CoSMeDis: A confidentiality-verified distributed social media platform

Thomas Bauereiss Andrei Popescu

March 19, 2025

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

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

CoSMed [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 CoSMed 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 CoSMed 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 infrastructures[1]
- defining the declassification bounds and triggers[2]
- 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.

[1] NB: The secrets are called "values" in the formalization.

[2] In 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* ≡ *Nam emptyStr*
datatype *inform* = *Info String.literal*
definition *emptyInfo* ≡ *Info emptyStr*

datatype *user* = *Usr name inform*
fun *nameUser* **where** *nameUser* (*Usr 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* ∈ *apiIDAsStr* ‘ *apiIDs* ⟷ *Aid str* ∈ *apiIDs*
⟨*proof*⟩

lemma *getFreshApiID*: ¬ *getFreshApiID apiIDs* ∈∈ *apiIDs*
⟨*proof*⟩

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

definition *getFreshUserID userIDs* ≡ *Uid* (*fresh* (*set* (*map userIDAsStr userIDs*)) (*STR ''2''*))

lemma *UserID-userIDAsStr*[*simp*]: *Uid* (*userIDAsStr userID*) = *userID*
⟨*proof*⟩

lemma *member-userIDAsStr-iff*[*simp*]: *str* ∈ *userIDAsStr* ‘ (*set userIDs*) ⟷ *Uid str* ∈∈ *userIDs*
⟨*proof*⟩

lemma *getFreshUserID*: ¬ *getFreshUserID userIDs* ∈∈ *userIDs*
⟨*proof*⟩

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

definition *getFreshPostID postIDs* ≡ *Pid* (*fresh* (*set* (*map postIDAsStr postIDs*)) (*STR ''3''*))

lemma *PostID-postIDAsStr*[*simp*]: *Pid* (*postIDAsStr postID*) = *postID*
⟨*proof*⟩

lemma *member-postIDAsStr-iff*[*simp*]: *str* ∈ *postIDAsStr* ‘ (*set postIDs*) ⟷ *Pid str* ∈∈ *postIDs*
⟨*proof*⟩

lemma *getFreshPostID*: ¬ *getFreshPostID postIDs* ∈∈ *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* ⇒ *requestInfo*
userIDs :: *userID list*
user :: *userID* ⇒ *user*
pass :: *userID* ⇒ *password*

pendingFReqs :: *userID* ⇒ *userID list*
friendReq :: *userID* ⇒ *userID* ⇒ *requestInfo*
friendIDs :: *userID* ⇒ *userID list*

sentOuterFriendIDs :: *userID* ⇒ (*apiID* × *userID*) *list*
recvOuterFriendIDs :: *userID* ⇒ (*apiID* × *userID*) *list*

postIDs :: *postID list*
post :: *postID* ⇒ *post*
owner :: *postID* ⇒ *userID*
vis :: *postID* ⇒ *vis*

pendingSApiReqs :: *apiID list*
sApiReq :: *apiID* ⇒ *requestInfo*
serverApiIDs :: *apiID list*

serverPass :: *apiID* ⇒ *password*
outerPostIDs :: *apiID* ⇒ *postID list*
outerPost :: *apiID* ⇒ *postID* ⇒ *post*
outerOwner :: *apiID* ⇒ *postID* ⇒ *userID*
outerVis :: *apiID* ⇒ *postID* ⇒ *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),
```

$owner = (\lambda\ pID.\ emptyUserID)$,
$vis = (\lambda\ pID.\ FriendV)$,

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

$pendingCApiReqs = []$,
$cApiReq = (\lambda\ aID.\ emptyRequestInfo)$,
$clientApiIDs = []$,
$clientPass = (\lambda\ aID.\ emptyPass)$,
$sharedWith = (\lambda pID.\ [])$
⦈

3.2.2 Starting action

definition *startSys* ::
$state \Rightarrow userID \Rightarrow password \Rightarrow state$
where
$startSys\ s\ uID\ p \equiv$
s (⦇$admin := uID$,
$userIDs := [uID]$,
$user := (user\ s)\ (uID := emptyUser)$,
$pass := (pass\ s)\ (uID := p)$⦈

definition *e-startSys* :: $state \Rightarrow userID \Rightarrow password \Rightarrow bool$
where
$e\text{-}startSys\ s\ uID\ p \equiv userIDs\ s = []$

3.2.3 Creation actions

definition *createNUReq* :: $state \Rightarrow userID \Rightarrow requestInfo \Rightarrow state$
where
$createNUReq\ s\ uID\ reqInfo \equiv$
s (⦇$pendingUReqs := pendingUReqs\ s\ @\ [uID]$,
$userReq := (userReq\ s)(uID := reqInfo)$
⦈

definition *e-createNUReq* :: $state \Rightarrow userID \Rightarrow requestInfo \Rightarrow bool$
where
$e\text{-}createNUReq\ s\ uID\ requestInfo \equiv$
$admin\ s \in\in userIDs\ s \wedge \neg\ uID \in\in userIDs\ s \wedge \neg\ uID \in\in pendingUReqs\ s$

definition *createUser* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *password* ⇒ *state*
where
createUser s uID p uID′ p′ ≡
s (|*userIDs* := *uID′* # (*userIDs s*),
user := (*user s*) (*uID′* := *emptyUser*),
pass := (*pass s*) (*uID′* := *p′*),
pendingUReqs := *remove1 uID′* (*pendingUReqs s*),
userReq := (*userReq s*)(*uID* := *emptyRequestInfo*)|)

definition *e-createUser* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-createUser s uID p uID′ p′ ≡
IDsOK s [*uID*] [] [] [] ∧ *pass s uID* = *p* ∧ *uID* = *admin s* ∧ *uID′* ∈∈ *pendingUReqs s*

definition *createPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *state*
where
createPost s uID p pID ≡
s (|*postIDs* := *pID* # *postIDs s*,
post := (*post s*) (*pID* := *emptyPost*),
owner := (*owner s*) (*pID* := *uID*)|)

definition *e-createPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *bool*
where
e-createPost s uID p pID ≡
IDsOK s [*uID*] [] [] [] ∧ *pass s uID* = *p* ∧
¬ *pID* ∈∈ *postIDs s*

definition *createFriendReq* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *requestInfo* ⇒ *state*
where
createFriendReq s uID p uID′ req ≡
let pfr = *pendingFReqs s in*
s (|*pendingFReqs* := *pfr* (*uID′* := *pfr uID′* @ [*uID*]),
friendReq := *fun-upd2* (*friendReq s*) *uID uID′ req*|)

definition *e-createFriendReq* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *requestInfo* ⇒ *bool*
where
e-createFriendReq s uID p uID′ req ≡
IDsOK s [*uID*,*uID′*] [] [] [] ∧ *pass s uID* = *p* ∧
¬ *uID* ∈∈ *pendingFReqs s uID′* ∧ ¬ *uID* ∈∈ *friendIDs s uID′*

definition *createFriend* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *state*
where
createFriend s uID p uID′ ≡
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* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *bool*
where
e-createFriend s uID p uID′ ≡
IDsOK s [*uID*,*uID′*] [] [] [] ∧ *pass s uID* = *p* ∧
uID′ ∈∈ *pendingFReqs s uID*

3.2.4 Deletion (removal) actions

definition *deleteFriend* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *state*
where
deleteFriend s uID p uID′ ≡
let fr = *friendIDs s in*
s (|*friendIDs* := *fr* (*uID* := *removeAll uID′* (*fr uID*), *uID′* := *removeAll uID* (*fr uID′*))|)

definition *e-deleteFriend* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *bool*
where
e-deleteFriend s uID p uID′ ≡
IDsOK s [*uID*,*uID′*] [] [] [] ∧ *pass s uID* = *p* ∧
uID′ ∈∈ *friendIDs s uID*

3.2.5 Updating actions

definition *updateUser* :: *state* ⇒ *userID* ⇒ *password* ⇒ *password* ⇒ *name* ⇒ *inform* ⇒ *state*
where
updateUser s uID p p′ name info ≡
s (|*user* := (*user s*) (*uID* := *Usr name info*),
pass := (*pass s*) (*uID* := *p′*)|)

definition *e-updateUser* :: *state* ⇒ *userID* ⇒ *password* ⇒ *password* ⇒ *name* ⇒ *inform* ⇒ *bool*
where
e-updateUser s uID p p′ name info ≡
IDsOK s [*uID*] [] [] [] ∧ *pass s uID* = *p*

definition *updatePost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *post* ⇒ *state*

where
updatePost s uID p pID pst ≡
let sW = *sharedWith s in*
s (|*post* := (*post s*) (*pID* := *pst*),
sharedWith := *sW* (*pID* := *map* (λ (*aID*,-). (*aID*,*False*)) (*sW pID*))|)

definition *e-updatePost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *post* ⇒ *bool*
where
e-updatePost s uID p pID pst ≡
IDsOK s [*uID*] [*pID*] [] [] ∧ *pass s uID* = *p* ∧
owner s pID = *uID*

definition *updateVisPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *vis* ⇒ *state*
where
updateVisPost s uID p pID vs ≡
s (|*vis* := (*vis s*) (*pID* := *vs*)|)

definition *e-updateVisPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *vis* ⇒ *bool*
where
e-updateVisPost s uID p pID vs ≡
IDsOK s [*uID*] [*pID*] [] [] ∧ *pass s uID* = *p* ∧
owner s pID = *uID* ∧ *vs* ∈ {*FriendV*, *PublicV*}

3.2.6 Reading actions

definition *readNUReq* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *requestInfo*
where
readNUReq s uID p uID′ ≡ *userReq s uID′*

definition *e-readNUReq* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *bool*
where
e-readNUReq s uID p uID′ ≡
IDsOK s [*uID*] [] [] [] ∧ *pass s uID* = *p* ∧
uID = *admin s* ∧ *uID′* ∈∈ *pendingUReqs s*

definition *readUser* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *name*
where
readUser s uID p uID′ ≡ *nameUser* (*user s uID′*)

definition *e-readUser* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *bool*
where
e-readUser s uID p uID′ ≡
IDsOK s [*uID*,*uID′*] [] [] [] ∧ *pass s uID* = *p*

definition *readAmIAdmin* :: *state* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
readAmIAdmin s uID p ≡ *uID* = *admin s*

definition *e-readAmIAdmin* :: *state* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-readAmIAdmin s uID p ≡
IDsOK s [*uID*] [] [] [] ∧ *pass s uID* = *p*

definition *readPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *post*
where
readPost s uID p pID ≡ *post s pID*

definition *e-readPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *bool*
where
e-readPost s uID p pID ≡
IDsOK s [*uID*] [*pID*] [] [] ∧ *pass s uID* = *p* ∧
(*owner s pID* = *uID* ∨ *uID* ∈∈ *friendIDs s* (*owner s pID*) ∨ *vis s pID* = *PublicV*)

definition *readOwnerPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *userID*
where
readOwnerPost s uID p pID ≡ *owner s pID*

definition *e-readOwnerPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *bool*
where
e-readOwnerPost s uID p pID ≡
IDsOK s [*uID*] [*pID*] [] [] ∧ *pass s uID* = *p* ∧
(*admin s* = *uID* ∨ *owner s pID* = *uID* ∨ *uID* ∈∈ *friendIDs s* (*owner s pID*) ∨
vis s pID = *PublicV*)

definition *readVisPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *vis*
where
readVisPost s uID p pID ≡ *vis s pID*

definition *e-readVisPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *bool*
where
e-readVisPost s uID p pID ≡
IDsOK s [*uID*] [*pID*] [] [] ∧ *pass s uID* = *p* ∧
(*admin s* = *uID* ∨ *owner s pID* = *uID* ∨ *uID* ∈∈ *friendIDs s* (*owner s pID*) ∨
vis s pID = *PublicV*)

definition *readOPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *postID* ⇒ *post*
where
readOPost s uID p aID pID ≡ *outerPost s aID pID*

definition *e-readOPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *postID* ⇒ *bool*
where
e-readOPost s uID p aID pID ≡

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

definition *readOwnerOPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *postID* ⇒ *userID*
where
readOwnerOPost s uID p aID pID ≡ *outerOwner s aID pID*

definition *e-readOwnerOPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *postID* ⇒ *bool*
where
e-readOwnerOPost s uID p aID pID ≡
$IDsOK\ s\ [uID]\ []\ [(aID,[pID])]\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
$(admin\ s = uID\ \vee\ (aID,outerOwner\ s\ aID\ pID) \in\in recvOuterFriendIDs\ s\ uID\ \vee$
$outerVis\ s\ aID\ pID = PublicV)$

definition *readVisOPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *postID* ⇒ *vis*
where
readVisOPost s uID p aID pID ≡ *outerVis s aID pID*

definition *e-readVisOPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *postID* ⇒ *bool*
where
e-readVisOPost s uID p aID pID ≡
$let\ post = outerPost\ s\ aID\ pID\ in$
$IDsOK\ s\ [uID]\ []\ [(aID,[pID])]\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
$(admin\ s = uID\ \vee\ (aID,outerOwner\ s\ aID\ pID) \in\in recvOuterFriendIDs\ s\ uID\ \vee$
$outerVis\ s\ aID\ pID = PublicV)$

definition *readFriendReqToMe* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *requestInfo*
where
readFriendReqToMe s uID p uID′ ≡ *friendReq s uID′ uID*

definition *e-readFriendReqToMe* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *bool*
where
e-readFriendReqToMe s uID p uID′ ≡
$IDsOK\ s\ [uID,uID']\ []\ []\ []\ \wedge\ pass\ s\ uID = p\ \wedge$
$uID' \in\in pendingFReqs\ s\ uID$

definition *readFriendReqFromMe* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *requestInfo*
where

readFriendReqFromMe s uID p uID′ ≡ friendReq s uID uID′

definition *e-readFriendReqFromMe :: state ⇒ userID ⇒ password ⇒ userID ⇒ bool*
where
e-readFriendReqFromMe s uID p uID′ ≡
IDsOK s [uID,uID′] [] [] [] ∧ pass s uID = p ∧
uID ∈∈ pendingFReqs s uID′

definition *readSApiReq :: state ⇒ userID ⇒ password ⇒ apiID ⇒ requestInfo*
where
readSApiReq s uID p uID′ ≡ sApiReq s uID′

definition *e-readSApiReq :: state ⇒ userID ⇒ password ⇒ apiID ⇒ bool*
where
e-readSApiReq s uID p uID′ ≡
IDsOK s [uID] [] [] [] ∧ pass s uID = p ∧
uID = admin s ∧ uID′ ∈∈ pendingSApiReqs s

definition *readCApiReq :: state ⇒ userID ⇒ password ⇒ apiID ⇒ requestInfo*
where
readCApiReq s uID p uID′ ≡ cApiReq s uID′

definition *e-readCApiReq :: state ⇒ userID ⇒ password ⇒ apiID ⇒ bool*
where
e-readCApiReq s uID p uID′ ≡
IDsOK s [uID] [] [] [] ∧ pass s uID = p ∧
uID = admin s ∧ uID′ ∈∈ pendingCApiReqs s

3.2.7 Listing actions

definition *listPendingUReqs :: state ⇒ userID ⇒ password ⇒ userID list*
where
listPendingUReqs s uID p ≡ pendingUReqs s

definition *e-listPendingUReqs :: state ⇒ userID ⇒ password ⇒ bool*
where
e-listPendingUReqs s uID p ≡
IDsOK s [uID] [] [] [] ∧ pass s uID = p ∧ uID = admin s

definition *listAllUsers :: state ⇒ userID ⇒ password ⇒ userID list*
where
listAllUsers s uID p ≡ userIDs s

definition *e-listAllUsers :: state ⇒ userID ⇒ password ⇒ bool*
where

e-listAllUsers s uID p ≡ *IDsOK s* [*uID*] [] [] [] ∧ *pass s uID* = *p*

definition *listFriends* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *userID list*
where
listFriends s uID p uID′ ≡ *friendIDs s uID′*

definition *e-listFriends* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *bool*
where
e-listFriends s uID p uID′ ≡
IDsOK s [*uID*,*uID′*] [] [] [] ∧ *pass s uID* = *p* ∧
(*uID* = *uID′* ∨ *uID* ∈∈ *friendIDs s uID′*)

definition *listSentOuterFriends* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ (*apiID* × *userID*) *list*
where
listSentOuterFriends s uID p uID′ ≡ *sentOuterFriendIDs s uID′*

definition *e-listSentOuterFriends* :: *state* ⇒ *userID* ⇒ *password* ⇒ *userID* ⇒ *bool*
where
e-listSentOuterFriends s uID p uID′ ≡
IDsOK s [*uID*,*uID′*] [] [] [] ∧ *pass s uID* = *p* ∧
(*uID* = *uID′* ∨ *uID* ∈∈ *friendIDs s uID′*)

definition *listRecvOuterFriends* :: *state* ⇒ *userID* ⇒ *password* ⇒ (*apiID* × *userID*) *list*
where
listRecvOuterFriends s uID p ≡ *recvOuterFriendIDs s uID*

definition *e-listRecvOuterFriends* :: *state* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-listRecvOuterFriends s uID p ≡
IDsOK s [*uID*] [] [] [] ∧ *pass s uID* = *p*

definition *listInnerPosts* :: *state* ⇒ *userID* ⇒ *password* ⇒ (*userID* × *postID*) *list*
where
listInnerPosts s uID p ≡
[(*owner s pID*, *pID*).
pID ← *postIDs s*,
vis s pID ≠ *FriendV* ∨ *uID* ∈∈ *friendIDs s* (*owner s pID*) ∨ *uID* = *owner s pID*
]

definition *e-listInnerPosts* :: *state* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-listInnerPosts s uID p ≡ *IDsOK s* [*uID*] [] [] [] ∧ *pass s uID* = *p*

definition *listOuterPosts* :: *state* ⇒ *userID* ⇒ *password* ⇒ (*apiID* × *postID*) *list*
where
listOuterPosts s uID p ≡
[(*saID*, *pID*).
saID ← *serverApiIDs s*,
pID ← *outerPostIDs s saID*,
outerVis s saID pID = *PublicV* ∨ (*saID*, *outerOwner s saID pID*) ∈∈ *recvOuterFriendIDs s uID*
]

definition *e-listOuterPosts* :: *state* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-listOuterPosts s uID p ≡ *IDsOK s* [*uID*] [] [] [] ∧ *pass s uID* = *p*

definition *listClientsPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ (*apiID* × *bool*) *list*
where
listClientsPost s uID p pID ≡ *sharedWith s pID*

definition *e-listClientsPost* :: *state* ⇒ *userID* ⇒ *password* ⇒ *postID* ⇒ *bool*
where
e-listClientsPost s uID p pID ≡
IDsOK s [*uID*] [*pID*] [] [] ∧ *pass s uID* = *p* ∧ *uID* = *admin s*

definition *listPendingSApiReqs* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID list*
where
listPendingSApiReqs s uID p ≡ *pendingSApiReqs s*

definition *e-listPendingSApiReqs* :: *state* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-listPendingSApiReqs s uID p ≡
IDsOK s [*uID*] [] [] [] ∧ *pass s uID* = *p* ∧ *uID* = *admin s*

definition *listServerAPIs* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID list*
where
listServerAPIs s uID p ≡ *serverApiIDs s*

definition *e-listServerAPIs* :: *state* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-listServerAPIs s uID p ≡
IDsOK s [*uID*] [] [] [] ∧ *pass s uID* = *p* ∧ *uID* = *admin s*

definition *listPendingCApiReqs* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID list*
where
listPendingCApiReqs s uID p ≡ *pendingCApiReqs s*

definition *e-listPendingCApiReqs* :: *state* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-listPendingCApiReqs s uID p ≡
IDsOK s [*uID*] [] [] [] ∧ *pass s uID* = *p* ∧ *uID* = *admin s*

definition *listClientAPIs* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID list*
where
listClientAPIs s uID p ≡ *clientApiIDs s*

definition *e-listClientAPIs* :: *state* ⇒ *userID* ⇒ *password* ⇒ *bool*
where
e-listClientAPIs s uID p ≡
IDsOK s [*uID*] [] [] [] ∧ *pass s uID* = *p* ∧ *uID* = *admin s*

3.2.8 Actions of communication with other APIs

definition *sendServerReq* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *requestInfo*
⇒ (*apiID* × *requestInfo*) × *state*
where
sendServerReq s uID p aID reqInfo ≡
((*aID*,*reqInfo*),
s (|*pendingSApiReqs* := *pendingSApiReqs s* @ [*aID*],
sApiReq := (*sApiReq s*) (*aID* := *reqInfo*)|))

definition *e-sendServerReq* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *requestInfo*
⇒ *bool*
where
e-sendServerReq s uID p aID reqInfo ≡
IDsOK s [*uID*] [] [] [] ∧ *pass s uID* = *p* ∧
uID = *admin s* ∧ ¬ *aID* ∈∈ *pendingSApiReqs s*

definition *receiveClientReq* :: *state* ⇒ *apiID* ⇒ *requestInfo* ⇒ *state*
where
receiveClientReq s aID reqInfo ≡
s (|*pendingCApiReqs* := *pendingCApiReqs s* @ [*aID*],
cApiReq := (*cApiReq s*) (*aID* := *reqInfo*)|)

definition *e-receiveClientReq* :: *state* ⇒ *apiID* ⇒ *requestInfo* ⇒ *bool*
where
e-receiveClientReq s aID reqInfo ≡

$\neg$ *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-sendPost s uID p aID pID ≡
IDsOK s [*uID*] [*pID*] [] [*aID*] ∧ *pass s uID* = *p* ∧
uID = *admin s* ∧ *aID* ∈∈ *clientApiIDs s*

definition *receivePost* :: *state* ⇒ *apiID* ⇒ *password* ⇒ *postID* ⇒ *post* ⇒ *userID* ⇒ *vis* ⇒ *state*
where
receivePost s aID sp pID pst uID vs ≡
let opIDs = *outerPostIDs s in*
s (|*outerPostIDs* := *opIDs* (*aID* := *List.insert pID* (*opIDs aID*)),
outerPost := *fun-upd2* (*outerPost s*) *aID pID pst*,
outerOwner := *fun-upd2* (*outerOwner s*) *aID pID uID*,
outerVis := *fun-upd2* (*outerVis s*) *aID pID vs*|)

definition *e-receivePost* :: *state* ⇒ *apiID* ⇒ *password* ⇒ *postID* ⇒ *post* ⇒ *userID* ⇒ *vis* ⇒ *bool*
where
e-receivePost s aID sp pID nt uID vs ≡
IDsOK s [] [] [(*aID*,[])] [] ∧ *serverPass s aID* = *sp*

definition *sendCreateOFriend* ::
state ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *userID* ⇒ (*apiID* × *password* × *userID* × *userID*) × *state*
where
sendCreateOFriend s uID p aID uID′ ≡
let ofr = *sentOuterFriendIDs s in*
((*aID*, *clientPass s aID*, *uID*, *uID′*),
s (|*sentOuterFriendIDs* := *ofr* (*uID* := *ofr uID* @ [(*aID*,*uID′*)])|))

definition *e-sendCreateOFriend* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *userID* ⇒ *bool*
where
e-sendCreateOFriend s uID p caID uID′ ≡
IDsOK s [*uID*] [] [] [*caID*] ∧ *pass s uID* = *p* ∧
¬ (*caID*,*uID′*) ∈∈ *sentOuterFriendIDs s uID*

definition *receiveCreateOFriend* :: *state* ⇒ *apiID* ⇒ *password* ⇒ *userID* ⇒ *userID* ⇒ *state*
where
receiveCreateOFriend s saID sp uID uID′ ≡

let ofr = recvOuterFriendIDs s in
s (|recvOuterFriendIDs := ofr (uID′ := ofr uID′ @ [(saID,uID)])|)

definition *e-receiveCreateOFriend* :: *state* ⇒ *apiID* ⇒ *password* ⇒ *userID* ⇒ *userID* ⇒ *bool*
where
e-receiveCreateOFriend s saID sp uID uID′ ≡
IDsOK s [] [] [(saID,[])] [] ∧ *serverPass s saID = sp* ∧
¬ *(saID,uID)* ∈∈ *recvOuterFriendIDs s uID′*

definition *sendDeleteOFriend* ::
state ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *userID* ⇒ (*apiID* × *password* × *userID* × *userID*) × *state*
where
sendDeleteOFriend s uID p aID uID′ ≡
let ofr = sentOuterFriendIDs s in
((aID, clientPass s aID, uID, uID′),
s (|sentOuterFriendIDs := ofr (uID := remove1 (aID,uID′) (ofr uID))|))

definition *e-sendDeleteOFriend* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *userID* ⇒ *bool*
where
e-sendDeleteOFriend s uID p caID uID′ ≡
IDsOK s [uID] [] [] [caID] ∧ *pass s uID = p* ∧
(caID,uID′) ∈∈ *sentOuterFriendIDs s uID*

definition *receiveDeleteOFriend* :: *state* ⇒ *apiID* ⇒ *password* ⇒ *userID* ⇒ *userID* ⇒ *state*
where
receiveDeleteOFriend s saID sp uID uID′ ≡
let ofr = recvOuterFriendIDs s in
s (|recvOuterFriendIDs := ofr (uID′ := remove1 (saID,uID) (ofr uID′))|)

definition *e-receiveDeleteOFriend* :: *state* ⇒ *apiID* ⇒ *password* ⇒ *userID* ⇒ *userID* ⇒ *bool*
where
e-receiveDeleteOFriend s saID sp uID uID′ ≡
IDsOK s [] [] [(saID,[])] [] ∧ *serverPass s saID = sp* ∧
(saID,uID) ∈∈ *recvOuterFriendIDs s uID′*

3.3 The step function

datatype *out* =

outOK | *outErr* |

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
 readAmIAdmin-def e-readAmIAdmin-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 ⇒ rActt ⇒ 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 (rAmIAdmin uID p) =
 (if e-readAmIAdmin s uID p then outBool (readAmIAdmin 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* ⇒ (*act* × *out*) ⇒ *com* **where**
comOfO aid (*COMact* (*comSendServerReq uid password aID req*), *ou*) =
(*if aid* ≠ *aID* ∧ *ou* ≠ *outErr then Send else Internal*)
| *comOfO aid* (*COMact* (*comConnectClient uID p aID sp*), *ou*) =
(*if aid* ≠ *aID* ∧ *ou* ≠ *outErr then Send else Internal*)
| *comOfO aid* (*COMact* (*comSendPost uID p aID nID*), *ou*) =
(*if aid* ≠ *aID* ∧ *ou* ≠ *outErr then Send else Internal*)
| *comOfO aid* (*COMact* (*comSendCreateOFriend uID p aID uID′*), *ou*) =
(*if aid* ≠ *aID* ∧ *ou* ≠ *outErr then Send else Internal*)
| *comOfO aid* (*COMact* (*comSendDeleteOFriend uID p aID uID′*), *ou*) =
(*if aid* ≠ *aID* ∧ *ou* ≠ *outErr then Send else Internal*)
| *comOfO aid* (*COMact* (*comReceiveClientReq aID req*), *ou*) =
(*if aid* ≠ *aID* ∧ *ou* ≠ *outErr then Recv else Internal*)
| *comOfO aid* (*COMact* (*comConnectServer aID sp*), *ou*) =
(*if aid* ≠ *aID* ∧ *ou* ≠ *outErr then Recv else Internal*)
| *comOfO aid* (*COMact* (*comReceivePost aID sp nID ntc uid v*), *ou*) =
(*if aid* ≠ *aID* ∧ *ou* ≠ *outErr then Recv else Internal*)
| *comOfO aid* (*COMact* (*comReceiveCreateOFriend aID sp uid uid′*), *ou*) =
(*if aid* ≠ *aID* ∧ *ou* ≠ *outErr then Recv else Internal*)
| *comOfO aid* (*COMact* (*comReceiveDeleteOFriend aID sp uid uid′*), *ou*) =
(*if aid* ≠ *aID* ∧ *ou* ≠ *outErr then Recv else Internal*)
| *comOfO* - - = *Internal*

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

fun *syncO* :: *apiID* ⇒ (*act* × *out*) ⇒ *apiID* ⇒ (*act* × *out*) ⇒ *bool* **where**
syncO aid1 (*COMact* (*comSendServerReq uid p aid req*), *ou1*) *aid2* (*a2*, *ou2*) =
(∃ *req2*. *a2* = (*COMact* (*comReceiveClientReq aid1 req2*)) ∧ *ou1* = *O-sendServerReq* (*aid2*,*req2*) ∧ *ou2* = *outOK*)
| *syncO aid1* (*COMact* (*comConnectClient uid p aid sp*), *ou1*) *aid2* (*a2*, *ou2*) =
(∃ *sp2*. *a2* = (*COMact* (*comConnectServer aid1 sp2*)) ∧ *ou1* = *O-connectClient*

```
(aid2,sp2) ∧ ou2 = outOK)
| syncO aid1 (COMact (comSendPost uid p aid nid), ou1) aid2 (a2, ou2) =
    (∃ sp2 nid2 ntc2 uid2 v. a2 = (COMact (comReceivePost aid1 sp2 nid2 ntc2
uid2 v)) ∧ ou1 = O-sendPost (aid2, sp2, nid2, ntc2, uid2, v) ∧ ou2 = outOK)
| syncO aid1 (COMact (comSendCreateOFriend uid p aid uid'), ou1) aid2 (a2,
ou2) =
    (∃ sp2 uid2 uid2'. a2 = (COMact (comReceiveCreateOFriend aid1 sp2 uid2
uid2')) ∧ ou1 = O-sendCreateOFriend (aid2, sp2, uid2, uid2') ∧ ou2 = outOK)
| syncO aid1 (COMact (comSendDeleteOFriend uid p aid uid'), ou1) aid2 (a2,
ou2) =
    (∃ sp2 uid2 uid2'. a2 = (COMact (comReceiveDeleteOFriend aid1 sp2 uid2
uid2')) ∧ ou1 = O-sendDeleteOFriend (aid2, sp2, uid2, uid2') ∧ ou2 = outOK)
| syncO - - - - = False

fun cmpO :: apiID ⇒ (act × out) ⇒ apiID ⇒ (act × out) ⇒ (apiID × act × out
× apiID × act × out) where
  cmpO aid1 obs1 aid2 obs2 = (aid1, fst obs1, snd obs1, aid2, fst obs2, snd obs2)

fun sync :: apiID ⇒ (state, act, out) trans ⇒ apiID ⇒ (state, act, out) trans ⇒
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* ⇒ (*act* × *out*) ⇒ *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* ⇒ (*state*, *act*, *out*) *trans* ⇒ *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 ∧
postIDs s = postIDs s1 ∧ *admin s = admin s1* ∧
user s = user s1 ∧ *pass s = pass s1* ∧ *pendingFReqs s = pendingFReqs s1* ∧ *friendReq s = friendReq s1* ∧ *friendIDs s = friendIDs s1* ∧
sentOuterFriendIDs s = sentOuterFriendIDs s1 ∧
recvOuterFriendIDs s = recvOuterFriendIDs s1 ∧
post s = post s1 ∧
owner s = owner s1 ∧ *vis s = vis 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* ∧
outerOwner s = outerOwner s1 ∧ *outerVis s = outerVis s1* ∧
pendingCApiReqs s = pendingCApiReqs s1 ∧ *cApiReq s = cApiReq s1* ∧ *clientApiIDs s = clientApiIDs s1* ∧ *clientPass s = clientPass s1* ∧
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* ∈ *Lact ‘ UNIV* ∪ *Ract ‘ UNIV*
shows *snd* (*step s a*) = *s*
⟨*proof*⟩

lemma *Lact-Ract-noStateChange-set*:

assumes *set al* $\subseteq$ *Lact ‘ UNIV* $\cup$ *Ract ‘ UNIV*
shows *snd* (*sstep s al*) = *s*
⟨*proof*⟩

lemma *reach-postIDs-persist*:
pID $\in\in$ *postIDs s* $\Longrightarrow$ *step s a* = (*ou*,*s'*) $\Longrightarrow$ *pID* $\in\in$ *postIDs s'*
⟨*proof*⟩

lemma *userOfA-not-userIDs-outErr*:
$\exists$ *uid*. *userOfA a* = *Some uid* $\wedge$ $\neg$ *uid* $\in\in$ *userIDs s* $\Longrightarrow$
$\forall$ *aID uID p name*. *a* $\neq$ *Sact* (*sSys uID p*) $\Longrightarrow$
$\forall$ *uID name*. *a* $\neq$ *Cact* (*cNUReq uID name*) $\Longrightarrow$
fst (*step s a*) = *outErr*
⟨*proof*⟩

lemma *reach-vis*: *reach s* $\Longrightarrow$ *vis s pID* $\in$ {*FriendV*, *PublicV*}
⟨*proof*⟩

lemma *reach-not-postIDs-emptyPost*:
reach s $\Longrightarrow$ *PID* $\notin$ *set* (*postIDs s*) $\Longrightarrow$ *post s PID* = *emptyPost*
⟨*proof*⟩

lemma *reach-not-postIDs-friendV*:
reach s $\Longrightarrow$ *PID* $\notin$ *set* (*postIDs s*) $\Longrightarrow$ *vis s PID* = *FriendV*
⟨*proof*⟩

lemma *reach-owner-userIDs*: *reach s* $\Longrightarrow$ *pID* $\in\in$ *postIDs s* $\Longrightarrow$ *owner s pID* $\in\in$ *userIDs s*
⟨*proof*⟩

lemma *reach-admin-userIDs*: *reach s* $\Longrightarrow$ *uID* $\in\in$ *userIDs s* $\Longrightarrow$ *admin s* $\in\in$ *userIDs s*
⟨*proof*⟩

lemma *reach-pendingUReqs-distinct*: *reach s* $\Longrightarrow$ *distinct* (*pendingUReqs s*)
⟨*proof*⟩

lemma *reach-pendingUReqs*:
reach s $\Longrightarrow$ *uid* $\in\in$ *pendingUReqs s* $\Longrightarrow$ *uid* $\notin$ *set* (*userIDs s*) $\wedge$ *admin s* $\in\in$ *userIDs s*
⟨*proof*⟩

lemma *reach-friendIDs-symmetric*:
reach s $\Longrightarrow$ *uID1* $\in\in$ *friendIDs s uID2* $\longleftrightarrow$ *uID2* $\in\in$ *friendIDs s uID1*
⟨*proof*⟩

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 $uid' \in\in pendingFReqs\ s\ uid \Longrightarrow uid' \notin set\ (friendIDs\ s\ uid)$
and $uid' \in\in pendingFReqs\ s\ uid \Longrightarrow uid \notin set\ (friendIDs\ s\ uid')$
⟨*proof*⟩

lemma *remove1-in-set*: $x \in\in remove1\ y\ xs \Longrightarrow x \in\in xs$
⟨*proof*⟩

lemma *reach-IDs-used-IDsOK*[*rule-format*]:
assumes *reach s*
shows $uid \in\in pendingFReqs\ s\ uid' \longrightarrow IDsOK\ s\ [uid, uid']\ []\ []\ []$ (**is** *?p*)
and $uid \in\in friendIDs\ s\ uid' \longrightarrow IDsOK\ s\ [uid, uid']\ []\ []\ []$ (**is** *?f*)
⟨*proof*⟩

lemma *reach-AID-used-valid*:
assumes *reach s*
and $aid \in\in serverApiIDs\ s \vee aid \in\in clientApiIDs\ s \vee aid \in\in pendingSApiReqs\ s$
$\vee\ aid \in\in pendingCApiReqs\ s$
shows $admin\ s \in\in userIDs\ s$
⟨*proof*⟩

lemma *IDs-mono*[*rule-format*]:
assumes $step\ s\ a = (ou, s')$
shows $uid \in\in userIDs\ s \longrightarrow uid \in\in userIDs\ s'$ (**is** *?u*)
and $nid \in\in postIDs\ s \longrightarrow nid \in\in postIDs\ s'$ (**is** *?n*)
and $aid \in\in clientApiIDs\ s \longrightarrow aid \in\in clientApiIDs\ s'$ (**is** *?c*)
and $sid \in\in serverApiIDs\ s \longrightarrow sid \in\in serverApiIDs\ s'$ (**is** *?s*)
and $nid \in\in outerPostIDs\ s\ aid \longrightarrow nid \in\in outerPostIDs\ s'\ aid$ (**is** *?o*)
⟨*proof*⟩

lemma *IDsOK-mono*:
assumes $step\ s\ a = (ou, s')$
and *IDsOK s uIDs pIDs saID-pIDs-s caIDs*
shows *IDsOK s' uIDs pIDs saID-pIDs-s caIDs*
⟨*proof*⟩

lemma *step-outerFriendIDs-idem*:
assumes $step\ s\ a = (ou, s')$
and $\forall\ uID\ p\ aID\ uID'.\ a \neq COMact\ (comSendCreateOFriend\ uID\ p\ aID\ uID') \wedge$
$a \neq COMact\ (comReceiveCreateOFriend\ aID\ p\ uID\ uID') \wedge$
$a \neq COMact\ (comSendDeleteOFriend\ uID\ p\ aID\ uID') \wedge$
$a \neq COMact\ (comReceiveDeleteOFriend\ aID\ p\ uID\ uID')$
shows $sentOuterFriendIDs\ s' = sentOuterFriendIDs\ s$ (**is** *?sent*)

and *recvOuterFriendIDs s′ = recvOuterFriendIDs s* (**is** *?recv*)
⟨*proof*⟩

lemma *istate-sSys*:
assumes *step istate a* = (*ou*, *s′*)
obtains *uid p* **where** *a* = *Sact* (*sSys uid p*)
| *s′* = *istate*
⟨*proof*⟩

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

[3]So *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 γ :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
γ (*Trans* - *a* - -) ⟷
(∃ *uid*. *userOfA a* = *Some uid* ∧ *uid* ∈ *UIDs*)
∨
(∃ *ca*. *a* = *COMact ca*)
∨

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

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

fun *comPurge* :: *comActt* ⇒ *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* ⇒ *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*))
|*outPurge ou* = *ou*

fun *g* :: (*state*,*act*,*out*)*trans* ⇒ *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* ⟷ (∃ *p′*. *ca* = *comSendServerReq uID p′ aID reqInfo* ∧ *p* = *emptyPass*)
comPurge ca = *comReceiveClientReq aID reqInfo* ⟷ *ca* = *comReceiveClientReq aID reqInfo*
comPurge ca = *comConnectClient uID p aID sp* ⟷ (∃ *p′*. *ca* = *comConnectClient uID p′ aID sp* ∧ *p* = *emptyPass*)
comPurge ca = *comConnectServer aID sp* ⟷ *ca* = *comConnectServer aID sp*
comPurge ca = *comReceivePost aID sp nID nt uID v* ⟷ *ca* = *comReceivePost aID sp nID nt uID v*
comPurge ca = *comSendPost uID p aID nID* ⟷ (∃ *p′*. *ca* = *comSendPost uID p′ aID nID* ∧ *p* = *emptyPass*)
comPurge ca = *comSendCreateOFriend uID p aID uID′* ⟷ (∃ *p′*. *ca* = *comSendCreateOFriend uID p′ aID uID′* ∧ *p* = *emptyPass*)
comPurge ca = *comReceiveCreateOFriend aID cp uID uID′* ⟷ *ca* = *comRe-*

ceiveCreateOFriend aID cp uID uID′
comPurge ca = comSendDeleteOFriend uID p aID uID′ ⟷ (∃ p′. ca = comSendDeleteOFriend uID p′ aID uID′ ∧ p = emptyPass)
comPurge ca = comReceiveDeleteOFriend aID cp uID uID′ ⟷ ca = comReceiveDeleteOFriend aID cp uID uID′
⟨*proof*⟩

lemma *outPurge-simps*[*simp*]:
outPurge ou = outErr ⟷ ou = outErr
outPurge ou = outOK ⟷ ou = outOK
outPurge ou = O-sendServerReq ossr ⟷ ou = O-sendServerReq ossr
outPurge ou = O-connectClient occ ⟷ ou = O-connectClient occ
outPurge ou = O-sendPost (aid, sp, pid, pst′, uid, vs) ⟷ (∃ pst.
ou = O-sendPost (aid, sp, pid, pst, uid, vs) ∧
pst′ = (if pid = PID then emptyPost else pst))
outPurge ou = O-sendCreateOFriend oscf ⟷ ou = O-sendCreateOFriend oscf
outPurge ou = O-sendDeleteOFriend osdf ⟷ ou = O-sendDeleteOFriend osdf
⟨*proof*⟩

lemma *g-simps*:
g (Trans s a ou s′) = (COMact (comSendServerReq uID p aID reqInfo), O-sendServerReq ossr)
⟷ (∃ p′. a = COMact (comSendServerReq uID p′ aID reqInfo) ∧ p = emptyPass ∧ ou = O-sendServerReq ossr)
g (Trans s a ou s′) = (COMact (comReceiveClientReq aID reqInfo), outOK)
⟷ a = COMact (comReceiveClientReq aID reqInfo) ∧ ou = outOK
g (Trans s a ou s′) = (COMact (comConnectClient uID p aID sp), O-connectClient occ)
⟷ (∃ p′. a = COMact (comConnectClient uID p′ aID sp) ∧ p = emptyPass ∧ ou = O-connectClient occ)
g (Trans s a ou s′) = (COMact (comConnectServer aID sp), outOK)
⟷ a = COMact (comConnectServer aID sp) ∧ ou = outOK
g (Trans s a ou s′) = (COMact (comReceivePost aID sp nID nt uID v), outOK)
⟷ a = COMact (comReceivePost aID sp nID nt uID v) ∧ ou = outOK
g (Trans s a ou s′) = (COMact (comSendPost uID p aID nID), O-sendPost (aid, sp, pid, pst′, uid, vs))
⟷ (∃ pst p′. a = COMact (comSendPost uID p′ aID nID) ∧ p = emptyPass ∧ ou = O-sendPost (aid, sp, pid, pst, uid, vs) ∧ 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′))
⟷ (∃ p′. a = (COMact (comSendCreateOFriend uID p′ aID uID′)) ∧ p = emptyPass ∧ ou = O-sendCreateOFriend (aid, sp, uid, uid′))
g (Trans s a ou s′) = (COMact (comReceiveCreateOFriend aID cp uID uID′), outOK)
⟷ a = COMact (comReceiveCreateOFriend aID cp uID uID′) ∧ ou = outOK
g (Trans s a ou s′) = (COMact (comSendDeleteOFriend uID p aID uID′), O-sendDeleteOFriend (aid, sp, uid, uid′))

$\longleftrightarrow$ ($\exists\, p'.\ a = COMact\ (comSendDeleteOFriend\ uID\ p'\ aID\ uID') \wedge p = emptyPass \wedge ou = O\text{-}sendDeleteOFriend\ (aid,\ sp,\ uid,\ uid')$)
  $g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'),\ outOK)$
$\longleftrightarrow$ $a = COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID') \wedge ou = outOK$
⟨*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 *eeqButPID* **where**
eeqButPID psts psts1 ≡
∀ *pid. if pid = PID then True else psts pid = psts1 pid*

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

lemma *eeqButPID-eeq*[*simp*,*intro*!]: *eeqButPID psts psts*
⟨*proof*⟩

lemma *eeqButPID-sym*:
assumes *eeqButPID psts psts1* **shows** *eeqButPID psts1 psts*
⟨*proof*⟩

lemma *eeqButPID-trans*:
assumes *eeqButPID psts psts1* **and** *eeqButPID psts1 psts2* **shows** *eeqButPID psts psts2*
⟨*proof*⟩

lemma *eeqButPID-cong*:
assumes *eeqButPID psts psts1*
and *pid = PID* ⟹ *eqButT uu uu1*
and *pid* ≠ *PID* ⟹ *uu = uu1*
shows *eeqButPID* (*psts* (*pid* := *uu*)) (*psts1*(*pid* := *uu1*))
⟨*proof*⟩

lemma *eeqButPID-not-PID*:
⟦*eeqButPID psts psts1*; *pid* ≠ *PID*⟧ ⟹ *psts pid = psts1 pid*
⟨*proof*⟩

lemma *eeqButPID-toEq*:
assumes *eeqButPID psts psts1*
shows *psts* (*PID* := *pid*) =
psts1 (*PID* := *pid*)
⟨*proof*⟩

lemma *eeqButPID-update-post*:
assumes *eeqButPID psts psts1*
shows *eeqButPID* (*psts* (*pid* := *pst*)) (*psts1* (*pid* := *pst*))
⟨*proof*⟩

fun *eqButF* :: (*apiID* × *bool*) *list* ⇒ (*apiID* × *bool*) *list* ⇒ *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*
⟨*proof*⟩

lemma *eqButF-sym*:
assumes *eqButF aID-bl aID-bl1*
shows *eqButF aID-bl1 aID-bl*
⟨*proof*⟩

lemma *eqButF-trans*:
assumes *eqButF aID-bl aID-bl1* **and** *eqButF aID-bl1 aID-bl2*
shows *eqButF aID-bl aID-bl2*
⟨*proof*⟩

lemma *eqButF-insert2*:
eqButF aID-bl aID-bl1 ⟹
eqButF (*insert2 aID b aID-bl*) (*insert2 aID b aID-bl1*)
⟨*proof*⟩

definition *eeqButPID-F* **where**
eeqButPID-F sw sw1 ≡
∀ *pid*. *if pid* = *PID then eqButF* (*sw PID*) (*sw1 PID*) *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*
⟨*proof*⟩

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

lemma *eeqButPID-F-eqButF*:
eeqButPID-F sw sw1 ⟹ *eqButF* (*sw PID*) (*sw1 PID*)
⟨*proof*⟩

lemma *eeqButPID-F-not-PID*:
⟦*eeqButPID-F sw sw1*; $pid \neq PID$⟧ ⟹ *sw pid* = *sw1 pid*
⟨*proof*⟩

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

lemma *eeqButPID-F-insert2*:
eeqButPID-F sw sw1 ⟹
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* (λ (*aID*,-). (*aID*,*b*)) (*sw PID*)) =
sw1 (*PID* := *map* (λ (*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* ⇒ *state* ⇒ *bool* **where**
eqButPID 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 ∧
sentOuterFriendIDs s = sentOuterFriendIDs s1 ∧ recvOuterFriendIDs s = recvOuterFriendIDs s1 ∧

postIDs s = postIDs s1 ∧ admin s = admin s1 ∧
eeqButPID (post s) (post s1) ∧
owner s = owner s1 ∧
vis s = vis 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 ∧
outerOwner s = outerOwner s1 ∧
outerVis s = outerVis s1 ∧

pendingCApiReqs s = pendingCApiReqs s1 ∧ cApiReq s = cApiReq s1 ∧
clientApiIDs s = clientApiIDs s1 ∧ clientPass s = clientPass s1 ∧
eeqButPID-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*
⟨*proof*⟩

lemma *eqButPID-sym*:
assumes *eqButPID s s1* **shows** *eqButPID s1 s*
⟨*proof*⟩

lemma *eqButPID-trans*:
assumes *eqButPID s s1* **and** *eqButPID s1 s2* **shows** *eqButPID s s2*
⟨*proof*⟩

lemma *eqButPID-stateSelectors*:
eqButPID 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 ∧
sentOuterFriendIDs s = sentOuterFriendIDs s1 ∧ recvOuterFriendIDs s = recvOuterFriendIDs s1 ∧

postIDs s = postIDs s1 ∧ admin s = admin s1 ∧
eeqButPID (post s) (post s1) ∧
owner s = owner s1 ∧
vis s = vis 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 ∧
outerOwner s = outerOwner s1 ∧
outerVis s = outerVis s1 ∧

pendingCApiReqs s = pendingCApiReqs s1 ∧ cApiReq s = cApiReq s1 ∧
clientApiIDs s = clientApiIDs s1 ∧ clientPass s = clientPass s1 ∧
eeqButPID-F (sharedWith s) (sharedWith s1) ∧

IDsOK s = IDsOK s1
⟨*proof*⟩

lemma *eqButPID-not-PID*:
eqButPID s s1 ⟹ pid ≠ PID ⟹ post s pid = post s1 pid
⟨*proof*⟩

lemma *eqButPID-eqButF*:
eqButPID s s1 ⟹ eqButF (sharedWith s PID) (sharedWith s1 PID)
⟨*proof*⟩

lemma *eqButPID-not-PID-sharedWith*:
eqButPID s s1 ⟹ pid ≠ PID ⟹ sharedWith s pid = sharedWith s1 pid
⟨*proof*⟩

lemma *eqButPID-insert2*:
eqButPID s s1 ⟹
eqButF (insert2 aID b (sharedWith s PID)) (insert2 aID b (sharedWith s1 PID))
⟨*proof*⟩

lemma *eqButPID-actions*:
assumes *eqButPID s s1*
shows *listInnerPosts s uid p = listInnerPosts s1 uid p*
⟨*proof*⟩

lemma *eqButPID-update*:
assumes *eqButPID s s1*
shows (*post s*)(*PID* := *txt*) = (*post s1*)(*PID* := *txt*)
⟨*proof*⟩

lemma *eqButPID-update-post*:
assumes *eqButPID s s1*
shows *eeqButPID* ((*post s*) (*pid* := *pst*)) ((*post s1*) (*pid* := *pst*))
⟨*proof*⟩

lemma *eqButPID-setShared*:
assumes *eqButPID s s1*
shows (*sharedWith s*) (*PID* := *map* (λ (*aID*,-). (*aID*,*b*)) (*sharedWith s PID*)) =
(*sharedWith s1*) (*PID* := *map* (λ (*aID*,-). (*aID*,*b*)) (*sharedWith s1 PID*))
⟨*proof*⟩

lemma *eqButPID-updateShared*:
assumes *eqButPID s s1*
shows *eeqButPID-F* ((*sharedWith s*) (*pid* := *aID-b*)) ((*sharedWith s1*) (*pid* := *aID-b*))
⟨*proof*⟩

lemma *eqButPID-cong*[*simp*]:
⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*admin* := *uu1*|)) (*s1* (|*admin* := *uu2*|))

⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*pendingUReqs* := *uu1*|)) (*s1* (|*pendingUReqs* := *uu2*|))
⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*userReq* := *uu1*|)) (*s1* (|*userReq* := *uu2*|))
⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*userIDs* := *uu1*|)) (*s1* (|*userIDs* := *uu2*|))
⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*user* := *uu1*|)) (*s1* (|*user* := *uu2*|))
⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*pass* := *uu1*|)) (*s1* (|*pass* := *uu2*|))

⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*postIDs* := *uu1*|)) (*s1* (|*postIDs* := *uu2*|))
⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *eeqButPID uu1 uu2* ⟹ *eqButPID* (*s* (|*post* := *uu1*|)) (*s1* (|*post* := *uu2*|))
⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*owner* := *uu1*|)) (*s1* (|*owner* := *uu2*|))
⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*vis* := *uu1*|)) (*s1* (|*vis* := *uu2*|))

⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*pendingFReqs* := *uu1*|)) (*s1* (|*pendingFReqs* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*friendReq* := *uu1*|)) (*s1* (|*friendReq* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*friendIDs* := *uu1*|)) (*s1* (|*friendIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*sentOuterFriendIDs* := *uu1*|)) (*s1* (|*sentOuterFriendIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*recvOuterFriendIDs* := *uu1*|)) (*s1* (|*recvOuterFriendIDs* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*pendingSApiReqs* := *uu1*|)) (*s1* (|*pendingSApiReqs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*sApiReq* := *uu1*|)) (*s1* (|*sApiReq* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*serverApiIDs* := *uu1*|)) (*s1* (|*serverApiIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*serverPass* := *uu1*|)) (*s1* (|*serverPass* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*outerPostIDs* := *uu1*|)) (*s1* (|*outerPostIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*outerPost* := *uu1*|)) (*s1* (|*outerPost* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*outerOwner* := *uu1*|)) (*s1* (|*outerOwner* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*outerVis* := *uu1*|)) (*s1* (|*outerVis* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*pendingCApiReqs* := *uu1*|)) (*s1* (|*pendingCApiReqs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*cApiReq* := *uu1*|)) (*s1* (|*cApiReq* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*clientApiIDs* := *uu1*|)) (*s1* (|*clientApiIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1* = *uu2* $\Longrightarrow$ *eqButPID* (*s* (|*clientPass* := *uu1*|)) (*s1* (|*clientPass* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *eeqButPID-F uu1 uu2* $\Longrightarrow$ *eqButPID* (*s* (|*shared-With* := *uu1*|)) (*s1* (|*sharedWith*:= *uu2*|))
⟨*proof*⟩

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'*
⟨*proof*⟩

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* = [] ⟷ *list-all isPVal vl*
⟨*proof*⟩

fun φ :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
φ (*Trans* - (*Uact* (*uPost uid p pid pst*)) *ou* -) = (*pid* = *PID* ∧ *ou* = *outOK*)
|
φ (*Trans* - (*COMact* (*comSendPost uid p aid pid*)) *ou* -) = (*pid* = *PID* ∧ *ou* ≠ *outErr*)

|
φ (*Trans s* - - *s'*) = *False*

lemma φ*-def2*:
shows
φ (*Trans s a ou s'*) ⟷
(∃ *uid p pst*. *a* = *Uact* (*uPost uid p PID pst*) ∧ *ou* = *outOK*) ∨
(∃ *uid p aid*. *a* = *COMact* (*comSendPost uid p aid PID*) ∧ *ou* ≠ *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* ∧ *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* ≠ *outErr*
shows *ou* = *O-sendPost* (*aid*, *clientPass s aid*, *PID*, *post s PID*, *owner s PID*, *vis s PID*)
∧ *uid* = *admin s* ∧ *p* = *pass s* (*admin s*)

⟨*proof*⟩

lemma *φ-def3*:
assumes *step s a* = (*ou*,*s'*)
shows
φ (*Trans s a ou s'*) ⟷
(∃ *pst*. *a* = *Uact* (*uPost* (*owner s PID*) (*pass s* (*owner s PID*)) *PID pst*) ∧ *ou* = *outOK*) ∨
(∃ *aid*. *a* = *COMact* (*comSendPost* (*admin s*) (*pass s* (*admin s*)) *aid PID*) ∧
ou = *O-sendPost* (*aid*, *clientPass s aid*, *PID*, *post s PID*, *owner s PID*, *vis s PID*))
⟨*proof*⟩

lemma *φ-cases*:
assumes *φ* (*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* ≠ *outErr p* = *pass s uid*
uid = *admin s*
⟨*proof*⟩

fun *f* :: (*state*,*act*,*out*) *trans* ⇒ *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* ⟨*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 *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* ⇒ *bool* **where**
T (*Trans s a ou s'*) ⟷
(∃ *uid* ∈ *UIDs*.
uid ∈∈ *userIDs s'* ∧ *PID* ∈∈ *postIDs s'* ∧

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

fun *corrFrom* :: *post* ⇒ *value list* ⇒ *bool* **where**
corrFrom pst [] = *True*
|*corrFrom pst* (*PVal pstt* # *vl*) = *corrFrom pstt vl*
|*corrFrom pst* (*PValS aid pstt* # *vl*) = (*pst* = *pstt* ∧ *corrFrom pst vl*)

abbreviation *corr* :: *value list* ⇒ *bool* **where** *corr* ≡ *corrFrom emptyPost*

definition *B* :: *value list* ⇒ *value list* ⇒ *bool* **where**
B vl vl1 ≡
corr vl1 ∧
(*vl* = [] ⟶ *vl1* = []) ∧
map PValS-tgtAPI (*filter isPValS vl*) = *map PValS-tgtAPI* (*filter isPValS vl1*)

sublocale *BD-Security-IO* **where**
istate = *istate* **and** *step* = *step* **and**
φ = φ **and** *f* = *f* **and** γ = γ **and** *g* = *g* **and** *T* = *T* **and** *B* = *B*
⟨*proof*⟩

6.1.5 Unwinding proof

lemma *reach-PublicV-imples-FriendV*[*simp*]:
assumes *reach s*
and *vis s pID* ≠ *PublicV*
shows *vis s pID* = *FriendV*
⟨*proof*⟩

lemma *reachNT-state*:
assumes *reachNT s*
shows ¬ (∃ *uid* ∈ *UIDs*.
uid ∈∈ *userIDs s* ∧ *PID* ∈∈ *postIDs s* ∧
(*uid* = *admin s* ∨ *uid* = *owner s PID* ∨ *uid* ∈∈ *friendIDs s* (*owner s PID*) ∨
vis s PID = *PublicV*))
⟨*proof*⟩

lemma *T*-φ-γ:
assumes *1*: *reachNT s* **and** *2*: *step s a* = (*ou*,*s'*)
and *3*: φ (*Trans s a ou s'*) **and**

4: ∀ *ca*. *a* ≠ *COMact ca*
shows ¬ γ (*Trans s a ou s'*)
⟨*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 *sT*: *reachNT s* **and** *T*: ¬ *T* (*Trans s a ou s′*)
and *s1*: *reach s1*
and γ: γ (*Trans s a ou s′*)
shows (∃ *uid p aid pid*. *a* = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨ *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′*
⟨*proof*⟩

definition Δ*0* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*0 s vl s1 vl1* ≡
¬ *PID* ∈∈ *postIDs s* ∧ *post s PID* = *emptyPost* ∧
s = *s1* ∧
corrFrom (*post s1 PID*) *vl1* ∧
(*vl* = [] ⟶ *vl1* = []) ∧
map PValS-tgtAPI (*filter isPValS vl*) = *map PValS-tgtAPI* (*filter isPValS vl1*)

definition Δ*1* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*1 s vl s1 vl1* ≡
PID ∈∈ *postIDs s* ∧
eqButPID s s1 ∧
corrFrom (*post s1 PID*) *vl1* ∧
(*vl* = [] ⟶ *vl1* = []) ∧
map PValS-tgtAPI (*filter isPValS vl*) = *map PValS-tgtAPI* (*filter isPValS vl1*)

definition Δ*2* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*2 s vl s1 vl1* ≡
PID ∈∈ *postIDs s* ∧
eqButPID s s1 ∧
vl = [] ∧ *list-all isPVal vl1*

lemma *istate-Δ0*:
assumes *B*: *B vl vl1*
shows Δ*0 istate vl istate vl1*
⟨*proof*⟩

lemma *unwind-cont-Δ0*: *unwind-cont* Δ*0* {Δ*0*,Δ*1*}
⟨*proof*⟩

lemma *unwind-cont-Δ1*: *unwind-cont* Δ*1* {Δ*1*,Δ*2*}

⟨*proof*⟩

lemma *unwind-cont-Δ2*: *unwind-cont Δ2 {Δ2}*
⟨*proof*⟩

definition *Gr* **where**
Gr =
{
(*Δ0*, {*Δ0*,*Δ1*}),
(*Δ1*, {*Δ1*,*Δ2*}),
(*Δ2*, {*Δ2*})
}

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 γ :: (*state*,*act*,*out*) *trans* $\Rightarrow$ *bool* **where**
γ (*Trans* - *a* - -) $\longleftrightarrow$
(∃ *uid*. *userOfA a* = *Some uid* ∧ *uid* ∈ *UIDs*)
∨
(∃ *ca*. *a* = *COMact ca*)
∨
(∃ *uid p*. *a* = *Sact* (*sSys uid p*))

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

fun *comPurge* :: *comActt* ⇒ *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* ∧ *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* ⇒ *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* $\longleftrightarrow$ (∃ *p'*. *ca* = *comSendServerReq uID p' aID reqInfo* ∧ *p* = *emptyPass*)
comPurge ca = *comReceiveClientReq aID reqInfo* $\longleftrightarrow$ *ca* = *comReceiveClientReq aID reqInfo*
comPurge ca = *comConnectClient uID p aID sp* $\longleftrightarrow$ (∃ *p'*. *ca* = *comConnectClient uID p' aID sp* ∧ *p* = *emptyPass*)
comPurge ca = *comConnectServer aID sp* $\longleftrightarrow$ *ca* = *comConnectServer aID sp*

comPurge ca = comReceivePost aID sp pID pst′ uID v ⟷ (∃ pst. ca = comReceivePost aID sp pID pst uID v ∧ pst′ = (if pID = PID ∧ aID = AID then emptyPost else pst))
comPurge ca = comSendPost uID p aID pID ⟷ (∃ p′. ca = comSendPost uID p′ aID pID ∧ p = emptyPass)
comPurge ca = comSendCreateOFriend uID p aID uID′ ⟷ (∃ p′. ca = comSendCreateOFriend uID p′ aID uID′ ∧ p = emptyPass)
comPurge ca = comReceiveCreateOFriend aID cp uID uID′ ⟷ ca = comReceiveCreateOFriend aID cp uID uID′
comPurge ca = comSendDeleteOFriend uID p aID uID′ ⟷ (∃ p′. ca = comSendDeleteOFriend uID p′ aID uID′ ∧ p = emptyPass)
comPurge ca = comReceiveDeleteOFriend aID cp uID uID′ ⟷ ca = comReceiveDeleteOFriend aID cp uID uID′
⟨*proof*⟩

lemma *g-simps*:
g (Trans s a ou s′) = (COMact (comSendServerReq uID p aID reqInfo), ou′) ⟷ (∃ p′. a = COMact (comSendServerReq uID p′ aID reqInfo) ∧ p = emptyPass ∧ ou = ou′)
g (Trans s a ou s′) = (COMact (comReceiveClientReq aID reqInfo), ou′) ⟷ a = COMact (comReceiveClientReq aID reqInfo) ∧ ou = ou′
g (Trans s a ou s′) = (COMact (comConnectClient uID p aID sp), ou′) ⟷ (∃ p′. a = COMact (comConnectClient uID p′ aID sp) ∧ p = emptyPass ∧ ou = ou′)
g (Trans s a ou s′) = (COMact (comConnectServer aID sp), ou′) ⟷ a = COMact (comConnectServer aID sp) ∧ ou = ou′
g (Trans s a ou s′) = (COMact (comReceivePost aID sp pID pst′ uID v), ou′) ⟷ (∃ pst. a = COMact (comReceivePost aID sp pID pst uID v) ∧ pst′ = (if pID = PID ∧ aID = AID then emptyPost else pst) ∧ ou = ou′)
g (Trans s a ou s′) = (COMact (comSendPost uID p aID nID), O-sendPost (aid, sp, pid, pst, own, v)) ⟷ (∃ p′. a = COMact (comSendPost uID p′ aID nID) ∧ p = emptyPass ∧ ou = O-sendPost (aid, sp, pid, pst, own, v))
g (Trans s a ou s′) = (COMact (comSendCreateOFriend uID p aID uID′), ou′) ⟷ (∃ p′. a = (COMact (comSendCreateOFriend uID p′ aID uID′)) ∧ p = emptyPass ∧ ou = ou′)
g (Trans s a ou s′) = (COMact (comReceiveCreateOFriend aID cp uID uID′), ou′) ⟷ a = COMact (comReceiveCreateOFriend aID cp uID uID′) ∧ ou = ou′
g (Trans s a ou s′) = (COMact (comSendDeleteOFriend uID p aID uID′), ou′) ⟷ (∃ p′. a = COMact (comSendDeleteOFriend uID p′ aID uID′) ∧ p = emptyPass ∧ ou = ou′)
g (Trans s a ou s′) = (COMact (comReceiveDeleteOFriend aID cp uID uID′), ou′) ⟷ a = COMact (comReceiveDeleteOFriend aID cp uID uID′) ∧ ou = ou′
⟨*proof*⟩

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 ≡
∀ aid pid. if aid = AID ∧ 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*
⟨*proof*⟩

lemma *eeqButPID-sym*:
assumes *eeqButPID psts psts1* **shows** *eeqButPID psts1 psts*
⟨*proof*⟩

lemma *eeqButPID-trans*:
assumes *eeqButPID psts psts1* **and** *eeqButPID psts1 psts2* **shows** *eeqButPID psts psts2*
⟨*proof*⟩

lemma *eeqButPID-cong*:
assumes *eeqButPID psts psts1*
and *aid = AID ⟹ pid = PID ⟹ eqButT uu uu1*
and *aid ≠ AID ∨ pid ≠ PID ⟹ uu = uu1*
shows *eeqButPID* (*fun-upd2 psts aid pid uu*) (*fun-upd2 psts1 aid pid uu1*)
⟨*proof*⟩

lemma *eeqButPID-not-PID*:
⟦*eeqButPID psts psts1*; *aid ≠ AID ∨ pid ≠ PID*⟧ *⟹ psts aid pid = psts1 aid pid*
⟨*proof*⟩

lemma *eeqButPID-toEq*:
assumes *eeqButPID psts psts1*
shows *fun-upd2 psts AID PID pst =*
fun-upd2 psts1 AID PID pst
⟨*proof*⟩

lemma *eeqButPID-update-post*:
assumes *eeqButPID psts psts1*
shows *eeqButPID* (*fun-upd2 psts aid pid pst*) (*fun-upd2 psts1 aid pid pst*)
⟨*proof*⟩

fun *eqButF* :: (*apiID* × *bool*) *list* ⇒ (*apiID* × *bool*) *list* ⇒ *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*
⟨*proof*⟩

lemma *eqButF-sym*:
assumes *eqButF aID-bl aID-bl1*
shows *eqButF aID-bl1 aID-bl*
⟨*proof*⟩

lemma *eqButF-trans*:
assumes *eqButF aID-bl aID-bl1* **and** *eqButF aID-bl1 aID-bl2*
shows *eqButF aID-bl aID-bl2*
⟨*proof*⟩

lemma *eqButF-insert2*:
eqButF aID-bl aID-bl1 ⟹
eqButF (*insert2 aID b aID-bl*) (*insert2 aID b aID-bl1*)
⟨*proof*⟩

definition *eqButPID* :: *state* ⇒ *state* ⇒ *bool* **where**
eqButPID 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* ∧
sentOuterFriendIDs s = *sentOuterFriendIDs s1* ∧ *recvOuterFriendIDs s* = *recvOuterFriendIDs s1* ∧

postIDs s = *postIDs s1* ∧ *admin s* = *admin s1* ∧
post s = *post s1* ∧
owner s = *owner s1* ∧
vis s = *vis s1* ∧

pendingSApiReqs s = *pendingSApiReqs s1* ∧ *sApiReq s* = *sApiReq s1* ∧
serverApiIDs s = *serverApiIDs s1* ∧ *serverPass s* = *serverPass s1* ∧

outerPostIDs s = outerPostIDs s1 ∧
eeqButPID (outerPost s) (outerPost s1) ∧
outerOwner s = outerOwner s1 ∧
outerVis s = outerVis s1 ∧

pendingCApiReqs s = pendingCApiReqs s1 ∧ cApiReq s = cApiReq s1 ∧
clientApiIDs s = clientApiIDs s1 ∧ clientPass s = clientPass s1 ∧
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*
⟨*proof*⟩

lemma *eqButPID-sym*:
assumes *eqButPID s s1* **shows** *eqButPID s1 s*
⟨*proof*⟩

lemma *eqButPID-trans*:
assumes *eqButPID s s1* **and** *eqButPID s1 s2* **shows** *eqButPID s s2*
⟨*proof*⟩

lemma *eqButPID-stateSelectors*:
eqButPID 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 ∧
sentOuterFriendIDs s = sentOuterFriendIDs s1 ∧ recvOuterFriendIDs s = recvOuterFriendIDs s1 ∧

postIDs s = postIDs s1 ∧ admin s = admin s1 ∧
post s = post s1 ∧
owner s = owner s1 ∧
vis s = vis s1 ∧

pendingSApiReqs s = pendingSApiReqs s1 ∧ sApiReq s = sApiReq s1 ∧
serverApiIDs s = serverApiIDs s1 ∧ serverPass s = serverPass s1 ∧
outerPostIDs s = outerPostIDs s1 ∧
eeqButPID (outerPost s) (outerPost s1) ∧
outerOwner s = outerOwner s1 ∧
outerVis s = outerVis s1 ∧

pendingCApiReqs s = pendingCApiReqs s1 ∧ cApiReq s = cApiReq s1 ∧
clientApiIDs s = clientApiIDs s1 ∧ clientPass s = clientPass s1 ∧

sharedWith s = sharedWith s1 $\wedge$

IDsOK s = IDsOK s1
⟨*proof*⟩

lemma *eqButPID-not-PID*:
eqButPID s s1 $\Longrightarrow$ *aid* $\neq$ *AID* $\vee$ *pid* $\neq$ *PID* $\Longrightarrow$ *outerPost s aid pid = outerPost s1 aid pid*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *eqButPID-update*:
assumes *eqButPID s s1*
shows *fun-upd2* (*outerPost s*) *AID PID pst = fun-upd2* (*outerPost s1*) *AID PID pst*
⟨*proof*⟩

lemma *eqButPID-update-post*:
assumes *eqButPID s s1*
shows *eeqButPID* (*fun-upd2* (*outerPost s*) *aid pid pst*) (*fun-upd2* (*outerPost s1*) *aid pid pst*)
⟨*proof*⟩

lemma *eqButPID-cong*[*simp*, *intro*]:
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*admin* := *uu1*|)) (*s1* (|*admin* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*pendingUReqs* := *uu1*|)) (*s1* (|*pendingUReqs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*userReq* := *uu1*|)) (*s1* (|*userReq* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*userIDs* := *uu1*|)) (*s1* (|*userIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*user* := *uu1*|)) (*s1* (|*user* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*pass* := *uu1*|)) (*s1* (|*pass* := *uu2*|))

$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*postIDs* := *uu1*|)) (*s1* (|*postIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*post* := *uu1*|)) (*s1* (|*post* := *uu2*|))
$\bigwedge$ *uu1 uu2*. *eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*owner* := *uu1*|)) (*s1* (|*owner* := *uu2*|))

$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*vis* := *uu1*|)) (*s1* (|*vis* := *uu2*|))

$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*pendingFReqs* := *uu1*|)) (*s1* (|*pendingFReqs* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*friendReq* := *uu1*|)) (*s1* (|*friendReq* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*friendIDs* := *uu1*|)) (*s1* (|*friendIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*sentOuterFriendIDs* := *uu1*|)) (*s1* (|*sentOuterFriendIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*recvOuterFriendIDs* := *uu1*|)) (*s1* (|*recvOuterFriendIDs* := *uu2*|))

$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*pendingSApiReqs* := *uu1*|)) (*s1* (|*pendingSApiReqs* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*sApiReq* := *uu1*|)) (*s1* (|*sApiReq* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*serverApiIDs* := *uu1*|)) (*s1* (|*serverApiIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*serverPass* := *uu1*|)) (*s1* (|*serverPass* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*outerPostIDs* := *uu1*|)) (*s1* (|*outerPostIDs* := *uu2*|))

$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *eeqButPID uu1 uu2* $\Longrightarrow$ *eqButPID* (*s* (|*outerPost* := *uu1*|)) (*s1* (|*outerPost* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*outerVis* := *uu1*|)) (*s1* (|*outerVis* := *uu2*|))

$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*outerOwner* := *uu1*|)) (*s1* (|*outerOwner* := *uu2*|))

$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*pendingCApiReqs* := *uu1*|)) (*s1* (|*pendingCApiReqs* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*cApiReq* := *uu1*|)) (*s1* (|*cApiReq* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*clientApiIDs* := *uu1*|)) (*s1* (|*clientApiIDs* := *uu2*|))
$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *eqButPID* (*s* (|*clientPass* := *uu1*|)) (*s1* (|*clientPass* := *uu2*|))

$\bigwedge$ *uu1 uu2. eqButPID s s1* $\Longrightarrow$ *uu1 = uu2* $\Longrightarrow$ *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′*
⟨*proof*⟩

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′*
⟨*proof*⟩

end

end

theory *Post-Value-Setup-RECEIVER*
imports
../Safety-Properties
Post-Observation-Setup-RECEIVER
Post-Unwinding-Helper-RECEIVER
begin

6.2.3 Value setup

locale *Post-RECEIVER = ObservationSetup-RECEIVER*
begin

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

fun *φ* :: *(state,act,out) trans ⇒ bool* **where**
φ (Trans - (COMact (comReceivePost aid sp pid pst uid vs)) ou -) =
(aid = AID ∧ pid = PID ∧ ou = outOK)
|
φ (Trans s - - s′) = False

lemma *φ-def2*:
φ (Trans s a ou s′) ⟷
(∃ uid p pst vs. a = COMact (comReceivePost AID p PID pst uid vs) ∧ ou = outOK)
⟨*proof*⟩

lemma *comReceivePost-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*
⟨*proof*⟩

lemma *φ-def3*:
assumes *step s a* = (*ou*,*s′*)
shows
φ (*Trans s a ou s′*) ⟷
(∃ *uid pst vs*. *a* = *COMact* (*comReceivePost AID* (*serverPass s AID*) *PID pst uid vs*) ∧ *ou* = *outOK*)
⟨*proof*⟩

lemma *φ-cases*:
assumes *φ* (*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*
⟨*proof*⟩

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

sublocale *Receiver-State-Equivalence-Up-To-PID* ⟨*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 *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*) *trans* ⇒ *bool* **where**
T (*Trans s a ou s'*) ⟷
(∃ *uid* ∈ *UIDs*.
uid ∈∈ *userIDs s'* ∧ *PID* ∈∈ *outerPostIDs s' AID* ∧
(*uid* = *admin s'* ∨
(*AID*,*outerOwner s' AID PID*) ∈∈ *recvOuterFriendIDs s' uid* ∨
outerVis s' AID PID = *PublicV*))

definition B :: *value list* ⇒ *value list* ⇒ *bool* **where**
B *vl vl1* ≡ *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** $B = B$
⟨*proof*⟩

6.2.5 Unwinding proof

lemma *reach-PublicV-imples-FriendV*[*simp*]:
assumes *reach s*
and *vis s pID* ≠ *PublicV*
shows *vis s pID* = *FriendV*
⟨*proof*⟩

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*))
⟨*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 *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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 0$: *unwind-cont* $\Delta 0$ {$\Delta 0$,$\Delta 1$}
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 1$: *unwind-cont* $\Delta 1$ {$\Delta 1$}
⟨*proof*⟩

definition *Gr* **where**
Gr =

{
(Δ*0*, {Δ*0*,Δ*1*}),
(Δ*1*, {Δ*1*})
}

theorem *Post-secure*: *secure*
⟨*proof*⟩

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* × *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 *φ1* ≡ *Iss.φ* **abbreviation** *f1* ≡ *Iss.f* **abbreviation** *γ1* ≡ *Iss.γ*
abbreviation *g1* ≡ *Iss.g*
abbreviation *T1* ≡ *Iss.T* **abbreviation** *B1* ≡ *Iss.B*
abbreviation *φ2* ≡ *Rcv.φ* **abbreviation** *f2* ≡ *Rcv.f* **abbreviation** *γ2* ≡ *Rcv.γ*
abbreviation *g2* ≡ *Rcv.g*
abbreviation *T2* ≡ *Rcv.T* **abbreviation** *B2* ≡ *Rcv.B*

fun *isCom1* :: *ttrans* ⇒ *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* ⇒ *obs2* ⇒ *bool* **where**
syncO (*COMact ca1*, *ou1*) (*COMact ca2*, *ou2*) =
(*ou1* ≠ *outErr* ∧ *ou2* ≠ *outErr* ∧
(*comSyncOA ou1 ca2* ∨ *comSyncOA ou2 ca1*)
)
|*syncO* - - = *False*

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

definition *cmpO* :: *obs1* ⇒ *obs2* ⇒ *cobs* **where**
cmpO o1 o2 ≡ (*o1*,*o2*)

lemma *isCom1-isComV1*:
assumes *validTrans trn1* **and** *reach* (*srcOf trn1*) **and** *φ1 trn1*
shows *isCom1 trn1* ⟷ *isComV1* (*f1 trn1*)
⟨*proof*⟩

lemma *isCom1-isComO1*:
assumes *validTrans trn1* **and** *reach* (*srcOf trn1*) **and** *γ1 trn1*
shows *isCom1 trn1* ⟷ *isComO1* (*g1 trn1*)
⟨*proof*⟩

lemma *isCom2-isComV2*:
assumes *validTrans trn2* **and** *reach* (*srcOf trn2*) **and** *φ2 trn2*
shows *isCom2 trn2* ⟷ *isComV2* (*f2 trn2*)
⟨*proof*⟩

lemma *isCom2-isComO2*:
assumes *validTrans trn2* **and** *reach* (*srcOf trn2*) **and** *γ2 trn2*
shows *isCom2 trn2* ⟷ *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** *φ1 trn1* **and** *φ2 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** $\gamma 1$ *trn1* **and** $\gamma 2$ *trn2*
and *sync trn1 trn2*
shows *syncO* (*g1 trn1*) (*g2 trn2*)
⟨*proof*⟩

lemma *sync-*$\varphi 1$*-*$\varphi 2$:
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* $\Longrightarrow$ $\varphi 2$ *trn2* $\Longrightarrow$ *syncV* (*f1 trn1*) (*f2 trn2*)
shows *sync trn1 trn2*
⟨*proof*⟩

lemma *isCom1-*$\gamma 1$:
assumes *validTrans trn1* **and** *reach* (*srcOf trn1*) **and** *isCom1 trn1*
shows $\gamma 1$ *trn1*
⟨*proof*⟩

lemma *isCom2-*$\gamma 2$:
assumes *validTrans trn2* **and** *reach* (*srcOf trn2*) **and** *isCom2 trn2*
shows $\gamma 2$ *trn2*
⟨*proof*⟩

lemma *isCom2-V2*:
assumes *validTrans trn2* **and** *reach* (*srcOf trn2*) **and** $\varphi 2$ *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 *φ1 = φ1* **and** *f1 = f1* **and** *γ1 = γ1* **and** *g1 = g1* **and** *T1 = T1* **and** *B1 = B1*
 and
 istate2 = istate **and** *validTrans2 = validTrans* **and** *srcOf2 = srcOf* **and** *tgtOf2 = tgtOf*
  and *φ2 = φ2* **and** *f2 = f2* **and** *γ2 = γ2* **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* ⇒ *userID set*
 and *AID* :: *apiID* **and** *PID* :: *postID*
 assumes *AID-in-AIDs*: *AID* ∈ *AIDs*

begin

sublocale *Iss*: *Post-ISSUER 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*)

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-ISSUER.PValS aid*′ *pst*)) = (*if aid*′ ≠ *aid then Send else Internal*)
| *comOfV aid* (*Inl* (*Post-ISSUER.PVal pst*)) = *Internal*
| *comOfV aid* (*Inr v*) = *Recv*

fun *tgtNodeOfV* :: *apiID* ⇒ *value* ⇒ *apiID* **where**
  tgtNodeOfV aid (*Inl* (*Post-ISSUER.PValS aid*′ *pst*)) = *aid*′
| *tgtNodeOfV aid* (*Inl* (*Post-ISSUER.PVal pst*)) = *undefined*
| *tgtNodeOfV aid* (*Inr v*) = *AID*

definition *syncV* :: *apiID* ⇒ *value* ⇒ *apiID* ⇒ *value* ⇒ *bool* **where**
  syncV aid1 v1 aid2 v2 =
    (∃ *pst*. *aid1* = *AID* ∧ *v1* = *Inl* (*Post-ISSUER.PValS aid2 pst*) ∧ *v2* = *Inr* (*Post-RECEIVER.PValR pst*))

lemma *syncVI*: *syncV AID* (*Inl* (*Post-ISSUER.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-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* ⟷ *isl v* ∧ *Iss.isPValS* (*projl v*) ∧ *Iss.PValS-tgtAPI* (*projl v*) ≠ *AID*
  comOfV AID v = *Recv* ⟷ *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** *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*
⟨*proof*⟩

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

context
fixes *AID′* :: *apiID*
assumes *AID′*: *AID′* ∈ *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* ⟹ *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* ⟹ *these* (*map* (*Some o Inr o projr*) *vl*) = *vl*
⟨*proof*⟩

sublocale *BD-Security-TS-Network-Preserve-Source-Security-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*
⟨*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 ≡
∃ *uid* ∈ *UIDs*.
  uid ∈∈ *userIDs s* ∧ *PID* ∈∈ *postIDs s* ∧
  (*uid* = *admin s* ∨ *uid* = *owner s PID* ∨ *uid* ∈∈ *friendIDs s* (*owner s PID*) ∨
   vis s PID = *PublicV*)

sublocale *Issuer-State-Equivalence-Up-To-PID* ⟨*proof*⟩

lemma *eqButPID-open*:
assumes *eqButPID s s1*
shows *open s* ⟷ *open s1*
⟨*proof*⟩

lemma *not-open-eqButPID*:
assumes *1*: ¬ *open s* **and** *2*: *eqButPID s s1*
shows ¬ *open s1*
⟨*proof*⟩

lemma *step-isCOMact-open*:
assumes *step s a* = (*ou*, *s'*)
and *isCOMact a*
shows *open s'* = *open s*
⟨*proof*⟩

lemma *validTrans-isCOMact-open*:
assumes *validTrans trn*
and *isCOMact* (*actOf trn*)
shows *open* (*tgtOf trn*) = *open* (*srcOf trn*)
⟨*proof*⟩

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

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

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

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

lemma *filter-list-all-isPVal-isOVal*:
assumes *list-all* (*Not* ∘ *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* ∘ *isOVal*) *ul* ∧ *filter isPVal ul* = []
⟨*proof*⟩

lemma *filter-isTValS-Nil*:
filter isPValS vl = [] ⟷
list-all (λ *v*. *isPVal v* ∨ *isOVal v*) *vl*
⟨*proof*⟩

fun φ :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
φ (*Trans* - (*Uact* (*uPost uid p pid pst*)) *ou* -) = (*pid* = *PID* ∧ *ou* = *outOK*)
|
φ (*Trans* - (*COMact* (*comSendPost uid p aid pid*)) *ou* -) = (*pid* = *PID* ∧ *ou* ≠ *outErr*)

|
φ (*Trans s* - - *s'*) = (*open s* ≠ *open s'*)

lemma *φ-def2*:

assumes *step s a* = (*ou*,*s'*)
shows
φ (*Trans s a ou s'*) $\longleftrightarrow$
(∃ *uid p pst. a* = *Uact* (*uPost uid p PID pst*) ∧ *ou* = *outOK*) ∨
(∃ *uid p aid. a* = *COMact* (*comSendPost uid p aid PID*) ∧ *ou* ≠ *outErr*) ∨
open s ≠ *open s'*
⟨*proof*⟩

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* ∧ *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* ≠ *outErr*
shows *ou* = *O-sendPost* (*aid*, *clientPass s aid*, *PID*, *post s PID*, *owner s PID*, *vis s PID*)
∧ *uid* = *admin s* ∧ *p* = *pass s* (*admin s*)
⟨*proof*⟩

lemma *step-open-isCOMact*:
assumes *step s a* = (*ou*,*s'*)
and *open s* ≠ *open s'*
shows ¬ *isCOMact a* ∧ ¬ (∃ *ua. isuPost ua* ∧ *a* = *Uact ua*)
⟨*proof*⟩

lemma φ*-def3*:
assumes *step s a* = (*ou*,*s'*)
shows
φ (*Trans s a ou s'*) $\longleftrightarrow$
(∃ *pst. a* = *Uact* (*uPost* (*owner s PID*) (*pass s* (*owner s PID*)) *PID pst*) ∧ *ou* = *outOK*)
∨
(∃ *aid. a* = *COMact* (*comSendPost* (*admin s*) (*pass s* (*admin s*)) *aid PID*) ∧
ou = *O-sendPost* (*aid*, *clientPass s aid*, *PID*, *post s PID*, *owner s PID*, *vis s PID*))
∨
open s ≠ *open s'* ∧ ¬ *isCOMact a* ∧ ¬ (∃ *ua. isuPost ua* ∧ *a* = *Uact ua*)
⟨*proof*⟩

fun *f* :: (*state*,*act*,*out*) *trans* ⇒ *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* ≠ *open s'* ∧ ¬ *isCOMact a* ∧ ¬ (∃ *ua*. *isuPost ua* ∧ *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** *φ* (*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** *φ* (*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** *φ* (*Trans s a ou s'*)
and *f* (*Trans s a ou s'*) = *OVal b*
shows *open s'* ≠ *open s*
⟨*proof*⟩

lemma *uPost-comSendPost-open-eq*:
assumes *step*: *step s a* = (*ou*, *s'*)
and *a*: *a* = *Uact* (*uPost uid p pid pst*) ∨ *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* ∈ *set* (*postIDs s*)
and *op*: ¬ *open s* **and** *fi*: *φ* (*Trans s a ou s'*) **and** *f*: *f* (*Trans s a ou s'*) = *PVal pst*
shows ¬ *γ* (*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* ⇒ *bool* **where** T - = *False*

inductive *BC* :: *value list* ⇒ *value list* ⇒ *bool*
and *BO* :: *value list* ⇒ *value list* ⇒ *bool*
where
BC-PVal[*simp*,*intro*!]:
list-all (*Not o isOVal*) *ul* ⟹ *list-all* (*Not o isOVal*) *ul1* ⟹
map tgtAPI (*filter isPValS ul*) = *map tgtAPI* (*filter isPValS ul1*) ⟹
(*ul* = [] ⟶ *ul1* = [])
⟹ *BC ul ul1*
|*BC-BO*[*intro*]:
BO vl vl1 ⟹
list-all (*Not o isOVal*) *ul* ⟹ *list-all* (*Not o isOVal*) *ul1* ⟹
map tgtAPI (*filter isPValS ul*) = *map tgtAPI* (*filter isPValS ul1*) ⟹
(*ul* = [] ⟷ *ul1* = []) ⟹
(*ul* ≠ [] ⟹ *isPVal* (*last ul*) ∧ *last ul* = *last ul1*) ⟹
list-all isPValS sul
⟹
BC (*ul* @ *sul* @ *OVal True* # *vl*)
(*ul1* @ *sul* @ *OVal True* # *vl1*)

|*BO-PVal*[*simp*,*intro*!]:
list-all (*Not o isOVal*) *ul* ⟹ *BO ul ul*
|*BO-BC*[*intro*]:
BC vl vl1 ⟹
list-all (*Not o isOVal*) *ul*
⟹
BO (*ul* @ *OVal False* # *vl*) (*ul* @ *OVal False* # *vl1*)

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

lemma *BC-not-Nil*: *BC vl vl1* ⟹ *vl* = [] ⟹ *vl1* = []
⟨*proof*⟩

lemma *BC-OVal-True*:
assumes *BC* (*OVal True* # *vl'*) *vl1*
shows ∃ *vl1'*. *BO vl' vl1'* ∧ *vl1* = *OVal True* # *vl1'*
⟨*proof*⟩

fun *corrFrom* :: *post* ⇒ *value list* ⇒ *bool* **where**
corrFrom pst [] = *True*
|*corrFrom pst* (*PVal pstt* # *vl*) = *corrFrom pstt vl*
|*corrFrom pst* (*PValS aid pstt* # *vl*) = (*pst* = *pstt* ∧ *corrFrom pst vl*)
|*corrFrom pst* (*OVal b* # *vl*) = (*corrFrom pst vl*)

abbreviation *corr* :: *value list* ⇒ *bool* **where** *corr* ≡ *corrFrom emptyPost*

definition *B* **where**
B vl vl1 ≡ *BC vl vl1* ∧ *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* ≠ *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*: ¬ *open s*
and *sT*: *reachNT s* **and** *s1*: *reach s1*
and γ: γ (*Trans s a ou s'*)
shows (∃ *uid p aid pid*. *a* = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨
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'*
⟨*proof*⟩

definition Δ*0* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*0 s vl s1 vl1* ≡
¬ *PID* ∈∈ *postIDs s* ∧
s = *s1* ∧ *BC vl vl1* ∧
corr vl1

definition Δ*1* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*1 s vl s1 vl1* ≡
PID ∈∈ *postIDs s* ∧
list-all (*Not o isOVal*) *vl* ∧ *list-all* (*Not o isOVal*) *vl1* ∧
map tgtAPI (*filter isPValS vl*) = *map tgtAPI* (*filter isPValS vl1*) ∧
(*vl* = [] ⟶ *vl1* = []) ∧
eqButPID s s1 ∧ ¬ *open s* ∧
corrFrom (*post s1 PID*) *vl1*

definition Δ*11* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*11 s vl s1 vl1* ≡
PID ∈∈ *postIDs s* ∧
vl = [] ∧ *list-all isPVal vl1* ∧
eqButPID s s1 ∧ ¬ *open s* ∧
corrFrom (*post s1 PID*) *vl1*

definition Δ*2* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*2 s vl s1 vl1* ≡
PID ∈∈ *postIDs s* ∧
list-all (*Not o isOVal*) *vl* ∧
vl = *vl1* ∧
s = *s1* ∧ *open s* ∧
corrFrom (*post s1 PID*) *vl1*

definition Δ*31* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*31 s vl s1 vl1* ≡
PID ∈∈ *postIDs s* ∧
(∃ *ul ul1 sul vll vll1*.
BO vll vll1 ∧
list-all (*Not o isOVal*) *ul* ∧ *list-all* (*Not o isOVal*) *ul1* ∧
map tgtAPI (*filter isPValS ul*) = *map tgtAPI* (*filter isPValS ul1*) ∧
ul ≠ [] ∧ *ul1* ≠ [] ∧
isPVal (*last ul*) ∧ *last ul* = *last ul1* ∧
list-all isPValS sul ∧
vl = *ul* @ *sul* @ *OVal True* # *vll* ∧ *vl1* = *ul1* @ *sul* @ *OVal True* # *vll1*) ∧
eqButPID s s1 ∧ ¬ *open s* ∧
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*.
BO vll vll1 $\wedge$
list-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 $\Delta 4$ :: *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*.
BC vll vll1 $\wedge$
list-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-*$\Delta 0$:
assumes *B*: *B vl vl1*
shows $\Delta 0$ *istate vl istate vl1*
⟨*proof*⟩

lemma *list-all-filter*[*simp*]:
assumes *list-all PP xs*
shows *filter PP xs* = *xs*
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 0$: *unwind-cont* $\Delta 0$ {$\Delta 0$,$\Delta 1$,$\Delta 2$,$\Delta 31$,$\Delta 32$,$\Delta 4$}
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 1$: *unwind-cont* $\Delta 1$ {$\Delta 1$,$\Delta 11$}
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 11$: *unwind-cont* $\Delta 11$ {$\Delta 11$}
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 31$: *unwind-cont* $\Delta 31$ {$\Delta 31$,$\Delta 32$}
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 32$: *unwind-cont* $\Delta 32$ {$\Delta 2$,$\Delta 32$,$\Delta 4$}
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 2$: *unwind-cont* $\Delta 2$ {$\Delta 2$}
⟨*proof*⟩

lemma *unwind-cont-Δ4*: *unwind-cont Δ4 {Δ1,Δ31,Δ32,Δ4}*
⟨*proof*⟩

definition *Gr* **where**
Gr =
{
(Δ*0*, {Δ*0*,Δ*1*,Δ*2*,Δ*31*,Δ*32*,Δ*4*}),
(Δ*1*, {Δ*1*,Δ*11*}),
(Δ*11*, {Δ*11*}),
(Δ*2*, {Δ*2*}),
(Δ*31*, {Δ*31*,Δ*32*}),
(Δ*32*, {Δ*2*,Δ*32*,Δ*4*}),
(Δ*4*, {Δ*1*,Δ*31*,Δ*32*,Δ*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 $\varphi 1 \equiv Iss.\varphi$ **abbreviation** $f1 \equiv Iss.f$ **abbreviation** $\gamma 1 \equiv Iss.\gamma$
abbreviation $g1 \equiv Iss.g$
abbreviation $T1 \equiv Iss.T$ **abbreviation** $B1 \equiv Iss.B$
abbreviation $\varphi 2 \equiv Rcv.\varphi$ **abbreviation** $f2 \equiv Rcv.f$ **abbreviation** $\gamma 2 \equiv Rcv.\gamma$
abbreviation $g2 \equiv Rcv.g$
abbreviation $T2 \equiv Rcv.T$ **abbreviation** $B2 \equiv Rcv.B$

fun *isCom1* :: *ttrans* ⇒ *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 pst1*) = *True*
|*isComV1* - = *False*

fun *isComV2* :: *value2* ⇒ *bool* **where**
isComV2 (*Rcv.PValR pst2*) = *True*

fun *syncV* :: *value1* ⇒ *value2* ⇒ *bool* **where**
syncV (*Iss.PValS aud1 pst1*) (*Rcv.PValR pst2*) = (*pst1* = *pst2*)
|*syncV* - - = *False*

fun *cmpV* :: *value1* ⇒ *value2* ⇒ *cval* **where**
cmpV (*Iss.PValS aid1 pst1*) (*Rcv.PValR pst2*) = *PValC pst1*
|*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 reqInfo2*) =
(*aid1* = *AID1* ∧ *aid2* = *AID2* ∧ *reqInfo1* = *reqInfo2*)
|*comSyncOA* (*O-connectClient* (*aid2*, *sp1*)) (*comConnectServer aid1 sp2*) =
(*aid1* = *AID1* ∧ *aid2* = *AID2* ∧ *sp1* = *sp2*)

$|comSyncOA\ (O\text{-}sendPost\ (aid2,\ sp1,\ pid1,\ pst1,\ uid1,\ vs1))\ (comReceivePost\ aid1\ sp2\ pid2\ pst2\ uid2\ vs2) =$
$\quad (aid1 = AID1 \wedge aid2 = AID2 \wedge (pid1,\ pst1,\ uid1,\ vs1) = (pid2,\ pst2,\ uid2,\ vs2))$
$|comSyncOA\ (O\text{-}sendCreateOFriend\ (aid2,\ sp1,\ uid1,\ uid1')) (comReceiveCreateOFriend\ aid1\ sp2\ uid2\ uid2') =$
$\quad (aid1 = AID1 \wedge aid2 = AID2 \wedge (uid1,\ uid1') = (uid2,\ uid2'))$
$|comSyncOA\ (O\text{-}sendDeleteOFriend\ (aid2,\ sp1,\ uid1,\ uid1')) (comReceiveDeleteOFriend\ aid1\ sp2\ uid2\ uid2') =$
$\quad (aid1 = AID1 \wedge aid2 = AID2 \wedge (uid1,\ uid1') = (uid2,\ uid2'))$
$|comSyncOA\ -\ -\ = False$

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

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 *snc*: *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-*$\varphi 1$*-*$\varphi 2$:
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* $\Longrightarrow$ $\varphi 2$ *trn2* $\Longrightarrow$ *syncV* (*f1 trn1*) (*f2 trn2*)
shows *sync trn1 trn2*
⟨*proof*⟩

lemma *isCom1-*$\gamma 1$:
assumes *validTrans trn1* **and** *reach* (*srcOf trn1*) **and** *isCom1 trn1*
shows $\gamma 1$ *trn1*
⟨*proof*⟩

lemma *isCom2-*$\gamma 2$:
assumes *validTrans trn2* **and** *reach* (*srcOf trn2*) **and** *isCom2 trn2*

shows *γ2 trn2*
⟨*proof*⟩

lemma *isCom2-V2*:
assumes *validTrans trn2* **and** *reach* (*srcOf trn2*) **and** *φ2 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 *φ1* = *φ1* **and** *f1* = *f1* **and** *γ1* = *γ1* **and** *g1* = *g1* **and** *T1* = *T1* **and** *B1* = *B1*
and
istate2 = *istate* **and** *validTrans2* = *validTrans* **and** *srcOf2* = *srcOf* **and** *tgtOf2* = *tgtOf*
and *φ2* = *φ2* **and** *f2* = *f2* **and** *γ2* = *γ2* **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 *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 =
  (∃ *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*
⟨*proof*⟩

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

context
fixes *AID′* :: *apiID*
assumes *AID′*: *AID′* ∈ *AIDs* − {*AID*}
begin

interpretation *Recv*: *Post-RECEIVER UIDs AID′ PID AID* ⟨*proof*⟩

lemma *Iss-BC-BO-tgtAPI*:
shows (*Iss.BC vl vl1* ⟶ *map Iss.tgtAPI* (*filter Iss.isPValS vl*) =
map Iss.tgtAPI (*filter Iss.isPValS vl1*)) ∧
(*Iss.BO vl vl1* ⟶ *map Iss.tgtAPI* (*filter Iss.isPValS vl*) =
map Iss.tgtAPI (*filter Iss.isPValS vl1*))
⟨*proof*⟩

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*))
⟨*proof*⟩

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*))
⟨*proof*⟩

end

lemma *map-projl-Inl*: *map* (*projl o Inl*) *vl* = *vl*
⟨*proof*⟩

lemma *these-map-Inl-projl*: *list-all isl vl* ⟹ *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* ⟹ *these* (*map* (*Some o Inr o projr*) *vl*) = *vl*

⟨*proof*⟩

sublocale *BD-Security-TS-Network-Preserve-Source-Security-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*
⟨*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* ⇒ *bool* **where**
γ (*Trans* - *a* - -) ⟷
(∃ *uid*. *userOfA a* = *Some uid* ∧ *uid* ∈ *UIDs*)
∨
— Communication actions are considered to be observable in order to make the security properties compositional
(∃ *ca*. *a* = *COMact ca*)
∨
— 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).

$(\exists\, uid\ p\ pid\ pst.\ a = Uact\ (uPost\ uid\ p\ pid\ pst) \wedge pid \neq PID)$
$\vee$
$(\exists\, uid\ p.\ a = Sact\ (sSys\ uid\ p))$
$\vee$
$(\exists\, uid\ p\ uid'\ p'.\ a = Cact\ (cUser\ uid\ p\ uid'\ p'))$
$\vee$
$(\exists\, uid\ p\ pid.\ a = Cact\ (cPost\ uid\ p\ pid))$
$\vee$
$(\exists\, uid\ p\ uid'.\ a = Cact\ (cFriend\ uid\ p\ uid'))$
$\vee$
$(\exists\, uid\ p\ uid'.\ a = Dact\ (dFriend\ uid\ p\ uid'))$
$\vee$
$(\exists\, uid\ p\ pid\ v.\ a = Uact\ (uVisPost\ uid\ p\ pid\ v))$

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

fun *comPurge* :: *comActt* ⇒ *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* ⇒ *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*))
|*outPurge ou* = *ou*

fun *g* :: (*state*,*act*,*out*)*trans* ⇒ *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*

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendPost\ uID\ p\ aID\ nID),\ O\text{-}sendPost\ (aid,\ sp,\ pid,\ pst',\ uid,\ vs))$
$\longleftrightarrow (\exists\, pst\ p'.\ a = COMact\ (comSendPost\ uID\ p'\ aID\ nID) \wedge p = emptyPass \wedge ou = O\text{-}sendPost\ (aid,\ sp,\ pid,\ pst,\ uid,\ vs) \wedge pst' = (if\ pid = PID\ then\ emptyPost\ else\ pst))$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendCreateOFriend\ uID\ p\ aID\ uID'),\ O\text{-}sendCreateOFriend\ (aid,\ sp,\ uid,\ uid'))$
$\longleftrightarrow (\exists\, p'.\ a = (COMact\ (comSendCreateOFriend\ uID\ p'\ aID\ uID')) \wedge p = emptyPass \wedge ou = O\text{-}sendCreateOFriend\ (aid,\ sp,\ uid,\ uid'))$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID'),\ outOK)$
$\longleftrightarrow a = COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID') \wedge ou = outOK$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendDeleteOFriend\ uID\ p\ aID\ uID'),\ O\text{-}sendDeleteOFriend\ (aid,\ sp,\ uid,\ uid'))$
$\longleftrightarrow (\exists\, p'.\ a = COMact\ (comSendDeleteOFriend\ uID\ p'\ aID\ uID') \wedge p = emptyPass \wedge ou = O\text{-}sendDeleteOFriend\ (aid,\ sp,\ uid,\ uid'))$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'),\ outOK)$
$\longleftrightarrow a = COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID') \wedge ou = outOK$
$\langle proof \rangle$

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 ≡
∃ *uid* ∈ *UIDs*.
uid ∈∈ *userIDs s* ∧ *PID* ∈∈ *postIDs s* ∧
(*uid* = *admin s* ∨ *uid* = *owner s PID* ∨ *uid* ∈∈ *friendIDs s* (*owner s PID*) ∨
vis s PID = *PublicV*)

sublocale *Issuer-State-Equivalence-Up-To-PID* ⟨*proof*⟩

lemma *eqButPID-open*:
assumes *eqButPID s s1*
shows *open s* ⟷ *open s1*
⟨*proof*⟩

lemma *not-open-eqButPID*:
assumes *1*: ¬ *open s* **and** *2*: *eqButPID s s1*
shows ¬ *open s1*
⟨*proof*⟩

lemma *step-isCOMact-open*:
assumes *step s a* = (*ou*, *s'*)
and *isCOMact a*
shows *open s'* = *open s*
⟨*proof*⟩

lemma *validTrans-isCOMact-open*:
assumes *validTrans trn*
and *isCOMact* (*actOf trn*)
shows *open* (*tgtOf trn*) = *open* (*srcOf trn*)
⟨*proof*⟩

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

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

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

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

lemma *filter-list-all-isPVal-isOVal*:
assumes *list-all* (*Not* ∘ *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* ∘ *isOVal*) *ul* ∧ *filter isPVal ul* = []
⟨*proof*⟩

lemma *filter-isTValS-Nil*:
filter isPValS vl = [] ⟷
list-all (λ *v*. *isPVal v* ∨ *isOVal v*) *vl*
⟨*proof*⟩

fun φ :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
φ (*Trans* - (*Uact* (*uPost uid p pid pst*)) *ou* -) = (*pid* = *PID* ∧ *ou* = *outOK*)
|
φ (*Trans* - (*COMact* (*comSendPost uid p aid pid*)) *ou* -) = (*pid* = *PID* ∧ *ou* ≠ *outErr*)

|
φ (*Trans s* - - *s'*) = (*open s* ≠ *open s'*)

lemma φ-*def1*:
φ *trn* ⟷
(∃ *uid p pst*. *actOf trn* = *Uact* (*uPost uid p PID pst*) ∧ *outOf trn* = *outOK*) ∨
(∃ *uid p aid*. *actOf trn* = *COMact* (*comSendPost uid p aid PID*) ∧ *outOf trn* ≠ *outErr*) ∨
((∀ *uid p pid pst*. *actOf trn* ≠ *Uact* (*uPost uid p pid pst*)) ∧
(∀ *uid p aid pid*. *actOf trn* ≠ *COMact* (*comSendPost uid p aid pid*)) ∧
open (*srcOf trn*) ≠ *open* (*tgtOf trn*))
⟨*proof*⟩

lemma φ-*def2*:
assumes *step s a* = (*ou*,*s'*)
shows
φ (*Trans s a ou s'*) ⟷
(∃ *uid p pst*. *a* = *Uact* (*uPost uid p PID pst*) ∧ *ou* = *outOK*) ∨
(∃ *uid p aid*. *a* = *COMact* (*comSendPost uid p aid PID*) ∧ *ou* ≠ *outErr*) ∨
open s ≠ *open s'*
⟨*proof*⟩

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* ∧ *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* ≠ *outErr*

shows *ou* = *O-sendPost* (*aid*, *clientPass s aid*, *PID*, *post s PID*, *owner s PID*, *vis s PID*)
 ∧ *uid* = *admin s* ∧ *p* = *pass s* (*admin s*)
⟨*proof*⟩

lemma *step-open-isCOMact*:
assumes *step s a* = (*ou*,*s'*)
and *open s* ≠ *open s'*
shows ¬ *isCOMact a* ∧ ¬ (∃ *ua*. *isuPost ua* ∧ *a* = *Uact ua*)
⟨*proof*⟩

lemma *φ-def3*:
assumes *step s a* = (*ou*,*s'*)
shows
φ (*Trans s a ou s'*) ⟷
(∃ *pst*. *a* = *Uact* (*uPost* (*owner s PID*) (*pass s* (*owner s PID*)) *PID pst*) ∧ *ou* = *outOK*)
∨
(∃ *aid*. *a* = *COMact* (*comSendPost* (*admin s*) (*pass s* (*admin s*)) *aid PID*) ∧
 ou = *O-sendPost* (*aid*, *clientPass s aid*, *PID*, *post s PID*, *owner s PID*, *vis s PID*))
∨
open s ≠ *open s'* ∧ ¬ *isCOMact a* ∧ ¬ (∃ *ua*. *isuPost ua* ∧ *a* = *Uact ua*)
⟨*proof*⟩

fun *f* :: (*state*,*act*,*out*) *trans* ⇒ *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* ≠ *open s'* ∧ ¬ *isCOMact a* ∧ ¬ (∃ *ua*. *isuPost ua* ∧ *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** *φ* (*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** *φ* (*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: $\varphi\ (Trans\ s\ a\ ou\ s')$
shows $\varphi\ (Trans\ s1\ a\ ou1\ s1')$
⟨*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 $\varphi\ (Trans\ s\ a\ ou\ s') = \varphi\ (Trans\ s1\ a\ ou1\ s1')$
⟨*proof*⟩

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* ⇒ *bool* **where** T - = *False*

We again use the dynamic declassification bound for the issuer node (Section 6.5.2).

inductive *BC* :: *value list* ⇒ *value list* ⇒ *bool*
and *BO* :: *value list* ⇒ *value list* ⇒ *bool*
where
BC-PVal[*simp*,*intro*!]:
list-all (*Not o isOVal*) *ul* ⟹ *list-all* (*Not o isOVal*) *ul1* ⟹
map tgtAPI (*filter isPValS ul*) = *map tgtAPI* (*filter isPValS ul1*) ⟹
(*ul* = [] ⟶ *ul1* = []) ⟹ *BC ul ul1*
|*BC-BO*[*intro*]:
BO vl vl1 ⟹
list-all (*Not o isOVal*) *ul* ⟹ *list-all* (*Not o isOVal*) *ul1* ⟹
map tgtAPI (*filter isPValS ul*) = *map tgtAPI* (*filter isPValS ul1*) ⟹
(*ul* = [] ⟷ *ul1* = []) ⟹
(*ul* ≠ [] ⟹ *isPVal* (*last ul*) ∧ *last ul* = *last ul1*) ⟹
list-all isPValS sul
⟹
BC (*ul* @ *sul* @ *OVal True* # *vl*)
(*ul1* @ *sul* @ *OVal True* # *vl1*)

|*BO-PVal*[*simp*,*intro*!]:
list-all (*Not o isOVal*) *ul* ⟹ *BO ul ul*
|*BO-BC*[*intro*]:
BC vl vl1 ⟹
list-all (*Not o isOVal*) *ul*
⟹
BO (*ul* @ *OVal False* # *vl*) (*ul* @ *OVal False* # *vl1*)

lemma *list-all-filter-Not-isOVal*:
assumes *list-all* (*Not* ∘ *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 ∃ *vl1'*. *BO vl' vl1'* ∧ *vl1* = *OVal True* # *vl1'*
⟨*proof*⟩

fun *corrFrom* :: *post* ⇒ *value list* ⇒ *bool* **where**
corrFrom pst [] = *True*
|*corrFrom pst* (*PVal pstt* # *vl*) = *corrFrom pstt vl*
|*corrFrom pst* (*PValS aid pstt* # *vl*) = (*pst* = *pstt* ∧ *corrFrom pst vl*)
|*corrFrom pst* (*OVal b* # *vl*) = (*corrFrom pst vl*)

abbreviation *corr* :: *value list* ⇒ *bool* **where** *corr* ≡ *corrFrom emptyPost*

definition *B* **where**
B vl vl1 ≡ *BC vl vl1* ∧ *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*⟩

6.6.4 Issuer unwinding proof

lemma *reach-PublicV-imples-FriendV*[*simp*]:
assumes *reach s*
and *vis s pid* ≠ *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*: ¬ *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'*
⟨*proof*⟩

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.*

$BO\ vll\ vll1\ \wedge$
$list\text{-}all\ (Not\ o\ isOVal)\ ul\ \wedge\ list\text{-}all\ (Not\ o\ isOVal)\ ul1\ \wedge$
$map\ tgtAPI\ (filter\ isPValS\ ul) = map\ tgtAPI\ (filter\ isPValS\ ul1)\ \wedge$
$ul \neq []\ \wedge\ ul1 \neq []\ \wedge$
$isPVal\ (last\ ul)\ \wedge\ last\ ul = last\ ul1\ \wedge$
$list\text{-}all\ isPValS\ sul\ \wedge$
$vl = ul\ @\ sul\ @\ OVal\ True\ \#\ vll\ \wedge\ vl1 = ul1\ @\ sul\ @\ OVal\ True\ \#\ vll1)\ \wedge$
$eqButPID\ s\ s1\ \wedge\ \neg\ open\ s\ \wedge$
$corrFrom\ (post\ s1\ PID)\ vl1$

definition $\Delta 32 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 32\ s\ vl\ s1\ vl1 \equiv$
$PID \in\in postIDs\ s\ \wedge$
$(\exists\ sul\ vll\ vll1.$
$BO\ vll\ vll1\ \wedge$
$list\text{-}all\ isPValS\ sul\ \wedge$
$vl = sul\ @\ OVal\ True\ \#\ vll\ \wedge\ vl1 = sul\ @\ OVal\ True\ \#\ vll1)\ \wedge$
$s = s1\ \wedge\ \neg\ open\ s\ \wedge$
$corrFrom\ (post\ s1\ PID)\ vl1$

definition $\Delta 4 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 4\ s\ vl\ s1\ vl1 \equiv$
$PID \in\in postIDs\ s\ \wedge$
$(\exists\ ul\ vll\ vll1.$
$BC\ vll\ vll1\ \wedge$
$list\text{-}all\ (Not\ o\ isOVal)\ ul\ \wedge$
$vl = ul\ @\ OVal\ False\ \#\ vll\ \wedge\ vl1 = ul\ @\ OVal\ False\ \#\ vll1)\ \wedge$
$s = s1\ \wedge\ open\ s\ \wedge$
$corrFrom\ (post\ s1\ PID)\ vl1$

lemma *istate-*$\Delta 0$:
assumes *B*: $B\ vl\ vl1$
shows $\Delta 0\ istate\ vl\ istate\ vl1$
⟨*proof*⟩

lemma *list-all-filter*[*simp*]:
assumes $list\text{-}all\ PP\ xs$
shows $filter\ PP\ xs = xs$
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 0$: $unwind\text{-}cont\ \Delta 0\ \{\Delta 0,\Delta 1,\Delta 2,\Delta 31,\Delta 32,\Delta 4\}$
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 1$: $unwind\text{-}cont\ \Delta 1\ \{\Delta 1,\Delta 11\}$
⟨*proof*⟩

lemma *unwind-cont-*$\Delta 11$: $unwind\text{-}cont\ \Delta 11\ \{\Delta 11\}$
⟨*proof*⟩

lemma *unwind-cont-Δ31*: *unwind-cont Δ31 {Δ31,Δ32}*
⟨*proof*⟩

lemma *unwind-cont-Δ32*: *unwind-cont Δ32 {Δ2,Δ32,Δ4}*
⟨*proof*⟩

lemma *unwind-cont-Δ2*: *unwind-cont Δ2 {Δ2}*
⟨*proof*⟩

lemma *unwind-cont-Δ4*: *unwind-cont Δ4 {Δ1,Δ31,Δ32,Δ4}*
⟨*proof*⟩

definition *Gr* **where**
Gr =
{
(*Δ0*, {*Δ0*,*Δ1*,*Δ2*,*Δ31*,*Δ32*,*Δ4*}),
(*Δ1*, {*Δ1*,*Δ11*}),
(*Δ11*, {*Δ11*}),
(*Δ2*, {*Δ2*}),
(*Δ31*, {*Δ31*,*Δ32*}),
(*Δ32*, {*Δ2*,*Δ32*,*Δ4*}),
(*Δ4*, {*Δ1*,*Δ31*,*Δ32*,*Δ4*})
}

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* ⇒ *bool* **where**

γ (*Trans - a - -*) $\longleftrightarrow$
(∃ *uid*. *userOfA a* = *Some uid* ∧ *uid* ∈ *UIDs*)
∨
— Communication actions are considered to be observable in order to make the security properties compositional
(∃ *ca*. *a* = *COMact ca*)
∨
— 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).
(∃ *uid p pid pst*. *a* = *Uact* (*uPost uid p pid pst*))
∨
(∃ *uid p*. *a* = *Sact* (*sSys uid p*))
∨
(∃ *uid p uid′ p′*. *a* = *Cact* (*cUser uid p uid′ p′*))
∨
(∃ *uid p pid*. *a* = *Cact* (*cPost uid p pid*))
∨
(∃ *uid p uid′*. *a* = *Cact* (*cFriend uid p uid′*))
∨
(∃ *uid p uid′*. *a* = *Dact* (*dFriend uid p uid′*))
∨
(∃ *uid p pid v*. *a* = *Uact* (*uVisPost uid p pid v*))

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

fun *comPurge* :: *comActt* ⇒ *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* ∧ *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**
$g\ (Trans\ \text{-}\ (Sact\ sa)\ ou\ \text{-}) = (Sact\ (sPurge\ sa),\ ou)$
$|g\ (Trans\ \text{-}\ (COMact\ ca)\ ou\ \text{-}) = (COMact\ (comPurge\ ca),\ ou)$
$|g\ (Trans\ \text{-}\ a\ ou\ \text{-}) = (a,ou)$

lemma *comPurge-simps*:
$comPurge\ ca = comSendServerReq\ uID\ p\ aID\ reqInfo \longleftrightarrow (\exists\, p'.\ ca = comSendServerReq\ uID\ p'\ aID\ reqInfo \wedge p = emptyPass)$
$comPurge\ ca = comReceiveClientReq\ aID\ reqInfo \longleftrightarrow ca = comReceiveClientReq\ aID\ reqInfo$
$comPurge\ ca = comConnectClient\ uID\ p\ aID\ sp \longleftrightarrow (\exists\, p'.\ ca = comConnectClient\ uID\ p'\ aID\ sp \wedge p = emptyPass)$
$comPurge\ ca = comConnectServer\ aID\ sp \longleftrightarrow ca = comConnectServer\ aID\ sp$
$comPurge\ ca = comReceivePost\ aID\ sp\ pID\ pst'\ uID\ v \longleftrightarrow (\exists\, pst.\ ca = comReceivePost\ aID\ sp\ pID\ pst\ uID\ v \wedge pst' = (if\ pID = PID \wedge aID = AID\ then\ emptyPost\ else\ pst))$
$comPurge\ ca = comSendPost\ uID\ p\ aID\ pID \longleftrightarrow (\exists\, p'.\ ca = comSendPost\ uID\ p'\ aID\ pID \wedge p = emptyPass)$
$comPurge\ ca = comSendCreateOFriend\ uID\ p\ aID\ uID' \longleftrightarrow (\exists\, p'.\ ca = comSendCreateOFriend\ uID\ p'\ aID\ uID' \wedge p = emptyPass)$
$comPurge\ ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow ca = comReceiveCreateOFriend\ aID\ cp\ uID\ uID'$
$comPurge\ ca = comSendDeleteOFriend\ uID\ p\ aID\ uID' \longleftrightarrow (\exists\, p'.\ ca = comSendDeleteOFriend\ uID\ p'\ aID\ uID' \wedge p = emptyPass)$
$comPurge\ ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID' \longleftrightarrow ca = comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'$
⟨*proof*⟩

lemma *g-simps*:
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendServerReq\ uID\ p\ aID\ reqInfo),\ ou')$
$\longleftrightarrow (\exists\, p'.\ a = COMact\ (comSendServerReq\ uID\ p'\ aID\ reqInfo) \wedge p = emptyPass \wedge ou = ou')$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveClientReq\ aID\ reqInfo),\ ou')$
$\longleftrightarrow a = COMact\ (comReceiveClientReq\ aID\ reqInfo) \wedge ou = ou'$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectClient\ uID\ p\ aID\ sp),\ ou')$
$\longleftrightarrow (\exists\, p'.\ a = COMact\ (comConnectClient\ uID\ p'\ aID\ sp) \wedge p = emptyPass \wedge ou = ou')$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comConnectServer\ aID\ sp),\ ou')$
$\longleftrightarrow a = COMact\ (comConnectServer\ aID\ sp) \wedge ou = ou'$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceivePost\ aID\ sp\ pID\ pst'\ uID\ v),\ ou')$
$\longleftrightarrow (\exists\, pst.\ a = COMact\ (comReceivePost\ aID\ sp\ pID\ pst\ uID\ v) \wedge pst' = (if\ pID = PID \wedge aID = AID\ then\ emptyPost\ else\ pst) \wedge ou = ou')$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendPost\ uID\ p\ aID\ nID),\ O\text{-}sendPost\ (aid,\ sp,\ pid,\ pst,\ own,\ v))$
$\longleftrightarrow (\exists\, p'.\ a = COMact\ (comSendPost\ uID\ p'\ aID\ nID) \wedge p = emptyPass \wedge ou = O\text{-}sendPost\ (aid,\ sp,\ pid,\ pst,\ own,\ v))$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendCreateOFriend\ uID\ p\ aID\ uID'),\ ou')$

$\longleftrightarrow (\exists\, p'.\ a = (COMact\ (comSendCreateOFriend\ uID\ p'\ aID\ uID'))\ \wedge\ p = emptyPass\ \wedge\ ou = ou')$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID'), ou')$
$\longleftrightarrow a = COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID')\ \wedge\ ou = ou'$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendDeleteOFriend\ uID\ p\ aID\ uID'), ou')$
$\longleftrightarrow (\exists\, p'.\ a = COMact\ (comSendDeleteOFriend\ uID\ p'\ aID\ uID')\ \wedge\ p = emptyPass\ \wedge\ ou = ou')$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'), ou')$
$\longleftrightarrow a = COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID')\ \wedge\ ou = ou'$
⟨*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 φ :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
$\varphi\ (Trans\ \text{-}\ (COMact\ (comReceivePost\ aid\ sp\ pid\ pst\ uid\ vs))\ ou\ \text{-}) =$
$(aid = AID\ \wedge\ pid = PID\ \wedge\ ou = outOK)$
|
$\varphi\ (Trans\ s\ \text{-}\ \text{-}\ s') = False$

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*: *a* = *COMact* (*comReceivePost AID p PID pst uid vs*) **and** *2*: *ou* = *outOK*
shows *p* = *serverPass s AID*
⟨*proof*⟩

lemma *φ-def3*:
assumes *step s a* = (*ou*,*s'*)
shows
φ (*Trans s a ou s'*) ⟷
(∃ *uid pst vs*. *a* = *COMact* (*comReceivePost AID* (*serverPass s AID*) *PID pst uid vs*) ∧ *ou* = *outOK*)
⟨*proof*⟩

lemma *φ-cases*:
assumes *φ* (*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*
⟨*proof*⟩

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

sublocale *Receiver-State-Equivalence-Up-To-PID* ⟨*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 *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* ⇒ *bool* **where**
T (*Trans s a ou s'*) ⟷
(∃ *uid* ∈ *UIDs*.
uid ∈∈ *userIDs s'* ∧ *PID* ∈∈ *outerPostIDs s' AID* ∧
(*uid* = *admin s'* ∨
(*AID*,*outerOwner s' AID PID*) ∈∈ *recvOuterFriendIDs s' uid* ∨
outerVis s' AID PID = *PublicV*))

definition B :: *value list* ⇒ *value list* ⇒ *bool* **where**
B *vl vl1* ≡ *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** $B = B$
⟨*proof*⟩

6.6.8 Receiver unwinding proof

lemma *reach-PublicV-imples-FriendV*[*simp*]:
assumes *reach s*
and *vis s pID* ≠ *PublicV*
shows *vis s pID* = *FriendV*
⟨*proof*⟩

lemma *reachNT-state*:
assumes *reachNT s*
shows
¬ (∃ *uid* ∈ *UIDs*.
uid ∈∈ *userIDs s* ∧ *PID* ∈∈ *outerPostIDs s AID* ∧
(*uid* = *admin s* ∨
(*AID*,*outerOwner s AID PID*) ∈∈ *recvOuterFriendIDs s uid* ∨
outerVis s AID PID = *PublicV*))
⟨*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 *sT*: *reachNT s* **and** *T*: ¬ *T* (*Trans s a ou s′*)
and *s1*: *reach s1*
and γ: γ (*Trans s a ou s′*)
shows *ou* = *ou1*
⟨*proof*⟩

definition Δ*0* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*0 s vl s1 vl1* ≡
¬ *AID* ∈∈ *serverApiIDs s* ∧
s = *s1* ∧
length vl = *length vl1*

definition Δ*1* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*1 s vl s1 vl1* ≡
AID ∈∈ *serverApiIDs s* ∧
eqButPID s s1 ∧
length vl = *length vl1*

lemma *istate-Δ0*:
assumes *B*: *B vl vl1*
shows Δ*0 istate vl istate vl1*
⟨*proof*⟩

lemma *unwind-cont-Δ0*: *unwind-cont* Δ*0* {Δ*0*,Δ*1*}
⟨*proof*⟩

lemma *unwind-cont-Δ1*: *unwind-cont* Δ*1* {Δ*1*}
⟨*proof*⟩

definition *Gr* **where**
Gr =
{
(Δ*0*, {Δ*0*,Δ*1*}),
(Δ*1*, {Δ*1*})
}

theorem *Post-secure*: *secure*
⟨*proof*⟩

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 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 Strong-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 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 $B :: apiID \Rightarrow value\ list \Rightarrow value\ list \Rightarrow bool$
where $B\ aid\ vl\ vl1 \equiv$
$(if\ aid = AID\ then\ list\text{-}all\ isl\ vl \wedge list\text{-}all\ isl\ vl1 \wedge Iss.B\ (map\ projl\ vl)\ (map\ projl\ vl1)$
$else\ list\text{-}all\ (Not\ o\ isl)\ vl \wedge list\text{-}all\ (Not\ o\ isl)\ vl1 \wedge Post\text{-}RECEIVER.B\ (map\ projr\ vl)\ (map\ projr\ vl1))$

fun $comOfV :: apiID \Rightarrow value \Rightarrow com$ **where**
$comOfV\ aid\ (Inl\ (Post.PValS\ aid'\ pst)) = (if\ aid' \neq aid\ then\ Send\ else\ Internal)$
$| comOfV\ aid\ (Inl\ (Post.PVal\ pst)) = Internal$
$| comOfV\ aid\ (Inl\ (Post.OVal\ ov)) = Internal$
$| comOfV\ aid\ (Inr\ v) = Recv$

fun $tgtNodeOfV :: apiID \Rightarrow value \Rightarrow apiID$ **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 $syncV :: apiID \Rightarrow value \Rightarrow apiID \Rightarrow value \Rightarrow bool$ **where**
$syncV\ aid1\ v1\ aid2\ v2 =$
$(\exists pst.\ aid1 = AID \wedge v1 = Inl\ (Post.PValS\ aid2\ pst) \wedge v2 = Inr\ (Post\text{-}RECEIVER.PValR\ pst))$

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

lemma *syncVE*:
assumes $syncV\ aid1\ v1\ aid2\ v2$
obtains pst **where** $aid1 = AID\ v1 = Inl\ (Post.PValS\ aid2\ pst)\ v2 = Inr\ (Post\text{-}RECEIVER.PValR\ pst)$
⟨*proof*⟩

fun $getTgtV$ **where**
$getTgtV\ (Inl\ (Post.PValS\ aid\ pst)) = Inr\ (Post\text{-}RECEIVER.PValR\ pst)$
$| getTgtV\ v = v$

lemma *comOfV-AID*:
$comOfV\ AID\ v = Send \longleftrightarrow isl\ v \wedge Iss.isPValS\ (projl\ v) \wedge Iss.tgtAPI\ (projl\ v) \neq AID$
$comOfV\ AID\ v = Recv \longleftrightarrow Not\ (isl\ v)$
⟨*proof*⟩

lemmas $\varphi\text{-}defs = Post\text{-}RECEIVER.\varphi\text{-}def2\ Iss.\varphi\text{-}def3$

sublocale *Net*: *BD-Security-TS-Network-getTgtV*
where $istate = \lambda\text{-}.\ istate$ **and** $validTrans = validTrans$ **and** $srcOf = \lambda\text{-}.\ srcOf$ **and** $tgtOf = \lambda\text{-}.\ tgtOf$

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*
⟨*proof*⟩

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

context
fixes *AID'* :: *apiID*
assumes *AID'*: *AID'* ∈ *AIDs* − {*AID*}
begin

interpretation *Recv*: *Post-RECEIVER UIDs AID' PID AID* ⟨*proof*⟩

lemma *Iss-BC-BO-tgtAPI*:
shows (*Iss.BC vl vl1* ⟶ *map Iss.tgtAPI* (*filter Iss.isPValS vl*) =
map Iss.tgtAPI (*filter Iss.isPValS vl1*)) ∧
(*Iss.BO vl vl1* ⟶ *map Iss.tgtAPI* (*filter Iss.isPValS vl*) =
map Iss.tgtAPI (*filter Iss.isPValS vl1*))
⟨*proof*⟩

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*))
⟨*proof*⟩

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*))
⟨*proof*⟩

end

lemma *map-projl-Inl*: *map* (*projl o Inl*) *vl* = *vl*
⟨*proof*⟩

lemma *these-map-Inl-projl*: *list-all isl vl* ⟹ *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* ⟹ *these* (*map* (*Some o Inr o projr*) *vl*) = *vl*
⟨*proof*⟩

sublocale *BD-Security-TS-Network-Preserve-Source-Security-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*
⟨*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* :: *state* ⇒ *obs* ⇒ *state* **where**
obsEffect s (*Uact* (*uPost uid p pid pst*), *ou*) = *updatePost s uid p pid pst*
| *obsEffect s* (*Uact* (*uVisPost uid p pid v*), *ou*) = *updateVisPost s uid p pid v*
| *obsEffect s* (*Sact* (*sSys uid p*), *ou*) = *startSys s uid p*
| *obsEffect s* (*Cact* (*cUser uid p uid′ p′*), *ou*) = *createUser s uid p uid′ p′*

| *obsEffect s* (*Cact* (*cPost uid p pid*), *ou*) = *createPost s uid p pid*
| *obsEffect s* (*Cact* (*cFriend uid p uid'*), *ou*) = *createFriend s uid p uid'*
| *obsEffect s* (*Dact* (*dFriend uid p uid'*), *ou*) = *deleteFriend s uid p uid'*
| *obsEffect s* (*COMact* (*comSendPost uid p aid pid*), *ou*) = *snd* (*sendPost s uid p aid pid*)
| *obsEffect s* (*COMact* (*comReceivePost aid p pid pst uid v*), *ou*) = *receivePost s aid p pid pst uid v*
| *obsEffect s* (*COMact* (*comReceiveCreateOFriend aid p uid uid'*), *ou*) = *receiveCreateOFriend s aid p uid uid'*
| *obsEffect s* (*COMact* (*comReceiveDeleteOFriend aid p uid uid'*), *ou*) = *receiveDeleteOFriend s aid p uid uid'*
| *obsEffect s - = s*

fun *obsStep* :: *state* ⇒ *obs* ⇒ *state* **where**
obsStep s (*a*,*ou*) = (*if ou* ≠ *outErr then obsEffect s* (*a*,*ou*) *else s*)

fun *obsSteps* :: *state* ⇒ *obs list* ⇒ *state* **where**
obsSteps s obsl = *foldl obsStep s obsl*

definition *triggerEq* :: *state* ⇒ *state* ⇒ *bool* **where**
triggerEq s s' ⟷ *userIDs s* = *userIDs s'* ∧ *postIDs s* = *postIDs s'* ∧ *admin s* = *admin s'* ∧
owner s = *owner s'* ∧ *friendIDs s* = *friendIDs s'* ∧ *vis s* = *vis s'* ∧
outerPostIDs s = *outerPostIDs s'* ∧ *outerOwner s* = *outerOwner s'* ∧
recvOuterFriendIDs s = *recvOuterFriendIDs s'* ∧ *outerVis s* = *outerVis s'*

lemma *triggerEq-refl*[*simp*]: *triggerEq s s*
and *triggerEq-sym*: *triggerEq s s'* ⟹ *triggerEq s' s*
and *triggerEq-trans*: *triggerEq s s'* ⟹ *triggerEq s' s''* ⟹ *triggerEq s s''*
⟨*proof*⟩

unbundle *no relcomp-syntax*

context *Post*
begin

lemma [*simp*]: *outOK* = *outPurge ou* ⟷ *ou* = *outOK* ⟨*proof*⟩
lemma [*simp*]: *sPurge sa* = *sSys* (*sUserOfA sa*) *emptyPass* ⟨*proof*⟩
lemma *sStep-unfold*: *sStep s sa* = (*if userIDs s* = []
then (*case sa of sSys uid p* ⇒ (*outOK*, *startSys s uid p*))
else (*outErr*, *s*))
⟨*proof*⟩

lemma *triggerEq-open*:
assumes *triggerEq s s'*
shows *open s* ⟷ *open s'*

⟨*proof*⟩

lemma *triggerEq-not-γ*:
assumes *validTrans* (*Trans s a ou s'*) **and** ¬*γ* (*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'* ≡
(∃ *uid* ∈ *UIDs*.
  uid ∈∈ *userIDs s'* ∧ *PID* ∈∈ *outerPostIDs s' AID* ∧
  (*uid* = *admin s'* ∨
   (*AID*,*outerOwner s' AID PID*) ∈∈ *recvOuterFriendIDs s' uid* ∨
   outerVis s' AID PID = *PublicV*))

lemma *T-T-state*: *T trn* ⟷ *T-state* (*tgtOf trn*)
⟨*proof*⟩

lemma *triggerEq-T*:
assumes *triggerEq s s'*
shows *T-state s* ⟷ *T-state s'*
⟨*proof*⟩

lemma *never-T-not-T-state*:
assumes *validFrom s tr* **and** *never T tr* **and** ¬*T-state s*
shows ¬*T-state* (*tgtOfTrFrom s tr*)
⟨*proof*⟩

lemma *triggerEq-not-γ*:
assumes *validTrans* (*Trans s a ou s'*) **and** ¬*γ* (*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-Network*
begin

fun *nObsStep* :: (*apiID* ⇒ *state*) ⇒ (*apiID*, *act* × *out*) *nobs* ⇒ (*apiID* ⇒ *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* ⇒ *state*) ⇒ (*apiID*, *act* × *out*) *nobs list* ⇒ (*apiID* ⇒ *state*) **where**
nObsSteps s obsl = *foldl nObsStep s obsl*

definition *nTriggerEq* :: (*apiID* ⇒ *state*) ⇒ (*apiID* ⇒ *state*) ⇒ *bool* **where**
nTriggerEq s s' ⟷ (∀ *aid*. *triggerEq* (*s aid*) (*s' aid*))

lemma *nTriggerEq-refl*[*simp*]: *nTriggerEq s s*
and *nTriggerEq-sym*: *nTriggerEq s s'* ⟹ *nTriggerEq s' s*
and *nTriggerEq-trans*: *nTriggerEq s s'* ⟹ *nTriggerEq s' s''* ⟹ *nTriggerEq s s''*
⟨*proof*⟩

lemma *nTriggerEq-open*:
assumes *nTriggerEq s s'*
shows ∀ *aid*. *Iss.open* (*s aid*) ⟷ *Iss.open* (*s' aid*)
⟨*proof*⟩

lemma *nTriggerEq-not-*γ:
assumes *nValidTrans trn* **and** ¬*Net.n*γ *trn*
shows *nTriggerEq* (*nSrcOf trn*) (*nTgtOf trn*)
⟨*proof*⟩

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*))
⟨*proof*⟩

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 *γ*: *Net.nγ trn* **and** *γ'*: *Net.nγ trn'* **and** *g*: *Net.ng trn* = *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* ⇒ *userID set*
and *AID1* :: *apiID* **and** *AID2* :: *apiID*
and *PID1* :: *postID* **and** *PID2* :: *postID*
+
assumes *AID1-neq-AID2*: *AID1* ≠ *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 *nγ* :: (*apiID*, *state*, (*state*, *act*, *out*) *trans*) *ntrans* ⇒ *bool* **where**
nγ (*LTrans s aid* (*Trans - a - -*)) = (∃ *uid*. *userOfA a* = *Some uid* ∧ *uid* ∈ *UIDs aid* ∧ (¬*isCOMact a*))
| *nγ* (*CTrans s aid1 trn1 aid2 trn2*) = *False*

fun *g* :: (*state*,*act*,*out*) *trans* ⇒ *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* ⇒ (*apiID*, *act* × *out*) *nobs* **where**
ng (*LTrans s aid trn*) = *LObs aid* (*g trn*)
| *ng* (*CTrans s aid1 trn1 aid2 trn2*) = *undefined*

abbreviation *validSystemTrace* ≡ *Post1.validFrom* (λ-. *istate*)

We now instantiate the generic technique for combining security properties with independent secret sources.

sublocale *BD-Security-TS-Two-Secrets* λ*-. istate Post1.nValidTrans Post1.nSrcOf Post1.nTgtOf*
Post1.Net.nφ *Post1.nf' Post1.Net.n*γ *Post1.Net.ng Post1.Net.nT Post1.B AID1*
Post2.Net.nφ *Post2.nf' Post2.Net.n*γ *Post2.Net.ng Post2.Net.nT Post2.B AID2*
nγ *ng*
⟨*proof*⟩

theorem *two-posts-secure*:
secure
⟨*proof*⟩

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 γ :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
γ (*Trans* - *a* - -) = (∃ *uid*. *userOfA a* = *Some uid* ∧ *uid* ∈ *UIDs* ∨ (∃ *ca*. *a* = *COMact ca*))

fun *g* :: (*state*,*act*,*out*)*trans* ⇒ *obs* **where**
g (*Trans* - *a ou* -) = (*a*,*ou*)

end

locale *FriendNetworkObservationSetup* =
fixes *UIDs* :: *apiID* ⇒ *userID set* — groups of observers at different nodes
begin

abbreviation γ :: *apiID* ⇒ (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
γ *aid trn* ≡ *FriendObservationSetup*.γ (*UIDs aid*) *trn*

abbreviation *g* :: *apiID* ⇒ (*state*,*act*,*out*)*trans* ⇒ *obs* **where**

g aid trn ≡ *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*} ∩ *UIDs* = {}
and
UID1-UID2: *UID1* ≠ *UID2*
begin

fun *eqButUIDl* :: *userID* ⇒ *userID list* ⇒ *userID list* ⇒ *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′* ∈∈ *uidl* ⟷ *uid′* ∈∈ *uidl1*
shows *eqButUIDl uid* (*uidl* ## *uid′*) (*uidl1* ## *uid′*)

⟨*proof*⟩

definition *eqButUIDf* **where**
eqButUIDf frds frds1 ≡
  eqButUIDl UID2 (*frds UID1*) (*frds1 UID1*)
∧ *eqButUIDl UID1* (*frds UID2*) (*frds1 UID2*)
∧ (∀ *uid*. *uid* ≠ *UID1* ∧ *uid* ≠ *UID2* ⟶ *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* ⟹ *eqButUIDl UID2 uu uu1*
and *uid* = *UID2* ⟹ *eqButUIDl UID1 uu uu1*
and *uid* ≠ *UID1* ⟹ *uid* ≠ *UID2* ⟹ *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*:
⟦*eqButUIDf frds frds1*; *uid* ≠ *UID1*; *uid* ≠ *UID2*⟧ ⟹ *frds uid* = *frds1 uid*
⟨*proof*⟩

lemma *eqButUIDf-not-UID′*:
assumes *eq1*: *eqButUIDf frds frds1*
and *uid*: (*uid*,*uid′*) ∉ {(*UID1*,*UID2*), (*UID2*,*UID1*)}
shows *uid* ∈∈ *frds uid′* ⟷ *uid* ∈∈ *frds1 uid′*
⟨*proof*⟩

definition *eqButUID12* **where**

eqButUID12 freq freq1 ≡
∀ *uid uid′*. *if* (*uid*,*uid′*) ∈ {(*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*
⟨*proof*⟩

lemma *eqButUID12-sym*:
assumes *eqButUID12 freq freq1* **shows** *eqButUID12 freq1 freq*
⟨*proof*⟩

lemma *eqButUID12-trans*:
assumes *eqButUID12 freq freq1* **and** *eqButUID12 freq1 freq2*
shows *eqButUID12 freq freq2*
⟨*proof*⟩

lemma *eqButUID12-cong*:
assumes *eqButUID12 freq freq1*

and ¬ (*uid*,*uid′*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)} ⟹ *uu* = *uu1*
shows *eqButUID12* (*fun-upd2 freq uid uid′ uu*) (*fun-upd2 freq1 uid uid′ uu1*)
⟨*proof*⟩

lemma *eqButUID12-not-UID*:
⟦*eqButUID12 freq freq1*; ¬ (*uid*,*uid′*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)}⟧ ⟹ *freq uid uid′* = *freq1 uid 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* ∧

eqButUIDf (*pendingFReqs s*) (*pendingFReqs s1*) ∧
eqButUID12 (*friendReq s*) (*friendReq s1*) ∧
eqButUIDf (*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 ∧
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
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 = outerVis 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
⟨*proof*⟩

lemma *eqButUID-eqButUID2*:
eqButUID s s1 ⟹ eqButUIDl UID2 (friendIDs s UID1) (friendIDs s1 UID1)
⟨*proof*⟩

lemma *eqButUID-not-UID*:
eqButUID s s1 ⟹ uid ≠ UID ⟹ post s uid = post s1 uid
⟨*proof*⟩

lemma *eqButUID-cong*[*simp*, *intro*]:
⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|admin := uu1|)) (s1 (|admin := uu2|))*

⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|pendingUReqs := uu1|)) (s1 (|pendingUReqs := uu2|))*
⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|userReq := uu1|)) (s1 (|userReq := uu2|))*
⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|userIDs := uu1|)) (s1 (|userIDs := uu2|))*
⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|user := uu1|)) (s1 (|user := uu2|))*
⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|pass := uu1|)) (s1 (|pass := uu2|))*

⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|postIDs := uu1|)) (s1 (|postIDs := uu2|))*
⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|owner := uu1|)) (s1 (|owner := uu2|))*
⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|post := uu1|)) (s1 (|post := uu2|))*
⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|vis := uu1|)) (s1 (|vis := uu2|))*

⋀ *uu1 uu2*. *eqButUID s s1 ⟹ eqButUIDf uu1 uu2 ⟹ eqButUID (s (|pendingFReqs := uu1|)) (s1 (|pendingFReqs := uu2|))*
⋀ *uu1 uu2*. *eqButUID s s1 ⟹ eqButUID12 uu1 uu2 ⟹ eqButUID (s (|friendReq := uu1|)) (s1 (|friendReq := uu2|))*
⋀ *uu1 uu2*. *eqButUID s s1 ⟹ eqButUIDf uu1 uu2 ⟹ eqButUID (s (|friendIDs := uu1|)) (s1 (|friendIDs := uu2|))*

⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|pendingSApiReqs := uu1|)) (s1 (|pendingSApiReqs := uu2|))*
⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|sApiReq := uu1|)) (s1 (|sApiReq := uu2|))*
⋀ *uu1 uu2*. *eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|serverApiIDs :=*

uu1⦈) (*s1* (⦇*serverApiIDs* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* (⦇*serverPass* := *uu1*⦈)
(*s1* (⦇*serverPass* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* (⦇*outerPostIDs* :=
uu1⦈) (*s1* (⦇*outerPostIDs* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* (⦇*outerPost* := *uu1*⦈)
(*s1* (⦇*outerPost* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* (⦇*outerVis* := *uu1*⦈)
(*s1* (⦇*outerVis* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* (⦇*outerOwner* :=
uu1⦈) (*s1* (⦇*outerOwner* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* (⦇*sentOuterFriendIDs*
:= *uu1*⦈) (*s1* (⦇*sentOuterFriendIDs* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* (⦇*recvOuterFriendIDs*
:= *uu1*⦈) (*s1* (⦇*recvOuterFriendIDs* := *uu2*⦈)

⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* (⦇*pendingCApiReqs*
:= *uu1*⦈) (*s1* (⦇*pendingCApiReqs* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* (⦇*cApiReq* := *uu1*⦈)
(*s1* (⦇*cApiReq* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* (⦇*clientApiIDs* :=
uu1⦈) (*s1* (⦇*clientApiIDs* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* (⦇*clientPass* := *uu1*⦈)
(*s1* (⦇*clientPass* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* (⦇*sharedWith* :=
uu1⦈) (*s1* (⦇*sharedWith*:= *uu2*⦈)
⟨*proof*⟩

definition *friends12* :: *state* ⇒ *bool*
where *friends12 s* ≡ *UID1* ∈∈ *friendIDs s UID2* ∧ *UID2* ∈∈ *friendIDs s UID1*

lemma *step-friendIDs*:
assumes *step s a* = (*ou*, *s'*)
and *a* ≠ *Cact* (*cFriend uid* (*pass s uid*) *uid'*) ∧ *a* ≠ *Cact* (*cFriend uid'* (*pass s*
uid') *uid*) ∧
    a ≠ *Dact* (*dFriend uid* (*pass s uid*) *uid'*) ∧ *a* ≠ *Dact* (*dFriend uid'* (*pass s*
uid') *uid*)
shows *uid* ∈∈ *friendIDs s' uid'* ⟷ *uid* ∈∈ *friendIDs s uid'* (**is** *?uid*)
  and *uid'* ∈∈ *friendIDs s' uid* ⟷ *uid'* ∈∈ *friendIDs s uid* (**is** *?uid'*)
⟨*proof*⟩

lemma *step-friends12*:
assumes *step s a* = (*ou*, *s'*)
and *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*)
shows *friends12 s'* ⟷ *friends12 s*
⟨*proof*⟩

lemma *step-pendingFReqs*:
assumes *step*: *step s a* = (*ou*, *s'*)
and ∀ *req*. *a* ≠ *Cact* (*cFriend uid* (*pass s uid*) *uid'*) ∧ *a* ≠ *Cact* (*cFriend uid'* (*pass s uid'*) *uid*) ∧
a ≠ *Dact* (*dFriend uid* (*pass s uid*) *uid'*) ∧ *a* ≠ *Dact* (*dFriend uid'* (*pass s uid'*) *uid*) ∧
a ≠ *Cact* (*cFriendReq uid* (*pass s uid*) *uid' req*) ∧
a ≠ *Cact* (*cFriendReq uid'* (*pass s uid'*) *uid req*)
shows *uid* ∈∈ *pendingFReqs s' uid'* ⟷ *uid* ∈∈ *pendingFReqs s uid'* (**is** *?uid*)
and *uid'* ∈∈ *pendingFReqs s' uid* ⟷ *uid'* ∈∈ *pendingFReqs s uid* (**is** *?uid'*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *distinct-remove1-idem*: *distinct xs* ⟹ *remove1 y* (*remove1 y xs*) = *remove1 y xs*
⟨*proof*⟩

lemma *Cact-cFriend-step-eqButUID*:
assumes *step*: *step s* (*Cact* (*cFriend uid p uid'*)) = (*ou*,*s'*)
and *s*: *reach s*
and *uids*: (*uid* = *UID1* ∧ *uid'* = *UID2*) ∨ (*uid* = *UID2* ∧ *uid'* = *UID1*) (**is** *?u12* ∨ *?u21*)
shows *eqButUID s s'*
⟨*proof*⟩

lemma *Cact-cFriendReq-step-eqButUID*:
assumes *step*: *step s* (*Cact* (*cFriendReq uid p uid' req*)) = (*ou*,*s'*)
and *uids*: (*uid* = *UID1* ∧ *uid'* = *UID2*) ∨ (*uid* = *UID2* ∧ *uid'* = *UID1*) (**is** *?u12* ∨ *?u21*)
shows *eqButUID s s'*
⟨*proof*⟩

lemma *Dact-dFriend-step-eqButUID*:
assumes *step*: *step s* (*Dact* (*dFriend uid p uid'*)) = (*ou*,*s'*)
and *s*: *reach s*
and *uids*: (*uid* = *UID1* ∧ *uid'* = *UID2*) ∨ (*uid* = *UID2* ∧ *uid'* = *UID1*) (**is** *?u12* ∨ *?u21*)
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-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*⟩

end

end

theory *Friend-Openness*
imports *Friend-State-Indistinguishability*
begin

7.3 Dynamic declassification trigger

context *Friend*
begin

The dynamic declassification trigger condition holds, i.e. the access window to the confidential information is open, as long as the two users have not been created yet (so there cannot be friendship between them) or while one of them is a local friend of an observer.

definition *openByA* :: *state* ⇒ *bool*
where *openByA s* ≡ ¬ *UID1* ∈∈ *userIDs s* ∨ ¬ *UID2* ∈∈ *userIDs s*

definition *openByF* :: *state* ⇒ *bool*
where *openByF s* ≡ ∃ *uid* ∈ *UIDs*. *uid* ∈∈ *friendIDs s UID1* ∨ *uid* ∈∈ *friendIDs s UID2*

definition *open* :: *state* ⇒ *bool*
where *open s* ≡ *openByA s* ∨ *openByF s*

lemmas *open-defs* = *open-def openByA-def openByF-def*

lemma *step-openByA-cases*:
assumes *step s a* = (*ou*, *s*′)
and *openByA s* ≠ *openByA s*′
obtains (*CloseA*) *uid p uid*′ *p*′ **where** *a* = *Cact* (*cUser uid p uid*′ *p*′)
uid′ = *UID1* ∨ *uid*′ = *UID2 ou* = *outOK p* = *pass s uid*
openByA s ¬*openByA s*′
⟨*proof*⟩

lemma *step-openByF-cases*:
assumes *step s a* = (*ou*, *s*′)
and *openByF s* ≠ *openByF s*′
obtains
(*OpenF*) *uid p uid*′ **where** *a* = *Cact* (*cFriend uid p uid*′) *ou* = *outOK p* = *pass s uid*
uid ∈ *UIDs* ∧ *uid*′ ∈ {*UID1*,*UID2*} ∨ *uid* ∈ {*UID1*,*UID2*} ∧ *uid*′ ∈ *UIDs*
openByF s′ ¬*openByF s*
| (*CloseF*) *uid p uid*′ **where** *a* = *Dact* (*dFriend uid p uid*′) *ou* = *outOK p* = *pass*

s uid
uid ∈ *UIDs* ∧ *uid′* ∈ {*UID1*,*UID2*} ∨ *uid* ∈ {*UID1*,*UID2*}
∧ *uid′* ∈ *UIDs*
openByF s ¬*openByF s′*
⟨*proof*⟩

lemma *step-open-cases*:
assumes *step*: *step s a* = (*ou*, *s′*)
and *op*: *open s* ≠ *open s′*
obtains
(*CloseA*) *uid p uid′ p′* **where** *a* = *Cact* (*cUser uid p uid′ p′*)
uid′ = *UID1* ∨ *uid′* = *UID2 ou* = *outOK p* = *pass s uid*
openByA s ¬*openByA s′* ¬*openByF s* ¬*openByF s′*
| (*OpenF*) *uid p uid′* **where** *a* = *Cact* (*cFriend uid p uid′*) *ou* = *outOK p* = *pass*
s uid
uid ∈ *UIDs* ∧ *uid′* ∈ {*UID1*,*UID2*} ∨ *uid* ∈ {*UID1*,*UID2*}
∧ *uid′* ∈ *UIDs*
openByF s′ ¬*openByF s* ¬*openByA s* ¬*openByA s′*
| (*CloseF*) *uid p uid′* **where** *a* = *Dact* (*dFriend uid p uid′*) *ou* = *outOK p* = *pass*
s uid
uid ∈ *UIDs* ∧ *uid′* ∈ {*UID1*,*UID2*} ∨ *uid* ∈ {*UID1*,*UID2*}
∧ *uid′* ∈ *UIDs*
openByF s ¬*openByF s′* ¬*openByA s* ¬*openByA s′*
⟨*proof*⟩

lemma *eqButUID-openByA-eq*:
assumes *eqButUID s s1*
shows *openByA s* = *openByA s1*
⟨*proof*⟩

lemma *eqButUID-openByF-eq*:
assumes *ss1*: *eqButUID s s1*
shows *openByF s* = *openByF s1*
⟨*proof*⟩

lemma *eqButUID-open-eq*: *eqButUID s s1* ⟹ *open s* = *open 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 γ: γ (*Trans s a ou s′*)
and *os*: *open s* ⟶ *friendIDs s* = *friendIDs s1*
shows *ou* = *ou1*
⟨*proof*⟩

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 φ :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
φ (*Trans s* (*Cact* (*cFriend uid p uid'*)) *ou s'*) =
((*uid*,*uid'*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)} ∧ *ou* = *outOK* ∨
open s ≠ *open s'*)
|
φ (*Trans s* (*Dact* (*dFriend uid p uid'*)) *ou s'*) =
((*uid*,*uid'*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)} ∧ *ou* = *outOK* ∨
open s ≠ *open s'*)
|
φ (*Trans s* (*Cact* (*cUser uid p uid' p'*)) *ou s'*) =
(*open s* ≠ *open s'*)
|
φ - = *False*

fun *f* :: (*state*,*act*,*out*) *trans* ⇒ *value* **where**
f (*Trans s* (*Cact* (*cFriend uid p uid'*)) *ou s'*) =
(*if* (*uid*,*uid'*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)} *then FrVal True*
else OVal True)
|
f (*Trans s* (*Dact* (*dFriend uid p uid'*)) *ou s'*) =
(*if* (*uid*,*uid'*) ∈ {(*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 φ: φ (*Trans s a ou s'*) (**is** φ *?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* ∧ *uid′* = *UID2* ∨ *uid* = *UID2* ∧ *uid′* = *UID1*
IDsOK s [*UID1*, *UID2*] [] [] []
¬*friends12 s friends12 s′*
| (*Unfriend*) *uid p uid′* **where** *a* = *Dact* (*dFriend uid p uid′*) *ou* = *outOK f ?trn* = *FrVal False*
uid = *UID1* ∧ *uid′* = *UID2* ∨ *uid* = *UID2* ∧ *uid′* = *UID1*
IDsOK s [*UID1*, *UID2*] [] [] []
friends12 s ¬*friends12 s′*
| (*OpenF*) *uid p uid′* **where** *a* = *Cact* (*cFriend uid p uid′*)
(*uid* ∈ *UIDs* ∧ *uid′* ∈ {*UID1*,*UID2*}) ∨ (*uid′* ∈ *UIDs* ∧ *uid* ∈ {*UID1*,*UID2*})
ou = *outOK f ?trn* = *OVal True* ¬*openByF s openByF s′*
¬*openByA s* ¬*openByA s′*
| (*CloseF*) *uid p uid′* **where** *a* = *Dact* (*dFriend uid p uid′*)
(*uid* ∈ *UIDs* ∧ *uid′* ∈ {*UID1*,*UID2*}) ∨ (*uid′* ∈ *UIDs* ∧ *uid* ∈ {*UID1*,*UID2*})
ou = *outOK f ?trn* = *OVal False openByF s* ¬*openByF s′*
¬*openByA s* ¬*openByA s′*
| (*CloseA*) *uid p uid′ p′* **where** *a* = *Cact* (*cUser uid p uid′ p′*)
uid′ ∈ {*UID1*,*UID2*} *openByA s* ¬*openByA s′*
¬*openByF s* ¬*openByF s′*
ou = *outOK f ?trn* = *OVal False*
⟨*proof*⟩

lemma *step-open-φ*:
assumes *step s a* = (*ou*, *s′*)
and *open s* ≠ *open s′*
shows *φ* (*Trans s a ou s′*)
⟨*proof*⟩

lemma *step-friends12-φ*:
assumes *step s a* = (*ou*, *s′*)
and *friends12 s* ≠ *friends12 s′*
shows *φ* (*Trans s a ou 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*: *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 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*: *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*)
shows φ (*Trans s a ou s'*) = φ (*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* ⇒ *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* ⇒ *bool* ⇒ *bool* **where**
alternatingFriends [] - = *True*
| *alternatingFriends* (*FrVal st* # *vl*) *st'* ⟷ *st'* = (¬*st*) ∧ *alternatingFriends vl st*
| *alternatingFriends* (*OVal* - # *vl*) *st* = *alternatingFriends vl st*

inductive *BO* :: *value list* ⇒ *value list* ⇒ *bool*

and $BC :: value\ list \Rightarrow value\ list \Rightarrow bool$
where
$BO\text{-}FrVal[simp,intro!]$:
$BO\ (map\ FrVal\ fs)\ (map\ FrVal\ fs)$
$|BO\text{-}BC[intro]$:
$BC\ vl\ vl1 \Longrightarrow$
$BO\ (map\ FrVal\ fs\ @\ OVal\ False\ \#\ vl)\ (map\ FrVal\ fs\ @\ OVal\ False\ \#\ vl1)$

$|BC\text{-}FrVal[simp,intro!]$:
$BC\ (map\ FrVal\ fs)\ (map\ FrVal\ fs1)$
$|BC\text{-}BO[intro]$:
$BO\ vl\ vl1 \Longrightarrow (fs = [] \longleftrightarrow fs1 = []) \Longrightarrow (fs \neq [] \Longrightarrow last\ fs = last\ fs1) \Longrightarrow$
$BC\ (map\ FrVal\ fs\ @\ OVal\ True\ \#\ vl)$
$(map\ FrVal\ fs1\ @\ OVal\ True\ \#\ vl1)$

definition $B\ vl\ vl1 \equiv BO\ vl\ vl1 \wedge alternatingFriends\ vl1\ False$

lemma *BO-Nil-Nil*: $BO\ vl\ vl1 \Longrightarrow vl = [] \Longrightarrow vl1 = []$
⟨*proof*⟩

unbundle *no relcomp-syntax*

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*⟩

7.6 Unwinding proof

lemma *toggle-friends12-True*:
assumes *rs*: *reach s*
and *IDs*: *IDsOK s* [*UID1*, *UID2*] [] [] []
and *nf12*: ¬*friends12 s*
obtains *al oul*
where *sstep s al* = (*oul*, *createFriend s UID1* (*pass s UID1*) *UID2*)
and $al \neq []$ **and** *eqButUID s* (*createFriend s UID1* (*pass s UID1*) *UID2*)
and *friends12* (*createFriend s UID1* (*pass s UID1*) *UID2*)
and $O\ (traceOf\ s\ al) = []$ **and** $V\ (traceOf\ s\ al) = [FrVal\ True]$
⟨*proof*⟩

lemma *toggle-friends12-False*:
assumes *rs*: *reach s*
and *IDs*: *IDsOK s* [*UID1*, *UID2*] [] [] []
and *f12*: *friends12 s*
obtains *al oul*
where *sstep s al* = (*oul*, *deleteFriend s UID1* (*pass s UID1*) *UID2*)
and $al \neq []$ **and** *eqButUID s* (*deleteFriend s UID1* (*pass s UID1*) *UID2*)
and ¬*friends12* (*deleteFriend s UID1* (*pass s UID1*) *UID2*)

and $O\ (traceOf\ s\ al) = []$ **and** $V\ (traceOf\ s\ al) = [FrVal\ False]$
⟨*proof*⟩

definition $\Delta 0 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 0\ s\ vl\ s1\ vl1 \equiv$
$eqButUID\ s\ s1 \wedge friendIDs\ s = friendIDs\ s1 \wedge open\ s \wedge$
$BO\ vl\ vl1 \wedge alternatingFriends\ vl1\ (friends12\ s1)$

definition $\Delta 1 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 1\ s\ vl\ s1\ vl1 \equiv (\exists fs\ fs1.$
$eqButUID\ s\ s1 \wedge \neg open\ s \wedge$
$alternatingFriends\ vl1\ (friends12\ s1) \wedge$
$vl = map\ FrVal\ fs \wedge vl1 = map\ FrVal\ fs1)$

definition $\Delta 2 :: state \Rightarrow value\ list \Rightarrow state \Rightarrow value\ list \Rightarrow bool$ **where**
$\Delta 2\ s\ vl\ s1\ vl1 \equiv (\exists fs\ fs1\ vlr\ vlr1.$
$eqButUID\ s\ s1 \wedge \neg open\ s \wedge BO\ vlr\ vlr1 \wedge$
$alternatingFriends\ vl1\ (friends12\ s1) \wedge$
$(fs = [] \longleftrightarrow fs1 = []) \wedge$
$(fs \neq [] \longrightarrow last\ fs = last\ fs1) \wedge$
$(fs = [] \longrightarrow friendIDs\ s = friendIDs\ s1) \wedge$
$vl = map\ FrVal\ fs\ @\ OVal\ True\ \#\ vlr \wedge$
$vl1 = map\ FrVal\ fs1\ @\ OVal\ True\ \#\ vlr1)$

lemma $\Delta 2$-I:
assumes $eqButUID\ s\ s1\ \neg open\ s\ BO\ vlr\ vlr1\ alternatingFriends\ vl1\ (friends12\ s1)$
$fs = [] \longleftrightarrow fs1 = []\ fs \neq [] \longrightarrow last\ fs = last\ fs1$
$fs = [] \longrightarrow friendIDs\ s = friendIDs\ s1$
$vl = map\ FrVal\ fs\ @\ OVal\ True\ \#\ vlr$
$vl1 = map\ FrVal\ fs1\ @\ OVal\ True\ \#\ vlr1$
shows $\Delta 2\ s\ vl\ s1\ vl1$
⟨*proof*⟩

lemma *istate*-$\Delta 0$:
assumes B: $B\ vl\ vl1$
shows $\Delta 0\ istate\ vl\ istate\ vl1$
⟨*proof*⟩

lemma *unwind-cont*-$\Delta 0$: $unwind\text{-}cont\ \Delta 0\ \{\Delta 0,\Delta 1,\Delta 2\}$
⟨*proof*⟩

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

lemma *unwind-cont*-$\Delta 2$: $unwind\text{-}cont\ \Delta 2\ \{\Delta 2,\Delta 0\}$
⟨*proof*⟩

definition *Gr* **where**
Gr =
{
(Δ*0*, {Δ*0*,Δ*1*,Δ*2*}),
(Δ*1*, {Δ*1*,Δ*0*}),
(Δ*2*, {Δ*2*,Δ*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*} ∩ (*UIDs AID*) = {}
and
UID1-UID2: *UID1* ≠ *UID2*
and
AID-AIDs: *AID* ∈ *AIDs*
begin

sublocale *Issuer*: *Friend UIDs AID UID1 UID2* ⟨*proof*⟩

abbreviation φ :: *apiID* ⇒ (*state*, *act*, *out*) *trans* ⇒ *bool*
where φ *aid trn* ≡ (*Issuer*.φ *trn* ∧ *aid* = *AID*)

abbreviation *f* :: *apiID* ⇒ (*state*, *act*, *out*) *trans* ⇒ *Friend.value*
where *f aid trn* ≡ *Friend.f UID1 UID2 trn*

abbreviation *T* :: *apiID* ⇒ (*state*, *act*, *out*) *trans* ⇒ *bool*

where *T aid trn* ≡ *False*

abbreviation *B* :: *apiID* ⇒ *Friend.value list* ⇒ *Friend.value list* ⇒ *bool*
where *B aid vl vl1* ≡ (*if aid* = *AID then Issuer.B vl vl1 else* (*vl* = [] ∧ *vl1* = []))

abbreviation *comOfV aid vl* ≡ *Internal*
abbreviation *tgtNodeOfV aid vl* ≡ *undefined*
abbreviation *syncV aid1 vl1 aid2 vl2* ≡ *False*

lemma [*simp*]: *validTrans aid trn* ⟹ *lreach aid* (*srcOf trn*) ⟹ *φ aid trn* ⟹ *comOf aid trn* = *Internal*
⟨*proof*⟩

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* = *id*
⟨*proof*⟩

sublocale *BD-Security-TS-Network-Preserve-Source-Security-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* = *id*
⟨*proof*⟩

theorem *secure*: *secure*
⟨*proof*⟩

end

end
theory *Friend-All*
imports *Friend-Network*
begin

end
theory *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 φ :: (*state*,*act*,*out*) *trans* $\Rightarrow$ *bool* **where**
φ (*Trans s* (*Cact* (*cFriendReq uid p uid′ req*)) *ou s′*) =
((*uid*,*uid′*) $\in$ {(*UID1*,*UID2*), (*UID2*,*UID1*)} $\wedge$ *ou* = *outOK*)
|
φ (*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′*)
|
φ (*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′*)
|
φ (*Trans s* (*Cact* (*cUser uid p uid′ p′*)) *ou s′*) =
(*open s* $\neq$ *open s′*)
|
φ - = *False*

fun *f* :: (*state*,*act*,*out*) *trans* $\Rightarrow$ *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 φ: φ (*Trans s a ou s′*) (**is** φ *?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*] [] [] []
¬*friends12 s* ¬*friends12 s′ open s′* = *open s*
UID1 $\in\in$ *pendingFReqs s′ UID2 UID1* $\notin$ *set* (*pendingFReqs s UID2*)
UID2 $\in\in$ *pendingFReqs s′ UID1* $\longleftrightarrow$ *UID2* $\in\in$ *pendingFReqs s UID1*

$$
\begin{aligned}
&\mid (FReq2)\ u\ p\ req\ \textbf{where}\ a = Cact\ (cFriendReq\ UID2\ p\ UID1\ req)\ ou = outOK \\
&\qquad f\ ?trn = FRVal\ u\ req\ u = U2\ IDsOK\ s\ [UID1,\ UID2]\ []\ []\ [] \\
&\qquad \neg friends12\ s\ \neg friends12\ s'\ open\ s' = open\ s \\
&\qquad UID2 \in\in pendingFReqs\ s'\ UID1\ UID2 \notin set\ (pendingFReqs\ s\ UID1) \\
&\qquad UID1 \in\in pendingFReqs\ s'\ UID2 \longleftrightarrow UID1 \in\in pendingFReqs\ s\ UID2 \\
&\mid (Friend)\ uid\ p\ uid'\ \textbf{where}\ a = Cact\ (cFriend\ uid\ p\ uid')\ ou = outOK\ f\ ?trn = FVal\ True \\
&\qquad uid = UID1 \wedge uid' = UID2 \vee uid = UID2 \wedge uid' = UID1 \\
&\qquad IDsOK\ s\ [UID1,\ UID2]\ []\ []\ [] \\
&\qquad \neg friends12\ s\ friends12\ s'\ uid' \in\in pendingFReqs\ s\ uid \\
&\qquad UID1 \notin set\ (pendingFReqs\ s'\ UID2) \\
&\qquad UID2 \notin set\ (pendingFReqs\ s'\ UID1) \\
&\mid (Unfriend)\ uid\ p\ uid'\ \textbf{where}\ a = Dact\ (dFriend\ uid\ p\ uid')\ ou = outOK\ f\ ?trn = FVal\ False \\
&\qquad uid = UID1 \wedge uid' = UID2 \vee uid = UID2 \wedge uid' = UID1 \\
&\qquad IDsOK\ s\ [UID1,\ UID2]\ []\ []\ [] \\
&\qquad friends12\ s\ \neg friends12\ s' \\
&\qquad UID1 \notin set\ (pendingFReqs\ s'\ UID2) \\
&\qquad UID1 \notin set\ (pendingFReqs\ s\ UID2) \\
&\qquad UID2 \notin set\ (pendingFReqs\ s'\ UID1) \\
&\qquad UID2 \notin set\ (pendingFReqs\ s\ UID1) \\
&\mid (OpenF)\ uid\ p\ uid'\ \textbf{where}\ a = Cact\ (cFriend\ uid\ p\ uid') \\
&\qquad (uid \in UIDs \wedge uid' \in \{UID1, UID2\}) \vee (uid' \in UIDs \wedge uid \in \{UID1, UID2\}) \\
&\qquad ou = outOK\ f\ ?trn = OVal\ True\ \neg openByF\ s\ openByF\ s' \\
&\qquad \neg openByA\ s\ \neg openByA\ s' \\
&\qquad friends12\ s' = friends12\ s \\
&\qquad\qquad UID1 \in\in pendingFReqs\ s'\ UID2 \longleftrightarrow UID1 \in\in pendingFReqs\ s\ UID2 \\
&\qquad\qquad UID2 \in\in pendingFReqs\ s'\ UID1 \longleftrightarrow UID2 \in\in pendingFReqs\ s\ UID1 \\
&\mid (CloseF)\ uid\ p\ uid'\ \textbf{where}\ a = Dact\ (dFriend\ uid\ p\ uid') \\
&\qquad (uid \in UIDs \wedge uid' \in \{UID1, UID2\}) \vee (uid' \in UIDs \wedge uid \in \{UID1, UID2\}) \\
&\qquad ou = outOK\ f\ ?trn = OVal\ False\ openByF\ s\ \neg openByF\ s' \\
&\qquad \neg openByA\ s\ \neg openByA\ s' \\
&\qquad friends12\ s' = friends12\ s \\
&\qquad\qquad UID1 \in\in pendingFReqs\ s'\ UID2 \longleftrightarrow UID1 \in\in pendingFReqs\ s\ UID2 \\
&\qquad\qquad UID2 \in\in pendingFReqs\ s'\ UID1 \longleftrightarrow UID2 \in\in pendingFReqs\ s\ UID1 \\
&\mid (CloseA)\ uid\ p\ uid'\ p'\ \textbf{where}\ a = Cact\ (cUser\ uid\ p\ uid'\ p') \\
&\qquad uid' \in \{UID1, UID2\}\ openByA\ s\ \neg openByA\ s'
\end{aligned}
$$

$\neg openByF\ s\ \neg openByF\ s'$
$ou = outOK\ f\ ?trn = OVal\ False$
$friends12\ s' = friends12\ s$
$UID1 \in\in pendingFReqs\ s'\ UID2 \longleftrightarrow UID1 \in\in pendingFReqs\ s\ UID2$
$UID2 \in\in pendingFReqs\ s'\ UID1 \longleftrightarrow UID2 \in\in pendingFReqs\ s\ UID1$
⟨*proof*⟩

lemma *step-open-φ*:
assumes $step\ s\ a = (ou,\ s')$
and $open\ s \neq open\ s'$
shows $\varphi\ (Trans\ s\ a\ ou\ s')$
⟨*proof*⟩

lemma *step-friends12-φ*:
assumes $step\ s\ a = (ou,\ s')$
and $friends12\ s \neq friends12\ s'$
shows $\varphi\ (Trans\ s\ a\ ou\ s')$
⟨*proof*⟩

lemma *step-pendingFReqs-φ*:
assumes $step\ s\ a = (ou,\ s')$
and $(UID1 \in\in pendingFReqs\ s\ UID2) \neq (UID1 \in\in pendingFReqs\ s'\ UID2)$
$\lor\ (UID2 \in\in pendingFReqs\ s\ UID1) \neq (UID2 \in\in pendingFReqs\ s'\ UID1)$
shows $\varphi\ (Trans\ s\ a\ ou\ s')$
⟨*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*: $\forall\, req.\ a \neq Cact\ (cFriend\ UID1\ (pass\ s\ UID1)\ UID2) \land$
$a \neq Cact\ (cFriend\ UID2\ (pass\ s\ UID2)\ UID1) \land$
$a \neq Cact\ (cFriendReq\ UID1\ (pass\ s\ UID1)\ UID2\ req) \land$
$a \neq Cact\ (cFriendReq\ UID2\ (pass\ s\ UID2)\ UID1\ req) \land$
$a \neq Dact\ (dFriend\ UID1\ (pass\ s\ UID1)\ UID2) \land$
$a \neq Dact\ (dFriend\ UID2\ (pass\ s\ UID2)\ UID1)$
and φ: $\varphi\ (Trans\ s\ a\ ou\ s')$
shows $\varphi\ (Trans\ s1\ a\ ou1\ s1')$
⟨*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*: $\forall\, req.\ a \neq Cact\ (cFriend\ UID1\ (pass\ s\ UID1)\ UID2) \land$
$a \neq Cact\ (cFriend\ UID2\ (pass\ s\ UID2)\ UID1) \land$
$a \neq Cact\ (cFriendReq\ UID1\ (pass\ s\ UID1)\ UID2\ req) \land$

$a \neq Cact\ (cFriendReq\ UID2\ (pass\ s\ UID2)\ UID1\ req) \wedge$
$a \neq Dact\ (dFriend\ UID1\ (pass\ s\ UID1)\ UID2) \wedge$
$a \neq Dact\ (dFriend\ UID2\ (pass\ s\ UID2)\ UID1)$
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 $T :: (state,act,out)\ trans \Rightarrow bool$
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-FVal[*simp*,*intro*!]:
BO (*map FVal fs*) (*map FVal fs*)
|*BO-BC*[*intro*]:
BC vl vl1 $\Longrightarrow$
BO (*map FVal fs* @ *OVal False* # *vl*) (*map FVal fs* @ *OVal False* # *vl1*)

|*BC-FVal*[*simp*,*intro*!]:
BC (*map FVal fs*) (*map FVal fs1*)
|*BC-BO*[*intro*]:
BO vl vl1 $\Longrightarrow$ (*fs* = [] $\longleftrightarrow$ *fs1* = []) $\Longrightarrow$ (*fs* $\neq$ [] $\Longrightarrow$ *last fs* = *last fs1*) $\Longrightarrow$
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 istate $\wedge$
(*vl* = [] $\longrightarrow$ *vl1* = []) $\wedge$
(*vl* $\neq$ [] $\wedge$ *hd vl* = *OVal False* $\longrightarrow$ *vl1* $\neq$ [] $\wedge$ *hd vl1* = *OVal False*)

lemma *BO-Nil-iff*: *BO vl vl1* $\Longrightarrow$ *vl* = [] $\longleftrightarrow$ *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*⟩

8.3 Unwinding proof

lemma *validFrom-validValSeq*:
assumes *validFrom s tr*
and *reach s*

shows *validValSeqFrom* (*V tr*) *s*
⟨*proof*⟩

lemma *validFrom istate tr* ⟹ *validValSeqFrom* (*V tr*) *istate*
⟨*proof*⟩

lemma *produce-FRVal*:
assumes *rs*: *reach s*
and *IDs*: *IDsOK s* [*UID1*, *UID2*] [] [] []
and *vVS*: *validValSeqFrom* (*FRVal u req # vl*) *s*
obtains *a uid uid′ s′*
where *step s a* = (*outOK*, *s′*)
and *a* = *Cact* (*cFriendReq uid* (*pass s uid*) *uid′ req*)
and *uid* = *UID1* ∧ *uid′* = *UID2* ∨ *uid* = *UID2* ∧ *uid′* = *UID1*
and *φ* (*Trans s a outOK s′*)
and *f* (*Trans s a outOK s′*) = *FRVal u req*
and *validValSeqFrom vl s′*
⟨*proof*⟩

lemma *toggle-friends12-True*:
assumes *rs*: *reach s*
and *IDs*: *IDsOK s* [*UID1*, *UID2*] [] [] []
and *nf12*: ¬*friends12 s*
and *vVS*: *validValSeqFrom* (*FVal True # vl*) *s*
obtains *a uid uid′ s′*
where *step s a* = (*outOK*, *s′*)
and *a* = *Cact* (*cFriend uid* (*pass s uid*) *uid′*)
and *s′* = *createFriend s UID1* (*pass s UID1*) *UID2*
and *uid* = *UID1* ∧ *uid′* = *UID2* ∨ *uid* = *UID2* ∧ *uid′* = *UID1*
and *friends12 s′*
and *eqButUID s s′*
and *φ* (*Trans s a outOK s′*)
and *f* (*Trans s a outOK s′*) = *FVal True*
and ¬*γ* (*Trans s a outOK s′*)
and *validValSeqFrom vl s′*
⟨*proof*⟩

lemma *toggle-friends12-False*:
assumes *rs*: *reach s*
and *IDs*: *IDsOK s* [*UID1*, *UID2*] [] [] []
and *f12*: *friends12 s*
and *vVS*: *validValSeqFrom* (*FVal False # vl*) *s*
obtains *a s′*
where *step s a* = (*outOK*, *s′*)
and *a* = *Dact* (*dFriend UID1* (*pass s UID1*) *UID2*)
and *s′* = *deleteFriend s UID1* (*pass s UID1*) *UID2*
and ¬*friends12 s′*
and *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* ≠ *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* # *map FVal fs* **and** *vl1* = []
| (*FVal1*) *fv fs fs1* **where** *vl* = *map FVal fs* **and** *vl1* = *FVal fv* # *map FVal fs1*
| (*BO-FVal*) *fv fv' fs vl' vl1'* **where** *vl* = *FVal fv* # *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* = *map FVal fs* @ *FVal fv'* # *OVal True* # *vl'*
and *vl1* = *FVal fv* # *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′*
⟨*proof*⟩

definition Δ*0* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*0 s vl s1 vl1* ≡
s = *s1* ∧ *B vl vl1* ∧ *open s* ∧ (¬*IDsOK s* [*UID1*, *UID2*] [] [] [])

definition Δ*1* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*1 s vl s1 vl1* ≡
eqButUID s s1 ∧ *friendIDs s* = *friendIDs s1* ∧ *open s* ∧
BO (*filter* (*Not o isFRVal*) *vl*) (*filter* (*Not o isFRVal*) *vl1*) ∧
validValSeqFrom vl1 s1 ∧
IDsOK s1 [*UID1*, *UID2*] [] [] []

definition Δ*2* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*2 s vl s1 vl1* ≡ (∃ *fs fs1*.
eqButUID s s1 ∧ ¬*open s* ∧
validValSeqFrom vl1 s1 ∧
filter (*Not o isFRVal*) *vl* = *map FVal fs* ∧
filter (*Not o isFRVal*) *vl1* = *map FVal fs1*)

definition Δ*3* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*3 s vl s1 vl1* ≡ (∃ *fs fs1 vlr vlr1*.
eqButUID s s1 ∧ ¬*open s* ∧ *BO vlr vlr1* ∧
validValSeqFrom vl1 s1 ∧
(*fs* = [] ⟷ *fs1* = []) ∧
(*fs* ≠ [] ⟶ *last fs* = *last fs1*) ∧
(*fs* = [] ⟶ *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*)

lemma Δ*2-I*:
assumes *eqButUID s s1* ¬*open s*
validValSeqFrom vl1 s1
filter (*Not o isFRVal*) *vl* = *map FVal fs*
filter (*Not o isFRVal*) *vl1* = *map FVal fs1*
shows Δ*2 s vl s1 vl1*
⟨*proof*⟩

lemma Δ*3-I*:
assumes *eqButUID s s1* ¬*open s BO vlr vlr1*
validValSeqFrom vl1 s1
fs = [] ⟷ *fs1* = [] *fs* ≠ [] ⟶ *last fs* = *last fs1*
fs = [] ⟶ *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 Δ*3 s vl s1 vl1*

⟨*proof*⟩

lemma *istate-Δ0*:
assumes *B*: *B vl vl1*
shows *Δ0 istate vl istate vl1*
⟨*proof*⟩

lemma *unwind-cont-Δ0*: *unwind-cont Δ0* {*Δ0*,*Δ1*,*Δ2*,*Δ3*}
⟨*proof*⟩

lemma *unwind-cont-Δ1*: *unwind-cont Δ1* {*Δ1*,*Δ2*,*Δ3*}
⟨*proof*⟩

lemma *unwind-cont-Δ2*: *unwind-cont Δ2* {*Δ2*, *Δ1*}
⟨*proof*⟩

lemma *unwind-cont-Δ3*: *unwind-cont Δ3* {*Δ3*,*Δ1*}
⟨*proof*⟩

definition *Gr* **where**
Gr =
{
(*Δ0*, {*Δ0*,*Δ1*,*Δ2*,*Δ3*}),
(*Δ1*, {*Δ1*,*Δ2*,*Δ3*}),
(*Δ2*, {*Δ2*,*Δ1*}),
(*Δ3*, {*Δ3*,*Δ1*})
}

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*} ∩ (*UIDs AID*) = {}
and
UID1-UID2: *UID1* ≠ *UID2*
and
AID-AIDs: *AID* ∈ *AIDs*
begin

sublocale *Issuer*: *Friend UIDs AID UID1 UID2* ⟨*proof*⟩

abbreviation φ :: *apiID* ⇒ (*state*, *act*, *out*) *trans* ⇒ *bool*
where φ *aid trn* ≡ (*Issuer*.φ *trn* ∧ *aid* = *AID*)

abbreviation *f* :: *apiID* ⇒ (*state*, *act*, *out*) *trans* ⇒ *Friend.value*
where *f aid trn* ≡ *Friend.f UID1 UID2 trn*

abbreviation *T* :: *apiID* ⇒ (*state*, *act*, *out*) *trans* ⇒ *bool*
where *T aid trn* ≡ *False*

abbreviation *B* :: *apiID* ⇒ *Friend.value list* ⇒ *Friend.value list* ⇒ *bool*
where *B aid vl vl1* ≡ (*if aid* = *AID then Issuer.B vl vl1 else* (*vl* = [] ∧ *vl1* = []))

abbreviation *comOfV aid vl* ≡ *Internal*
abbreviation *tgtNodeOfV aid vl* ≡ *undefined*
abbreviation *syncV aid1 vl1 aid2 vl2* ≡ *False*

lemma [*simp*]: *validTrans aid trn* ⟹ *lreach aid* (*srcOf trn*) ⟹ φ *aid trn* ⟹ *comOf aid trn* = *Internal*
⟨*proof*⟩

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* = *id*
⟨*proof*⟩

sublocale *BD-Security-TS-Network-Preserve-Source-Security-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* = *id*
⟨*proof*⟩

theorem *secure*: *secure*
⟨*proof*⟩

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* ⇒ *userID set*
and *AID* :: *apiID*
and *UID* :: *userID*
assumes *UID-UIDs*: *UID* ∉ *UIDs AID*
and *emptyUserID-not-UIDs*: ⋀*aid*. *emptyUserID* ∉ *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 γ :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
γ (*Trans - a ou -*) ⟷ (∃ *uid*. *userOfA a* = *Some uid* ∧ *uid* ∈ *UIDs AID*) ∨
(∃ *ca*. *a* = *COMact ca* ∧ *ou* ≠ *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* ⇒ *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′*) =
(*if uID* = *UID* ∧ *uID′* ∉ *UIDs aID*
then comSendCreateOFriend uID emptyPass aID emptyUserID
else comSendCreateOFriend uID emptyPass aID uID′)
|*comPurge* (*comReceiveCreateOFriend aID cp uID uID′*) = *comReceiveCreateOFriend aID cp uID uID′*
|*comPurge* (*comSendDeleteOFriend uID p aID uID′*) =
(*if uID* = *UID* ∧ *uID′* ∉ *UIDs aID*
then comSendCreateOFriend uID emptyPass aID emptyUserID
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* ⟷ (∃ *p′*. *ca* = *comSendServerReq uID p′ aID reqInfo* ∧ *p* = *emptyPass*)
comPurge ca = *comReceiveClientReq aID reqInfo* ⟷ *ca* = *comReceiveClientReq aID reqInfo*
comPurge ca = *comConnectClient uID p aID sp* ⟷ (∃ *p′*. *ca* = *comConnectClient uID p′ aID sp* ∧ *p* = *emptyPass*)
comPurge ca = *comConnectServer aID sp* ⟷ *ca* = *comConnectServer aID sp*
comPurge ca = *comReceivePost aID sp nID nt uID v* ⟷ *ca* = *comReceivePost aID sp nID nt uID v*
comPurge ca = *comSendPost uID p aID nID* ⟷ (∃ *p′*. *ca* = *comSendPost uID p′ aID nID* ∧ *p* = *emptyPass*)
comPurge ca = *comSendCreateOFriend uID p aID uID′*
⟷ (∃ *p′ uid″*. (*ca* = *comSendCreateOFriend uID p′ aID uid″* ∨ *ca* = *comSendDeleteOFriend uID p′ aID uid″*) ∧ *uID* = *UID* ∧ *uid″* ∉ *UIDs aID* ∧ *uID′* = *emptyUserID* ∧ *p* = *emptyPass*)
∨ (∃ *p′*. *ca* = *comSendCreateOFriend uID p′ aID uID′* ∧ ¬(*uID* = *UID* ∧ *uID′* ∉ *UIDs aID*) ∧ *p* = *emptyPass*)
comPurge ca = *comReceiveCreateOFriend aID cp uID uID′* ⟷ *ca* = *comReceiveCreateOFriend aID cp uID uID′*
comPurge ca = *comSendDeleteOFriend uID p aID uID′* ⟷ (∃ *p′*. *ca* = *comSendDeleteOFriend uID p′ aID uID′* ∧ ¬(*uID* = *UID* ∧ *uID′* ∉ *UIDs aID*) ∧ *p* = *emptyPass*)

comPurge ca = comReceiveDeleteOFriend aID cp uID uID′ ⟷ 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* ⇒ *out* **where**
outPurge (*O-sendCreateOFriend* (*aID*, *sp*, *uID*, *uID′*)) =
(*if uID = UID ∧ uID′ ∉ 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 ∧ uID′ ∉ 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 ⟷ ou = outErr
outPurge ou = outOK ⟷ ou = outOK
outPurge ou = O-sendServerReq ossr ⟷ ou = O-sendServerReq ossr
outPurge ou = O-connectClient occ ⟷ ou = O-connectClient occ
outPurge ou = O-sendPost osn ⟷ ou = O-sendPost osn
outPurge ou = O-sendCreateOFriend (*aID*, *sp*, *uID*, *uID′*)
⟷ (∃ *uid″*. (*ou = O-sendCreateOFriend* (*aID*, *sp*, *uID*, *uid″*) ∨ *ou = O-sendDeleteOFriend* (*aID*, *sp*, *uID*, *uid″*)) ∧ *uID = UID ∧ uid″ ∉ UIDs aID ∧ uID′ = emptyUserID*)
∨ (*ou = O-sendCreateOFriend* (*aID*, *sp*, *uID*, *uID′*) ∧ ¬(*uID = UID ∧ uID′ ∉ UIDs aID*))
outPurge ou = O-sendDeleteOFriend (*aID*, *sp*, *uID*, *uID′*)
⟷ (*ou = O-sendDeleteOFriend* (*aID*, *sp*, *uID*, *uID′*) ∧ ¬(*uID = UID ∧ uID′ ∉ UIDs aID*))
⟨*proof*⟩

fun *g* :: (*state*,*act*,*out*)*trans* ⇒ *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*)
⟷ (∃ *p′*. *a = COMact* (*comSendServerReq uID p′ aID reqInfo*) ∧ *p = emptyPass ∧ ou = O-sendServerReq ossr*)
g (*Trans s a ou s′*) = (*COMact* (*comReceiveClientReq aID reqInfo*), *outOK*)
⟷ *a = COMact* (*comReceiveClientReq aID reqInfo*) ∧ *ou = outOK*
g (*Trans s a ou s′*) = (*COMact* (*comConnectClient uID p aID sp*), *O-connectClient occ*)
⟷ (∃ *p′*. *a = COMact* (*comConnectClient uID p′ aID sp*) ∧ *p = emptyPass ∧ ou = O-connectClient occ*)
g (*Trans s a ou s′*) = (*COMact* (*comConnectServer aID sp*), *outOK*)

$\longleftrightarrow a = COMact\ (comConnectServer\ aID\ sp) \wedge ou = outOK$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceivePost\ aID\ sp\ nID\ nt\ uID\ v),\ outOK)$
$\longleftrightarrow a = COMact\ (comReceivePost\ aID\ sp\ nID\ nt\ uID\ v) \wedge ou = outOK$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendPost\ uID\ p\ aID\ nID),\ O\text{-}sendPost\ osn)$
$\longleftrightarrow (\exists p'.\ a = COMact\ (comSendPost\ uID\ p'\ aID\ nID) \wedge p = emptyPass \wedge ou = O\text{-}sendPost\ osn)$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendCreateOFriend\ uID\ p\ aID\ uID'),\ O\text{-}sendCreateOFriend\ (aid,\ sp,\ uid,\ uid'))$
$\longleftrightarrow ((\exists p'\ uid''.\ (a = COMact\ (comSendCreateOFriend\ uID\ p'\ aID\ uid'') \vee a = COMact\ (comSendDeleteOFriend\ uID\ p'\ aID\ uid'')) \wedge uID = UID \wedge uid'' \notin UIDs\ aID \wedge uID' = emptyUserID \wedge p = emptyPass)$
$\vee (\exists p'.\ a = COMact\ (comSendCreateOFriend\ uID\ p'\ aID\ uID') \wedge \neg(uID = UID \wedge uID' \notin UIDs\ aID) \wedge p = emptyPass))$
$\wedge ((\exists uid''.\ (ou = O\text{-}sendCreateOFriend\ (aid,\ sp,\ uid,\ uid'') \vee ou = O\text{-}sendDeleteOFriend\ (aid,\ sp,\ uid,\ uid'')) \wedge uid = UID \wedge uid'' \notin UIDs\ aid \wedge uid' = emptyUserID)$
$\vee (ou = O\text{-}sendCreateOFriend\ (aid,\ sp,\ uid,\ uid') \wedge \neg(uid = UID \wedge uid' \notin UIDs\ aid)))$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID'),\ outOK)$
$\longleftrightarrow a = COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID') \wedge ou = outOK$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendDeleteOFriend\ uID\ p\ aID\ uID'),\ O\text{-}sendDeleteOFriend\ (aid,\ sp,\ uid,\ uid'))$
$\longleftrightarrow (\exists p'.\ a = COMact\ (comSendDeleteOFriend\ uID\ p'\ aID\ uID') \wedge \neg(uID = UID \wedge uID' \notin UIDs\ aID) \wedge p = emptyPass)$
$\wedge (ou = O\text{-}sendDeleteOFriend\ (aid,\ sp,\ uid,\ uid') \wedge \neg(uid = UID \wedge uid' \notin UIDs\ aid))$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'),\ outOK)$
$\longleftrightarrow a = COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID') \wedge ou = outOK$
$\langle proof \rangle$

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* :: $(apiID \times userID)\ list \Rightarrow (apiID \times userID)\ list$ **where**
$filterUIDs\ auidl = filter\ (\lambda auid.\ (snd\ auid) \in UIDs\ (fst\ auid))\ auidl$

fun *removeUIDs* :: $(apiID \times userID)\ list \Rightarrow (apiID \times userID)\ list$ **where**

removeUIDs auidl = *filter* (λ*auid*. (*snd auid*) ∉ *UIDs* (*fst auid*)) *auidl*

fun *eqButUIDs* :: (*apiID* × *userID*) *list* ⇒ (*apiID* × *userID*) *list* ⇒ *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 ≡
  eqButUIDs (*frds UID*) (*frds1 UID*)
∧ (∀ *uid*. *uid* ≠ *UID* ⟶ *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*:
⟦*eqButUIDf frds frds1*; *uid* ≠ *UID*⟧ ⟹ *frds uid* = *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 ⟹ eqButUIDs (*sentOuterFriendIDs s UID*) (*sentOuterFriendIDs s1 UID*)
⟨*proof*⟩

lemma *eqButUID-not-UID*:
eqButUID s s1 ⟹ uid ≠ UID ⟹ sentOuterFriendIDs s uid = sentOuterFriendIDs s1 uid
⟨*proof*⟩

lemma *eqButUID-sentOuterFriends-UIDs*:
assumes *eqButUID s s1*
and *uid′ ∈ 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*:
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|admin := uu1|\!))\ (s1\ (\!|admin := uu2|\!))$

$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|pendingUReqs := uu1|\!))\ (s1\ (\!|pendingUReqs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|userReq := uu1|\!))\ (s1\ (\!|userReq := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|userIDs := uu1|\!))\ (s1\ (\!|userIDs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|user := uu1|\!))\ (s1\ (\!|user := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|pass := uu1|\!))\ (s1\ (\!|pass := uu2|\!))$

$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|postIDs := uu1|\!))\ (s1\ (\!|postIDs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|owner := uu1|\!))\ (s1\ (\!|owner := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|post := uu1|\!))\ (s1\ (\!|post := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|vis := uu1|\!))\ (s1\ (\!|vis := uu2|\!))$

$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|pendingFReqs := uu1|\!))\ (s1\ (\!|pendingFReqs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|friendReq := uu1|\!))\ (s1\ (\!|friendReq := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|friendIDs := uu1|\!))\ (s1\ (\!|friendIDs := uu2|\!))$

$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|pendingSApiReqs$

$:= uu1\rparr)\ (s1\ (\lparr pendingSApiReqs := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr sApiReq := uu1\rparr)$
$(s1\ (\lparr sApiReq := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr serverApiIDs :=$
$uu1\rparr)\ (s1\ (\lparr serverApiIDs := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr serverPass := uu1\rparr)$
$(s1\ (\lparr serverPass := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr outerPostIDs :=$
$uu1\rparr)\ (s1\ (\lparr outerPostIDs := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr outerPost := uu1\rparr)$
$(s1\ (\lparr outerPost := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr outerVis := uu1\rparr)$
$(s1\ (\lparr outerVis := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr outerOwner :=$
$uu1\rparr)\ (s1\ (\lparr outerOwner := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow eqButUIDf\ uu1\ uu2 \Longrightarrow eqButUID\ (s\ (\lparr sentOuter-$
$FriendIDs := uu1\rparr)\ (s1\ (\lparr sentOuterFriendIDs := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr recvOuterFriendIDs$
$:= uu1\rparr)\ (s1\ (\lparr recvOuterFriendIDs := uu2\rparr)$

$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr pendingCApiReqs$
$:= uu1\rparr)\ (s1\ (\lparr pendingCApiReqs := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr cApiReq := uu1\rparr)$
$(s1\ (\lparr cApiReq := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr clientApiIDs :=$
$uu1\rparr)\ (s1\ (\lparr clientApiIDs := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr clientPass := uu1\rparr)$
$(s1\ (\lparr clientPass := uu2\rparr)$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\lparr sharedWith :=$
$uu1\rparr)\ (s1\ (\lparr sharedWith:= uu2\rparr)$
$\langle proof \rangle$

lemma *distinct-remove1-idem*: $distinct\ xs \Longrightarrow remove1\ y\ (remove1\ y\ xs) = remove1\ y\ xs$
$\langle proof \rangle$

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* ⇒ *bool*
where *open s* ≡ ∃ *uid* ∈ *UIDs AID*. *uid* ∈∈ *friendIDs s UID*

lemma *open-step-cases*:
assumes *open s* ≠ *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 ∈ *UIDs AID* ∧ *uid'* = *UID* ∨ *uid* = *UID* ∧ *uid'* ∈ *UIDs AID*
open s' ¬*open s*
| (*CloseF*) *uid p uid'* **where** *a* = *Dact* (*dFriend uid p uid'*) *ou* = *outOK p* = *pass s uid*
uid ∈ *UIDs AID* ∧ *uid'* = *UID* ∨ *uid* = *UID* ∧ *uid'* ∈ *UIDs AID*
open s ¬*open s'*
⟨*proof*⟩

lemma *COMact-open*:
assumes *step s a* = (*ou*, *s'*)
and *a* = *COMact ca*
shows *open s* = *open s'*
⟨*proof*⟩

lemma *eqButUID-open-eq*: *eqButUID s s1* ⟹ *open s* = *open s1*
⟨*proof*⟩

end

end

theory *Outer-Friend-Issuer-Value-Setup*
imports *Outer-Friend-Issuer-Openness*
begin

9.1.4 Value setup

context *OuterFriendIssuer*
begin

fun φ :: (*state*,*act*,*out*) *trans* $\Rightarrow$ *bool* **where**
φ (*Trans s* (*COMact* (*comSendCreateOFriend uID p aID uID'*)) *ou s'*) =
(*uID* = *UID* $\wedge$ *uID'* $\notin$ *UIDs aID* $\wedge$ *ou* $\neq$ *outErr*)
|
φ (*Trans s* (*COMact* (*comSendDeleteOFriend uID p aID uID'*)) *ou s'*) =
(*uID* = *UID* $\wedge$ *uID'* $\notin$ *UIDs aID* $\wedge$ *ou* $\neq$ *outErr*)
|
φ (*Trans s - - s'*) = (*open s* $\neq$ *open s'*)

fun *f* :: (*state*,*act*,*out*) *trans* $\Rightarrow$ *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 φ: φ (*Trans s a ou s'*) (**is** φ *?trn*)
and *step*: *step s a* = (*ou*, *s'*)
and *rs*: *reach s*
obtains
(*Friend*) *p aID uID'* **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'* **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'* **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'* **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*: ¬*open s1*
and *φ*: *φ* (*Trans s1 a ou1 s1'*) ⟷ *φ* (*Trans s a ou s'*)
shows *ou* = *ou1*
⟨*proof*⟩

lemma *step-open-φ*:
assumes *step s a* = (*ou*, *s'*)
and *open s* ≠ *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'* ∉ *UIDs aID*
and *a* = *COMact* (*comSendCreateOFriend UID* (*pass s UID*) *aID uID'*) ∨
    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*: ∀ *aID uID'*. *uID'* ∉ *UIDs aID* ⟶
        a ≠ *COMact* (*comSendCreateOFriend UID* (*pass s UID*) *aID uID'*)
∧
        a ≠ *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*: ∀ *aID uID'*. *uID'* ∉ *UIDs aID* ⟶
        a ≠ *COMact* (*comSendCreateOFriend UID* (*pass s UID*) *aID uID'*)
∧
        a ≠ *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*: ∀ *aID uID'*. *uID'* ∉ *UIDs aID* ⟶
a ≠ *COMact* (*comSendCreateOFriend UID* (*pass s UID*) *aID uID'*)
∧
a ≠ *COMact* (*comSendDeleteOFriend UID* (*pass s UID*) *aID uID'*)
shows γ (*Trans s a ou s'*) = γ (*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* ⇒ *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* ⇒ (*apiID* × *userID*) *list* ⇒ *bool* **where**
validValSeq [] - = *True*
| *validValSeq* (*FrVal aid uid True* # *vl*) *auidl* ⟷ (*aid*, *uid*) ∉ *set auidl* ∧ *uid* ∉ *UIDs aid* ∧ *validValSeq vl* (*auidl* ## (*aid*, *uid*))
| *validValSeq* (*FrVal aid uid False* # *vl*) *auidl* ⟷ (*aid*, *uid*) ∈∈ *auidl* ∧ *uid* ∉ *UIDs aid* ∧ *validValSeq vl* (*removeAll* (*aid*, *uid*) *auidl*)
| *validValSeq* (*OVal* - # *vl*) *auidl* = *validValSeq vl auidl*

abbreviation *validValSeqFrom* :: *value list* ⇒ *state* ⇒ *bool* **where**
validValSeqFrom vl s ≡ *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* :: *value list* ⇒ *value list* ⇒ *bool*
where
BC-Nil[*simp*,*intro*]: *BC* [] []
| *BC-FrVal*[*intro*]:
BC vl vl1 ⟹ *uid′* ∉ *UIDs aid* ⟹ *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* ≡
(*vl1* = *vl*) ∨
(∃ *vl0 vl′ vl1′*. *vl* = *vl0* @ *OVal False* # *vl′* ∧ *vl1* = *vl0* @ *OVal False* # *vl1′* ∧ *BC vl′ vl1′*)

definition *B vl vl1* ≡ (*BC vl vl1* ∨ *BO vl vl1*) ∧ *validValSeqFrom vl1 istate*

lemma *B-Nil-Nil*: *B vl vl1* ⟹ *vl1* = [] ⟷ *vl* = []
⟨*proof*⟩

sublocale *BD-Security-IO* **where**
istate = *istate* **and** *step* = *step* **and**
φ = φ **and** *f* = *f* **and** γ = γ **and** *g* = *g* **and** *T* = *T* **and** *B* = *B*
⟨*proof*⟩

9.1.6 Unwinding proof

definition Δ*0* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*0 s vl s1 vl1* ≡
s1 = *istate* ∧ *s* = *istate* ∧ *B vl vl1*

definition Δ*1* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*1 s vl s1 vl1* ≡
BO vl vl1 ∧
s1 = *s* ∧
validValSeqFrom vl1 s1

definition Δ*2* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*2 s vl s1 vl1* ≡
BC vl vl1 ∧
eqButUID s s1 ∧ ¬*open s1* ∧
validValSeqFrom vl1 s1

lemma *validValSeq-prefix*: *validValSeq* (*vl* @ *vl'*) *auidl* ⟹ *validValSeq vl auidl*
⟨*proof*⟩

lemma *filter-removeAll*: *filter P* (*removeAll x xs*) = *removeAll x* (*filter P xs*)
⟨*proof*⟩

lemma *step-validValSeqFrom*:
assumes *step*: *step s a* = (*ou*, *s'*)
and *rs*: *reach s*
and *c*: *consume* (*Trans s a ou s'*) *vl vl'* (**is** *consume ?trn vl vl'*)
and *vVS*: *validValSeqFrom vl s*
shows *validValSeqFrom vl' s'*
⟨*proof*⟩

lemma *istate-*Δ*0*:
assumes *B*: *B vl vl1*
shows Δ*0 istate vl istate vl1*
⟨*proof*⟩

lemma *unwind-cont-*Δ*0*: *unwind-cont* Δ*0* {Δ*1*,Δ*2*}
⟨*proof*⟩

lemma *unwind-cont-*Δ*1*: *unwind-cont* Δ*1* {Δ*1*,Δ*2*}
⟨*proof*⟩

lemma *unwind-cont-*Δ*2*: *unwind-cont* Δ*2* {Δ*2*}
⟨*proof*⟩

definition *Gr* **where**
Gr =
{
(Δ*0*, {Δ*1*,Δ*2*}),
(Δ*1*, {Δ*1*,Δ*2*}),
(Δ*2*, {Δ*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'*)
⟨*proof*⟩

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 *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'*)

$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID'),\ ou')$
$\longleftrightarrow (((\exists\, uid''.\ (a = COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uid'') \vee a = COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uid'')) \wedge aID = AID \wedge uID = UID \wedge uid'' \notin UIDs\ AID' \wedge uID' = emptyUserID)$
$\vee\ (a = COMact\ (comReceiveCreateOFriend\ aID\ cp\ uID\ uID') \wedge \neg(aID = AID \wedge uID = UID \wedge uID' \notin UIDs\ AID')))$
$\wedge\ ou = ou')$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comSendDeleteOFriend\ uID\ p\ aID\ uID'),\ ou')$
$\longleftrightarrow (\exists\, p'.\ a = COMact\ (comSendDeleteOFriend\ uID\ p'\ aID\ uID') \wedge p = emptyPass \wedge ou = ou')$
$g\ (Trans\ s\ a\ ou\ s') = (COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID'),\ ou')$
$\longleftrightarrow a = COMact\ (comReceiveDeleteOFriend\ aID\ cp\ uID\ uID') \wedge \neg(aID = AID \wedge uID = UID \wedge uID' \notin UIDs\ AID') \wedge ou = ou'$
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *eqButUIDl-sym*:
assumes *eqButUIDl auidl auidl1*
shows *eqButUIDl auidl1 auidl*
⟨*proof*⟩

lemma *eqButUIDl-trans*:
assumes *eqButUIDl auidl auidl1* **and** *eqButUIDl auidl1 auidl2*
shows *eqButUIDl auidl auidl2*
⟨*proof*⟩

lemma *eqButUIDl-remove1-cong*:
assumes *eqButUIDl auidl auidl1*
shows *eqButUIDl* (*remove1 auid auidl*) (*remove1 auid auidl1*)
⟨*proof*⟩

lemma *eqButUIDl-snoc-cong*:
assumes *eqButUIDl auidl auidl1*
and *auid′* ∈∈ *auidl* ⟷ *auid′* ∈∈ *auidl1*
shows *eqButUIDl* (*auidl ## auid′*) (*auidl1 ## auid′*)
⟨*proof*⟩

definition *eqButUIDf* **where**
eqButUIDf frds frds1 ≡
(∀ *uid*. *if uid* ∈ *UIDs AID′ then frds uid* = *frds1 uid else eqButUIDl* (*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* ∈ *UIDs AID′* ⟹ *uu* = *uu1*
and *uid* ∉ *UIDs AID′* ⟹ *eqButUIDl uu uu1*
shows *eqButUIDf* (*frds* (*uid* := *uu*)) (*frds1*(*uid* := *uu1*))
⟨*proof*⟩

lemma *eqButUIDf-UIDs*:
⟦*eqButUIDf frds frds1*; *uid* ∈ *UIDs AID′*⟧ ⟹ *frds uid* = *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 ∧
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

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
⟨*proof*⟩

lemma *eqButUID-UIDs*:
eqButUID s s1 ⟹ uid ∈ UIDs AID' ⟹ recvOuterFriendIDs s uid = recvOuterFriendIDs s1 uid
⟨*proof*⟩

lemma *eqButUID-recvOuterFriends-UIDs*:
assumes *eqButUID s s1*
and *uid ≠ UID ∨ aid ≠ AID*
shows (*aid*, *uid*) ∈∈ *recvOuterFriendIDs s uid'* ⟷ (*aid*, *uid*) ∈∈ *recvOuterFriendIDs s1 uid'*
⟨*proof*⟩

lemma *eqButUID-remove1-UID-recvOuterFriends*:
assumes *eqButUID s s1*
shows *remove1 (AID,UID) (recvOuterFriendIDs s uid) = remove1 (AID,UID) (recvOuterFriendIDs s1 uid)*
⟨*proof*⟩

lemma *eqButUID-cong*:
⋀ *uu1 uu2. eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|admin := uu1|)) (s1 (|admin := uu2|))*

⋀ *uu1 uu2. eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|pendingUReqs := uu1|)) (s1 (|pendingUReqs := uu2|))*
⋀ *uu1 uu2. eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|userReq := uu1|)) (s1 (|userReq := uu2|))*
⋀ *uu1 uu2. eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|userIDs := uu1|)) (s1 (|userIDs := uu2|))*
⋀ *uu1 uu2. eqButUID s s1 ⟹ uu1 = uu2 ⟹ eqButUID (s (|user := uu1|)) (s1 (|user := uu2|))*

$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|pass := uu1|\!))\ (s1\ (\!|pass := uu2|\!))$

$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|postIDs := uu1|\!))\ (s1\ (\!|postIDs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|owner := uu1|\!))\ (s1\ (\!|owner := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|post := uu1|\!))\ (s1\ (\!|post := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|vis := uu1|\!))\ (s1\ (\!|vis := uu2|\!))$

$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|pendingFReqs := uu1|\!))\ (s1\ (\!|pendingFReqs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|friendReq := uu1|\!))\ (s1\ (\!|friendReq := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|friendIDs := uu1|\!))\ (s1\ (\!|friendIDs := uu2|\!))$

$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|pendingSApiReqs := uu1|\!))\ (s1\ (\!|pendingSApiReqs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|sApiReq := uu1|\!))\ (s1\ (\!|sApiReq := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|serverApiIDs := uu1|\!))\ (s1\ (\!|serverApiIDs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|serverPass := uu1|\!))\ (s1\ (\!|serverPass := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|outerPostIDs := uu1|\!))\ (s1\ (\!|outerPostIDs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|outerPost := uu1|\!))\ (s1\ (\!|outerPost := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|outerVis := uu1|\!))\ (s1\ (\!|outerVis := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|outerOwner := uu1|\!))\ (s1\ (\!|outerOwner := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|sentOuterFriendIDs := uu1|\!))\ (s1\ (\!|sentOuterFriendIDs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow eqButUIDf\ uu1\ uu2 \Longrightarrow eqButUID\ (s\ (\!|recvOuterFriendIDs := uu1|\!))\ (s1\ (\!|recvOuterFriendIDs := uu2|\!))$

$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|pendingCApiReqs := uu1|\!))\ (s1\ (\!|pendingCApiReqs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|cApiReq := uu1|\!))\ (s1\ (\!|cApiReq := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|clientApiIDs := uu1|\!))\ (s1\ (\!|clientApiIDs := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|clientPass := uu1|\!))\ (s1\ (\!|clientPass := uu2|\!))$
$\bigwedge uu1\ uu2.\ eqButUID\ s\ s1 \Longrightarrow uu1 = uu2 \Longrightarrow eqButUID\ (s\ (\!|sharedWith :=$

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 φ :: (*state*,*act*,*out*) *trans* ⇒ *bool* **where**
φ (*Trans s* (*COMact* (*comReceiveCreateOFriend aID cp uID uID'*)) *ou s'*) =
(*aID* = *AID* ∧ *uID* = *UID* ∧ *uID'* ∉ *UIDs AID'* ∧ *ou* = *outOK*)
|
φ (*Trans s* (*COMact* (*comReceiveDeleteOFriend aID cp uID uID'*)) *ou s'*) =
(*aID* = *AID* ∧ *uID* = *UID* ∧ *uID'* ∉ *UIDs AID'* ∧ *ou* = *outOK*)
|
φ - = *False*

fun *f* :: (*state*,*act*,*out*) *trans* ⇒ *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'*) ∨ *a* = *COMact* (*comReceiveDeleteOFriend AID cp UID uID'*)
and *uID'* ∉ *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 *φ*: *φ* (*Trans s1 a ou1 s1′*) ⟷ *φ* (*Trans s a ou s′*)
and *γ*: *γ* (*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 *φ*: *φ* (*Trans s1 a ou1 s1′*) ⟷ *φ* (*Trans s a ou s′*)
shows *γ* (*Trans s a ou s′*) = *γ* (*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*) *trans* ⇒ *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* ⇒ *userID set* **where**
friendsOfUID s = {*uid*. (*AID*,*UID*) ∈∈ *recvOuterFriendIDs s uid* ∧ *uid* ∉ *UIDs AID′*}

fun *validValSeq* :: *value list* ⇒ *userID set* ⇒ *bool* **where**
validValSeq [] - = *True*
| *validValSeq* (*FrVal aid uid True* # *vl*) *uids* ⟷ *uid* ∉ *uids* ∧ *aid* = *AID′* ∧ *uid* ∉ *UIDs AID′* ∧ *validValSeq vl* (*insert uid uids*)
| *validValSeq* (*FrVal aid uid False* # *vl*) *uids* ⟷ *uid* ∈ *uids* ∧ *aid* = *AID′* ∧ *uid* ∉ *UIDs AID′* ∧ *validValSeq vl* (*uids* − {*uid*})
| *validValSeq* (*OVal ov* # *vl*) *uids* ⟷ *False*

abbreviation *validValSeqFrom vl s* ≡ *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* ⇒ *value list* ⇒ *bool*
where
BC-Nil[*simp*,*intro*]: *BC* [] []
| *BC-FrVal*[*intro*]:
BC vl vl1 ⟹ *uid′* ∉ *UIDs AID′* ⟹ *BC* (*FrVal aid uid st* # *vl*) (*FrVal AID′ uid′ st′* # *vl1*)

definition *B vl vl1* ≡ *BC vl vl1* ∧ *validValSeqFrom vl1 istate*

lemma *BC-Nil-Nil*: *BC vl vl1* ⟹ *vl1* = [] ⟷ *vl* = []
⟨*proof*⟩

lemma *BC-id*: *validValSeq vl uids* ⟹ *BC vl vl*
⟨*proof*⟩

lemma *BC-append*: *BC vl vl1* ⟹ *BC vl′ vl1′* ⟹ *BC* (*vl* @ *vl′*) (*vl1* @ *vl1′*)
⟨*proof*⟩

sublocale *BD-Security-IO* **where**
istate = *istate* **and** *step* = *step* **and**
φ = φ **and** *f* = *f* **and** γ = γ **and** *g* = *g* **and** *T* = *T* **and** *B* = *B*
⟨*proof*⟩

9.2.5 Unwinding proof

definition Δ*0* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*0 s vl s1 vl1* ≡ *BC vl vl1* ∧ *eqButUID s s1* ∧ *validValSeqFrom vl1 s1*

lemma *istate-Δ0*:
assumes *B*: *B vl vl1*
shows *Δ0 istate vl istate vl1*
⟨*proof*⟩

lemma *friendsOfUID-cong*:
assumes *recvOuterFriendIDs s = recvOuterFriendIDs s'*
shows *friendsOfUID s = friendsOfUID s'*
⟨*proof*⟩

lemma *friendsOfUID-step-not-UID*:
assumes *uid ≠ UID ∨ aid ≠ AID ∨ uid' ∈ UIDs AID'*
shows *friendsOfUID (receiveCreateOFriend s aid sp uid uid') = friendsOfUID s*
and *friendsOfUID (receiveDeleteOFriend s aid sp uid uid') = friendsOfUID s*
⟨*proof*⟩

lemma *friendsOfUID-step-Create-UID*:
assumes *uid' ∉ UIDs AID'*
shows *friendsOfUID (receiveCreateOFriend s AID sp UID uid') = insert uid' (friendsOfUID s)*
⟨*proof*⟩

lemma *friendsOfUID-step-Delete-UID*:
assumes *e-receiveDeleteOFriend s AID sp UID uid'*
and *rs*: *reach s*
shows *friendsOfUID (receiveDeleteOFriend s AID sp UID uid') = friendsOfUID s − {uid'}*
⟨*proof*⟩

lemma *step-validValSeqFrom*:
assumes *step*: *step s a = (ou, s')*
and *rs*: *reach s*
and *c*: *consume (Trans s a ou s') vl vl'* (**is** *consume ?trn vl vl'*)
and *vVS*: *validValSeqFrom vl s*
shows *validValSeqFrom vl' s'*
⟨*proof*⟩

lemma *unwind-cont-Δ0*: *unwind-cont Δ0 {Δ0}*
⟨*proof*⟩

definition *Gr* **where**
Gr =
{
(*Δ0*, {*Δ0*})
}

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 φ :: *apiID* ⇒ (*state*, *act*, *out*) *trans* ⇒ *bool*
where φ *aid trn* ≡ (*if aid* = *AID then Issuer.*φ *trn else OuterFriendReceiver.*φ *UIDs AID UID aid trn*)

abbreviation *f* :: *apiID* ⇒ (*state*, *act*, *out*) *trans* ⇒ *value*
where *f aid trn* ≡ (*if aid* = *AID then Issuer.f trn else OuterFriendReceiver.f aid trn*)

abbreviation γ :: *apiID* ⇒ (*state*, *act*, *out*) *trans* ⇒ *bool*
where γ *aid trn* ≡ (*if aid* = *AID then Issuer.*γ *trn else OuterFriendReceiver.*γ *UIDs aid trn*)

abbreviation *g* :: *apiID* ⇒ (*state*, *act*, *out*) *trans* ⇒ *obs*
where *g aid trn* ≡ (*if aid* = *AID then Issuer.g trn else OuterFriendReceiver.g UIDs AID UID aid trn*)

abbreviation *T* :: *apiID* ⇒ (*state*, *act*, *out*) *trans* ⇒ *bool*
where *T aid trn* ≡ *False*

abbreviation *B* :: *apiID* ⇒ *value list* ⇒ *value list* ⇒ *bool*
where *B aid vl vl1* ≡ (*if aid* = *AID then Issuer.B vl vl1 else OuterFriendReceiver.B UIDs AID UID aid vl vl1*)

fun *comOfV* **where**

comOfV aid (FrVal aid' uid' st) = (if aid ≠ AID then Recv else (if aid' ≠ 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 ≡ (v1 = v2)*

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* = *id*
⟨*proof*⟩

context
fixes *AID'* :: *apiID*
assumes *AID'*: *AID'* ∈ *AIDs* − {*AID*}
begin

interpretation *Receiver*: *OuterFriendReceiver UIDs AID UID AID'* ⟨*proof*⟩

lemma *Issuer-BC-Receiver-BC*:
assumes *Issuer.BC vl vl1*
shows *Receiver.BC* (*Net.projectSrcV AID' vl*) (*Net.projectSrcV AID' vl1*)
⟨*proof*⟩

lemma *Collect-setminus*: *Collect P − A* = {*u*. *u* ∉ *A* ∧ *P u*}
⟨*proof*⟩

lemma *Issuer-vVS-Receiver-vVS*:
assumes *Issuer.validValSeq vl auidl*
shows *Receiver.validValSeq* (*Net.projectSrcV AID' vl*) {*uid*. (*AID'*,*uid*) ∈∈ *auidl*}
⟨*proof*⟩

lemma *Issuer-B-Receiver-B*:
assumes *Issuer.B vl vl1*
shows *Receiver.B* (*Net.projectSrcV AID' vl*) (*Net.projectSrcV AID' vl1*)
⟨*proof*⟩

end

sublocale *BD-Security-TS-Network-Preserve-Source-Security-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* = *id*
⟨*proof*⟩

theorem *secure*: *secure*
⟨*proof*⟩

end

end
theory *Outer-Friend-All*
imports *Outer-Friend-Network*
begin

end

References

[1] The Diaspora project. https://diasporafoundation.org/, 2021.

[2] T. Bauereiß, 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. Bauereiß, 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. Bauereiß, 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.