } s' \ \text{UID2} \longleftrightarrow UID1 \in \in$
 $\text{pendingFReqs } s \ \text{UID2}$
 $UID2 \in \in \text{pendingFReqs } s' \ \text{UID1} \longleftrightarrow UID2 \in \in$
 $\text{pendingFReqs } s \ \text{UID1}$
 $\langle \text{proof} \rangle$

lemma *step-open-φ*:
assumes $\text{step } s \ a = (ou, s')$
and $\text{open } s \neq \text{open } s'$
shows $\varphi (\text{Trans } s \ a \ ou \ s')$
 $\langle \text{proof} \rangle$

lemma *step-friends12-φ*:
assumes $\text{step } s \ a = (ou, s')$
and $\text{friends12 } s \neq \text{friends12 } s'$
shows $\varphi (\text{Trans } s \ a \ ou \ s')$
 $\langle \text{proof} \rangle$

lemma *step-pendingFReqs-φ*:
assumes $\text{step } s \ a = (ou, s')$
and $(UID1 \in \in \text{pendingFReqs } s \ \text{UID2}) \neq (UID1 \in \in \text{pendingFReqs } s' \ \text{UID2})$
 $\vee (UID2 \in \in \text{pendingFReqs } s \ \text{UID1}) \neq (UID2 \in \in \text{pendingFReqs } s' \ \text{UID1})$
shows $\varphi (\text{Trans } s \ a \ ou \ s')$
 $\langle \text{proof} \rangle$

lemma *eqButUID-step-φ-imp*:
assumes $ss1: \text{eqButUID } s \ s1$
and $rs: \text{reach } s$ **and** $rs1: \text{reach } s1$
and $\text{step}: \text{step } s \ a = (ou, s')$ **and** $\text{step1}: \text{step } s1 \ a = (ou1, s1')$
and $a: \forall \text{req}. a \neq \text{Cact } (c\text{Friend } UID1 \ (\text{pass } s \ \text{UID1}) \ \text{UID2}) \wedge$
 $a \neq \text{Cact } (c\text{Friend } UID2 \ (\text{pass } s \ \text{UID2}) \ \text{UID1}) \wedge$
 $a \neq \text{Cact } (c\text{FriendReq } UID1 \ (\text{pass } s \ \text{UID1}) \ \text{UID2 } \text{req}) \wedge$
 $a \neq \text{Cact } (c\text{FriendReq } UID2 \ (\text{pass } s \ \text{UID2}) \ \text{UID1 } \text{req}) \wedge$
 $a \neq \text{Dact } (d\text{Friend } UID1 \ (\text{pass } s \ \text{UID1}) \ \text{UID2}) \wedge$
 $a \neq \text{Dact } (d\text{Friend } UID2 \ (\text{pass } s \ \text{UID2}) \ \text{UID1})$
and $\varphi: \varphi (\text{Trans } s \ a \ ou \ s')$
shows $\varphi (\text{Trans } s1 \ a \ ou1 \ s1')$
 $\langle \text{proof} \rangle$

lemma *eqButUID-step-φ*:
assumes $ss1: \text{eqButUID } s \ s1$
and $rs: \text{reach } s$ **and** $rs1: \text{reach } s1$
and $\text{step}: \text{step } s \ a = (ou, s')$ **and** $\text{step1}: \text{step } s1 \ a = (ou1, s1')$
and $a: \forall \text{req}. a \neq \text{Cact } (c\text{Friend } UID1 \ (\text{pass } s \ \text{UID1}) \ \text{UID2}) \wedge$
 $a \neq \text{Cact } (c\text{Friend } UID2 \ (\text{pass } s \ \text{UID2}) \ \text{UID1}) \wedge$
 $a \neq \text{Cact } (c\text{FriendReq } UID1 \ (\text{pass } s \ \text{UID1}) \ \text{UID2 } \text{req}) \wedge$

```

      a ≠ Cact (cFriendReq UID2 (pass s UID2) UID1 req) ∧
      a ≠ Dact (dFriend UID1 (pass s UID1) UID2) ∧
      a ≠ Dact (dFriend UID2 (pass s UID2) UID1)
shows  $\varphi$  (Trans s a ou s') =  $\varphi$  (Trans s1 a ou1 s1')
⟨proof⟩

```

end

end

theory *Friend-Request*

imports

Friend-Request-Value-Setup

Bounded-Deducibility-Security.Compositional-Reasoning

begin

8.2 Declassification bound

context *Friend*

begin

fun *T* :: (state,act,out) trans ⇒ bool

where *T trn* = *False*

Friendship updates form an alternating sequence of friending and unfriending, and every successful friend creation is preceded by one or two friendship requests.

fun *validValSeq* :: value list ⇒ bool ⇒ bool ⇒ bool ⇒ bool **where**

validValSeq [] - - - = *True*

| *validValSeq* (FRVal *U1 req* # *vl*) *st r1 r2* ↔ (¬*st*) ∧ (¬*r1*) ∧ *validValSeq vl st True r2*

| *validValSeq* (FRVal *U2 req* # *vl*) *st r1 r2* ↔ (¬*st*) ∧ (¬*r2*) ∧ *validValSeq vl st r1 True*

| *validValSeq* (FVal *True* # *vl*) *st r1 r2* ↔ (¬*st*) ∧ (*r1* ∨ *r2*) ∧ *validValSeq vl True False False*

| *validValSeq* (FVal *False* # *vl*) *st r1 r2* ↔ *st* ∧ (¬*r1*) ∧ (¬*r2*) ∧ *validValSeq vl False False False*

| *validValSeq* (OVal *True* # *vl*) *st r1 r2* ↔ *validValSeq vl st r1 r2*

| *validValSeq* (OVal *False* # *vl*) *st r1 r2* ↔ *validValSeq vl st r1 r2*

abbreviation *validValSeqFrom* :: value list ⇒ state ⇒ bool

where *validValSeqFrom vl s*

≡ *validValSeq vl (friends12 s) (UID1 ∈∈ pendingFReqs s UID2) (UID2 ∈∈ pendingFReqs s UID1)*

With respect to the friendship status updates, we use the same “while-or-last-before” bound as for friendship status confidentiality.

inductive *BO* :: value list ⇒ value list ⇒ bool

and *BC* :: value list ⇒ value list ⇒ bool

where

$BO\text{-}FVal[simp, intro!]:$
 $BO (map\ FVal\ fs) (map\ FVal\ fs)$
 $|BO\text{-}BC[intro]:$
 $BC\ vl\ vl1 \implies$
 $BO (map\ FVal\ fs @ OVal\ False\ \# vl) (map\ FVal\ fs @ OVal\ False\ \# vl1)$

 $|BC\text{-}FVal[simp, intro!]:$
 $BC (map\ FVal\ fs) (map\ FVal\ fs1)$
 $|BC\text{-}BO[intro]:$
 $BO\ vl\ vl1 \implies (fs = [] \longleftrightarrow fs1 = []) \implies (fs \neq [] \implies last\ fs = last\ fs1) \implies$
 $BC (map\ FVal\ fs @ OVal\ True\ \# vl)$
 $(map\ FVal\ fs1 @ OVal\ True\ \# vl1)$

Taking into account friendship requests, two value sequences vl and $vl1$ are in the bound if

- $vl1$ (with friendship requests) forms a valid value sequence,
- vl and $vl1$ are in BO (without friendship requests),
- $vl1$ is empty if vl is empty, and
- $vl1$ begins with $OVal\ False$ if vl begins with $OVal\ False$.

The last two points are due to the fact that $UID1$ and $UID1$ might not exist yet if vl is empty (or before $OVal\ False$), in which case the observer can deduce that no friendship request has happened yet.

definition $B\ vl\ vl1 \equiv BO (filter\ (Not\ o\ isFRVal)\ vl) (filter\ (Not\ o\ isFRVal)\ vl1)$
 \wedge
 $validValSeqFrom\ vl1\ ystate \wedge$
 $(vl = [] \longrightarrow vl1 = []) \wedge$
 $(vl \neq [] \wedge hd\ vl = OVal\ False \longrightarrow vl1 \neq [] \wedge hd\ vl1 = OVal$
 $False)$

lemma $BO\text{-}Nil\text{-}iff: BO\ vl\ vl1 \implies vl = [] \longleftrightarrow vl1 = []$
 $\langle proof \rangle$

sublocale $BD\text{-}Security\text{-}IO$ **where**
 $ystate = ystate$ **and** $step = step$ **and**
 $\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$
 $\langle proof \rangle$

8.3 Unwinding proof

lemma $validFrom\text{-}validValSeq:$
assumes $validFrom\ s\ tr$
and $reach\ s$

shows $\text{validValSeqFrom } (V \text{ tr}) \ s$
 $\langle \text{proof} \rangle$

lemma $\text{validFrom } \text{istate } \text{tr} \implies \text{validValSeqFrom } (V \text{ tr}) \ \text{istate}$
 $\langle \text{proof} \rangle$

lemma *produce-FRVal*:
assumes $\text{rs}: \text{reach } s$
and $\text{IDs}: \text{IDsOK } s \ [UID1, UID2] \ [] \ [] \ []$
and $\text{vVS}: \text{validValSeqFrom } (FRVal \ u \ \text{req} \ \# \ \text{vl}) \ s$
obtains $a \ \text{uid} \ \text{uid}' \ s'$
where $\text{step } s \ a = (\text{outOK}, s')$
and $a = \text{Cact } (cFriendReq \ \text{uid} \ (\text{pass } s \ \text{uid}) \ \text{uid}' \ \text{req})$
and $\text{uid} = UID1 \wedge \text{uid}' = UID2 \vee \text{uid} = UID2 \wedge \text{uid}' = UID1$
and $\varphi \ (\text{Trans } s \ a \ \text{outOK } s')$
and $f \ (\text{Trans } s \ a \ \text{outOK } s') = FRVal \ u \ \text{req}$
and $\text{validValSeqFrom } \text{vl} \ s'$
 $\langle \text{proof} \rangle$

lemma *toggle-friends12-True*:
assumes $\text{rs}: \text{reach } s$
and $\text{IDs}: \text{IDsOK } s \ [UID1, UID2] \ [] \ [] \ []$
and $\text{nf12}: \neg \text{friends12 } s$
and $\text{vVS}: \text{validValSeqFrom } (FVal \ \text{True} \ \# \ \text{vl}) \ s$
obtains $a \ \text{uid} \ \text{uid}' \ s'$
where $\text{step } s \ a = (\text{outOK}, s')$
and $a = \text{Cact } (cFriend \ \text{uid} \ (\text{pass } s \ \text{uid}) \ \text{uid}')$
and $s' = \text{createFriend } s \ UID1 \ (\text{pass } s \ UID1) \ UID2$
and $\text{uid} = UID1 \wedge \text{uid}' = UID2 \vee \text{uid} = UID2 \wedge \text{uid}' = UID1$
and $\text{friends12 } s'$
and $\text{eqButUID } s \ s'$
and $\varphi \ (\text{Trans } s \ a \ \text{outOK } s')$
and $f \ (\text{Trans } s \ a \ \text{outOK } s') = FVal \ \text{True}$
and $\neg \gamma \ (\text{Trans } s \ a \ \text{outOK } s')$
and $\text{validValSeqFrom } \text{vl} \ s'$
 $\langle \text{proof} \rangle$

lemma *toggle-friends12-False*:
assumes $\text{rs}: \text{reach } s$
and $\text{IDs}: \text{IDsOK } s \ [UID1, UID2] \ [] \ [] \ []$
and $\text{f12}: \text{friends12 } s$
and $\text{vVS}: \text{validValSeqFrom } (FVal \ \text{False} \ \# \ \text{vl}) \ s$
obtains $a \ s'$
where $\text{step } s \ a = (\text{outOK}, s')$
and $a = \text{Dact } (dFriend \ UID1 \ (\text{pass } s \ UID1) \ UID2)$
and $s' = \text{deleteFriend } s \ UID1 \ (\text{pass } s \ UID1) \ UID2$
and $\neg \text{friends12 } s'$
and $\text{eqButUID } s \ s'$

and φ (*Trans* s a *outOK* s')
and f (*Trans* s a *outOK* s') = *FVal False*
and $\neg\gamma$ (*Trans* s a *outOK* s')
and *validValSeqFrom* vl s'
<proof>

lemma *toggle-friends12*:
assumes rs : *reach* s
and IDs : *IDsOK* s [*UID1*, *UID2*] [] [] []
and $f12$: *friends12* $s \neq fv$
and vVS : *validValSeqFrom* (*FVal* $fv \# vl$) s
obtains a s'
where *step* s a = (*outOK*, s')
and *friends12* $s' = fv$
and *eqButUID* s s'
and φ (*Trans* s a *outOK* s')
and f (*Trans* s a *outOK* s') = *FVal fv*
and $\neg\gamma$ (*Trans* s a *outOK* s')
and *validValSeqFrom* vl s'
<proof>

lemma *BO-cases*:
assumes BO vl $vl1$
obtains (*Nil*) $vl = []$ **and** $vl1 = []$
| (*FVal*) fv $vl' vl1'$ **where** $vl = FVal$ $fv \# vl'$ **and** $vl1 = FVal$ $fv \# vl1'$ **and**
BO $vl' vl1'$
| (*OVal*) $vl' vl1'$ **where** $vl = OVal$ *False* $\# vl'$ **and** $vl1 = OVal$ *False* $\# vl1'$
and BC $vl' vl1'$
<proof>

lemma *BC-cases*:
assumes BC vl $vl1$
obtains (*Nil*) $vl = []$ **and** $vl1 = []$
| (*FVal*) fv fs **where** $vl = FVal$ $fv \# \text{map } FVal$ fs **and** $vl1 = []$
| (*FVal1*) fv fs $fs1$ **where** $vl = \text{map } FVal$ fs **and** $vl1 = FVal$ $fv \# \text{map } FVal$
 $fs1$
| (*BO-FVal*) fv fv' fs $vl' vl1'$ **where** $vl = FVal$ $fv \# \text{map } FVal$ fs @ *FVal* fv'
 $\# OVal$ *True* $\# vl'$
and $vl1 = FVal$ $fv' \# OVal$ *True* $\# vl1'$ **and** BO
 $vl' vl1'$
| (*BO-FVal1*) fv fv' fs $fs1$ $vl' vl1'$ **where** $vl = \text{map } FVal$ fs @ *FVal* fv' $\# OVal$
True $\# vl'$
and $vl1 = FVal$ $fv \# \text{map } FVal$ $fs1$ @ *FVal* $fv' \#$
 $OVal$ *True* $\# vl1'$
and BO $vl' vl1'$
| (*FVal-BO*) fv $vl' vl1'$ **where** $vl = FVal$ $fv \# OVal$ *True* $\# vl'$
and $vl1 = FVal$ $fv \# OVal$ *True* $\# vl1'$ **and** BO $vl' vl1'$
| (*OVal*) $vl' vl1'$ **where** $vl = OVal$ *True* $\# vl'$ **and** $vl1 = OVal$ *True* $\# vl1'$

and $BO\ vl'\ vl1'$
 $\langle proof \rangle$

definition $\Delta 0 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta 0\ s\ vl\ s1\ vl1 \equiv$
 $s = s1 \wedge B\ vl\ vl1 \wedge open\ s \wedge (\neg IDsOK\ s\ [UID1,\ UID2])\ []\ []\ []$

definition $\Delta 1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta 1\ s\ vl\ s1\ vl1 \equiv$
 $eqButUID\ s\ s1 \wedge friendIDs\ s = friendIDs\ s1 \wedge open\ s \wedge$
 $BO\ (filter\ (Not\ o\ isFRVal)\ vl)\ (filter\ (Not\ o\ isFRVal)\ vl1) \wedge$
 $validValSeqFrom\ vl1\ s1 \wedge$
 $IDsOK\ s1\ [UID1,\ UID2]\ []\ []\ []$

definition $\Delta 2 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta 2\ s\ vl\ s1\ vl1 \equiv (\exists fs\ fs1.$
 $eqButUID\ s\ s1 \wedge \neg open\ s \wedge$
 $validValSeqFrom\ vl1\ s1 \wedge$
 $filter\ (Not\ o\ isFRVal)\ vl = map\ FVal\ fs \wedge$
 $filter\ (Not\ o\ isFRVal)\ vl1 = map\ FVal\ fs1)$

definition $\Delta 3 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
 $\Delta 3\ s\ vl\ s1\ vl1 \equiv (\exists fs\ fs1\ vlr\ vlr1.$
 $eqButUID\ s\ s1 \wedge \neg open\ s \wedge BO\ vlr\ vlr1 \wedge$
 $validValSeqFrom\ vl1\ s1 \wedge$
 $(fs = [] \longleftrightarrow fs1 = []) \wedge$
 $(fs \neq [] \longrightarrow last\ fs = last\ fs1) \wedge$
 $(fs = [] \longrightarrow friendIDs\ s = friendIDs\ s1) \wedge$
 $filter\ (Not\ o\ isFRVal)\ vl = map\ FVal\ fs\ @\ Oval\ True\ \#\ vlr \wedge$
 $filter\ (Not\ o\ isFRVal)\ vl1 = map\ FVal\ fs1\ @\ Oval\ True\ \#\ vlr1)$

lemma $\Delta 2-I:$

assumes $eqButUID\ s\ s1\ \neg open\ s$
 $validValSeqFrom\ vl1\ s1$
 $filter\ (Not\ o\ isFRVal)\ vl = map\ FVal\ fs$
 $filter\ (Not\ o\ isFRVal)\ vl1 = map\ FVal\ fs1$
shows $\Delta 2\ s\ vl\ s1\ vl1$
 $\langle proof \rangle$

lemma $\Delta 3-I:$

assumes $eqButUID\ s\ s1\ \neg open\ s\ BO\ vlr\ vlr1$
 $validValSeqFrom\ vl1\ s1$
 $fs = [] \longleftrightarrow fs1 = []\ fs \neq [] \longrightarrow last\ fs = last\ fs1$
 $fs = [] \longrightarrow friendIDs\ s = friendIDs\ s1$
 $filter\ (Not\ o\ isFRVal)\ vl = map\ FVal\ fs\ @\ Oval\ True\ \#\ vlr$
 $filter\ (Not\ o\ isFRVal)\ vl1 = map\ FVal\ fs1\ @\ Oval\ True\ \#\ vlr1$
shows $\Delta 3\ s\ vl\ s1\ vl1$

<proof>

lemma *istate- $\Delta 0$* :
assumes *B: B vl vl1*
shows *$\Delta 0$ istate vl istate vl1*
<proof>

lemma *unwind-cont- $\Delta 0$* : *unwind-cont $\Delta 0$ { $\Delta 0, \Delta 1, \Delta 2, \Delta 3$ }*
<proof>

lemma *unwind-cont- $\Delta 1$* : *unwind-cont $\Delta 1$ { $\Delta 1, \Delta 2, \Delta 3$ }*
<proof>

lemma *unwind-cont- $\Delta 2$* : *unwind-cont $\Delta 2$ { $\Delta 2, \Delta 1$ }*
<proof>

lemma *unwind-cont- $\Delta 3$* : *unwind-cont $\Delta 3$ { $\Delta 3, \Delta 1$ }*
<proof>

definition *Gr* where

Gr =
{
 ($\Delta 0$, { $\Delta 0, \Delta 1, \Delta 2, \Delta 3$ }),
 ($\Delta 1$, { $\Delta 1, \Delta 2, \Delta 3$ }),
 ($\Delta 2$, { $\Delta 2, \Delta 1$ }),
 ($\Delta 3$, { $\Delta 3, \Delta 1$ })
}

theorem *secure: secure*
<proof>

end

end

theory *Friend-Request-Network*

imports

../API-Network

Friend-Request

BD-Security-Compositional.Composing-Security-Network

begin

8.4 Confidentiality for the N-ary composition

locale *FriendRequestNetwork* = *Network* + *FriendNetworkObservationSetup* +
fixes
AID :: *apiID*
and
UID1 :: *userID*
and
UID2 :: *userID*
assumes
UID1-UID2-UIDs: $\{UID1, UID2\} \cap (UIDs\ AID) = \{\}$
and
UID1-UID2: $UID1 \neq UID2$
and
AID-AIDs: $AID \in AIDs$
begin

sublocale *Issuer*: *Friend* *UIDs* *AID* *UID1* *UID2* $\langle proof \rangle$

abbreviation φ :: *apiID* \Rightarrow (*state*, *act*, *out*) *trans* \Rightarrow *bool*
where $\varphi\ aid\ trn \equiv (Issuer.\varphi\ trn \wedge aid = AID)$

abbreviation f :: *apiID* \Rightarrow (*state*, *act*, *out*) *trans* \Rightarrow *Friend.value*
where $f\ aid\ trn \equiv Friend.f\ UID1\ UID2\ trn$

abbreviation T :: *apiID* \Rightarrow (*state*, *act*, *out*) *trans* \Rightarrow *bool*
where $T\ aid\ trn \equiv False$

abbreviation B :: *apiID* \Rightarrow *Friend.value\ list* \Rightarrow *Friend.value\ list* \Rightarrow *bool*
where $B\ aid\ vl\ vl1 \equiv (if\ aid = AID\ then\ Issuer.B\ vl\ vl1\ else\ (vl = [] \wedge vl1 = []))$

abbreviation $comOfV\ aid\ vl \equiv Internal$
abbreviation $tgtNodeOfV\ aid\ vl \equiv undefined$
abbreviation $syncV\ aid1\ vl1\ aid2\ vl2 \equiv False$

lemma [*simp*]: $validTrans\ aid\ trn \Longrightarrow lreach\ aid\ (srcOf\ trn) \Longrightarrow \varphi\ aid\ trn \Longrightarrow comOf\ aid\ trn = Internal$
 $\langle proof \rangle$

sublocale *Net*: *BD-Security-TS-Network-getTgtV*
where *istate* = $\lambda\cdot. istate$ **and** *validTrans* = *validTrans* **and** *srcOf* = $\lambda\cdot. srcOf$
and *tgtOf* = $\lambda\cdot. tgtOf$
and *nodes* = *AIDs* **and** *comOf* = *comOf* **and** *tgtNodeOf* = *tgtNodeOf*
and *sync* = *sync* **and** $\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and**
 $B = B$
and *comOfV* = *comOfV* **and** *tgtNodeOfV* = *tgtNodeOfV* **and** *syncV* = *syncV*
and *comOfO* = *comOfO* **and** *tgtNodeOfO* = *tgtNodeOfO* **and** *syncO* = *syncO*
and *source* = *AID* **and** *getTgtV* = *id*
 $\langle proof \rangle$

```

sublocale BD-Security-TS-Network-Preserve-Source-Security-getTgtV
where istate =  $\lambda$ -. istate and validTrans = validTrans and srcOf =  $\lambda$ -. srcOf
and tgtOf =  $\lambda$ -. tgtOf
  and nodes = AIDs and comOf = comOf and tgtNodeOf = tgtNodeOf
  and sync = sync and  $\varphi$  =  $\varphi$  and f = f and  $\gamma$  =  $\gamma$  and g = g and T = T and
  B = B
  and comOfV = comOfV and tgtNodeOfV = tgtNodeOfV and syncV = syncV
  and comOfO = comOfO and tgtNodeOfO = tgtNodeOfO and syncO = syncO
  and source = AID and getTgtV = id
   $\langle$ proof $\rangle$ 

theorem secure: secure
   $\langle$ proof $\rangle$ 

end

end
theory Friend-Request-All
imports Friend-Request-Network
begin

end
theory Outer-Friend-Intro
  imports ../Safety-Properties
begin

```

9 Remote (outer) friendship status confidentiality

We verify the following property, which is specific to CosMeDis, in that it does not have a CoSMed counterpart: Given a coalition consisting of groups of users *UIDs* *j* from multiple nodes *j* and a user *UID* at some node *i* not in these groups,

the coalition may learn about the *occurrence* of remote friendship actions of *UID* (because network traffic is assumed to be observable),

but they learn nothing about the *content* (who was added or deleted as a friend) of remote friendship actions between *UID* and remote users who are not in the coalition

beyond what everybody knows, namely that, with respect to each other user *uid'*, those actions form an alternating sequence of friending and unfriending, unless a user in *UIDs* *i* becomes a local friend of *UID*.

Similarly to the other properties, this property is proved using the system compositionality and transport theorems for BD security.

Note that, unlike inner friendship, outer friendship is not necessarily sym-

metric. It is always established from a user of a server to a user of a client, the former giving the latter unilateral access to his friend-only posts. These unilateral friendship permissions are stored on the client.

When proving the single-node BD security properties, the bound refers to outer friendship-status changes issued by the user UID concerning friending or unfriending some user UID' located at a node j different from i . Such changes occur as communicating actions between the “secret issuer” node i and the “secret receiver” nodes j .

```

end
theory Outer-Friend
  imports Outer-Friend-Intro
begin

```

```

type-synonym obs = act * out

```

The observers $UIDs$ j are an arbitrary, but fixed sets of users at each node j of the network, and the secret is the friendship information of user UID at node AID .

```

locale OuterFriend =
fixes UIDs :: apiID  $\Rightarrow$  userID set
and AID :: apiID
and UID :: userID
assumes UID-UIDs:  $UID \notin UIDs$  AID
and emptyUserID-not-UIDs:  $\bigwedge aid. emptyUserID \notin UIDs$  aid

```

```

datatype value =
  isFrVal: FrVal apiID userID bool — updates to the friendship status of UID
| isOVal: OVal bool — a change in the “openness” status of the UID friendship info

```

```

end
theory Outer-Friend-Issuer-Observation-Setup
  imports ../Outer-Friend
begin

```

9.1 Issuer node

9.1.1 Observation setup

We now consider the network node AID , at which the user UID is registered, whose remote friends are to be kept confidential.

```

locale OuterFriendIssuer = OuterFriend
begin

```

```

fun  $\gamma$  :: (state,act,out) trans  $\Rightarrow$  bool where
 $\gamma$  (Trans - a ou -)  $\longleftrightarrow$  ( $\exists uid. userOfA$  a = Some uid  $\wedge uid \in UIDs$  AID)  $\vee$ 
  ( $\exists ca. a = COMact$  ca  $\wedge ou \neq outErr$ )

```

Purging communicating actions: password information is removed, the user IDs of friends added or deleted by UID are removed, and the information whether UID added or deleted a friend is removed

fun $comPurge :: comActt \Rightarrow comActt$ **where**
 $comPurge (comSendServerReq uID p aID reqInfo) = comSendServerReq uID emptyPass aID reqInfo$
 $| comPurge (comReceiveClientReq aID reqInfo) = comReceiveClientReq aID reqInfo$
 $| comPurge (comConnectClient uID p aID sp) = comConnectClient uID emptyPass aID sp$
 $| comPurge (comConnectServer aID sp) = comConnectServer aID sp$
 $| comPurge (comReceivePost aID sp nID nt uID v) = comReceivePost aID sp nID nt uID v$
 $| comPurge (comSendPost uID p aID nID) = comSendPost uID emptyPass aID nID$
 $| comPurge (comSendCreateOFriend uID p aID uID') =$
 $\quad (if\ uID = UID \wedge uID' \notin UIDs\ aID$
 $\quad \quad then\ comSendCreateOFriend\ uID\ emptyPass\ aID\ emptyUserID$
 $\quad \quad else\ comSendCreateOFriend\ uID\ emptyPass\ aID\ uID')$
 $| comPurge (comReceiveCreateOFriend aID cp uID uID') = comReceiveCreateOFriend aID cp uID uID'$
 $| comPurge (comSendDeleteOFriend uID p aID uID') =$
 $\quad (if\ uID = UID \wedge uID' \notin UIDs\ aID$
 $\quad \quad then\ comSendCreateOFriend\ uID\ emptyPass\ aID\ emptyUserID$
 $\quad \quad else\ comSendDeleteOFriend\ uID\ emptyPass\ aID\ uID')$
 $| comPurge (comReceiveDeleteOFriend aID cp uID uID') = comReceiveDeleteOFriend aID cp uID uID'$

lemma $comPurge$ -simps:

$comPurge\ ca = comSendServerReq\ uID\ p\ aID\ reqInfo \longleftrightarrow (\exists p'. ca = comSendServerReq\ uID\ p'\ aID\ reqInfo \wedge p = emptyPass)$
 $comPurge\ ca = comReceiveClientReq\ aID\ reqInfo \longleftrightarrow ca = comReceiveClientReq\ aID\ reqInfo$
 $comPurge\ ca = comConnectClient\ uID\ p\ aID\ sp \longleftrightarrow (\exists p'. ca = comConnectClient\ uID\ p'\ aID\ sp \wedge p = emptyPass)$
 $comPurge\ ca = comConnectServer\ aID\ sp \longleftrightarrow ca = comConnectServer\ aID\ sp$
 $comPurge\ ca = comReceivePost\ aID\ sp\ nID\ nt\ uID\ v \longleftrightarrow ca = comReceivePost\ aID\ sp\ nID\ nt\ uID\ v$
 $comPurge\ ca = comSendPost\ uID\ p\ aID\ nID \longleftrightarrow (\exists p'. ca = comSendPost\ uID\ p'\ aID\ nID \wedge p = emptyPass)$
 $comPurge\ ca = comSendCreateOFriend\ uID\ p\ aID\ uID'$
 $\longleftrightarrow (\exists p'\ uid''. (ca = comSendCreateOFriend\ uID\ p'\ aID\ uid'' \vee ca = comSendDeleteOFriend\ uID\ p'\ aID\ uid'') \wedge uID = UID \wedge uid'' \notin UIDs\ aID \wedge uID' = emptyUserID \wedge p = emptyPass)$
 $\vee (\exists p'. ca = comSendCreateOFriend\ uID\ p'\ aID\ uID' \wedge \neg(uID = UID \wedge uID' \notin UIDs\ aID) \wedge p = emptyPass)$
 $comPurge\ ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uID'$
 $comPurge\ ca = comSendDeleteOFriend\ uID\ p\ aID\ uID' \longleftrightarrow (\exists p'. ca = comSendDeleteOFriend\ uID\ p'\ aID\ uID' \wedge \neg(uID = UID \wedge uID' \notin UIDs\ aID) \wedge p = emptyPass)$

$comPurge\ ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'$
 ⟨proof⟩

Purging outputs: the user IDs of friends added or deleted by UID are removed from outer friend creation and deletion outputs.

fun $outPurge :: out \Rightarrow out$ **where**
 $outPurge\ (O-sendCreateOFriend\ (aID,\ sp,\ uID,\ uID')) =$
 (if $uID = UID \wedge uID' \notin UIDs\ aID$
 then $O-sendCreateOFriend\ (aID,\ sp,\ uID,\ emptyUserID)$
 else $O-sendCreateOFriend\ (aID,\ sp,\ uID,\ uID')$)
 $| outPurge\ (O-sendDeleteOFriend\ (aID,\ sp,\ uID,\ uID')) =$
 (if $uID = UID \wedge uID' \notin UIDs\ aID$
 then $O-sendCreateOFriend\ (aID,\ sp,\ uID,\ emptyUserID)$
 else $O-sendDeleteOFriend\ (aID,\ sp,\ uID,\ uID')$)
 $| outPurge\ ou = ou$

lemma $outPurge-simps[simp]$:

$outPurge\ ou = outErr \longleftrightarrow ou = outErr$
 $outPurge\ ou = outOK \longleftrightarrow ou = outOK$
 $outPurge\ ou = O-sendServerReq\ ossr \longleftrightarrow ou = O-sendServerReq\ ossr$
 $outPurge\ ou = O-connectClient\ occ \longleftrightarrow ou = O-connectClient\ occ$
 $outPurge\ ou = O-sendPost\ osn \longleftrightarrow ou = O-sendPost\ osn$
 $outPurge\ ou = O-sendCreateOFriend\ (aID,\ sp,\ uID,\ uID')$
 $\longleftrightarrow (\exists\ uid''. (ou = O-sendCreateOFriend\ (aID,\ sp,\ uID,\ uid'') \vee ou = O-sendDeleteOFriend$
 $(aID,\ sp,\ uID,\ uid'')) \wedge uID = UID \wedge uid'' \notin UIDs\ aID \wedge uID' = emptyUserID$
 $\vee (ou = O-sendCreateOFriend\ (aID,\ sp,\ uID,\ uID') \wedge \neg(uID = UID \wedge uID'$
 $\notin UIDs\ aID)))$
 $outPurge\ ou = O-sendDeleteOFriend\ (aID,\ sp,\ uID,\ uID')$
 $\longleftrightarrow (ou = O-sendDeleteOFriend\ (aID,\ sp,\ uID,\ uID') \wedge \neg(uID = UID \wedge uID' \notin$
 $UIDs\ aID))$
 ⟨proof⟩

fun $g :: (state,act,out)trans \Rightarrow obs$ **where**

$g\ (Trans - (COMact\ ca)\ ou\ -) = (COMact\ (comPurge\ ca),\ outPurge\ ou)$
 $| g\ (Trans - a\ ou\ -) = (a,ou)$

lemma $g-simps$:

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendServerReq\ uID\ p\ aID\ reqInfo),\ O-sendServerReq$
 $ossr)$
 $\longleftrightarrow (\exists\ p'. a = COMact\ (comSendServerReq\ uID\ p'\ aID\ reqInfo) \wedge p = emptyPass$
 $\wedge ou = O-sendServerReq\ ossr)$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveClientReq\ aID\ reqInfo),\ outOK)$
 $\longleftrightarrow a = COMact\ (comReceiveClientReq\ aID\ reqInfo) \wedge ou = outOK$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectClient\ uID\ p\ aID\ sp),\ O-connectClient$
 $occ)$
 $\longleftrightarrow (\exists\ p'. a = COMact\ (comConnectClient\ uID\ p'\ aID\ sp) \wedge p = emptyPass \wedge$
 $ou = O-connectClient\ occ)$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectServer\ aID\ sp),\ outOK)$

```

 $\longleftrightarrow a = \text{COMact } (\text{comConnectServer } aID \text{ sp}) \wedge ou = \text{outOK}$ 
   $g (\text{Trans } s \ a \ ou \ s') = (\text{COMact } (\text{comReceivePost } aID \ sp \ nID \ nt \ uID \ v), \text{outOK})$ 
 $\longleftrightarrow a = \text{COMact } (\text{comReceivePost } aID \ sp \ nID \ nt \ uID \ v) \wedge ou = \text{outOK}$ 
   $g (\text{Trans } s \ a \ ou \ s') = (\text{COMact } (\text{comSendPost } uID \ p \ aID \ nID), \text{O-sendPost } osn)$ 
 $\longleftrightarrow (\exists p'. a = \text{COMact } (\text{comSendPost } uID \ p' \ aID \ nID) \wedge p = \text{emptyPass} \wedge ou = \text{O-sendPost } osn)$ 
   $g (\text{Trans } s \ a \ ou \ s') = (\text{COMact } (\text{comSendCreateOFriend } uID \ p \ aID \ uID'), \text{O-sendCreateOFriend } (aid, sp, uid, uid'))$ 
 $\longleftrightarrow ((\exists p' \ uid''. (a = \text{COMact } (\text{comSendCreateOFriend } uID \ p' \ aID \ uid'') \vee a = \text{COMact } (\text{comSendDeleteOFriend } uID \ p' \ aID \ uid'')) \wedge uID = \text{UID} \wedge uid'' \notin \text{UIDs } aID \wedge uID' = \text{emptyUserID} \wedge p = \text{emptyPass})$ 
   $\vee (\exists p'. a = \text{COMact } (\text{comSendCreateOFriend } uID \ p' \ aID \ uID') \wedge \neg(uID = \text{UID} \wedge uID' \notin \text{UIDs } aID) \wedge p = \text{emptyPass}))$ 
   $\wedge ((\exists uid''. (ou = \text{O-sendCreateOFriend } (aid, sp, uid, uid'') \vee ou = \text{O-sendDeleteOFriend } (aid, sp, uid, uid'')) \wedge uid = \text{UID} \wedge uid'' \notin \text{UIDs } aid \wedge uid' = \text{emptyUserID})$ 
   $\vee (ou = \text{O-sendCreateOFriend } (aid, sp, uid, uid') \wedge \neg(uid = \text{UID} \wedge uid' \notin \text{UIDs } aid)))$ 
   $g (\text{Trans } s \ a \ ou \ s') = (\text{COMact } (\text{comReceiveCreateOFriend } aID \ cp \ uID \ uID'), \text{outOK})$ 
 $\longleftrightarrow a = \text{COMact } (\text{comReceiveCreateOFriend } aID \ cp \ uID \ uID') \wedge ou = \text{outOK}$ 
   $g (\text{Trans } s \ a \ ou \ s') = (\text{COMact } (\text{comSendDeleteOFriend } uID \ p \ aID \ uID'), \text{O-sendDeleteOFriend } (aid, sp, uid, uid'))$ 
 $\longleftrightarrow (\exists p'. a = \text{COMact } (\text{comSendDeleteOFriend } uID \ p' \ aID \ uID') \wedge \neg(uID = \text{UID} \wedge uID' \notin \text{UIDs } aID) \wedge p = \text{emptyPass})$ 
   $\wedge (ou = \text{O-sendDeleteOFriend } (aid, sp, uid, uid') \wedge \neg(uid = \text{UID} \wedge uid' \notin \text{UIDs } aid))$ 
   $g (\text{Trans } s \ a \ ou \ s') = (\text{COMact } (\text{comReceiveDeleteOFriend } aID \ cp \ uID \ uID'), \text{outOK})$ 
 $\longleftrightarrow a = \text{COMact } (\text{comReceiveDeleteOFriend } aID \ cp \ uID \ uID') \wedge ou = \text{outOK}$ 
   $\langle \text{proof} \rangle$ 

```

end

end

theory *Outer-Friend-Issuer-State-Indistinguishability*

imports *Outer-Friend-Issuer-Observation-Setup*

begin

9.1.2 Unwinding helper definitions and lemmas

context *OuterFriendIssuer*

begin

fun *filterUIDs* :: $(\text{apiID} \times \text{userID}) \text{ list} \Rightarrow (\text{apiID} \times \text{userID}) \text{ list}$ **where**
filterUIDs *awidl* = *filter* $(\lambda \text{auid}. (\text{snd } \text{auid}) \in \text{UIDs } (\text{fst } \text{auid}))$ *awidl*

fun *removeUIDs* :: $(\text{apiID} \times \text{userID}) \text{ list} \Rightarrow (\text{apiID} \times \text{userID}) \text{ list}$ **where**

$removeUIDs\ auidl = filter\ (\lambda auid.\ (snd\ auid) \notin\ UIDs\ (fst\ auid))\ auidl$

fun $eqButUIDs :: (apiID \times userID)\ list \Rightarrow (apiID \times userID)\ list \Rightarrow bool$ **where**
 $eqButUIDs\ uidl\ uidl1 = (filterUIDs\ uidl = filterUIDs\ uidl1)$

lemma $eqButUIDs-eq[simp,intro!]$: $eqButUIDs\ uidl\ uidl$
<proof>

lemma $eqButUIDs-sym$:
assumes $eqButUIDs\ uidl\ uidl1$
shows $eqButUIDs\ uidl1\ uidl$
<proof>

lemma $eqButUIDs-trans$:
assumes $eqButUIDs\ uidl\ uidl1$ **and** $eqButUIDs\ uidl1\ uidl2$
shows $eqButUIDs\ uidl\ uidl2$
<proof>

lemma $eqButUIDs-remove1-cong$:
assumes $eqButUIDs\ uidl\ uidl1$
shows $eqButUIDs\ (remove1\ auid\ uidl)\ (remove1\ auid\ uidl1)$
<proof>

lemma $eqButUIDs-snoc-cong$:
assumes $eqButUIDs\ uidl\ uidl1$
shows $eqButUIDs\ (uidl\ \#\#\ auid')\ (uidl1\ \#\#\ auid')$
<proof>

definition $eqButUIDf$ **where**
 $eqButUIDf\ frds\ frds1 \equiv$
 $eqButUIDs\ (frds\ UID)\ (frds1\ UID)$
 $\wedge (\forall\ uid.\ uid \neq\ UID \longrightarrow frds\ uid = frds1\ uid)$

lemmas $eqButUIDf-intro = eqButUIDf-def[THEN\ meta-eq-to-obj-eq,\ THEN\ iffD2]$

lemma $eqButUIDf-eeq[simp,intro!]$: $eqButUIDf\ frds\ frds$
<proof>

lemma $eqButUIDf-sym$:
assumes $eqButUIDf\ frds\ frds1$ **shows** $eqButUIDf\ frds1\ frds$
<proof>

lemma $eqButUIDf-trans$:
assumes $eqButUIDf\ frds\ frds1$ **and** $eqButUIDf\ frds1\ frds2$
shows $eqButUIDf\ frds\ frds2$

<proof>

lemma *eqButUIDf-cong*:

assumes *eqButUIDf frds frds1*

and *uid ≠ UID ⇒ uu = uu1*

and *uid = UID ⇒ eqButUIDs uu uu1*

shows *eqButUIDf (frds (uid := uu)) (frds1 (uid := uu1))*

<proof>

lemma *eqButUIDf-not-UID*:

$\llbracket \text{eqButUIDf frds frds1; uid} \neq \text{UID} \rrbracket \Rightarrow \text{frds uid} = \text{frds1 uid}$

<proof>

definition *eqButUID* :: *state ⇒ state ⇒ bool where*

eqButUID s s1 ≡

admin s = admin s1 ∧

pendingUReqs s = pendingUReqs s1 ∧ userReq s = userReq s1 ∧
userIDs s = userIDs s1 ∧ user s = user s1 ∧ pass s = pass s1 ∧

pendingFReqs s = pendingFReqs s1 ∧
friendReq s = friendReq s1 ∧
friendIDs s = friendIDs s1 ∧

postIDs s = postIDs s1 ∧ admin s = admin s1 ∧
post s = post s1 ∧ vis s = vis s1 ∧
owner s = owner s1 ∧

pendingSApiReqs s = pendingSApiReqs s1 ∧ sApiReq s = sApiReq s1 ∧
serverApiIDs s = serverApiIDs s1 ∧ serverPass s = serverPass s1 ∧
outerPostIDs s = outerPostIDs s1 ∧ outerPost s = outerPost s1 ∧ outerVis s =
outerVis s1 ∧
outerOwner s = outerOwner s1 ∧
eqButUIDf (sentOuterFriendIDs s) (sentOuterFriendIDs s1) ∧
recvOuterFriendIDs s = recvOuterFriendIDs s1 ∧

pendingCApiReqs s = pendingCApiReqs s1 ∧ cApiReq s = cApiReq s1 ∧
clientApiIDs s = clientApiIDs s1 ∧ clientPass s = clientPass s1 ∧
sharedWith s = sharedWith s1

lemmas *eqButUID-intro = eqButUID-def[THEN meta-eq-to-obj-eq, THEN iffD2]*

lemma *eqButUID-refl[simp,intro!]*: *eqButUID s s*

<proof>

lemma *eqButUID-sym[sym]*:

assumes *eqButUID s s1* **shows** *eqButUID s1 s*

<proof>

lemma *eqButUID-trans[trans]*:
assumes *eqButUID s s1* **and** *eqButUID s1 s2* **shows** *eqButUID s s2*
 ⟨*proof*⟩

lemma *eqButUID-stateSelectors*:
assumes *eqButUID s s1*
shows *admin s = admin s1*
pendingUReqs s = pendingUReqs s1 *userReq s = userReq s1*
userIDs s = userIDs s1 *user s = user s1* *pass s = pass s1*
pendingFReqs s = pendingFReqs s1
friendReq s = friendReq s1
friendIDs s = friendIDs s1

postIDs s = postIDs s1
post s = post s1 *vis s = vis s1*
owner s = owner s1

pendingSApiReqs s = pendingSApiReqs s1 *sApiReq s = sApiReq s1*
serverApiIDs s = serverApiIDs s1 *serverPass s = serverPass s1*
outerPostIDs s = outerPostIDs s1 *outerPost s = outerPost s1* *outerVis s = outer-*
Vis s1
outerOwner s = outerOwner s1
eqButUIDf (sentOuterFriendIDs s) (sentOuterFriendIDs s1)
recvOuterFriendIDs s = recvOuterFriendIDs s1

pendingCApiReqs s = pendingCApiReqs s1 *cApiReq s = cApiReq s1*
clientApiIDs s = clientApiIDs s1 *clientPass s = clientPass s1*
sharedWith s = sharedWith s1

IDsOK s = IDsOK s1
 ⟨*proof*⟩

lemmas *eqButUID-eqButUIDf = eqButUID-stateSelectors(22)*

lemma *eqButUID-eqButUIDs*:
eqButUID s s1 \implies *eqButUIDs (sentOuterFriendIDs s UID) (sentOuterFriendIDs*
s1 UID)
 ⟨*proof*⟩

lemma *eqButUID-not-UID*:
eqButUID s s1 \implies *uid* \neq *UID* \implies *sentOuterFriendIDs s uid = sentOuterFrien-*
dIDs s1 uid
 ⟨*proof*⟩

lemma *eqButUID-sentOuterFriends-UIDs*:
assumes *eqButUID s s1*
and *uid' \in UIDs aid*

shows $(aid, uid') \in \in sentOuterFriendIDs\ s\ UID \longleftrightarrow (aid, uid') \in \in sentOuterFriendIDs\ s1\ UID$
 ⟨proof⟩

lemma *eqButUID-sentOuterFriendIDs-cong*:

assumes *eqButUID s s1*

and $uid' \notin UIDs\ aid$

shows *eqButUID (s (sentOuterFriendIDs := (sentOuterFriendIDs s) (UID := sentOuterFriendIDs s UID ## (aid, uid')))) s1*

and *eqButUID s (s1 (sentOuterFriendIDs := (sentOuterFriendIDs s1) (UID := sentOuterFriendIDs s1 UID ## (aid, uid'))))*

and *eqButUID s (s1 (sentOuterFriendIDs := (sentOuterFriendIDs s1) (UID := remove1 (aid, uid') (sentOuterFriendIDs s1 UID))))*

and *eqButUID (s (sentOuterFriendIDs := (sentOuterFriendIDs s) (UID := remove1 (aid, uid') (sentOuterFriendIDs s UID)))) s1*
 ⟨proof⟩

lemma *eqButUID-cong*:

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!admin := uu1))\ (s1\ (\!admin := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!pendingUReqs := uu1))\ (s1\ (\!pendingUReqs := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!userReq := uu1))\ (s1\ (\!userReq := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!userIDs := uu1))\ (s1\ (\!userIDs := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!user := uu1))\ (s1\ (\!user := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!pass := uu1))\ (s1\ (\!pass := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!postIDs := uu1))\ (s1\ (\!postIDs := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!owner := uu1))\ (s1\ (\!owner := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!post := uu1))\ (s1\ (\!post := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!vis := uu1))\ (s1\ (\!vis := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!pendingFReqs := uu1))\ (s1\ (\!pendingFReqs := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!friendReq := uu1))\ (s1\ (\!friendReq := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!friendIDs := uu1))\ (s1\ (\!friendIDs := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\!pendingSApiReqs := uu1))\ (s1\ (\!pendingSApiReqs := uu2))$

$:= uu1)) (s1 \langle \text{pendingSApiReqs} := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle sApiReq := uu1 \rangle)$
 $(s1 \langle sApiReq := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle serverApiIDs :=$
 $uu1 \rangle) (s1 \langle serverApiIDs := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle serverPass := uu1 \rangle)$
 $(s1 \langle serverPass := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle outerPostIDs :=$
 $uu1 \rangle) (s1 \langle outerPostIDs := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle outerPost := uu1 \rangle)$
 $(s1 \langle outerPost := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle outerVis := uu1 \rangle)$
 $(s1 \langle outerVis := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle outerOwner :=$
 $uu1 \rangle) (s1 \langle outerOwner := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies eqButUIDf uu1 uu2 \implies eqButUID (s \langle sentOuter-$
 $FriendIDs := uu1 \rangle) (s1 \langle sentOuterFriendIDs := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle recvOuterFriendIDs$
 $:= uu1 \rangle) (s1 \langle recvOuterFriendIDs := uu2 \rangle)$

$\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle pendingCApiReqs$
 $:= uu1 \rangle) (s1 \langle pendingCApiReqs := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle cApiReq := uu1 \rangle)$
 $(s1 \langle cApiReq := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle clientApiIDs :=$
 $uu1 \rangle) (s1 \langle clientApiIDs := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle clientPass := uu1 \rangle)$
 $(s1 \langle clientPass := uu2 \rangle)$
 $\wedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s \langle sharedWith :=$
 $uu1 \rangle) (s1 \langle sharedWith := uu2 \rangle)$
 $\langle proof \rangle$

lemma *distinct-remove1-idem*: $distinct\ xs \implies remove1\ y\ (remove1\ y\ xs) = re-$
 $move1\ y\ xs$
 $\langle proof \rangle$

lemma *eqButUID-step*:
assumes $ss1: eqButUID\ s\ s1$
and $step: step\ s\ a = (ou, s')$
and $step1: step\ s1\ a = (ou1, s1')$
and $rs: reach\ s$
and $rs1: reach\ s1$
shows $eqButUID\ s'\ s1'$
 $\langle proof \rangle$

end

end

```

theory Outer-Friend-Issuer-Openness
  imports Outer-Friend-Issuer-State-Indistinguishability
begin

```

9.1.3 Dynamic declassification trigger

```

context OuterFriendIssuer
begin

```

The dynamic declassification trigger condition holds, i.e. the access window to the confidential information is open, while an observer is a local friend of the user UID .

```

definition open :: state  $\Rightarrow$  bool
where open s  $\equiv$   $\exists$  uid  $\in$   $UIDs$  AID. uid  $\in$  friendIDs s UID

```

lemma open-step-cases:

assumes open s \neq open s'

and step s a = (ou, s')

obtains

(OpenF) uid p uid' **where** a = Cact (cFriend uid p uid') ou = outOK p = pass s uid

$uid \in UIDs$ AID \wedge uid' = UID \vee uid = UID \wedge uid' \in $UIDs$

AID

open s' \neg open s

| (CloseF) uid p uid' **where** a = Dact (dFriend uid p uid') ou = outOK p = pass s uid

$uid \in UIDs$ AID \wedge uid' = UID \vee uid = UID \wedge uid' \in $UIDs$

AID

open s \neg open s'

\langle proof \rangle

lemma COMact-open:

assumes step s a = (ou, s')

and a = COMact ca

shows open s = open s'

\langle proof \rangle

lemma eqButUID-open-eq: eqButUID s s1 \implies open s = open s1

\langle proof \rangle

end

end

theory Outer-Friend-Issuer-Value-Setup

imports Outer-Friend-Issuer-Openness

begin

9.1.4 Value setup

context *OuterFriendIssuer*

begin

fun $\varphi :: (state, act, out) \text{ trans} \Rightarrow \text{bool}$ **where**

$\varphi (Trans\ s\ (COMact\ (comSendCreateOFriend\ uID\ p\ aID\ uID'))\ ou\ s') =$
 $(uID = UID \wedge uID' \notin UIDs\ aID \wedge ou \neq outErr)$

|
 $\varphi (Trans\ s\ (COMact\ (comSendDeleteOFriend\ uID\ p\ aID\ uID'))\ ou\ s') =$
 $(uID = UID \wedge uID' \notin UIDs\ aID \wedge ou \neq outErr)$

|
 $\varphi (Trans\ s\ -\ -\ s') = (open\ s \neq open\ s')$

fun $f :: (state, act, out) \text{ trans} \Rightarrow \text{value}$ **where**

$f (Trans\ s\ (COMact\ (comSendCreateOFriend\ uID\ p\ aID\ uID'))\ ou\ s') = FrVal\ aID\ uID'\ True$

|
 $f (Trans\ s\ (COMact\ (comSendDeleteOFriend\ uID\ p\ aID\ uID'))\ ou\ s') = FrVal\ aID\ uID'\ False$

|
 $f (Trans\ s\ -\ -\ s') = OVal\ (open\ s')$

lemma φE :

assumes $\varphi: \varphi (Trans\ s\ a\ ou\ s')\ (\text{is}\ \varphi\ ?trn)$

and $step: step\ s\ a = (ou,\ s')$

and $rs: reach\ s$

obtains

$(Friend)\ p\ aID\ uID'\ \text{where}\ a = COMact\ (comSendCreateOFriend\ UID\ p\ aID\ uID')\ ou \neq outErr$

$f\ ?trn = FrVal\ aID\ uID'\ True\ uID' \notin UIDs\ aID$

| $(Unfriend)\ p\ aID\ uID'\ \text{where}\ a = COMact\ (comSendDeleteOFriend\ UID\ p\ aID\ uID')\ ou \neq outErr$

$f\ ?trn = FrVal\ aID\ uID'\ False\ uID' \notin UIDs\ aID$

| $(OpenF)\ uid\ p\ uid'\ \text{where}\ a = Cact\ (cFriend\ uid\ p\ uid')\ ou = outOK\ p = pass\ s\ uid$

$uid \in UIDs\ aID \wedge uid' = UID \vee uid = UID \wedge uid' \in UIDs$

AID

$open\ s' \neg open\ s$

$f\ ?trn = OVal\ True$

| $(CloseF)\ uid\ p\ uid'\ \text{where}\ a = Dact\ (dFriend\ uid\ p\ uid')\ ou = outOK\ p = pass\ s\ uid$

$uid \in UIDs\ aID \wedge uid' = UID \vee uid = UID \wedge uid' \in UIDs$

AID

$open\ s \neg open\ s'$

$f\ ?trn = OVal\ False$

$\langle proof \rangle$

lemma *eqButUID-step- γ -out*:
assumes *ss1*: *eqButUID s s1*
and *step*: *step s a = (ou,s')* **and** *step1*: *step s1 a = (ou1,s1')*

and γ : γ (*Trans s a ou s'*)
and *os1*: \neg *open s1*
and φ : φ (*Trans s1 a ou1 s1'*) \longleftrightarrow φ (*Trans s a ou s'*)
shows *ou = ou1*
<proof>

lemma *step-open- φ* :
assumes *step s a = (ou, s')*
and *open s \neq open s'*
shows φ (*Trans s a ou s'*)
<proof>

lemma *step-sendOFriend-eqButUID*:
assumes *step s a = (ou, s')*
and *reach s*
and *uID' \notin UIDs aID*
and *a = COMact (comSendCreateOFriend UID (pass s UID) aID uID') \vee*
a = COMact (comSendDeleteOFriend UID (pass s UID) aID uID')
shows *eqButUID s s'*
<proof>

lemma *eqButUID-step- φ -imp*:
assumes *ss1*: *eqButUID s s1*
and *rs*: *reach s* **and** *rs1*: *reach s1*
and *step*: *step s a = (ou,s')* **and** *step1*: *step s1 a = (ou1,s1')*
and *a*: \forall *aID uID'*. *uID' \notin UIDs aID \longrightarrow*
 $a \neq$ *COMact (comSendCreateOFriend UID (pass s UID) aID uID')*
 \wedge
 $a \neq$ *COMact (comSendDeleteOFriend UID (pass s UID) aID uID')*
and φ : φ (*Trans s a ou s'*)
shows φ (*Trans s1 a ou1 s1'*)
<proof>

lemma *eqButUID-step- φ* :
assumes *ss1*: *eqButUID s s1*
and *rs*: *reach s* **and** *rs1*: *reach s1*
and *step*: *step s a = (ou,s')* **and** *step1*: *step s1 a = (ou1,s1')*
and *a*: \forall *aID uID'*. *uID' \notin UIDs aID \longrightarrow*
 $a \neq$ *COMact (comSendCreateOFriend UID (pass s UID) aID uID')*
 \wedge
 $a \neq$ *COMact (comSendDeleteOFriend UID (pass s UID) aID uID')*
shows φ (*Trans s a ou s'*) = φ (*Trans s1 a ou1 s1'*)
<proof>

lemma *eqButUID-step- γ* :
assumes *ss1: eqButUID s s1*
and *rs: reach s and rs1: reach s1*
and *step: step s a = (ou,s[^]) and step1: step s1 a = (ou1,s1[^])*
and *a: $\forall aID\ uID'. uID' \notin \text{UIDs } aID \longrightarrow$*
 $a \neq \text{COMact } (\text{comSendCreateOFriend } \text{UID } (\text{pass } s\ \text{UID})\ aID\ uID')$
 \wedge
 $a \neq \text{COMact } (\text{comSendDeleteOFriend } \text{UID } (\text{pass } s\ \text{UID})\ aID\ uID')$
shows $\gamma (\text{Trans } s\ a\ ou\ s[^]) = \gamma (\text{Trans } s1\ a\ ou1\ s1[^])$
<proof>

end

end

theory *Outer-Friend-Issuer*

imports

Outer-Friend-Issuer-Value-Setup

Bounded-Deducibility-Security.Compositional-Reasoning

begin

9.1.5 Declassification bound

context *OuterFriendIssuer*

begin

fun *T :: (state,act,out) trans \Rightarrow bool*

where *T trn = False*

For each user *uid* at a node *aid*, the remote friendship updates with the fixed user *UID* at node *AID* form an alternating sequence of friending and unfriending.

Note that actions involving remote users who are observers do not produce secret values; instead, those actions are observable, and the property we verify does not protect their confidentiality.

fun *validValSeq :: value list \Rightarrow (apiID \times userID) list \Rightarrow bool* **where**

validValSeq [] = True

| *validValSeq (FrVal aid uid True # vl) auidl \longleftrightarrow (aid, uid) \notin set auidl \wedge uid \notin UIDs aid \wedge validValSeq vl (auidl ## (aid, uid))*

| *validValSeq (FrVal aid uid False # vl) auidl \longleftrightarrow (aid, uid) $\in\in$ auidl \wedge uid \notin UIDs aid \wedge validValSeq vl (removeAll (aid, uid) auidl)*

| *validValSeq (OVal - # vl) auidl = validValSeq vl auidl*

abbreviation *validValSeqFrom :: value list \Rightarrow state \Rightarrow bool* **where**

validValSeqFrom vl s \equiv validValSeq vl (removeUIDs (sentOuterFriendIDs s UID))

When the access window is closed, observers may learn about the occurrence of remote friendship actions (by observing network traffic), but not their

content; the actions can be replaced by different actions involving different users (who are not observers) without affecting the observations.

inductive $BC :: \text{value list} \Rightarrow \text{value list} \Rightarrow \text{bool}$

where

$BC\text{-Nil}[simp,intro]: BC [] []$

| $BC\text{-FrVal}[intro]:$

$BC\ vl\ vl1 \Longrightarrow uid' \notin \text{UIDs}\ aid \Longrightarrow BC\ (FrVal\ aid\ uid\ st\ \#\ vl)\ (FrVal\ aid\ uid'\ st'\ \#\ vl1)$

When the access window is open, i.e. the user UID is a local friend of an observer, all information about the remote friends of UID is declassified; when the access window closes again, the contents of future updates are kept confidential.

definition $BO\ vl\ vl1 \equiv$

$(vl1 = vl) \vee$

$(\exists vl0\ vl'\ vl1'. vl = vl0 @ OVal\ False\ \# vl' \wedge vl1 = vl0 @ OVal\ False\ \# vl1' \wedge BC\ vl'\ vl1')$

definition $B\ vl\ vl1 \equiv (BC\ vl\ vl1 \vee BO\ vl\ vl1) \wedge \text{validValSeqFrom}\ vl1\ \text{istate}$

lemma $B\text{-Nil-Nil}: B\ vl\ vl1 \Longrightarrow vl1 = [] \longleftrightarrow vl = []$

$\langle \text{proof} \rangle$

sublocale $BD\text{-Security-IO}$ **where**

$\text{istate} = \text{istate}$ **and** $\text{step} = \text{step}$ **and**

$\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and** $B = B$

$\langle \text{proof} \rangle$

9.1.6 Unwinding proof

definition $\Delta0 :: \text{state} \Rightarrow \text{value list} \Rightarrow \text{state} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**

$\Delta0\ s\ vl\ s1\ vl1 \equiv$

$s1 = \text{istate} \wedge s = \text{istate} \wedge B\ vl\ vl1$

definition $\Delta1 :: \text{state} \Rightarrow \text{value list} \Rightarrow \text{state} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**

$\Delta1\ s\ vl\ s1\ vl1 \equiv$

$BO\ vl\ vl1 \wedge$

$s1 = s \wedge$

$\text{validValSeqFrom}\ vl1\ s1$

definition $\Delta2 :: \text{state} \Rightarrow \text{value list} \Rightarrow \text{state} \Rightarrow \text{value list} \Rightarrow \text{bool}$ **where**

$\Delta2\ s\ vl\ s1\ vl1 \equiv$

$BC\ vl\ vl1 \wedge$

$\text{eqButUID}\ s\ s1 \wedge \neg \text{open}\ s1 \wedge$

$\text{validValSeqFrom}\ vl1\ s1$

lemma *validValSeq-prefix*: $\text{validValSeq } (vl @ vl') \text{ auidl} \implies \text{validValSeq } vl \text{ auidl}$
<proof>

lemma *filter-removeAll*: $\text{filter } P (\text{removeAll } x \text{ } xs) = \text{removeAll } x (\text{filter } P \text{ } xs)$
<proof>

lemma *step-validValSeqFrom*:
assumes *step*: $\text{step } s \ a = (ou, s')$
and *rs*: *reach* s
and *c*: $\text{consume } (\text{Trans } s \ a \ ou \ s') \ vl \ vl' \ (\text{is consume ?trn } vl \ vl')$
and *vVS*: $\text{validValSeqFrom } vl \ s$
shows $\text{validValSeqFrom } vl' \ s'$
<proof>

lemma *istate- $\Delta 0$* :
assumes *B*: $B \ vl \ vl1$
shows $\Delta 0 \ \text{istate } vl \ \text{istate } vl1$
<proof>

lemma *unwind-cont- $\Delta 0$* : $\text{unwind-cont } \Delta 0 \ \{\Delta 1, \Delta 2\}$
<proof>

lemma *unwind-cont- $\Delta 1$* : $\text{unwind-cont } \Delta 1 \ \{\Delta 1, \Delta 2\}$
<proof>

lemma *unwind-cont- $\Delta 2$* : $\text{unwind-cont } \Delta 2 \ \{\Delta 2\}$
<proof>

definition *Gr* where
 $Gr =$
{
 ($\Delta 0$, { $\Delta 1, \Delta 2$ }),
 ($\Delta 1$, { $\Delta 1, \Delta 2$ }),
 ($\Delta 2$, { $\Delta 2$ })
}

theorem *secure*: *secure*
<proof>

end

end

theory *Outer-Friend-Receiver-Observation-Setup*
 imports *../Outer-Friend*
begin

9.2 Receiver nodes

9.2.1 Observation setup

locale *OuterFriendReceiver* = *OuterFriend* +
fixes *AID'* :: *apiID* — The ID of this (arbitrary, but fixed) receiver node
begin

fun γ :: (*state,act,out*) *trans* \Rightarrow *bool* **where**
 γ (*Trans* - *a ou* -) \longleftrightarrow (\exists *uid*. *userOfA* *a* = *Some uid* \wedge *uid* \in *UIDs AID'*) \vee
(\exists *ca*. *a* = *COMact ca* \wedge *ou* \neq *outErr*)

fun *sPurge* :: *sActt* \Rightarrow *sActt* **where**
sPurge (*sSys uid pwd*) = *sSys uid emptyPass*

fun *comPurge* :: *comActt* \Rightarrow *comActt* **where**
comPurge (*comSendServerReq uID p aID reqInfo*) = *comSendServerReq uID emptyPass aID reqInfo*
comPurge (*comReceiveClientReq aID reqInfo*) = *comReceiveClientReq aID reqInfo*
comPurge (*comConnectClient uID p aID sp*) = *comConnectClient uID emptyPass aID sp*
comPurge (*comConnectServer aID sp*) = *comConnectServer aID sp*
comPurge (*comReceivePost aID sp nID nt uID v*) = *comReceivePost aID sp nID nt uID v*
comPurge (*comSendPost uID p aID nID*) = *comSendPost uID emptyPass aID nID*
comPurge (*comSendCreateOFriend uID p aID uID'*) = *comSendCreateOFriend uID emptyPass aID uID'*

comPurge (*comReceiveCreateOFriend aID cp uID uID'*) =
 (*if* *aID* = *AID* \wedge *uID* = *UID* \wedge *uID'* \notin *UIDs AID'*
 then *comReceiveCreateOFriend aID cp uID emptyUserID*
 else *comReceiveCreateOFriend aID cp uID uID'*)
comPurge (*comSendDeleteOFriend uID p aID uID'*) = *comSendDeleteOFriend uID emptyPass aID uID'*

comPurge (*comReceiveDeleteOFriend aID cp uID uID'*) =
 (*if* *aID* = *AID* \wedge *uID* = *UID* \wedge *uID'* \notin *UIDs AID'*
 then *comReceiveCreateOFriend aID cp uID emptyUserID*
 else *comReceiveDeleteOFriend aID cp uID uID'*)

lemma *comPurge-simps*:

comPurge ca = *comSendServerReq uID p aID reqInfo* \longleftrightarrow (\exists *p'*. *ca* = *comSendServerReq uID p' aID reqInfo* \wedge *p* = *emptyPass*)
comPurge ca = *comReceiveClientReq aID reqInfo* \longleftrightarrow *ca* = *comReceiveClientReq aID reqInfo*

$comPurge\ ca = comConnectClient\ uID\ p\ aID\ sp \longleftrightarrow (\exists p'.\ ca = comConnectClient\ uID\ p' aID\ sp \wedge p = emptyPass)$
 $comPurge\ ca = comConnectServer\ aID\ sp \longleftrightarrow ca = comConnectServer\ aID\ sp$
 $comPurge\ ca = comReceivePost\ aID\ sp\ nID\ nt\ uID\ v \longleftrightarrow ca = comReceivePost\ aID\ sp\ nID\ nt\ uID\ v$
 $comPurge\ ca = comSendPost\ uID\ p\ aID\ nID \longleftrightarrow (\exists p'.\ ca = comSendPost\ uID\ p' aID\ nID \wedge p = emptyPass)$
 $comPurge\ ca = comSendCreateOFriend\ uID\ p\ aID\ uID' \longleftrightarrow (\exists p'.\ ca = comSendCreateOFriend\ uID\ p' aID\ uID' \wedge p = emptyPass)$
 $comPurge\ ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow (\exists uid''.\ (ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uid'' \vee ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uid'') \wedge aID = AID \wedge uID = UID \wedge uid'' \notin UIDs\ AID' \wedge uID' = emptyUserID)$
 $\vee (ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uID' \wedge \neg(aID = AID \wedge uID = UID \wedge uID' \notin UIDs\ AID'))$
 $comPurge\ ca = comSendDeleteOFriend\ uID\ p\ aID\ uID' \longleftrightarrow (\exists p'.\ ca = comSendDeleteOFriend\ uID\ p' aID\ uID' \wedge p = emptyPass)$
 $comPurge\ ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID' \wedge \neg(aID = AID \wedge uID = UID \wedge uID' \notin UIDs\ AID')$
 $\langle proof \rangle$

fun $g :: (state, act, out)trans \Rightarrow obs\ \mathbf{where}$

$g\ (Trans - (Sact\ sa)\ ou\ -) = (Sact\ (sPurge\ sa),\ ou)$
 $|g\ (Trans - (COMact\ ca)\ ou\ -) = (COMact\ (comPurge\ ca),\ ou)$
 $|g\ (Trans - a\ ou\ -) = (a, ou)$

lemma g -simps:

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendServerReq\ uID\ p\ aID\ reqInfo),\ ou')$
 $\longleftrightarrow (\exists p'.\ a = COMact\ (comSendServerReq\ uID\ p' aID\ reqInfo) \wedge p = emptyPass \wedge ou = ou')$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveClientReq\ aID\ reqInfo),\ ou')$
 $\longleftrightarrow a = COMact\ (comReceiveClientReq\ aID\ reqInfo) \wedge ou = ou'$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectClient\ uID\ p\ aID\ sp),\ ou')$
 $\longleftrightarrow (\exists p'.\ a = COMact\ (comConnectClient\ uID\ p' aID\ sp) \wedge p = emptyPass \wedge ou = ou')$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectServer\ aID\ sp),\ ou')$
 $\longleftrightarrow a = COMact\ (comConnectServer\ aID\ sp) \wedge ou = ou'$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceivePost\ aID\ sp\ nID\ nt\ uID\ v),\ ou')$
 $\longleftrightarrow a = COMact\ (comReceivePost\ aID\ sp\ nID\ nt\ uID\ v) \wedge ou = ou'$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendPost\ uID\ p\ aID\ nID),\ ou')$
 $\longleftrightarrow (\exists p'.\ a = COMact\ (comSendPost\ uID\ p' aID\ nID) \wedge p = emptyPass \wedge ou = ou')$
 $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendCreateOFriend\ uID\ p\ aID\ uID'),\ ou')$
 $\longleftrightarrow (\exists p'.\ a = COMact\ (comSendCreateOFriend\ uID\ p' aID\ uID') \wedge p = emptyPass \wedge ou = ou')$

$g (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID'), ou')$
 $\longleftrightarrow ((\exists\ uid''. (a = COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uid'') \vee a = COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uid'')) \wedge aID = AID \wedge uID = UID \wedge uid'' \notin UIDs\ AID' \wedge uID' = emptyUserID)$
 $\vee (a = COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID') \wedge \neg(aID = AID \wedge uID = UID \wedge uID' \notin UIDs\ AID'))))$
 $\wedge ou = ou')$
 $g (Trans\ s\ a\ ou\ s') = (COMact\ (comSendDeleteOFriend\ uID\ p\ aID\ uID'), ou')$
 $\longleftrightarrow (\exists\ p'. a = COMact\ (comSendDeleteOFriend\ uID\ p\ aID\ uID') \wedge p = emptyPass \wedge ou = ou')$
 $g (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'), ou')$
 $\longleftrightarrow a = COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID') \wedge \neg(aID = AID \wedge uID = UID \wedge uID' \notin UIDs\ AID') \wedge ou = ou'$
 $\langle proof \rangle$

end

end

theory *Outer-Friend-Receiver-State-Indistinguishability*
imports *Outer-Friend-Receiver-Observation-Setup*
begin

9.2.2 Unwinding helper definitions and lemmas

context *OuterFriendReceiver*
begin

fun *eqButUIDl* :: $(apiID \times userID)\ list \Rightarrow (apiID \times userID)\ list \Rightarrow bool$ **where**
 $eqButUIDl\ auidl\ auidl1 = (remove1\ (AID, UID)\ auidl = remove1\ (AID, UID)\ auidl1)$

lemma *eqButUIDl-eq[simp,intro!]*: $eqButUIDl\ auidl\ auidl$
 $\langle proof \rangle$

lemma *eqButUIDl-sym*:
assumes $eqButUIDl\ auidl\ auidl1$
shows $eqButUIDl\ auidl1\ auidl$
 $\langle proof \rangle$

lemma *eqButUIDl-trans*:
assumes $eqButUIDl\ auidl\ auidl1$ **and** $eqButUIDl\ auidl1\ auidl2$
shows $eqButUIDl\ auidl\ auidl2$
 $\langle proof \rangle$

lemma *eqButUIDl-remove1-cong*:
assumes *eqButUIDl auidl auidl1*
shows *eqButUIDl (remove1 auid auidl) (remove1 auid auidl1)*
 \langle *proof* \rangle

lemma *eqButUIDl-snoc-cong*:
assumes *eqButUIDl auidl auidl1*
and *auid' $\in\in$ auidl \longleftrightarrow auid' $\in\in$ auidl1*
shows *eqButUIDl (auidl ## auid') (auidl1 ## auid')*
 \langle *proof* \rangle

definition *eqButUIDf where*
eqButUIDf frds frds1 \equiv
 $(\forall uid. \text{if } uid \in \text{UIDs } AID' \text{ then } frds \text{ uid} = frds1 \text{ uid} \text{ else } eqButUIDl (frds \text{ uid}) (frds1 \text{ uid}))$

lemmas *eqButUIDf-intro = eqButUIDf-def[THEN meta-eq-to-obj-eq, THEN iffD2]*

lemma *eqButUIDf-eeq[simp,intro!]*: *eqButUIDf frds frds*
 \langle *proof* \rangle

lemma *eqButUIDf-sym*:
assumes *eqButUIDf frds frds1* **shows** *eqButUIDf frds1 frds*
 \langle *proof* \rangle

lemma *eqButUIDf-trans*:
assumes *eqButUIDf frds frds1* **and** *eqButUIDf frds1 frds2*
shows *eqButUIDf frds frds2*
 \langle *proof* \rangle

lemma *eqButUIDf-cong*:
assumes *eqButUIDf frds frds1*
and *uid \in UIDs AID' \implies uu = uu1*
and *uid \notin UIDs AID' \implies eqButUIDl uu uu1*
shows *eqButUIDf (frds (uid := uu)) (frds1 (uid := uu1))*
 \langle *proof* \rangle

lemma *eqButUIDf-UIDs*:
 $\llbracket eqButUIDf frds frds1; uid \in \text{UIDs } AID' \rrbracket \implies frds \text{ uid} = frds1 \text{ uid}$
 \langle *proof* \rangle

definition *eqButUID :: state \Rightarrow state \Rightarrow bool where*
eqButUID s s1 \equiv

$admin\ s = admin\ s1 \wedge$

$pendingUReqs\ s = pendingUReqs\ s1 \wedge userReq\ s = userReq\ s1 \wedge$
 $userIDs\ s = userIDs\ s1 \wedge user\ s = user\ s1 \wedge pass\ s = pass\ s1 \wedge$

$pendingFReqs\ s = pendingFReqs\ s1 \wedge$
 $friendReq\ s = friendReq\ s1 \wedge$
 $friendIDs\ s = friendIDs\ s1 \wedge$

$postIDs\ s = postIDs\ s1 \wedge admin\ s = admin\ s1 \wedge$
 $post\ s = post\ s1 \wedge vis\ s = vis\ s1 \wedge$
 $owner\ s = owner\ s1 \wedge$

$pendingSApiReqs\ s = pendingSApiReqs\ s1 \wedge sApiReq\ s = sApiReq\ s1 \wedge$
 $serverApiIDs\ s = serverApiIDs\ s1 \wedge serverPass\ s = serverPass\ s1 \wedge$
 $outerPostIDs\ s = outerPostIDs\ s1 \wedge outerPost\ s = outerPost\ s1 \wedge outerVis\ s =$
 $outerVis\ s1 \wedge$
 $outerOwner\ s = outerOwner\ s1 \wedge$
 $sentOuterFriendIDs\ s = sentOuterFriendIDs\ s1 \wedge$
 $eqButUIDf\ (recvOuterFriendIDs\ s)\ (recvOuterFriendIDs\ s1) \wedge$

$pendingCApiReqs\ s = pendingCApiReqs\ s1 \wedge cApiReq\ s = cApiReq\ s1 \wedge$
 $clientApiIDs\ s = clientApiIDs\ s1 \wedge clientPass\ s = clientPass\ s1 \wedge$
 $sharedWith\ s = sharedWith\ s1$

lemmas $eqButUID-intro = eqButUID-def[THEN\ meta-eq-to-obj-eq,\ THEN\ iffD2]$

lemma $eqButUID-refl[simp,intro!]$: $eqButUID\ s\ s$
{proof}

lemma $eqButUID-sym[sym]$:
assumes $eqButUID\ s\ s1$ **shows** $eqButUID\ s1\ s$
{proof}

lemma $eqButUID-trans[trans]$:
assumes $eqButUID\ s\ s1$ **and** $eqButUID\ s1\ s2$ **shows** $eqButUID\ s\ s2$
{proof}

lemma $eqButUID-stateSelectors$:
assumes $eqButUID\ s\ s1$
shows $admin\ s = admin\ s1$
 $pendingUReqs\ s = pendingUReqs\ s1$ $userReq\ s = userReq\ s1$
 $userIDs\ s = userIDs\ s1$ $user\ s = user\ s1$ $pass\ s = pass\ s1$
 $pendingFReqs\ s = pendingFReqs\ s1$
 $friendReq\ s = friendReq\ s1$
 $friendIDs\ s = friendIDs\ s1$

$postIDs\ s = postIDs\ s1$

$post\ s = post\ s1$ $vis\ s = vis\ s1$
 $owner\ s = owner\ s1$

$pendingSApiReqs\ s = pendingSApiReqs\ s1$ $sApiReq\ s = sApiReq\ s1$
 $serverApiIDs\ s = serverApiIDs\ s1$ $serverPass\ s = serverPass\ s1$
 $outerPostIDs\ s = outerPostIDs\ s1$ $outerPost\ s = outerPost\ s1$ $outerVis\ s = outerVis\ s1$
 $outerOwner\ s = outerOwner\ s1$
 $sentOuterFriendIDs\ s = sentOuterFriendIDs\ s1$
 $eqButUIDf\ (recvOuterFriendIDs\ s)\ (recvOuterFriendIDs\ s1)$

$pendingCApiReqs\ s = pendingCApiReqs\ s1$ $cApiReq\ s = cApiReq\ s1$
 $clientApiIDs\ s = clientApiIDs\ s1$ $clientPass\ s = clientPass\ s1$
 $sharedWith\ s = sharedWith\ s1$

$IDsOK\ s = IDsOK\ s1$
 $\langle proof \rangle$

lemma $eqButUID-UIDs$:

$eqButUID\ s\ s1 \implies uid \in UIDs\ AID' \implies recvOuterFriendIDs\ s\ uid = recvOuterFriendIDs\ s1\ uid$
 $\langle proof \rangle$

lemma $eqButUID-recvOuterFriends-UIDs$:

assumes $eqButUID\ s\ s1$

and $uid \neq UID \vee aid \neq AID$

shows $(aid, uid) \in \in recvOuterFriendIDs\ s\ uid' \longleftrightarrow (aid, uid) \in \in recvOuterFriendIDs\ s1\ uid'$

$\langle proof \rangle$

lemma $eqButUID-remove1-UID-recvOuterFriends$:

assumes $eqButUID\ s\ s1$

shows $remove1\ (AID, UID)\ (recvOuterFriendIDs\ s\ uid) = remove1\ (AID, UID)\ (recvOuterFriendIDs\ s1\ uid)$

$\langle proof \rangle$

lemma $eqButUID-cong$:

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\downarrow admin := uu1))\ (s1\ (\downarrow admin := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\downarrow pendingUReqs := uu1))\ (s1\ (\downarrow pendingUReqs := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\downarrow userReq := uu1))\ (s1\ (\downarrow userReq := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\downarrow userIDs := uu1))\ (s1\ (\downarrow userIDs := uu2))$

$\bigwedge uu1\ uu2. eqButUID\ s\ s1 \implies uu1 = uu2 \implies eqButUID\ (s\ (\downarrow user := uu1))\ (s1\ (\downarrow user := uu2))$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash pass := uu1)) (s1 (\backslash pass := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash postIDs := uu1)) (s1 (\backslash postIDs := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash owner := uu1)) (s1 (\backslash owner := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash post := uu1)) (s1 (\backslash post := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash vis := uu1)) (s1 (\backslash vis := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash pendingFReqs := uu1)) (s1 (\backslash pendingFReqs := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash friendReq := uu1)) (s1 (\backslash friendReq := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash friendIDs := uu1)) (s1 (\backslash friendIDs := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash pendingSApiReqs := uu1)) (s1 (\backslash pendingSApiReqs := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash sApiReq := uu1)) (s1 (\backslash sApiReq := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash serverApiIDs := uu1)) (s1 (\backslash serverApiIDs := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash serverPass := uu1)) (s1 (\backslash serverPass := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash outerPostIDs := uu1)) (s1 (\backslash outerPostIDs := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash outerPost := uu1)) (s1 (\backslash outerPost := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash outerVis := uu1)) (s1 (\backslash outerVis := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash outerOwner := uu1)) (s1 (\backslash outerOwner := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash sentOuterFriendIDs := uu1)) (s1 (\backslash sentOuterFriendIDs := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies eqButUIDf uu1 uu2 \implies eqButUID (s (\backslash recvOuterFriendIDs := uu1)) (s1 (\backslash recvOuterFriendIDs := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash pendingCApiReqs := uu1)) (s1 (\backslash pendingCApiReqs := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash cApiReq := uu1)) (s1 (\backslash cApiReq := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash clientApiIDs := uu1)) (s1 (\backslash clientApiIDs := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash clientPass := uu1)) (s1 (\backslash clientPass := uu2))$$

$$\bigwedge uu1 uu2. eqButUID s s1 \implies uu1 = uu2 \implies eqButUID (s (\backslash sharedWith := uu1)) (s1 (\backslash sharedWith := uu2))$$

```

uu1)) (s1 (sharedWith:= uu2))
⟨proof⟩

```

end

end

```

theory Outer-Friend-Receiver-Value-Setup
  imports Outer-Friend-Receiver-State-Indistinguishability
begin

```

9.2.3 Value Setup

```

context OuterFriendReceiver
begin

```

```

fun  $\varphi$  :: (state,act,out) trans  $\Rightarrow$  bool where
 $\varphi$  (Trans s (COMact (comReceiveCreateOFriend aID cp uID uID')) ou s') =
  (aID = AID  $\wedge$  uID = UID  $\wedge$  uID'  $\notin$  UIDs AID'  $\wedge$  ou = outOK)
|
 $\varphi$  (Trans s (COMact (comReceiveDeleteOFriend aID cp uID uID')) ou s') =
  (aID = AID  $\wedge$  uID = UID  $\wedge$  uID'  $\notin$  UIDs AID'  $\wedge$  ou = outOK)
|
 $\varphi$  - = False

```

```

fun f :: (state,act,out) trans  $\Rightarrow$  value where
f (Trans s (COMact (comReceiveCreateOFriend aID cp uID uID')) ou s') = FrVal
  AID' uID' True
|
f (Trans s (COMact (comReceiveDeleteOFriend aID cp uID uID')) ou s') = FrVal
  AID' uID' False
|
f - = undefined

```

```

lemma recvOFriend-eqButUID:
assumes step s a = (ou, s')
and reach s
and a = COMact (comReceiveCreateOFriend AID cp UID uID')  $\vee$  a = COMact
  (comReceiveDeleteOFriend AID cp UID uID')
and uID'  $\notin$  UIDs AID'
shows eqButUID s s'
⟨proof⟩

```

```

lemma eqButUID-step:
assumes ss1: eqButUID s s1
and step: step s a = (ou,s')

```

```

and step1: step s1 a = (ou1,s1')
and rs: reach s
and rs1: reach s1
shows eqButUID s' s1'
⟨proof⟩

```

```

lemma eqButUID-step-γ-out:
assumes ss1: eqButUID s s1
and step: step s a = (ou,s') and step1: step s1 a = (ou1,s1')
and  $\varphi$ :  $\varphi (\text{Trans } s1 \ a \ ou1 \ s1')$   $\longleftrightarrow$   $\varphi (\text{Trans } s \ a \ ou \ s')$ 
and  $\gamma$ :  $\gamma (\text{Trans } s \ a \ ou \ s')$ 
shows ou = ou1
⟨proof⟩

```

```

lemma eqButUID-step-γ:
assumes ss1: eqButUID s s1
and step: step s a = (ou,s') and step1: step s1 a = (ou1,s1')
and  $\varphi$ :  $\varphi (\text{Trans } s1 \ a \ ou1 \ s1')$   $\longleftrightarrow$   $\varphi (\text{Trans } s \ a \ ou \ s')$ 
shows  $\gamma (\text{Trans } s \ a \ ou \ s') = \gamma (\text{Trans } s1 \ a \ ou1 \ s1')$ 
⟨proof⟩

```

end

end

theory *Outer-Friend-Receiver*

imports

Outer-Friend-Receiver-Value-Setup

Bounded-Deducibility-Security.Compositional-Reasoning

begin

9.2.4 Declassification bound

context *OuterFriendReceiver*

begin

fun *T* :: $(state,act,out) \text{ trans} \Rightarrow bool$

where *T trn = False*

For each user *uid* at this receiver node *AID'*, the remote friendship updates with the fixed user *UID* at the issuer node *AID* form an alternating sequence of friending and unfriending.

Note that actions involving remote users who are observers do not produce secret values; instead, those actions are observable, and the property we verify does not protect their confidentiality.

Moreover, there is no declassification trigger on the receiver side, so *OVal*

values are never produced by receiver nodes, only by the issuer node.

definition $friendsOfUID :: state \Rightarrow userID\ set\ \mathbf{where}$

$friendsOfUID\ s = \{uid. (AID, UID) \in \in\ recvOuterFriendIDs\ s\ uid \wedge uid \notin\ UIDs\ AID'\}$

fun $validValSeq :: value\ list \Rightarrow userID\ set \Rightarrow bool\ \mathbf{where}$

$validValSeq\ [] = True$
 $| validValSeq\ (FrVal\ aid\ uid\ True\ \#\ vl)\ uids \longleftrightarrow uid \notin\ uids \wedge aid = AID' \wedge uid \notin\ UIDs\ AID' \wedge validValSeq\ vl\ (insert\ uid\ uids)$
 $| validValSeq\ (FrVal\ aid\ uid\ False\ \#\ vl)\ uids \longleftrightarrow uid \in\ uids \wedge aid = AID' \wedge uid \notin\ UIDs\ AID' \wedge validValSeq\ vl\ (uids - \{uid\})$
 $| validValSeq\ (OVal\ ov\ \#\ vl)\ uids \longleftrightarrow False$

abbreviation $validValSeqFrom\ vl\ s \equiv validValSeq\ vl\ (friendsOfUID\ s)$

Observers may learn about the occurrence of remote friendship actions (by observing network traffic), but not their content; remote friendship actions at a receiver node AID' can be replaced by different actions involving different users of that node (who are not observers) without affecting the observations.

inductive $BC :: value\ list \Rightarrow value\ list \Rightarrow bool$

where

$BC\ Nil[simp, intro]: BC\ []\ []$
 $| BC\ FrVal[intro]:$
 $BC\ vl\ vl1 \Longrightarrow uid' \notin\ UIDs\ AID' \Longrightarrow BC\ (FrVal\ aid\ uid\ st\ \#\ vl)\ (FrVal\ AID'\ uid'\ st'\ \#\ vl1)$

definition $B\ vl\ vl1 \equiv BC\ vl\ vl1 \wedge validValSeqFrom\ vl1\ istate$

lemma $BC\ Nil\ Nil: BC\ vl\ vl1 \Longrightarrow vl1 = [] \longleftrightarrow vl = []$

$\langle proof \rangle$

lemma $BC\ id: validValSeq\ vl\ uids \Longrightarrow BC\ vl\ vl$

$\langle proof \rangle$

lemma $BC\ append: BC\ vl\ vl1 \Longrightarrow BC\ vl'\ vl1' \Longrightarrow BC\ (vl\ @\ vl')\ (vl1\ @\ vl1')$

$\langle proof \rangle$

sublocale $BD\ Security\ IO\ \mathbf{where}$

$istate = istate\ \mathbf{and}\ step = step\ \mathbf{and}$

$\varphi = \varphi\ \mathbf{and}\ f = f\ \mathbf{and}\ \gamma = \gamma\ \mathbf{and}\ g = g\ \mathbf{and}\ T = T\ \mathbf{and}\ B = B$

$\langle proof \rangle$

9.2.5 Unwinding proof

definition $\Delta 0 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool\ \mathbf{where}$

$\Delta 0\ s\ vl\ s1\ vl1 \equiv BC\ vl\ vl1 \wedge eqButUID\ s\ s1 \wedge validValSeqFrom\ vl1\ s1$

lemma *istate- $\Delta 0$* :
assumes $B: B \text{ vl vl1}$
shows $\Delta 0 \text{ istate vl istate vl1}$
 $\langle \text{proof} \rangle$

lemma *friendsOfUID-cong*:
assumes $\text{recvOuterFriendIDs } s = \text{recvOuterFriendIDs } s'$
shows $\text{friendsOfUID } s = \text{friendsOfUID } s'$
 $\langle \text{proof} \rangle$

lemma *friendsOfUID-step-not-UID*:
assumes $\text{uid} \neq \text{UID} \vee \text{aid} \neq \text{AID} \vee \text{uid}' \in \text{UIDs AID}'$
shows $\text{friendsOfUID } (\text{receiveCreateOfFriend } s \text{ aid sp uid uid}') = \text{friendsOfUID } s$
and $\text{friendsOfUID } (\text{receiveDeleteOfFriend } s \text{ aid sp uid uid}') = \text{friendsOfUID } s$
 $\langle \text{proof} \rangle$

lemma *friendsOfUID-step-Create-UID*:
assumes $\text{uid}' \notin \text{UIDs AID}'$
shows $\text{friendsOfUID } (\text{receiveCreateOfFriend } s \text{ AID sp UID uid}') = \text{insert uid}'$
 $(\text{friendsOfUID } s)$
 $\langle \text{proof} \rangle$

lemma *friendsOfUID-step-Delete-UID*:
assumes $e\text{-receiveDeleteOfFriend } s \text{ AID sp UID uid}'$
and $rs: \text{reach } s$
shows $\text{friendsOfUID } (\text{receiveDeleteOfFriend } s \text{ AID sp UID uid}') = \text{friendsOfUID}$
 $s - \{\text{uid}'\}$
 $\langle \text{proof} \rangle$

lemma *step-validValSeqFrom*:
assumes $\text{step: step } s \text{ a} = (\text{ou}, s')$
and $rs: \text{reach } s$
and $c: \text{consume } (\text{Trans } s \text{ a ou } s') \text{ vl vl}'$ (**is** $\text{consume ?trn vl vl}'$)
and $vVS: \text{validValSeqFrom vl } s$
shows $\text{validValSeqFrom vl}' s'$
 $\langle \text{proof} \rangle$

lemma *unwind-cont- $\Delta 0$* : $\text{unwind-cont } \Delta 0 \{\Delta 0\}$
 $\langle \text{proof} \rangle$

definition Gr where
 $Gr =$
 $\{$
 $(\Delta 0, \{\Delta 0\})$
 $\}$

theorem *secure: secure*
<proof>

end

end

theory *Outer-Friend-Network*

imports

../API-Network

Issuer/Outer-Friend-Issuer

Receiver/Outer-Friend-Receiver

BD-Security-Compositional.Composing-Security-Network

begin

9.3 Confidentiality for the N-ary composition

locale *OuterFriendNetwork = OuterFriend + Network +*

assumes *AID-AIDs: AID ∈ AIDs*

begin

sublocale *Issuer: OuterFriendIssuer* *UIDs AID UID <proof>*

abbreviation $\varphi :: \text{apiID} \Rightarrow (\text{state}, \text{act}, \text{out}) \text{ trans} \Rightarrow \text{bool}$

where $\varphi \text{ aid trn} \equiv (\text{if } \text{aid} = \text{AID} \text{ then } \text{Issuer}.\varphi \text{ trn} \text{ else } \text{OuterFriendReceiver}.\varphi \text{ UIDs AID UID aid trn})$

abbreviation $f :: \text{apiID} \Rightarrow (\text{state}, \text{act}, \text{out}) \text{ trans} \Rightarrow \text{value}$

where $f \text{ aid trn} \equiv (\text{if } \text{aid} = \text{AID} \text{ then } \text{Issuer}.f \text{ trn} \text{ else } \text{OuterFriendReceiver}.f \text{ aid trn})$

abbreviation $\gamma :: \text{apiID} \Rightarrow (\text{state}, \text{act}, \text{out}) \text{ trans} \Rightarrow \text{bool}$

where $\gamma \text{ aid trn} \equiv (\text{if } \text{aid} = \text{AID} \text{ then } \text{Issuer}.\gamma \text{ trn} \text{ else } \text{OuterFriendReceiver}.\gamma \text{ UIDs aid trn})$

abbreviation $g :: \text{apiID} \Rightarrow (\text{state}, \text{act}, \text{out}) \text{ trans} \Rightarrow \text{obs}$

where $g \text{ aid trn} \equiv (\text{if } \text{aid} = \text{AID} \text{ then } \text{Issuer}.g \text{ trn} \text{ else } \text{OuterFriendReceiver}.g \text{ UIDs AID UID aid trn})$

abbreviation $T :: \text{apiID} \Rightarrow (\text{state}, \text{act}, \text{out}) \text{ trans} \Rightarrow \text{bool}$

where $T \text{ aid trn} \equiv \text{False}$

abbreviation $B :: \text{apiID} \Rightarrow \text{value list} \Rightarrow \text{value list} \Rightarrow \text{bool}$

where $B \text{ aid vl vl1} \equiv (\text{if } \text{aid} = \text{AID} \text{ then } \text{Issuer}.B \text{ vl vl1} \text{ else } \text{OuterFriendReceiver}.B \text{ UIDs AID UID aid vl vl1})$

fun *comOfV* **where**

$comOfV\ aid\ (FrVal\ aid'\ uid'\ st) = (if\ aid \neq AID\ then\ Recv\ else\ (if\ aid' \neq aid\ then\ Send\ else\ Internal))$
 $| comOfV\ aid\ (OVal\ ov) = Internal$

fun $tgtNodeOfV$ **where**

$tgtNodeOfV\ aid\ (FrVal\ aid'\ uid'\ st) = (if\ aid = AID\ then\ aid'\ else\ AID)$
 $| tgtNodeOfV\ aid\ (OVal\ ov) = AID$

abbreviation $syncV\ aid1\ v1\ aid2\ v2 \equiv (v1 = v2)$

sublocale $Net: BD\text{-}Security\text{-}TS\text{-}Network\text{-}getTgtV$

where $istate = \lambda\cdot. istate$ **and** $validTrans = validTrans$ **and** $srcOf = \lambda\cdot. srcOf$
and $tgtOf = \lambda\cdot. tgtOf$

and $nodes = AIDs$ **and** $comOf = comOf$ **and** $tgtNodeOf = tgtNodeOf$
and $sync = sync$ **and** $\varphi = \varphi$ **and** $f = f$ **and** $\gamma = \gamma$ **and** $g = g$ **and** $T = T$ **and**
 $B = B$

and $comOfV = comOfV$ **and** $tgtNodeOfV = tgtNodeOfV$ **and** $syncV = syncV$
and $comOfO = comOfO$ **and** $tgtNodeOfO = tgtNodeOfO$ **and** $syncO = syncO$
and $source = AID$ **and** $getTgtV = id$

$\langle proof \rangle$

context

fixes $AID' :: apiID$

assumes $AID': AID' \in AIDs - \{AID\}$

begin

interpretation $Receiver: OuterFriendReceiver\ UIDs\ AID\ UID\ AID' \langle proof \rangle$

lemma $Issuer\text{-}BC\text{-}Receiver\text{-}BC:$

assumes $Issuer.BC\ vl\ vl1$

shows $Receiver.BC\ (Net.projectSrcV\ AID'\ vl)\ (Net.projectSrcV\ AID'\ vl1)$

$\langle proof \rangle$

lemma $Collect\text{-}setminus: Collect\ P - A = \{u. u \notin A \wedge P\ u\}$

$\langle proof \rangle$

lemma $Issuer\text{-}vVS\text{-}Receiver\text{-}vVS:$

assumes $Issuer.validValSeq\ vl\ auidl$

shows $Receiver.validValSeq\ (Net.projectSrcV\ AID'\ vl)\ \{uid. (AID', uid) \in\ auidl\}$

$\langle proof \rangle$

lemma $Issuer\text{-}B\text{-}Receiver\text{-}B:$

assumes $Issuer.B\ vl\ vl1$

shows $Receiver.B\ (Net.projectSrcV\ AID'\ vl)\ (Net.projectSrcV\ AID'\ vl1)$

$\langle proof \rangle$

end

```

sublocale BD-Security-TS-Network-Preserve-Source-Security-getTgtV
where istate =  $\lambda$ -. istate and validTrans = validTrans and srcOf =  $\lambda$ -. srcOf
and tgtOf =  $\lambda$ -. tgtOf
  and nodes = AIDs and comOf = comOf and tgtNodeOf = tgtNodeOf
  and sync = sync and  $\varphi$  =  $\varphi$  and  $f$  =  $f$  and  $\gamma$  =  $\gamma$  and  $g$  =  $g$  and  $T$  =  $T$  and
   $B$  =  $B$ 
  and comOfV = comOfV and tgtNodeOfV = tgtNodeOfV and syncV = syncV
  and comOfO = comOfO and tgtNodeOfO = tgtNodeOfO and syncO = syncO
  and source = AID and getTgtV = id
   $\langle$ proof $\rangle$ 

theorem secure: secure
   $\langle$ proof $\rangle$ 

end

end

theory Outer-Friend-All
imports Outer-Friend-Network
begin

end

```

References

- [1] The Diaspora project. <https://diasporafoundation.org/>, 2021.
- [2] T. Bauerei, A. Pesenti Gritti, A. Popescu, and F. Raimondi. CoSMed: A confidentiality-verified social media platform. In J. C. Blanchette and S. Merz, editors, *Interactive Theorem Proving - 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings*, volume 9807 of *Lecture Notes in Computer Science*, pages 87–106. Springer, 2016.
- [3] T. Bauerei, A. Pesenti Gritti, A. Popescu, and F. Raimondi. CoSMedis: A distributed social media platform with formally verified confidentiality guarantees. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 729–748. IEEE Computer Society, 2017.
- [4] T. Bauerei, A. Pesenti Gritti, A. Popescu, and F. Raimondi. CoSMed: A confidentiality-verified social media platform. *J. Autom. Reason.*, 61(1-4):113–139, 2018.

- [5] T. Bauereiss and A. Popescu. Compositional BD Security. In M. Eberl, G. Klein, A. Lochbihler, T. Nipkow, L. Paulson, and R. Thiemann, editors, *Archive of Formal Proofs*, 2021.
- [6] T. Bauereiss and A. Popescu. CoSMed: A confidentiality-verified social media platform. In M. Eberl, G. Klein, A. Lochbihler, T. Nipkow, L. Paulson, and R. Thiemann, editors, *Archive of Formal Proofs*, 2021.
- [7] A. Popescu, T. Bauereiss, and P. Lammich. Bounded-Deducibility security (invited paper). In L. Cohen and C. Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference)*, volume 193 of *LIPICs*, pages 3:1–3:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- [8] A. Popescu, P. Lammich, and T. Bauereiss. Bounded-deducibility security. In G. Klein, T. Nipkow, and L. Paulson, editors, *Archive of Formal Proofs*, 2014.