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*

unfolding *emptyPost-def* **by** *simp-all*

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*

by(*cases ntc*, *auto*)+

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*

by (*cases pst*, *auto*)+

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*
by (*cases apiID*) *auto*

lemma *member-apiIDAsStr-iff* [*simp*]: *str* ∈ *apiIDAsStr* ' *apiIDs* ⟷ *Aid str* ∈ *apiIDs*
by (*metis ApiID-apiIDAsStr image-iff apiIDAsStr.simps*)

lemma *getFreshApiID*: ¬ *getFreshApiID apiIDs* ∈∈ *apiIDs*
using *fresh-notIn*[*of set* (*map apiIDAsStr apiIDs*)] **unfolding** *getFreshApiID-def*
by *auto*

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*
by (*cases userID*) *auto*

lemma *member-userIDAsStr-iff* [*simp*]: *str* ∈ *userIDAsStr* ' (*set userIDs*) ⟷ *Uid str* ∈∈ *userIDs*
by (*metis UserID-userIDAsStr image-iff userIDAsStr.simps*)

lemma *getFreshUserID*: ¬ *getFreshUserID userIDs* ∈∈ *userIDs*
using *fresh-notIn*[*of set* (*map userIDAsStr userIDs*)] **unfolding** *getFreshUserID-def*
by *auto*

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*
by (*cases postID*) *auto*

lemma *member-postIDAsStr-iff* [*simp*]: *str* ∈ *postIDAsStr* ' (*set postIDs*) ⟷ *Pid str* ∈∈ *postIDs*
by (*metis PostID-postIDAsStr image-iff postIDAsStr.simps*)

lemma *getFreshPostID*: ¬ *getFreshPostID postIDs* ∈∈ *postIDs*
using *fresh-notIn*[*of set* (*map postIDAsStr postIDs*)] **unfolding** *getFreshPostID-def*
by *auto*

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*. []),

$recvOuterFriendIDs = (\lambda\ uID.\ [])$,

$postIDs = []$,
$post = (\lambda\ papID.\ emptyPost)$,
$owner = (\lambda\ pID.\ emptyUserID)$,
$vis = (\lambda\ pID.\ FriendV)$,

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

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

3.2.2 Starting action

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

3.2.3 Creation actions

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

definition *e-createNUReq* :: *state* ⇒ *userID* ⇒ *requestInfo* ⇒ *bool*
where
$e\text{-}createNUReq\ s\ uID\ requestInfo \equiv$

admin s ∈∈ *userIDs s* ∧ ¬ *uID* ∈∈ *userIDs s* ∧ ¬ *uID* ∈∈ *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*])] [] ∧ *pass s uID* = *p* ∧
(*admin s* = *uID* ∨ (*aID*,*outerOwner s aID pID*) ∈∈ *recvOuterFriendIDs s uID* ∨
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*])] [] ∧ *pass s uID* = *p* ∧
(*admin s* = *uID* ∨ (*aID*,*outerOwner s aID pID*) ∈∈ *recvOuterFriendIDs s uID* ∨
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*])] [] ∧ *pass s uID* = *p* ∧
(*admin s* = *uID* ∨ (*aID*,*outerOwner s aID pID*) ∈∈ *recvOuterFriendIDs s uID* ∨
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'*] [] [] [] ∧ *pass s uID* = *p* ∧
uID' ∈∈ *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

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

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

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

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

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

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

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

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

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

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*) ∈∈ *recvOuter-FriendIDs 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 ≡
¬ *aID* ∈∈ *pendingCApiReqs s* ∧ *admin s* ∈∈ *userIDs s*

definition *connectClient* :: *state* ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *password* ⇒ (*apiID* × *password*) × *state*
where
connectClient s uID p aID cp ≡
((*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* ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *password* ⇒ *bool*
where
e-connectClient s uID p aID cp ≡
IDsOK s [*uID*] [] [] [] ∧ *pass s uID* = *p* ∧
uID = *admin s* ∧
aID ∈∈ *pendingCApiReqs s* ∧ ¬ *aID* ∈∈ *clientApiIDs s*

definition *connectServer* :: *state* ⇒ *apiID* ⇒ *password* ⇒ *state*
where
connectServer s aID sp ≡
s (|*serverApiIDs* := (*aID* # *serverApiIDs s*),
serverPass := (*serverPass s*) (*aID* := *sp*),
pendingSApiReqs := *remove1 aID* (*pendingSApiReqs s*),
sApiReq := (*sApiReq s*)(*aID* := *emptyRequestInfo*)|)

definition *e-connectServer* :: *state* ⇒ *apiID* ⇒ *password* ⇒ *bool*
where
e-connectServer s aID sp ≡
aID ∈∈ *pendingSApiReqs s* ∧ ¬ *aID* ∈∈ *serverApiIDs s*

definition *sendPost* ::
state ⇒ *userID* ⇒ *password* ⇒ *apiID* ⇒ *postID* ⇒ (*apiID* × *password* × *postID* × *post* × *userID* × *vis*) × *state*
where
sendPost s uID p aID pID ≡

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

definition *e-sendPost* :: $state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow postID \Rightarrow bool$
where
$e\text{-}sendPost\ s\ uID\ p\ aID\ pID \equiv$
$IDsOK\ s\ [uID]\ [pID]\ []\ [aID] \wedge pass\ s\ uID = p \wedge$
$uID = admin\ s \wedge aID \in\in clientApiIDs\ s$

definition *receivePost* :: $state \Rightarrow apiID \Rightarrow password \Rightarrow postID \Rightarrow post \Rightarrow userID \Rightarrow vis \Rightarrow state$
where
$receivePost\ s\ aID\ sp\ pID\ pst\ uID\ vs \equiv$
$let\ opIDs = outerPostIDs\ s\ in$
$s\ (\!|outerPostIDs := opIDs\ (aID := List.insert\ pID\ (opIDs\ aID)),$
$outerPost := fun\text{-}upd2\ (outerPost\ s)\ aID\ pID\ pst,$
$outerOwner := fun\text{-}upd2\ (outerOwner\ s)\ aID\ pID\ uID,$
$outerVis := fun\text{-}upd2\ (outerVis\ s)\ aID\ pID\ vs|\!)$

definition *e-receivePost* :: $state \Rightarrow apiID \Rightarrow password \Rightarrow postID \Rightarrow post \Rightarrow userID \Rightarrow vis \Rightarrow bool$
where
$e\text{-}receivePost\ s\ aID\ sp\ pID\ nt\ uID\ vs \equiv$
$IDsOK\ s\ []\ []\ [(aID,[])]\ [] \wedge serverPass\ s\ aID = sp$

definition *sendCreateOFriend* ::
$state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow userID \Rightarrow (apiID \times password \times userID \times userID) \times state$
where
$sendCreateOFriend\ s\ uID\ p\ aID\ uID' \equiv$
$let\ ofr = sentOuterFriendIDs\ s\ in$
$((aID,\ clientPass\ s\ aID,\ uID,\ uID'),$
$s\ (\!|sentOuterFriendIDs := ofr\ (uID := ofr\ uID\ @\ [(aID,uID')])|\!))$

definition *e-sendCreateOFriend* :: $state \Rightarrow userID \Rightarrow password \Rightarrow apiID \Rightarrow userID \Rightarrow bool$
where
$e\text{-}sendCreateOFriend\ s\ uID\ p\ caID\ uID' \equiv$
$IDsOK\ s\ [uID]\ []\ []\ [caID] \wedge pass\ s\ uID = p \wedge$
$\neg\ (caID,uID') \in\in sentOuterFriendIDs\ s\ uID$

definition *receiveCreateOFriend* :: *state* ⇒ *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 outErr*)
|
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*
done

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*)

using *assms* **by** (*cases* (*aid1*,*obs1*,*aid2*,*obs2*) *rule*: *syncO.cases*) *auto*

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′*
using *assms*
by (*cases trn1*; *cases trn2*) (*auto elim*!: *syncO-cases simp*: *com-defs split*: *if-splits*)

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 (*unfold-locales*, *goal-cases*)
case (*1*) **show** *?case* **using** *finite-AIDs* **. next**
case (*2 aid trn*)
from *2* **show** *?case*
by (*cases* (*aid*, *trn*) *rule*: *tgtNodeOf.cases*) *auto*
qed

end

end
theory *Automation-Setup*
imports *System-Specification*
begin

lemma *add-prop*:
assumes *PROP* (*T*)
shows *A* ==> *PROP* (*T*)
using *assms* **.**

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*
using *assms* **apply** (*cases s*, *cases s1*) **by** *auto*

end

5 Safety properties

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

theory *Safety-Properties*
imports
Automation-Setup
begin

declare *Let-def*[*simp*]
declare *if-splits*[*split*]
declare *IDsOK-def*[*simp*]

lemmas *eff-defs* = *s-defs c-defs d-defs u-defs*
lemmas *obs-defs* = *r-defs l-defs*
lemmas *effc-defs* = *eff-defs com-defs*
lemmas *all-defs* = *effc-defs obs-defs*

declare *sstep-Cons*[*simp*]

lemma *Lact-Ract-noStateChange*[*simp*]:
assumes $a \in$ *Lact* ‘ *UNIV* $\cup$ *Ract* ‘ *UNIV*
shows *snd* (*step s a*) = *s*
using *assms* **by** (*cases a*) *auto*

lemma *Lact-Ract-noStateChange-set*:
assumes *set al* $\subseteq$ *Lact* ‘ *UNIV* $\cup$ *Ract* ‘ *UNIV*
shows *snd* (*sstep s al*) = *s*
using *assms* **by** (*induct al*) (*auto split*: *prod.splits*)

lemma *reach-postIDs-persist*:
pID $\in\in$ *postIDs s* $\Longrightarrow$ *step s a* = (*ou*,*s'*) $\Longrightarrow$ *pID* $\in\in$ *postIDs s'*
apply (*cases a*)
  subgoal for *x1* **apply**(*cases x1*, *auto simp*: *effc-defs*) **.**
  subgoal for *x2* **apply**(*cases x2*, *auto simp*: *effc-defs*) **.**
  subgoal for *x3* **apply**(*cases x3*, *auto simp*: *effc-defs*) **.**
  subgoal for *x4* **apply**(*cases x4*, *auto simp*: *effc-defs*) **.**
  subgoal by *auto*
  subgoal by *auto*
  subgoal for *x7* **apply**(*cases x7*, *auto simp*: *effc-defs*) **.**
done

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*
apply (*cases a*)
  subgoal for *x1* **apply**(*cases x1*, *auto simp*: *all-defs*) **.**
  subgoal for *x2* **apply**(*cases x2*, *auto simp*: *all-defs*) **.**
  subgoal for *x3* **apply**(*cases x3*, *auto simp*: *all-defs*) **.**
  subgoal for *x4* **apply**(*cases x4*, *auto simp*: *all-defs*) **.**
  subgoal for *x5* **apply**(*cases x5*, *auto simp*: *all-defs*) **.**
  subgoal for *x6* **apply**(*cases x6*, *auto simp*: *all-defs*) **.**
  subgoal for *x7* **apply**(*cases x7*, *auto simp*: *all-defs*) **.**
done

lemma *reach-vis*: *reach s* $\Longrightarrow$ *vis s pID* $\in$ {*FriendV*, *PublicV*}
proof (*induction rule*: *reach-step-induct*)
  case (*Step s a*) **then show** *?case* **proof** (*cases a*)
     case (*Sact sAct*) **with** *Step* **show** *?thesis*
     apply (*cases sAct*) **by** (*auto simp add*: *s-defs*)
  next
    case (*Cact cAct*) **with** *Step* **show** *?thesis*
    apply (*cases cAct*) **by** (*auto simp add*: *c-defs*)
  next
    case (*Dact dAct*) **with** *Step* **show** *?thesis*

apply (*cases dAct*) **by** (*auto simp add*: *d-defs*)
next
case (*Uact uAct*) **with** *Step* **show** *?thesis*
apply (*cases uAct*) **by** (*auto simp add*: *u-defs*)
next
case (*COMact comAct*) **with** *Step* **show** *?thesis* **apply** (*cases comAct*)
by (*auto simp add*: *com-defs*)
qed *auto*
qed (*auto simp add*: *istate-def*)

lemma *reach-not-postIDs-emptyPost*:
reach s $\Longrightarrow$ *PID* $\notin$ *set* (*postIDs s*) $\Longrightarrow$ *post s PID* = *emptyPost*
proof (*induction rule*: *reach-step-induct*)
case (*Step s a*) **then show** *?case* **proof** (*cases a*)
case (*Sact sAct*) **with** *Step* **show** *?thesis*
apply (*cases sAct*) **by** (*auto simp add*: *s-defs*)
next
case (*Cact cAct*) **with** *Step* **show** *?thesis*
apply (*cases cAct*) **by** (*auto simp add*: *c-defs*)
next
case (*Dact dAct*) **with** *Step* **show** *?thesis*
apply (*cases dAct*) **by** (*auto simp add*: *d-defs*)
next
case (*Uact uAct*) **with** *Step* **show** *?thesis*
apply (*cases uAct*) **by** (*auto simp add*: *u-defs*)
next
case (*COMact comAct*) **with** *Step* **show** *?thesis* **apply** (*cases comAct*)
by (*auto simp add*: *com-defs*)
qed *auto*
qed (*auto simp add*: *istate-def*)

lemma *reach-not-postIDs-friendV*:
reach s $\Longrightarrow$ *PID* $\notin$ *set* (*postIDs s*) $\Longrightarrow$ *vis s PID* = *FriendV*
proof (*induction rule*: *reach-step-induct*)
case (*Step s a*) **then show** *?case* **proof** (*cases a*)
case (*Sact sAct*) **with** *Step* **show** *?thesis*
apply (*cases sAct*) **by** (*auto simp add*: *s-defs*)
next
case (*Cact cAct*) **with** *Step* **show** *?thesis*
apply (*cases cAct*) **by** (*auto simp add*: *c-defs*)
next
case (*Dact dAct*) **with** *Step* **show** *?thesis*
apply (*cases dAct*) **by** (*auto simp add*: *d-defs*)
next
case (*Uact uAct*) **with** *Step* **show** *?thesis*
apply (*cases uAct*) **by** (*auto simp add*: *u-defs*)
next
case (*COMact comAct*) **with** *Step* **show** *?thesis* **apply** (*cases comAct*)
by (*auto simp add*: *com-defs*)

  qed *auto*
qed (*auto simp add*: *istate-def*)

lemma *reach-owner-userIDs*: *reach s* ⟹ *pID* ∈∈ *postIDs s* ⟹ *owner s pID* ∈∈ *userIDs s*
proof (*induction rule*: *reach-step-induct*)
  case (*Step s a*) **then show** *?case* **proof** (*cases a*)
    case (*Sact sAct*) **with** *Step* **show** *?thesis*
    apply (*cases sAct*) **by** (*auto simp add*: *s-defs*)
  next
    case (*Cact cAct*) **with** *Step* **show** *?thesis*
    apply (*cases cAct*) **by** (*auto simp add*: *c-defs*)
  next
    case (*Dact dAct*) **with** *Step* **show** *?thesis*
    apply (*cases dAct*) **by** (*auto simp add*: *d-defs*)
  next
    case (*Uact uAct*) **with** *Step* **show** *?thesis*
    apply (*cases uAct*) **by** (*auto simp add*: *u-defs*)
  next
    case (*COMact comAct*) **with** *Step* **show** *?thesis* **apply** (*cases comAct*)
    by (*auto simp add*: *com-defs*)
  qed *auto*
qed (*auto simp add*: *istate-def*)

lemma *reach-admin-userIDs*: *reach s* ⟹ *uID* ∈∈ *userIDs s* ⟹ *admin s* ∈∈ *userIDs s*
proof (*induction rule*: *reach-step-induct*)
  case (*Step s a*) **then show** *?case* **proof** (*cases a*)
    case (*Sact sAct*) **with** *Step* **show** *?thesis*
    apply (*cases sAct*) **by** (*auto simp add*: *s-defs*)
  next
    case (*Cact cAct*) **with** *Step* **show** *?thesis*
    apply (*cases cAct*) **by** (*auto simp add*: *c-defs*)
  next
    case (*Dact dAct*) **with** *Step* **show** *?thesis*
    apply (*cases dAct*) **by** (*auto simp add*: *d-defs*)
  next
    case (*Uact uAct*) **with** *Step* **show** *?thesis*
    apply (*cases uAct*) **by** (*auto simp add*: *u-defs*)
  next
    case (*COMact comAct*) **with** *Step* **show** *?thesis* **apply** (*cases comAct*)
    by (*auto simp add*: *com-defs*)
  qed *auto*
qed (*auto simp add*: *istate-def*)

lemma *reach-pendingUReqs-distinct*: *reach s* ⟹ *distinct* (*pendingUReqs s*)

proof (*induction rule*: *reach-step-induct*)
case (*Step s a*) **then show** *?case* **proof** (*cases a*)
case (*Sact sAct*) **with** *Step* **show** *?thesis* **by** (*cases sAct*) (*auto simp add*: *s-defs*) **next**
case (*Cact cAct*) **with** *Step* **show** *?thesis* **by** (*cases cAct*) (*auto simp add*: *c-defs*) **next**
case (*Dact dAct*) **with** *Step* **show** *?thesis* **by** (*cases dAct*) (*auto simp add*: *d-defs*) **next**
case (*Uact uAct*) **with** *Step* **show** *?thesis* **by** (*cases uAct*) (*auto simp add*: *u-defs*) **next**
case (*COMact comAct*) **with** *Step* **show** *?thesis* **by** (*cases comAct*) (*auto simp add*: *com-defs*)
qed *auto*
qed (*auto simp*: *istate-def*)

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 (*induction rule*: *reach-step-induct*)
case (*Step s a*) **then show** *?case* **proof** (*cases a*)
case (*Sact sAct*) **with** *Step* **show** *?thesis* **by** (*cases sAct*) (*auto simp add*: *s-defs*) **next**
case (*Cact cAct*)
with *Step reach-pendingUReqs-distinct* **show** *?thesis*
by (*cases cAct*) (*auto simp add*: *c-defs*) **next**
case (*Dact dAct*) **with** *Step* **show** *?thesis* **by** (*cases dAct*) (*auto simp add*: *d-defs*) **next**
case (*Uact uAct*) **with** *Step* **show** *?thesis* **by** (*cases uAct*) (*auto simp add*: *u-defs*) **next**
case (*COMact comAct*) **with** *Step* **show** *?thesis* **by** (*cases comAct*) (*auto simp add*: *com-defs*)
qed *auto*
qed (*auto simp*: *istate-def*)

lemma *reach-friendIDs-symmetric*:
reach s $\Longrightarrow$ *uID1* $\in\in$ *friendIDs s uID2* $\longleftrightarrow$ *uID2* $\in\in$ *friendIDs s uID1*
proof (*induction rule*: *reach-step-induct*)
case (*Step s a*) **then show** *?case* **proof** (*cases a*)
case (*Sact sAct*) **with** *Step* **show** *?thesis* **by** (*cases sAct*) (*auto simp add*: *s-defs*) **next**
case (*Cact cAct*) **with** *Step* **show** *?thesis* **by** (*cases cAct*) (*auto simp add*: *c-defs*) **next**
case (*Dact dAct*) **with** *Step* **show** *?thesis* **by** (*cases dAct*) (*auto simp add*: *d-defs*) **next**
case (*Uact uAct*) **with** *Step* **show** *?thesis* **by** (*cases uAct*) (*auto simp add*: *u-defs*) **next**
case (*COMact comAct*) **with** *Step* **show** *?thesis* **by** (*cases comAct*) (*auto simp add*: *com-defs*)
qed *auto*

qed (*auto simp add*: *istate-def*)

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′* ∈∈ *pendingFReqs s uid* ⟹ *uid′* ∉ *set* (*friendIDs s uid*)
and *uid′* ∈∈ *pendingFReqs s uid* ⟹ *uid* ∉ *set* (*friendIDs s uid′*)
using *assms* **proof** (*induction arbitrary*: *uid uid′ rule*: *reach-step-induct*)
case *Istate*
fix *uid uid′*
show *distinct* (*friendIDs istate uid*) **and** *distinct* (*pendingFReqs istate uid*)
and *distinct* (*sentOuterFriendIDs istate uid*) **and** *distinct* (*recvOuterFriendIDs istate uid*)
and *uid′* ∈∈ *pendingFReqs istate uid* ⟹ *uid′* ∉ *set* (*friendIDs istate uid*)
and *uid′* ∈∈ *pendingFReqs istate uid* ⟹ *uid* ∉ *set* (*friendIDs istate uid′*)
unfolding *istate-def* **by** *auto*
next
case (*Step s a*)
have *s′*: *reach* (*snd* (*step s a*)) **using** *reach-step*[*OF Step*(*1*)] **.**
{ fix *uid uid′*
have *distinct* (*friendIDs* (*snd* (*step s a*)) *uid*) ∧ *distinct* (*pendingFReqs* (*snd* (*step s a*)) *uid*)
∧ *distinct* (*sentOuterFriendIDs* (*snd* (*step s a*)) *uid*)
∧ *distinct* (*recvOuterFriendIDs* (*snd* (*step s a*)) *uid*)
∧ (*uid′* ∈∈ *pendingFReqs* (*snd* (*step s a*)) *uid* ⟶ *uid′* ∉ *set* (*friendIDs* (*snd* (*step s a*)) *uid*))
proof (*cases a*)
case (*Sact sa*) **with** *Step* **show** *?thesis* **by** (*cases sa*) (*auto simp add*: *s-defs*)
next
case (*Cact ca*) **with** *Step* **show** *?thesis* **by** (*cases ca*) (*auto simp add*: *c-defs*)
next
case (*Dact da*) **with** *Step* **show** *?thesis* **by** (*cases da*) (*auto simp add*: *d-defs distinct-removeAll*) **next**
case (*Uact ua*) **with** *Step* **show** *?thesis* **by** (*cases ua*) (*auto simp add*: *u-defs*) **next**
case (*Ract ra*) **with** *Step* **show** *?thesis* **by** *auto* **next**
case (*Lact ra*) **with** *Step* **show** *?thesis* **by** *auto* **next**
case (*COMact ca*) **with** *Step* **show** *?thesis* **by** (*cases ca*) (*auto simp add*: *com-defs*) **next**
qed
} note *goal* = *this*
fix *uid uid′*
from *goal* **show** *distinct* (*friendIDs* (*snd* (*step s a*)) *uid*)
and *distinct* (*pendingFReqs* (*snd* (*step s a*)) *uid*)
and *distinct* (*sentOuterFriendIDs* (*snd* (*step s a*)) *uid*)
and *distinct* (*recvOuterFriendIDs* (*snd* (*step s a*)) *uid*)

by *auto*
  assume *uid'* ∈∈ *pendingFReqs* (*snd* (*step s a*)) *uid*
  with *goal* **show** *uid'* ∉ *set* (*friendIDs* (*snd* (*step s a*)) *uid*) **by** *auto*
  then show *uid* ∉ *set* (*friendIDs* (*snd* (*step s a*)) *uid'*)
   using *reach-friendIDs-symmetric*[*OF s'*] **by** *simp*
qed

lemma *remove1-in-set*: *x* ∈∈ *remove1 y xs* ⟹ *x* ∈∈ *xs*
by (*induction xs*) *auto*

lemma *reach-IDs-used-IDsOK*[*rule-format*]:
assumes *reach s*
shows *uid* ∈∈ *pendingFReqs s uid'* ⟶ *IDsOK s* [*uid*, *uid'*] [] [] [] (**is** *?p*)
and *uid* ∈∈ *friendIDs s uid'* ⟶ *IDsOK s* [*uid*, *uid'*] [] [] [] (**is** *?f*)
using *assms* **proof** −
 from *assms* **have** *uid* ∈∈ *pendingFReqs s uid'* ∨ *uid* ∈∈ *friendIDs s uid'*
   ⟶ *IDsOK s* [*uid*, *uid'*] [] [] []
 proof (*induction rule*: *reach-step-induct*)
  case *Istate* **then show** *?case* **by** (*auto simp add*: *istate-def*)
 next
  case (*Step s a*) **then show** *?case* **proof** (*cases a*)
   case (*Sact sa*) **with** *Step* **show** *?thesis* **by** (*cases sa*) (*auto simp*: *s-defs*) **next**
   case (*Cact ca*) **with** *Step* **show** *?thesis* **by** (*cases ca*) (*auto simp*: *c-defs intro*: *remove1-in-set*) **next**
    case (*Dact da*) **with** *Step* **show** *?thesis* **by** (*cases da*) (*auto simp*: *d-defs*) **next**
    case (*Uact ua*) **with** *Step* **show** *?thesis* **by** (*cases ua*) (*auto simp*: *u-defs*) **next**
   case (*COMact ca*) **with** *Step* **show** *?thesis* **by** (*cases ca*) (*auto simp*: *com-defs*)
  qed *auto*
 qed
 then show *?p* **and** *?f* **by** *auto*
qed

lemma *reach-AID-used-valid*:
assumes *reach s*
and *aid* ∈∈ *serverApiIDs s* ∨ *aid* ∈∈ *clientApiIDs s* ∨ *aid* ∈∈ *pendingSApiReqs s* ∨ *aid* ∈∈ *pendingCApiReqs s*
shows *admin s* ∈∈ *userIDs s*
using *assms* **proof** (*induction rule*: *reach-step-induct*)
 case *Istate* **then show** *?case* **by** (*auto simp*: *istate-def*)
next
 case (*Step s a*) **then show** *?case* **proof** (*cases a*)
  case (*Sact sa*) **with** *Step* **show** *?thesis* **by** (*cases sa*) (*auto simp*: *s-defs*) **next**
  case (*Cact ca*) **with** *Step* **show** *?thesis* **by** (*cases ca*) (*auto simp*: *c-defs*) **next**
  case (*Dact da*) **with** *Step* **show** *?thesis* **by** (*cases da*) (*auto simp*: *d-defs*) **next**
  case (*Uact ua*) **with** *Step* **show** *?thesis* **by** (*cases ua*) (*auto simp*: *u-defs*) **next**
  case (*COMact ca*) **with** *Step* **show** *?thesis* **by** (*cases ca*) (*auto simp*: *com-defs intro*: *remove1-in-set*)

qed *auto*
qed

lemma *IDs-mono*[*rule-format*]:
assumes *step s a* = (*ou*, *s'*)
shows *uid* ∈∈ *userIDs s* ⟶ *uid* ∈∈ *userIDs s'* (**is** *?u*)
and *nid* ∈∈ *postIDs s* ⟶ *nid* ∈∈ *postIDs s'* (**is** *?n*)
and *aid* ∈∈ *clientApiIDs s* ⟶ *aid* ∈∈ *clientApiIDs s'* (**is** *?c*)
and *sid* ∈∈ *serverApiIDs s* ⟶ *sid* ∈∈ *serverApiIDs s'* (**is** *?s*)
and *nid* ∈∈ *outerPostIDs s aid* ⟶ *nid* ∈∈ *outerPostIDs s' aid* (**is** *?o*)
proof −
from *assms* **have** *?u* ∧ *?n* ∧ *?c* ∧ *?s* ∧ *?o* **proof** (*cases a*)
case (*Sact sa*) **with** *assms* **show** *?thesis* **by** (*cases sa*) (*auto simp add*: *s-defs*)
next
case (*Cact ca*) **with** *assms* **show** *?thesis* **by** (*cases ca*) (*auto simp add*: *c-defs*)
next
case (*Dact da*) **with** *assms* **show** *?thesis* **by** (*cases da*) (*auto simp add*: *d-defs*)
next
case (*Uact ua*) **with** *assms* **show** *?thesis* **by** (*cases ua*) (*auto simp add*: *u-defs*)
next
case (*COMact ca*) **with** *assms* **show** *?thesis* **by** (*cases ca*) (*auto simp add*: *com-defs*)
qed (*auto*)
then show *?u ?n ?c ?s ?o* **by** *auto*
qed

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*
using *IDs-mono*[*OF assms*(*1*)] *assms*(*2*)
by (*auto simp add*: *list-all-iff*)

lemma *step-outerFriendIDs-idem*:
assumes *step s a* = (*ou*, *s'*)
and ∀ *uID p aID uID'*. *a* ≠ *COMact* (*comSendCreateOFriend uID p aID uID'*) ∧
a ≠ *COMact* (*comReceiveCreateOFriend aID p uID uID'*) ∧
a ≠ *COMact* (*comSendDeleteOFriend uID p aID uID'*) ∧
a ≠ *COMact* (*comReceiveDeleteOFriend aID p uID uID'*)
shows *sentOuterFriendIDs s'* = *sentOuterFriendIDs s* (**is** *?sent*)
and *recvOuterFriendIDs s'* = *recvOuterFriendIDs s* (**is** *?recv*)
proof −
have *?sent* ∧ *?recv* **using** *assms* **proof** (*cases a*)
case (*Sact sa*) **with** *assms* **show** *?thesis* **by** (*cases sa*) (*auto simp add*: *s-defs*)
next
case (*Cact ca*) **with** *assms* **show** *?thesis* **by** (*cases ca*) (*auto simp add*: *c-defs*)
next
case (*Dact da*) **with** *assms* **show** *?thesis* **by** (*cases da*) (*auto simp add*: *d-defs*)

next
  case (*Uact ua*) **with** *assms* **show** *?thesis* **by** (*cases ua*) (*auto simp add*: *u-defs*)
next
  case (*COMact ca*) **with** *assms* **show** *?thesis* **by** (*cases ca*) (*auto simp add*: *com-defs*)
 qed *auto*
 then show *?sent* **and** *?recv* **by** *auto*
qed

lemma *istate-sSys*:
assumes *step istate a* = (*ou*, *s′*)
obtains *uid p* **where** *a* = *Sact* (*sSys uid p*)
  | *s′* = *istate*
using *assms* **proof** (*cases a*)
 case (*Sact sa*) **with** *assms* **show** *?thesis* **by** (*cases sa*) (*auto intro*: *that*) **next**
 case (*Cact ca*) **with** *assms that*(*2*) **show** *?thesis* **by** (*cases ca*) (*auto simp add*: *c-defs istate-def*) **next**
 case (*Dact da*) **with** *assms that*(*2*) **show** *?thesis* **by** (*cases da*) (*auto simp add*: *d-defs istate-def*) **next**
 case (*Uact ua*) **with** *assms that*(*2*) **show** *?thesis* **by** (*cases ua*) (*auto simp add*: *u-defs istate-def*) **next**
 case (*COMact ca*) **with** *assms that*(*2*) **show** *?thesis* **by** (*cases ca*) (*auto simp add*: *com-defs istate-def*) **next**
 case (*Ract ra*) **with** *assms that*(*2*) **show** *?thesis* **by** (*cases ra*) (*auto simp add*: *r-defs istate-def*) **next**
 case (*Lact la*) **with** *assms that*(*2*) **show** *?thesis* **by** (*cases la*) (*auto simp add*: *l-defs istate-def*)
qed

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 (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.

[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*.

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*)
∨
(∃ *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* = *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'*
by (*cases ca*; *auto*)+

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*
by (*cases ou*; *auto simp*: *ObservationSetup-ISSUER.outPurge.simps*)+

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*) ∧ *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*) ∧ *ou* = *outOK*
g (*Trans s a ou s'*) = (*COMact* (*comSendPost uID p aID nID*), *O-sendPost* (*aid*, *sp*, *pid*, *pst'*, *uid*, *vs*))
$\longleftrightarrow$ (∃ *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'*))
$\longleftrightarrow$ (∃ *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*)
$\longleftrightarrow$ *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'*))
$\longleftrightarrow$ (∃ *p'*. *a* = *COMact* (*comSendDeleteOFriend uID p' aID uID'*) ∧ *p* = *emptyPass* ∧ *ou* = *O-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'*) ∧ *ou* = *outOK*
by (*cases a*; *auto simp*: *comPurge-simps*)+

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*
unfolding *eeqButPID-def* **by** *auto*

lemma *eeqButPID-sym*:
assumes *eeqButPID psts psts1* **shows** *eeqButPID psts1 psts*
using *assms* **unfolding** *eeqButPID-def* **by** *auto*

lemma *eeqButPID-trans*:
assumes *eeqButPID psts psts1* **and** *eeqButPID psts1 psts2* **shows** *eeqButPID psts psts2*
using *assms* **unfolding** *eeqButPID-def* **by** (*auto split*: *if-splits*)

lemma *eeqButPID-cong*:
assumes *eeqButPID psts psts1*
and $pid = PID \Longrightarrow$ *eqButT uu uu1*
and $pid \neq PID \Longrightarrow uu = uu1$
shows *eeqButPID* (*psts* (*pid* := *uu*)) (*psts1*(*pid* := *uu1*))
using *assms* **unfolding** *eeqButPID-def* **by** (*auto split*: *if-splits*)

lemma *eeqButPID-not-PID*:
⟦*eeqButPID psts psts1*; $pid \neq PID$⟧ $\Longrightarrow$ *psts pid* = *psts1 pid*
unfolding *eeqButPID-def* **by** (*auto split*: *if-splits*)

lemma *eeqButPID-toEq*:
assumes *eeqButPID psts psts1*
shows *psts* (*PID* := *pid*) =
psts1 (*PID* := *pid*)
using *eeqButPID-not-PID*[*OF assms*] **by** *auto*

lemma *eeqButPID-update-post*:
assumes *eeqButPID psts psts1*
shows *eeqButPID* (*psts* (*pid* := *pst*)) (*psts1* (*pid* := *pst*))
using *eeqButPID-not-PID*[*OF assms*]
using *assms* **unfolding** *eeqButPID-def* **by** *auto*

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*
by *auto*

lemma *eqButF-sym*:
assumes *eqButF aID-bl aID-bl1*
shows *eqButF aID-bl1 aID-bl*
using *assms* **by** *auto*

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

using *assms* **by** *auto*

lemma *eqButF-insert2*:
eqButF aID-bl aID-bl1 ⟹
eqButF (*insert2 aID b aID-bl*) (*insert2 aID b aID-bl1*)
unfolding *insert2-def*
by *simp* (*smt map-eq-conv o-apply o-id prod.collapse prod.sel*(*1*) *split-conv*)

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*
unfolding *eeqButPID-F-def* **by** *auto*

lemma *eeqButPID-F-sym*:
assumes *eeqButPID-F sw sw1* **shows** *eeqButPID-F sw1 sw*
using *assms eqButF-sym* **unfolding** *eeqButPID-F-def*
by *presburger*

lemma *eeqButPID-F-trans*:
assumes *eeqButPID-F sw sw1* **and** *eeqButPID-F sw1 sw2* **shows** *eeqButPID-F sw sw2*
using *assms* **unfolding** *eeqButPID-F-def* **by** (*auto split*: *if-splits*)

lemma *eeqButPID-F-cong*:
assumes *eeqButPID-F sw sw1*
and *PID* = *PID* ⟹ *eqButF uu uu1*
and *pid* ≠ *PID* ⟹ *uu* = *uu1*
shows *eeqButPID-F* (*sw* (*pid* := *uu*)) (*sw1*(*pid* := *uu1*))
using *assms* **unfolding** *eeqButPID-F-def* **by** (*auto split*: *if-splits*)

lemma *eeqButPID-F-eqButF*:
eeqButPID-F sw sw1 ⟹ *eqButF* (*sw PID*) (*sw1 PID*)
unfolding *eeqButPID-F-def* **by** (*auto split*: *if-splits*)

lemma *eeqButPID-F-not-PID*:
⟦*eeqButPID-F sw sw1*; *pid* ≠ *PID*⟧ ⟹ *sw pid* = *sw1 pid*
unfolding *eeqButPID-F-def* **by** (*auto split*: *if-splits*)

lemma *eeqButPID-F-postSelectors*:
eeqButPID-F sw sw1 ⟹ *map fst* (*sw pid*) = *map fst* (*sw1 pid*)
unfolding *eeqButPID-F-def* **by** (*metis eqButF.simps*)

lemma *eeqButPID-F-insert2*:
eeqButPID-F sw sw1 $\Longrightarrow$
eqButF (*insert2 aID b* (*sw PID*)) (*insert2 aID b* (*sw1 PID*))
unfolding *eeqButPID-F-def* **using** *eqButF-insert2* **by** *metis*

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*))
using *length-map eeqButPID-F-eqButF*[*OF assms*] *eeqButPID-F-not-PID*[*OF assms*]
apply(*auto simp*: *o-def split-def map-replicate-const intro*!: *map-prod-cong ext*)
by (*metis length-map*)

lemma *eeqButPID-F-updateShared*:
assumes *eeqButPID-F sw sw1*
shows *eeqButPID-F* (*sw* (*pid* := *aID-b*)) (*sw1* (*pid* := *aID-b*))
using *eeqButPID-F-eqButF*[*OF assms*] *eeqButPID-F-not-PID*[*OF assms*]
using *assms* **unfolding** *eeqButPID-F-def* **by** *auto*

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* ∧
eqButPID (*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*
unfolding *eqButPID-def* **by** *auto*

lemma *eqButPID-sym*:
assumes *eqButPID s s1* **shows** *eqButPID s1 s*
using *assms eeqButPID-sym eeqButPID-F-sym* **unfolding** *eqButPID-def* **by** *auto*

lemma *eqButPID-trans*:
assumes *eqButPID s s1* **and** *eqButPID s1 s2* **shows** *eqButPID s s2*
using *assms eeqButPID-trans eeqButPID-F-trans* **unfolding** *eqButPID-def*
by *simp blast*

lemma *eqButPID-stateSelectors*:
eqButPID s s1 $\Longrightarrow$
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*
unfolding *eqButPID-def IDsOK-def*[*abs-def*] **by** *auto*

lemma *eqButPID-not-PID*:
eqButPID s s1 $\Longrightarrow$ *pid* ≠ *PID* $\Longrightarrow$ *post s pid* = *post s1 pid*
unfolding *eqButPID-def* **using** *eeqButPID-not-PID* **by** *auto*

lemma *eqButPID-eqButF*:
eqButPID s s1 $\Longrightarrow$ *eqButF* (*sharedWith s PID*) (*sharedWith s1 PID*)
unfolding *eqButPID-def* **using** *eeqButPID-F-eqButF* **by** *auto*

lemma *eqButPID-not-PID-sharedWith*:
eqButPID s s1 $\Longrightarrow$ *pid* $\neq$ *PID* $\Longrightarrow$ *sharedWith s pid* = *sharedWith s1 pid*
unfolding *eqButPID-def* **using** *eeqButPID-F-not-PID* **by** *auto*

lemma *eqButPID-insert2*:
eqButPID s s1 $\Longrightarrow$
eqButF (*insert2 aID b* (*sharedWith s PID*)) (*insert2 aID b* (*sharedWith s1 PID*))
unfolding *eqButPID-def* **using** *eeqButPID-F-insert2* **by** *metis*

lemma *eqButPID-actions*:
assumes *eqButPID s s1*
shows *listInnerPosts s uid p* = *listInnerPosts s1 uid p*
using *eqButPID-stateSelectors*[*OF assms*]
by (*auto simp*: *l-defs intro*!: *arg-cong2*[*of - - - - cmap*])

lemma *eqButPID-update*:
assumes *eqButPID s s1*
shows (*post s*)(*PID* := *txt*) = (*post s1*)(*PID* := *txt*)
using *assms* **unfolding** *eqButPID-def* **using** *eeqButPID-toEq* **by** *auto*

lemma *eqButPID-update-post*:
assumes *eqButPID s s1*
shows *eeqButPID* ((*post s*) (*pid* := *pst*)) ((*post s1*) (*pid* := *pst*))
using *assms* **unfolding** *eqButPID-def* **using** *eeqButPID-update-post* **by** *auto*

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*))
using *assms* **unfolding** *eqButPID-def* **using** *eeqButPID-F-toEq* **by** *auto*

lemma *eqButPID-updateShared*:
assumes *eqButPID s s1*
shows *eeqButPID-F* ((*sharedWith s*) (*pid* := *aID-b*)) ((*sharedWith s1*) (*pid* := *aID-b*))
using *assms* **unfolding** *eqButPID-def* **using** *eeqButPID-F-updateShared* **by** *auto*

lemma *eqButPID-cong*[*simp*]:
$\bigwedge$ *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* $\Longrightarrow$ *eeqButPID 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* $\Longrightarrow$ *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* (|*sharedWith* := *uu1*|)) (*s1* (|*sharedWith*:= *uu2*|))
unfolding *eqButPID-def* **by** *auto*

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 –
  note [*simp*] = *all-defs eeqButPID-F-def*
  note [*intro*!] = *eqButPID-cong*
  note ∗ = *step step1 ss1 eqButPID-stateSelectors*[*OF ss1*]
    eqButPID-update[*OF ss1*] *eqButPID-update-post*[*OF ss1*]
    eqButPID-setShared[*OF ss1*] *eqButPID-updateShared*[*OF ss1*]
    eqButPID-insert2[*OF ss1*]
  then show *?thesis*
  proof (*cases a*)
    case (*Sact x1*)
    with ∗ **show** *?thesis* **by** (*cases x1*) *auto*
  next
    case (*Cact x2*)
    with ∗ **show** *?thesis* **by** (*cases x2*) *auto*
  next
    case (*Dact x3*)
    with ∗ **show** *?thesis* **by** (*cases x3*) *auto*
  next
    case (*Uact x4*)
    with ∗ **show** *?thesis*
    proof (*cases x4*)
      case (*uPost x21 x22 x23 x24*)
      with *Uact* ∗ **show** *?thesis* **by** (*cases x23 = PID*) *auto*
    next
      case (*uVisPost x31 x32 x33 x34*)
      with *Uact* ∗ **show** *?thesis* **by** (*cases x33 = PID*) *auto*
    qed *auto*

next
  case (*COMact x7*)
  with ∗ **show** *?thesis*
  proof (*cases x7*)
    case (*comSendPost x61 x62 x63 x64*)
    with *COMact* ∗ **show** *?thesis* **by** (*cases x64 = PID*) *auto*
  qed *auto*
qed *auto*
qed

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(*induct vl*)
  case (*Cons v vl*)
  thus *?case* **by** (*cases v*) *auto*
qed *auto*

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*) ∨

($\exists$ *uid p aid*. *a* = *COMact* (*comSendPost uid p aid PID*) $\wedge$ *ou* $\neq$ *outErr*)
by (*cases Trans s a ou s′ rule*: *φ.cases*) *auto*

lemma *uPost-out*:
assumes *1*: *step s a* = (*ou*,*s′*) **and** *a*: *a* = *Uact* (*uPost uid p PID pst*) **and** *2*: *ou* = *outOK*
shows *uid* = *owner s PID* $\wedge$ *p* = *pass s uid*
using *1 2* **unfolding** *a* **by** (*auto simp*: *u-defs*)

lemma *comSendPost-out*:
assumes *1*: *step s a* = (*ou*,*s′*) **and** *a*: *a* = *COMact* (*comSendPost uid p aid PID*) **and** *2*: *ou* $\neq$ *outErr*
shows *ou* = *O-sendPost* (*aid*, *clientPass s aid*, *PID*, *post s PID*, *owner s PID*, *vis s PID*)
$\wedge$ *uid* = *admin s* $\wedge$ *p* = *pass s* (*admin s*)
using *1 2* **unfolding** *a* **by** (*auto simp*: *com-defs*)

lemma *φ-def3*:
assumes *step s a* = (*ou*,*s′*)
shows
φ (*Trans s a ou s′*) $\longleftrightarrow$
($\exists$ *pst*. *a* = *Uact* (*uPost* (*owner s PID*) (*pass s* (*owner s PID*)) *PID pst*) $\wedge$ *ou* = *outOK*) $\vee$
($\exists$ *aid*. *a* = *COMact* (*comSendPost* (*admin s*) (*pass s* (*admin s*)) *aid PID*) $\wedge$
ou = *O-sendPost* (*aid*, *clientPass s aid*, *PID*, *post s PID*, *owner s PID*, *vis s PID*))
unfolding *φ-def2*
using *comSendPost-out*[*OF assms*] *uPost-out*[*OF assms*]
by *blast*

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* $\neq$ *outErr* *p* = *pass s uid*
uid = *admin s*
proof −
from *assms*(*1*) **obtain** *uid p pst aid* **where** (*a* = *Uact* (*uPost uid p PID pst*) $\wedge$ *ou* = *outOK*) $\vee$
(*a* = *COMact* (*comSendPost uid p aid PID*) $\wedge$ *ou* $\neq$ *outErr*)
unfolding *φ-def2* **by** *auto*
then show *thesis* **proof**(*elim disjE*)
assume *a* = *Uact* (*uPost uid p PID pst*) $\wedge$ *ou* = *outOK*

with *assms*(*2*) **show** *thesis* **by** (*intro UpdateT*) (*auto simp*: *u-defs*)
next
assume *a* = *COMact* (*comSendPost uid p aid PID*) ∧ *ou* ≠ *outErr*
with *assms*(*2*) **show** *thesis* **by** (*intro Send*) (*auto simp*: *com-defs*)
qed
qed

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* **.**

lemma *Uact-uPaperC-step-eqButPID*:
assumes *a*: *a* = *Uact* (*uPost uid p PID pst*)
and *step s a* = (*ou*,*s'*)
shows *eqButPID s s'*
using *assms* **unfolding** *eqButPID-def eeqButPID-def eeqButPID-F-def*
by (*auto simp*: *u-defs split*: *if-splits*)

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−
have *s's1'*: *eqButPID s' s1'*
using *eqButPID-step local.step ss1 step1* **by** *blast*
show *?thesis* **using** *step step1 φ*
using *eqButPID-stateSelectors*[*OF ss1*]
unfolding *φ-def2*
by (*auto simp*: *u-defs com-defs*)
qed

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'*)
by (*metis eqButPID-step-φ-imp eqButPID-sym assms*)

end

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

6.1.4 Issuer declassification bound

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

(1) the presence of the sends (i.e., the number of the sending actions)

(2) the public knowledge that what is being sent is always the last version (modeled as the correlation predicate)

unless:

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

See [3] for more details.

context *Post-ISSUER*
begin

fun T :: ($state$,act,out) $trans \Rightarrow bool$ **where**
T ($Trans\ s\ a\ ou\ s'$) $\longleftrightarrow$
($\exists$ $uid \in UIDs$.
$uid \in\in userIDs\ s' \wedge PID \in\in postIDs\ s' \wedge$
($uid = admin\ s' \vee$
$uid = owner\ s'\ PID \vee$
$uid \in\in friendIDs\ s'\ (owner\ s'\ PID) \vee$
$vis\ s'\ PID = PublicV$))

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

abbreviation *corr* :: *value list* ⇒ *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*
done

6.1.5 Unwinding proof

lemma *reach-PublicV-imples-FriendV*[*simp*]:
assumes *reach s*
and *vis s pID* ≠ *PublicV*
shows *vis s pID* = *FriendV*
using *assms reach-vis* **by** *auto*

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*))
using *assms* **proof** *induct*
case (*Step trn*) **thus** *?case*
by (*cases trn*) *auto*
qed (*simp add*: *istate-def*)

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'*)
using *reachNT-state*[*OF 1*] *2 3 4* **using** *φ-def2*
by (*auto simp add*: *u-defs com-defs*)

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−
  have *s'T*: *reachNT s'* **using** *step sT T* **using** *reachNT-PairI* **by** *blast*
  note *op* = *reachNT-state*[*OF s'T*]
  note [*simp*] = *all-defs*
  note *s* = *reachNT-reach*[*OF sT*]
  note *willUse* =
    step step1 γ
    op
    reach-vis[*OF s*]
    eqButPID-stateSelectors[*OF ss1*]
    eqButPID-actions[*OF ss1*]
    eqButPID-update[*OF ss1*] *eqButPID-not-PID*[*OF ss1*]

    eqButPID-eqButF[*OF ss1*]
    eqButPID-setShared[*OF ss1*] *eqButPID-updateShared*[*OF ss1*]
    eeqButPID-F-not-PID eqButPID-not-PID-sharedWith
    eqButPID-insert2[*OF ss1*]
  show *?thesis*
  proof (*cases a*)
    case (*Sact x1*)
    with *willUse* **show** *?thesis* **by** (*cases x1*) *auto*
  next
    case (*Cact x2*)
    with *willUse* **show** *?thesis* **by** (*cases x2*) *auto*
  next
    case (*Dact x3*)
    with *willUse* **show** *?thesis* **by** (*cases x3*) *auto*
  next
    case (*Uact x4*)
    with *willUse* **show** *?thesis* **by** (*cases x4*) *auto*
  next
    case (*Ract x5*)
    with *willUse* **show** *?thesis*
    proof (*cases x5*)
      case (*rPost uid p pid*)
      with *Ract willUse* **show** *?thesis* **by** (*cases pid* = *PID*) *auto*
    qed *auto*
  next
    case (*Lact x6*)
    with *willUse* **show** *?thesis*
    proof (*cases x6*)
      case (*lClientsPost uid p pid*)
      with *Lact willUse* **show** *?thesis* **by** (*cases pid* = *PID*) *auto*
    qed *auto*
  next
    case (*COMact x7*)
    with *willUse* **show** *?thesis* **by** (*cases x7*) *auto*

qed
qed

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'*
using *ss1 step step1*
using *eqButPID-stateSelectors*[*OF ss1*]
eqButPID-update[*OF ss1*] *eqButPID-setShared*[*OF ss1*]
unfolding *a* **by** (*auto simp*: *u-defs*)

definition $\Delta 0$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 0$ *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 $\Delta 1$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 1$ *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 $\Delta 2$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 2$ *s vl s1 vl1* ≡
PID ∈∈ *postIDs s* ∧
eqButPID s s1 ∧
vl = [] ∧ *list-all isPVal vl1*

lemma *istate-*$\Delta 0$:
assumes *B*: *B vl vl1*
shows $\Delta 0$ *istate vl istate vl1*
using *assms* **unfolding** $\Delta 0$*-def B-def istate-def* **by** *auto*

lemma *unwind-cont-*$\Delta 0$: *unwind-cont* $\Delta 0$ {$\Delta 0$,$\Delta 1$}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. $\Delta 0$ *s vl s1 vl1* ∨ $\Delta 1$ *s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** $\Delta 0$ *s vl s1 vl1*
hence *rs*: *reach s* **and** *ss1*: *s1* = *s* **and** *pPID*: *post s PID* = *emptyPost*
and *ch*: *corrFrom* (*post s1 PID*) *vl1*
and *l*: *map PValS-tgtAPI* (*filter isPValS vl*) = *map PValS-tgtAPI* (*filter isPValS vl1*)

and *PID*: ¬ *PID* ∈∈ *postIDs s* **and** *vlvl1*: *vl* = [] ⟹ *vl1* = []
using *reachNT-reach* **unfolding** Δ*0-def* **by** *auto*
show *iaction ?*Δ *s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'* **let** *?trn1* = *Trans s1 a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
show *match ?*Δ *s s1 vl1 a ou s' vl'* ∨ *ignore ?*Δ *s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
proof−
have φ: ¬ φ *?trn* **using** *PID step* **unfolding** φ*-def2* **by** (*auto simp*: *u-defs com-defs*)
hence *vl'*: *vl'* = *vl* **using** *c* φ **unfolding** *consume-def* **by** *simp*
have *pPID'*: *post s' PID* = *emptyPost*
using *pPID PID step*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*, *auto simp*: *all-defs*) **.**
subgoal for *x2* **apply**(*cases x2*, *auto simp*: *all-defs*) **.**
subgoal for *x3* **apply**(*cases x3*, *auto simp*: *all-defs*) **.**
subgoal for *x4* **apply**(*cases x4*, *auto simp*: *all-defs*) **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*, *auto simp*: *all-defs*) **.**
done
have *?match* **proof**(*cases* ∃ *uid p*. *a* = *Cact* (*cPost uid p PID*) ∧ *ou* = *outOK*)
case *True*
then obtain *uid p* **where** *a*: *a* = *Cact* (*cPost uid p PID*) **and** *ou*: *ou* = *outOK* **by** *auto*
have *PID'*: *PID* ∈∈ *postIDs s'*
using *step PID* **unfolding** *a ou* **by** (*auto simp*: *c-defs*)
note *uid* = *reachNT-state*[*OF rsT*]
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** φ **unfolding** *consume-def ss1* **by** *auto*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
have Δ*1 s' vl' s' vl1* **using** *l PID' c ch vlvl1 pPID' pPID*
unfolding *ss1* Δ*1-def vl'* **by** *auto*
thus *?*Δ *s' vl' s' vl1* **by** *simp*
qed
next

```
          case False note a = False
          have PID': ¬ PID ∈∈ postIDs s'
            using step PID a
            apply(cases a)
            subgoal for x1 apply(cases x1) apply(fastforce simp: s-defs)+ .
            subgoal for x2 apply(cases x2) apply(fastforce simp: c-defs)+ .
            subgoal for x3 apply(cases x3) apply(fastforce simp: d-defs)+ .
            subgoal for x4 apply(cases x4) apply(fastforce simp: u-defs)+ .
            subgoal by auto
            subgoal by auto
            subgoal for x7 apply(cases x7) apply(fastforce simp: com-defs)+ .
            done
          show ?thesis proof
            show validTrans ?trn1 unfolding ss1 using step by simp
          next
              show consume ?trn1 vl1 vl1 using φ unfolding consume-def ss1 by
auto
          next
            show γ ?trn = γ ?trn1 unfolding ss1 by simp
          next
            assume γ ?trn thus g ?trn = g ?trn1 unfolding ss1 by simp
          next
              have Δ0 s' vl' s' vl1 using a PID' pPID pPID' ch vlvl1 l unfolding
Δ0-def vl' ss1 by simp
            thus ?Δ s' vl' s' vl1 by simp
          qed
        qed
        thus ?thesis by simp
      qed
    qed
  thus ?thesis using vlvl1 by simp
  qed
qed

lemma unwind-cont-Δ1: unwind-cont Δ1 {Δ1,Δ2}
proof(rule, simp)
  let ?Δ = λs vl s1 vl1. Δ1 s vl s1 vl1 ∨ Δ2 s vl s1 vl1
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and Δ1 s vl s1 vl1
  hence vlvl1: vl = [] ⟶ vl1 = [] and ch1: corrFrom (post s1 PID) vl1
  and rs: reach s and ss1: eqButPID s s1 and PID: PID ∈∈ postIDs s
  and l: map PValS-tgtAPI (filter isPValS vl) = map PValS-tgtAPI (filter isPValS
vl1)
  using reachNT-reach unfolding Δ1-def by auto
  have PID1: PID ∈∈ postIDs s1 using eqButPID-stateSelectors[OF ss1] PID by
auto
   have own: owner s PID ∈ set (userIDs s) using reach-owner-userIDs[OF rs
PID] .
 hence own1: owner s1 PID ∈ set (userIDs s1) using eqButPID-stateSelectors[OF
```

ss1] **by** *auto*
show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof(*cases vl1*)
case (*Cons v1 vll1*) **note** *vl1* = *Cons*
obtain *v vll* **where** *vl*: *vl* = *v* # *vll* **using** *vl1 vlvl1* **by** (*cases vl*) *auto*
show *?thesis*
proof(*cases v1*)
case (*PVal pst1*) **note** *v1* = *PVal*
let *?uid1* = *owner s1 PID* **let** *?p1* = *pass s1 ?uid1*
have *uid1*: *?uid1* ∈∈ *userIDs s1* **using** *reach-owner-userIDs*[*OF rs1 PID1*] **.**
define *a1* **where** *a1* ≡ *Uact* (*uPost ?uid1 ?p1 PID pst1*)
obtain *s1′ ou1* **where** *step1*: *step s1 a1* = (*ou1*,*s1′*) **by** *force*
hence *ou1*: *ou1* = *outOK* **using** *PID1 uid1* **unfolding** *a1-def* **by** (*auto simp*: *u-defs*)
let *?trn1* = *Trans s1 a1 ou1 s1′*
have *φ1*: *φ ?trn1* **unfolding** *a1-def PID1 ou1* **by** *simp*
have *2*[*simp*]: *post s1′ PID* = *pst1*
using *step1* **unfolding** *a1-def ou1* **by** (*auto simp*: *u-defs*)
have *?uid1* = *owner s PID* **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *simp*
hence *uid1*: *?uid1* ∉ *UIDs* **using** *reachNT-state own rsT PID* **by** *auto*
have *eqButPID s1 s1′* **using** *step1 a1-def Uact-uPaperC-step-eqButPID* **by** *auto*
hence *ss1′*: *eqButPID s s1′* **using** *ss1* **using** *eqButPID-trans* **by** *blast*
have *?iact* **proof**
show *step s1 a1* = (*ou1*, *s1′*) *φ ?trn1* **by** *fact+*
show *consume* (*Trans s1 a1 ou1 s1′*) *vl1 vll1*
using *φ1* **unfolding** *consume-def vl1 a1-def v1* **by** *simp*
show ¬ *γ ?trn1* **using** *uid1* **unfolding** *a1-def* **by** *auto*
show *?Δ s vl s1′ vll1*
proof(*cases vll1*)
case *Nil*
have *Δ1 s vl s1′ vll1* **using** *PID ss1′ l* **unfolding** *Δ1-def B-def vl1 v1 Nil* **by** *auto*
thus *?thesis* **by** *simp*
next
case (*Cons w1 vlll1*) **note** *vll1* = *Cons*
have *Δ1 s vl s1′ vll1* **using** *PID ss1′ l ch1*
unfolding *Δ1-def B-def vl1 v1 vl* **by** *auto*
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **by** *simp*
next
case (*PValS aid1 pst1*) **note** *v1* = *PValS*
have *pst1*: *pst1* = *post s1 PID* **using** *ch1* **unfolding** *vl1 v1* **by** *simp*
show *?thesis*
proof(*cases v*)
case (*PVal pst*) **note** *v* = *PVal*

hence *vll*: *vll* ≠ [] **using** *vlvl1 l* **unfolding** *vl vl1 v v1* **by** *auto*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
have *vl'*: *vl'* = *vl* ∨ *vl'* = *vll* **using** *c* **unfolding** *vl consume-def* **by** (*cases φ ?trn*) *auto*
hence *vl'NE*: *vl'* ≠ [] **using** *vll vl* **by** *auto*
have *fvl'*: *filter isPValS vl'* = *filter isPValS vll* **using** *vl'* **unfolding** *vl v* **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by** *fastforce*
let *?trn1* = *Trans s1 a ou1 s1'*
have *s's1'*: *eqButPID s' s1'* **using** *eqButPID-step ss1 step step1* **by** *blast*
have *γγ1*: *γ ?trn* ⟷ *γ ?trn1* **by** *simp*
have *PID'*: *PID* ∈∈ *postIDs s'* **using** *step rs PID* **using** *reach-postIDs-persist* **by** *blast*
have *2*[*simp*]: ¬ *φ ?trn1* ⟹ *post s1' PID* = *post s1 PID*
using *step1 PID1* **unfolding** *φ-def2*
apply(*cases a*, *auto*)
subgoal for *x1* **apply**(*cases x1*, *auto simp*: *all-defs*) **.**
subgoal for *x2* **apply**(*cases x2*, *auto simp*: *all-defs*) **.**
subgoal for *x3* **apply**(*cases x3*, *auto simp*: *all-defs*) **.**
subgoal for *x4* **apply**(*cases x4*, *auto simp*: *all-defs*) **.**
subgoal for *x4* **apply**(*cases x4*, *auto simp*: *all-defs*) **.**
subgoal for *x7* **apply**(*cases x7*, *auto simp*: *all-defs*) **.**
subgoal for *x7* **apply**(*cases x7*, *auto simp*: *all-defs*) **.**
done
show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
proof(*cases γ ?trn*)
case *True* **note** *γ* = *True*
have *ou*: (∃ *uid p aid pid*. *a* = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨
ou = *ou1*
using *eqButPID-step-γ-out*[*OF ss1 step step1 rsT T rs1 γ*] **.**
{**assume** *φ*: *φ ?trn*
hence *f ?trn* = *v* **using** *c* **unfolding** *consume-def vl* **by** *simp*
hence ∀ *ca*. *a* ≠ *COMact ca* **using** *φ* **unfolding** *φ-def3*[*OF step*] *v* **by** *auto*
hence *False* **using** *T-φ-γ*[*OF rsT step φ*] *γ* **by** *auto*
}
hence *φ*: ¬ *φ ?trn* **by** *auto*
have *vl'*: *vl'* = *vl* **using** *φ c* **unfolding** *consume-def* **by** *simp*
have *φ1*: ¬ *φ ?trn1* **using** *step step1 ss1 φ eqButPID-step-φ* **by** *blast*
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *auto*
show *consume ?trn1 vl1 vl1* **using** *φ1* **unfolding** *consume-def* **by** *simp*

show γ *?trn* $\longleftrightarrow$ γ *?trn1* **by** *fact*
next
show *g ?trn* = *g ?trn1* **using** *ou* **by** (*cases a*) *auto*
have Δ*1 s' vl' s1' vl1*
using *PID' s's1' vlvl1 l ch1* φ*1* **unfolding** Δ*1-def vl'* **by** *auto*
thus *?*Δ *s' vl' s1' vl1* **by** *simp*
qed
thus *?thesis* **by** *simp*
next
case *False* **note** γ = *False*
show *?thesis*
proof(*cases* φ *?trn*)
case *True* **note** φ = *True*
hence *f ?trn* = *v* **using** *c* **unfolding** *consume-def vl* **by** *simp*
hence $\forall$ *ca. a* $\neq$ *COMact ca* **using** φ **unfolding** φ*-def3*[*OF step*] *v* **by** *auto*
then obtain *uid p pstt* **where** *a*: *a* = *Uact* (*uPost uid p PID pstt*)
using φ **unfolding** φ*-def2* **by** *auto*
hence *ss'*: *eqButPID s s'* **using** *step Uact-uPaperC-step-eqButPID* **by** *auto*
hence *s's1*: *eqButPID s' s1* **using** *ss1 eqButPID-sym eqButPID-trans* **by** *blast*
have *?ignore* **proof**
show $\neg$ γ *?trn* **by** *fact*
have Δ*1 s' vl' s1 vl1*
using *PID' s's1' ch1 l vl'NE s's1* **unfolding** Δ*1-def fvl' vl v* **by** *auto*
thus *?*Δ *s' vl' s1 vl1* **by** *simp*
qed
thus *?thesis* **by** *simp*
next
case *False* **note** φ = *False*
have *vl'*: *vl'* = *vl* **using** φ *c* **unfolding** *consume-def* **by** *simp*
have φ*1*: $\neg$ φ *?trn1* **using** *step step1 ss1* φ *eqButPID-step-*φ **by** *blast*
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *auto*
show *consume ?trn1 vl1 vl1* **using** φ*1* **unfolding** *consume-def* **by** *simp*
show γ *?trn* $\longleftrightarrow$ γ *?trn1* **by** *fact*
next
assume γ *?trn* **thus** *g ?trn* = *g ?trn1* **using** γ **by** *simp*
next
have Δ*1 s' vl' s1' vl1*
using *PID' s's1' vlvl1 l ch1* φ*1* **unfolding** Δ*1-def vl'* **by** *auto*
thus *?*Δ *s' vl' s1' vl1* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
qed
qed

thus *?thesis* **using** *vlvl1* **by** *simp*
next
case (*PValS aid pst*) **note** $v = PValS$
have *?react* **proof**
fix $a :: act$ **and** $ou :: out$ **and** $s' :: state$ **and** vl'
let $?trn = Trans\ s\ a\ ou\ s'$
assume *step*: $step\ s\ a = (ou, s')$ **and** T: $\neg\ T\ ?trn$ **and** c: $consume\ ?trn\ vl\ vl'$
have vl': $vl' = vl \lor vl' = vll$ **using** c **unfolding** *vl consume-def* **by** (*cases* φ *?trn*) *auto*
obtain *ou1 s1'* **where** *step1*: $step\ s1\ a = (ou1, s1')$ **by** *fastforce*
let $?trn1 = Trans\ s1\ a\ ou1\ s1'$
have $s's1'$: *eqButPID* s' $s1'$ **using** *eqButPID-step ss1 step step1* **by** *blast*
have $\gamma\gamma 1$: $\gamma\ ?trn \longleftrightarrow \gamma\ ?trn1$ **by** *simp*
have PID': $PID \in\in postIDs\ s'$ **using** *step rs PID* **using** *reach-postIDs-persist* **by** *blast*
show *match* $?\Delta$ *s s1 vl1 a ou* s' vl' $\lor$ *ignore* $?\Delta$ *s s1 vl1 a ou* s' vl' (**is** *?match* $\lor$ *?ignore*)
proof−
have *?match* **proof**(*cases* φ *?trn*)
case *True* **note** $\varphi = True$
have $\varphi 1$: φ *?trn1* **using** *step step1 ss1* φ *eqButPID-step-*φ **by** *blast*
let $?ad = admin\ s$ **let** $?p = pass\ s\ ?ad$ **let** $?pst = post\ s\ PID$
let $?uid = owner\ s\ PID$ **let** $?vs = vis\ s\ PID$
obtain vl': $vl' = vll$
and *ou*: $ou = O\text{-}sendPost\ (aid, clientPass\ s\ aid, PID, pst, ?uid, ?vs)$
and *a*: $a = COMact\ (comSendPost\ ?ad\ ?p\ aid\ PID)$ **and** *pst*: $pst = ?pst$
using φ *c* **unfolding** φ*-def3*[*OF step*] *consume-def vl v* **by** *auto*
let $?pst1 = post\ s1\ PID$
have $clientPass\ s\ aid = clientPass\ s1\ aid$ **and** $?uid = owner\ s1\ PID$
and $?vs = vis\ s1\ PID$
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
hence *ou1*: $ou1 = O\text{-}sendPost\ (aid, clientPass\ s\ aid, PID, ?pst1, ?uid, ?vs)$
using *step1* $\varphi 1$ **unfolding** φ*-def3*[*OF step1*] *vl1 v1 a*
by (*auto simp*: *com-defs*)
have *2*[*simp*]: $post\ s1'\ PID = pst1$
using *step1* **unfolding** *a ou1 pst1* **by** (*auto simp*: *com-defs*)
have *ch-vll1*: *corrFrom pst1 vll1* **using** *ch1* **unfolding** *pst1*[*symmetric*] *vl1 v1* **by** *auto*
show *?thesis* **proof**
show *validTrans ?trn1* **using** *step1* **by** *auto*
show *consume* (*Trans s1 a ou1 s1'*) *vl1 vll1*
using *l* $\varphi 1$ *pst1* **unfolding** *consume-def vl vl1 v v1 a ou1* **by** *simp*
show $\gamma\ ?trn \longleftrightarrow \gamma\ ?trn1$ **by** *fact*
next
show $g\ ?trn = g\ ?trn1$ **unfolding** *a ou ou1* **by** (*simp add*: *ss1*)
show $?\Delta\ s'\ vl'\ s1'\ vll1$

proof(*cases vll* = [])
case *False*
hence Δ*1 s' vl' s1' vll1* **using** *PID' s's1' vlvl1 l ch1 ch-vll1*
unfolding Δ*1-def vl' vl vl1 v v1* **by** *auto*
thus *?thesis* **by** *simp*
next
case *True*
hence *list-all isPVal vll1* **using** *l* **unfolding** *vl vl1 v v1* **by** (*simp add*: *filter-isPValS-Nil*)
hence Δ*2 s' vl' s1' vll1* **using** *True PID' s's1' vlvl1 l ch1 ch-vll1*
unfolding Δ*2-def vl' vl vl1 v v1* **by** *simp*
thus *?thesis* **by** *simp*
qed
qed
next
case *False* **note** φ = *False*
have *vl'*: *vl'* = *vl* **using** φ *c* **unfolding** *consume-def* **by** *simp*
have φ*1*: $\neg$ φ *?trn1* **using** *step step1 ss1* φ *eqButPID-step-*φ **by** *blast*
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *auto*
show *consume ?trn1 vl1 vl1* **using** φ*1* **unfolding** *consume-def* **by** *simp*
show γ *?trn* $\longleftrightarrow$ γ *?trn1* **by** *fact*
next
assume γ *?trn* **note** γ = *this*
have *ou*: ($\exists$ *uid p aid pid*. *a* = *COMact* (*comSendPost uid p aid pid*) $\wedge$ *outPurge ou* = *outPurge ou1*) $\vee$
ou = *ou1*
using *eqButPID-step-*γ*-out*[*OF ss1 step step1 rsT T rs1* γ] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
have *2*[*simp*]: *post s1' PID* = *post s1 PID*
using *step1 PID1* φ*1* **unfolding** φ*-def2*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*, *auto simp*: *all-defs*) **.**
subgoal for *x2* **apply**(*cases x2*, *auto simp*: *all-defs*) **.**
subgoal for *x3* **apply**(*cases x3*, *auto simp*: *all-defs*) **.**
subgoal for *x4* **apply**(*cases x4*, *auto simp*: *all-defs*) **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*, *auto simp*: *all-defs*) **.**
done
have Δ*1 s' vl' s1' vl1* **using** *PID' s's1' vlvl1 l ch1*
unfolding Δ*1-def vl' vl vl1 v v1* **by** *auto*
thus *?*Δ *s' vl' s1' vl1* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
thus *?thesis* **by** *simp*

qed
qed
thus *?thesis* **using** *vlvl1* **by** *simp*
qed
qed
next
case *Nil* **note** *vl1 = Nil*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn = Trans s a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl′*
obtain *ou1 s1′* **where** *step1*: *step s1 a* = (*ou1*, *s1′*) **by** *fastforce*
let *?trn1 = Trans s1 a ou1 s1′*
have *s′s1′*: *eqButPID s′ s1′* **using** *eqButPID-step ss1 step step1* **by** *blast*
have *γγ1*: *γ ?trn* ⟷ *γ ?trn1* **by** *simp*
have *PID′*: *PID* ∈∈ *postIDs s′* **using** *step rs PID* **using** *reach-postIDs-persist* **by** *blast*
show *match ?Δ s s1 vl1 a ou s′ vl′* ∨ *ignore ?Δ s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
proof(*cases φ ?trn*)
case *True* **note** *φ = True*
then obtain *v vll* **where** *vl*: *vl = v # vll*
and *f*: *f ?trn = v* **using** *c* **unfolding** *consume-def* **by** (*cases vl*) *auto*
obtain *pst* **where** *v*: *v = PVal pst* **using** *l* **unfolding** *vl1 vl* **by** (*cases v*) *auto*
have *fvll*: *filter isPValS vll* = []**using** *l* **unfolding** *vl1 vl* **by** *auto*
have *vl′*: *vl′ = vll* **using** *c φ* **unfolding** *vl consume-def* **by** *auto*
hence *0*: ∀ *ca. a* ≠ *COMact ca* **using** *φ v f* **unfolding** *φ-def3*[*OF step*] **by** *auto*
then obtain *uid p pstt* **where** *a*: *a = Uact* (*uPost uid p PID pstt*)
using *φ* **unfolding** *φ-def2* **by** *auto*
hence *ss′*: *eqButPID s s′* **using** *step Uact-uPaperC-step-eqButPID* **by** *auto*
hence *s′s1*: *eqButPID s′ s1* **using** *ss1 eqButPID-sym eqButPID-trans* **by** *blast*
have *?ignore* **proof**
show ¬ *γ ?trn* **using** *T-φ-γ*[*OF rsT step φ 0*] **.**
have *Δ1 s′ vl′ s1 vl1*
using *PID′ s′s1′ ch1 l s′s1 vl1 fvll* **unfolding** *Δ1-def vl v vl′* **by** *auto*
thus *?Δ s′ vl′ s1 vl1* **by** *simp*
qed
thus *?thesis* **by** *simp*
next
case *False* **note** *φ = False*
have *vl′*: *vl′ = vl* **using** *φ c* **unfolding** *consume-def* **by** *simp*
have *φ1*: ¬ *φ ?trn1* **using** *step step1 ss1 φ eqButPID-step-φ* **by** *blast*
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *auto*
show *consume ?trn1 vl1 vl1* **using** *φ1* **unfolding** *consume-def* **by** *simp*
show *γ ?trn* ⟷ *γ ?trn1* **by** *fact*

next
assume γ *?trn* **note** γ = *this*
have *ou*: (∃ *uid p aid pid*. *a* = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨
ou = *ou1*
using *eqButPID-step-γ-out*[*OF ss1 step step1 rsT T rs1 γ*] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
have *2*[*simp*]: *post s1′ PID* = *post s1 PID*
using *step1 PID1 φ1* **unfolding** *φ-def2*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*, *auto simp*: *all-defs*) **.**
subgoal for *x2* **apply**(*cases x2*, *auto simp*: *all-defs*) **.**
subgoal for *x3* **apply**(*cases x3*, *auto simp*: *all-defs*) **.**
subgoal for *x4* **apply**(*cases x4*, *auto simp*: *all-defs*) **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*, *auto simp*: *all-defs*) **.**
done
have Δ*1 s′ vl′ s1′ vl1* **using** *PID′ s′s1′ vlvl1 l ch1*
unfolding Δ*1-def vl′ vl1* **by** *auto*
thus *?*Δ *s′ vl′ s1′ vl1* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vl1* **by** *simp*
qed
qed

lemma *unwind-cont-*Δ*2*: *unwind-cont* Δ*2* {Δ*2*}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. Δ*2 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** Δ*2 s vl s1 vl1*
hence *PID*: *PID* ∈∈ *postIDs s* **and** *ss1*: *eqButPID s s1* **and** *vl*: *vl* = [] **and** *lvl1*: *list-all isPVal vl1*
and *rs*: *reach s* **using** *reachNT-reach* **unfolding** Δ*2-def* **by** *auto*
have *PID1*: *PID* ∈∈ *postIDs s1* **using** *eqButPID-stateSelectors*[*OF ss1*] *PID* **by** *auto*
have *own*: *owner s PID* ∈ *set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
hence *own1*: *owner s1 PID* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
show *iaction ?*Δ *s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof(*cases vl1*)
case (*Cons v1 vll1*) **note** *vl1* = *Cons*
obtain *pst1* **where** *v1*: *v1* = *PVal pst1* **and** *lvll1*: *list-all isPVal vll1*

using *lvl1* **unfolding** *vl1* **by** (*cases v1*) *auto*
define *uid* **where** *uid* ≡ *owner s PID* **define** *p* **where** *p* ≡ *pass s uid*
define *a1* **where** *a1* ≡ *Uact* (*uPost uid p PID pst1*)
have *uid1*: *uid* = *owner s1 PID* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid-def p-def*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1′* **where** *step1*: *step s1 a1* = (*ou1*, *s1′*) **by**(*cases step s1 a1*) *auto*
have *ou1*: *ou1* = *outOK* **using** *step1 PID1 own1* **unfolding** *a1-def uid1 p1* **by** (*auto simp*: *u-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid-def* **using** *rsT reachNT-state own PID* **by** *blast*
let *?trn1* = *Trans s1 a1 ou1 s1′*
have *?iact* **proof**
show *step s1 a1* = (*ou1*, *s1′*) **using** *step1* **.**
next
show *φ1*: *φ ?trn1* **unfolding** *φ-def2 a1-def ou1* **by** *simp*
show *consume ?trn1 vl1 vll1*
using *φ1* **unfolding** *vl1 consume-def a1-def v1* **by** *simp*
show ¬ *γ ?trn1* **using** *uid* **unfolding** *a1-def* **by** *simp*
next
have *eqButPID s1 s1′* **using** *Uact-uPaperC-step-eqButPID*[*OF - step1*] *a1-def* **by** *auto*
hence *ss1′*: *eqButPID s s1′* **using** *eqButPID-trans ss1* **by** *blast*
show Δ*2 s vl s1′ vll1*
using *PID ss1′ lvll1* **unfolding** Δ*2-def vl* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case *Nil* **note** *vl1* = *Nil*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn* = *Trans s a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl′*
obtain *ou1 s1′* **where** *step1*: *step s1 a* = (*ou1*, *s1′*) **by** *fastforce*
let *?trn1* = *Trans s1 a ou1 s1′*
have *s′s1′*: *eqButPID s′ s1′* **using** *eqButPID-step ss1 step step1* **by** *blast*
have *γγ1*: *γ ?trn* ⟷ *γ ?trn1* **by** *simp*
have *PID′*: *PID* ∈∈ *postIDs s′* **using** *step rs PID* **using** *reach-postIDs-persist* **by** *blast*
have *φ*: ¬ *φ ?trn* **and** *vl′*: *vl′* = *vl* **using** *c* **unfolding** *vl consume-def* **by** *auto*
hence *φ1*: ¬ *φ ?trn1* **using** *eqButPID-step-φ step ss1 step1* **by** *auto*
show *match ?*Δ *s s1 vl1 a ou s′ vl′* ∨ *ignore ?*Δ *s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
proof−
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *auto*
show *consume ?trn1 vl1 vl1* **using** *φ1* **unfolding** *consume-def* **by** *simp*

show γ *?trn* $\longleftrightarrow$ γ *?trn1* **by** *fact*
next
assume γ *?trn* **note** γ = *this*
have *ou*: ($\exists$ *uid p aid pid. a* = *COMact* (*comSendPost uid p aid pid*) $\wedge$
outPurge ou = *outPurge ou1*) $\vee$
ou = *ou1*
using *eqButPID-step-*γ*-out*[*OF ss1 step step1 rsT T rs1* γ] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
have *2*[*simp*]: *textPost* (*post s1' PID*) = *textPost* (*post s1 PID*)
using *step1 PID1* φ*1* **unfolding** φ*-def2*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*, *auto simp*: *all-defs*) **.**
subgoal for *x2* **apply**(*cases x2*, *auto simp*: *all-defs*) **.**
subgoal for *x3* **apply**(*cases x3*, *auto simp*: *all-defs*) **.**
subgoal for *x4* **apply**(*cases x4*, *auto simp*: *all-defs*) **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*, *auto simp*: *all-defs*) **.**
done
show Δ*2 s' vl' s1' vl1* **using** *PID' s's1' vl*
unfolding Δ*2-def vl1 vl'* **by** *auto*
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vl1* **by** *simp*
qed
qed

definition *Gr* **where**
Gr =
{
(Δ*0*, {Δ*0*,Δ*1*}),
(Δ*1*, {Δ*1*,Δ*2*}),
(Δ*2*, {Δ*2*})
}

theorem *Post-secure*: *secure*
apply (*rule unwind-decomp-secure-graph*[*of Gr* Δ*0*])
unfolding *Gr-def*
apply (*simp*, *smt insert-subset order-refl*)
using *istate-*Δ*0 unwind-cont-*Δ*0 unwind-cont-*Δ*1 unwind-cont-*Δ*2*
unfolding *Gr-def* **by** *auto*

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* ⇒ *bool* **where**
γ (*Trans* - *a* - -) ⟷
(∃ *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*

$|comPurge\ (comConnectClient\ uID\ p\ aID\ sp) = comConnectClient\ uID\ emptyPass\ aID\ sp$

$|comPurge\ (comReceivePost\ aID\ sp\ pID\ pst\ uID\ vs) =$
$(let\ pst' = (if\ aID = AID \wedge pID = PID\ then\ emptyPost\ else\ pst)$
$in\ comReceivePost\ aID\ sp\ pID\ pst'\ uID\ vs)$

$|comPurge\ (comSendPost\ uID\ p\ aID\ pID) = comSendPost\ uID\ emptyPass\ aID\ pID$
$|comPurge\ (comSendCreateOFriend\ uID\ p\ aID\ uID') = comSendCreateOFriend\ uID\ emptyPass\ aID\ uID'$
$|comPurge\ (comSendDeleteOFriend\ uID\ p\ aID\ uID') = comSendDeleteOFriend\ uID\ emptyPass\ aID\ uID'$
$|comPurge\ ca = ca$

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

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

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*) ∧ *ou* = *ou'*
g (*Trans s a ou s'*) = (*COMact* (*comConnectClient uID p aID sp*), *ou'*)
$\longleftrightarrow$ (∃ *p'*. *a* = *COMact* (*comConnectClient uID p' aID sp*) ∧ *p* = *emptyPass* ∧ *ou* = *ou'*)
g (*Trans s a ou s'*) = (*COMact* (*comConnectServer aID sp*), *ou'*)
$\longleftrightarrow$ *a* = *COMact* (*comConnectServer aID sp*) ∧ *ou* = *ou'*
g (*Trans s a ou s'*) = (*COMact* (*comReceivePost aID sp pID pst' uID v*), *ou'*)
$\longleftrightarrow$ (∃ *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*))
$\longleftrightarrow$ (∃ *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'*)
$\longleftrightarrow$ (∃ *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'*)
$\longleftrightarrow$ *a* = *COMact* (*comReceiveCreateOFriend aID cp uID uID'*) ∧ *ou* = *ou'*
g (*Trans s a ou s'*) = (*COMact* (*comSendDeleteOFriend uID p aID uID'*), *ou'*)
$\longleftrightarrow$ (∃ *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'*)
$\longleftrightarrow$ *a* = *COMact* (*comReceiveDeleteOFriend aID cp uID uID'*) ∧ *ou* = *ou'*
by (*cases a*; *auto simp*: *comPurge-simps ObservationSetup-RECEIVER.comPurge.simps*)+

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*
unfolding *eeqButPID-def* **by** *auto*

lemma *eeqButPID-sym*:
assumes *eeqButPID psts psts1* **shows** *eeqButPID psts1 psts*
using *assms* **unfolding** *eeqButPID-def* **by** *auto*

lemma *eeqButPID-trans*:
assumes *eeqButPID psts psts1* **and** *eeqButPID psts1 psts2* **shows** *eeqButPID psts psts2*
using *assms* **unfolding** *eeqButPID-def* **by** (*auto split*: *if-splits*)

lemma *eeqButPID-cong*:
assumes *eeqButPID psts psts1*
and $aid = AID \Longrightarrow pid = PID \Longrightarrow eqButT\ uu\ uu1$
and $aid \neq AID \vee pid \neq PID \Longrightarrow uu = uu1$
shows *eeqButPID* (*fun-upd2 psts aid pid uu*) (*fun-upd2 psts1 aid pid uu1*)
using *assms* **unfolding** *eeqButPID-def fun-upd2-def* **by** (*auto split*: *if-splits*)

lemma *eeqButPID-not-PID*:
$[\![eeqButPID\ psts\ psts1;\ aid \neq AID \vee pid \neq PID]\!] \Longrightarrow psts\ aid\ pid = psts1\ aid\ pid$
unfolding *eeqButPID-def* **by** (*auto split*: *if-splits*)

lemma *eeqButPID-toEq*:
assumes *eeqButPID psts psts1*
shows *fun-upd2 psts AID PID pst* =
  fun-upd2 psts1 AID PID pst
using *eeqButPID-not-PID*[*OF assms*]
unfolding *fun-upd2-def* **by** (*auto split*: *if-splits intro*!: *ext*)

lemma *eeqButPID-update-post*:
assumes *eeqButPID psts psts1*
shows *eeqButPID* (*fun-upd2 psts aid pid pst*) (*fun-upd2 psts1 aid pid pst*)
using *eeqButPID-not-PID*[*OF assms*]
unfolding *fun-upd2-def*
using *assms* **unfolding** *eeqButPID-def* **by** *auto*

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*
by *auto*

lemma *eqButF-sym*:
assumes *eqButF aID-bl aID-bl1*
shows *eqButF aID-bl1 aID-bl*
using *assms* **by** *auto*

lemma *eqButF-trans*:
assumes *eqButF aID-bl aID-bl1* **and** *eqButF aID-bl1 aID-bl2*
shows *eqButF aID-bl aID-bl2*
using *assms* **by** *auto*

lemma *eqButF-insert2*:
eqButF aID-bl aID-bl1 $\Longrightarrow$
eqButF (*insert2 aID b aID-bl*) (*insert2 aID b aID-bl1*)
unfolding *insert2-def*
by *simp* (*smt comp-apply fst-conv map-eq-conv split-def*)

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

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

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

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

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

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

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

lemma *eqButPID-refl*[*simp*,*intro*!]: *eqButPID s s*
unfolding *eqButPID-def* **by** *auto*

lemma *eqButPID-sym*:
assumes *eqButPID s s1* **shows** *eqButPID s1 s*

using *assms eeqButPID-sym* **unfolding** *eqButPID-def* **by** *auto*

lemma *eqButPID-trans*:
assumes *eqButPID s s1* **and** *eqButPID s1 s2* **shows** *eqButPID s s2*
using *assms eeqButPID-trans* **unfolding** *eqButPID-def*
by *simp blast*

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

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

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

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

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

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

IDsOK s = IDsOK s1
unfolding *eqButPID-def IDsOK-def*[*abs-def*] **by** *auto*

lemma *eqButPID-not-PID*:
eqButPID s s1 $\Longrightarrow$ *aid* $\neq$ *AID* $\vee$ *pid* $\neq$ *PID* $\Longrightarrow$ *outerPost s aid pid = outerPost s1 aid pid*
unfolding *eqButPID-def* **using** *eeqButPID-not-PID* **by** *auto*

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*
using *eqButPID-stateSelectors*[*OF assms*]

by (*auto simp*: *l-defs intro*!: *arg-cong2*[*of - - - - cmap*])

lemma *eqButPID-update*:
assumes *eqButPID s s1*
shows *fun-upd2* (*outerPost s*) *AID PID pst* = *fun-upd2* (*outerPost s1*) *AID PID pst*
using *assms* **unfolding** *eqButPID-def* **using** *eeqButPID-toEq* **by** (*metis fun-upd2-absorb*)

lemma *eqButPID-update-post*:
assumes *eqButPID s s1*
shows *eeqButPID* (*fun-upd2* (*outerPost s*) *aid pid pst*) (*fun-upd2* (*outerPost s1*) *aid pid pst*)
using *assms* **unfolding** *eqButPID-def* **using** *eeqButPID-update-post* **by** *auto*

lemma *eqButPID-cong*[*simp*, *intro*]:
⋀ *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* ⟹ *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*|))
⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*friendReq* := *uu1*|)) (*s1* (|*friendReq* := *uu2*|))
⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*friendIDs* := *uu1*|)) (*s1* (|*friendIDs* := *uu2*|))
⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButPID* (*s* (|*sentOuterFriendIDs* := *uu1*|)) (*s1* (|*sentOuterFriendIDs* := *uu2*|))
⋀ *uu1 uu2*. *eqButPID s s1* ⟹ *uu1* = *uu2* ⟹ *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*|))
unfolding *eqButPID-def* **by** *auto*

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'*
using *assms* **unfolding** *eqButPID-def eeqButPID-def*
unfolding *a a1* **by** (*fastforce simp*: *com-defs fun-upd2-def*)

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 –
  note [*simp*] = *all-defs*

note $*$ = *step step1 ss1 eqButPID-stateSelectors*[*OF ss1*] *eqButPID-update-post*[*OF ss1*]

then show *?thesis*
proof (*cases a*)
case (*Sact x1*)
with $*$ **show** *?thesis* **by** (*cases x1*) *auto*
next
case (*Cact x2*)
with $*$ **show** *?thesis* **by** (*cases x2*) *auto*
next
case (*Dact x3*)
with $*$ **show** *?thesis* **by** (*cases x3*) *auto*
next
case (*Uact x4*)
with $*$ **show** *?thesis* **by** (*cases x4*) *auto*
next
case (*COMact x7*)
with $*$ **show** *?thesis* **by** (*cases x7*) *auto*
qed *auto*
qed

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* $\Rightarrow$ *bool* **where**
φ (*Trans* - (*COMact* (*comReceivePost aid sp pid pst uid vs*)) *ou* -) =
(*aid* = *AID* $\wedge$ *pid* = *PID* $\wedge$ *ou* = *outOK*)
|
φ (*Trans s* - - *s'*) = *False*

lemma φ*-def2*:
φ (*Trans s a ou s'*) $\longleftrightarrow$

(∃ *uid p pst vs. a* = *COMact* (*comReceivePost AID p PID pst uid vs*) ∧ *ou* = *outOK*)
by (*cases Trans s a ou s′ rule*: *φ.cases*) *auto*

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*
using *1 2* **unfolding** *a* **by** (*auto simp*: *com-defs*)

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*)
unfolding *φ-def2*
using *comReceivePost-out*[*OF assms*]
by *blast*

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 −
from *assms*(*1*) **obtain** *sp pst uid vs* **where** *a* = *COMact* (*comReceivePost AID sp PID pst uid vs*) ∧ *ou* = *outOK*
unfolding *φ-def2* **by** *auto*
then show *thesis* **proof** −
assume *a* = *COMact* (*comReceivePost AID sp PID pst uid vs*) ∧ *ou* = *outOK*
with *assms*(*2*) **show** *thesis* **by** (*intro Recv*) (*auto simp*: *com-defs*)
qed
qed

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* **.**

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−
  have *s's1'*: *eqButPID s' s1'*
  using *eqButPID-step local.step ss1 step1* **by** *blast*
  show *?thesis* **using** *step step1 φ*
  using *eqButPID-stateSelectors*[*OF ss1*]
  unfolding *φ-def2*
  by (*auto simp*: *u-defs com-defs*)
qed

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'*)
by (*metis eqButPID-step-φ-imp eqButPID-sym assms*)

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**
φ = *φ* **and** *f* = *f* **and** *γ* = *γ* **and** *g* = *g* **and** *T* = *T* **and** *B* = *B*
done

6.2.5 Unwinding proof

lemma *reach-PublicV-imples-FriendV*[*simp*]:
assumes *reach s*
and *vis s pID* ≠ *PublicV*
shows *vis s pID* = *FriendV*
using *assms reach-vis* **by** *auto*

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*))
using *assms* **proof** *induct*
case (*Step trn*) **thus** *?case*
by (*cases trn*) *auto*
qed (*simp add*: *istate-def*)

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−

have *s'T*: *reachNT s'* **using** *step sT T* **using** *reachNT-PairI* **by** *blast*
note *op* = *reachNT-state*[*OF s'T*]
note [*simp*] = *all-defs*
note *s* = *reachNT-reach*[*OF sT*]
note *willUse* =
step step1 γ
op
reach-vis[*OF s*]
eqButPID-stateSelectors[*OF ss1*]
eqButPID-actions[*OF ss1*]
eqButPID-update[*OF ss1*] *eqButPID-not-PID*[*OF ss1*]
show *?thesis*
proof (*cases a*)
case (*Sact x1*)
with *willUse* **show** *?thesis* **by** (*cases x1*) *auto*
next
case (*Cact x2*)
with *willUse* **show** *?thesis* **by** (*cases x2*) *auto*
next
case (*Dact x3*)
with *willUse* **show** *?thesis* **by** (*cases x3*) *auto*
next
case (*Uact x4*)
with *willUse* **show** *?thesis* **by** (*cases x4*) *auto*
next
case (*Ract x5*)
with *willUse* **show** *?thesis*
proof (*cases x5*)
case (*rOPost uid p aid pid*)
with *Ract willUse* **show** *?thesis* **by** (*cases aid* = *AID* ∧ *pid* = *PID*) *auto*
qed *auto*
next
case (*Lact x6*)
with *willUse* **show** *?thesis* **by** (*cases x6*) *auto*
next
case (*COMact x7*)
with *willUse* **show** *?thesis* **by** (*cases x7*) *auto*
qed
qed

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-*Δ*0*:
assumes *B*: *B vl vl1*
shows Δ*0 istate vl istate vl1*
using *assms* **unfolding** Δ*0-def B-def istate-def* **by** *auto*

lemma *unwind-cont-*Δ*0*: *unwind-cont* Δ*0* {Δ*0*,Δ*1*}
proof(*rule*, *simp*)
  let *?*Δ = λ*s vl s1 vl1*. Δ*0 s vl s1 vl1* ∨ Δ*1 s vl s1 vl1*
  fix *s s1* :: *state* **and** *vl vl1* :: *value list*
  assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** Δ*0 s vl s1 vl1*
  hence *rs*: *reach s* **and** *ss1*: *s1* = *s* **and** *l*: *length vl* = *length vl1*
  and *AID*: ¬ *AID* ∈∈ *serverApiIDs s*
  using *reachNT-reach* **unfolding** Δ*0-def* **by** *auto*
  show *iaction ?*Δ *s vl s1 vl1* ∨
    ((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
  proof−
    have *?react* **proof**
      fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
      let *?trn* = *Trans s a ou s'* **let** *?trn1* = *Trans s1 a ou s'*
      assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
      show *match ?*Δ *s s1 vl1 a ou s' vl'* ∨ *ignore ?*Δ *s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
      proof−
        have φ: ¬ φ *?trn* **using** *AID step* **unfolding** φ*-def2* **by** (*auto simp*: *u-defs com-defs*)
        hence *vl'*: *vl'* = *vl* **using** *c* φ **unfolding** *consume-def* **by** *simp*
        have *?match* **proof**(*cases* ∃ *p*. *a* = *COMact* (*comConnectServer AID p*) ∧ *ou* = *outOK*)
          case *True*
          then obtain *p* **where** *a*: *a* = *COMact* (*comConnectServer AID p*) **and** *ou*: *ou* = *outOK* **by** *auto*
          have *AID'*: *AID* ∈∈ *serverApiIDs s'*
          using *step AID* **unfolding** *a ou* **by** (*auto simp*: *com-defs*)
          note *uid* = *reachNT-state*[*OF rsT*]
          show *?thesis* **proof**
            show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
          next
            show *consume ?trn1 vl1 vl1* **using** φ **unfolding** *consume-def ss1* **by** *auto*
          next
            show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
          next
            assume γ *?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
          next
            have Δ*1 s' vl' s' vl1* **using** *l AID' c* **unfolding** *ss1* Δ*1-def vl'* **by** *auto*

thus *?Δ s′ vl′ s′ vl1* **by** *simp*
qed
next
case *False* **note** *a = False*
have *AID′*: ¬ *AID* ∈∈ *serverApiIDs s′*
using *step AID a*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** *φ* **unfolding** *consume-def ss1* **by** *auto*
next
show *γ ?trn = γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **thus** *g ?trn = g ?trn1* **unfolding** *ss1* **by** *simp*
next
have *Δ0 s′ vl′ s′ vl1* **using** *a AID′ l* **unfolding** *Δ0-def vl′ ss1* **by** *simp*
thus *?Δ s′ vl′ s′ vl1* **by** *simp*
qed
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *l* **by** *auto*
qed
qed

lemma *unwind-cont-Δ1*: *unwind-cont Δ1* {*Δ1*}
proof(*rule*, *simp*)
let *?Δ = λs vl s1 vl1. Δ1 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *Δ1 s vl s1 vl1*
hence *rs*: *reach s* **and** *ss1*: *eqButPID s s1*
and *l*: *length vl = length vl1* **and** *AID*: *AID* ∈∈ *serverApiIDs s*
using *reachNT-reach* **unfolding** *Δ1-def* **by** *auto*
have *AID1*: *AID* ∈∈ *serverApiIDs s1* **using** *eqButPID-stateSelectors*[*OF ss1*] *AID* **by** *auto*

show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
proof−
have *?match* **proof**(*cases* ∃ *p pst uid vs. a* = *COMact* (*comReceivePost AID p PID pst uid vs*) ∧ *ou* = *outOK*)
case *True*
then obtain *p pst uid vs* **where**
a: *a* = *COMact* (*comReceivePost AID p PID pst uid vs*) **and** *ou*: *ou* = *outOK* **by** *auto*
have *p*: *p* = *serverPass s AID* **using** *comReceivePost-out*[*OF step a ou*] **.**
have *p1*: *p* = *serverPass s1 AID* **using** *p ss1 eqButPID-stateSelectors* **by** *auto*
have *φ*: *φ ?trn* **using** *a ou step φ-def2* **by** *auto*
obtain *v* **where** *vl*: *vl* = *v* # *vl'* **and** *f*: *f ?trn* = *v*
using *c φ* **unfolding** *consume-def* **by** (*cases vl*) *auto*
have *AID'*: *AID* ∈∈ *serverApiIDs s'* **using** *step AID* **unfolding** *a ou* **by** (*auto simp*: *com-defs*)
note *uid* = *reachNT-state*[*OF rsT*]
obtain *v1 vl1'* **where** *vl1*: *vl1* = *v1* # *vl1'* **using** *l* **unfolding** *vl* **by** (*cases vl1*) *auto*
obtain *pst1* **where** *v1*: *v1* = *PValR pst1* **by** (*cases v1*) *auto*
define *a1* **where** *a1* ≡ *COMact* (*comReceivePost AID p PID pst1 uid vs*)
note *a1* = *this*
obtain *s1'* **where** *step1*: *step s1 a1* = (*outOK*, *s1'*) **using** *AID1* **unfolding** *a1-def p1* **by** (*simp add*: *com-defs*)
have *s's1'*: *eqButPID s' s1'* **using** *comReceivePost-step-eqButPID*[*OF a -step step1 ss1*] *a1* **by** *simp*
let *?trn1* = *Trans s1 a1 outOK s1'*
have *φ1*: *φ ?trn1* **unfolding** *φ-def2* **unfolding** *a1* **by** *auto*
have *f1*: *f ?trn1* = *v1* **unfolding** *a1 v1* **by** *simp*
show *?thesis* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vl1'* **using** *φ1 f1* **unfolding** *consume-def ss1 vl1* **by** *simp*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *a a1* **by** *simp*
next
assume *γ ?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *a a1 ou* **by** *simp*
next
show *Δ1 s' vl' s1' vl1'* **using** *l AID' c s's1'* **unfolding** *Δ1-def vl vl1* **by** *simp*

qed
next
case *False* **note** $a = False$
obtain *s1' ou1* **where** *step1*: $step\ s1\ a = (ou1,\ s1')$ **by** *fastforce*
let $?trn1 = Trans\ s1\ a\ ou1\ s1'$
have φ: $\neg\ \varphi\ ?trn$ **using** *a step* φ*-def2* **by** *auto*
have $\varphi 1$: $\neg\ \varphi\ ?trn1$ **using** φ *ss1 step step1 eqButPID-step-*φ **by** *blast*
have *s's1'*: *eqButPID s' s1'* **using** *ss1 step step1 eqButPID-step* **by** *blast*
have *ouou1*: $\gamma\ ?trn \Longrightarrow ou = ou1$ **using** *eqButPID-step-*γ*-out ss1 step step1 T rs1 rsT* **by** *blast*
have *AID'*: *AID* $\in\in$ *serverApiIDs s'* **using** *AID step rs* **using** *IDs-mono* **by** *auto*
have *vl'*: $vl' = vl$ **using** *c* φ **unfolding** *consume-def* **by** *simp*
show *?thesis* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** $\varphi 1$ **unfolding** *consume-def ss1* **by** *auto*
next
show *1*: $\gamma\ ?trn = \gamma\ ?trn1$ **unfolding** *ss1* **by** *simp*
next
assume $\gamma\ ?trn$ **hence** $ou = ou1$ **using** *ouou1* **by** *auto*
thus $g\ ?trn = g\ ?trn1$ **using** *ouou1* **by** (*cases a*) *auto*
next
show $\Delta 1\ s'\ vl'\ s1'\ vl1$ **using** *a l s's1' AID'* **unfolding** $\Delta 1$*-def vl'* **by** *simp*
qed
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *l* **by** *auto*
qed
qed

definition *Gr* **where**
$Gr =$
$\{$
$(\Delta 0,\ \{\Delta 0,\Delta 1\}),$
$(\Delta 1,\ \{\Delta 1\})$
$\}$

theorem *Post-secure*: *secure*
apply (*rule unwind-decomp-secure-graph*[*of Gr* $\Delta 0$])
unfolding *Gr-def*
apply (*simp*, *smt insert-subset order-refl*)
using *istate-*$\Delta 0$ *unwind-cont-*$\Delta 0$ *unwind-cont-*$\Delta 1$
unfolding *Gr-def* **by** *auto*

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 $\varphi 1 \equiv Iss.\varphi$ **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 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*)
using *assms* **apply**(*cases trn1*) **by** (*auto simp*: *Iss.φ-def2 split*: *prod.splits*)

lemma *isCom1-isComO1*:
assumes *validTrans trn1* **and** *reach* (*srcOf trn1*) **and** *γ1 trn1*
shows *isCom1 trn1* ⟷ *isComO1* (*g1 trn1*)
using *assms* **by** (*cases trn1 rule*: *isCom1.cases*) *auto*

lemma *isCom2-isComV2*:
assumes *validTrans trn2* **and** *reach* (*srcOf trn2*) **and** *φ2 trn2*
shows *isCom2 trn2* ⟷ *isComV2* (*f2 trn2*)
using *assms* **apply**(*cases trn2*) **by** (*auto simp*: *Rcv.φ-def2 split*: *prod.splits*)

lemma *isCom2-isComO2*:
assumes *validTrans trn2* **and** *reach* (*srcOf trn2*) **and** *γ2 trn2*
shows *isCom2 trn2* ⟷ *isComO2* (*g2 trn2*)
using *assms* **by** (*cases trn2 rule*: *isCom2.cases*) *auto*

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*)
using *assms* **apply**(*cases trn1*, *cases trn2*)
by (*auto simp*: *Iss.φ-def2 Rcv.φ-def2 split*: *prod.splits*)

lemma *sync-syncO*:
assumes *validTrans trn1* **and** *reach* (*srcOf trn1*)
and *validTrans trn2* **and** *reach* (*srcOf trn2*)
and *isCom1 trn1* **and** *isCom2 trn2* **and** *γ1 trn1* **and** *γ2 trn2*
and *sync trn1 trn2*
shows *syncO* (*g1 trn1*) (*g2 trn2*)
proof(*cases trn1*)
  case (*Trans s1 a1 ou1 s1'*) **note** *trn1* = *Trans*
  show *?thesis* **proof**(*cases trn2*)
    case (*Trans s2 a2 ou2 s2'*) **note** *trn2* = *Trans*

```
    show ?thesis
    proof(cases a1)
      case (COMact ca1) note a1 = COMact
      show ?thesis
      proof(cases a2)
        case (COMact ca2) note a2 = COMact
        show ?thesis
        using assms unfolding trn1 trn2 a1 a2
        apply(cases ca1) by (cases ca2, auto split: prod.splits)+
      qed(insert assms, unfold trn1 trn2, auto)
    qed(insert assms, unfold trn1 trn2, auto)
  qed
qed

lemma sync-φ1-φ2:
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 φ1 trn1 ⟷ φ2 trn2 (is ?A ⟷ ?B)
proof(cases trn1)
  case (Trans s1 a1 ou1 s1') note trn1 = Trans
  hence step1: step s1 a1 = (ou1,s1') using v1 by auto
  show ?thesis proof(cases trn2)
    case (Trans s2 a2 ou2 s2') note trn2 = Trans
    hence step2: step s2 a2 = (ou2,s2') using v2 by auto
    show ?thesis
    proof(cases a1)
      case (COMact ca1) note a1 = COMact
      show ?thesis
      proof(cases a2)
        case (COMact ca2) note a2 = COMact

        have ?A ⟷ (∃ aid1. ca1 =
             (comSendPost (admin s1) (pass s1 (admin s1)) aid1
               PID) ∧
            ou1 =
            O-sendPost
             (aid1, clientPass s1 aid1, PID, post s1 PID,
              owner s1 PID, vis s1 PID))
        using c1 unfolding trn1 Iss.φ-def3[OF step1] unfolding a1 by auto
        also have ... ⟷ (∃ uid2 pst2 vs2.
         ca2 = comReceivePost AID1 (serverPass s2 AID1) PID pst2 uid2 vs2 ∧
ou2 = outOK)
        using sn step1 step2 unfolding trn1 trn2 a1 a2
        apply(cases ca1) by (cases ca2, auto simp: all-defs)+
        also have ... ⟷ ?B
        using c2 unfolding trn2 Rcv.φ-def3[OF step2] unfolding a2 by auto
        finally show ?thesis .
```

qed(*insert assms*, *unfold trn1 trn2*, *auto*)
qed(*insert assms*, *unfold trn1 trn2*, *auto*)
qed
qed

lemma *textPost-textPost-cong*[*intro*]:
assumes *textPost pst1* = *textPost pst2*
and *setTextPost pst1 emptyText* = *setTextPost pst2 emptyText*
shows *pst1* = *pst2*
using *assms* **by** (*cases pst1*, *cases pst2*) *auto*

lemma *sync*-φ-γ:
assumes *validTrans trn1* **and** *reach* (*srcOf trn1*)
and *validTrans trn2* **and** *reach* (*srcOf trn2*)
and *isCom1 trn1* **and** *isCom2 trn2*
and γ*1 trn1* **and** γ*2 trn2*
and *so*: *syncO* (*g1 trn1*) (*g2 trn2*)
and φ*1 trn1* $\Longrightarrow$ φ*2 trn2* $\Longrightarrow$ *syncV* (*f1 trn1*) (*f2 trn2*)
shows *sync trn1 trn2*
proof(*cases trn1*, *cases trn2*)
fix *s1 a1 ou1 s1' s2 a2 ou2 s2'*
assume *trn1*: *trn1* = *Trans s1 a1 ou1 s1'*
and *trn2*: *trn2* = *Trans s2 a2 ou2 s2'*
hence *step1*: *step s1 a1* = (*ou1*,*s1'*) **and** *step2*: *step s2 a2* = (*ou2*,*s2'*) **using** *assms* **by** *auto*
show *?thesis*
proof(*cases a1*)
case (*COMact ca1*) **note** *a1* = *COMact*
show *?thesis*
proof(*cases a2*)
case (*COMact ca2*) **note** *a2* = *COMact*
show *?thesis*
proof(*cases ca1*) **term** *comReceivePost*
case (*comSendPost uid1 p1 aid1 pid*) **note** *ca1* = *comSendPost*
then obtain *pst* **where** *p1*: *p1* = *pass s1* (*admin s1*) **and**
aid1: *aid1* = *AID2* **and** *ou2*: *ou2* = *outOK* **and** *ou1*: *ou1* ≠ *outErr* **and**
ca2: *ca2* = *comReceivePost AID1* (*serverPass s2 AID1*) *pid pst* (*owner s1 pid*) (*vis s1 pid*)
using *so step1 step2* **unfolding** *trn1 trn2 a1 a2 ca1*
by (*cases ca2*, *auto simp*: *all-defs*)
have *ou1*: *ou1* = *O-sendPost* (*AID2*,*clientPass s1 AID2*,*pid*, *post s1 pid*, *owner s1 pid*, *vis s1 pid*)
using *step1 ou1* **unfolding** *a1 ca1 aid1* **by** (*auto simp*: *all-defs*)
show *?thesis* **proof**(*cases pid* = *PID*)
case *False* **thus** *?thesis* **using** *so step1 step2* **unfolding** *trn1 trn2 a1 a2 ca1 ca2*
by (*auto simp*: *all-defs*)
next
case *True* **note** *pid* = *True*

hence *φ1 trn1* ∧ *φ2 trn2* **using** *ou1 ou2* **unfolding** *trn1 trn2 a1 a2 ca1 ca2* **by** *auto*
hence *syncV* (*f1 trn1*) (*f2 trn2*) **using** *assms* **by** *simp*
hence *pst*: *pst* = *post s1 PID* **using** *pid* **unfolding** *trn1 trn2 a1 a2 ca1 ca2 aid1 ou1* **by** *auto*
show *?thesis* **unfolding** *trn1 trn2 a1 a2 ca1 ca2 ou1 ou2 pst pid* **by** *auto*
qed
qed(*insert so step1 step2*, *unfold trn1 trn2 a1 a2*, (*cases ca2*, *auto simp*: *all-defs*)+)
qed(*insert assms*, *unfold trn1 trn2*, *auto*)
qed(*insert assms*, *unfold trn1 trn2*, *auto*)
qed

lemma *isCom1-γ1*:
assumes *validTrans trn1* **and** *reach* (*srcOf trn1*) **and** *isCom1 trn1*
shows *γ1 trn1*
proof(*cases trn1*)
case (*Trans s1 a1 ou1 s1'*)
thus *?thesis* **using** *assms* **by** (*cases a1*) *auto*
qed

lemma *isCom2-γ2*:
assumes *validTrans trn2* **and** *reach* (*srcOf trn2*) **and** *isCom2 trn2*
shows *γ2 trn2*
proof(*cases trn2*)
case (*Trans s2 a2 ou2 s2'*)
thus *?thesis* **using** *assms* **by** (*cases a2*) *auto*
qed

lemma *isCom2-V2*:
assumes *validTrans trn2* **and** *reach* (*srcOf trn2*) **and** *φ2 trn2*
shows *isCom2 trn2*
proof(*cases trn2*)
case (*Trans s2 a2 ou2 s2'*) **note** *trn2* = *Trans*
show *?thesis*
proof(*cases a2*)
case (*COMact ca2*)
thus *?thesis* **using** *assms trn2* **by** (*cases ca2*) *auto*
qed(*insert assms trn2*, *auto*)
qed

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*

apply *standard*
using *isCom1-isComV1 isCom1-isComO1 isCom2-isComV2 isCom2-isComO2 sync-syncV sync-syncO*
apply *auto*
apply (*meson sync-φ1-φ2*, *meson sync-φ1-φ2*)
using *sync-φ-γ* **apply** *auto*
using *isCom1-γ1 isCom2-γ2 isCom2-V2* **apply** *auto*
by (*meson isCom2-V2*)

context *Post-COMPOSE2*
begin

theorem *secure*: *secure*
using *secure1-secure2-secure*[*OF Iss.Post-secure Rcv.Post-secure*] **.**

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 (*cases v*)
case (*Inl vl*) **then show** *thesis* **using** *PVal PValS* **by** (*cases vl rule*: *Post-ISSUER.value.exhaust*)

auto **next**
case (*Inr vr*) **then show** *thesis* **using** *PValR* **by** (*cases vr rule*: *Post-RECEIVER.value.exhaust*) *auto*
qed

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* **.**

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*))
unfolding *syncV-def* **by** *auto*

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*)
using *assms* **unfolding** *syncV-def* **by** *auto*

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*)
by (*cases v rule*: *value-cases*; *auto*)+

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*
using *AID-in-AIDs* **proof** (*unfold-locales*, *goal-cases*)
case (*1 nid trn*) **then show** *?case* **by** (*cases trn*) (*auto simp*: *φ-defs split*: *prod.splits*) **next**
case (*2 nid trn*) **then show** *?case* **by** (*cases trn*) (*auto simp*: *φ-defs split*: *prod.splits*) **next**
case (*3 nid trn*)
interpret *Sink*: *Post-RECEIVER UIDs nid PID AID* **.**
show *?case* **using** *3* **by** (*cases* (*nid*,*trn*) *rule*: *tgtNodeOf.cases*) (*auto split*: *prod.splits*) **next**
case (*4 nid trn*)
interpret *Sink*: *Post-RECEIVER UIDs nid PID AID* **.**

show *?case* **using** *4* **by** (*cases* (*nid*,*trn*) *rule*: *tgtNodeOf.cases*) (*auto split*: *prod.splits*) **next**
case (*5 nid1 trn1 nid2 trn2*)
interpret *Sink1*: *Post-RECEIVER UIDs nid1 PID AID* **.**
interpret *Sink2*: *Post-RECEIVER UIDs nid2 PID AID* **.**
show *?case* **using** *5* **by** (*elim sync-cases*) (*auto intro*: *syncVI*) **next**
case (*6 nid1 trn1 nid2 trn2*)
interpret *Sink1*: *Post-RECEIVER UIDs nid1 PID AID* **.**
interpret *Sink2*: *Post-RECEIVER UIDs nid2 PID AID* **.**
show *?case* **using** *6* **by** (*elim sync-cases*) *auto* **next**
case (*7 nid1 trn1 nid2 trn2*)
interpret *Sink1*: *Post-RECEIVER UIDs nid1 PID AID* **.**
interpret *Sink2*: *Post-RECEIVER UIDs nid2 PID AID* **.**
show *?case* **using** *7* **by** (*elim sync-cases*) (*auto split*: *prod.splits*, *auto simp*: *sendPost-def*) **next**
case (*8 nid1 trn1 nid2 trn2*)
interpret *Sink1*: *Post-RECEIVER UIDs nid1 PID AID* **.**
interpret *Sink2*: *Post-RECEIVER UIDs nid2 PID AID* **.**
show *?case* **using** *8*
apply (*elim syncO-cases*; *cases trn1*; *cases trn2*)
apply (*auto simp*: *Iss.g-simps ObservationSetup-RECEIVER.g-simps split*: *prod.splits*)
apply (*auto simp*: *sendPost-def split*: *prod.splits elim*: *syncVE*)[]
done next
case (*9 nid trn*)
then show *?case*
by (*cases* (*nid*,*trn*) *rule*: *tgtNodeOf.cases*)
(*auto simp*: *ObservationSetup-RECEIVER.γ.simps*) **next**
case (*10 nid trn*) **then show** *?case* **by** (*cases trn*) (*auto simp*: *φ-defs*) **next**
case (*11 vSrc nid vn*) **then show** *?case* **by** (*cases vSrc rule*: *value-cases*) (*auto simp*: *syncV-def*) **next**
case (*12 vSrc nid vn*) **then show** *?case* **by** (*cases vSrc rule*: *value-cases*) (*auto simp*: *syncV-def*)
qed

lemma *list-all-Not-isl-projectSrcV*: *list-all* (*Not o isl*) (*Net.projectSrcV aid vlSrc*)
proof (*induction vlSrc*)
case (*Cons vSrc vlSrc′*) **then show** *?case* **by** (*cases vSrc rule*: *value-cases*) *auto*
qed *auto*

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

interpretation *Sink*: *Post-RECEIVER UIDs AID′ PID AID* **by** *unfold-locales*

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*))
using *assms* **proof** (*induction vl vl1 rule*: *list22-induct*)
case (*ConsCons v vl v1 vl1*)
consider (*SendSend*) *aid pst pst1* **where** *v* = *Inl* (*Iss.PValS aid pst*) *v1* = *Inl* (*Iss.PValS aid pst1*)
| (*Internal*) *comOfV AID v* = *Internal* ¬*Iss.isPValS* (*projl v*)
| (*Internal1*) *comOfV AID v1* = *Internal* ¬*Iss.isPValS* (*projl v1*)
using *ConsCons*(*4*−*6*) **by** (*cases v rule*: *value-cases*; *cases v1 rule*: *value-cases*) *auto*
then show *?case* **proof** *cases*
case (*SendSend*) **then show** *?thesis* **using** *ConsCons.IH*(*1*) *ConsCons.prems* **by** *auto*
next
case (*Internal*) **then show** *?thesis* **using** *ConsCons.IH*(*2*)[*of v1* # *vl1*] *ConsCons.prems* **by** *auto*
next
case (*Internal1*) **then show** *?thesis* **using** *ConsCons.IH*(*3*)[*of v* # *vl*] *ConsCons.prems* **by** *auto*
qed
qed (*auto simp*: *comOfV-AID*)

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*))
using *assms Source-B-Sink-B-aux* **by** (*auto simp*: *Iss.B-def Sink.B-def*)

end

lemma *map-projl-Inl*: *map* (*projl o Inl*) *vl* = *vl*
by (*induction vl*) *auto*

lemma *these-map-Inl-projl*: *list-all isl vl* ⟹ *these* (*map* (*Some o Inl o projl*) *vl*) = *vl*
by (*induction vl*) *auto*

lemma *map-projr-Inr*: *map* (*projr o Inr*) *vl* = *vl*
by (*induction vl*) *auto*

lemma *these-map-Inr-projr*: *list-all* (*Not o isl*) *vl* ⟹ *these* (*map* (*Some o Inr o projr*) *vl*) = *vl*
by (*induction vl*) *auto*

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 (*unfold-locales*, *goal-cases*)
case *1* **show** *?case* **using** *AID-in-AIDs* **. next**
case *2*
interpret *Iss′*: *BD-Security-TS-Trans*
istate System-Specification.validTrans srcOf tgtOf Iss.φ Iss.f Iss.γ Iss.g Iss.T Iss.B
istate System-Specification.validTrans srcOf tgtOf Iss.φ λtrn. Inl (*Iss.f trn*) *Iss.γ Iss.g Iss.T B AID*
id id Some Some o Inl
proof (*unfold-locales*, *goal-cases*)
case (*11 vl′ vl1′ tr*) **then show** *?case*
by (*intro exI*[*of - map projl vl1′*]) (*auto simp*: *map-projl-Inl these-map-Inl-projl*)
qed *auto*
show *?case* **using** *Iss.Post-secure Iss′.translate-secure* **by** *auto*
next
case (*3 aid tr vl′ vl1*)
then show *?case*
using *Source-B-Sink-B*[*of aid* (*Net.lV AID tr*) *vl1*] *list-all-Not-isl-projectSrcV*
by *auto*
qed

theorem *secure*: *secure*
proof (*intro preserve-source-secure ballI*)
fix *aid*
assume *aid*: *aid* ∈ *AIDs* − {*AID*}
interpret *Node*: *Post-RECEIVER UIDs aid PID AID* **.**
interpret *Node′*: *BD-Security-TS-Trans*
istate System-Specification.validTrans srcOf tgtOf Node.φ Node.f Node.γ Node.g Node.T Node.B
istate System-Specification.validTrans srcOf tgtOf Node.φ λtrn. Inr (*Node.f trn*) *Node.γ Node.g Node.T B aid*
id id Some Some o Inr
proof (*unfold-locales*, *goal-cases*)
case (*11 vl′ vl1′ tr*) **then show** *?case* **using** *aid*
by (*intro exI*[*of - map projr vl1′*]) (*auto simp*: *map-projr-Inr these-map-Inr-projr*)
qed *auto*
show *Net.lsecure aid*
using *aid Node.Post-secure Node′.translate-secure* **by** *auto*
qed

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* **.**

lemma *eqButPID-open*:
assumes *eqButPID s s1*
shows *open s* ⟷ *open s1*
using *eqButPID-stateSelectors*[*OF assms*]
unfolding *open-def* **by** *auto*

lemma *not-open-eqButPID*:
assumes *1*: ¬ *open s* **and** *2*: *eqButPID s s1*
shows ¬ *open s1*
using *1* **unfolding** *eqButPID-open*[*OF 2*] **.**

lemma *step-isCOMact-open*:

assumes *step s a* = (*ou*, *s'*)
and *isCOMact a*
shows *open s'* = *open s*
using *assms* **proof** (*cases a*)
  case (*COMact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *open-def com-defs*)
qed *auto*

lemma *validTrans-isCOMact-open*:
assumes *validTrans trn*
and *isCOMact* (*actOf trn*)
shows *open* (*tgtOf trn*) = *open* (*srcOf trn*)
using *assms step-isCOMact-open* **by** (*cases trn*) *auto*

lemma *list-all-isOVal-filter-isPValS*:
list-all isOVal vl ⟹ *filter* (*Not o isPValS*) *vl* = *vl*
by (*induct vl*) *auto*

lemma *list-all-Not-isOVal-OVal-True*:
assumes *list-all* (*Not* ∘ *isOVal*) *ul*
and *ul* @ *vll* = *OVal True* # *vll'*
shows *ul* = []
using *assms* **by** (*cases ul*) *auto*

lemma *list-all-filter-isOVal-isPVal-isPValS*:
assumes *list-all* (*Not* ∘ *isOVal*) *ul*
and *filter isPValS ul* = [] **and** *filter isPVal ul* = []
shows *ul* = []
using *assms value.exhaust-disc* **by** (*induct ul*) *auto*

lemma *filter-list-all-isPValS-isOVal*:
assumes *list-all* (*Not* ∘ *isOVal*) *ul* **and** *filter isPVal ul* = []
shows *list-all isPValS ul*
using *assms value.exhaust-disc* **by** (*induct ul*) *auto*

lemma *filter-list-all-isPVal-isOVal*:
assumes *list-all* (*Not* ∘ *isOVal*) *ul* **and** *filter isPValS ul* = []
shows *list-all isPVal ul*
using *assms value.exhaust-disc* **by** (*induct ul*) *auto*

lemma *list-all-isPValS-Not-isOVal-filter*:
assumes *list-all isPValS ul* **shows** *list-all* (*Not* ∘ *isOVal*) *ul* ∧ *filter isPVal ul* = []
using *assms value.exhaust-disc* **by** (*induct ul*) *auto*

lemma *filter-isTValS-Nil*:
filter isPValS vl = [] ⟷

list-all (λ *v*. *isPVal v* ∨ *isOVal v*) *vl*
proof(*induct vl*)
case (*Cons v vl*)
thus *?case* **by** (*cases v*) *auto*
qed *auto*

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'*) ⟷
(∃ *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'*
using *assms* **by** (*cases Trans s a ou s' rule*: *φ.cases*) (*auto simp*: *all-defs open-def*)

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*
using *1 2* **unfolding** *a* **by** (*auto simp*: *u-defs*)

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*)
using *1 2* **unfolding** *a* **by** (*auto simp*: *com-defs*)

lemma *step-open-isCOMact*:
assumes *step s a* = (*ou*,*s'*)
and *open s* ≠ *open s'*
shows ¬ *isCOMact a* ∧ ¬ (∃ *ua*. *isuPost ua* ∧ *a* = *Uact ua*)
using *assms* **unfolding** *open-def*
apply(*cases a*)
subgoal by (*auto simp*: *all-defs*)
subgoal by (*auto simp*: *all-defs*)
subgoal by (*auto simp*: *all-defs*)
subgoal for *x4* **by** (*cases x4*) (*auto simp*: *all-defs*)

subgoal by (*auto simp*: *all-defs*)
subgoal by (*auto simp*: *all-defs*)
subgoal for *x7* **by** (*cases x7*) (*auto simp*: *all-defs*)
done

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*)
unfolding *φ-def2*[*OF assms*]
using *comSendPost-out*[*OF assms*] *uTextPost-out*[*OF assms*]
step-open-isCOMact[*OF assms*]
by *blast*

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'*)
using *assms* **by** (*cases Trans s a ou s' rule*: *f.cases*) *auto*

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*)
using *assms* **by** (*cases Trans s a ou s' rule*: *f.cases*) (*auto simp*: *u-defs com-defs*)

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*)
using *assms* **by** (*cases Trans s a ou s' rule*: *f.cases*) (*auto simp*: *com-defs*)

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*
using *assms* **by** (*auto simp*: *φ-def2 com-defs*)

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*
using *assms a* **unfolding** *open-def*
by (*cases a*) (*auto simp*: *u-defs com-defs*)

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−
  have *a*: *a* = *Uact* (*uPost* (*owner s PID*) (*pass s* (*owner s PID*)) *PID pst*)
  using *f-eq-PVal*[*OF step fi f*] **.**
  have *ou*: *ou* = *outOK* **using** *fi op* **unfolding** *a φ-def2*[*OF step*] **by** *auto*
  have *owner s PID* ∈∈ *userIDs s* **using** *s* **by** (*simp add*: *PID reach-owner-userIDs*)
  hence *owner s PID* ∉ *UIDs* **using** *op PID* **unfolding** *open-def* **by** *auto*
  thus *?thesis* **unfolding** *a* **by** *simp*
qed

lemma *Uact-uPaperC-step-eqButPID*:
assumes *a*: *a* = *Uact* (*uPost uid p PID pst*)
and *step s a* = (*ou*,*s'*)
shows *eqButPID s s'*
using *assms* **unfolding** *eqButPID-def eeqButPID-def eeqButPID-F-def*
by (*auto simp*: *all-defs split*: *if-splits*)

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−
  have *s's1'*: *eqButPID s' s1'*
  using *eqButPID-step local.step ss1 step1* **by** *blast*
  show *?thesis* **using** *step step1 φ eqButPID-open*[*OF ss1*] *eqButPID-open*[*OF s's1'*]
  using *eqButPID-stateSelectors*[*OF ss1*]

unfolding *φ-def2*[*OF step*] *φ-def2*[*OF step1*]
by (*auto simp*: *all-defs*)
qed

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′*)
by (*metis eqButPID-step-φ-imp eqButPID-sym assms*)

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 = []
using assms value.exhaust-disc by (induct ul) auto

lemma BC-not-Nil: BC vl vl1 ⟹ vl = [] ⟹ vl1 = []
by(auto elim: BC.cases)

lemma BC-OVal-True:
assumes BC (OVal True # vl′) vl1
shows ∃ vl1′. BO vl′ vl1′ ∧ vl1 = OVal True # vl1′
proof−
  define vl where vl: vl ≡ OVal True # vl′
  have BC vl vl1 using assms unfolding vl by auto
  thus ?thesis proof cases
    case (BC-BO vll vll1 ul ul1 sul)
```

hence *ul*: *ul* = [] **unfolding** *vl* **apply** *simp*
by (*metis* (*no-types*) *Post.value.disc*(*9*) *append-eq-Cons-conv*
list.map(*2*) *list.pred-inject*(*2*) *list-all-map*)
have *sul*: *sul* = [] **using** *BC-BO* **unfolding** *vl ul* **apply** *simp*
by (*metis Post.value.disc*(*6*) *append-eq-Cons-conv list.pred-inject*(*2*))
show *?thesis*
apply − **apply**(*rule exI*[*of - vll1*])
using *BC-BO* **using** *list-all-filter-Not-isOVal*[*of ul1*]
unfolding *ul sul vl* **by** *auto*
qed(*unfold vl*, *auto*)
qed

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* = []
using *B Post.BC-not-Nil Post.B-def vl* **by** *blast*

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*
done

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*
using *assms reach-vis* **by** *auto*

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−
note [*simp*] = *all-defs*
open-def
note *s* = *reachNT-reach*[*OF sT*]
note *willUse* =
step step1 γ
not-open-eqButPID[*OF op ss1*]
reach-vis[*OF s*]
eqButPID-stateSelectors[*OF ss1*]
eqButPID-actions[*OF ss1*]
eqButPID-update[*OF ss1*] *eqButPID-not-PID*[*OF ss1*]

eqButPID-eqButF[*OF ss1*]
eqButPID-setShared[*OF ss1*] *eqButPID-updateShared*[*OF ss1*]
eeqButPID-F-not-PID eqButPID-not-PID-sharedWith
eqButPID-insert2[*OF ss1*]
show *?thesis*
proof (*cases a*)
case (*Sact x1*)
with *willUse* **show** *?thesis* **by** (*cases x1*) *auto*
next
case (*Cact x2*)
with *willUse* **show** *?thesis* **by** (*cases x2*) *auto*
next
case (*Dact x3*)
with *willUse* **show** *?thesis* **by** (*cases x3*) *auto*
next
case (*Uact x4*)
with *willUse* **show** *?thesis* **by** (*cases x4*) *auto*
next
case (*Ract x5*)
with *willUse* **show** *?thesis*
proof (*cases x5*)
case (*rPost uid p pid*)
with *Ract willUse* **show** *?thesis* **by** (*cases pid* = *PID*) *auto*
qed *auto*
next
case (*Lact x6*)
with *willUse* **show** *?thesis*
proof (*cases x6*)
case (*lClientsPost uid p pid*)
with *Lact willUse* **show** *?thesis* **by** (*cases pid* = *PID*) *auto*

qed *auto*
next
case (*COMact x7*)
with *willUse* **show** *?thesis* **by** (*cases x7*) *auto*
qed
qed

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'*
using *ss1 step step1*
using *eqButPID-stateSelectors*[*OF ss1*]
eqButPID-update[*OF ss1*] *eqButPID-setShared*[*OF ss1*]
unfolding *a* **by** (*auto simp*: *u-defs*)

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**

$\Delta 31$ *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* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 32$ *s vl s1 vl1* ≡
PID ∈∈ *postIDs s* ∧
(∃ *sul vll vll1*.
BO vll vll1 ∧
list-all isPValS sul ∧
vl = *sul* @ *OVal True* # *vll* ∧ *vl1* = *sul* @ *OVal True* # *vll1*) ∧
s = *s1* ∧ ¬ *open s* ∧
corrFrom (*post s1 PID*) *vl1*

definition $\Delta 4$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 4$ *s vl s1 vl1* ≡
PID ∈∈ *postIDs s* ∧
(∃ *ul vll vll1*.
BC vll vll1 ∧
list-all (*Not o isOVal*) *ul* ∧
vl = *ul* @ *OVal False* # *vll* ∧ *vl1* = *ul* @ *OVal False* # *vll1*) ∧
s = *s1* ∧ *open s* ∧
corrFrom (*post s1 PID*) *vl1*

lemma *istate-*$\Delta 0$:
assumes *B*: *B vl vl1*
shows $\Delta 0$ *istate vl istate vl1*
using *assms* **unfolding** $\Delta 0$*-def istate-def B-def* **by** *auto*

lemma *list-all-filter*[*simp*]:
assumes *list-all PP xs*
shows *filter PP xs* = *xs*
using *assms* **by** (*induct xs*) *auto*

lemma *unwind-cont-*$\Delta 0$: *unwind-cont* $\Delta 0$ {$\Delta 0$,$\Delta 1$,$\Delta 2$,$\Delta 31$,$\Delta 32$,$\Delta 4$}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. $\Delta 0$ *s vl s1 vl1* ∨
$\Delta 1$ *s vl s1 vl1* ∨ $\Delta 2$ *s vl s1 vl1* ∨
$\Delta 31$ *s vl s1 vl1* ∨ $\Delta 32$ *s vl s1 vl1* ∨ $\Delta 4$ *s vl s1 vl1*

fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** $\Delta 0$ *s vl s1 vl1*
hence *rs*: *reach s* **and** *ss1*: *s1* = *s* **and** *BC*: *BC vl vl1* **and** *PID*: ¬ *PID* ∈∈ *postIDs s*
and *cor1*: *corr vl1* **using** *reachNT-reach* **unfolding** Δ*0-def* **by** *auto*
have *vis*: *vis s PID* = *FriendV* **using** *reach-not-postIDs-friendV*[*OF rs PID*] **.**
have *pPID*: *post s1 PID* = *emptyPost* **by** (*simp add*: *PID reach-not-postIDs-emptyPost rs ss1*)
have *vlvl1*: *vl* = [] ⟹ *vl1* = [] **using** *BC-not-Nil BC* **by** *auto*
have *op*: ¬ *open s* **using** *PID* **unfolding** *open-def* **by** *auto*
show *iaction ?*Δ *s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'* **let** *?trn1* = *Trans s1 a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
hence *pPID'*: *post s' PID* = *emptyPost*
using *step pPID ss1 PID*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by (*fastforce simp*: *d-defs*)
subgoal by (*fastforce simp*: *d-defs*)
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
show *match ?*Δ *s s1 vl1 a ou s' vl'* ∨ *ignore ?*Δ *s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
proof−
have *?match*
proof(*cases* ∃ *uid p*. *a* = *Cact* (*cPost uid p PID*) ∧ *ou* = *outOK*)
case *True*
then obtain *uid p* **where** *a*: *a* = *Cact* (*cPost uid p PID*) **and** *ou*: *ou* = *outOK* **by** *auto*
have *PID'*: *PID* ∈∈ *postIDs s'*
using *step PID* **unfolding** *a ou* **by** (*auto simp*: *c-defs*)
show *?thesis* **proof**(*cases*
∃ *uid'* ∈ *UIDs*. *uid'* ∈∈ *userIDs s* ∧
(*uid'* = *admin s* ∨ *uid'* = *uid* ∨ *uid'* ∈∈ *friendIDs s uid*))
case *True* **note** *uid* = *True*
have *op'*: *open s'* **using** *uid* **using** *step PID'* **unfolding** *a ou* **by** (*auto simp*: *c-defs open-def*)
have φ: φ *?trn* **using** *op op'* **unfolding** φ*-def2*[*OF step*] **by** *simp*
then obtain *v* **where** *vl*: *vl* = *v* # *vl'* **and** *f*: *f ?trn* = *v*
using *c* **unfolding** *consume-def* φ*-def2* **by**(*cases vl*) *auto*
have *v*: *v* = *OVal True* **using** *f op op'* **unfolding** *a* **by** *simp*
then obtain *ul1 vl1'* **where** *BO'*: *BO vl' vl1'* **and** *vl1*: *vl1* = *ul1* @

```
OVal True # vl1'
          and ul1: list-all (Not ∘ isOVal) ul1
          using BC-OVal-True[OF BC[unfolded vl v]] by auto
          have ul1: ul1 = []
           using BC BC-OVal-True list-all-Not-isOVal-OVal-True ul1 v vl vl1 by
blast
          hence vl1: vl1 = OVal True # vl1' using vl1 by simp
          show ?thesis proof
            show validTrans ?trn1 unfolding ss1 using step by simp
          next
            show consume ?trn1 vl1 vl1' using φ f unfolding vl1 v consume-def
ss1 by simp
          next
            show γ ?trn = γ ?trn1 unfolding ss1 by simp
          next
            assume γ ?trn thus g ?trn = g ?trn1 unfolding ss1 by simp
          next
            show ?Δ s' vl' s' vl1' using BO' proof(cases rule: BO.cases)
              case (BO-PVal)
               hence Δ2 s' vl' s' vl1' using PID' op' cor1 unfolding Δ2-def vl1
pPID' by auto
              thus ?thesis by simp
            next
              case (BO-BC vll vll1 textl)
               hence Δ4 s' vl' s' vl1' using PID' op' cor1 unfolding Δ4-def vl1
pPID' by auto
              thus ?thesis by simp
            qed
          qed
        next
          case False note uid = False
         have op': ¬ open s' using step op uid vis unfolding open-def a by (auto
simp: c-defs)
          have φ: ¬ φ ?trn using op op' a unfolding φ-def2[OF step] by auto
          hence vl': vl' = vl using c unfolding consume-def by simp
          show ?thesis proof
            show validTrans ?trn1 unfolding ss1 using step by simp
          next
            show consume ?trn1 vl1 vl1 using φ unfolding consume-def ss1 by
auto
          next
            show γ ?trn = γ ?trn1 unfolding ss1 by simp
          next
            assume γ ?trn thus g ?trn = g ?trn1 unfolding ss1 by simp
          next
            show ?Δ s' vl' s' vl1 using BC proof(cases rule: BC.cases)
              case (BC-PVal)
                hence Δ1 s' vl' s' vl1 using PID' op' cor1 unfolding Δ1-def vl'
pPID' by auto
```

```
      thus ?thesis by simp
    next
      case (BC-BO vll vll1 ul ul1 sul)
      show ?thesis
      proof(cases ul ≠ [] ∧ ul1 ≠ [])
        case True
        hence Δ31 s' vl' s' vl1 using BC-BO PID' op' cor1
        unfolding Δ31-def vl' pPID' apply auto
        apply (rule exI[of - ul]) apply (rule exI[of - ul1])
        apply (rule exI[of - sul])
        apply (rule exI[of - vll]) apply (rule exI[of - vll1])
        by auto
        thus ?thesis by simp
      next
        case False
        hence 0: ul = [] ∧ ul1 = [] using BC-BO by simp
        hence 1: list-all isPValS ul ∧ list-all isPValS ul1
        using ‹list-all (Not ∘ isOVal) ul› ‹list-all (Not ∘ isOVal) ul1›
        using filter-list-all-isPValS-isOVal by auto

        have Δ32 s' vl' s' vl1 using BC-BO PID' op' cor1 0 1
        unfolding Δ32-def vl' pPID' apply simp
        apply(rule exI[of - sul])
        apply(rule exI[of - vll]) apply(rule exI[of - vll1])
        by auto
        thus ?thesis by simp
      qed
    qed
  qed
qed
next
  case False note a = False
  have op': ¬ open s'
    using a step PID op unfolding open-def
    apply(cases a)
    subgoal for x1 apply(cases x1) apply(fastforce simp: s-defs)+ .
    subgoal for x2 apply(cases x2) apply(fastforce simp: c-defs)+ .
    subgoal for x3 apply(cases x3) apply(fastforce simp: d-defs)+ .
    subgoal for x4 apply(cases x4) apply(fastforce simp: u-defs)+ .
    subgoal by (fastforce simp: u-defs)
    subgoal by (fastforce simp: u-defs)
    subgoal for x7 apply(cases x7) apply(fastforce simp: com-defs)+ .
    done
  have φ: ¬ φ ?trn using PID step op op' unfolding φ-def2[OF step]
  by (auto simp: u-defs com-defs)
  hence vl': vl' = vl using c unfolding consume-def by simp
  have PID': ¬ PID ∈∈ postIDs s'
    using step PID a
    apply(cases a)
```

subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by (*fastforce simp*: *u-defs*)
subgoal by (*fastforce simp*: *u-defs*)
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** φ **unfolding** *consume-def ss1* **by** *auto*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
have Δ*0 s′ vl′ s′ vl1* **using** *a BC PID′ cor1* **unfolding** Δ*0-def vl′* **by** *simp*
thus *?*Δ *s′ vl′ s′ vl1* **by** *simp*
qed
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vlvl1* **by** *simp*
qed
qed

lemma *unwind-cont-*Δ*1*: *unwind-cont* Δ*1* {Δ*1*,Δ*11*}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. Δ*1 s vl s1 vl1* $\vee$ Δ*11 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** Δ*1 s vl s1 vl1*
then obtain
lvl: *list-all* (*Not* ∘ *isOVal*) *vl* **and** *lvl1*: *list-all* (*Not* ∘ *isOVal*) *vl1*
and *map*: *map tgtAPI* (*filter isPValS vl*) = *map tgtAPI* (*filter isPValS vl1*)
and *rs*: *reach s* **and** *ss1*: *eqButPID s s1* **and** *op*: ¬ *open s* **and** *PID*: *PID* ∈∈ *postIDs s*
and *vlvl1*: *vl* = [] ⟹ *vl1* = [] **and** *cor1*: *corrFrom* (*post s1 PID*) *vl1*
using *reachNT-reach* **unfolding** Δ*1-def* **by** *auto*
have *PID1*: *PID* ∈∈ *postIDs s1* **using** *eqButPID-stateSelectors*[*OF ss1*] *PID* **by** *auto*
have *own*: *owner s PID* ∈ *set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
hence *own1*: *owner s1 PID* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
have *adm*: *admin s* ∈ *set* (*userIDs s*) **using** *reach-admin-userIDs*[*OF rs own*] **.**

hence *adm1*: *admin s1* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
have *op1*: ¬ *open s1* **using** *op ss1 eqButPID-open* **by** *auto*
show *iaction ?*Δ *s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof(*cases vl1*)
case (*Cons v1 vll1*) **note** *vl1* = *Cons*
show *?thesis* **proof**(*cases v1*)
case (*PVal pst1*) **note** *v1* = *PVal*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
define *a1* **where** *a1*: *a1* ≡ *Uact* (*uPost uid p PID pst1*)
have *uid1*: *uid* = *owner s1 PID* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid p*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a1* = (*ou1*, *s1'*) **by**(*cases step s1 a1*) *auto*
have *ou1*: *ou1* = *outOK* **using** *step1 PID1 own1* **unfolding** *a1 uid1 p1* **by** (*auto simp*: *u-defs*)
have *op1'*: ¬ *open s1'* **using** *step1 op1* **unfolding** *a1 ou1 open-def* **by** (*auto simp*: *u-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid* **using** *op PID own* **unfolding** *open-def* **by** *auto*
have *pPID1'*: *post s1' PID* = *pst1* **using** *step1* **unfolding** *a1 ou1* **by** (*auto simp*: *u-defs*)
let *?trn1* = *Trans s1 a1 ou1 s1'*
have *?iact* **proof**
show *step s1 a1* = (*ou1*, *s1'*) **using** *step1* **.**
next
show φ: φ *?trn1* **unfolding** φ*-def2*[*OF step1*] *a1 ou1* **by** *simp*
show *consume ?trn1 vl1 vll1*
using φ **unfolding** *vl1 consume-def v1 a1* **by** *auto*
next
show ¬ γ *?trn1* **using** *uid* **unfolding** *a1* **by** *auto*
next
have *eqButPID s1 s1'* **using** *Uact-uPaperC-step-eqButPID*[*OF - step1*] *a1* **by** *auto*
hence *ss1'*: *eqButPID s s1'* **using** *eqButPID-trans ss1* **by** *blast*
show *?*Δ *s vl s1' vll1* **using** *PID op ss1' lvl lvl1 map vlvl1 cor1*
unfolding Δ*1-def vl1 v1 pPID1'* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case (*PValS aid1 pst1*) **note** *v1* = *PValS*
have *pPID1*: *post s1 PID* = *pst1* **using** *cor1* **unfolding** *vl1 v1* **by** *auto*
then obtain *v vll* **where** *vl*: *vl* = *v* # *vll*
using *map* **unfolding** *vl1 v1* **by** (*cases vl*) *auto*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'*

assume *step*: *step s a* = (*ou*, *s'*) **and** *c*: *consume ?trn vl vl'*
have *PID'*: *PID* ∈∈ *postIDs s'* **using** *reach-postIDs-persist*[*OF PID step*] **.**
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'*
(**is** *?match* ∨ *?ignore*)
proof(*cases φ ?trn*)
case *True* **note** *φ* = *True*
then obtain *f*: *f ?trn* = *v* **and** *vl'*: *vl'* = *vll*
using *c* **unfolding** *vl consume-def φ-def2* **by** *auto*
show *?thesis*
proof(*cases v*)
case (*PVal pst*) **note** *v* = *PVal*
have *vll*: *vll* ≠ [] **using** *map* **unfolding** *vl1 v1 vl v* **by** *auto*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
have *a*: *a* = *Uact* (*uPost uid p PID pst*)
using *f-eq-PVal*[*OF step φ f*[*unfolded v*]] **unfolding** *uid p* **.**
have *eqButPID s s'* **using** *Uact-uPaperC-step-eqButPID*[*OF a step*] **by** *auto*
hence *s's1*: *eqButPID s' s1* **using** *eqButPID-sym eqButPID-trans ss1* **by** *blast*
have *op'*: ¬ *open s'* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*
have *?ignore* **proof**
show *γ*: ¬ *γ ?trn* **using** *step-open-φ-f-PVal-γ*[*OF rs step PID op φ f*[*unfolded v*]] **.**
show *?Δ s' vl' s1 vl1*
using *lvl1 lvl PID' map s's1 op' vll cor1* **unfolding** *Δ1-def vl1 vl vl' v*
by *auto*
qed
thus *?thesis* **by** *simp*
next
case (*PValS aid pst*) **note** *v* = *PValS*
define *uid* **where** *uid*: *uid* ≡ *admin s* **define** *p* **where** *p*: *p* ≡ *pass s uid*
have *a*: *a* = *COMact* (*comSendPost* (*admin s*) *p aid PID*)
using *f-eq-PValS*[*OF step φ f*[*unfolded v*]] **unfolding** *uid p* **.**
have *op'*: ¬ *open s'* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*
have *aid1*: *aid1* = *aid* **using** *map* **unfolding** *vl1 v1 vl v* **by** *simp*
have *uid1*: *uid* = *admin s1* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid p*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
have *pPID1'*: *post s1' PID* = *pst1* **using** *pPID1 step1* **unfolding** *a*
by (*auto simp*: *com-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid* **using** *op PID adm* **unfolding** *open-def* **by** *auto*

have *op1'*: ¬ *open s1'* **using** *step1 op1* **unfolding** *a open-def*
by (*auto simp*: *u-defs com-defs*)
let *?trn1* = *Trans s1 a ou1 s1'*
have *φ1*: *φ ?trn1* **using** *eqButPID-step-φ-imp*[*OF ss1 step step1 φ*] **.**
have *ou1*: *ou1* =
O-sendPost (*aid*, *clientPass s1 aid*, *PID*, *post s1 PID*, *owner s1 PID*, *vis s1 PID*)
using *φ1 step1 adm1 PID1* **unfolding** *a* **by** (*cases ou1*, *auto simp*: *com-defs*)
have *f1*: *f ?trn1* = *v1* **using** *φ1* **unfolding** *φ-def2*[*OF step1*] *v1 a ou1 aid1 pPID1* **by** *auto*
have *s's1'*: *eqButPID s' s1'* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vll1* **using** *φ1* **unfolding** *consume-def vl1 f1* **by** *simp*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **note** *γ* = *this*
have *ou*: (∃ *uid p aid pid*.
a = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨
ou = *ou1*
using *eqButPID-step-γ-out*[*OF ss1 step step1 op rsT rs1 γ*] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
show *?Δ s' vl' s1' vll1*
proof(*cases vll* = [])
case *True* **note** *vll* = *True*
hence *filter isPValS vll1* = [] **using** *map lvl lvl1* **unfolding** *vl vl1 v v1* **by** *simp*
hence *lvl1*: *list-all isPVal vll1*
using *filter-list-all-isPVal-isOVal lvl1* **unfolding** *vl1 v1* **by** *auto*
hence *Δ11 s' vl' s1' vll1* **using** *s's1' op1' op' PID' lvl lvl1 map cor1 pPID1 pPID1'*
unfolding *Δ11-def vl vl' vl1 v v1 vll* **by** *auto*
thus *?thesis* **by** *auto*
next
case *False* **note** *vll* = *False*
hence *Δ1 s' vl' s1' vll1* **using** *s's1' op1' op' PID' lvl lvl1 map cor1 pPID1 pPID1'*
unfolding *Δ1-def vl vl' vl1 v v1* **by** *auto*
thus *?thesis* **by** *auto*
qed
qed
thus *?thesis* **using** *vl* **by** *simp*
qed(*insert lvl vl*, *auto*)

```
      next
        case False note φ = False
        hence vl': vl' = vl using c unfolding consume-def by auto
         obtain ou1 s1' where step1: step s1 a = (ou1, s1') by(cases step s1 a)
auto
        have s's1': eqButPID s' s1' using eqButPID-step[OF ss1 step step1] .
        let ?trn1 = Trans s1 a ou1 s1'
       have φ1: ¬ φ ?trn1 using φ ss1 by (simp add: eqButPID-step-φ step step1)
        have pPID1': post s1' PID = pst1 using PID1 pPID1 step1 φ1
          apply(cases a)
          subgoal for x1 apply(cases x1) apply(fastforce simp: s-defs)+ .
          subgoal for x2 apply(cases x2) apply(fastforce simp: c-defs)+ .
          subgoal for x3 apply(cases x3) apply(fastforce simp: d-defs)+ .
          subgoal for x4 apply(cases x4) apply(fastforce simp: u-defs)+ .
          subgoal by (fastforce simp: u-defs)
          subgoal by (fastforce simp: u-defs)
          subgoal for x7 apply(cases x7) apply(fastforce simp: com-defs)+ .
          done
        have op': ¬ open s'
        using PID step φ op unfolding φ-def2[OF step] by auto
        have ?match proof
          show validTrans ?trn1 using step1 by simp
        next
          show consume ?trn1 vl1 vl1 using φ1 unfolding consume-def by simp
        next
          show γ ?trn = γ ?trn1 unfolding ss1 by simp
        next
          assume γ ?trn note γ = this
          have ou: (∃ uid p aid pid.
                a = COMact (comSendPost uid p aid pid) ∧ outPurge ou = outPurge
ou1) ∨
                  ou = ou1
          using eqButPID-step-γ-out[OF ss1 step step1 op rsT rs1 γ] .
          thus g ?trn = g ?trn1 by (cases a) auto
        next
          have Δ1 s' vl' s1' vl1 using s's1' PID' pPID1 pPID1' lvl lvl1 map cor1
op'
          unfolding Δ1-def vl vl' by auto
          thus ?Δ s' vl' s1' vl1 by simp
        qed
        thus ?thesis by simp
      qed
    qed
    thus ?thesis using vlvl1 by simp
  qed(insert lvl1 vl1, auto)
next
  case Nil note vl1 = Nil
  have ?react proof
    fix a :: act and ou :: out and s' :: state and vl'
```

let *?trn* = *Trans s a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
have *PID'*: *PID* ∈∈ *postIDs s'* **using** *reach-postIDs-persist*[*OF PID step*] **.**
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
proof(*cases* ∃ *uid p pstt*. *a* = *Uact* (*uPost uid p PID pstt*) ∧ *ou* = *outOK*)
case *True* **then obtain** *uid p pstt* **where**
a: *a* = *Uact* (*uPost uid p PID pstt*) **and** *ou*: *ou* = *outOK* **by** *auto*
hence *φ*: *φ ?trn* **unfolding** *φ-def2*[*OF step*] **by** *auto*
then obtain *v* **where** *vl*: *vl* = *v* # *vl'* **and** *f*: *f ?trn* = *v*
using *c* **unfolding** *consume-def φ-def2* **by** (*cases vl*) *auto*
obtain *pst* **where** *v*: *v* = *PVal pst* **using** *map lvl* **unfolding** *vl vl1* **by** (*cases v*) *auto*
have *pstt*: *pstt* = *pst* **using** *f* **unfolding** *a v* **by** *auto*
have *uid*: *uid* ∉ *UIDs* **using** *step op PID* **unfolding** *a ou open-def* **by** (*auto simp*: *u-defs*)
have *eqButPID s s'* **using** *Uact-uPaperC-step-eqButPID*[*OF a step*] **by** *auto*
hence *s's1*: *eqButPID s' s1* **using** *eqButPID-sym eqButPID-trans ss1* **by** *blast*
have *op'*: ¬ *open s'* **using** *step PID' op* **unfolding** *a ou open-def* **by** (*auto simp*: *u-defs*)
have *?ignore* **proof**
show ¬ *γ ?trn* **unfolding** *a* **using** *uid* **by** *auto*
next
show *?Δ s' vl' s1 vl1* **using** *PID' s's1 op' lvl map*
unfolding *Δ1-def vl1 vl* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case *False* **note** *a* = *False*
{**assume** *φ*: *φ ?trn*
then obtain *v vl'* **where** *vl*: *vl* = *v* # *vl'* **and** *f*: *f ?trn* = *v*
using *c* **unfolding** *consume-def* **by** (*cases vl*) *auto*
obtain *pst* **where** *v*: *v* = *PVal pst* **using** *map lvl* **unfolding** *vl vl1* **by** (*cases v*) *auto*
have *False* **using** *f f-eq-PVal*[*OF step φ*, *of pst*] *a φ v* **by** *auto*
}
hence *φ*: ¬ *φ ?trn* **by** *auto*
have *φ1*: ¬ *φ ?trn1* **by** (*metis φ eqButPID-step-φ step ss1 step1*)
have *op'*: ¬ *open s'* **using** *a op φ* **unfolding** *φ-def2*[*OF step*] **by** *auto*
have *vl'*: *vl'* = *vl* **using** *c φ* **unfolding** *consume-def* **by** *auto*
have *s's1'*: *eqButPID s' s1'* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
have *op1'*: ¬ *open s1'* **using** *op' eqButPID-open*[*OF s's1'*] **by** *simp*
have ⋀ *uid p pst*. *e-updatePost s1 uid p PID pst* ⟷ *e-updatePost s uid p PID pst*
using *eqButPID-stateSelectors*[*OF ss1*] **unfolding** *u-defs* **by** *auto*
hence *ou1*: ⋀ *uid p pst*. *a* = *Uact* (*uPost uid p PID pst*) ⟹ *ou1* = *ou*

using *step step1* **by** *auto*
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** *φ1* **unfolding** *consume-def* **by** *simp*
next
show *γ ?trn = γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **note** *γ = this*
have *ou*: (∃ *uid p aid pid.*
a = COMact (comSendPost uid p aid pid) ∧ outPurge ou = outPurge ou1) ∨
ou = ou1
using *eqButPID-step-γ-out*[*OF ss1 step step1 op rsT rs1 γ*] **.**
thus *g ?trn = g ?trn1* **by** (*cases a*) *auto*
next
show *?Δ s′ vl′ s1′ vl1* **using** *s′s1′ op′ PID′ lvl map*
unfolding *Δ1-def vl′ vl1* **by** *auto*
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vlvl1* **by** *simp*
qed
qed

lemma *unwind-cont-Δ11*: *unwind-cont Δ11* {*Δ11*}
proof(*rule*, *simp*)
let *?Δ = λs vl s1 vl1. Δ11 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *Δ11 s vl s1 vl1*
hence *vl*: *vl* = [] **and** *lvl1*: *list-all isPVal vl1*
and *rs*: *reach s* **and** *ss1*: *eqButPID s s1* **and** *op*: ¬ *open s* **and** *PID*: *PID* ∈∈ *postIDs s*
and *cor1*: *corrFrom* (*post s1 PID*) *vl1*
using *reachNT-reach* **unfolding** *Δ11-def* **by** *auto*
have *PID1*: *PID* ∈∈ *postIDs s1* **using** *eqButPID-stateSelectors*[*OF ss1*] *PID* **by** *auto*
have *own*: *owner s PID* ∈ *set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
hence *own1*: *owner s1 PID* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
have *adm*: *admin s* ∈ *set* (*userIDs s*) **using** *reach-admin-userIDs*[*OF rs own*] **.**
hence *adm1*: *admin s1* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
have *op1*: ¬ *open s1* **using** *op ss1 eqButPID-open* **by** *auto*
show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof(*cases vl1*)

case (*Cons v1 vll1*) **note** *vl1* = *Cons*
then obtain *pst1* **where** *v1*: *v1* = *PVal pst1* **using** *lvl1* **unfolding** *vl1* **by** (*cases v1*) *auto*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
define *a1* **where** *a1*: *a1* ≡ *Uact* (*uPost uid p PID pst1*)
have *uid1*: *uid* = *owner s1 PID* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid p*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a1* = (*ou1*, *s1'*) **by**(*cases step s1 a1*) *auto*
have *ou1*: *ou1* = *outOK* **using** *step1 PID1 own1* **unfolding** *a1 uid1 p1* **by** (*auto simp*: *u-defs*)
have *op1'*: ¬ *open s1'* **using** *step1 op1* **unfolding** *a1 ou1 open-def* **by** (*auto simp*: *u-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid* **using** *op PID own* **unfolding** *open-def* **by** *auto*
have *pPID1'*: *post s1' PID* = *pst1* **using** *step1* **unfolding** *a1 ou1* **by** (*auto simp*: *u-defs*)
let *?trn1* = *Trans s1 a1 ou1 s1'*
have *?iact* **proof**
show *step s1 a1* = (*ou1*, *s1'*) **using** *step1* **.**
next
show *φ*: *φ ?trn1* **unfolding** *φ-def2*[*OF step1*] *a1 ou1* **by** *simp*
show *consume ?trn1 vl1 vll1*
using *φ* **unfolding** *vl1 consume-def v1 a1* **by** *auto*
next
show ¬ *γ ?trn1* **using** *uid* **unfolding** *a1* **by** *auto*
next
have *eqButPID s1 s1'* **using** *Uact-uPaperC-step-eqButPID*[*OF - step1*] *a1* **by** *auto*
hence *ss1'*: *eqButPID s s1'* **using** *eqButPID-trans ss1* **by** *blast*
show *?Δ s vl s1' vll1*
using *PID op ss1' lvl1 cor1* **unfolding** *Δ11-def vl1 v1 vl pPID1'* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case *Nil* **note** *vl1* = *Nil*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *c*: *consume ?trn vl vl'*
have *PID'*: *PID* ∈∈ *postIDs s'* **using** *reach-postIDs-persist*[*OF PID step*] **.**
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
have *φ*: ¬ *φ ?trn* **using** *c* **unfolding** *consume-def vl* **by** *auto*
show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'*
(**is** *?match* ∨ *?ignore*)
proof−
have *vl'*: *vl'* = *vl* **using** *c* **unfolding** *vl consume-def* **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*)

auto
  have *s′s1′*: *eqButPID s′ s1′* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
  let *?trn1* = *Trans s1 a ou1 s1′*
  have *φ1*: ¬ *φ ?trn1* **using** *φ ss1* **by** (*simp add*: *eqButPID-step-φ step step1*)
  have *pPID1′*: *post s1′ PID* = *post s1 PID* **using** *PID1 step1 φ1*
    apply(*cases a*)
    subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
    subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
    subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
    subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
    subgoal by *fastforce*
    subgoal by *fastforce*
    subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
    done
  have *op′*: ¬ *open s′* **using** *PID step φ op* **unfolding** *φ-def2*[*OF step*] **by**
auto
  have *?match* **proof**
    show *validTrans ?trn1* **using** *step1* **by** *simp*
  next
    show *consume ?trn1 vl1 vl1* **using** *φ1* **unfolding** *consume-def* **by** *simp*
  next
    show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
  next
    assume *γ ?trn* **note** *γ* = *this*
    have *ou*: (∃ *uid p aid pid*.
      a = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge*
ou1) ∨
      ou = *ou1*
    using *eqButPID-step-γ-out*[*OF ss1 step step1 op rsT rs1 γ*] **.**
    thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
  next
    have *?Δ s′ vl′ s1′ vl1* **using** *s′s1′ PID′ pPID1′ lvl1 cor1 op′*
    unfolding *Δ11-def vl vl′* **by** *auto*
    thus *?Δ s′ vl′ s1′ vl1* **by** *simp*
  qed
  thus *?thesis* **by** *simp*
  qed
  qed
  thus *?thesis* **using** *vl1* **by** *simp*
  qed
qed

lemma *unwind-cont-Δ31*: *unwind-cont Δ31* {*Δ31*,*Δ32*}
proof(*rule*, *simp*)
  let *?Δ* = *λs vl s1 vl1*. *Δ31 s vl s1 vl1* ∨ *Δ32 s vl s1 vl1*
  fix *s s1* :: *state* **and** *vl vl1* :: *value list*
  assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *Δ31 s vl s1 vl1*
  then obtain *ul ul1 sul vll vll1* **where**
  lul: *list-all* (*Not* ∘ *isOVal*) *ul* **and** *lul1*: *list-all* (*Not* ∘ *isOVal*) *ul1*

and *map*: *map tgtAPI* (*filter isPValS ul*) = *map tgtAPI* (*filter isPValS ul1*)
and *rs*: *reach s* **and** *ss1*: *eqButPID s s1* **and** *op*: ¬ *open s* **and** *PID*: *PID* ∈∈ *postIDs s*
and *cor1*: *corrFrom* (*post s1 PID*) *vl1*
and *ful*: *ul* ≠ [] **and** *ful1*: *ul1* ≠ []
and *lastul*: *isPVal* (*last ul*) **and** *ulul1*: *last ul* = *last ul1*
and *lsul*: *list-all isPValS sul*
and *vl*: *vl* = *ul* @ *sul* @ *OVal True* # *vll*
and *vl1*: *vl1* = *ul1* @ *sul* @ *OVal True* # *vll1*
and *BO*: *BO vll vll1*
using *reachNT-reach* **unfolding** *Δ31-def* **by** *auto*
have *ulNE*: *ul* ≠ [] **and** *ul1NE*: *ul1* ≠ [] **using** *ful ful1* **by** *auto*
have *PID1*: *PID* ∈∈ *postIDs s1* **using** *eqButPID-stateSelectors*[*OF ss1*] *PID* **by** *auto*
have *own*: *owner s PID* ∈ *set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
hence *own1*: *owner s1 PID* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
have *adm*: *admin s* ∈ *set* (*userIDs s*) **using** *reach-admin-userIDs*[*OF rs own*] **.**
hence *adm1*: *admin s1* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
have *op1*: ¬ *open s1* **using** *op ss1 eqButPID-open* **by** *auto*
obtain *v1 ull1* **where** *ul1*: *ul1* = *v1* # *ull1* **using** *ful1* **by** (*cases ul1*) *auto*
show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof(*cases v1*)
case (*PVal pst1*) **note** *v1* = *PVal*
show *?thesis* **proof**(*cases list-ex isPVal ull1*)
case *True* **note** *lull1* = *True*
hence *full1*: *filter isPVal ull1* ≠ [] **by** (*induct ull1*) *auto*
hence *ull1NE*: *ull1* ≠ [] **by** *auto*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
define *a1* **where** *a1*: *a1* ≡ *Uact* (*uPost uid p PID pst1*)
have *uid1*: *uid* = *owner s1 PID* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid p*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a1* = (*ou1*, *s1'*) **by**(*cases step s1 a1*) *auto*
have *ou1*: *ou1* = *outOK* **using** *step1 PID1 own1* **unfolding** *a1 uid1 p1* **by** (*auto simp*: *u-defs*)
have *op1'*: ¬ *open s1'* **using** *step1 op1* **unfolding** *a1 ou1 open-def* **by** (*auto simp*: *u-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid* **using** *op PID own* **unfolding** *open-def* **by** *auto*
have *pPID1'*: *post s1' PID* = *pst1* **using** *step1* **unfolding** *a1 ou1* **by** (*auto simp*: *u-defs*)
let *?trn1* = *Trans s1 a1 ou1 s1'*
let *?vl1'* = *ull1* @ *sul* @ *OVal True* # *vll1*
have *?iact* **proof**

show *step s1 a1* = (*ou1*, *s1'*) **using** *step1* **.**
next
show *φ*: *φ ?trn1* **unfolding** *φ-def2*[*OF step1*] *a1 ou1* **by** *simp*
show *consume ?trn1 vl1 ?vl1'*
using *φ* **unfolding** *vl1 ul1 consume-def v1 a1* **by** *simp*
next
show ¬ *γ ?trn1* **using** *uid* **unfolding** *a1* **by** *auto*
next
have *eqButPID s1 s1'* **using** *Uact-uPaperC-step-eqButPID*[*OF - step1*] *a1* **by** *auto*
hence *ss1'*: *eqButPID s s1'* **using** *eqButPID-trans ss1* **by** *blast*
have Δ*31 s vl s1' ?vl1'*
using *PID op ss1' lul lul1 map ulul1 cor1 BO ull1NE ful ful1 full1 lastul ulul1 lsul*
unfolding Δ*31-def vl vl1 ul1 v1 pPID1'* **apply** *auto*
apply(*rule exI*[*of - ul*]) **apply**(*rule exI*[*of - ull1*]) **apply**(*rule exI*[*of - sul*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*
thus *?*Δ *s vl s1' ?vl1'* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case *False* **note** *lull1* = *False*
hence *ull1*: *ull1* = [] **using** *lastul* **unfolding** *ulul1 ul1 v1* **by** *simp*(*meson Bex-set last-in-set*)
hence *ul1*: *ul1* = [*PVal pst1*] **unfolding** *ul1 v1* **by** *simp*
obtain *ulll* **where** *ul-ulll*: *ul* = *ulll* ## *PVal pst1* **using** *lastul ulul1 ulNE* **unfolding** *ul1 ull1 v1*
by (*cases ul rule*: *rev-cases*) *auto*
hence *ulNE*: *ul* ≠ [] **by** *simp*

have *filter isPValS ulll* = [] **using** *map* **unfolding** *ul-ulll ul1 v1 ull1* **by** *simp*
hence *lull*: *list-all isPVal ulll* **using** *lul filter-list-all-isPVal-isOVal*
unfolding *ul-ulll* **by** *auto*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *c*: *consume ?trn vl vl'*
have *PID'*: *PID* ∈∈ *postIDs s'* **using** *reach-postIDs-persist*[*OF PID step*] **.**
obtain *ul'* **where** *cc*: *consume ?trn ul ul'* **and**
vl': *vl'* = *ul'* @ *sul* @ *OVal True* # *vll* **using** *c ulNE* **unfolding** *consume-def vl*
by (*cases φ ?trn*) *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
show *match ?*Δ *s s1 vl1 a ou s' vl'* ∨ *ignore ?*Δ *s s1 vl1 a ou s' vl'*
(**is** *?match* ∨ *?ignore*)
proof(*cases ulll*)
case *Nil*

hence *ul*: *ul* = [*PVal pst1*] **unfolding** *ul-ulll* **by** *simp*
have *?match* **proof**(*cases* φ *?trn*)
case *True* **note** φ = *True*
then obtain *f*: *f ?trn* = *PVal pst1* **and** *ul'*: *ul'* = []
using *cc* **unfolding** *ul consume-def* φ*-def2* **by** *auto*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
have *a*: *a* = *Uact* (*uPost uid p PID pst1*)
using *f-eq-PVal*[*OF step* φ *f*] **unfolding** *uid p* **.**
have *uid1*: *uid* = *owner s1 PID* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid p*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
have φ*1*: φ *?trn1* **using** *eqButPID-step-*φ*-imp*[*OF ss1 step step1* φ] **.**
have *ou1*: *ou1* = *outOK*
using φ*1 step1 PID1* **unfolding** *a* **by** (*cases ou1*, *auto simp*: *com-defs*)
have *pPID'*: *post s' PID* = *pst1* **using** *step* φ **unfolding** *a* **by** (*auto simp*: *u-defs*)
have *pPID1'*: *post s1' PID* = *pst1* **using** *step1* φ*1* **unfolding** *a* **by** (*auto simp*: *u-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid* **using** *op PID own* **unfolding** *open-def* **by** *auto*
have *op1'*: ¬ *open s1'* **using** *step1 op1* **unfolding** *a open-def*
by (*auto simp*: *u-defs com-defs*)
have *f1*: *f ?trn1* = *PVal pst1* **using** φ*1* **unfolding** φ*-def2*[*OF step1*] *v1 a ou1* **by** *auto*
have *s's1'*: *eqButPID s' s1'* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
have *op'*: ¬ *open s'* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*
have *ou*: *ou* = *outOK* **using** φ *op op'* **unfolding** φ*-def2*[*OF step*] *a* **by** *auto*
let *?vl'* = *sul* @ *OVal True* # *vll*
let *?vl1'* = *sul* @ *OVal True* # *vll1*
show *?thesis* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 ?vl1'*
using φ*1* **unfolding** *consume-def ul1 f1 vl1* **by** *simp*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **note** γ = *this*
thus *g ?trn* = *g ?trn1* **using** *ou ou1* **by** (*cases a*) *auto*
next
have *s'*: *s'* = *s1'* **using** *eqButPID-step-eq*[*OF ss1 a ou step step1*] **.**
have *corr1*: *corrFrom* (*post s1' PID*) *?vl1'*
using *cor1* **unfolding** *vl1 ul1 v1 pPID1'* **by** *auto*

have $\Delta 32$ *s' vl' s1' ?vl1'*
using *PID' op1 op' s's1' lul lul1 map ulul1 cor1 BO ful ful1 lastul ulul1 lsul corr1*
unfolding $\Delta 32$*-def vl vl1 v1 vl' ul' ul ul1 s'* **apply** *simp*
apply(*rule exI*[*of - sul*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*
thus *?*Δ *s' vl' s1' ?vl1'* **by** *simp*
qed
next
case *False* **note** φ = *False*
hence *ul'*: *ul'* = *ul* **using** *cc* **unfolding** *consume-def* **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
have *s's1'*: *eqButPID s' s1'* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
let *?trn1* = *Trans s1 a ou1 s1'*
have φ*1*: ¬ φ *?trn1* **using** φ *ss1* **by** (*simp add*: *eqButPID-step-*φ *step step1*)
have *pPID1'*: *post s1' PID* = *post s1 PID* **using** *PID1 step1* φ*1*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *fastforce*
subgoal by *fastforce*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op'*: ¬ *open s'* **using** *PID step* φ *op* **unfolding** φ*-def2*[*OF step*] **by** *auto*
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** φ*1* **unfolding** *consume-def* **by** *simp*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **note** γ = *this*
have *ou*: (∃ *uid p aid pid*.
a = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨
ou = *ou1*
using *eqButPID-step-*γ*-out*[*OF ss1 step step1 op rsT rs1* γ] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
have $\Delta 31$ *s' vl' s1' vl1*
using *PID' pPID1' op' s's1' lul lul1 map ulul1 cor1*
BO ful ful1 lastul ulul1 lsul cor1
unfolding $\Delta 31$*-def vl vl1 v1 vl' ul'* **apply** *simp*

apply(*rule exI*[*of* - *ul*]) **apply**(*rule exI*[*of* - *ul1*]) **apply**(*rule exI*[*of* - *sul*])
apply(*rule exI*[*of* - *vll*]) **apply**(*rule exI*[*of* - *vll1*]) **by** *auto*
thus *?*Δ *s′ vl′ s1′ vl1* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
thus *?thesis* **by** *simp*
next
case (*Cons v ullll*) **note** *ulll* = *Cons*
then obtain *pst* **where** *v*: *v* = *PVal pst* **using** *lull ul-ulll ulll lul* **by** (*cases v*) *auto*
define *ull* **where** *ull*: *ull* ≡ *ullll* ## *PVal pst1*
have *ul*: *ul* = *v* # *ull* **unfolding** *ul-ulll ull ulll* **by** *simp*
show *?thesis* **proof**(*cases* φ *?trn*)
case *True* **note** φ = *True*
then obtain *f*: *f ?trn* = *v* **and** *ul′*: *ul′* = *ull*
using *cc* **unfolding** *ul consume-def* φ*-def2* **by** *auto*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
have *a*: *a* = *Uact* (*uPost uid p PID pst*)
using *f-eq-PVal*[*OF step* φ *f*[*unfolded v*]] **unfolding** *uid p* **.**
have *eqButPID s s′* **using** *Uact-uPaperC-step-eqButPID*[*OF a step*] **by** *auto*
hence *s′s1*: *eqButPID s′ s1* **using** *eqButPID-sym eqButPID-trans ss1* **by** *blast*
have *op′*: ¬ *open s′* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*
have *?ignore* **proof**
show γ: ¬ γ *?trn* **using** *step-open-*φ*-f-PVal-*γ[*OF rs step PID op* φ *f*[*unfolded v*]] **.**
have Δ*31 s′ vl′ s1 vl1*
using *PID′ op′ s′s1 lul lul1 map ulul1 cor1 BO ful ful1 lastul ulul1 lsul ull*
unfolding Δ*31-def vl vl1 v1 vl′ ul′ ul v* **apply** *simp*
apply(*rule exI*[*of* - *ull*]) **apply**(*rule exI*[*of* - *ul1*]) **apply**(*rule exI*[*of* - *sul*])
apply(*rule exI*[*of* - *vll*]) **apply**(*rule exI*[*of* - *vll1*]) **by** *auto*
thus *?*Δ *s′ vl′ s1 vl1* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case *False* **note** φ = *False*
hence *ul′*: *ul′* = *ul* **using** *cc* **unfolding** *consume-def* **by** *auto*
obtain *ou1 s1′* **where** *step1*: *step s1 a* = (*ou1*, *s1′*) **by**(*cases step s1 a*) *auto*
have *s′s1′*: *eqButPID s′ s1′* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
let *?trn1* = *Trans s1 a ou1 s1′*
have φ*1*: ¬ φ *?trn1* **using** φ *ss1* **by** (*simp add*: *eqButPID-step-*φ *step*

```
step1)
      have pPID1': post s1' PID = post s1 PID using PID1 step1 φ1
       apply(cases a)
       subgoal for x1 apply(cases x1) apply(fastforce simp: s-defs)+ .
       subgoal for x2 apply(cases x2) apply(fastforce simp: c-defs)+ .
       subgoal for x3 apply(cases x3) apply(fastforce simp: d-defs)+ .
       subgoal for x4 apply(cases x4) apply(fastforce simp: u-defs)+ .
       subgoal by fastforce
       subgoal by fastforce
       subgoal for x7 apply(cases x7) apply(fastforce simp: com-defs)+ .
       done
      have op': ¬ open s' using PID step φ op unfolding φ-def2[OF step]
by auto
      have ?match proof
       show validTrans ?trn1 using step1 by simp
      next
        show consume ?trn1 vl1 vl1 using φ1 unfolding consume-def by
simp
      next
       show γ ?trn = γ ?trn1 unfolding ss1 by simp
      next
       assume γ ?trn note γ = this
       have ou: (∃ uid p aid pid.
         a = COMact (comSendPost uid p aid pid) ∧ outPurge ou = outPurge
ou1) ∨
          ou = ou1
       using eqButPID-step-γ-out[OF ss1 step step1 op rsT rs1 γ] .
       thus g ?trn = g ?trn1 by (cases a) auto
      next
       have Δ31 s' vl' s1' vl1
       using PID' pPID1' op' s's1' lul lul1 map ulul1 cor1
       BO ful ful1 lastul ulul1 lsul cor1
       unfolding Δ31-def vl vl1 v1 vl' ul' apply simp
        apply(rule exI[of - ul]) apply(rule exI[of - ul1]) apply(rule exI[of -
sul])
       apply(rule exI[of - vll]) apply(rule exI[of - vll1]) by auto
       thus ?Δ s' vl' s1' vl1 by simp
      qed
     thus ?thesis by simp
     qed
    qed
   qed
   thus ?thesis using vl by simp
  qed
 next
  case (PValS aid1 pst1) note v1 = PValS
  have pPID1: post s1 PID = pst1 using cor1 unfolding vl1 ul1 v1 by auto
  then obtain v ull where ul: ul = v # ull
  using map unfolding ul1 v1 by (cases ul) auto
```

let *?vl1′* = *ull1* @ *sul* @ *OVal True* # *vll1*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn* = *Trans s a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *c*: *consume ?trn vl vl′*
have *PID′*: *PID* ∈∈ *postIDs s′* **using** *reach-postIDs-persist*[*OF PID step*] **.**
obtain *ul′* **where** *cc*: *consume ?trn ul ul′* **and**
vl′: *vl′* = *ul′* @ *sul* @ *OVal True* # *vll* **using** *c ul* **unfolding** *consume-def vl*
by (*cases φ ?trn*) *auto*
obtain *ou1 s1′* **where** *step1*: *step s1 a* = (*ou1*, *s1′*) **by**(*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1′*
show *match ?Δ s s1 vl1 a ou s′ vl′* ∨ *ignore ?Δ s s1 vl1 a ou s′ vl′*
(**is** *?match* ∨ *?ignore*)
proof(*cases φ ?trn*)
case *True* **note** *φ* = *True*
then obtain *f*: *f ?trn* = *v* **and** *ul′*: *ul′* = *ull*
using *cc* **unfolding** *ul consume-def φ-def2* **by** *auto*
show *?thesis*
proof(*cases v*)
case (*PVal pst*) **note** *v* = *PVal*
have *full*: *ull* ≠ [] **using** *map* **unfolding** *ul1 v1 ul v* **by** *auto*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
have *a*: *a* = *Uact* (*uPost uid p PID pst*)
using *f-eq-PVal*[*OF step φ f*[*unfolded v*]] **unfolding** *uid p* **.**
have *eqButPID s s′* **using** *Uact-uPaperC-step-eqButPID*[*OF a step*] **by** *auto*
hence *s′s1*: *eqButPID s′ s1* **using** *eqButPID-sym eqButPID-trans ss1* **by** *blast*
have *op′*: ¬ *open s′* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*

have *?ignore* **proof**
show *γ*: ¬ *γ ?trn* **using** *step-open-φ-f-PVal-γ*[*OF rs step PID op φ f*[*unfolded v*]] **.**
have *Δ31 s′ vl′ s1 vl1*
using *PID′ op′ s′s1 lul lul1 map ulul1 cor1 BO ful ful1 lastul ulul1 lsul full*
unfolding *Δ31-def vl vl1 v1 vl′ ul′ ul v* **apply** *simp*
apply(*rule exI*[*of - ull*]) **apply**(*rule exI*[*of - ul1*]) **apply**(*rule exI*[*of - sul*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*
thus *?Δ s′ vl′ s1 vl1* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case (*PValS aid pst*) **note** *v* = *PValS*
define *uid* **where** *uid*: *uid* ≡ *admin s* **define** *p* **where** *p*: *p* ≡ *pass s uid*
have *a*: *a* = *COMact* (*comSendPost* (*admin s*) *p aid PID*)

using *f-eq-PValS*[*OF step φ f*[*unfolded v*]] **unfolding** *uid p* **.**
have *op′*: ¬ *open s′* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*
have *aid1*: *aid1* = *aid* **using** *map* **unfolding** *ul1 v1 ul v* **by** *simp*
have *uid1*: *uid* = *admin s1* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid p*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1′* **where** *step1*: *step s1 a* = (*ou1*, *s1′*) **by**(*cases step s1 a*) *auto*
have *pPID1′*: *post s1′ PID* = *pst1* **using** *pPID1 step1* **unfolding** *a*
by (*auto simp*: *com-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid* **using** *op PID adm* **unfolding** *open-def* **by** *auto*
have *op1′*: ¬ *open s1′* **using** *step1 op1* **unfolding** *a open-def*
by (*auto simp*: *u-defs com-defs*)
let *?trn1* = *Trans s1 a ou1 s1′*
have *φ1*: *φ ?trn1* **using** *eqButPID-step-φ-imp*[*OF ss1 step step1 φ*] **.**
have *ou1*: *ou1* =
O-sendPost (*aid*, *clientPass s1 aid*, *PID*, *post s1 PID*, *owner s1 PID*, *vis s1 PID*)
using *φ1 step1 adm1 PID1* **unfolding** *a* **by** (*cases ou1*, *auto simp*: *com-defs*)
have *f1*: *f ?trn1* = *v1* **using** *φ1* **unfolding** *φ-def2*[*OF step1*] *v1 a ou1 aid1 pPID1* **by** *auto*
have *s′s1′*: *eqButPID s′ s1′* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 ?vl1′* **using** *φ1* **unfolding** *consume-def ul1 f1 vl1* **by** *simp*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **note** *γ* = *this*
have *ou*: (∃ *uid p aid pid*.
a = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨
ou = *ou1*
using *eqButPID-step-γ-out*[*OF ss1 step step1 op rsT rs1 γ*] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
have *corr1*: *corrFrom* (*post s1′ PID*) *?vl1′*
using *cor1* **unfolding** *vl1 ul1 v1 pPID1′* **by** *auto*
have *ullull1*: *ull1* ≠ [] ⟶ *ull* ≠ [] **using** *ul ul1 lastul ulul1* **unfolding** *v v1*
by *fastforce*
have *Δ31 s′ vl′ s1′ ?vl1′*
using *PID′ op′ s′s1′ lul lul1 map ulul1 cor1 BO ful ful1 lastul ulul1 lsul corr1 ullull1*
unfolding *Δ31-def vl vl1 v1 vl′ ul′ ul ul1 v* **apply** *auto*

apply(*rule exI*[*of - ull*]) **apply**(*rule exI*[*of - ull1*]) **apply**(*rule exI*[*of - sul*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*
thus *?*Δ *s' vl' s1' ?vl1'* **by** *simp*
qed
thus *?thesis* **using** *ul* **by** *simp*
next
qed(*insert lul ul*, *auto*)
next
case *False* **note** φ = *False*
hence *ul'*: *ul'* = *ul* **using** *cc* **unfolding** *consume-def* **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
have *s's1'*: *eqButPID s' s1'* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
let *?trn1* = *Trans s1 a ou1 s1'*
have *φ1*: ¬ φ *?trn1* **using** φ *ss1* **by** (*simp add*: *eqButPID-step-φ step step1*)
have *pPID1'*: *post s1' PID* = *pst1* **using** *PID1 pPID1 step1 φ1*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *fastforce*
subgoal by *fastforce*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op'*: ¬ *open s'* **using** *PID step* φ *op* **unfolding** *φ-def2*[*OF step*] **by** *auto*
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** *φ1* **unfolding** *consume-def* **by** *simp*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **note** γ = *this*
have *ou*: (∃ *uid p aid pid*.
a = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨
ou = *ou1*
using *eqButPID-step-γ-out*[*OF ss1 step step1 op rsT rs1* γ] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
have Δ*31 s' vl' s1' vl1*
using *PID' pPID1 pPID1' op' s's1' lul lul1 map ulul1 cor1*
BO ful ful1 lastul ulul1 lsul cor1
unfolding Δ*31-def vl vl1 v1 vl' ul'* **apply** *simp*
apply(*rule exI*[*of - ul*]) **apply**(*rule exI*[*of - ul1*]) **apply**(*rule exI*[*of - sul*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*

thus *?Δ s′ vl′ s1′ vl1* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vl* **by** *simp*
qed(*insert lul1 ul1*, *auto*)
qed

lemma *unwind-cont-Δ32*: *unwind-cont* Δ*32* {Δ*2*,Δ*32*,Δ*4*}
proof(*rule*, *simp*)
let *?Δ* = *λs vl s1 vl1*. Δ*2 s vl s1 vl1* ∨ Δ*32 s vl s1 vl1* ∨ Δ*4 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** Δ*32 s vl s1 vl1*
then obtain *ul vll vll1* **where**
lul: *list-all isPValS ul*
and *rs*: *reach s* **and** *ss1*: *s1* = *s* **and** *op*: ¬ *open s* **and** *PID*: *PID* ∈∈ *postIDs s*
and *cor1*: *corrFrom* (*post s1 PID*) *vl1*
and *vl*: *vl* = *ul* @ *OVal True* # *vll*
and *vl1*: *vl1* = *ul* @ *OVal True* # *vll1*
and *BO*: *BO vll vll1*
using *reachNT-reach* **unfolding** Δ*32-def* **by** *blast*
have *own*: *owner s PID* ∈ *set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
have *adm*: *admin s* ∈ *set* (*userIDs s*) **using** *reach-admin-userIDs*[*OF rs own*] **.**
show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn* = *Trans s a ou s′* **let** *?trn1* = *Trans s1 a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *c*: *consume ?trn vl vl′*
have *PID′*: *PID* ∈∈ *postIDs s′* **using** *reach-postIDs-persist*[*OF PID step*] **.**
show *match ?Δ s s1 vl1 a ou s′ vl′* ∨ *ignore ?Δ s s1 vl1 a ou s′ vl′*
(**is** *?match* ∨ *?ignore*)
proof−
have *?match* **proof**(*cases ul* = [])
case *False* **note** *ul* = *False*
then obtain *ul′* **where** *cc*: *consume ?trn ul ul′*
and *vl′*: *vl′* = *ul′* @ *OVal True* # *vll* **using** *vl c* **unfolding** *consume-def*
by (*cases φ ?trn*) *auto*
let *?vl1′* = *ul′* @ *OVal True* # *vll1*
show *?thesis* **proof**
show *validTrans ?trn1* **using** *step* **unfolding** *ss1* **by** *simp*
next
show *consume ?trn1 vl1 ?vl1′* **using** *cc ul* **unfolding** *vl1 consume-def ss1*
by (*cases φ ?trn*) *auto*
next

show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **note** γ = *this*
thus *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
have Δ*32 s' vl' s' ?vl1'*
proof(*cases* φ *?trn*)
case *True* **note** φ = *True*
then obtain *v* **where** *f*: *f ?trn* = *v* **and** *ul*: *ul* = *v* # *ul'*
using *cc* **unfolding** *consume-def* **by** (*cases ul*) *auto*
define *uid* **where** *uid*: *uid* ≡ *admin s* **define** *p* **where** *p*: *p* ≡ *pass s uid*
obtain *aid pst* **where** *v*: *v* = *PValS aid pst* **using** *lul* **unfolding** *ul* **by** (*cases v*) *auto*
have *a*: *a* = *COMact* (*comSendPost* (*admin s*) *p aid PID*)
using *f-eq-PValS*[*OF step* φ *f*[*unfolded v*]] **unfolding** *uid p* **.**
have *op'*: ¬ *open s'* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*
have *pPID'*: *post s' PID* = *post s PID*
using *step* **unfolding** *a* **by** (*auto simp*: *com-defs*)
show *?thesis* **using** *PID' pPID' op' cor1 BO lul*
unfolding Δ*32-def vl1 ul ss1 v vl'* **by** *auto*
next
case *False* **note** φ = *False*
hence *ul*: *ul* = *ul'* **using** *cc* **unfolding** *consume-def* **by** (*cases ul*) *auto*
have *pPID'*: *post s' PID* = *post s PID*
using *step* φ *PID op*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *fastforce*
subgoal by *fastforce*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op'*: ¬ *open s'* **using** *PID step* φ *op* **unfolding** φ*-def2*[*OF step*] **by** *auto*
show *?thesis* **using** *PID' pPID' op' cor1 BO lul*
unfolding Δ*32-def vl1 ul ss1 vl'* **by** *auto*
qed
thus *?*Δ *s' vl' s' ?vl1'* **by** *simp*
qed
next
case *True* **note** *ul* = *True*
show *?thesis* **proof**(*cases* φ *?trn*)
case *True* **note** φ = *True*
hence *f*: *f ?trn* = *OVal True* **and** *vl'*: *vl'* = *vll*
using *vl c* **unfolding** *consume-def ul* **by** *auto*

```
      have op′: open s′ using f-eq-OVal[OF step φ f] op by simp
      show ?thesis proof
        show validTrans ?trn1 using step unfolding ss1 by simp
      next
        show consume ?trn1 vl1 vll1 using ul φ c
        unfolding vl1 vl′ vl ss1 consume-def by auto
      next
        show γ ?trn = γ ?trn1 unfolding ss1 by simp
      next
        assume γ ?trn note γ = this
        thus g ?trn = g ?trn1 unfolding ss1 by simp
      next
        have pPID′: post s′ PID = post s PID
          using step φ PID op op′ f
          apply(cases a)
          subgoal for x1 apply(cases x1) apply(fastforce simp: s-defs)+ .
          subgoal for x2 apply(cases x2) apply(fastforce simp: c-defs)+ .
          subgoal for x3 apply(cases x3) apply(fastforce simp: d-defs)+ .
          subgoal for x4 apply(cases x4) apply(fastforce simp: u-defs)+ .
          subgoal by fastforce
          subgoal by fastforce
          subgoal for x7 apply(cases x7) apply(fastforce simp: com-defs)+ .
          done
        show ?Δ s′ vl′ s′ vll1 using BO proof cases
          case BO-PVal
          hence Δ2 s′ vl′ s′ vll1 using PID′ pPID′ op′ cor1 BO lul
          unfolding Δ2-def vl1 ul ss1 vl′ by auto
          thus ?thesis by simp
        next
          case BO-BC
          hence Δ4 s′ vl′ s′ vll1 using PID′ pPID′ op′ cor1 BO lul
          unfolding Δ4-def vl1 ul ss1 vl′ by auto
          thus ?thesis by simp
        qed
      qed
    next
      case False note φ = False
      hence vl′: vl′ = vl using c unfolding consume-def by auto
      have pPID′: post s′ PID = post s PID
        using step φ PID op
        apply(cases a)
        subgoal for x1 apply(cases x1) apply(fastforce simp: s-defs)+ .
        subgoal for x2 apply(cases x2) apply(fastforce simp: c-defs)+ .
        subgoal for x3 apply(cases x3) apply(fastforce simp: d-defs)+ .
        subgoal for x4 apply(cases x4) apply(fastforce simp: u-defs)+ .
        subgoal by fastforce
        subgoal by fastforce
        subgoal for x7 apply(cases x7) apply(fastforce simp: com-defs)+ .
        done
```

have *op'*: ¬ *open s'* **using** *PID step φ op* **unfolding** *φ-def2*[*OF step*] **by** *auto*
show *?thesis* **proof**
show *validTrans ?trn1* **using** *step* **unfolding** *ss1* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** *ul φ* **unfolding** *vl1 consume-def ss1* **by** *simp*
next
show *γ ?trn = γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **note** *γ = this*
thus *g ?trn = g ?trn1* **unfolding** *ss1* **by** *simp*
next
have *Δ32 s' vl' s' vl1* **using** *PID' pPID' op' cor1 BO lul*
unfolding *Δ32-def vl vl1 ul ss1 vl'* **apply** *simp*
apply(*rule exI*[*of* - []])
apply(*rule exI*[*of* - *vll*]) **apply**(*rule exI*[*of* - *vll1*]) **by** *auto*
thus *?Δ s' vl' s' vl1* **by** *simp*
qed
qed
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vl* **by** *simp*
qed
qed

lemma *unwind-cont-Δ2*: *unwind-cont Δ2* {*Δ2*}
proof(*rule*, *simp*)
let *?Δ = λs vl s1 vl1. Δ2 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *Δ2 s vl s1 vl1*
hence *vlvl1*: *vl = vl1*
and *rs*: *reach s* **and** *ss1*: *s1 = s* **and** *op*: *open s* **and** *PID*: *PID ∈∈ postIDs s*
and *cor1*: *corrFrom* (*post s1 PID*) *vl1* **and** *lvl*: *list-all* (*Not ∘ isOVal*) *vl*
using *reachNT-reach* **unfolding** *Δ2-def* **by** *auto*
have *own*: *owner s PID ∈ set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
have *adm*: *admin s ∈ set* (*userIDs s*) **using** *reach-admin-userIDs*[*OF rs own*] **.**
show *iaction ?Δ s vl s1 vl1 ∨*
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn = Trans s a ou s'* **let** *?trn1 = Trans s1 a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *c*: *consume ?trn vl vl'*
have *PID'*: *PID ∈∈ postIDs s'* **using** *reach-postIDs-persist*[*OF PID step*] **.**
show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'* (**is** *?match*

∨ *?ignore*)
proof−
have *?match* **proof**(*cases* φ *?trn*)
case *True* **note** φ = *True*
then obtain *v* **where** *vl*: *vl* = *v* # *vl′* **and** *f*: *f ?trn* = *v*
using *c* **unfolding** *consume-def* φ*-def2* **by**(*cases vl*) *auto*
show *?thesis* **proof**(*cases v*)
case (*PVal pst*) **note** *v* = *PVal*
have *a*: *a* = *Uact* (*uPost* (*owner s PID*) (*pass s* (*owner s PID*)) *PID pst*)
using *f-eq-PVal*[*OF step* φ *f*[*unfolded v*]] **.**
have *ou*: *ou* = *outOK* **using** *step own PID* **unfolding** *a* **by** (*auto simp*: *u-defs*)
have *op′*: *open s′* **using** *step op PID PID′* φ
unfolding *open-def a* **by** (*auto simp*: *u-defs*)
have *pPID′*: *post s′ PID* = *pst*
using *step* φ *PID op f op′* **unfolding** *a* **by**(*auto simp*: *u-defs*)
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl′* **using** φ *vlvl1* **unfolding** *ss1 consume-def vl f* **by** *auto*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
show *?*Δ *s′ vl′ s′ vl′* **using** *cor1 PID′ pPID′ op′ lvl vlvl1 ss1*
unfolding Δ*2-def vl v* **by** *auto*
qed
next
case (*PValS aid pid*) **note** *v* = *PValS*
have *a*: *a* = *COMact* (*comSendPost* (*admin s*) (*pass s* (*admin s*)) *aid PID*)
using *f-eq-PValS*[*OF step* φ *f*[*unfolded v*]] **.**
have *op′*: *open s′* **using** *step op PID PID′* φ
unfolding *open-def a* **by** (*auto simp*: *com-defs*)
have *ou*: *ou* ≠ *outErr* **using** φ *op op′* **unfolding** φ*-def2*[*OF step*] **unfolding** *a* **by** *auto*
have *pPID′*: *post s′ PID* = *post s PID*
using *step* φ *PID op f op′* **unfolding** *a* **by**(*auto simp*: *com-defs*)
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl′* **using** φ *vlvl1* **unfolding** *ss1 consume-def vl f* **by** *auto*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*

next
show *?Δ s′ vl′ s′ vl′* **using** *cor1 PID′ pPID′ op′ lvl vlvl1 ss1*
unfolding *Δ2-def vl v* **by** *auto*
qed
qed(*insert vl lvl*, *auto*)
next
case *False* **note** *φ = False*
hence *vl′*: *vl′ = vl* **using** *c* **unfolding** *consume-def* **by** *auto*
have *pPID′*: *post s′ PID = post s PID*
using *step φ PID op*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *fastforce*
subgoal by *fastforce*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op′*: *open s′* **using** *PID step φ op* **unfolding** *φ-def2*[*OF step*] **by** *auto*
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl* **using** *φ vlvl1* **unfolding** *ss1 consume-def vl′* **by** *simp*
next
show *γ ?trn = γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **thus** *g ?trn = g ?trn1* **unfolding** *ss1* **by** *simp*
next
show *?Δ s′ vl′ s′ vl* **using** *cor1 PID′ op′ lvl vlvl1 pPID′*
unfolding *Δ2-def vl′ ss1* **by** *auto*
qed
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vlvl1* **by** *simp*
qed
qed

lemma *unwind-cont-Δ4*: *unwind-cont Δ4* {*Δ1*,*Δ31*,*Δ32*,*Δ4*}
proof(*rule*, *simp*)
let *?Δ = λs vl s1 vl1. Δ1 s vl s1 vl1 ∨ Δ31 s vl s1 vl1 ∨ Δ32 s vl s1 vl1 ∨ Δ4 s vl s1 vl1*
fix *s s1 :: state* **and** *vl vl1 :: value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *Δ4 s vl s1 vl1*
then obtain *ul vll vll1* **where** *vl*: *vl = ul @ OVal False # vll* **and** *vl1*: *vl1 =*

ul @ *OVal False* # *vll1*
and *rs*: *reach s* **and** *ss1*: *s1* = *s* **and** *op*: *open s* **and** *PID*: *PID* ∈∈ *postIDs s*
and *cor1*: *corrFrom* (*post s1 PID*) *vl1* **and** *lul*: *list-all* (*Not* ∘ *isOVal*) *ul*
and *BC*: *BC vll vll1*
using *reachNT-reach* **unfolding** *Δ4-def* **by** *blast*
have *own*: *owner s PID* ∈ *set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
have *adm*: *admin s* ∈ *set* (*userIDs s*) **using** *reach-admin-userIDs*[*OF rs own*] **.**
show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'* **let** *?trn1* = *Trans s1 a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *c*: *consume ?trn vl vl'*
have *PID'*: *PID* ∈∈ *postIDs s'* **using** *reach-postIDs-persist*[*OF PID step*] **.**
show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
proof−
have *?match* **proof**(*cases φ ?trn*)
case *True* **note** *φ* = *True*
then obtain *v* **where** *vl-vl'*: *vl* = *v* # *vl'* **and** *f*: *f ?trn* = *v*
using *c* **unfolding** *consume-def φ-def2* **by**(*cases vl*) *auto*
show *?thesis* **proof**(*cases ul* = [])
case *False* **note** *ul* = *False*
then obtain *ul'* **where** *ul*: *ul* = *v* # *ul'* **and** *vl'*: *vl'* = *ul'* @ *OVal False* # *vll*
using *c φ f* **unfolding** *consume-def vl* **by** (*cases ul*) *auto*
let *?vl1'* = *ul'* @ *OVal False* # *vll1*
show *?thesis* **proof**(*cases v*)
case (*PVal pst*) **note** *v* = *PVal*
have *a*: *a* = *Uact* (*uPost* (*owner s PID*) (*pass s* (*owner s PID*)) *PID pst*)
using *f-eq-PVal*[*OF step φ f*[*unfolded v*]] **.**
have *ou*: *ou* = *outOK* **using** *step own PID* **unfolding** *a* **by** (*auto simp*: *u-defs*)
have *op'*: *open s'* **using** *step op PID PID' φ*
unfolding *open-def a* **by** (*auto simp*: *u-defs*)
have *pPID'*: *post s' PID* = *pst*
using *step φ PID op f op'* **unfolding** *a* **by**(*auto simp*: *u-defs*)
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 ?vl1'* **using** *φ*
unfolding *ss1 consume-def vl f ul vl1 vl'* **by** *simp*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*

```
        next
          have Δ4 s' vl' s' ?vl1' using cor1 PID' pPID' op' vl1 ss1 lul BC
          unfolding Δ4-def vl v ul vl' apply simp
          apply(rule exI[of - ul'])
          apply(rule exI[of - vll]) apply(rule exI[of - vll1])
          by auto
          thus ?Δ s' vl' s' ?vl1' by simp
        qed
      next
        case (PValS aid pid) note v = PValS
        have a: a = COMact (comSendPost (admin s) (pass s (admin s)) aid
PID)
        using f-eq-PValS[OF step φ f[unfolded v]] .
        have op': open s' using step op PID PID' φ
        unfolding open-def a by (auto simp: com-defs)
         have ou: ou ≠ outErr using φ op op' unfolding φ-def2[OF step]
unfolding a by auto
        have pPID': post s' PID = post s PID
        using step φ PID op f op' unfolding a by(auto simp: com-defs)
        show ?thesis proof
          show validTrans ?trn1 unfolding ss1 using step by simp
        next
          show consume ?trn1 vl1 ?vl1' using φ
          unfolding ss1 consume-def vl f ul vl1 vl' by simp
        next
          show γ ?trn = γ ?trn1 unfolding ss1 by simp
        next
          assume γ ?trn thus g ?trn = g ?trn1 unfolding ss1 by simp
        next
          have Δ4 s' vl' s' ?vl1' using cor1 PID' pPID' op' vl1 ss1 lul BC
          unfolding Δ4-def vl v ul vl' by auto
          thus ?Δ s' vl' s' ?vl1' by simp
        qed
      qed(insert vl lul ul, auto)
    next
      case True note ul = True
      hence f: f ?trn = OVal False and vl': vl' = vll
      using vl c f φ unfolding consume-def ul by auto
      have op': ¬ open s' using f-eq-OVal[OF step φ f] op by simp
      show ?thesis proof
        show validTrans ?trn1 using step unfolding ss1 by simp
      next
        show consume ?trn1 vl1 vll1 using ul φ c
        unfolding vl1 vl' vl ss1 consume-def by auto
      next
        show γ ?trn = γ ?trn1 unfolding ss1 by simp
      next
        assume γ ?trn note γ = this
        thus g ?trn = g ?trn1 unfolding ss1 by simp
```

```
    next
      have pPID': post s' PID = post s PID
        using step φ PID op op' f
        apply(cases a)
        subgoal for x1 apply(cases x1) apply(fastforce simp: s-defs)+ .
        subgoal for x2 apply(cases x2) apply(fastforce simp: c-defs)+ .
        subgoal for x3 apply(cases x3) apply(fastforce simp: d-defs)+ .
        subgoal for x4 apply(cases x4) apply(fastforce simp: u-defs)+ .
        subgoal by fastforce
        subgoal by fastforce
        subgoal for x7 apply(cases x7) apply(fastforce simp: com-defs)+ .
        done
      show ?Δ s' vl' s' vll1 using BC proof cases
        case BC-PVal
        hence Δ1 s' vl' s' vll1 using PID' pPID' op' cor1 BC lul
        unfolding Δ1-def vl1 ul ss1 vl' by auto
        thus ?thesis by simp
      next
        case (BC-BO Vll Vll1 Ul Ul1 Sul)
        show ?thesis proof(cases Ul ≠ [] ∧ Ul1 ≠ [])
          case True
          hence Δ31 s' vl' s' vll1 using PID' pPID' op' cor1 BC BC-BO lul
          unfolding Δ31-def vl1 ul ss1 vl' apply simp
          apply(rule exI[of - Ul]) apply(rule exI[of - Ul1])
          apply(rule exI[of - Sul])
          apply(rule exI[of - Vll]) apply(rule exI[of - Vll1])
          by auto
          thus ?thesis by simp
        next
          case False
          hence 0: Ul = [] Ul1 = [] using BC-BO by auto
          hence Δ32 s' vl' s' vll1 using PID' pPID' op' cor1 BC BC-BO lul
          unfolding Δ32-def vl1 ul ss1 vl' apply simp
          apply(rule exI[of - Sul])
          apply(rule exI[of - Vll]) apply(rule exI[of - Vll1])
          by auto
          thus ?thesis by simp
        qed
      qed
    qed
  qed
next
  case False note φ = False
  hence vl': vl' = vl using c unfolding consume-def by auto
  have pPID': post s' PID = post s PID
    using step φ PID op
    apply(cases a)
    subgoal for x1 apply(cases x1) apply(fastforce simp: s-defs)+ .
    subgoal for x2 apply(cases x2) apply(fastforce simp: c-defs)+ .
```

subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *fastforce*
subgoal by *fastforce*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op′*: *open s′* **using** *PID step φ op* **unfolding** *φ-def2*[*OF step*] **by** *auto*
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** *φ* **unfolding** *ss1 consume-def vl′ vl vl1* **by** *simp*
next
show *γ ?trn = γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **thus** *g ?trn = g ?trn1* **unfolding** *ss1* **by** *simp*
next
have *Δ4 s′ vl′ s′ vl1* **using** *cor1 PID′ pPID′ op′ vl1 ss1 lul BC*
unfolding *Δ4-def vl vl′* **by** *auto*
thus *?Δ s′ vl′ s′ vl1* **by** *simp*
qed
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vl* **by** *simp*
qed
qed

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*
apply (*rule unwind-decomp-secure-graph*[*of Gr Δ0*])
unfolding *Gr-def*
apply (*simp*, *smt insert-subset order-refl*)
using
istate-Δ0 unwind-cont-Δ0 unwind-cont-Δ1 unwind-cont-Δ11

unwind-cont-Δ31 unwind-cont-Δ32 unwind-cont-Δ2 unwind-cont-Δ4
unfolding *Gr-def* **by** *auto*

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 *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 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 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, vs1)) (comReceivePost aid1
sp2 pid2 pst2 uid2 vs2) =
   (aid1 = AID1 ∧ aid2 = AID2 ∧ (pid1, pst1, uid1, vs1) = (pid2, pst2, uid2,
vs2))
|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 v: validTrans trn1 and r: reach (srcOf trn1) and φ1: φ1 trn1
shows isCom1 trn1 ⟷ isComV1 (f1 trn1)
proof (cases trn1)
  case (Trans s1 a1 o1 s1')
  hence step: step s1 a1 = (o1, s1') using v by simp
  show ?thesis using φ1[unfolded Trans] unfolding Iss.φ-def3[OF step]
  proof (elim exE disjE conjE)
    assume Iss.open s1 ≠ Iss.open s1'
    and a1: ¬ isCOMact a1 ¬ (∃ ua. isuPost ua ∧ a1 = Uact ua)
   hence Iss.f (Trans s1 a1 o1 s1') = Iss.OVal (Iss.open s1') using Iss.f-open-OVal[OF
step] by auto
    thus ?thesis unfolding Trans using a1 by (cases a1) auto
  qed(unfold Trans, auto)
qed

lemma isCom1-isComO1:
assumes validTrans trn1 and reach (srcOf trn1) and γ1 trn1
shows isCom1 trn1 ⟷ isComO1 (g1 trn1)
using assms apply(cases trn1)
subgoal for - x2 apply(cases x2) by auto .

lemma isCom2-isComV2:
assumes validTrans trn2 and reach (srcOf trn2) and φ2 trn2
shows isCom2 trn2 ⟷ isComV2 (f2 trn2)
using assms apply(cases trn2) by (auto simp: Rcv.φ-def2 split: prod.splits)

lemma isCom2-isComO2:
assumes validTrans trn2 and reach (srcOf trn2) and γ2 trn2
shows isCom2 trn2 ⟷ isComO2 (g2 trn2)
using assms apply(cases trn2)
subgoal for - x2 apply(cases x2) by auto .

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 φ1: φ1 trn1 and φ2: φ2 trn2
and snc: sync trn1 trn2
shows syncV (f1 trn1) (f2 trn2)
proof (cases trn1)
  case (Trans s1 a1 o1 s1') note trn1 = Trans
  show ?thesis proof(cases trn2)
    case (Trans s2 a2 o2 s2') note trn2 = Trans
    have step1: step s1 a1 = (o1, s1') and step2: step s2 a2 = (o2, s2')
    using v1 v2 trn1 trn2 by auto
    obtain uid2 pst2 vs2
    where a2: a2 = COMact
        (comReceivePost AID1 (serverPass s2 AID1) PID pst2 uid2 vs2)
    and o2: o2 = outOK using φ2[unfolded trn2]
    unfolding Rcv.φ-def3[OF step2] by auto
    hence f2: Rcv.f trn2 = Rcv.PValR pst2 unfolding trn2 by simp
    show ?thesis using φ1[unfolded trn1]
    unfolding Iss.φ-def3[OF step1]
    proof (elim exE disjE conjE)
      assume Iss.open s1 ≠ Iss.open s1'
      and a1: ¬ isCOMact a1 ¬ (∃ ua. isuPost ua ∧ a1 = Uact ua)
      hence f1: Iss.f (Trans s1 a1 o1 s1') = Iss.OVal (Iss.open s1')
      using Iss.f-open-OVal step1 step2 by auto
      thus ?thesis using a1 c1 c2 unfolding trn1 trn2 a2 o2 f2
      by (cases a1, auto)
    qed(insert snc c1 c2, unfold trn1 trn2 a2, auto)
  qed
qed

lemma sync-syncO:
assumes validTrans trn1 and reach (srcOf trn1)
and validTrans trn2 and reach (srcOf trn2)
and isCom1 trn1 and isCom2 trn2 and γ1 trn1 and γ2 trn2
and sync trn1 trn2
shows syncO (g1 trn1) (g2 trn2)
proof(cases trn1)
  case (Trans s1 a1 ou1 s1') note trn1 = Trans
  show ?thesis proof(cases trn2)
    case (Trans s2 a2 ou2 s2') note trn2 = Trans
    show ?thesis
    proof(cases a1)
      case (COMact ca1) note a1 = COMact
      show ?thesis
      proof(cases a2)
        case (COMact ca2) note a2 = COMact
        show ?thesis
        using assms unfolding trn1 trn2 a1 a2
        apply(cases ca1) by (cases ca2, auto split: prod.splits)+
```

qed(*insert assms*, *unfold trn1 trn2*, *auto*)
qed(*insert assms*, *unfold trn1 trn2*, *auto*)
qed
qed

lemma *sync-φ1-φ2*:
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 *φ1 trn1* ⟷ *φ2 trn2* (**is** *?A* ⟷ *?B*)
proof(*cases trn1*)
case (*Trans s1 a1 ou1 s1'*) **note** *trn1* = *Trans*
hence *step1*: *step s1 a1* = (*ou1*,*s1'*) **using** *v1* **by** *auto*
show *?thesis* **proof**(*cases trn2*)
case (*Trans s2 a2 ou2 s2'*) **note** *trn2* = *Trans*
hence *step2*: *step s2 a2* = (*ou2*,*s2'*) **using** *v2* **by** *auto*
show *?thesis*
proof(*cases a1*)
case (*COMact ca1*) **note** *a1* = *COMact*
show *?thesis*
proof(*cases a2*)
case (*COMact ca2*) **note** *a2* = *COMact*

have *?A* ⟷ (∃ *aid1*. *ca1* =
(*comSendPost* (*admin s1*) (*pass s1* (*admin s1*)) *aid1*
PID) ∧
ou1 =
O-sendPost
(*aid1*, *clientPass s1 aid1*, *PID*, *post s1 PID*,
owner s1 PID, *vis s1 PID*))
using *c1* **unfolding** *trn1 Iss.φ-def3*[*OF step1*] **unfolding** *a1* **by** *auto*
also have ... ⟷ (∃ *uid2 pst2 vs2*.
ca2 = *comReceivePost AID1* (*serverPass s2 AID1*) *PID pst2 uid2 vs2* ∧
ou2 = *outOK*)
using *sn step1 step2* **unfolding** *trn1 trn2 a1 a2*
apply(*cases ca1*) **by** (*cases ca2*, *auto simp*: *all-defs*)+
also have ... ⟷ *?B*
using *c2* **unfolding** *trn2 Rcv.φ-def3*[*OF step2*] **unfolding** *a2* **by** *auto*
finally show *?thesis* **.**
qed(*insert assms*, *unfold trn1 trn2*, *auto*)
qed(*insert assms*, *unfold trn1 trn2*, *auto*)
qed
qed

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

using *assms* **by** (*cases pst1*, *cases pst2*) *auto*

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 *γ1 trn1* **and** *γ2 trn2*
and *so*: *syncO* (*g1 trn1*) (*g2 trn2*)
and *φ1 trn1* ⟹ *φ2 trn2* ⟹ *syncV* (*f1 trn1*) (*f2 trn2*)
shows *sync trn1 trn2*
proof(*cases trn1*, *cases trn2*)
  fix *s1 a1 ou1 s1′ s2 a2 ou2 s2′*
  assume *trn1*: *trn1* = *Trans s1 a1 ou1 s1′*
  and *trn2*: *trn2* = *Trans s2 a2 ou2 s2′*
  hence *step1*: *step s1 a1* = (*ou1*,*s1′*) **and** *step2*: *step s2 a2* = (*ou2*,*s2′*) **using** *assms* **by** *auto*
  show *?thesis*
  proof(*cases a1*)
    case (*COMact ca1*) **note** *a1* = *COMact*
    show *?thesis*
    proof(*cases a2*)
      case (*COMact ca2*) **note** *a2* = *COMact*
      show *?thesis*
      proof(*cases ca1*) **term** *comReceivePost*
        case (*comSendPost uid1 p1 aid1 pid*) **note** *ca1* = *comSendPost*
        then obtain *pst* **where** *p1*: *p1* = *pass s1* (*admin s1*) **and**
        aid1: *aid1* = *AID2* **and** *ou2*: *ou2* = *outOK* **and** *ou1*: *ou1* ≠ *outErr* **and**
        ca2: *ca2* = *comReceivePost AID1* (*serverPass s2 AID1*) *pid pst* (*owner s1 pid*) (*vis s1 pid*)
        using *so step1 step2* **unfolding** *trn1 trn2 a1 a2 ca1*
        by (*cases ca2*, *auto simp*: *all-defs*)
        have *ou1*: *ou1* = *O-sendPost* (*AID2*,*clientPass s1 AID2*,*pid*, *post s1 pid*, *owner s1 pid*, *vis s1 pid*)
        using *step1 ou1* **unfolding** *a1 ca1 aid1* **by** (*auto simp*: *all-defs*)
        show *?thesis* **proof**(*cases pid* = *PID*)
          case *False* **thus** *?thesis* **using** *so step1 step2* **unfolding** *trn1 trn2 a1 a2 ca1 ca2*
          by (*auto simp*: *all-defs*)
        next
          case *True* **note** *pid* = *True*
          hence *φ1 trn1* ∧ *φ2 trn2* **using** *ou1 ou2* **unfolding** *trn1 trn2 a1 a2 ca1 ca2* **by** *auto*
          hence *syncV* (*f1 trn1*) (*f2 trn2*) **using** *assms* **by** *simp*
          hence *pst*: *pst* = *post s1 PID* **using** *pid* **unfolding** *trn1 trn2 a1 a2 ca1 ca2 aid1 ou1* **by** *auto*
          show *?thesis* **unfolding** *trn1 trn2 a1 a2 ca1 ca2 ou1 ou2 pst pid* **by** *auto*
        qed
      qed(*insert so step1 step2*, *unfold trn1 trn2 a1 a2*, (*cases ca2*, *auto simp*: *all-defs*)+)

qed(*insert assms*, *unfold trn1 trn2*, *auto*)
qed(*insert assms*, *unfold trn1 trn2*, *auto*)
qed

lemma *isCom1-γ1*:
assumes *validTrans trn1* **and** *reach* (*srcOf trn1*) **and** *isCom1 trn1*
shows *γ1 trn1*
proof(*cases trn1*)
case (*Trans s1 a1 ou1 s1'*)
thus *?thesis* **using** *assms* **by** (*cases a1*) *auto*
qed

lemma *isCom2-γ2*:
assumes *validTrans trn2* **and** *reach* (*srcOf trn2*) **and** *isCom2 trn2*
shows *γ2 trn2*
proof(*cases trn2*)
case (*Trans s2 a2 ou2 s2'*)
thus *?thesis* **using** *assms* **by** (*cases a2*) *auto*
qed

lemma *isCom2-V2*:
assumes *validTrans trn2* **and** *reach* (*srcOf trn2*) **and** *φ2 trn2*
shows *isCom2 trn2*
proof(*cases trn2*)
case (*Trans s2 a2 ou2 s2'*) **note** *trn2 = Trans*
show *?thesis*
proof(*cases a2*)
case (*COMact ca2*)
thus *?thesis* **using** *assms trn2* **by** (*cases ca2*) *auto*
qed(*insert assms trn2*, *auto*)
qed

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*
apply *standard*

using *isCom1-isComV1 isCom1-isComO1 isCom2-isComV2 isCom2-isComO2 sync-syncV sync-syncO*
apply *auto*
apply (*meson sync-φ1-φ2*, *meson sync-φ1-φ2*)
using *sync-φ-γ* **apply** *auto*
using *isCom1-γ1 isCom2-γ2 isCom2-V2* **apply** *auto*
by (*meson isCom2-V2*)

context *Post-COMPOSE2*
begin

theorem *secure*: *secure*
using *secure1-secure2-secure*[*OF Iss.secure Rcv.Post-secure*] **.**

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 (*cases v*)
case (*Inl vl*) **then show** *thesis* **using** *PVal PValS OVal* **by** (*cases vl rule*: *Post.value.exhaust*) *auto* **next**
case (*Inr vr*) **then show** *thesis* **using** *PValR* **by** (*cases vr rule*: *Post-RECEIVER.value.exhaust*) *auto*
qed

locale *Post-Network* = *Network*
+ **fixes** *UIDs* :: *apiID* ⇒ *userID set*

and *AID* :: *apiID* **and** *PID* :: *postID*
assumes *AID-in-AIDs*: $AID \in AIDs$
begin

sublocale *Iss*: *Post UIDs AID PID* **.**

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*
by *auto*

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*))
unfolding *syncV-def* **by** *auto*

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*)
using *assms* **unfolding** *syncV-def* **by** *auto*

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*)
by (*cases v rule*: *value-cases*; *auto*)+

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*
using *AID-in-AIDs* **proof** (*unfold-locales*, *goal-cases*)
case (*1 nid trn*) **then show** *?case* **using** *Iss.validTrans-isCOMact-open*[*of trn*] **by** (*cases trn rule*: *Iss.φ.cases*) (*auto simp*: *φ-defs split*: *prod.splits*) **next**
case (*2 nid trn*) **then show** *?case* **using** *Iss.validTrans-isCOMact-open*[*of trn*] **by** (*cases trn rule*: *Iss.φ.cases*) (*auto simp*: *φ-defs split*: *prod.splits*) **next**
case (*3 nid trn*)
interpret *Sink*: *Post-RECEIVER UIDs nid PID AID* **.**
show *?case* **using** *3* **by** (*cases* (*nid*,*trn*) *rule*: *tgtNodeOf.cases*) (*auto split*: *prod.splits*)
next
case (*4 nid trn*)
interpret *Sink*: *Post-RECEIVER UIDs nid PID AID* **.**

show *?case* **using** *4* **by** (*cases* (*nid*,*trn*) *rule*: *tgtNodeOf.cases*) (*auto split*: *prod.splits*)
next
case (*5 nid1 trn1 nid2 trn2*)
interpret *Sink1*: *Post-RECEIVER UIDs nid1 PID AID* **.**
interpret *Sink2*: *Post-RECEIVER UIDs nid2 PID AID* **.**
show *?case* **using** *5* **by** (*elim sync-cases*) (*auto intro*: *syncVI*)
next
case (*6 nid1 trn1 nid2 trn2*)
interpret *Sink1*: *Post-RECEIVER UIDs nid1 PID AID* **.**
interpret *Sink2*: *Post-RECEIVER UIDs nid2 PID AID* **.**
show *?case* **using** *6* **by** (*elim sync-cases*) *auto*
next
case (*7 nid1 trn1 nid2 trn2*)
interpret *Sink1*: *Post-RECEIVER UIDs nid1 PID AID* **.**
interpret *Sink2*: *Post-RECEIVER UIDs nid2 PID AID* **.**
show *?case* **using** *7*(*2*,*4*,*6*−*10*)
using *Iss.validTrans-isCOMact-open*[*OF 7*(*2*)] *Iss.validTrans-isCOMact-open*[*OF 7*(*4*)]
by (*elim sync-cases*) (*auto split*: *prod.splits*, *auto simp*: *sendPost-def*)
next
case (*8 nid1 trn1 nid2 trn2*)
interpret *Sink1*: *Post-RECEIVER UIDs nid1 PID AID* **.**
interpret *Sink2*: *Post-RECEIVER UIDs nid2 PID AID* **.**
show *?case* **using** *8*(*2*,*4*,*6*−*10*,*11*,*12*,*13*)
apply (*elim syncO-cases*; *cases trn1*; *cases trn2*)
apply (*auto simp*: *Iss.g-simps ObservationSetup-RECEIVER.g-simps split*: *prod.splits*)
apply (*auto simp*: *sendPost-def split*: *prod.splits elim*: *syncVE*)[]
done
next
case (*9 nid trn*)
then show *?case*
by (*cases* (*nid*,*trn*) *rule*: *tgtNodeOf.cases*)
(*auto simp*: *ObservationSetup-RECEIVER.γ.simps*)
next
case (*10 nid trn*) **then show** *?case* **by** (*cases trn*) (*auto simp*: *φ-defs*)
next
case (*11 vSrc nid vn*) **then show** *?case* **by** (*cases vSrc rule*: *value-cases*) (*auto simp*: *syncV-def*)
next
case (*12 vSrc nid vn*) **then show** *?case* **by** (*cases vSrc rule*: *value-cases*) (*auto simp*: *syncV-def*)
qed

lemma *list-all-Not-isl-projectSrcV*: *list-all* (*Not o isl*) (*Net.projectSrcV aid vlSrc*)
proof (*induction vlSrc*)
case (*Cons vSrc vlSrc′*) **then show** *?case* **by** (*cases vSrc rule*: *value-cases*) *auto*
qed *auto*

context
fixes *AID′* :: *apiID*
assumes *AID′*: $AID' \in AIDs - \{AID\}$
begin

interpretation *Recv*: *Post-RECEIVER UIDs AID′ PID AID* **by** *unfold-locales*

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*))
by (*induction rule*: *Iss.BC-BO.induct*) *auto*

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*))
using *assms* **proof** (*induction vl vl1 rule*: *list22-induct*)
case (*ConsCons v vl v1 vl1*)
consider (*SendSend*) *aid pst pst1* **where** *v* = *Inl* (*Iss.PValS aid pst*) *v1* = *Inl* (*Iss.PValS aid pst1*)
| (*Internal*) *comOfV AID v* = *Internal* ¬*Iss.isPValS* (*projl v*)
| (*Internal1*) *comOfV AID v1* = *Internal* ¬*Iss.isPValS* (*projl v1*)
using *ConsCons*(*4*−*6*) **by** (*cases v rule*: *value-cases*; *cases v1 rule*: *value-cases*) *auto*
then show *?case* **proof** *cases*
case (*SendSend*) **then show** *?thesis* **using** *ConsCons.IH*(*1*) *ConsCons.prems* **by** *auto*
next
case (*Internal*) **then show** *?thesis* **using** *ConsCons.IH*(*2*)[*of v1 # vl1*] *ConsCons.prems* **by** *auto*
next
case (*Internal1*) **then show** *?thesis* **using** *ConsCons.IH*(*3*)[*of v # vl*] *ConsCons.prems* **by** *auto*
qed
qed (*auto simp*: *comOfV-AID*)

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*))
using *assms Iss-B-Recv-B-aux Iss-BC-BO-tgtAPI* **by** (*auto simp*: *Iss.B-def Recv.B-def*)

end

lemma *map-projl-Inl*: *map* (*projl o Inl*) *vl* = *vl*
by (*induction vl*) *auto*

lemma *these-map-Inl-projl*: *list-all isl vl* ⟹ *these* (*map* (*Some o Inl o projl*) *vl*) = *vl*
by (*induction vl*) *auto*

lemma *map-projr-Inr*: *map* (*projr o Inr*) *vl* = *vl*
by (*induction vl*) *auto*

lemma *these-map-Inr-projr*: *list-all* (*Not o isl*) *vl* ⟹ *these* (*map* (*Some o Inr o projr*) *vl*) = *vl*
by (*induction vl*) *auto*

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 (*unfold-locales*, *goal-cases*)
case *1* **show** *?case* **using** *AID-in-AIDs* **.**
next
case *2*
interpret *Iss'*: *BD-Security-TS-Trans*
istate System-Specification.validTrans srcOf tgtOf Iss.φ Iss.f Iss.γ Iss.g Iss.T Iss.B
istate System-Specification.validTrans srcOf tgtOf Iss.φ λtrn. Inl (*Iss.f trn*) *Iss.γ Iss.g Iss.T B AID*
id id Some Some o Inl
proof (*unfold-locales*, *goal-cases*)
case (*11 vl' vl1' tr*) **then show** *?case*
by (*intro exI*[*of - map projl vl1'*]) (*auto simp*: *map-projl-Inl these-map-Inl-projl*)
qed *auto*
show *?case* **using** *Iss.secure Iss'.translate-secure* **by** *auto*
next
case (*3 aid tr vl' vl1*)
then show *?case*
using *Iss-B-Recv-B*[*of aid* (*Net.lV AID tr*) *vl1*] *list-all-Not-isl-projectSrcV*
by *auto*
qed

theorem *secure*: *secure*
proof (*intro preserve-source-secure ballI*)
fix *aid*

assume *aid*: $aid \in AIDs - \{AID\}$
interpret *Node*: *Post-RECEIVER UIDs aid PID AID* **.**
interpret *Node'*: *BD-Security-TS-Trans*
istate System-Specification.validTrans srcOf tgtOf Node.φ *Node.f Node.*γ *Node.g Node.T Node.B*
istate System-Specification.validTrans srcOf tgtOf Node.φ λ*trn. Inr* (*Node.f trn*) *Node.*γ *Node.g Node.T B aid*
id id Some Some o Inr
proof (*unfold-locales*, *goal-cases*)
case (*11 vl' vl1' tr*) **then show** *?case* **using** *aid*
by (*intro exI*[*of - map projr vl1'*]) (*auto simp*: *map-projr-Inr these-map-Inr-projr*)
qed *auto*
show *Net.lsecure aid*
using *aid Node.Post-secure Node'.translate-secure* **by** *auto*
qed

end

end

theory *Independent-Post-Observation-Setup-ISSUER*
imports
../../Safety-Properties
../Post-Observation-Setup-ISSUER
begin

6.6 Variation with multiple independent secret posts

This section formalizes the lifting of the confidentiality of one given (arbitrary but fixed) post to the confidentiality of two posts of arbitrary nodes of the network, as described in [3, Appendix E].

6.6.1 Issuer observation setup

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

fun γ :: (*state*,*act*,*out*) *trans* $\Rightarrow$ *bool* **where**
γ (*Trans - a - -*) $\longleftrightarrow$
($\exists$ *uid. userOfA a* = *Some uid* $\wedge$ *uid* $\in$ *UIDs*)
$\vee$
— Communication actions are considered to be observable in order to make the security properties compositional
($\exists$ *ca. a* = *COMact ca*)
$\vee$
— The following actions are added to strengthen the observers in order to show that all posts *other than PID* are completely independent of *PID*; the confidentiality

of *PID* is protected even if the observers can see all updates to other posts (and actions contributing to the declassification triggers of those posts).

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

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'*
by (*cases ca*; *auto*)+

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*
by (*cases ou*; *auto simp*: *Strong-ObservationSetup-ISSUER.outPurge.simps*)+

lemma *g-simps*:
g (*Trans s a ou s'*) = (*COMact* (*comSendServerReq uID p aID reqInfo*), *O-sendServerReq ossr*)
$\longleftrightarrow$ ($\exists$ *p'*. *a = COMact* (*comSendServerReq uID p' aID reqInfo*) $\wedge$ *p = emptyPass* $\wedge$ *ou = O-sendServerReq ossr*)
g (*Trans s a ou s'*) = (*COMact* (*comReceiveClientReq aID reqInfo*), *outOK*)
$\longleftrightarrow$ *a = COMact* (*comReceiveClientReq aID reqInfo*) $\wedge$ *ou = outOK*
g (*Trans s a ou s'*) = (*COMact* (*comConnectClient uID p aID sp*), *O-connectClient occ*)
$\longleftrightarrow$ ($\exists$ *p'*. *a = COMact* (*comConnectClient uID p' aID sp*) $\wedge$ *p = emptyPass* $\wedge$ *ou = O-connectClient occ*)
g (*Trans s a ou s'*) = (*COMact* (*comConnectServer aID sp*), *outOK*)
$\longleftrightarrow$ *a = COMact* (*comConnectServer aID sp*) $\wedge$ *ou = outOK*
g (*Trans s a ou s'*) = (*COMact* (*comReceivePost aID sp nID nt uID v*), *outOK*)
$\longleftrightarrow$ *a = COMact* (*comReceivePost aID sp nID nt uID v*) $\wedge$ *ou = outOK*

g (*Trans s a ou s'*) = (*COMact* (*comSendPost uID p aID nID*), *O-sendPost* (*aid*, *sp*, *pid*, *pst'*, *uid*, *vs*))
⟷ (∃ *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'*))
⟷ (∃ *p'*. *a* = *COMact* (*comSendDeleteOFriend uID p' aID uID'*) ∧ *p* = *emptyPass* ∧ *ou* = *O-sendDeleteOFriend* (*aid*, *sp*, *uid*, *uid'*))
g (*Trans s a ou s'*) = (*COMact* (*comReceiveDeleteOFriend aID cp uID uID'*), *outOK*)
⟷ *a* = *COMact* (*comReceiveDeleteOFriend aID cp uID uID'*) ∧ *ou* = *outOK*
by (*cases a*; *auto simp*: *comPurge-simps*)+

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* **.**

lemma *eqButPID-open*:
assumes *eqButPID s s1*
shows *open s* $\longleftrightarrow$ *open s1*
using *eqButPID-stateSelectors*[*OF assms*]
unfolding *open-def* **by** *auto*

lemma *not-open-eqButPID*:
assumes *1*: ¬ *open s* **and** *2*: *eqButPID s s1*
shows ¬ *open s1*
using *1* **unfolding** *eqButPID-open*[*OF 2*] **.**

lemma *step-isCOMact-open*:
assumes *step s a* = (*ou*, *s'*)
and *isCOMact a*
shows *open s'* = *open s*
using *assms* **proof** (*cases a*)
  case (*COMact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *open-def com-defs*)
qed *auto*

lemma *validTrans-isCOMact-open*:
assumes *validTrans trn*
and *isCOMact* (*actOf trn*)
shows *open* (*tgtOf trn*) = *open* (*srcOf trn*)
using *assms step-isCOMact-open* **by** (*cases trn*) *auto*

lemma *list-all-isOVal-filter-isPValS*:
list-all isOVal vl $\Longrightarrow$ *filter* (*Not o isPValS*) *vl* = *vl*
by (*induct vl*) *auto*

lemma *list-all-Not-isOVal-OVal-True*:
assumes *list-all* (*Not* ∘ *isOVal*) *ul*
and *ul* @ *vll* = *OVal True* # *vll'*
shows *ul* = []
using *assms* **by** (*cases ul*) *auto*

lemma *list-all-filter-isOVal-isPVal-isPValS*:
assumes *list-all* (*Not* ∘ *isOVal*) *ul*
and *filter isPValS ul* = [] **and** *filter isPVal ul* = []
shows *ul* = []
using *assms value.exhaust-disc* **by** (*induct ul*) *auto*

lemma *filter-list-all-isPValS-isOVal*:
assumes *list-all* (*Not* ∘ *isOVal*) *ul* **and** *filter isPVal ul* = []
shows *list-all isPValS ul*
using *assms value.exhaust-disc* **by** (*induct ul*) *auto*

lemma *filter-list-all-isPVal-isOVal*:
assumes *list-all* (*Not* ∘ *isOVal*) *ul* **and** *filter isPValS ul* = []
shows *list-all isPVal ul*
using *assms value.exhaust-disc* **by** (*induct ul*) *auto*

lemma *list-all-isPValS-Not-isOVal-filter*:
assumes *list-all isPValS ul* **shows** *list-all* (*Not* ∘ *isOVal*) *ul* ∧ *filter isPVal ul* = []
using *assms value.exhaust-disc* **by** (*induct ul*) *auto*

lemma *filter-isTValS-Nil*:
filter isPValS vl = [] ⟷
list-all (λ *v*. *isPVal v* ∨ *isOVal v*) *vl*
proof(*induct vl*)
case (*Cons v vl*)
thus *?case* **by** (*cases v*) *auto*
qed *auto*

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*))
by (*cases trn rule*: φ.*cases*) *auto*

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'*
using *assms*
by (*cases Trans s a ou s' rule*: φ.*cases*) (*auto simp*: *all-defs open-def*)

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*
using *1 2* **unfolding** *a* **by** (*auto simp*: *u-defs*)

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*)
using *1 2* **unfolding** *a* **by** (*auto simp*: *com-defs*)

lemma *step-open-isCOMact*:
assumes *step s a* = (*ou*,*s'*)
and *open s* ≠ *open s'*
shows ¬ *isCOMact a* ∧ ¬ (∃ *ua*. *isuPost ua* ∧ *a* = *Uact ua*)
using *assms* **unfolding** *open-def*
apply(*cases a*)
subgoal by (*auto simp*: *all-defs*)
subgoal by (*auto simp*: *all-defs*)
subgoal by (*auto simp*: *all-defs*)
subgoal for *x4* **by** (*cases x4*) (*auto simp*: *all-defs*)
subgoal by (*auto simp*: *all-defs*)
subgoal by (*auto simp*: *all-defs*)
subgoal for *x7* **by** (*cases x7*) (*auto simp*: *all-defs*)
done

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*)
unfolding *φ-def2*[*OF assms*]
using *comSendPost-out*[*OF assms*] *uTextPost-out*[*OF assms*]
step-open-isCOMact[*OF assms*]
by *blast*

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'*)
using *assms* **by** (*cases Trans s a ou s' rule*: *f.cases*) *auto*

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*)
using *assms* **by** (*cases Trans s a ou s' rule*: *f.cases*) (*auto simp*: *u-defs com-defs*)

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*)
using *assms* **by** (*cases Trans s a ou s' rule*: *f.cases*) (*auto simp*: *com-defs*)

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*
using *assms* **by** (*auto simp*: φ*-def2 com-defs*)

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*
using *assms a* **unfolding** *open-def*
by (*cases a*) (*auto simp*: *u-defs com-defs*)

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−
  have *a*: *a* = *Uact* (*uPost* (*owner s PID*) (*pass s* (*owner s PID*)) *PID pst*)
  using *f-eq-PVal*[*OF step fi f*] **.**
  have *ou*: *ou* = *outOK* **using** *fi op* **unfolding** *a* φ*-def2*[*OF step*] **by** *auto*

have *owner s PID* ∈∈ *userIDs s* **using** *s* **by** (*simp add*: *PID reach-owner-userIDs*)
hence *owner s PID* ∉ *UIDs* **using** *op PID* **unfolding** *open-def* **by** *auto*
thus *?thesis* **unfolding** *a* **by** *simp*
qed

lemma *Uact-uPaperC-step-eqButPID*:
assumes *a*: *a* = *Uact* (*uPost uid p PID pst*)
and *step s a* = (*ou*,*s'*)
shows *eqButPID s s'*
using *assms* **unfolding** *eqButPID-def eeqButPID-def eeqButPID-F-def*
by (*auto simp*: *all-defs split*: *if-splits*)

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−
have *s's1'*: *eqButPID s' s1'*
using *eqButPID-step local.step ss1 step1* **by** *blast*
show *?thesis* **using** *step step1 φ eqButPID-open*[*OF ss1*] *eqButPID-open*[*OF s's1'*]
using *eqButPID-stateSelectors*[*OF ss1*]
unfolding *φ-def2*[*OF step*] *φ-def2*[*OF step1*]
by (*auto simp*: *all-defs*)
qed

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'*)
by (*metis eqButPID-step-φ-imp eqButPID-sym assms*)

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* = []
using *assms value.exhaust-disc* **by** (*induct ul*) *auto*

lemma *BC-not-Nil*: *BC vl vl1* ⟹ *vl* = [] ⟹ *vl1* = []
by(*auto elim*: *BC.cases*)

lemma *BC-OVal-True*:
assumes *BC* (*OVal True* # *vl'*) *vl1*
shows ∃ *vl1'*. *BO vl' vl1'* ∧ *vl1* = *OVal True* # *vl1'*
proof−
define *vl* **where** *vl*: *vl* ≡ *OVal True* # *vl'*
have *BC vl vl1* **using** *assms* **unfolding** *vl* **by** *auto*
thus *?thesis* **proof** *cases*

case (*BC-BO vll vll1 ul ul1 sul*)
hence *ul*: *ul* = [] **unfolding** *vl* **apply** *simp*
by (*metis* (*no-types*) *Post.value.disc*(*9*) *append-eq-Cons-conv*
list.map(*2*) *list.pred-inject*(*2*) *list-all-map*)
have *sul*: *sul* = [] **using** *BC-BO* **unfolding** *vl ul* **apply** *simp*
by (*metis Post.value.disc*(*6*) *append-eq-Cons-conv list.pred-inject*(*2*))
show *?thesis*
apply − **apply**(*rule exI*[*of - vll1*])
using *BC-BO* **using** *list-all-filter-Not-isOVal*[*of ul1*]
unfolding *ul sul vl* **by** *auto*
qed(*unfold vl*, *auto*)
qed

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* = []
using *B Post.BC-not-Nil Post.B-def vl* **by** *blast*

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*
done

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*
using *assms reach-vis* **by** *auto*

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−
note [*simp*] = *all-defs*
open-def
note *s* = *reachNT-reach*[*OF sT*]
note *willUse* =
step step1 γ
not-open-eqButPID[*OF op ss1*]
reach-vis[*OF s*]
eqButPID-stateSelectors[*OF ss1*]
eqButPID-actions[*OF ss1*]
eqButPID-update[*OF ss1*] *eqButPID-not-PID*[*OF ss1*]

eqButPID-eqButF[*OF ss1*]
eqButPID-setShared[*OF ss1*] *eqButPID-updateShared*[*OF ss1*]
eeqButPID-F-not-PID eqButPID-not-PID-sharedWith
eqButPID-insert2[*OF ss1*]
show *?thesis*
proof (*cases a*)
case (*Sact x1*)
with *willUse* **show** *?thesis* **by** (*cases x1*) *auto*
next
case (*Cact x2*)
with *willUse* **show** *?thesis* **by** (*cases x2*) *auto*
next
case (*Dact x3*)
with *willUse* **show** *?thesis* **by** (*cases x3*) *auto*
next
case (*Uact x4*)
with *willUse* **show** *?thesis* **by** (*cases x4*) *auto*
next
case (*Ract x5*)
with *willUse* **show** *?thesis*
proof (*cases x5*)
case (*rPost uid p pid*)
with *Ract willUse* **show** *?thesis* **by** (*cases pid* = *PID*) *auto*
qed *auto*
next
case (*Lact x6*)
with *willUse* **show** *?thesis*
proof (*cases x6*)
case (*lClientsPost uid p pid*)

with *Lact willUse* **show** *?thesis* **by** (*cases pid* = *PID*) *auto*
qed *auto*
next
case (*COMact x7*)
with *willUse* **show** *?thesis* **by** (*cases x7*) *auto*
qed
qed

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'*
using *ss1 step step1*
using *eqButPID-stateSelectors*[*OF ss1*]
eqButPID-update[*OF ss1*] *eqButPID-setShared*[*OF ss1*]
unfolding *a* **by** (*auto simp*: *u-defs*)

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 $\Delta 31$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 31$ *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* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 32$ *s vl s1 vl1* ≡
PID ∈∈ *postIDs s* ∧
(∃ *sul vll vll1*.
BO vll vll1 ∧
list-all isPValS sul ∧
vl = *sul* @ *OVal True* # *vll* ∧ *vl1* = *sul* @ *OVal True* # *vll1*) ∧
s = *s1* ∧ ¬ *open s* ∧
corrFrom (*post s1 PID*) *vl1*

definition $\Delta 4$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 4$ *s vl s1 vl1* ≡
PID ∈∈ *postIDs s* ∧
(∃ *ul vll vll1*.
BC vll vll1 ∧
list-all (*Not o isOVal*) *ul* ∧
vl = *ul* @ *OVal False* # *vll* ∧ *vl1* = *ul* @ *OVal False* # *vll1*) ∧
s = *s1* ∧ *open s* ∧
corrFrom (*post s1 PID*) *vl1*

lemma *istate-*$\Delta 0$:
assumes *B*: *B vl vl1*
shows $\Delta 0$ *istate vl istate vl1*
using *assms* **unfolding** $\Delta 0$*-def istate-def B-def* **by** *auto*

lemma *list-all-filter*[*simp*]:
assumes *list-all PP xs*
shows *filter PP xs* = *xs*
using *assms* **by** (*induct xs*) *auto*

lemma *unwind-cont-*$\Delta 0$: *unwind-cont* $\Delta 0$ {$\Delta 0$,$\Delta 1$,$\Delta 2$,$\Delta 31$,$\Delta 32$,$\Delta 4$}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. $\Delta 0$ *s vl s1 vl1* ∨
$\Delta 1$ *s vl s1 vl1* ∨ $\Delta 2$ *s vl s1 vl1* ∨

Δ31 s vl s1 vl1 ∨ *Δ32 s vl s1 vl1* ∨ *Δ4 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *Δ0 s vl s1 vl1*
hence *rs*: *reach s* **and** *ss1*: *s1* = *s* **and** *BC*: *BC vl vl1* **and** *PID*: ¬ *PID* ∈∈ *postIDs s*
and *cor1*: *corr vl1* **using** *reachNT-reach* **unfolding** *Δ0-def* **by** *auto*
have *vis*: *vis s PID* = *FriendV* **using** *reach-not-postIDs-friendV*[*OF rs PID*] **.**
have *pPID*: *post s1 PID* = *emptyPost* **by** (*simp add*: *PID reach-not-postIDs-emptyPost rs ss1*)
have *vlvl1*: *vl* = [] ⟹ *vl1* = [] **using** *BC-not-Nil BC* **by** *auto*
have *op*: ¬ *open s* **using** *PID* **unfolding** *open-def* **by** *auto*
show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn* = *Trans s a ou s′* **let** *?trn1* = *Trans s1 a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl′*
hence *pPID′*: *post s′ PID* = *emptyPost* **using** *step pPID ss1 PID*
apply (*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
show *match ?Δ s s1 vl1 a ou s′ vl′* ∨ *ignore ?Δ s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
proof−
have *?match*
proof(*cases* ∃ *uid p*. *a* = *Cact* (*cPost uid p PID*) ∧ *ou* = *outOK*)
case *True*
then obtain *uid p* **where** *a*: *a* = *Cact* (*cPost uid p PID*) **and** *ou*: *ou* = *outOK* **by** *auto*
have *PID′*: *PID* ∈∈ *postIDs s′*
using *step PID* **unfolding** *a ou* **by** (*auto simp*: *c-defs*)
show *?thesis* **proof**(*cases*
∃ *uid′* ∈ *UIDs*. *uid′* ∈∈ *userIDs s* ∧
(*uid′* = *admin s* ∨ *uid′* = *uid* ∨ *uid′* ∈∈ *friendIDs s uid*))
case *True* **note** *uid* = *True*
have *op′*: *open s′* **using** *uid* **using** *step PID′* **unfolding** *a ou* **by** (*auto simp*: *c-defs open-def*)
have *φ*: *φ ?trn* **using** *op op′* **unfolding** *φ-def2*[*OF step*] **by** *simp*
then obtain *v* **where** *vl*: *vl* = *v* # *vl′* **and** *f*: *f ?trn* = *v*
using *c* **unfolding** *consume-def φ-def2* **by**(*cases vl*) *auto*
have *v*: *v* = *OVal True* **using** *f op op′* **unfolding** *a* **by** *simp*
then obtain *ul1 vl1′* **where** *BO′*: *BO vl′ vl1′* **and** *vl1*: *vl1* = *ul1* @

OVal True # vl1'
and *ul1*: *list-all* (*Not* ∘ *isOVal*) *ul1*
using *BC-OVal-True*[*OF BC*[*unfolded vl v*]] **by** *auto*
have *ul1*: *ul1* = []
using *BC BC-OVal-True list-all-Not-isOVal-OVal-True ul1 v vl vl1* **by** *blast*
hence *vl1*: *vl1* = *OVal True # vl1'* **using** *vl1* **by** *simp*
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl1'* **using** *φ f* **unfolding** *vl1 v consume-def ss1* **by** *simp*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
show *?Δ s' vl' s' vl1'* **using** *BO'* **proof**(*cases rule*: *BO.cases*)
case (*BO-PVal*)
hence *Δ2 s' vl' s' vl1'* **using** *PID' op' cor1* **unfolding** *Δ2-def vl1 pPID'* **by** *auto*
thus *?thesis* **by** *simp*
next
case (*BO-BC vll vll1 textl*)
hence *Δ4 s' vl' s' vl1'* **using** *PID' op' cor1* **unfolding** *Δ4-def vl1 pPID'* **by** *auto*
thus *?thesis* **by** *simp*
qed
qed
next
case *False* **note** *uid* = *False*
have *op'*: ¬ *open s'* **using** *step op uid vis* **unfolding** *open-def a* **by** (*auto simp*: *c-defs*)
have *φ*: ¬ *φ ?trn* **using** *op op' a* **unfolding** *φ-def2*[*OF step*] **by** *auto*
hence *vl'*: *vl'* = *vl* **using** *c* **unfolding** *consume-def* **by** *simp*
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** *φ* **unfolding** *consume-def ss1* **by** *auto*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
show *?Δ s' vl' s' vl1* **using** *BC* **proof**(*cases rule*: *BC.cases*)
case (*BC-PVal*)
hence *Δ1 s' vl' s' vl1* **using** *PID' op' cor1* **unfolding** *Δ1-def vl' pPID'* **by** *auto*

```
      thus ?thesis by simp
    next
      case (BC-BO vll vll1 ul ul1 sul)
      show ?thesis
      proof(cases ul ≠ [] ∧ ul1 ≠ [])
        case True
        hence Δ31 s′ vl′ s′ vl1 using BC-BO PID′ op′ cor1
        unfolding Δ31-def vl′ pPID′ apply auto
        apply (rule exI[of - ul]) apply (rule exI[of - ul1])
        apply (rule exI[of - sul])
        apply (rule exI[of - vll]) apply (rule exI[of - vll1])
        by auto
        thus ?thesis by simp
      next
        case False
        hence 0: ul = [] ∧ ul1 = [] using BC-BO by simp
        hence 1: list-all isPValS ul ∧ list-all isPValS ul1
        using ‹list-all (Not ∘ isOVal) ul› ‹list-all (Not ∘ isOVal) ul1›
        using filter-list-all-isPValS-isOVal by auto

        have Δ32 s′ vl′ s′ vl1 using BC-BO PID′ op′ cor1 0 1
        unfolding Δ32-def vl′ pPID′ apply simp
        apply(rule exI[of - sul])
        apply(rule exI[of - vll]) apply(rule exI[of - vll1])
        by auto
        thus ?thesis by simp
      qed
    qed
  qed
 qed
next
 case False note a = False
 have op′: ¬ open s′
  using a step PID op unfolding open-def
  apply(cases a)
  subgoal for x1 apply(cases x1) apply(fastforce simp: s-defs)+ .
  subgoal for x2 apply(cases x2) apply(fastforce simp: c-defs)+ .
  subgoal for x3 apply(cases x3) apply(fastforce simp: d-defs)+ .
  subgoal for x4 apply(cases x4) apply(fastforce simp: u-defs)+ .
  subgoal by auto
  subgoal by auto
  subgoal for x7 apply(cases x7) apply(fastforce simp: com-defs)+ .
  done
 have φ: ¬ φ ?trn using PID step op op′ unfolding φ-def2[OF step]
 by (auto simp: u-defs com-defs)
 hence vl′: vl′ = vl using c unfolding consume-def by simp
 have PID′: ¬ PID ∈∈ postIDs s′
  using step PID a
  apply(cases a)
```

subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** φ **unfolding** *consume-def ss1* **by** *auto*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
have Δ*0 s′ vl′ s′ vl1* **using** *a BC PID′ cor1* **unfolding** Δ*0-def vl′* **by** *simp*
thus *?*Δ *s′ vl′ s′ vl1* **by** *simp*
qed
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vlvl1* **by** *simp*
qed
qed

lemma *unwind-cont-*Δ*1*: *unwind-cont* Δ*1* {Δ*1*,Δ*11*}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. Δ*1 s vl s1 vl1* ∨ Δ*11 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** Δ*1 s vl s1 vl1*
then obtain
lvl: *list-all* (*Not* ∘ *isOVal*) *vl* **and** *lvl1*: *list-all* (*Not* ∘ *isOVal*) *vl1*
and *map*: *map tgtAPI* (*filter isPValS vl*) = *map tgtAPI* (*filter isPValS vl1*)
and *rs*: *reach s* **and** *ss1*: *eqButPID s s1* **and** *op*: ¬ *open s* **and** *PID*: *PID* ∈∈ *postIDs s*
and *vlvl1*: *vl* = [] ⟹ *vl1* = [] **and** *cor1*: *corrFrom* (*post s1 PID*) *vl1*
using *reachNT-reach* **unfolding** Δ*1-def* **by** *auto*
have *PID1*: *PID* ∈∈ *postIDs s1* **using** *eqButPID-stateSelectors*[*OF ss1*] *PID* **by** *auto*
have *own*: *owner s PID* ∈ *set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
hence *own1*: *owner s1 PID* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
have *adm*: *admin s* ∈ *set* (*userIDs s*) **using** *reach-admin-userIDs*[*OF rs own*] **.**

hence *adm1*: *admin s1* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
have *op1*: ¬ *open s1* **using** *op ss1 eqButPID-open* **by** *auto*
show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof(*cases vl1*)
case (*Cons v1 vll1*) **note** *vl1* = *Cons*
show *?thesis* **proof**(*cases v1*)
case (*PVal pst1*) **note** *v1* = *PVal*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
define *a1* **where** *a1*: *a1* ≡ *Uact* (*uPost uid p PID pst1*)
have *uid1*: *uid* = *owner s1 PID* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid p*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a1* = (*ou1*, *s1'*) **by**(*cases step s1 a1*) *auto*
have *ou1*: *ou1* = *outOK* **using** *step1 PID1 own1* **unfolding** *a1 uid1 p1* **by** (*auto simp*: *u-defs*)
have *op1'*: ¬ *open s1'* **using** *step1 op1* **unfolding** *a1 ou1 open-def* **by** (*auto simp*: *u-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid* **using** *op PID own* **unfolding** *open-def* **by** *auto*
have *pPID1'*: *post s1' PID* = *pst1* **using** *step1* **unfolding** *a1 ou1* **by** (*auto simp*: *u-defs*)
let *?trn1* = *Trans s1 a1 ou1 s1'*
have *?iact* **proof**
show *step s1 a1* = (*ou1*, *s1'*) **using** *step1* **.**
next
show *φ*: *φ ?trn1* **unfolding** *φ-def2*[*OF step1*] *a1 ou1* **by** *simp*
show *consume ?trn1 vl1 vll1*
using *φ* **unfolding** *vl1 consume-def v1 a1* **by** *auto*
next
show ¬ *γ ?trn1* **using** *uid* **unfolding** *a1* **by** *auto*
next
have *eqButPID s1 s1'* **using** *Uact-uPaperC-step-eqButPID*[*OF - step1*] *a1* **by** *auto*
hence *ss1'*: *eqButPID s s1'* **using** *eqButPID-trans ss1* **by** *blast*
show *?Δ s vl s1' vll1* **using** *PID op ss1' lvl lvl1 map vlvl1 cor1*
unfolding *Δ1-def vl1 v1 pPID1'* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case (*PValS aid1 pst1*) **note** *v1* = *PValS*
have *pPID1*: *post s1 PID* = *pst1* **using** *cor1* **unfolding** *vl1 v1* **by** *auto*
then obtain *v vll* **where** *vl*: *vl* = *v* # *vll*
using *map* **unfolding** *vl1 v1* **by** (*cases vl*) *auto*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'*

assume *step*: *step s a* = (*ou*, *s'*) **and** *c*: *consume ?trn vl vl'*
have *PID'*: *PID* ∈∈ *postIDs s'* **using** *reach-postIDs-persist*[*OF PID step*] **.**
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'*
(**is** *?match* ∨ *?ignore*)
proof(*cases φ ?trn*)
case *True* **note** *φ* = *True*
then obtain *f*: *f ?trn* = *v* **and** *vl'*: *vl'* = *vll*
using *c* **unfolding** *vl consume-def φ-def2* **by** *auto*
show *?thesis*
proof(*cases v*)
case (*PVal pst*) **note** *v* = *PVal*
have *vll*: *vll* ≠ [] **using** *map* **unfolding** *vl1 v1 vl v* **by** *auto*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
have *a*: *a* = *Uact* (*uPost uid p PID pst*)
using *f-eq-PVal*[*OF step φ f*[*unfolded v*]] **unfolding** *uid p* **.**
have *eqButPID s s'* **using** *Uact-uPaperC-step-eqButPID*[*OF a step*] **by** *auto*
hence *s's1*: *eqButPID s' s1* **using** *eqButPID-sym eqButPID-trans ss1* **by** *blast*
have *op'*: ¬ *open s'* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*
have *?ignore* **proof**
show *γ*: ¬ *γ ?trn* **using** *step-open-φ-f-PVal-γ*[*OF rs step PID op φ f*[*unfolded v*]] **.**
show *?Δ s' vl' s1 vl1*
using *lvl1 lvl PID' map s's1 op' vll cor1* **unfolding** *Δ1-def vl1 vl vl' v*
by *auto*
qed
thus *?thesis* **by** *simp*
next
case (*PValS aid pst*) **note** *v* = *PValS*
define *uid* **where** *uid*: *uid* ≡ *admin s* **define** *p* **where** *p*: *p* ≡ *pass s uid*
have *a*: *a* = *COMact* (*comSendPost* (*admin s*) *p aid PID*)
using *f-eq-PValS*[*OF step φ f*[*unfolded v*]] **unfolding** *uid p* **.**
have *op'*: ¬ *open s'* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*
have *aid1*: *aid1* = *aid* **using** *map* **unfolding** *vl1 v1 vl v* **by** *simp*
have *uid1*: *uid* = *admin s1* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid p*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
have *pPID1'*: *post s1' PID* = *pst1* **using** *pPID1 step1* **unfolding** *a*
by (*auto simp*: *com-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid* **using** *op PID adm* **unfolding** *open-def* **by** *auto*

have *op1'*: ¬ *open s1'* **using** *step1 op1* **unfolding** *a open-def*
by (*auto simp*: *u-defs com-defs*)
let *?trn1* = *Trans s1 a ou1 s1'*
have *φ1*: *φ ?trn1* **using** *eqButPID-step-φ-imp*[*OF ss1 step step1 φ*] **.**
have *ou1*: *ou1* =
O-sendPost (*aid*, *clientPass s1 aid*, *PID*, *post s1 PID*, *owner s1 PID*, *vis s1 PID*)
using *φ1 step1 adm1 PID1* **unfolding** *a* **by** (*cases ou1*, *auto simp*: *com-defs*)
have *f1*: *f ?trn1* = *v1* **using** *φ1* **unfolding** *φ-def2*[*OF step1*] *v1 a ou1 aid1 pPID1* **by** *auto*
have *s's1'*: *eqButPID s' s1'* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vll1* **using** *φ1* **unfolding** *consume-def vl1 f1* **by** *simp*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **note** *γ* = *this*
have *ou*: (∃ *uid p aid pid*.
a = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨
ou = *ou1*
using *eqButPID-step-γ-out*[*OF ss1 step step1 op rsT rs1 γ*] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
show *?Δ s' vl' s1' vll1*
proof(*cases vll* = [])
case *True* **note** *vll* = *True*
hence *filter isPValS vll1* = [] **using** *map lvl lvl1* **unfolding** *vl vl1 v v1* **by** *simp*
hence *lvl1*: *list-all isPVal vll1*
using *filter-list-all-isPVal-isOVal lvl1* **unfolding** *vl1 v1* **by** *auto*
hence *Δ11 s' vl' s1' vll1* **using** *s's1' op1' op' PID' lvl lvl1 map cor1 pPID1 pPID1'*
unfolding *Δ11-def vl vl' vl1 v v1 vll* **by** *auto*
thus *?thesis* **by** *auto*
next
case *False* **note** *vll* = *False*
hence *Δ1 s' vl' s1' vll1* **using** *s's1' op1' op' PID' lvl lvl1 map cor1 pPID1 pPID1'*
unfolding *Δ1-def vl vl' vl1 v v1* **by** *auto*
thus *?thesis* **by** *auto*
qed
qed
thus *?thesis* **using** *vl* **by** *simp*
qed(*insert lvl vl*, *auto*)

next
case *False* **note** φ = *False*
hence *vl′*: *vl′* = *vl* **using** *c* **unfolding** *consume-def* **by** *auto*
obtain *ou1 s1′* **where** *step1*: *step s1 a* = (*ou1*, *s1′*) **by**(*cases step s1 a*) *auto*
have *s′s1′*: *eqButPID s′ s1′* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
let *?trn1* = *Trans s1 a ou1 s1′*
have φ*1*: ¬ φ *?trn1* **using** φ *ss1* **by** (*simp add*: *eqButPID-step-*φ *step step1*)
have *pPID1′*: *post s1′ PID* = *pst1*
using *PID1 pPID1 step1* φ*1*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op′*: ¬ *open s′*
using *PID step* φ *op*
unfolding φ*-def2*[*OF step1*]
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x4* **using** φ*-def2* φ *step* **by** *blast*
subgoal by *auto*
subgoal by *auto*
subgoal using φ*-def2* φ *step* **by** *blast*
done
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** φ*1* **unfolding** *consume-def* **by** *simp*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **note** γ = *this*
have *ou*: (∃ *uid p aid pid*.
a = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨
ou = *ou1*
using *eqButPID-step-*γ*-out*[*OF ss1 step step1 op rsT rs1* γ] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
have Δ*1 s′ vl′ s1′ vl1* **using** *s′s1′ PID′ pPID1 pPID1′ lvl lvl1 map cor1 op′*
unfolding Δ*1-def vl vl′* **by** *auto*

thus *?Δ s′ vl′ s1′ vl1* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vlvl1* **by** *simp*
qed(*insert lvl1 vl1*, *auto*)
next
case *Nil* **note** *vl1 = Nil*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn = Trans s a ou s′*
assume *step*: *step s a = (ou, s′)* **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl′*
have *PID′*: *PID* ∈∈ *postIDs s′* **using** *reach-postIDs-persist*[*OF PID step*] **.**
obtain *ou1 s1′* **where** *step1*: *step s1 a = (ou1, s1′)* **by**(*cases step s1 a*) *auto*
let *?trn1 = Trans s1 a ou1 s1′*
show *match ?Δ s s1 vl1 a ou s′ vl′* ∨ *ignore ?Δ s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
proof(*cases* ∃ *uid p pstt. a = Uact (uPost uid p PID pstt)* ∧ *ou = outOK*)
case *True* **then obtain** *uid p pstt* **where**
a: *a = Uact (uPost uid p PID pstt)* **and** *ou*: *ou = outOK* **by** *auto*
hence *φ*: *φ ?trn* **unfolding** *φ-def2*[*OF step*] **by** *auto*
then obtain *v* **where** *vl*: *vl = v # vl′* **and** *f*: *f ?trn = v*
using *c* **unfolding** *consume-def φ-def2* **by** (*cases vl*) *auto*
obtain *pst* **where** *v*: *v = PVal pst* **using** *map lvl* **unfolding** *vl vl1* **by** (*cases v*) *auto*
have *pstt*: *pstt = pst* **using** *f* **unfolding** *a v* **by** *auto*
have *uid*: *uid* ∉ *UIDs* **using** *step op PID* **unfolding** *a ou open-def* **by** (*auto simp*: *u-defs*)
have *eqButPID s s′* **using** *Uact-uPaperC-step-eqButPID*[*OF a step*] **by** *auto*
hence *s′s1*: *eqButPID s′ s1* **using** *eqButPID-sym eqButPID-trans ss1* **by** *blast*
have *op′*: ¬ *open s′* **using** *step PID′ op* **unfolding** *a ou open-def* **by** (*auto simp*: *u-defs*)
have *?ignore* **proof**
show ¬ *γ ?trn* **unfolding** *a* **using** *uid* **by** *auto*
next
show *?Δ s′ vl′ s1 vl1* **using** *PID′ s′s1 op′ lvl map*
unfolding *Δ1-def vl1 vl* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case *False* **note** *a = False*
{**assume** *φ*: *φ ?trn*
then obtain *v vl′* **where** *vl*: *vl = v # vl′* **and** *f*: *f ?trn = v*
using *c* **unfolding** *consume-def* **by** (*cases vl*) *auto*
obtain *pst* **where** *v*: *v = PVal pst* **using** *map lvl* **unfolding** *vl vl1* **by** (*cases v*) *auto*
have *False* **using** *f f-eq-PVal*[*OF step φ*, *of pst*] *a φ v* **by** *auto*

}
hence φ: ¬ φ *?trn* **by** *auto*
have $\varphi 1$: ¬ φ *?trn1* **by** (*metis* φ *eqButPID-step-*φ *step ss1 step1*)
have *op'*: ¬ *open s'* **using** *a op* φ **unfolding** φ*-def2*[*OF step*] **by** *auto*
have *vl'*: *vl' = vl* **using** *c* φ **unfolding** *consume-def* **by** *auto*
have *s's1'*: *eqButPID s' s1'* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
have *op1'*: ¬ *open s1'* **using** *op' eqButPID-open*[*OF s's1'*] **by** *simp*
have $\bigwedge$ *uid p pst. e-updatePost s1 uid p PID pst* $\longleftrightarrow$ *e-updatePost s uid p PID pst*
using *eqButPID-stateSelectors*[*OF ss1*] **unfolding** *u-defs* **by** *auto*
hence *ou1*: $\bigwedge$ *uid p pst. a = Uact* (*uPost uid p PID pst*) $\Longrightarrow$ *ou1 = ou*
using *step step1* **by** *auto*
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** $\varphi 1$ **unfolding** *consume-def* **by** *simp*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **note** γ = *this*
have *ou*: (∃ *uid p aid pid.*
a = COMact (*comSendPost uid p aid pid*) ∧ *outPurge ou = outPurge ou1*) ∨
ou = ou1
using *eqButPID-step-*γ*-out*[*OF ss1 step step1 op rsT rs1* γ] **.**
thus *g ?trn = g ?trn1* **by** (*cases a*) *auto*
next
show *?*Δ *s' vl' s1' vl1* **using** *s's1' op' PID' lvl map*
unfolding Δ*1-def vl' vl1* **by** *auto*
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vlvl1* **by** *simp*
qed
qed

lemma *unwind-cont-*Δ*11*: *unwind-cont* Δ*11* {Δ*11*}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. Δ*11 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** Δ*11 s vl s1 vl1*
hence *vl*: *vl* = [] **and** *lvl1*: *list-all isPVal vl1*
and *rs*: *reach s* **and** *ss1*: *eqButPID s s1* **and** *op*: ¬ *open s* **and** *PID*: *PID* ∈∈ *postIDs s*
and *cor1*: *corrFrom* (*post s1 PID*) *vl1*
using *reachNT-reach* **unfolding** Δ*11-def* **by** *auto*
have *PID1*: *PID* ∈∈ *postIDs s1* **using** *eqButPID-stateSelectors*[*OF ss1*] *PID* **by** *auto*

have *own*: *owner s PID* ∈ *set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
hence *own1*: *owner s1 PID* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
have *adm*: *admin s* ∈ *set* (*userIDs s*) **using** *reach-admin-userIDs*[*OF rs own*] **.**
hence *adm1*: *admin s1* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
have *op1*: ¬ *open s1* **using** *op ss1 eqButPID-open* **by** *auto*
show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof(*cases vl1*)
case (*Cons v1 vll1*) **note** *vl1* = *Cons*
then obtain *pst1* **where** *v1*: *v1* = *PVal pst1* **using** *lvl1* **unfolding** *vl1* **by** (*cases v1*) *auto*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
define *a1* **where** *a1*: *a1* ≡ *Uact* (*uPost uid p PID pst1*)
have *uid1*: *uid* = *owner s1 PID* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid p*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1′* **where** *step1*: *step s1 a1* = (*ou1*, *s1′*) **by**(*cases step s1 a1*) *auto*
have *ou1*: *ou1* = *outOK* **using** *step1 PID1 own1* **unfolding** *a1 uid1 p1* **by** (*auto simp*: *u-defs*)
have *op1′*: ¬ *open s1′* **using** *step1 op1* **unfolding** *a1 ou1 open-def* **by** (*auto simp*: *u-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid* **using** *op PID own* **unfolding** *open-def* **by** *auto*
have *pPID1′*: *post s1′ PID* = *pst1* **using** *step1* **unfolding** *a1 ou1* **by** (*auto simp*: *u-defs*)
let *?trn1* = *Trans s1 a1 ou1 s1′*
have *?iact* **proof**
show *step s1 a1* = (*ou1*, *s1′*) **using** *step1* **.**
next
show *φ*: *φ ?trn1* **unfolding** *φ-def2*[*OF step1*] *a1 ou1* **by** *simp*
show *consume ?trn1 vl1 vll1*
using *φ* **unfolding** *vl1 consume-def v1 a1* **by** *auto*
next
show ¬ *γ ?trn1* **using** *uid* **unfolding** *a1* **by** *auto*
next
have *eqButPID s1 s1′* **using** *Uact-uPaperC-step-eqButPID*[*OF - step1*] *a1* **by** *auto*
hence *ss1′*: *eqButPID s s1′* **using** *eqButPID-trans ss1* **by** *blast*
show *?Δ s vl s1′ vll1*
using *PID op ss1′ lvl1 cor1* **unfolding** *Δ11-def vl1 v1 vl pPID1′* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case *Nil* **note** *vl1* = *Nil*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*

let *?trn* = *Trans s a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *c*: *consume ?trn vl vl'*
have *PID'*: *PID* ∈∈ *postIDs s'* **using** *reach-postIDs-persist*[*OF PID step*] **.**
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
have *φ*: ¬ *φ ?trn* **using** *c* **unfolding** *consume-def vl* **by** *auto*
show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'*
(**is** *?match* ∨ *?ignore*)
proof−
have *vl'*: *vl'* = *vl* **using** *c* **unfolding** *vl consume-def* **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*)
auto
have *s's1'*: *eqButPID s' s1'* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
let *?trn1* = *Trans s1 a ou1 s1'*
have *φ1*: ¬ *φ ?trn1* **using** *φ ss1* **by** (*simp add*: *eqButPID-step-φ step step1*)
have *pPID1'*: *post s1' PID* = *post s1 PID* **using** *PID1 step1 φ1*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op'*: ¬ *open s'*
using *PID step φ op* **unfolding** *φ-def2*[*OF step*]
by *auto*
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** *φ1* **unfolding** *consume-def* **by** *simp*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **note** *γ* = *this*
have *ou*: (∃ *uid p aid pid*.
a = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge*
ou1) ∨
ou = *ou1*
using *eqButPID-step-γ-out*[*OF ss1 step step1 op rsT rs1 γ*] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
have *?Δ s' vl' s1' vl1* **using** *s's1' PID' pPID1' lvl1 cor1 op'*
unfolding *Δ11-def vl vl'* **by** *auto*
thus *?Δ s' vl' s1' vl1* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed

qed
thus *?thesis* **using** *vl1* **by** *simp*
qed
qed

lemma *unwind-cont-Δ31*: *unwind-cont Δ31* {*Δ31*,*Δ32*}
proof(*rule*, *simp*)
let *?Δ* = *λs vl s1 vl1*. *Δ31 s vl s1 vl1* ∨ *Δ32 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *Δ31 s vl s1 vl1*
then obtain *ul ul1 sul vll vll1* **where**
lul: *list-all* (*Not* ∘ *isOVal*) *ul* **and** *lul1*: *list-all* (*Not* ∘ *isOVal*) *ul1*
and *map*: *map tgtAPI* (*filter isPValS ul*) = *map tgtAPI* (*filter isPValS ul1*)
and *rs*: *reach s* **and** *ss1*: *eqButPID s s1* **and** *op*: ¬ *open s* **and** *PID*: *PID* ∈∈ *postIDs s*
and *cor1*: *corrFrom* (*post s1 PID*) *vl1*
and *ful*: *ul* ≠ [] **and** *ful1*: *ul1* ≠ []
and *lastul*: *isPVal* (*last ul*) **and** *ulul1*: *last ul* = *last ul1*
and *lsul*: *list-all isPValS sul*
and *vl*: *vl* = *ul* @ *sul* @ *OVal True* # *vll*
and *vl1*: *vl1* = *ul1* @ *sul* @ *OVal True* # *vll1*
and *BO*: *BO vll vll1*
using *reachNT-reach* **unfolding** *Δ31-def* **by** *auto*
have *ulNE*: *ul* ≠ [] **and** *ul1NE*: *ul1* ≠ [] **using** *ful ful1* **by** *auto*
have *PID1*: *PID* ∈∈ *postIDs s1* **using** *eqButPID-stateSelectors*[*OF ss1*] *PID* **by** *auto*
have *own*: *owner s PID* ∈ *set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
hence *own1*: *owner s1 PID* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
have *adm*: *admin s* ∈ *set* (*userIDs s*) **using** *reach-admin-userIDs*[*OF rs own*] **.**
hence *adm1*: *admin s1* ∈ *set* (*userIDs s1*) **using** *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
have *op1*: ¬ *open s1* **using** *op ss1 eqButPID-open* **by** *auto*
obtain *v1 ull1* **where** *ul1*: *ul1* = *v1* # *ull1* **using** *ful1* **by** (*cases ul1*) *auto*
show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof(*cases v1*)
case (*PVal pst1*) **note** *v1* = *PVal*
show *?thesis* **proof**(*cases list-ex isPVal ull1*)
case *True* **note** *lull1* = *True*
hence *full1*: *filter isPVal ull1* ≠ [] **by** (*induct ull1*) *auto*
hence *ull1NE*: *ull1* ≠ [] **by** *auto*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
define *a1* **where** *a1*: *a1* ≡ *Uact* (*uPost uid p PID pst1*)
have *uid1*: *uid* = *owner s1 PID* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid p*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1′* **where** *step1*: *step s1 a1* = (*ou1*, *s1′*) **by**(*cases step s1 a1*)

auto
have *ou1*: *ou1* = *outOK* **using** *step1 PID1 own1* **unfolding** *a1 uid1 p1* **by** (*auto simp*: *u-defs*)
have *op1'*: ¬ *open s1'* **using** *step1 op1* **unfolding** *a1 ou1 open-def* **by** (*auto simp*: *u-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid* **using** *op PID own* **unfolding** *open-def* **by** *auto*
have *pPID1'*: *post s1' PID* = *pst1* **using** *step1* **unfolding** *a1 ou1* **by** (*auto simp*: *u-defs*)
let *?trn1* = *Trans s1 a1 ou1 s1'*
let *?vl1'* = *ull1* @ *sul* @ *OVal True* # *vll1*
have *?iact* **proof**
show *step s1 a1* = (*ou1*, *s1'*) **using** *step1* **.**
next
show *φ*: *φ ?trn1* **unfolding** *φ-def2*[*OF step1*] *a1 ou1* **by** *simp*
show *consume ?trn1 vl1 ?vl1'*
using *φ* **unfolding** *vl1 ul1 consume-def v1 a1* **by** *simp*
next
show ¬ *γ ?trn1* **using** *uid* **unfolding** *a1* **by** *auto*
next
have *eqButPID s1 s1'* **using** *Uact-uPaperC-step-eqButPID*[*OF - step1*] *a1* **by** *auto*
hence *ss1'*: *eqButPID s s1'* **using** *eqButPID-trans ss1* **by** *blast*
have *Δ31 s vl s1' ?vl1'*
using *PID op ss1' lul lul1 map ulul1 cor1 BO ull1NE ful ful1 full1 lastul ulul1 lsul*
unfolding *Δ31-def vl vl1 ul1 v1 pPID1'* **apply** *auto*
apply(*rule exI*[*of - ul*]) **apply**(*rule exI*[*of - ull1*]) **apply**(*rule exI*[*of - sul*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*
thus *?Δ s vl s1' ?vl1'* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case *False* **note** *lull1* = *False*
hence *ull1*: *ull1* = [] **using** *lastul* **unfolding** *ulul1 ul1 v1* **by** *simp*(*meson Bex-set last-in-set*)
hence *ul1*: *ul1* = [*PVal pst1*] **unfolding** *ul1 v1* **by** *simp*
obtain *ulll* **where** *ul-ulll*: *ul* = *ulll* ## *PVal pst1* **using** *lastul ulul1 ulNE* **unfolding** *ul1 ull1 v1*
by (*cases ul rule*: *rev-cases*) *auto*
hence *ulNE*: *ul* ≠ [] **by** *simp*

have *filter isPValS ulll* = [] **using** *map* **unfolding** *ul-ulll ul1 v1 ull1* **by** *simp*
hence *lull*: *list-all isPVal ulll* **using** *lul filter-list-all-isPVal-isOVal*
unfolding *ul-ulll* **by** *auto*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *c*: *consume ?trn vl vl'*

have *PID'*: *PID* ∈∈ *postIDs s'* **using** *reach-postIDs-persist*[*OF PID step*] **.**
obtain *ul'* **where** *cc*: *consume ?trn ul ul'* **and**
vl': *vl' = ul' @ sul @ OVal True # vll* **using** *c ulNE* **unfolding** *consume-def vl*
by (*cases φ ?trn*) *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'*
(**is** *?match* ∨ *?ignore*)
proof(*cases ulll*)
case *Nil*
hence *ul*: *ul* = [*PVal pst1*] **unfolding** *ul-ulll* **by** *simp*
have *?match* **proof**(*cases φ ?trn*)
case *True* **note** *φ* = *True*
then obtain *f*: *f ?trn* = *PVal pst1* **and** *ul'*: *ul'* = []
using *cc* **unfolding** *ul consume-def φ-def2* **by** *auto*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
have *a*: *a* = *Uact* (*uPost uid p PID pst1*)
using *f-eq-PVal*[*OF step φ f*] **unfolding** *uid p* **.**
have *uid1*: *uid* = *owner s1 PID* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid p*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by**(*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
have *φ1*: *φ ?trn1* **using** *eqButPID-step-φ-imp*[*OF ss1 step step1 φ*] **.**
have *ou1*: *ou1* = *outOK*
using *φ1 step1 PID1* **unfolding** *a* **by** (*cases ou1*, *auto simp*: *com-defs*)
have *pPID'*: *post s' PID* = *pst1* **using** *step φ* **unfolding** *a* **by** (*auto simp*: *u-defs*)
have *pPID1'*: *post s1' PID* = *pst1* **using** *step1 φ1* **unfolding** *a* **by** (*auto simp*: *u-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid* **using** *op PID own* **unfolding** *open-def* **by** *auto*
have *op1'*: ¬ *open s1'* **using** *step1 op1* **unfolding** *a open-def*
by (*auto simp*: *u-defs com-defs*)
have *f1*: *f ?trn1* = *PVal pst1* **using** *φ1* **unfolding** *φ-def2*[*OF step1*] *v1 a ou1* **by** *auto*
have *s's1'*: *eqButPID s' s1'* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
have *op'*: ¬ *open s'* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*
have *ou*: *ou* = *outOK* **using** *φ op op'* **unfolding** *φ-def2*[*OF step*] *a* **by** *auto*
let *?vl'* = *sul @ OVal True # vll*
let *?vl1'* = *sul @ OVal True # vll1*
show *?thesis* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*

next
show *consume ?trn1 vl1 ?vl1'*
using *φ1* **unfolding** *consume-def ul1 f1 vl1* **by** *simp*
next
show *γ ?trn = γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **note** *γ = this*
thus *g ?trn = g ?trn1* **using** *ou ou1* **by** (*cases a*) *auto*
next
have *s'*: *s' = s1'* **using** *eqButPID-step-eq*[*OF ss1 a ou step step1*] **.**
have *corr1*: *corrFrom* (*post s1' PID*) *?vl1'*
using *cor1* **unfolding** *vl1 ul1 v1 pPID1'* **by** *auto*
have *Δ32 s' vl' s1' ?vl1'*
using *PID' op1 op' s's1' lul lul1 map ulul1 cor1 BO ful ful1 lastul ulul1 lsul corr1*
unfolding *Δ32-def vl vl1 v1 vl' ul' ul ul1 s'* **apply** *simp*
apply(*rule exI*[*of - sul*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*
thus *?Δ s' vl' s1' ?vl1'* **by** *simp*
qed
next
case *False* **note** *φ = False*
hence *ul'*: *ul' = ul* **using** *cc* **unfolding** *consume-def* **by** *auto*
obtain *ou1 s1'* **where** *step1*: *step s1 a = (ou1, s1')* **by**(*cases step s1 a*) *auto*
have *s's1'*: *eqButPID s' s1'* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
let *?trn1 = Trans s1 a ou1 s1'*
have *φ1*: ¬ *φ ?trn1* **using** *φ ss1* **by** (*simp add*: *eqButPID-step-φ step step1*)
have *pPID1'*: *post s1' PID = post s1 PID* **using** *PID1 step1 φ1*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op'*: ¬ *open s'*
using *PID step φ op* **unfolding** *φ-def2*[*OF step*] **by** *auto*
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** *φ1* **unfolding** *consume-def* **by** *simp*
next
show *γ ?trn = γ ?trn1* **unfolding** *ss1* **by** *simp*
next

assume γ *?trn* **note** γ = *this*
have *ou*: ($\exists$ *uid p aid pid*.
a = *COMact* (*comSendPost uid p aid pid*) $\wedge$ *outPurge ou* = *outPurge ou1*) $\vee$
ou = *ou1*
using *eqButPID-step-γ-out*[*OF ss1 step step1 op rsT rs1* γ] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
have Δ*31 s′ vl′ s1′ vl1*
using *PID′ pPID1′ op′ s′s1′ lul lul1 map ulul1 cor1*
BO ful ful1 lastul ulul1 lsul cor1
unfolding Δ*31-def vl vl1 v1 vl′ ul′* **apply** *simp*
apply(*rule exI*[*of - ul*]) **apply**(*rule exI*[*of - ul1*]) **apply**(*rule exI*[*of - sul*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*
thus *?*Δ *s′ vl′ s1′ vl1* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
thus *?thesis* **by** *simp*
next
case (*Cons v ullll*) **note** *ulll* = *Cons*
then obtain *pst* **where** *v*: *v* = *PVal pst* **using** *lull ul-ulll ulll lul* **by** (*cases v*) *auto*
define *ull* **where** *ull*: *ull* ≡ *ullll* ## *PVal pst1*
have *ul*: *ul* = *v* # *ull* **unfolding** *ul-ulll ull ulll* **by** *simp*
show *?thesis* **proof**(*cases* φ *?trn*)
case *True* **note** φ = *True*
then obtain *f*: *f ?trn* = *v* **and** *ul′*: *ul′* = *ull*
using *cc* **unfolding** *ul consume-def* φ*-def2* **by** *auto*
define *uid* **where** *uid*: *uid* ≡ *owner s PID* **define** *p* **where** *p*: *p* ≡ *pass s uid*
have *a*: *a* = *Uact* (*uPost uid p PID pst*)
using *f-eq-PVal*[*OF step* φ *f*[*unfolded v*]] **unfolding** *uid p* **.**
have *eqButPID s s′* **using** *Uact-uPaperC-step-eqButPID*[*OF a step*] **by** *auto*
hence *s′s1*: *eqButPID s′ s1* **using** *eqButPID-sym eqButPID-trans ss1* **by** *blast*
have *op′*: ¬ *open s′* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*
have *?ignore* **proof**
show γ: ¬ γ *?trn* **using** *step-open-φ-f-PVal-γ*[*OF rs step PID op* φ *f*[*unfolded v*]] **.**
have Δ*31 s′ vl′ s1 vl1*
using *PID′ op′ s′s1 lul lul1 map ulul1 cor1 BO ful ful1 lastul ulul1 lsul ull*
unfolding Δ*31-def vl vl1 v1 vl′ ul′ ul v* **apply** *simp*
apply(*rule exI*[*of - ull*]) **apply**(*rule exI*[*of - ul1*]) **apply**(*rule exI*[*of - sul*])

apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*
thus *?*Δ *s′ vl′ s1 vl1* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case *False* **note** φ = *False*
hence *ul′*: *ul′* = *ul* **using** *cc* **unfolding** *consume-def* **by** *auto*
obtain *ou1 s1′* **where** *step1*: *step s1 a* = (*ou1*, *s1′*) **by**(*cases step s1 a*) *auto*
have *s′s1′*: *eqButPID s′ s1′* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
let *?trn1* = *Trans s1 a ou1 s1′*
have φ*1*: ¬ φ *?trn1* **using** φ *ss1* **by** (*simp add*: *eqButPID-step-*φ *step step1*)
have *pPID1′*: *post s1′ PID* = *post s1 PID* **using** *PID1 step1* φ*1*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op′*: ¬ *open s′*
using *PID step* φ *op* **unfolding** φ*-def2*[*OF step*] **by** *auto*
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** φ*1* **unfolding** *consume-def* **by** *simp*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **note** γ = *this*
have *ou*: (∃ *uid p aid pid*.
a = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨
ou = *ou1*
using *eqButPID-step-*γ*-out*[*OF ss1 step step1 op rsT rs1* γ] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
have Δ*31 s′ vl′ s1′ vl1*
using *PID′ pPID1′ op′ s′s1′ lul lul1 map ulul1 cor1*
BO ful ful1 lastul ulul1 lsul cor1
unfolding Δ*31-def vl vl1 v1 vl′ ul′* **apply** *simp*
apply(*rule exI*[*of - ul*]) **apply**(*rule exI*[*of - ul1*]) **apply**(*rule exI*[*of - sul*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*
thus *?*Δ *s′ vl′ s1′ vl1* **by** *simp*

qed
thus *?thesis* **by** *simp*
qed
qed
qed
thus *?thesis* **using** *vl* **by** *simp*
qed
next
case (*PValS aid1 pst1*) **note** $v1 = PValS$
have *pPID1*: *post s1 PID = pst1* **using** *cor1* **unfolding** *vl1 ul1 v1* **by** *auto*
then obtain *v ull* **where** *ul*: *ul = v # ull*
using *map* **unfolding** *ul1 v1* **by** (*cases ul*) *auto*
let *?vl1′ = ull1 @ sul @ OVal True # vll1*
have *?react* **proof**
fix *a :: act* **and** *ou :: out* **and** *s′ :: state* **and** *vl′*
let *?trn = Trans s a ou s′*
assume *step*: *step s a = (ou, s′)* **and** *c*: *consume ?trn vl vl′*
have *PID′*: *PID ∈∈ postIDs s′* **using** *reach-postIDs-persist*[*OF PID step*] **.**
obtain *ul′* **where** *cc*: *consume ?trn ul ul′* **and**
vl′: *vl′ = ul′ @ sul @ OVal True # vll* **using** *c ul* **unfolding** *consume-def vl*
by (*cases φ ?trn*) *auto*
obtain *ou1 s1′* **where** *step1*: *step s1 a = (ou1, s1′)* **by**(*cases step s1 a*) *auto*
let *?trn1 = Trans s1 a ou1 s1′*
show *match ?Δ s s1 vl1 a ou s′ vl′ ∨ ignore ?Δ s s1 vl1 a ou s′ vl′*
(**is** *?match ∨ ?ignore*)
proof(*cases φ ?trn*)
case *True* **note** *φ = True*
then obtain *f*: *f ?trn = v* **and** *ul′*: *ul′ = ull*
using *cc* **unfolding** *ul consume-def φ-def2* **by** *auto*
show *?thesis*
proof(*cases v*)
case (*PVal pst*) **note** *v = PVal*
have *full*: *ull ≠ []* **using** *map* **unfolding** *ul1 v1 ul v* **by** *auto*
define *uid* **where** *uid*: *uid ≡ owner s PID* **define** *p* **where** *p*: *p ≡ pass s uid*
have *a*: *a = Uact (uPost uid p PID pst)*
using *f-eq-PVal*[*OF step φ f*[*unfolded v*]] **unfolding** *uid p* **.**
have *eqButPID s s′* **using** *Uact-uPaperC-step-eqButPID*[*OF a step*] **by** *auto*
hence *s′s1*: *eqButPID s′ s1* **using** *eqButPID-sym eqButPID-trans ss1* **by** *blast*
have *op′*: *¬ open s′* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*

have *?ignore* **proof**
show *γ*: *¬ γ ?trn* **using** *step-open-φ-f-PVal-γ*[*OF rs step PID op φ f*[*unfolded v*]] **.**
have *Δ31 s′ vl′ s1 vl1*
using *PID′ op′ s′s1 lul lul1 map ulul1 cor1 BO ful ful1 lastul ulul1 lsul*

full
unfolding Δ*31-def vl vl1 v1 vl′ ul′ ul v* **apply** *simp*
apply(*rule exI*[*of - ull*]) **apply**(*rule exI*[*of - ul1*]) **apply**(*rule exI*[*of - sul*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*
thus *?*Δ *s′ vl′ s1 vl1* **by** *auto*
qed
thus *?thesis* **by** *simp*
next
case (*PValS aid pst*) **note** *v* = *PValS*
define *uid* **where** *uid*: *uid* ≡ *admin s* **define** *p* **where** *p*: *p* ≡ *pass s uid*
have *a*: *a* = *COMact* (*comSendPost* (*admin s*) *p aid PID*)
using *f-eq-PValS*[*OF step φ f*[*unfolded v*]] **unfolding** *uid p* **.**
have *op′*: ¬ *open s′* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*
have *aid1*: *aid1* = *aid* **using** *map* **unfolding** *ul1 v1 ul v* **by** *simp*
have *uid1*: *uid* = *admin s1* **and** *p1*: *p* = *pass s1 uid* **unfolding** *uid p*
using *eqButPID-stateSelectors*[*OF ss1*] **by** *auto*
obtain *ou1 s1′* **where** *step1*: *step s1 a* = (*ou1*, *s1′*) **by**(*cases step s1 a*) *auto*
have *pPID1′*: *post s1′ PID* = *pst1* **using** *pPID1 step1* **unfolding** *a*
by (*auto simp*: *com-defs*)
have *uid*: *uid* ∉ *UIDs* **unfolding** *uid* **using** *op PID adm* **unfolding** *open-def* **by** *auto*
have *op1′*: ¬ *open s1′* **using** *step1 op1* **unfolding** *a open-def*
by (*auto simp*: *u-defs com-defs*)
let *?trn1* = *Trans s1 a ou1 s1′*
have *φ1*: *φ ?trn1* **using** *eqButPID-step-φ-imp*[*OF ss1 step step1 φ*] **.**
have *ou1*: *ou1* =
O-sendPost (*aid*, *clientPass s1 aid*, *PID*, *post s1 PID*, *owner s1 PID*, *vis s1 PID*)
using *φ1 step1 adm1 PID1* **unfolding** *a* **by** (*cases ou1*, *auto simp*: *com-defs*)
have *f1*: *f ?trn1* = *v1* **using** *φ1* **unfolding** *φ-def2*[*OF step1*] *v1 a ou1 aid1 pPID1* **by** *auto*
have *s′s1′*: *eqButPID s′ s1′* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 ?vl1′* **using** *φ1* **unfolding** *consume-def ul1 f1 vl1* **by** *simp*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **note** *γ* = *this*
have *ou*: (∃ *uid p aid pid*.
a = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨
ou = *ou1*

using *eqButPID-step-*γ*-out*[*OF ss1 step step1 op rsT rs1* γ] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
have *corr1*: *corrFrom* (*post s1′ PID*) *?vl1′*
using *cor1* **unfolding** *vl1 ul1 v1 pPID1′* **by** *auto*
have *ullull1*: *ull1* ≠ [] ⟶ *ull* ≠ [] **using** *ul ul1 lastul ulul1* **unfolding** *v v1*
by *fastforce*
have Δ*31 s′ vl′ s1′ ?vl1′*
using *PID′ op′ s′s1′ lul lul1 map ulul1 cor1 BO ful ful1 lastul ulul1 lsul corr1 ullull1*
unfolding Δ*31-def vl vl1 v1 vl′ ul′ ul ul1 v* **apply** *auto*
apply(*rule exI*[*of - ull*]) **apply**(*rule exI*[*of - ull1*]) **apply**(*rule exI*[*of - sul*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*
thus *?*Δ *s′ vl′ s1′ ?vl1′* **by** *simp*
qed
thus *?thesis* **using** *ul* **by** *simp*
next
qed(*insert lul ul*, *auto*)
next
case *False* **note** φ = *False*
hence *ul′*: *ul′* = *ul* **using** *cc* **unfolding** *consume-def* **by** *auto*
obtain *ou1 s1′* **where** *step1*: *step s1 a* = (*ou1*, *s1′*) **by**(*cases step s1 a*) *auto*
have *s′s1′*: *eqButPID s′ s1′* **using** *eqButPID-step*[*OF ss1 step step1*] **.**
let *?trn1* = *Trans s1 a ou1 s1′*
have φ*1*: ¬ φ *?trn1* **using** φ *ss1* **by** (*simp add*: *eqButPID-step-*φ *step step1*)
have *pPID1′*: *post s1′ PID* = *pst1* **using** *PID1 pPID1 step1* φ*1*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op′*: ¬ *open s′*
using *PID step* φ *op* **unfolding** φ*-def2*[*OF step*] **by** *auto*
have *?match* **proof**
show *validTrans ?trn1* **using** *step1* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** φ*1* **unfolding** *consume-def* **by** *simp*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **note** γ = *this*
have *ou*: (∃ *uid p aid pid*.

a = *COMact* (*comSendPost uid p aid pid*) ∧ *outPurge ou* = *outPurge ou1*) ∨
ou = *ou1*
using *eqButPID-step-γ-out*[*OF ss1 step step1 op rsT rs1 γ*] **.**
thus *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
next
have Δ*31 s′ vl′ s1′ vl1*
using *PID′ pPID1 pPID1′ op′ s′s1′ lul lul1 map ulul1 cor1*
BO ful ful1 lastul ulul1 lsul cor1
unfolding Δ*31-def vl vl1 v1 vl′ ul′* **apply** *simp*
apply(*rule exI*[*of - ul*]) **apply**(*rule exI*[*of - ul1*]) **apply**(*rule exI*[*of - sul*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*]) **by** *auto*
thus *?*Δ *s′ vl′ s1′ vl1* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vl* **by** *simp*
qed(*insert lul1 ul1*, *auto*)
qed

lemma *unwind-cont-*Δ*32*: *unwind-cont* Δ*32* {Δ*2*,Δ*32*,Δ*4*}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. Δ*2 s vl s1 vl1* ∨ Δ*32 s vl s1 vl1* ∨ Δ*4 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** Δ*32 s vl s1 vl1*
then obtain *ul vll vll1* **where**
lul: *list-all isPValS ul*
and *rs*: *reach s* **and** *ss1*: *s1* = *s* **and** *op*: ¬ *open s* **and** *PID*: *PID* ∈∈ *postIDs s*
and *cor1*: *corrFrom* (*post s1 PID*) *vl1*
and *vl*: *vl* = *ul* @ *OVal True* # *vll*
and *vl1*: *vl1* = *ul* @ *OVal True* # *vll1*
and *BO*: *BO vll vll1*
using *reachNT-reach* **unfolding** Δ*32-def* **by** *blast*
have *own*: *owner s PID* ∈ *set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
have *adm*: *admin s* ∈ *set* (*userIDs s*) **using** *reach-admin-userIDs*[*OF rs own*] **.**
show *iaction ?*Δ *s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn* = *Trans s a ou s′* **let** *?trn1* = *Trans s1 a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *c*: *consume ?trn vl vl′*
have *PID′*: *PID* ∈∈ *postIDs s′* **using** *reach-postIDs-persist*[*OF PID step*] **.**
show *match ?*Δ *s s1 vl1 a ou s′ vl′* ∨ *ignore ?*Δ *s s1 vl1 a ou s′ vl′*
(**is** *?match* ∨ *?ignore*)
proof−
have *?match* **proof**(*cases ul* = [])

case *False* **note** *ul* = *False*
then obtain *ul′* **where** *cc*: *consume ?trn ul ul′*
and *vl′*: *vl′* = *ul′* @ *OVal True* # *vll* **using** *vl c* **unfolding** *consume-def*
by (*cases φ ?trn*) *auto*
let *?vl1′* = *ul′* @ *OVal True* # *vll1*
show *?thesis* **proof**
show *validTrans ?trn1* **using** *step* **unfolding** *ss1* **by** *simp*
next
show *consume ?trn1 vl1 ?vl1′* **using** *cc ul* **unfolding** *vl1 consume-def ss1*
by (*cases φ ?trn*) *auto*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **note** *γ* = *this*
thus *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
have *Δ32 s′ vl′ s′ ?vl1′*
proof(*cases φ ?trn*)
case *True* **note** *φ* = *True*
then obtain *v* **where** *f*: *f ?trn* = *v* **and** *ul*: *ul* = *v* # *ul′*
using *cc* **unfolding** *consume-def* **by** (*cases ul*) *auto*
define *uid* **where** *uid*: *uid* ≡ *admin s* **define** *p* **where** *p*: *p* ≡ *pass s uid*
obtain *aid pst* **where** *v*: *v* = *PValS aid pst* **using** *lul* **unfolding** *ul* **by** (*cases v*) *auto*
have *a*: *a* = *COMact* (*comSendPost* (*admin s*) *p aid PID*)
using *f-eq-PValS*[*OF step φ f*[*unfolded v*]] **unfolding** *uid p* **.**
have *op′*: ¬ *open s′* **using** *uPost-comSendPost-open-eq*[*OF step*] *a op* **by** *auto*
have *pPID′*: *post s′ PID* = *post s PID*
using *step* **unfolding** *a* **by** (*auto simp*: *com-defs*)
show *?thesis* **using** *PID′ pPID′ op′ cor1 BO lul*
unfolding *Δ32-def vl1 ul ss1 v vl′* **by** *auto*
next
case *False* **note** *φ* = *False*
hence *ul*: *ul* = *ul′* **using** *cc* **unfolding** *consume-def* **by** (*cases ul*) *auto*
have *pPID′*: *post s′ PID* = *post s PID*
using *step φ PID op*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*auto simp*: *u-defs*) **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op′*: ¬ *open s′*

using *PID step* φ *op* **unfolding** φ*-def2*[*OF step*] **by** *auto*
show *?thesis* **using** *PID′ pPID′ op′ cor1 BO lul*
unfolding Δ*32-def vl1 ul ss1 vl′* **by** *auto*
qed
thus *?*Δ *s′ vl′ s′ ?vl1′* **by** *simp*
qed
next
case *True* **note** *ul = True*
show *?thesis* **proof**(*cases* φ *?trn*)
case *True* **note** φ = *True*
hence *f*: *f ?trn = OVal True* **and** *vl′*: *vl′ = vll*
using *vl c* **unfolding** *consume-def ul* **by** *auto*
have *op′*: *open s′* **using** *f-eq-OVal*[*OF step* φ *f*] *op* **by** *simp*
show *?thesis* **proof**
show *validTrans ?trn1* **using** *step* **unfolding** *ss1* **by** *simp*
next
show *consume ?trn1 vl1 vll1* **using** *ul* φ *c*
unfolding *vl1 vl′ vl ss1 consume-def* **by** *auto*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **note** γ = *this*
thus *g ?trn = g ?trn1* **unfolding** *ss1* **by** *simp*
next
have *pPID′*: *post s′ PID = post s PID*
using *step* φ *PID op op′ f*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*auto simp*: *u-defs*) **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
show *?*Δ *s′ vl′ s′ vll1* **using** *BO* **proof** *cases*
case *BO-PVal*
hence Δ*2 s′ vl′ s′ vll1* **using** *PID′ pPID′ op′ cor1 BO lul*
unfolding Δ*2-def vl1 ul ss1 vl′* **by** *auto*
thus *?thesis* **by** *simp*
next
case *BO-BC*
hence Δ*4 s′ vl′ s′ vll1* **using** *PID′ pPID′ op′ cor1 BO lul*
unfolding Δ*4-def vl1 ul ss1 vl′* **by** *auto*
thus *?thesis* **by** *simp*
qed
qed
next
case *False* **note** φ = *False*

hence *vl'*: *vl'* = *vl* **using** *c* **unfolding** *consume-def* **by** *auto*
have *pPID'*: *post s' PID* = *post s PID*
using *step φ PID op*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*auto simp*: *u-defs*) **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op'*: ¬ *open s'*
using *PID step φ op* **unfolding** *φ-def2*[*OF step*] **by** (*cases a*) *auto*
show *?thesis* **proof**
show *validTrans ?trn1* **using** *step* **unfolding** *ss1* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** *ul φ* **unfolding** *vl1 consume-def*
ss1 **by** *simp*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **note** *γ* = *this*
thus *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
have *Δ32 s' vl' s' vl1* **using** *PID' pPID' op' cor1 BO lul*
unfolding *Δ32-def vl vl1 ul ss1 vl'* **apply** *simp*
apply(*rule exI*[*of* - []])
apply(*rule exI*[*of* - *vll*]) **apply**(*rule exI*[*of* - *vll1*]) **by** *auto*
thus *?Δ s' vl' s' vl1* **by** *simp*
qed
qed
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vl* **by** *simp*
qed
qed

lemma *unwind-cont-Δ2*: *unwind-cont Δ2* {*Δ2*}
proof(*rule*, *simp*)
let *?Δ* = *λs vl s1 vl1*. *Δ2 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *Δ2 s vl s1 vl1*
hence *vlvl1*: *vl* = *vl1*
and *rs*: *reach s* **and** *ss1*: *s1* = *s* **and** *op*: *open s* **and** *PID*: *PID* ∈∈ *postIDs s*
and *cor1*: *corrFrom* (*post s1 PID*) *vl1* **and** *lvl*: *list-all* (*Not* ∘ *isOVal*) *vl*
using *reachNT-reach* **unfolding** *Δ2-def* **by** *auto*

have *own*: *owner s PID* ∈ *set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
have *adm*: *admin s* ∈ *set* (*userIDs s*) **using** *reach-admin-userIDs*[*OF rs own*] **.**
show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn* = *Trans s a ou s′* **let** *?trn1* = *Trans s1 a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *c*: *consume ?trn vl vl′*
have *PID′*: *PID* ∈∈ *postIDs s′* **using** *reach-postIDs-persist*[*OF PID step*] **.**
show *match ?Δ s s1 vl1 a ou s′ vl′* ∨ *ignore ?Δ s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
proof−
have *?match* **proof**(*cases φ ?trn*)
case *True* **note** *φ* = *True*
then obtain *v* **where** *vl*: *vl* = *v* # *vl′* **and** *f*: *f ?trn* = *v*
using *c* **unfolding** *consume-def φ-def2* **by**(*cases vl*) *auto*
show *?thesis* **proof**(*cases v*)
case (*PVal pst*) **note** *v* = *PVal*
have *a*: *a* = *Uact* (*uPost* (*owner s PID*) (*pass s* (*owner s PID*)) *PID pst*)
using *f-eq-PVal*[*OF step φ f*[*unfolded v*]] **.**
have *ou*: *ou* = *outOK* **using** *step own PID* **unfolding** *a* **by** (*auto simp*: *u-defs*)
have *op′*: *open s′* **using** *step op PID PID′ φ*
unfolding *open-def a* **by** (*auto simp*: *u-defs*)
have *pPID′*: *post s′ PID* = *pst*
using *step φ PID op f op′* **unfolding** *a* **by**(*auto simp*: *u-defs*)
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl′* **using** *φ vlvl1* **unfolding** *ss1 consume-def vl f* **by** *auto*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
show *?Δ s′ vl′ s′ vl′* **using** *cor1 PID′ pPID′ op′ lvl vlvl1 ss1*
unfolding *Δ2-def vl v* **by** *auto*
qed
next
case (*PValS aid pid*) **note** *v* = *PValS*
have *a*: *a* = *COMact* (*comSendPost* (*admin s*) (*pass s* (*admin s*)) *aid PID*)
using *f-eq-PValS*[*OF step φ f*[*unfolded v*]] **.**
have *op′*: *open s′* **using** *step op PID PID′ φ*
unfolding *open-def a* **by** (*auto simp*: *com-defs*)
have *ou*: *ou* ≠ *outErr* **using** *φ op op′* **unfolding** *φ-def2*[*OF step*]

```
unfolding a by auto
      have pPID': post s' PID = post s PID
      using step φ PID op f op' unfolding a by(auto simp: com-defs)
      show ?thesis proof
        show validTrans ?trn1 unfolding ss1 using step by simp
      next
        show consume ?trn1 vl1 vl' using φ vlvl1 unfolding ss1 consume-def
vl f by auto
      next
        show γ ?trn = γ ?trn1 unfolding ss1 by simp
      next
        assume γ ?trn thus g ?trn = g ?trn1 unfolding ss1 by simp
      next
        show ?Δ s' vl' s' vl' using cor1 PID' pPID' op' lvl vlvl1 ss1
        unfolding Δ2-def vl v by auto
      qed
    qed(insert vl lvl, auto)
  next
    case False note φ = False
    hence vl': vl' = vl using c unfolding consume-def by auto
    have pPID': post s' PID = post s PID
      using step φ PID op
      apply(cases a)
        subgoal for x1 apply(cases x1) apply(fastforce simp: s-defs)+ .
        subgoal for x2 apply(cases x2) apply(fastforce simp: c-defs)+ .
        subgoal for x3 apply(cases x3) apply(fastforce simp: d-defs)+ .
        subgoal for x4 apply(cases x4) apply(auto simp: u-defs) .
        subgoal by auto
        subgoal by auto
        subgoal for x7 apply(cases x7) apply(fastforce simp: com-defs)+ .
      done
    have op': open s'
      using PID step φ op unfolding φ-def2[OF step] by (cases a) auto
    show ?thesis proof
      show validTrans ?trn1 unfolding ss1 using step by simp
    next
      show consume ?trn1 vl1 vl using φ vlvl1 unfolding ss1 consume-def vl'
by simp
    next
      show γ ?trn = γ ?trn1 unfolding ss1 by simp
    next
      assume γ ?trn thus g ?trn = g ?trn1 unfolding ss1 by simp
    next
      show ?Δ s' vl' s' vl using cor1 PID' op' lvl vlvl1 pPID'
      unfolding Δ2-def vl' ss1 by auto
    qed
  qed
thus ?thesis by simp
qed
```

    qed
  thus *?thesis* **using** *vlvl1* **by** *simp*
  qed
qed

lemma *unwind-cont-Δ4*: *unwind-cont* Δ*4* {Δ*1*,Δ*31*,Δ*32*,Δ*4*}
proof(*rule*, *simp*)
  let *?*Δ = λ*s vl s1 vl1*. Δ*1 s vl s1 vl1* ∨ Δ*31 s vl s1 vl1* ∨ Δ*32 s vl s1 vl1* ∨ Δ*4 s vl s1 vl1*
  fix *s s1* :: *state* **and** *vl vl1* :: *value list*
  assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** Δ*4 s vl s1 vl1*
  then obtain *ul vll vll1* **where** *vl*: *vl* = *ul* @ *OVal False* # *vll* **and** *vl1*: *vl1* = *ul* @ *OVal False* # *vll1*
  and *rs*: *reach s* **and** *ss1*: *s1* = *s* **and** *op*: *open s* **and** *PID*: *PID* ∈∈ *postIDs s*
  and *cor1*: *corrFrom* (*post s1 PID*) *vl1* **and** *lul*: *list-all* (*Not* ∘ *isOVal*) *ul*
  and *BC*: *BC vll vll1*
  using *reachNT-reach* **unfolding** Δ*4-def* **by** *blast*
  have *own*: *owner s PID* ∈ *set* (*userIDs s*) **using** *reach-owner-userIDs*[*OF rs PID*] **.**
  have *adm*: *admin s* ∈ *set* (*userIDs s*) **using** *reach-admin-userIDs*[*OF rs own*] **.**
  show *iaction ?*Δ *s vl s1 vl1* ∨
      ((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
  proof−
    have *?react* **proof**
      fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
      let *?trn* = *Trans s a ou s′* **let** *?trn1* = *Trans s1 a ou s′*
      assume *step*: *step s a* = (*ou*, *s′*) **and** *c*: *consume ?trn vl vl′*
      have *PID′*: *PID* ∈∈ *postIDs s′* **using** *reach-postIDs-persist*[*OF PID step*] **.**
      show *match ?*Δ *s s1 vl1 a ou s′ vl′* ∨ *ignore ?*Δ *s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
      proof−
        have *?match* **proof**(*cases* φ *?trn*)
          case *True* **note** φ = *True*
          then obtain *v* **where** *vl-vl′*: *vl* = *v* # *vl′* **and** *f*: *f ?trn* = *v*
          using *c* **unfolding** *consume-def* φ*-def2* **by**(*cases vl*) *auto*
          show *?thesis* **proof**(*cases ul* = [])
            case *False* **note** *ul* = *False*
            then obtain *ul′* **where** *ul*: *ul* = *v* # *ul′* **and** *vl′*: *vl′* = *ul′* @ *OVal False* # *vll*
            using *c* φ *f* **unfolding** *consume-def vl* **by** (*cases ul*) *auto*
            let *?vl1′* = *ul′* @ *OVal False* # *vll1*
            show *?thesis* **proof**(*cases v*)
              case (*PVal pst*) **note** *v* = *PVal*
              have *a*: *a* = *Uact* (*uPost* (*owner s PID*) (*pass s* (*owner s PID*)) *PID pst*)
              using *f-eq-PVal*[*OF step* φ *f*[*unfolded v*]] **.**
              have *ou*: *ou* = *outOK* **using** *step own PID* **unfolding** *a* **by** (*auto simp*: *u-defs*)
              have *op′*: *open s′* **using** *step op PID PID′* φ

unfolding *open-def a* **by** (*auto simp*: *u-defs*)
have *pPID′*: *post s′ PID* = *pst*
using *step φ PID op f op′* **unfolding** *a* **by**(*auto simp*: *u-defs*)
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 ?vl1′* **using** *φ*
unfolding *ss1 consume-def vl f ul vl1 vl′* **by** *simp*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
have *Δ4 s′ vl′ s′ ?vl1′* **using** *cor1 PID′ pPID′ op′ vl1 ss1 lul BC*
unfolding *Δ4-def vl v ul vl′* **apply** *simp*
apply(*rule exI*[*of - ul′*])
apply(*rule exI*[*of - vll*]) **apply**(*rule exI*[*of - vll1*])
by *auto*
thus *?Δ s′ vl′ s′ ?vl1′* **by** *simp*
qed
next
case (*PValS aid pid*) **note** *v* = *PValS*
have *a*: *a* = *COMact* (*comSendPost* (*admin s*) (*pass s* (*admin s*)) *aid PID*)
using *f-eq-PValS*[*OF step φ f*[*unfolded v*]] **.**
have *op′*: *open s′* **using** *step op PID PID′ φ*
unfolding *open-def a* **by** (*auto simp*: *com-defs*)
have *ou*: *ou* ≠ *outErr* **using** *φ op op′* **unfolding** *φ-def2*[*OF step*] **unfolding** *a* **by** *auto*
have *pPID′*: *post s′ PID* = *post s PID*
using *step φ PID op f op′* **unfolding** *a* **by**(*auto simp*: *com-defs*)
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 ?vl1′* **using** *φ*
unfolding *ss1 consume-def vl f ul vl1 vl′* **by** *simp*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
have *Δ4 s′ vl′ s′ ?vl1′* **using** *cor1 PID′ pPID′ op′ vl1 ss1 lul BC*
unfolding *Δ4-def vl v ul vl′* **by** *auto*
thus *?Δ s′ vl′ s′ ?vl1′* **by** *simp*
qed
qed(*insert vl lul ul*, *auto*)
next
case *True* **note** *ul* = *True*
hence *f*: *f ?trn* = *OVal False* **and** *vl′*: *vl′* = *vll*

using *vl c f* φ **unfolding** *consume-def ul* **by** *auto*
have *op*′: ¬ *open s*′ **using** *f-eq-OVal*[*OF step* φ *f*] *op* **by** *simp*
show *?thesis* **proof**
  show *validTrans ?trn1* **using** *step* **unfolding** *ss1* **by** *simp*
next
  show *consume ?trn1 vl1 vll1* **using** *ul* φ *c*
  unfolding *vl1 vl*′ *vl ss1 consume-def* **by** *auto*
next
  show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
  assume γ *?trn* **note** γ = *this*
  thus *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
  have *pPID*′: *post s*′ *PID* = *post s PID*
  using *step* φ *PID op op*′ *f*
  apply(*cases a*)
    subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
    subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
    subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
    subgoal for *x4* **apply**(*cases x4*) **apply**(*auto simp*: *u-defs*) **.**
    subgoal by *auto*
    subgoal by *auto*
    subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
  done
  show *?*Δ *s*′ *vl*′ *s*′ *vll1* **using** *BC* **proof** *cases*
    case *BC-PVal*
    hence Δ*1 s*′ *vl*′ *s*′ *vll1* **using** *PID*′ *pPID*′ *op*′ *cor1 BC lul*
    unfolding Δ*1-def vl1 ul ss1 vl*′ **by** *auto*
    thus *?thesis* **by** *simp*
  next
    case (*BC-BO Vll Vll1 Ul Ul1 Sul*)
    show *?thesis* **proof**(*cases Ul* ≠ [] ∧ *Ul1* ≠ [])
      case *True*
      hence Δ*31 s*′ *vl*′ *s*′ *vll1* **using** *PID*′ *pPID*′ *op*′ *cor1 BC BC-BO lul*
      unfolding Δ*31-def vl1 ul ss1 vl*′ **apply** *simp*
      apply(*rule exI*[*of - Ul*]) **apply**(*rule exI*[*of - Ul1*])
      apply(*rule exI*[*of - Sul*])
      apply(*rule exI*[*of - Vll*]) **apply**(*rule exI*[*of - Vll1*])
      by *auto*
      thus *?thesis* **by** *simp*
    next
      case *False*
      hence *0*: *Ul* = [] *Ul1* = [] **using** *BC-BO* **by** *auto*
      hence Δ*32 s*′ *vl*′ *s*′ *vll1* **using** *PID*′ *pPID*′ *op*′ *cor1 BC BC-BO lul*
      unfolding Δ*32-def vl1 ul ss1 vl*′ **apply** *simp*
      apply(*rule exI*[*of - Sul*])
      apply(*rule exI*[*of - Vll*]) **apply**(*rule exI*[*of - Vll1*])
      by *auto*
      thus *?thesis* **by** *simp*

qed
qed
qed
qed
next
case *False* **note** φ = *False*
hence *vl'*: *vl'* = *vl* **using** *c* **unfolding** *consume-def* **by** *auto*
have *pPID'*: *post s' PID* = *post s PID*
using *step* φ *PID op*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*auto simp*: *u-defs*) **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
have *op'*: *open s'*
using *PID step* φ *op* **unfolding** φ*-def2*[*OF step*] **by** (*cases a*) *auto*
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** φ **unfolding** *ss1 consume-def vl' vl vl1* **by** *simp*
next
show γ *?trn* = γ *?trn1* **unfolding** *ss1* **by** *simp*
next
assume γ *?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
have Δ*4 s' vl' s' vl1* **using** *cor1 PID' pPID' op' vl1 ss1 lul BC*
unfolding Δ*4-def vl vl'* **by** *auto*
thus *?*Δ *s' vl' s' vl1* **by** *simp*
qed
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *vl* **by** *simp*
qed
qed

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*
apply (*rule unwind-decomp-secure-graph*[*of Gr* Δ*0*])
unfolding *Gr-def*
apply (*simp*, *smt insert-subset order-refl*)
using
istate-Δ*0 unwind-cont-*Δ*0 unwind-cont-*Δ*1 unwind-cont-*Δ*11*
unwind-cont-Δ*31 unwind-cont-*Δ*32 unwind-cont-*Δ*2 unwind-cont-*Δ*4*
unfolding *Gr-def* **by** *auto*

end

end

theory *Independent-Post-Observation-Setup-RECEIVER*
imports
../../Safety-Properties
../Post-Observation-Setup-RECEIVER
begin

6.6.5 Receiver observation setup

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

fun γ :: (*state*,*act*,*out*) *trans* $\Rightarrow$ *bool* **where**
γ (*Trans - a - -*) $\longleftrightarrow$
(∃ *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*))

$\vee$
$(\exists\, uid\ p\ uid'\ p'.\ a = Cact\ (cUser\ uid\ p\ uid'\ p'))$
$\vee$
$(\exists\, uid\ p\ pid.\ a = Cact\ (cPost\ uid\ p\ pid))$
$\vee$
$(\exists\, uid\ p\ uid'.\ a = Cact\ (cFriend\ uid\ p\ uid'))$
$\vee$
$(\exists\, uid\ p\ uid'.\ a = Dact\ (dFriend\ uid\ p\ uid'))$
$\vee$
$(\exists\, uid\ p\ pid\ v.\ a = Uact\ (uVisPost\ uid\ p\ pid\ v))$

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* (*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* ⟷ (∃ *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 $\longleftrightarrow$ *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′*
by (*cases ca*; *auto*)+

lemma *g-simps*:
g (*Trans s a ou s′*) = (*COMact* (*comSendServerReq uID p aID reqInfo*), *ou′*)
$\longleftrightarrow$ ($\exists$ *p′. a = COMact* (*comSendServerReq uID p′ aID reqInfo*) $\wedge$ *p = emptyPass* $\wedge$ *ou = ou′*)
g (*Trans s a ou s′*) = (*COMact* (*comReceiveClientReq aID reqInfo*), *ou′*)
$\longleftrightarrow$ *a = COMact* (*comReceiveClientReq aID reqInfo*) $\wedge$ *ou = ou′*
g (*Trans s a ou s′*) = (*COMact* (*comConnectClient uID p aID sp*), *ou′*)
$\longleftrightarrow$ ($\exists$ *p′. a = COMact* (*comConnectClient uID p′ aID sp*) $\wedge$ *p = emptyPass* $\wedge$ *ou = ou′*)
g (*Trans s a ou s′*) = (*COMact* (*comConnectServer aID sp*), *ou′*)
$\longleftrightarrow$ *a = COMact* (*comConnectServer aID sp*) $\wedge$ *ou = ou′*
g (*Trans s a ou s′*) = (*COMact* (*comReceivePost aID sp pID pst′ uID v*), *ou′*)
$\longleftrightarrow$ ($\exists$ *pst. a = COMact* (*comReceivePost aID sp pID pst uID v*) $\wedge$ *pst′* = (*if pID = PID* $\wedge$ *aID = AID then emptyPost else pst*) $\wedge$ *ou = ou′*)
g (*Trans s a ou s′*) = (*COMact* (*comSendPost uID p aID nID*), *O-sendPost* (*aid*, *sp*, *pid*, *pst*, *own*, *v*))
$\longleftrightarrow$ ($\exists$ *p′. a = COMact* (*comSendPost uID p′ aID nID*) $\wedge$ *p = emptyPass* $\wedge$ *ou = O-sendPost* (*aid*, *sp*, *pid*, *pst*, *own*, *v*))
g (*Trans s a ou s′*) = (*COMact* (*comSendCreateOFriend uID p aID uID′*), *ou′*)
$\longleftrightarrow$ ($\exists$ *p′. a =* (*COMact* (*comSendCreateOFriend uID p′ aID uID′*)) $\wedge$ *p = emptyPass* $\wedge$ *ou = ou′*)
g (*Trans s a ou s′*) = (*COMact* (*comReceiveCreateOFriend aID cp uID uID′*), *ou′*)
$\longleftrightarrow$ *a = COMact* (*comReceiveCreateOFriend aID cp uID uID′*) $\wedge$ *ou = ou′*
g (*Trans s a ou s′*) = (*COMact* (*comSendDeleteOFriend uID p aID uID′*), *ou′*)
$\longleftrightarrow$ ($\exists$ *p′. a = COMact* (*comSendDeleteOFriend uID p′ aID uID′*) $\wedge$ *p = emptyPass* $\wedge$ *ou = ou′*)
g (*Trans s a ou s′*) = (*COMact* (*comReceiveDeleteOFriend aID cp uID uID′*), *ou′*)
$\longleftrightarrow$ *a = COMact* (*comReceiveDeleteOFriend aID cp uID uID′*) $\wedge$ *ou = ou′*
by (*cases a*; *auto simp*: *comPurge-simps ObservationSetup-RECEIVER.comPurge.simps*)+

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**
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*:

shows
φ (*Trans s a ou s'*) ⟷
(∃ *uid p pst vs*. *a* = *COMact* (*comReceivePost AID p PID pst uid vs*) ∧ *ou* = *outOK*)

by (*cases Trans s a ou s' rule*: φ.*cases*) *auto*

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*
using *1 2* **unfolding** *a* **by** (*auto simp*: *com-defs*)

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*)
unfolding φ-*def2*
using *comReceivePost-out*[*OF assms*]
by *blast*

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 −
from *assms*(*1*) **obtain** *sp pst uid vs* **where** *a* = *COMact* (*comReceivePost AID sp PID pst uid vs*) ∧ *ou* = *outOK*
unfolding *φ-def2* **by** *auto*
then show *thesis* **proof** −
assume *a* = *COMact* (*comReceivePost AID sp PID pst uid vs*) ∧ *ou* = *outOK*
with *assms*(*2*) **show** *thesis* **by** (*intro Recv*) (*auto simp*: *com-defs*)
qed
qed

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* **.**

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−
have *s's1'*: *eqButPID s' s1'*
using *eqButPID-step local.step ss1 step1* **by** *blast*
show *?thesis* **using** *step step1 φ*
using *eqButPID-stateSelectors*[*OF ss1*]
unfolding *φ-def2*
by (*auto simp*: *u-defs com-defs*)
qed

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'*)
by (*metis eqButPID-step-φ-imp eqButPID-sym assms*)

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** $B = B$
done

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*
using *assms reach-vis* **by** *auto*

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*))

using *assms* **proof** *induct*
  case (*Step trn*) **thus** *?case*
  by (*cases trn*) *auto*
qed (*simp add*: *istate-def*)

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−
  have *s*′*T*: *reachNT s*′ **using** *step sT T* **using** *reachNT-PairI* **by** *blast*
  note *op* = *reachNT-state*[*OF s*′*T*]
  note [*simp*] = *all-defs*
  note *s* = *reachNT-reach*[*OF sT*]
  note *willUse* =
  step step1 γ
  op
  reach-vis[*OF s*]
  eqButPID-stateSelectors[*OF ss1*]
  eqButPID-actions[*OF ss1*]
  eqButPID-update[*OF ss1*] *eqButPID-not-PID*[*OF ss1*]
  show *?thesis*
  proof (*cases a*)
    case (*Sact x1*)
    with *willUse* **show** *?thesis* **by** (*cases x1*) *auto*
  next
    case (*Cact x2*)
    with *willUse* **show** *?thesis* **by** (*cases x2*) *auto*
  next
    case (*Dact x3*)
    with *willUse* **show** *?thesis* **by** (*cases x3*) *auto*
  next
    case (*Uact x4*)
    with *willUse* **show** *?thesis* **by** (*cases x4*) *auto*
  next
    case (*Ract x5*)
    with *willUse* **show** *?thesis*
    proof (*cases x5*)
      case (*rOPost uid p aid pid*)
      with *Ract willUse* **show** *?thesis* **by** (*cases aid* = *AID* ∧ *pid* = *PID*) *auto*
    qed *auto*
  next
    case (*Lact x6*)
    with *willUse* **show** *?thesis* **by** (*cases x6*) *auto*
  next

case (*COMact x7*)
with *willUse* **show** *?thesis* **by** (*cases x7*) *auto*
qed
qed

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-*Δ*0*:
assumes *B*: *B vl vl1*
shows Δ*0 istate vl istate vl1*
using *assms* **unfolding** Δ*0-def B-def istate-def* **by** *auto*

lemma *unwind-cont-*Δ*0*: *unwind-cont* Δ*0* {Δ*0*,Δ*1*}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. Δ*0 s vl s1 vl1* ∨ Δ*1 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** Δ*0 s vl s1 vl1*
hence *rs*: *reach s* **and** *ss1*: *s1* = *s* **and** *l*: *length vl* = *length vl1*
and *AID*: ¬ *AID* ∈∈ *serverApiIDs s*
using *reachNT-reach* **unfolding** Δ*0-def* **by** *auto*
show *iaction ?*Δ *s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'* **let** *?trn1* = *Trans s1 a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
show *match ?*Δ *s s1 vl1 a ou s' vl'* ∨ *ignore ?*Δ *s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
proof−
have φ: ¬ φ *?trn* **using** *AID step* **unfolding** φ*-def2* **by** (*auto simp*: *u-defs com-defs*)
hence *vl'*: *vl'* = *vl* **using** *c* φ **unfolding** *consume-def* **by** *simp*
have *?match* **proof**(*cases* ∃ *p*. *a* = *COMact* (*comConnectServer AID p*) ∧ *ou* = *outOK*)
case *True*
then obtain *p* **where** *a*: *a* = *COMact* (*comConnectServer AID p*) **and** *ou*: *ou* = *outOK* **by** *auto*

have *AID′*: *AID* ∈∈ *serverApiIDs s′*
using *step AID* **unfolding** *a ou* **by** (*auto simp*: *com-defs*)
note *uid* = *reachNT-state*[*OF rsT*]
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** *φ* **unfolding** *consume-def ss1* **by** *auto*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
have *Δ1 s′ vl′ s′ vl1* **using** *l AID′ c* **unfolding** *ss1 Δ1-def vl′* **by** *auto*
thus *?Δ s′ vl′ s′ vl1* **by** *simp*
qed
next
case *False* **note** *a* = *False*
have *AID′*: ¬ *AID* ∈∈ *serverApiIDs s′*
using *step AID a*
apply(*cases a*)
subgoal for *x1* **apply**(*cases x1*) **apply**(*fastforce simp*: *s-defs*)+ **.**
subgoal for *x2* **apply**(*cases x2*) **apply**(*fastforce simp*: *c-defs*)+ **.**
subgoal for *x3* **apply**(*cases x3*) **apply**(*fastforce simp*: *d-defs*)+ **.**
subgoal for *x4* **apply**(*cases x4*) **apply**(*fastforce simp*: *u-defs*)+ **.**
subgoal by *auto*
subgoal by *auto*
subgoal for *x7* **apply**(*cases x7*) **apply**(*fastforce simp*: *com-defs*)+ **.**
done
show *?thesis* **proof**
show *validTrans ?trn1* **unfolding** *ss1* **using** *step* **by** *simp*
next
show *consume ?trn1 vl1 vl1* **using** *φ* **unfolding** *consume-def ss1* **by** *auto*
next
show *γ ?trn* = *γ ?trn1* **unfolding** *ss1* **by** *simp*
next
assume *γ ?trn* **thus** *g ?trn* = *g ?trn1* **unfolding** *ss1* **by** *simp*
next
have *Δ0 s′ vl′ s′ vl1* **using** *a AID′ l* **unfolding** *Δ0-def vl′ ss1* **by** *simp*
thus *?Δ s′ vl′ s′ vl1* **by** *simp*
qed
qed
thus *?thesis* **by** *simp*
qed
qed
thus *?thesis* **using** *l* **by** *auto*
qed
qed

lemma *unwind-cont-Δ1*: *unwind-cont Δ1* {*Δ1*}
proof(*rule*, *simp*)
let *?Δ* = *λs vl s1 vl1*. *Δ1 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *Δ1 s vl s1 vl1*
hence *rs*: *reach s* **and** *ss1*: *eqButPID s s1*
and *l*: *length vl* = *length vl1* **and** *AID*: *AID* ∈∈ *serverApiIDs s*
using *reachNT-reach* **unfolding** *Δ1-def* **by** *auto*
have *AID1*: *AID* ∈∈ *serverApiIDs s1* **using** *eqButPID-stateSelectors*[*OF ss1*] *AID* **by** *auto*

show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
proof−
have *?match* **proof**(*cases* ∃ *p pst uid vs*. *a* = *COMact* (*comReceivePost AID p PID pst uid vs*) ∧ *ou* = *outOK*)
case *True*
then obtain *p pst uid vs* **where**
a: *a* = *COMact* (*comReceivePost AID p PID pst uid vs*) **and** *ou*: *ou* = *outOK* **by** *auto*
have *p*: *p* = *serverPass s AID* **using** *comReceivePost-out*[*OF step a ou*] **.**
have *p1*: *p* = *serverPass s1 AID* **using** *p ss1 eqButPID-stateSelectors* **by** *auto*
have *φ*: *φ ?trn* **using** *a ou step φ-def2* **by** *auto*
obtain *v* **where** *vl*: *vl* = *v* # *vl'* **and** *f*: *f ?trn* = *v*
using *c φ* **unfolding** *consume-def* **by** (*cases vl*) *auto*
have *AID'*: *AID* ∈∈ *serverApiIDs s'* **using** *step AID* **unfolding** *a ou* **by** (*auto simp*: *com-defs*)
note *uid* = *reachNT-state*[*OF rsT*]
obtain *v1 vl1'* **where** *vl1*: *vl1* = *v1* # *vl1'* **using** *l* **unfolding** *vl* **by** (*cases vl1*) *auto*
obtain *pst1* **where** *v1*: *v1* = *PValR pst1* **by** (*cases v1*) *auto*
define *a1* **where** *a1*: *a1* ≡ *COMact* (*comReceivePost AID p PID pst1 uid vs*)
obtain *s1'* **where** *step1*: *step s1 a1* = (*outOK*, *s1'*) **using** *AID1* **unfolding** *a1 p1* **by** (*simp add*: *com-defs*)
have *s's1'*: *eqButPID s' s1'* **using** *comReceivePost-step-eqButPID*[*OF a -step step1 ss1*] *a1* **by** *simp*
let *?trn1* = *Trans s1 a1 outOK s1'*
have *φ1*: *φ ?trn1* **unfolding** *φ-def2* **unfolding** *a1* **by** *auto*
have *f1*: *f ?trn1* = *v1* **unfolding** *a1 v1* **by** *simp*

```
          show ?thesis proof
            show validTrans ?trn1 using step1 by simp
          next
             show consume ?trn1 vl1 vl1' using φ1 f1 unfolding consume-def ss1
vl1 by simp
          next
            show γ ?trn = γ ?trn1 unfolding a a1 by simp
          next
            assume γ ?trn thus g ?trn = g ?trn1 unfolding a a1 ou by simp
          next
             show Δ1 s' vl' s1' vl1' using l AID' c s's1' unfolding Δ1-def vl vl1
by simp
          qed
        next
          case False note a = False
          obtain s1' ou1 where step1: step s1 a = (ou1, s1') by fastforce
          let ?trn1 = Trans s1 a ou1 s1'
          have φ: ¬ φ ?trn using a step φ-def2 by auto
          have φ1: ¬ φ ?trn1 using φ ss1 step step1 eqButPID-step-φ by blast
          have s's1': eqButPID s' s1' using ss1 step step1 eqButPID-step by blast
           have ouou1: γ ?trn ⟹ ou = ou1 using eqButPID-step-γ-out ss1 step
step1 T rs1 rsT by blast
          have AID': AID ∈∈ serverApiIDs s' using AID step rs using IDs-mono
by auto
          have vl': vl' = vl using c φ unfolding consume-def by simp
          show ?thesis proof
            show validTrans ?trn1 using step1 by simp
          next
            show consume ?trn1 vl1 vl1 using φ1 unfolding consume-def ss1 by
auto
          next
            show 1: γ ?trn = γ ?trn1 unfolding ss1 by simp
          next
            assume γ ?trn hence ou = ou1 using ouou1 by auto
            thus g ?trn = g ?trn1 using ouou1 by (cases a) auto
          next
             show Δ1 s' vl' s1' vl1 using a l s's1' AID' unfolding Δ1-def vl' by
simp
          qed
        qed
        thus ?thesis by simp
      qed
    qed
  thus ?thesis using l by auto
  qed
qed

definition Gr where
Gr =
```

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

theorem *Post-secure*: *secure*
apply (*rule unwind-decomp-secure-graph*[*of Gr* Δ*0*])
unfolding *Gr-def*
apply (*simp*, *smt insert-subset order-refl*)
using *istate-*Δ*0 unwind-cont-*Δ*0 unwind-cont-*Δ*1*
unfolding *Gr-def* **by** *auto*

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 (*cases v*)
case (*Inl vl*) **then show** *thesis* **using** *PVal PValS OVal* **by** (*cases vl rule*: *Post.value.exhaust*) *auto* **next**
case (*Inr vr*) **then show** *thesis* **using** *PValR* **by** (*cases vr rule*: *Post-RECEIVER.value.exhaust*) *auto*
qed

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* **.**

abbreviation φ :: *apiID* $\Rightarrow$ (*state*, *act*, *out*) *trans* $\Rightarrow$ *bool*
where φ *aid trn* $\equiv$ (*if aid* = *AID then Iss.*φ *trn else Post-RECEIVER.*φ *PID AID trn*)

abbreviation *f* :: *apiID* $\Rightarrow$ (*state*, *act*, *out*) *trans* $\Rightarrow$ *value*
where *f aid trn* $\equiv$ (*if aid* = *AID then Inl* (*Iss.f trn*) *else Inr* (*Post-RECEIVER.f PID AID trn*))

abbreviation γ :: *apiID* $\Rightarrow$ (*state*, *act*, *out*) *trans* $\Rightarrow$ *bool*
where γ *aid trn* $\equiv$ (*if aid* = *AID then Iss.*γ *trn else Strong-ObservationSetup-RECEIVER.*γ (*UIDs aid*) *trn*)

abbreviation *g* :: *apiID* $\Rightarrow$ (*state*, *act*, *out*) *trans* $\Rightarrow$ *obs*
where *g aid trn* $\equiv$ (*if aid* = *AID then Iss.g trn else Strong-ObservationSetup-RECEIVER.g PID AID trn*)

abbreviation *T* :: *apiID* $\Rightarrow$ (*state*, *act*, *out*) *trans* $\Rightarrow$ *bool*
where *T aid trn* $\equiv$ (*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-all isl vl* $\wedge$ *list-all isl vl1* $\wedge$ *Iss.B* (*map projl vl*) (*map projl vl1*)
else list-all (*Not o isl*) *vl* $\wedge$ *list-all* (*Not o isl*) *vl1* $\wedge$ *Post-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-RECEIVER.PValR pst*))

lemma *syncVI*: *syncV AID* (*Inl* (*Post.PValS aid′ pst*)) *aid′* (*Inr* (*Post-RECEIVER.PValR pst*))

unfolding *syncV-def* **by** *auto*

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)*
using *assms* **unfolding** *syncV-def* **by** *auto*

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

lemma *comOfV-AID*:
comOfV AID v = Send $\longleftrightarrow$ *isl v* $\wedge$ *Iss.isPValS (projl v)* $\wedge$ *Iss.tgtAPI (projl v)* $\neq$ *AID*
comOfV AID v = Recv $\longleftrightarrow$ *Not (isl v)*
by (*cases v rule*: *value-cases*; *auto*)+

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** $\varphi = \varphi$ **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*
using *AID-in-AIDs* **proof** (*unfold-locales*, *goal-cases*)
case (*1 nid trn*) **then show** *?case* **using** *Iss.validTrans-isCOMact-open*[*of trn*] **by** (*cases trn rule*: *Iss.*φ*.cases*) (*auto simp*: φ*-defs split*: *prod.splits*) **next**
case (*2 nid trn*) **then show** *?case* **using** *Iss.validTrans-isCOMact-open*[*of trn*] **by** (*cases trn rule*: *Iss.*φ*.cases*) (*auto simp*: φ*-defs split*: *prod.splits*) **next**
case (*3 nid trn*)
interpret *Sink*: *Post-RECEIVER UIDs nid PID AID* **.**
show *?case* **using** *3* **by** (*cases* (*nid*,*trn*) *rule*: *tgtNodeOf.cases*) (*auto split*: *prod.splits*)
next
case (*4 nid trn*)
interpret *Sink*: *Post-RECEIVER UIDs nid PID AID* **.**
show *?case* **using** *4* **by** (*cases* (*nid*,*trn*) *rule*: *tgtNodeOf.cases*) (*auto split*: *prod.splits*)
next
case (*5 nid1 trn1 nid2 trn2*)
interpret *Sink1*: *Post-RECEIVER UIDs nid1 PID AID* **.**
interpret *Sink2*: *Post-RECEIVER UIDs nid2 PID AID* **.**
show *?case* **using** *5* **by** (*elim sync-cases*) (*auto intro*: *syncVI*)
next

case (*6 nid1 trn1 nid2 trn2*)
interpret *Sink1*: *Post-RECEIVER UIDs nid1 PID AID* **.**
interpret *Sink2*: *Post-RECEIVER UIDs nid2 PID AID* **.**
show *?case* **using** *6* **by** (*elim sync-cases*) *auto*
next
case (*7 nid1 trn1 nid2 trn2*)
interpret *Sink1*: *Post-RECEIVER UIDs nid1 PID AID* **.**
interpret *Sink2*: *Post-RECEIVER UIDs nid2 PID AID* **.**
show *?case* **using** *7*(*2*,*4*,*6*−*10*)
using *Iss.validTrans-isCOMact-open*[*OF 7*(*2*)] *Iss.validTrans-isCOMact-open*[*OF 7*(*4*)]
by (*elim sync-cases*) (*auto split*: *prod.splits*, *auto simp*: *sendPost-def*)
next
case (*8 nid1 trn1 nid2 trn2*)
interpret *Sink1*: *Post-RECEIVER UIDs nid1 PID AID* **.**
interpret *Sink2*: *Post-RECEIVER UIDs nid2 PID AID* **.**
show *?case* **using** *8*(*2*,*4*,*6*−*10*,*11*,*12*,*13*)
apply (*elim syncO-cases*; *cases trn1*; *cases trn2*)
apply (*auto simp*: *Iss.g-simps Strong-ObservationSetup-RECEIVER.g-simps split*: *prod.splits*)
apply (*auto simp*: *sendPost-def split*: *prod.splits elim*: *syncVE*)[]
done
next
case (*9 nid trn*)
then show *?case*
by (*cases* (*nid*,*trn*) *rule*: *tgtNodeOf.cases*)
(*auto simp*: *Strong-ObservationSetup-RECEIVER.γ.simps*)
next
case (*10 nid trn*) **then show** *?case* **by** (*cases trn*) (*auto simp*: *φ-defs*)
next
case (*11 vSrc nid vn*) **then show** *?case* **by** (*cases vSrc rule*: *value-cases*) (*auto simp*: *syncV-def*)
next
case (*12 vSrc nid vn*) **then show** *?case* **by** (*cases vSrc rule*: *value-cases*) (*auto simp*: *syncV-def*)
qed

lemma *list-all-Not-isl-projectSrcV*: *list-all* (*Not o isl*) (*Net.projectSrcV aid vlSrc*)
proof (*induction vlSrc*)
case (*Cons vSrc vlSrc′*) **then show** *?case* **by** (*cases vSrc rule*: *value-cases*) *auto*
qed *auto*

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

interpretation *Recv*: *Post-RECEIVER UIDs AID′ PID AID* **by** *unfold-locales*

lemma *Iss-BC-BO-tgtAPI*:
shows (*Iss.BC vl vl1* $\longrightarrow$ *map Iss.tgtAPI* (*filter Iss.isPValS vl*) =
map Iss.tgtAPI (*filter Iss.isPValS vl1*)) $\wedge$
(*Iss.BO vl vl1* $\longrightarrow$ *map Iss.tgtAPI* (*filter Iss.isPValS vl*) =
map Iss.tgtAPI (*filter Iss.isPValS vl1*))
by (*induction rule*: *Iss.BC-BO.induct*) *auto*

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*))
using *assms* **proof** (*induction vl vl1 rule*: *list22-induct*)
case (*ConsCons v vl v1 vl1*)
consider (*SendSend*) *aid pst pst1* **where** *v* = *Inl* (*Iss.PValS aid pst*) *v1* = *Inl* (*Iss.PValS aid pst1*)
| (*Internal*) *comOfV AID v* = *Internal* ¬*Iss.isPValS* (*projl v*)
| (*Internal1*) *comOfV AID v1* = *Internal* ¬*Iss.isPValS* (*projl v1*)
using *ConsCons*(*4*−*6*) **by** (*cases v rule*: *value-cases*; *cases v1 rule*: *value-cases*) *auto*
then show *?case* **proof** *cases*
case (*SendSend*) **then show** *?thesis* **using** *ConsCons.IH*(*1*) *ConsCons.prems* **by** *auto*
next
case (*Internal*) **then show** *?thesis* **using** *ConsCons.IH*(*2*)[*of v1 # vl1*] *ConsCons.prems* **by** *auto*
next
case (*Internal1*) **then show** *?thesis* **using** *ConsCons.IH*(*3*)[*of v # vl*] *ConsCons.prems* **by** *auto*
qed
qed (*auto simp*: *comOfV-AID*)

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*))
using *assms Iss-B-Recv-B-aux Iss-BC-BO-tgtAPI* **by** (*auto simp*: *Iss.B-def Recv.B-def*)

end

lemma *map-projl-Inl*: *map* (*projl o Inl*) *vl* = *vl*
by (*induction vl*) *auto*

lemma *these-map-Inl-projl*: *list-all isl vl* $\Longrightarrow$ *these* (*map* (*Some o Inl o projl*) *vl*) = *vl*
by (*induction vl*) *auto*

lemma *map-projr-Inr*: *map* (*projr o Inr*) *vl* = *vl*
by (*induction vl*) *auto*

lemma *these-map-Inr-projr*: *list-all* (*Not o isl*) *vl* ⟹ *these* (*map* (*Some o Inr o projr*) *vl*) = *vl*
by (*induction vl*) *auto*

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 (*unfold-locales*, *goal-cases*)
case *1* **show** *?case* **using** *AID-in-AIDs* **.**
next
case *2*
interpret *Iss'*: *BD-Security-TS-Trans*
istate System-Specification.validTrans srcOf tgtOf Iss.φ Iss.f Iss.γ Iss.g Iss.T Iss.B
istate System-Specification.validTrans srcOf tgtOf Iss.φ λtrn. Inl (*Iss.f trn*) *Iss.γ Iss.g Iss.T B AID*
id id Some Some o Inl
proof (*unfold-locales*, *goal-cases*)
case (*11 vl' vl1' tr*) **then show** *?case*
by (*intro exI*[*of - map projl vl1'*]) (*auto simp*: *map-projl-Inl these-map-Inl-projl*)
qed *auto*
show *?case* **using** *Iss.secure Iss'.translate-secure* **by** *auto*
next
case (*3 aid tr vl' vl1*)
then show *?case*
using *Iss-B-Recv-B*[*of aid* (*Net.lV AID tr*) *vl1*] *list-all-Not-isl-projectSrcV*
by *auto*
qed

theorem *secure*: *secure*
proof (*intro preserve-source-secure ballI*)
fix *aid*
assume *aid*: *aid* ∈ *AIDs* − {*AID*}
interpret *Node*: *Post-RECEIVER UIDs aid PID AID* **.**
interpret *Node'*: *BD-Security-TS-Trans*
istate System-Specification.validTrans srcOf tgtOf Node.φ Node.f Node.γ Node.g Node.T Node.B
istate System-Specification.validTrans srcOf tgtOf Node.φ λtrn. Inr (*Node.f trn*) *Node.γ Node.g Node.T B aid*
id id Some Some o Inr

proof (*unfold-locales*, *goal-cases*)
case (*11 vl′ vl1′ tr*) **then show** *?case* **using** *aid*
by (*intro exI*[*of - map projr vl1′*]) (*auto simp*: *map-projr-Inr these-map-Inr-projr*)
qed *auto*
show *Net.lsecure aid*
using *aid Node.Post-secure Node′.translate-secure* **by** *auto*
qed

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′′*
unfolding *triggerEq-def* **by** *auto*

unbundle *no relcomp-syntax*

context *Post*
begin

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

lemma *triggerEq-open*:
assumes *triggerEq s s′*
shows *open s* ⟷ *open s′*
using *assms* **unfolding** *triggerEq-def open-def* **by** *auto*

lemma *triggerEq-not-γ*:
assumes *validTrans* (*Trans s a ou s′*) **and** ¬γ (*Trans s a ou s′*)
shows *triggerEq s s′*
proof (*cases a*)
case (*Sact sa*) **then show** *?thesis* **using** *assms* **by** (*cases sa*) (*auto simp*: *triggerEq-def s-defs*) **next**
case (*Cact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *triggerEq-def c-defs*) **next**
case (*Dact da*) **then show** *?thesis* **using** *assms* **by** (*cases da*) (*auto simp*: *triggerEq-def d-defs*) **next**
case (*Uact ua*) **then show** *?thesis* **using** *assms* **by** (*cases ua*) (*auto simp*: *trig-*

gerEq-def u-defs) **next**
  case (*Ract ra*) **then show** *?thesis* **using** *assms* **by** (*auto simp*: *triggerEq-def*)
next
  case (*Lact ra*) **then show** *?thesis* **using** *assms* **by** (*auto simp*: *triggerEq-def*)
next
  case (*COMact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *triggerEq-def com-defs*)
qed

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 (*cases a*)
  case (*Sact sa*) **then show** *?thesis* **using** *assms* **by** (*cases sa*) (*auto simp*: *triggerEq-def s-defs*) **next**
  case (*Cact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *triggerEq-def c-defs*) **next**
  case (*Dact da*) **then show** *?thesis* **using** *assms* **by** (*cases da*) (*auto simp*: *triggerEq-def d-defs*) **next**
  case (*Uact ua*) **then show** *?thesis* **using** *assms* **by** (*cases ua*) (*auto simp*: *triggerEq-def u-defs*) **next**
  case (*Ract ra*) **then show** *?thesis* **using** *assms* **by** (*auto simp*: *triggerEq-def*)
next
  case (*Lact ra*) **then show** *?thesis* **using** *assms* **by** (*auto simp*: *triggerEq-def*)
next
  case (*COMact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *triggerEq-def com-defs*)
qed

lemma *triggerEq-obsSteps*:
assumes *validFrom s tr* **and** *triggerEq s s'*
shows *triggerEq* (*tgtOfTrFrom s tr*) (*obsSteps s'* (*O tr*))
using *assms* **proof** (*induction tr arbitrary*: *s s'*)
  case (*Nil s s'*)
  then show *?case* **by** *auto*
next
  case (*Cons trn tr s s'*)
  then obtain *a ou s''* **where** *trn*: *trn* = *Trans s a ou s''* **and** *step*: *step s a* = (*ou*, *s''*)
    by (*cases trn*) (*auto simp*: *validFrom-Cons*)
  show *?case* **proof** *cases*
    assume γ: γ *trn*
    then have *triggerEq s''* (*obsStep s'* (*g trn*)) **unfolding** *trn* **using** *step Cons*(*3*) **by** (*auto intro*: *triggerEq-obsStep*)
    with *Cons.IH*[*OF - this*] *Cons*(*2*) γ *trn* **show** *?thesis* **by** (*auto simp*: *validFrom-Cons*)
  next
    assume *n*γ: $\neg$ γ *trn*
      then have *triggerEq s s''* **using** *Cons*(*2*) **unfolding** *trn* **by** (*intro trig-*

gerEq-not-γ) (*auto simp*: *validFrom-Cons*)
with *Cons*(*3*) **have** *triggerEq s'' s'* **by** (*auto intro*: *triggerEq-sym triggerEq-trans*)
with *Cons.IH*[*OF - this*] *Cons*(*2*) *nγ trn* **show** *?thesis* **by** (*auto simp*: *validFrom-Cons*)
qed
qed

end

context *Post-RECEIVER*
begin

lemma *sPurge-simp*[*simp*]: *sPurge sa* = *sSys* (*sUserOfA sa*) *emptyPass* **by** (*cases sa*) *auto*

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*)
by (*cases trn*) (*auto simp*: *T-state-def*)

lemma *triggerEq-T*:
assumes *triggerEq s s'*
shows *T-state s* ⟷ *T-state s'*
using *assms* **unfolding** *triggerEq-def T-state-def* **by** *auto*

lemma *never-T-not-T-state*:
assumes *validFrom s tr* **and** *never T tr* **and** ¬*T-state s*
shows ¬*T-state* (*tgtOfTrFrom s tr*)
using *assms* **by** (*induction tr arbitrary*: *s rule*: *rev-induct*) (*auto simp*: *T-T-state*)

lemma *triggerEq-not-γ*:
assumes *validTrans* (*Trans s a ou s'*) **and** ¬γ (*Trans s a ou s'*)
shows *triggerEq s s'*
proof (*cases a*)
case (*Sact sa*) **then show** *?thesis* **using** *assms* **by** (*cases sa*) (*auto simp*: *triggerEq-def s-defs*) **next**
case (*Cact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *triggerEq-def c-defs*) **next**
case (*Dact da*) **then show** *?thesis* **using** *assms* **by** (*cases da*) (*auto simp*: *triggerEq-def d-defs*) **next**
case (*Uact ua*) **then show** *?thesis* **using** *assms* **by** (*cases ua*) (*auto simp*: *triggerEq-def u-defs*) **next**
case (*Ract ra*) **then show** *?thesis* **using** *assms* **by** (*auto simp*: *triggerEq-def*)
next

case (*Lact ra*) **then show** *?thesis* **using** *assms* **by** (*auto simp*: *triggerEq-def*)
next
case (*COMact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *triggerEq-def com-defs*)
qed

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 (*cases a*)
case (*Sact sa*) **then show** *?thesis* **using** *assms* **by** (*cases sa*) (*auto simp*: *triggerEq-def s-defs*) **next**
case (*Cact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *triggerEq-def c-defs*) **next**
case (*Dact da*) **then show** *?thesis* **using** *assms* **by** (*cases da*) (*auto simp*: *triggerEq-def d-defs*) **next**
case (*Uact ua*) **then show** *?thesis* **using** *assms* **by** (*cases ua*) (*auto simp*: *triggerEq-def u-defs*) **next**
case (*Ract ra*) **then show** *?thesis* **using** *assms* **by** (*auto simp*: *triggerEq-def*)
next
case (*Lact ra*) **then show** *?thesis* **using** *assms* **by** (*auto simp*: *triggerEq-def*)
next
case (*COMact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *triggerEq-def com-defs*)
qed

lemma *triggerEq-obsSteps*:
assumes *validFrom s tr* **and** *triggerEq s s'*
shows *triggerEq* (*tgtOfTrFrom s tr*) (*obsSteps s'* (*O tr*))
using *assms* **proof** (*induction tr arbitrary*: *s s'*)
case (*Nil s s'*)
then show *?case* **by** *auto*
next
case (*Cons trn tr s s'*)
then obtain *a ou s''* **where** *trn*: *trn* = *Trans s a ou s''* **and** *step*: *step s a* = (*ou*, *s''*)
by (*cases trn*) (*auto simp*: *validFrom-Cons*)
show *?case* **proof** *cases*
assume γ: γ *trn*
then have *triggerEq s''* (*obsStep s'* (*g trn*)) **unfolding** *trn* **using** *step Cons*(*3*) **by** (*auto intro*: *triggerEq-obsStep*)
with *Cons.IH*[*OF - this*] *Cons*(*2*) γ *trn* **show** *?thesis* **by** (*auto simp*: *validFrom-Cons*)
next
assume *n*γ: $\neg$ γ *trn*
then have *triggerEq s s''* **using** *Cons*(*2*) **unfolding** *trn* **by** (*intro triggerEq-not-*γ) (*auto simp*: *validFrom-Cons*)
with *Cons*(*3*) **have** *triggerEq s'' s'* **by** (*auto intro*: *triggerEq-sym triggerEq-trans*)

with *Cons.IH*[*OF - this*] *Cons*(*2*) *nγ trn* **show** *?thesis* **by** (*auto simp*: *valid-From-Cons*)
qed
qed

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″*
unfolding *nTriggerEq-def* **by** (*auto intro*: *triggerEq-sym triggerEq-trans*)

lemma *nTriggerEq-open*:
assumes *nTriggerEq s s′*
shows ∀ *aid*. *Iss.open* (*s aid*) ⟷ *Iss.open* (*s′ aid*)
using *assms* **unfolding** *nTriggerEq-def* **by** (*auto intro*!: *Iss.triggerEq-open*)

lemma *nTriggerEq-not-γ*:
assumes *nValidTrans trn* **and** ¬*Net.nγ trn*
shows *nTriggerEq* (*nSrcOf trn*) (*nTgtOf trn*)
proof (*cases trn*)
case (*LTrans s aid1 trn1*)
with *assms* **show** *?thesis* **using** *Iss.triggerEq-not-γ Post-RECEIVER.triggerEq-not-γ*
by (*cases trn1*) (*auto simp*: *nTriggerEq-def*)
next
case (*CTrans s aid1 trn1 aid2 trn2*)
with *assms* **show** *?thesis*
by (*auto elim*: *sync-cases simp*: *Strong-ObservationSetup-RECEIVER.γ.simps Strong-ObservationSetup-ISSUER.γ.simps*)
qed

lemma *nTriggerEq-obsStep*:
assumes *nValidTrans trn* **and** *Net.nγ trn* **and** *nTriggerEq* (*nSrcOf trn*) *s1*

shows *nTriggerEq* (*nTgtOf trn*) (*nObsStep s1* (*Net.ng trn*))
unfolding *nTriggerEq-def* **proof**
fix *aid*
show *triggerEq* (*nTgtOf trn aid*) (*nObsStep s1* (*Net.ng trn*) *aid*)
proof (*cases trn*)
case (*LTrans s aid1 trn1*)
with *assms* **show** *?thesis* **unfolding** *nTriggerEq-def*
by (*cases trn1*) (*auto intro*: *Iss.triggerEq-obsStep Post-RECEIVER.triggerEq-obsStep*)
next
case (*CTrans s aid1 trn1 aid2 trn2*)
then have *sync aid1 trn1 aid2 trn2* **using** *assms* **by** *auto*
moreover obtain *a1 ou1 s1′ a2 ou2 s2′*
where *trn1* = *Trans* (*s aid1*) *a1 ou1 s1′* **and** *trn2* = *Trans* (*s aid2*) *a2 ou2 s2′*
using *CTrans assms* **by** (*cases trn1*, *cases trn2*) *auto*
ultimately show *?thesis* **using** *CTrans assms* **unfolding** *nTriggerEq-def*
using *Iss.triggerEq-obsStep*[*of s aid1 a1 ou1 s1′ s1 aid1*]
using *Iss.triggerEq-obsStep*[*of s aid2 a2 ou2 s2′ s1 aid2*]
using *Post-RECEIVER.triggerEq-obsStep*[*of s aid1 a1 ou1 s1′ UIDs aid1 s1 aid1*]
using *Post-RECEIVER.triggerEq-obsStep*[*of s aid2 a2 ou2 s2′ UIDs aid2 s1 aid2*]
by (*elim sync-cases*) (*auto simp*: *Strong-ObservationSetup-RECEIVER.γ.simps*)
qed
qed

lemma *triggerEq-obsSteps*:
assumes *validFrom s tr* **and** *nTriggerEq s s′*
shows *nTriggerEq* (*nTgtOfTrFrom s tr*) (*nObsSteps s′* (*O tr*))
using *assms* **proof** (*induction tr arbitrary*: *s s′*)
case (*Nil s s′*)
then show *?case* **by** *auto*
next
case (*Cons trn tr s s′*)
have *tr*: *local.validFrom* (*nTgtOf trn*) *tr nTgtOfTrFrom s* (*trn # tr*) = *nTgtOfTrFrom* (*nTgtOf trn*) *tr*
using *Cons*(*2*) **by** *auto*
show *?case* **proof** *cases*
assume *γ*: *Net.nγ trn*
then have *O*: *nObsSteps s′* (*O* (*trn # tr*)) = *nObsSteps* (*nObsStep s′* (*Net.ng trn*)) (*O tr*) **by** *auto*
have *nTriggerEq* (*nTgtOf trn*) (*nObsStep s′* (*Net.ng trn*)) **using** *Cons*(*2*,*3*) *γ*
by (*intro nTriggerEq-obsStep*) *auto*
from *Cons.IH*[*OF tr*(*1*) *this*] **show** *?thesis* **unfolding** *O tr*(*2*) **.**
next
assume *nγ*: ¬ *Net.nγ trn*
then have *O*: *O* (*trn # tr*) = *O tr* **by** *auto*
have *nTriggerEq* (*nSrcOf trn*) (*nTgtOf trn*) **using** *nγ Cons*(*2*) **by** (*intro nTriggerEq-not-γ*) *auto*

with *Cons*(*3*) **have** *nTriggerEq* (*nTgtOf trn*) *s′* **using** *Cons*(*2*) **by** (*auto intro*: *nTriggerEq-sym nTriggerEq-trans*)
from *Cons.IH*[*OF tr*(*1*) *this*] **show** *?thesis* **unfolding** *O tr*(*2*) **.**
qed
qed

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.n*γ *trn* **and** γ': *Net.n*γ *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 −
have ∗: *nTriggerEq* (*nTgtOfTrFrom s tr*) (*nObsSteps s′* (*O tr*)) **using** *tr s-s′*
by (*intro triggerEq-obsSteps*) *auto*
have ∗∗: *nTriggerEq* (*nTgtOfTrFrom s′ tr′*) (*nObsSteps s′* (*O tr′*)) **using** *tr′*
by (*intro triggerEq-obsSteps*) *auto*
from *nTriggerEq-trans*[*OF* ∗[*unfolded O*] ∗∗[*THEN nTriggerEq-sym*]]
show *src*: *nTriggerEq* (*nSrcOf trn*) (*nSrcOf trn′*) **using** *tr tr′*
by (*auto simp*: *nTgtOfTrFrom-nTgtOf-last*)
have *nTriggerEq* (*nTgtOf trn*) (*nObsStep* (*nSrcOf trn′*) (*Net.ng trn*)) **using** *tr* γ *src*
by (*intro nTriggerEq-obsStep*) *auto*
moreover have *nTriggerEq* (*nTgtOf trn′*) (*nObsStep* (*nSrcOf trn′*) (*Net.ng trn′*)) **using** *tr′* γ'
by (*intro nTriggerEq-obsStep*) *auto*
ultimately show *nTriggerEq* (*nTgtOf trn*) (*nTgtOf trn′*) **unfolding** *g*
by (*auto intro*: *nTriggerEq-sym nTriggerEq-trans*)
qed

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.nγ Post1.Net.ng Post1.Net.nT Post1.B AID1
Post2.Net.nφ Post2.nf′ Post2.Net.nγ Post2.Net.ng Post2.Net.nT Post2.B AID2
nγ ng
proof
fix *tr trn*
assume *nγ trn*
then show *Post1.Net.nγ trn* ∧ *Post2.Net.nγ trn*
by (*cases trn rule*: *nγ.cases*) (*auto simp*: *Strong-ObservationSetup-RECEIVER.γ.simps*)
next
fix *tr tr′ trn trn′*
assume *tr*: *validSystemTrace* (*tr* ## *trn*) **and** *tr′*: *validSystemTrace* (*tr′* ## *trn′*)
and *γ*: *Post1.Net.nγ trn* **and** *γ′*: *Post1.Net.nγ trn′* **and** *g*: *Post1.Net.ng trn* = *Post1.Net.ng trn′*
from *tr tr′* **have** *trn*: *Post1.nValidTrans trn Post1.nValidTrans trn′* **by** *auto*
show *nγ trn* = *nγ trn′* **proof** (*cases trn*)
case (*LTrans s aid1 trn1*)
then obtain *s′ trn1′* **where** *trn′*: *trn′* = *LTrans s′ aid1 trn1′* **using** *g* **by** (*cases trn′*) *auto*
then show *?thesis* **using** *LTrans g*
by (*cases trn1 rule*: *Strong-ObservationSetup-ISSUER.g.cases*;
cases trn1′ rule: *Strong-ObservationSetup-ISSUER.g.cases*)
(*auto simp*: *Strong-ObservationSetup-RECEIVER.g.simps Post-RECEIVER.sPurge-simp*)
next
case (*CTrans s aid1 trn1 aid2 trn2*)
then show *?thesis* **using** *g* **by** (*cases trn′*) *auto*
qed
next

```
  fix tr tr′ trn trn′
  assume O: Post1.O tr = Post1.O tr′ and γ: Post1.Net.nγ trn Post1.Net.nγ
trn′
    and tr: validSystemTrace (tr ## trn) and tr′: validSystemTrace (tr′ ## trn′)
     and g: Post1.Net.ng trn = Post1.Net.ng trn′ and γ: nγ trn and γ′: nγ trn′
  then have trn: Post1.nValidTrans trn and trn′: Post1.nValidTrans trn′ by auto
  show ng trn = ng trn′ proof (cases trn)
    case (LTrans s aid1 trn1)
     then obtain s′ trn1′ where trn′ = LTrans s′ aid1 trn1′ using g by (cases
trn′) auto
    then show ?thesis using LTrans γ γ′ g trn trn′
      by (cases (aid1,trn1) rule: Post1.tgtNodeOf.cases;
          cases (aid1,trn1′) rule: Post1.tgtNodeOf.cases)
       (auto simp: Strong-ObservationSetup-RECEIVER.g.simps Post-RECEIVER.sPurge-simp
              simp: Post1.Iss.sStep-unfold split: sActt.splits)
  next
    case (CTrans s aid1 trn1 aid2 trn2)
    with γ show ?thesis by auto
  qed
next
  fix tr tr′ trn trn′
  assume tr: validSystemTrace (tr ## trn) and tr′: validSystemTrace (tr′ ##
trn′)
     and γ: Post2.Net.nγ trn and γ′: Post2.Net.nγ trn′ and g: Post2.Net.ng trn
= Post2.Net.ng trn′
  from tr tr′ have trn: Post1.nValidTrans trn Post1.nValidTrans trn′ by auto
  show nγ trn = nγ trn′ proof (cases trn)
    case (LTrans s aid1 trn1)
     then obtain s′ trn1′ where trn′: trn′ = LTrans s′ aid1 trn1′ using g by
(cases trn′) auto
    then show ?thesis using LTrans g
      by (cases trn1 rule: Strong-ObservationSetup-ISSUER.g.cases;
          cases trn1′ rule: Strong-ObservationSetup-ISSUER.g.cases)
       (auto simp: Strong-ObservationSetup-RECEIVER.g.simps Post-RECEIVER.sPurge-simp)
  next
    case (CTrans s aid1 trn1 aid2 trn2)
    then show ?thesis using g by (cases trn′) auto
  qed
next
  fix tr tr′ trn trn′
  assume O: Post2.O tr = Post2.O tr′ and γ: Post2.Net.nγ trn Post2.Net.nγ
trn′
    and tr: validSystemTrace (tr ## trn) and tr′: validSystemTrace (tr′ ## trn′)
     and g: Post2.Net.ng trn = Post2.Net.ng trn′ and γ: nγ trn and γ′: nγ trn′
  then have trn: Post1.nValidTrans trn and trn′: Post1.nValidTrans trn′ by auto
  show ng trn = ng trn′ proof (cases trn)
    case (LTrans s aid1 trn1)
     then obtain s′ trn1′ where trn′ = LTrans s′ aid1 trn1′ using g by (cases
trn′) auto
```

then show *?thesis* **using** *LTrans* γ γ' *g trn trn'*
by (*cases* (*aid1*,*trn1*) *rule*: *Post1.tgtNodeOf.cases*;
cases (*aid1*,*trn1'*) *rule*: *Post1.tgtNodeOf.cases*)
(*auto simp*: *Strong-ObservationSetup-RECEIVER.g.simps Post-RECEIVER.sPurge-simp*
simp: *Post1.Iss.sStep-unfold split*: *sActt.splits*)
next
case (*CTrans s aid1 trn1 aid2 trn2*)
with γ **show** *?thesis* **by** *auto*
qed
next
fix *tr trn*
assume *validSystemTrace* (*tr ## trn*) **and** *n*φ: *Post2.Net.n*φ *trn*
then have *trn*: *Post1.nValidTrans trn* **by** *auto*
show *Post1.Net.n*γ *trn* **proof** (*cases trn*)
case (*LTrans s aid1 trn1*)
then obtain *a ou s1'* **where** *trn1*: *trn1* = *Trans* (*s aid1*) *a ou s1'* **using** *trn*
by (*cases trn1*) *auto*
then show *?thesis* **using** *n*φ *trn LTrans AID1-neq-AID2*
using *Post2.Iss.triggerEq-not-*γ[*THEN Post2.Iss.triggerEq-open*]
by (*cases Post2.Iss.*γ *trn1*) (*auto simp*: *Post2.*φ*-defs Strong-ObservationSetup-RECEIVER.*γ*.simps*)
next
case (*CTrans s aid1 trn1 aid2 trn2*)
with *trn* **have** *Post1.sync aid1 trn1 aid2 trn2* **by** *auto*
then show *?thesis* **using** *trn CTrans*
by (*elim Post1.sync-cases*) (*auto simp*: *Strong-ObservationSetup-RECEIVER.*γ*.simps*)
qed
next
fix *tr tr' trn trn'*
assume *O*: *Post1.O tr* = *Post1.O tr'* **and** γ: *Post1.Net.n*γ *trn Post1.Net.n*γ
trn'
and *tr*: *validSystemTrace* (*tr ## trn*) **and** *tr'*: *validSystemTrace* (*tr' ## trn'*)
and *g*: *Post1.Net.ng trn* = *Post1.Net.ng trn'*
have *op*: $\forall$ *aid. Post2.Iss.open* (*Post1.nSrcOf trn aid*) $\longleftrightarrow$ *Post2.Iss.open* (*Post1.nSrcOf*
trn' aid)
using *O* γ *tr tr' g* **by** (*intro Post2.nTriggerEq-open Post1.O-eq-nTriggerEq*)
auto
have *op'*: $\forall$ *aid. Post2.Iss.open* (*Post1.nTgtOf trn aid*) $\longleftrightarrow$ *Post2.Iss.open* (*Post1.nTgtOf*
trn' aid)
using *O* γ *tr tr' g* **by** (*intro Post2.nTriggerEq-open Post1.O-eq-nTriggerEq*)
auto
have *trn*: *Post1.nValidTrans trn* **and** *trn'*: *Post1.nValidTrans trn'* **using** *tr tr'*
by *auto*
show *Post2.Net.n*φ *trn* = *Post2.Net.n*φ *trn'*
proof (*cases trn*)
case (*LTrans s aid1 trn1*)
then obtain *s' trn1'* **where** *s'*: *trn'* = *LTrans s' aid1 trn1'* **using** *g* **by** (*cases*
trn') *auto*
moreover then have *srcOf trn1* = *s aid1 srcOf trn1'* = *s' aid1*
tgtOf trn1 = *Post1.nTgtOf trn aid1 tgtOf trn1'* = *Post1.nTgtOf*

trn′ aid1
using *LTrans trn trn′* **by** *auto*
ultimately show *?thesis* **using** *LTrans op op′ g AID1-neq-AID2*
by (*cases trn1 rule*: *Post.φ.cases*; *cases trn1′ rule*: *Post.φ.cases*)
(*auto simp*: *Strong-ObservationSetup-RECEIVER.g.simps Strong-ObservationSetup-RECEIVER.comPurg*
Post.φ.simps Post-RECEIVER.φ.simps)
next
case (*CTrans s aid1 trn1 aid2 trn2*)
then obtain *s′ trn1′ trn2′* **where** *CTrans′*: *trn′* = *CTrans s′ aid1 trn1′ aid2*
trn2′
using *g* **by** (*cases trn′*) *auto*
have *Post1.sync aid1 trn1 aid2 trn2 Post1.sync aid1 trn1′ aid2 trn2′*
using *CTrans CTrans′ trn trn′* **by** *auto*
then show *?thesis* **using** *CTrans CTrans′ trn trn′ op op′ g*
by (*elim Post1.sync-cases*)
(*auto simp*: *Post-RECEIVER.φ.simps Strong-ObservationSetup-RECEIVER.g.simps*
Strong-ObservationSetup-RECEIVER.comPurge.simps)
qed
next
fix *tr tr′ trn trn′*
assume *O*: *Post1.O tr* = *Post1.O tr′* **and** *γ*: *Post1.Net.nγ trn Post1.Net.nγ*
trn′
and *tr*: *validSystemTrace* (*tr ## trn*) **and** *tr′*: *validSystemTrace* (*tr′ ## trn′*)
and *g*: *Post1.Net.ng trn* = *Post1.Net.ng trn′*
and *φ*: *Post2.Net.nφ trn* **and** *φ′*: *Post2.Net.nφ trn′*
have *op*: ∀ *aid*. *Post2.Iss.open* (*Post1.nSrcOf trn aid*) ⟷ *Post2.Iss.open* (*Post1.nSrcOf*
trn′ aid)
using *O γ tr tr′ g* **by** (*intro Post2.nTriggerEq-open Post1.O-eq-nTriggerEq*)
auto
have *op′*: ∀ *aid*. *Post2.Iss.open* (*Post1.nTgtOf trn aid*) ⟷ *Post2.Iss.open* (*Post1.nTgtOf*
trn′ aid)
using *O γ tr tr′ g* **by** (*intro Post2.nTriggerEq-open Post1.O-eq-nTriggerEq*)
auto
have *trn*: *Post1.nValidTrans trn* **and** *trn′*: *Post1.nValidTrans trn′* **using** *tr tr′*
by *auto*
show *Post2.nf′ trn* = *Post2.nf′ trn′*
proof (*cases trn*)
case (*LTrans s aid1 trn1*)
then obtain *s′ trn1′* **where** *s′*: *trn′* = *LTrans s′ aid1 trn1′* **using** *g* **by** (*cases*
trn′) *auto*
moreover then have *srcOf trn1* = *s aid1 srcOf trn1′* = *s′ aid1*
tgtOf trn1 = *Post1.nTgtOf trn aid1 tgtOf trn1′* = *Post1.nTgtOf*
trn′ aid1
using *LTrans trn trn′* **by** *auto*
ultimately show *?thesis* **using** *LTrans φ φ′ op′ g AID1-neq-AID2*
by (*cases trn1 rule*: *Post.φ.cases*; *cases trn1′ rule*: *Post.f.cases*)
(*auto simp*: *Strong-ObservationSetup-RECEIVER.g.simps Strong-ObservationSetup-RECEIVER.comPurg*
Post.φ.simps Post-RECEIVER.φ.simps)
next

case (*CTrans s aid1 trn1 aid2 trn2*)
then obtain *s′ trn1′ trn2′* **where** *CTrans′*: *trn′* = *CTrans s′ aid1 trn1′ aid2 trn2′*
using *g* **by** (*cases trn′*) *auto*
then have *trn1*: *validTrans trn1* **and** *trn1′*: *validTrans trn1′* **using** *trn trn′ CTrans* **by** *auto*
have *states*: *tgtOf trn1* = *Post1.nTgtOf trn aid1 tgtOf trn2* = *Post1.nTgtOf trn aid2*
tgtOf trn1′ = *Post1.nTgtOf trn′ aid1 tgtOf trn2′* = *Post1.nTgtOf trn′ aid2*
using *trn trn′ CTrans CTrans′* **by** *auto*
have *Post1.sync aid1 trn1 aid2 trn2 Post1.sync aid1 trn1′ aid2 trn2′*
using *CTrans CTrans′ trn trn′* **by** *auto*
then show *?thesis* **using** *CTrans CTrans′ op′ g states AID1-neq-AID2*
by (*elim Post1.sync-cases*[*OF - trn1*] *Post1.sync-cases*[*OF - trn1′*])
(*auto simp*: *Post-RECEIVER.φ.simps Strong-ObservationSetup-RECEIVER.g.simps Strong-ObservationSetup-RECEIVER.comPurge.simps*)
qed
next
fix *tr trn*
assume *validSystemTrace* (*tr ## trn*) **and** *nφ*: *Post1.Net.nφ trn*
then have *trn*: *Post1.nValidTrans trn* **by** *auto*
show *Post2.Net.nγ trn* **proof** (*cases trn*)
case (*LTrans s aid1 trn1*)
then obtain *a ou s1′* **where** *trn1*: *trn1* = *Trans* (*s aid1*) *a ou s1′* **using** *trn* **by** (*cases trn1*) *auto*
then show *?thesis* **using** *nφ trn LTrans AID1-neq-AID2*
using *Post1.Iss.triggerEq-not-γ*[*THEN Post1.Iss.triggerEq-open*]
by (*cases Post1.Iss.γ trn1*) (*auto simp*: *Post1.φ-defs Strong-ObservationSetup-RECEIVER.γ.simps*)
next
case (*CTrans s aid1 trn1 aid2 trn2*)
with *trn* **have** *Post1.sync aid1 trn1 aid2 trn2* **by** *auto*
then show *?thesis*
using *trn CTrans*
by (*elim Post1.sync-cases*) (*auto simp*: *Strong-ObservationSetup-RECEIVER.γ.simps*)
qed
next
fix *tr tr′ trn trn′*
assume *O*: *Post2.O tr* = *Post2.O tr′* **and** *γ*: *Post2.Net.nγ trn Post2.Net.nγ trn′*
and *tr*: *validSystemTrace* (*tr ## trn*) **and** *tr′*: *validSystemTrace* (*tr′ ## trn′*)
and *g*: *Post2.Net.ng trn* = *Post2.Net.ng trn′*
have *op*: ∀ *aid*. *Post1.Iss.open* (*Post1.nSrcOf trn aid*) ⟷ *Post1.Iss.open* (*Post1.nSrcOf trn′ aid*)
using *O γ tr tr′ g* **by** (*intro Post1.nTriggerEq-open Post2.O-eq-nTriggerEq*) *auto*
have *op′*: ∀ *aid*. *Post1.Iss.open* (*Post1.nTgtOf trn aid*) ⟷ *Post1.Iss.open* (*Post1.nTgtOf trn′ aid*)
using *O γ tr tr′ g* **by** (*intro Post1.nTriggerEq-open Post2.O-eq-nTriggerEq*)

auto
have *trn*: *Post1.nValidTrans trn* **and** *trn'*: *Post1.nValidTrans trn'* **using** *tr tr'*
by *auto*
show *Post1.Net.nφ trn = Post1.Net.nφ trn'*
proof (*cases trn*)
case (*LTrans s aid1 trn1*)
then obtain *s' trn1'* **where** *s'*: *trn' = LTrans s' aid1 trn1'* **using** *g* **by** (*cases*
trn') *auto*
moreover then have *srcOf trn1 = s aid1 srcOf trn1' = s' aid1*
tgtOf trn1 = Post1.nTgtOf trn aid1 tgtOf trn1' = Post1.nTgtOf
trn' aid1
using *LTrans trn trn'* **by** *auto*
ultimately show *?thesis* **using** *LTrans op op' g AID1-neq-AID2*
by (*cases trn1 rule*: *Post.φ.cases*; *cases trn1' rule*: *Post.φ.cases*)
(*auto simp*: *Strong-ObservationSetup-RECEIVER.g.simps Strong-ObservationSetup-RECEIVER.comPur*
Post.φ.simps Post-RECEIVER.φ.simps)
next
case (*CTrans s aid1 trn1 aid2 trn2*)
then obtain *s' trn1' trn2'* **where** *CTrans'*: *trn' = CTrans s' aid1 trn1' aid2*
trn2'
using *g* **by** (*cases trn'*) *auto*
have *Post1.sync aid1 trn1 aid2 trn2 Post1.sync aid1 trn1' aid2 trn2'*
using *CTrans CTrans' trn trn'* **by** *auto*
then show *?thesis* **using** *CTrans CTrans' trn trn' op op' g*
by (*elim Post1.sync-cases*)
(*auto simp*: *Post-RECEIVER.φ.simps Strong-ObservationSetup-RECEIVER.g.simps*
Strong-ObservationSetup-RECEIVER.comPurge.simps)
qed
next
fix *tr tr' trn trn'*
assume *O*: *Post2.O tr = Post2.O tr'* **and** *γ*: *Post2.Net.nγ trn Post2.Net.nγ*
trn'
and *tr*: *validSystemTrace* (*tr ## trn*) **and** *tr'*: *validSystemTrace* (*tr' ## trn'*)
and *g*: *Post2.Net.ng trn = Post2.Net.ng trn'*
and *φ*: *Post1.Net.nφ trn* **and** *φ'*: *Post1.Net.nφ trn'*
have *op*: ∀ *aid. Post1.Iss.open* (*Post1.nSrcOf trn aid*) ⟷ *Post1.Iss.open* (*Post1.nSrcOf*
trn' aid)
using *O γ tr tr' g* **by** (*intro Post1.nTriggerEq-open Post2.O-eq-nTriggerEq*)
auto
have *op'*: ∀ *aid. Post1.Iss.open* (*Post1.nTgtOf trn aid*) ⟷ *Post1.Iss.open* (*Post1.nTgtOf*
trn' aid)
using *O γ tr tr' g* **by** (*intro Post1.nTriggerEq-open Post2.O-eq-nTriggerEq*)
auto
have *trn*: *Post1.nValidTrans trn* **and** *trn'*: *Post1.nValidTrans trn'* **using** *tr tr'*
by *auto*
show *Post1.nf' trn = Post1.nf' trn'*
proof (*cases trn*)
case (*LTrans s aid1 trn1*)
then obtain *s' trn1'* **where** *s'*: *trn' = LTrans s' aid1 trn1'* **using** *g* **by** (*cases*

trn') *auto*
moreover then have *srcOf trn1* = *s aid1 srcOf trn1'* = *s' aid1*
tgtOf trn1 = *Post1.nTgtOf trn aid1 tgtOf trn1'* = *Post1.nTgtOf trn' aid1*
using *LTrans trn trn'* **by** *auto*
ultimately show *?thesis* **using** *LTrans φ φ' op' g AID1-neq-AID2*
by (*cases trn1 rule*: *Post.φ.cases*; *cases trn1' rule*: *Post.f.cases*)
(*auto simp*: *Strong-ObservationSetup-RECEIVER.g.simps Strong-ObservationSetup-RECEIVER.comPur*
Post.φ.simps Post-RECEIVER.φ.simps)
next
case (*CTrans s aid1 trn1 aid2 trn2*)
then obtain *s' trn1' trn2'* **where** *CTrans'*: *trn'* = *CTrans s' aid1 trn1' aid2 trn2'*
using *g* **by** (*cases trn'*) *auto*
then have *trn1*: *validTrans trn1* **and** *trn1'*: *validTrans trn1'* **using** *trn trn' CTrans* **by** *auto*
have *states*: *tgtOf trn1* = *Post1.nTgtOf trn aid1 tgtOf trn2* = *Post1.nTgtOf trn aid2*
tgtOf trn1' = *Post1.nTgtOf trn' aid1 tgtOf trn2'* = *Post1.nTgtOf trn' aid2*
using *trn trn' CTrans CTrans'* **by** *auto*
have *Post1.sync aid1 trn1 aid2 trn2 Post1.sync aid1 trn1' aid2 trn2'*
using *CTrans CTrans' trn trn'* **by** *auto*
then show *?thesis* **using** *CTrans CTrans' op' g states AID1-neq-AID2*
by (*elim Post1.sync-cases*[*OF - trn1*] *Post1.sync-cases*[*OF - trn1'*])
(*auto simp*: *Post-RECEIVER.φ.simps Strong-ObservationSetup-RECEIVER.g.simps*
Strong-ObservationSetup-RECEIVER.comPurge.simps)
qed
next
fix *tr trn*
assume *nT-trn*: *Post2.Net.nT trn* **and** *tr*: *validSystemTrace* (*tr* ## *trn*)
and *nT-tr*: *never Post2.Net.nT tr*
show *Post1.Net.nγ trn* **proof** (*cases trn*)
case (*CTrans s aid1 trn1 aid2 trn2*)
then have *Post1.sync aid1 trn1 aid2 trn2* **using** *tr* **by** *auto*
then show *?thesis* **using** *tr CTrans*
by (*elim Post1.sync-cases*) (*auto simp*: *Strong-ObservationSetup-RECEIVER.γ.simps*)
next
case (*LTrans s aid1 trn1*)
then obtain *a ou s1'* **where** *trn1*: *trn1* = *Trans* (*s aid1*) *a ou s1'* **using** *tr* **by** (*cases trn1*) *auto*
interpret *R*: *Post-RECEIVER UIDs aid1 PID2 AID2* **.**
interpret *R'*: *Post-RECEIVER UIDs aid1 PID1 AID1* **.**
from *nT-trn* **have** *aid1*: *aid1* ≠ *AID2* **and** *Ttgt*: *R.T-state s1'*
using *LTrans R.T-T-state trn1* **by** *auto*
have *decomp-tr*: *Post1.Iss.validFrom istate* (*Post1.decomp* (*tr* ## *trn*) *aid1*)
using *LTrans tr Post1.validFrom-lValidFrom*[*of* λ-. *istate*] **by** *auto*
then have *s-aid1*: *s aid1* = *tgtOfTrFrom istate* (*Post1.decomp tr aid1*)
using *LTrans trn1* **unfolding** *Post1.decomp-append*

by (*auto simp*: *Post1.Iss.validFrom-Cons Post1.Iss.validFrom-append*)
have ¬*R.T-state* (*s aid1*) **unfolding** *s-aid1* **proof** (*intro R.never-T-not-T-state*)
show *Post1.Iss.validFrom istate* (*Post1.decomp tr aid1*) **using** *decomp-tr*
unfolding *Post1.decomp-append* **by** (*auto simp*: *Post1.Iss.validFrom-append*)
show *never R.T* (*Post1.decomp tr aid1*) **using** *aid1 Post2.Net.nTT-TT*[*OF nT-tr*, *of aid1*] **by** *auto*
show ¬ *R.T-state istate* **unfolding** *istate-def R.T-state-def* **by** *auto*
qed
then have *s-s1'*: ¬*triggerEq* (*s aid1*) *s1'* **using** *Ttgt* **by** (*auto simp*: *triggerEq-def R.T-state-def*)
show *?thesis* **proof** (*cases aid1* = *AID1*)
case *True*
then show *?thesis* **using** *s-s1' Post1.Iss.triggerEq-not-γ tr* **unfolding** *trn1 LTrans*
by (*cases Post1.Iss.γ* (*Trans* (*s aid1*) *a ou s1'*)) *auto*
next
case *False*
then show *?thesis* **using** *s-s1' R'.triggerEq-not-γ tr* **unfolding** *trn1 LTrans*
by (*cases R'.γ* (*Trans* (*s aid1*) *a ou s1'*)) *auto*
qed
qed
next
fix *tr tr' trn trn'*
assume *O*: *Post1.O tr* = *Post1.O tr'* **and** *γ*: *Post1.Net.nγ trn Post1.Net.nγ trn'*
and *tr*: *validSystemTrace* (*tr ## trn*) **and** *tr'*: *validSystemTrace* (*tr' ## trn'*)
and *g*: *Post1.Net.ng trn* = *Post1.Net.ng trn'*
have *op'*: *Post1.nTriggerEq* (*Post1.nTgtOf trn*) (*Post1.nTgtOf trn'*)
using *O γ tr tr' g* **by** (*intro Post1.O-eq-nTriggerEq*) *auto*
have *trn*: *Post1.nValidTrans trn* **and** *trn'*: *Post1.nValidTrans trn'* **using** *tr tr'* **by** *auto*
show *Post2.Net.nT trn* = *Post2.Net.nT trn'* **proof** (*cases trn*)
case (*LTrans s aid1 trn1*)
moreover then obtain *s' trn1'* **where** *LTrans'*: *trn'* = *LTrans s' aid1 trn1'*
using *g* **by** (*cases trn'*) *auto*
ultimately have *t*: *triggerEq* (*tgtOf trn1*) (*tgtOf trn1'*) **using** *op'* **unfolding** *Post1.nTriggerEq-def*
by *auto*
interpret *R*: *Post-RECEIVER UIDs aid1 PID2 AID2* **.**
from *t* **have** *R.T-state* (*tgtOf trn1*) ⟷ *R.T-state* (*tgtOf trn1'*) **by** (*intro R.triggerEq-T*)
then show *?thesis* **using** *LTrans LTrans'* **by** (*auto simp*: *R.T-T-state*)
next
case (*CTrans s aid1 trn1 aid2 trn2*)
moreover then obtain *s' trn1' trn2'* **where** *CTrans'*: *trn'* = *CTrans s' aid1 trn1' aid2 trn2'*
using *g* **by** (*cases trn'*) *auto*
moreover then have *aid1* ≠ *aid2* **using** *trn'* **by** *auto*
ultimately have *t*: *triggerEq* (*tgtOf trn1*) (*tgtOf trn1'*) *triggerEq* (*tgtOf trn2*)

(*tgtOf trn2'*)
using *op'* **unfolding** *Post1.nTriggerEq-def* **by** *auto*
interpret *R1*: *Post-RECEIVER UIDs aid1 PID2 AID2* **.**
interpret *R2*: *Post-RECEIVER UIDs aid2 PID2 AID2* **.**
from *t* **have** *R1.T-state* (*tgtOf trn1*) ⟷ *R1.T-state* (*tgtOf trn1'*)
R2.T-state (*tgtOf trn2*) ⟷ *R2.T-state* (*tgtOf trn2'*)
by (*auto intro*!: *R1.triggerEq-T R2.triggerEq-T*)
then show *?thesis* **using** *CTrans CTrans'* **by** (*auto simp*: *R1.T-T-state R2.T-T-state*)
qed
qed

theorem *two-posts-secure*:
secure
using *Post1.secure Post2.secure*
by (*rule two-secure*)

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*
by *auto*

lemma *eqButUIDl-sym*:
assumes *eqButUIDl uid uidl uidl1*
shows *eqButUIDl uid uidl1 uidl*
using *assms* **by** *auto*

lemma *eqButUIDl-trans*:
assumes *eqButUIDl uid uidl uidl1* **and** *eqButUIDl uid uidl1 uidl2*
shows *eqButUIDl uid uidl uidl2*
using *assms* **by** *auto*

lemma *eqButUIDl-remove1-cong*:
assumes *eqButUIDl uid uidl uidl1*
shows *eqButUIDl uid* (*remove1 uid′ uidl*) (*remove1 uid′ uidl1*)
proof −
have *remove1 uid* (*remove1 uid′ uidl*) = *remove1 uid′* (*remove1 uid uidl*) **by** (*simp add*: *remove1-commute*)
also have ... = *remove1 uid′* (*remove1 uid uidl1*) **using** *assms* **by** *simp*

also have ... = *remove1 uid* (*remove1 uid′ uidl1*) **by** (*simp add*: *remove1-commute*)
finally show *?thesis* **by** *simp*
qed

lemma *eqButUIDl-snoc-cong*:
assumes *eqButUIDl uid uidl uidl1*
and *uid′* ∈∈ *uidl* ⟷ *uid′* ∈∈ *uidl1*
shows *eqButUIDl uid* (*uidl* ## *uid′*) (*uidl1* ## *uid′*)
using *assms* **by** (*auto simp add*: *remove1-append remove1-idem*)

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*
unfolding *eqButUIDf-def* **by** *auto*

lemma *eqButUIDf-sym*:
assumes *eqButUIDf frds frds1* **shows** *eqButUIDf frds1 frds*
using *assms eqButUIDl-sym* **unfolding** *eqButUIDf-def*
by *presburger*

lemma *eqButUIDf-trans*:
assumes *eqButUIDf frds frds1* **and** *eqButUIDf frds1 frds2*
shows *eqButUIDf frds frds2*
using *assms eqButUIDl-trans* **unfolding** *eqButUIDf-def* **by** (*auto split*: *if-splits*)

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*))
using *assms* **unfolding** *eqButUIDf-def* **by** (*auto split*: *if-splits*)

lemma *eqButUIDf-eqButUIDl*:
assumes *eqButUIDf frds frds1*
shows *eqButUIDl UID2* (*frds UID1*) (*frds1 UID1*)
and *eqButUIDl UID1* (*frds UID2*) (*frds1 UID2*)
using *assms* **unfolding** *eqButUIDf-def* **by** (*auto split*: *if-splits*)

lemma *eqButUIDf-not-UID*:
⟦*eqButUIDf frds frds1*; *uid* ≠ *UID1*; *uid* ≠ *UID2*⟧ ⟹ *frds uid* = *frds1 uid*
unfolding *eqButUIDf-def* **by** (*auto split*: *if-splits*)

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 −
  from *uid* **have** (*uid'* = *UID1* ∧ *uid* ≠ *UID2*)
        ∨ (*uid'* = *UID2* ∧ *uid* ≠ *UID1*)
        ∨ (*uid'* ∉ {*UID1*,*UID2*}) (**is** *?u1* ∨ *?u2* ∨ *?n12*)
    by *auto*
  then show *?thesis* **proof** (*elim disjE*)
    assume *?u1*
    moreover then have *uid* ∈∈ *remove1 UID2* (*frds uid'*) ⟷ *uid* ∈∈ *remove1 UID2* (*frds1 uid'*)
      using *eq1* **unfolding** *eqButUIDf-def* **by** *auto*
    ultimately show *?thesis* **by** *auto*
  next
    assume *?u2*
    moreover then have *uid* ∈∈ *remove1 UID1* (*frds uid'*) ⟷ *uid* ∈∈ *remove1 UID1* (*frds1 uid'*)
      using *eq1* **unfolding** *eqButUIDf-def* **by** *auto*
    ultimately show *?thesis* **by** *auto*
  next
    assume *?n12*
    then show *?thesis* **using** *eq1* **unfolding** *eqButUIDf-def* **by** *auto*
  qed
qed

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*
unfolding *eqButUID12-def* **by** *auto*

lemma *eqButUID12-sym*:
assumes *eqButUID12 freq freq1* **shows** *eqButUID12 freq1 freq*
using *assms* **unfolding** *eqButUID12-def*
by *presburger*

lemma *eqButUID12-trans*:
assumes *eqButUID12 freq freq1* **and** *eqButUID12 freq1 freq2*
shows *eqButUID12 freq freq2*
using *assms* **unfolding** *eqButUID12-def* **by** (*auto split*: *if-splits*)

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*)
using *assms* **unfolding** *eqButUID12-def fun-upd2-def* **by** (*auto split*: *if-splits*)

lemma *eqButUID12-not-UID*:
⟦*eqButUID12 freq freq1*; ¬ (*uid*,*uid'*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)}⟧ ⟹ *freq uid uid'* = *freq1 uid uid'*
unfolding *eqButUID12-def* **by** (*auto split*: *if-splits*)

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*
unfolding *eqButUID-def* **by** *auto*

lemma *eqButUID-sym*[*sym*]:
assumes *eqButUID s s1* **shows** *eqButUID s1 s*

using *assms eqButUIDf-sym eqButUID12-sym* **unfolding** *eqButUID-def* **by** *auto*

lemma *eqButUID-trans*[*trans*]:
assumes *eqButUID s s1* **and** *eqButUID s1 s2* **shows** *eqButUID s s2*
using *assms eqButUIDf-trans eqButUID12-trans* **unfolding** *eqButUID-def* **by** *metis*

lemma *eqButUID-stateSelectors*:
assumes *eqButUID s s1*
shows *admin s = admin s1*
pendingUReqs s = pendingUReqs s1 userReq s = userReq s1
userIDs s = userIDs s1 user s = user s1 pass s = pass s1
eqButUIDf (pendingFReqs s) (pendingFReqs s1)
eqButUID12 (friendReq s) (friendReq s1)
eqButUIDf (friendIDs s) (friendIDs s1)

postIDs s = postIDs s1
post s = post s1 vis s = vis s1
owner s = owner s1

pendingSApiReqs s = pendingSApiReqs s1 sApiReq s = sApiReq s1
serverApiIDs s = serverApiIDs s1 serverPass s = serverPass s1
outerPostIDs s = outerPostIDs s1 outerPost s = outerPost s1 outerVis s = outer-Vis s1
outerOwner s = outerOwner s1
sentOuterFriendIDs s = sentOuterFriendIDs s1
recvOuterFriendIDs s = recvOuterFriendIDs s1

pendingCApiReqs s = pendingCApiReqs s1 cApiReq s = cApiReq s1
clientApiIDs s = clientApiIDs s1 clientPass s = clientPass s1
sharedWith s = sharedWith s1

IDsOK s = IDsOK s1
using *assms* **unfolding** *eqButUID-def IDsOK-def*[*abs-def*] **by** *auto*

lemma *eqButUID-eqButUID2*:
eqButUID s s1 ⟹ *eqButUIDl UID2 (friendIDs s UID1) (friendIDs s1 UID1)*
unfolding *eqButUID-def* **using** *eqButUIDf-eqButUIDl*
by (*smt eqButUIDf-eqButUIDl eqButUIDl.simps*)

lemma *eqButUID-not-UID*:
eqButUID s s1 ⟹ *uid* ≠ *UID* ⟹ *post s uid = post s1 uid*
unfolding *eqButUID-def* **by** *auto*

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*⦈)
unfolding *eqButUID-def* **by** *auto*

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 −
from *assms* **have** *?uid* ∧ *?uid'*
proof (*cases a*)
case (*Sact sa*) **then show** *?thesis* **using** *assms* **by** (*cases sa*) (*auto simp*: *s-defs*)
next
case (*Uact ua*) **then show** *?thesis* **using** *assms* **by** (*cases ua*) (*auto simp*: *u-defs*) **next**
case (*COMact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *com-defs*) **next**
case (*Cact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *c-defs*) **next**
case (*Dact da*) **then show** *?thesis* **using** *assms* **by** (*cases da*) (*auto simp*: *d-defs*)
qed *auto*
then show *?uid* **and** *?uid'* **by** *auto*
qed

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*
using *step-friendIDs*[*OF assms*] **unfolding** *friends12-def* **by** *auto*

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 −
from *assms* **have** *?uid* ∧ *?uid′*
proof (*cases a*)
case (*Sact sa*) **then show** *?thesis* **using** *assms* **by** (*cases sa*) (*auto simp*: *s-defs*)
next
case (*Uact ua*) **then show** *?thesis* **using** *assms* **by** (*cases ua*) (*auto simp*: *u-defs*) **next**
case (*COMact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *com-defs*) **next**
case (*Cact ca*) **then show** *?thesis* **using** *assms* **proof** (*cases ca*)
case (*cFriend uid1 p uid1′*)
then have ((*uid1* = *uid* ⟶ *uid1′* ≠ *uid′*) ∧ (*uid1* = *uid′* ⟶ *uid1′* ≠ *uid*)) ∨ *ou* = *outErr*
using *Cact assms* **by** (*auto simp*: *c-defs*)
then show *?thesis* **using** *step Cact cFriend* **by** (*auto simp*: *c-defs*)
qed (*auto simp*: *c-defs*) **next**
case (*Dact da*) **then show** *?thesis* **using** *assms* **by** (*cases da*) (*auto simp*: *d-defs*)
qed *auto*
then show *?uid* **and** *?uid′* **by** *auto*
qed

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 −
have *dfIDs*: *distinct* (*friendIDs s uid*) *distinct* (*friendIDs s1 uid*)
using *reach-distinct-friends-reqs*[*OF rs*] *reach-distinct-friends-reqs*[*OF rs1*] **by** *auto*
from *f12* **have** *uid12*: *UID1* ∈∈ *friendIDs s UID2* ⟷ *UID1* ∈∈ *friendIDs s1 UID2*
UID2 ∈∈ *friendIDs s UID1* ⟷ *UID2* ∈∈ *friendIDs s1 UID1*
using *reach-friendIDs-symmetric*[*OF rs*] *reach-friendIDs-symmetric*[*OF rs1*]
unfolding *friends12-def* **by** *auto*
from *ss1* **have** *fIDs*: *eqButUIDf* (*friendIDs s*) (*friendIDs s1*) **unfolding** *eqButUID-def* **by** *simp*
show *set* (*friendIDs s uid*) = *set* (*friendIDs s1 uid*)
proof (*intro equalityI subsetI*)

fix *uid′*
assume *uid′* ∈∈ *friendIDs s uid*
then show *uid′* ∈∈ *friendIDs s1 uid*
using *fIDs dfIDs uid12 eqButUIDf-not-UID′* **unfolding** *eqButUIDf-def*
by (*metis* (*no-types*, *lifting*) *insert-iff prod.inject singletonD*)
next
fix *uid′*
assume *uid′* ∈∈ *friendIDs s1 uid*
then show *uid′* ∈∈ *friendIDs s uid*
using *fIDs dfIDs uid12 eqButUIDf-not-UID′* **unfolding** *eqButUIDf-def*
by (*metis* (*no-types*, *lifting*) *insert-iff prod.inject singletonD*)
qed
qed

lemma *distinct-remove1-idem*: *distinct xs* ⟹ *remove1 y* (*remove1 y xs*) = *remove1 y xs*
by (*induction xs*) (*auto simp add*: *remove1-idem*)

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′*
using *assms* **proof** (*cases*)
assume *ou*: *ou* = *outOK*
then have *uid′* ∈∈ *pendingFReqs s uid* **using** *step* **by** (*auto simp add*: *c-defs*)
then have *fIDs*: *uid′* ∉ *set* (*friendIDs s uid*) *uid* ∉ *set* (*friendIDs s uid′*)
and *fRs*: *distinct* (*pendingFReqs s uid*) *distinct* (*pendingFReqs s uid′*)
using *reach-distinct-friends-reqs*[*OF s*] **by** *auto*
have *eqButUIDf* (*friendIDs s*) (*friendIDs* (*createFriend s uid p uid′*))
using *fIDs uids UID1-UID2* **unfolding** *eqButUIDf-def*
by (*cases ?u12*) (*auto simp add*: *c-defs remove1-idem remove1-append*)
moreover have *eqButUIDf* (*pendingFReqs s*) (*pendingFReqs* (*createFriend s uid p uid′*))
using *fRs uids UID1-UID2* **unfolding** *eqButUIDf-def*
by (*cases ?u12*) (*auto simp add*: *c-defs distinct-remove1-idem*)
moreover have *eqButUID12* (*friendReq s*) (*friendReq* (*createFriend s uid p uid′*))
using *uids* **unfolding** *eqButUID12-def*
by (*auto simp add*: *c-defs fun-upd2-eq-but-a-b*)
ultimately show *eqButUID s s′* **using** *step ou* **unfolding** *eqButUID-def* **by** (*auto simp add*: *c-defs*)
qed (*auto*)

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′*
using *assms* **proof** (*cases*)
  assume *ou*: *ou* = *outOK*
  then have *uid* ∉ *set* (*pendingFReqs s uid′*) *uid* ∉ *set* (*friendIDs s uid′*)
    using *step* **by** (*auto simp add*: *c-defs*)
  then have *eqButUIDf* (*pendingFReqs s*) (*pendingFReqs* (*createFriendReq s uid p uid′ req*))
    using *uids UID1-UID2* **unfolding** *eqButUIDf-def*
    by (*cases ?u12*) (*auto simp add*: *c-defs remove1-idem remove1-append*)
  moreover have *eqButUID12* (*friendReq s*) (*friendReq* (*createFriendReq s uid p uid′ req*))
    using *uids* **unfolding** *eqButUID12-def*
    by (*auto simp add*: *c-defs fun-upd2-eq-but-a-b*)
  ultimately show *eqButUID s s′* **using** *step ou* **unfolding** *eqButUID-def* **by** (*auto simp add*: *c-defs*)
qed (*auto*)

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′*
using *assms* **proof** (*cases*)
  assume *ou*: *ou* = *outOK*
  then have *uid′* ∈∈ *friendIDs s uid* **using** *step* **by** (*auto simp add*: *d-defs*)
  then have *fRs*: *distinct* (*friendIDs s uid*) *distinct* (*friendIDs s uid′*)
    using *reach-distinct-friends-reqs*[*OF s*] **by** *auto*
  have *eqButUIDf* (*friendIDs s*) (*friendIDs* (*deleteFriend s uid p uid′*))
    using *fRs uids UID1-UID2* **unfolding** *eqButUIDf-def*
    by (*cases ?u12*) (*auto simp add*: *d-defs remove1-idem distinct-remove1-removeAll*)
  then show *eqButUID s s′* **using** *step ou* **unfolding** *eqButUID-def* **by** (*auto simp add*: *d-defs*)
qed (*auto*)

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 −
  note *simps* = *eqButUID-stateSelectors s-defs c-defs u-defs r-defs l-defs com-defs*
  from *assms* **show** *?thesis* **proof** (*cases a*)
    case (*Sact sa*) **with** *assms* **show** *?thesis* **by** (*cases sa*) (*auto simp add*: *simps*)
  next

case (*Cact ca*) **note** *a* = *this*
with *assms* **show** *?thesis* **proof** (*cases ca*)
case (*cFriendReq uid p uid′ req*) **note** *ca* = *this*
then show *?thesis*
proof (*cases* (*uid* = *UID1* ∧ *uid′* = *UID2*) ∨ (*uid* = *UID2* ∧ *uid′* = *UID1*))
case *True*
then have *eqButUID s s′* **and** *eqButUID s1 s1′*
using *step step1* **unfolding** *a ca*
by (*auto intro*: *Cact-cFriendReq-step-eqButUID*)
with *ss1* **show** *eqButUID s′ s1′* **by** (*auto intro*: *eqButUID-sym eqButUID-trans*)
next
case *False*
have *fRs*: *eqButUIDf* (*pendingFReqs s*) (*pendingFReqs s1*)
and *fIDs*: *eqButUIDf* (*friendIDs s*) (*friendIDs s1*) **using** *ss1* **by** (*auto simp*: *simps*)
then have *uid-uid′*: *uid* ∈∈ *pendingFReqs s uid′* ⟷ *uid* ∈∈ *pendingFReqs s1 uid′*
uid ∈∈ *friendIDs s uid′* ⟷ *uid* ∈∈ *friendIDs s1 uid′*
using *False* **by** (*auto intro*!: *eqButUIDf-not-UID′*)
have *eqButUIDf* ((*pendingFReqs s*)(*uid′* := *pendingFReqs s uid′* ## *uid*))
((*pendingFReqs s1*)(*uid′* := *pendingFReqs s1 uid′* ## *uid*))
using *fRs False*
by (*intro eqButUIDf-cong*) (*auto simp add*: *remove1-append remove1-idem eqButUIDf-def*)
moreover have *eqButUID12* (*fun-upd2* (*friendReq s*) *uid uid′ req*)
(*fun-upd2* (*friendReq s1*) *uid uid′ req*)
using *ss1* **by** (*intro eqButUID12-cong*) (*auto simp*: *simps*)
moreover have *e-createFriendReq s uid p uid′ req*
⟷ *e-createFriendReq s1 uid p uid′ req*
using *uid-uid′ ss1* **by** (*auto simp*: *simps*)
ultimately show *?thesis* **using** *assms* **unfolding** *a ca* **by** (*auto simp*: *simps*)
qed
next
case (*cFriend uid p uid′*) **note** *ca* = *this*
then show *?thesis*
proof (*cases* (*uid* = *UID1* ∧ *uid′* = *UID2*) ∨ (*uid* = *UID2* ∧ *uid′* = *UID1*))
case *True*
then have *eqButUID s s′* **and** *eqButUID s1 s1′*
using *step step1 rs rs1* **unfolding** *a ca*
by (*auto intro*!: *Cact-cFriend-step-eqButUID*)+
with *ss1* **show** *eqButUID s′ s1′* **by** (*auto intro*: *eqButUID-sym eqButUID-trans*)
next
case *False*

have *fRs*: *eqButUIDf* (*pendingFReqs s*) (*pendingFReqs s1*)
(**is** *eqButUIDf* (*?pfr s*) (*?pfr s1*))
and *fIDs*: *eqButUIDf* (*friendIDs s*) (*friendIDs s1*) **using** *ss1* **by** (*auto simp*: *simps*)
then have *uid-uid'*: *uid* ∈∈ *pendingFReqs s uid'* ⟷ *uid* ∈∈ *pendingFReqs s1 uid'*
uid' ∈∈ *pendingFReqs s uid* ⟷ *uid'* ∈∈ *pendingFReqs s1 uid*
uid ∈∈ *friendIDs s uid'* ⟷ *uid* ∈∈ *friendIDs s1 uid'*
uid' ∈∈ *friendIDs s uid* ⟷ *uid'* ∈∈ *friendIDs s1 uid*
using *False* **by** (*auto intro*!: *eqButUIDf-not-UID'*)
have *eqButUIDl UID1* (*remove1 uid'* (*?pfr s UID2*)) (*remove1 uid'* (*?pfr s1 UID2*))
and *eqButUIDl UID2* (*remove1 uid'* (*?pfr s UID1*)) (*remove1 uid'* (*?pfr s1 UID1*))
and *eqButUIDl UID1* (*remove1 uid* (*?pfr s UID2*)) (*remove1 uid* (*?pfr s1 UID2*))
and *eqButUIDl UID2* (*remove1 uid* (*?pfr s UID1*)) (*remove1 uid* (*?pfr s1 UID1*))
using *fRs* **unfolding** *eqButUIDf-def*
by (*auto intro*!: *eqButUIDl-remove1-cong simp del*: *eqButUIDl.simps*)
then have *1*: *eqButUIDf* ((*?pfr s*)(*uid* := *remove1 uid'* (*?pfr s uid*),
uid' := *remove1 uid* (*?pfr s uid'*)))
((*?pfr s1*)(*uid* := *remove1 uid'* (*?pfr s1 uid*),
uid' := *remove1 uid* (*?pfr s1 uid'*)))
using *fRs False*
by (*intro eqButUIDf-cong*) (*auto simp add*: *eqButUIDf-def*)
have *uid* = *UID1* ⟹ *eqButUIDl UID2* (*friendIDs s UID1* ## *uid'*) (*friendIDs s1 UID1* ## *uid'*)
and *uid* = *UID2* ⟹ *eqButUIDl UID1* (*friendIDs s UID2* ## *uid'*) (*friendIDs s1 UID2* ## *uid'*)
and *uid'* = *UID1* ⟹ *eqButUIDl UID2* (*friendIDs s UID1* ## *uid*) (*friendIDs s1 UID1* ## *uid*)
and *uid'* = *UID2* ⟹ *eqButUIDl UID1* (*friendIDs s UID2* ## *uid*) (*friendIDs s1 UID2* ## *uid*)
using *fIDs uid-uid'* **by** − (*intro eqButUIDl-snoc-cong*; *simp add*: *eqButUIDf-def*)+
then have *2*: *eqButUIDf* ((*friendIDs s*)(*uid* := *friendIDs s uid* ## *uid'*,
uid' := *friendIDs s uid'* ## *uid*))
((*friendIDs s1*)(*uid* := *friendIDs s1 uid* ## *uid'*,
uid' := *friendIDs s1 uid'* ## *uid*))
using *fIDs* **by** (*intro eqButUIDf-cong*) (*auto simp add*: *eqButUIDf-def*)
have *3*: *eqButUID12* (*fun-upd2* (*fun-upd2* (*friendReq s*) *uid' uid emptyRequestInfo*)
uid uid' emptyRequestInfo)
(*fun-upd2* (*fun-upd2* (*friendReq s1*) *uid' uid emptyRequestInfo*)
uid uid' emptyRequestInfo)

```
            using ss1 by (intro eqButUID12-cong) (auto simp: simps)
          have e-createFriend s uid p uid'
            ⟷ e-createFriend s1 uid p uid'
            using uid-uid' ss1 by (auto simp: simps)
          with 1 2 3 show ?thesis using assms unfolding a ca by (auto simp:
simps)
      qed
    qed (auto simp: simps)
  next
    case (Uact ua) with assms show ?thesis by (cases ua) (auto simp add: simps)
  next
    case (Ract ra) with assms show ?thesis by (cases ra) (auto simp add: simps)
  next
    case (Lact la) with assms show ?thesis by (cases la) (auto simp add: simps)
  next
    case (COMact ca) with assms show ?thesis by (cases ca) (auto simp add:
simps)
  next
    case (Dact da) note a = this
      with assms show ?thesis proof (cases da)
        case (dFriend uid p uid') note ca = this
          then show ?thesis
          proof (cases (uid = UID1 ∧ uid' = UID2) ∨ (uid = UID2 ∧ uid' =
UID1))
            case True
              then have eqButUID s s' and eqButUID s1 s1'
                using step step1 rs rs1 unfolding a ca
                by (auto intro!: Dact-dFriend-step-eqButUID)+
                  with ss1 show eqButUID s' s1' by (auto intro: eqButUID-sym
eqButUID-trans)
          next
            case False
             have fIDs: eqButUIDf (friendIDs s) (friendIDs s1) using ss1 by (auto
simp: simps)
             then have uid-uid': uid ∈∈ friendIDs s uid' ⟷ uid ∈∈ friendIDs s1
uid'
                                uid' ∈∈ friendIDs s uid ⟷ uid' ∈∈ friendIDs s1 uid
               using False by (auto intro!: eqButUIDf-not-UID')
             have dfIDs: distinct (friendIDs s uid) distinct (friendIDs s uid')
                        distinct (friendIDs s1 uid) distinct (friendIDs s1 uid')
              using reach-distinct-friends-reqs[OF rs] reach-distinct-friends-reqs[OF
rs1] by auto
               have uid = UID1 ⟹ eqButUIDl UID2 (remove1 uid' (friendIDs s
UID1)) (remove1 uid' (friendIDs s1 UID1))
                and uid = UID2 ⟹ eqButUIDl UID1 (remove1 uid' (friendIDs s
UID2)) (remove1 uid' (friendIDs s1 UID2))
                and uid' = UID1 ⟹ eqButUIDl UID2 (remove1 uid (friendIDs s
UID1)) (remove1 uid (friendIDs s1 UID1))
                and uid' = UID2 ⟹ eqButUIDl UID1 (remove1 uid (friendIDs s
```

UID2)) (*remove1 uid* (*friendIDs s1 UID2*))
using *fIDs uid-uid'* **by** − (*intro eqButUIDl-remove1-cong*; *simp add*: *eqButUIDf-def*)+
then have *1*: *eqButUIDf* ((*friendIDs s*)(*uid* := *remove1 uid'* (*friendIDs s uid*),
uid' := *remove1 uid* (*friendIDs s uid'*)))
((*friendIDs s1*)(*uid* := *remove1 uid'* (*friendIDs s1 uid*),
uid' := *remove1 uid* (*friendIDs s1 uid'*)))
using *fIDs* **by** (*intro eqButUIDf-cong*) (*auto simp add*: *eqButUIDf-def*)
have *e-deleteFriend s uid p uid'*
⟷ *e-deleteFriend s1 uid p uid'*
using *uid-uid' ss1* **by** (*auto simp*: *simps d-defs*)
with *1* **show** *?thesis* **using** *assms dfIDs* **unfolding** *a ca*
by (*auto simp*: *simps d-defs distinct-remove1-removeAll*)
qed
qed
qed
qed

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'*
using *assms* **proof** (*cases a*)
case (*Sact sa*) **then show** *?thesis* **using** *assms* **by** (*cases sa*) (*auto simp*: *s-defs*)
next
case (*Uact ua*) **then show** *?thesis* **using** *assms* **by** (*cases ua*) (*auto simp*: *u-defs*)
next
case (*COMact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *com-defs*) **next**
case (*Dact da*) **then show** *?thesis* **using** *assms* **proof** (*cases da*)
case (*dFriend uid p uid'*)
with *Dact assms* **show** *?thesis*
by (*cases* (*uid*,*uid'*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)})
(*auto simp*: *d-defs eqButUID-stateSelectors eqButUIDf-not-UID'*)
qed
next
case (*Cact ca*) **then show** *?thesis* **using** *assms* **proof** (*cases ca*)
case (*cFriend uid p uid'*)
{ **assume** *p* = *pass s uid*
then have *uid'* ∈∈ *pendingFReqs s uid* ⟷ *uid'* ∈∈ *pendingFReqs s1 uid*
using *Cact cFriend ss1 a* **by** (*intro eqButUIDf-not-UID'*) (*auto simp*:

eqButUID-stateSelectors)
      }
      with *Cact cFriend assms* **show** *?thesis*
        by (*auto simp*: *c-defs eqButUID-stateSelectors*)
    qed (*auto simp*: *c-defs*)
qed *auto*

lemma *createFriend-sym*: *createFriend s uid p uid′* = *createFriend s uid′ p′ uid*
unfolding *c-defs* **by** (*cases uid* = *uid′*) (*auto simp*: *fun-upd2-comm fun-upd-twist*)

lemma *deleteFriend-sym*: *deleteFriend s uid p uid′* = *deleteFriend s uid′ p′ uid*
unfolding *d-defs* **by** (*cases uid* = *uid′*) (*auto simp*: *fun-upd-twist*)

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′*
using *assms* **unfolding** *c-defs* **by** (*auto simp*: *remove1-idem remove1-append fun-upd2-absorb*)

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 −
  have *distinct* (*friendIDs s UID1*) *distinct* (*friendIDs s UID2*)
       distinct (*friendIDs s1 UID1*) *distinct* (*friendIDs s1 UID2*)
    using *rs rs1* **by** (*auto intro*: *reach-distinct-friends-reqs*)
  then show *?thesis*
    using *ss1* **unfolding** *eqButUID-def eqButUIDf-def* **unfolding** *d-defs*
    by (*auto simp*: *distinct-remove1-removeAll*)
qed

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 −
  have *f12′*: *UID1* ∉ *set* (*friendIDs s UID2*) *UID2* ∉ *set* (*friendIDs s UID1*)
           UID1 ∉ *set* (*friendIDs s1 UID2*) *UID2* ∉ *set* (*friendIDs s1 UID1*)
    using *f12 rs rs1 reach-friendIDs-symmetric* **unfolding** *friends12-def* **by** *auto*
  have *friendIDs s* = *friendIDs s1*
  proof (*intro ext*)
    fix *uid*
    show *friendIDs s uid* = *friendIDs s1 uid*
      using *ss1 f12′* **unfolding** *eqButUID-def eqButUIDf-def*
      by (*cases uid* = *UID1* ∨ *uid* = *UID2*) (*auto simp*: *remove1-idem*)

qed
then show *?thesis* **by** (*auto simp*: *c-defs*)
qed

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'*
using *assms* **proof** (*cases a*)
case (*Dact da*) **then show** *?thesis* **using** *assms* **by** (*cases da*) (*auto simp*: *d-defs openByA-def*) **next**
case (*Uact ua*) **then show** *?thesis* **using** *assms* **by** (*cases ua*) (*auto simp*: *u-defs openByA-def*) **next**
case (*COMact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *com-defs openByA-def*) **next**
case (*Sact sa*)

then show *?thesis* **using** *assms UID1-UID2* **by** (*cases sa*) (*auto simp*: *s-defs openByA-def*) **next**
case (*Cact ca*)
then show *?thesis* **using** *assms UID1-UID2* **proof** (*cases ca*)
case (*cUser uid p uid′ p′*)
then show *?thesis* **using** *Cact assms* **by** (*intro that*) (*auto simp*: *c-defs openByA-def*)
qed (*auto simp*: *c-defs openByA-def*)
qed *auto*

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′*
using *assms* **proof** (*cases a*)
case (*Uact ua*) **then show** *?thesis* **using** *assms* **by** (*cases ua*) (*auto simp*: *u-defs openByF-def*) **next**
case (*COMact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *com-defs openByF-def*) **next**
case (*Sact sa*)
then show *?thesis* **using** *assms UID1-UID2* **by** (*cases sa*) (*auto simp*: *s-defs openByF-def*)
next
case (*Cact ca*)
then show *?thesis* **using** *assms UID1-UID2* **proof** (*cases ca*)
case (*cFriend uid p uid′*)
then show *?thesis* **using** *Cact assms* **by** (*intro OpenF*) (*auto simp*: *c-defs openByF-def*)
qed (*auto simp*: *c-defs openByF-def*)
next
case (*Dact da*)
then show *?thesis* **using** *assms* **proof** (*cases da*)
case (*dFriend uid p uid′*)
then show *?thesis* **using** *Dact assms* **by** (*intro CloseF*) (*auto simp*: *d-defs openByF-def*)
qed
qed *auto*

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 −
from *op* **have** *openByF s* ≠ *openByF s'* ∨ *openByA s* ≠ *openByA s'*
unfolding *open-def* **by** *auto*
then show *thesis* **proof**
assume *openByF s* ≠ *openByF s'*
with *step* **show** *thesis* **proof** (*cases rule*: *step-openByF-cases*)
case (*OpenF uid p uid'*)
then have *openByA s* = *openByA s'* **using** *step*
by (*cases openByA s* ≠ *openByA s'*, *elim step-openByA-cases*) *auto*
then have ¬*openByA s* ∧ ¬*openByA s'* **using** *op* **unfolding** *open-def* **by** *auto*
with *OpenF* **show** *thesis* **by** (*intro that*(*2*)) *auto*
next
case (*CloseF uid p uid'*)
then have *openByA s* = *openByA s'* **using** *step*
by (*cases openByA s* ≠ *openByA s'*, *elim step-openByA-cases*) *auto*
then have ¬*openByA s* ∧ ¬*openByA s'* **using** *op* **unfolding** *open-def* **by** *auto*
with *CloseF* **show** *thesis* **by** (*intro that*(*3*)) *auto*
qed
next
assume *openByA s* ≠ *openByA s'*
with *step* **show** *thesis* **proof** (*cases rule*: *step-openByA-cases*)
case (*CloseA uid p uid' p'*)
then have *openByF s* = *openByF s'* **using** *step*
by (*cases openByF s* ≠ *openByF s'*, *elim step-openByF-cases*) *auto*
then have ¬*openByF s* ∧ ¬*openByF s'* **using** *op* **unfolding** *open-def* **by** *auto*
with *CloseA* **show** *thesis* **by** (*intro that*(*1*)) *auto*
qed
qed
qed

lemma *eqButUID-openByA-eq*:
assumes *eqButUID s s1*
shows *openByA s = openByA s1*
using *assms* **unfolding** *openByA-def eqButUID-def* **by** *auto*

lemma *eqButUID-openByF-eq*:
assumes *ss1*: *eqButUID s s1*
shows *openByF s = openByF s1*
proof −
  from *ss1* **have** *fIDs*: *eqButUIDf* (*friendIDs s*) (*friendIDs s1*) **unfolding** *eqButUID-def* **by** *auto*
  have ∀ *uid* ∈ *UIDs. uid* ∈∈ *friendIDs s UID1* ⟷ *uid* ∈∈ *friendIDs s1 UID1*
    using *UID1-UID2-UIDs UID1-UID2* **by** (*intro ballI eqButUIDf-not-UID*′[*OF fIDs*]; *auto*)
  moreover have ∀ *uid* ∈ *UIDs. uid* ∈∈ *friendIDs s UID2* ⟷ *uid* ∈∈ *friendIDs s1 UID2*
    using *UID1-UID2-UIDs UID1-UID2* **by** (*intro ballI eqButUIDf-not-UID*′[*OF fIDs*]; *auto*)
  ultimately show *openByF s = openByF s1* **unfolding** *openByF-def* **by** *auto*
qed

lemma *eqButUID-open-eq*: *eqButUID s s1* ⟹ *open s = open s1*
using *eqButUID-openByA-eq eqButUID-openByF-eq* **unfolding** *open-def* **by** *blast*

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 −
  obtain *uid sa com-act* **where** *uid-a*: (*userOfA a = Some uid* ∧ *uid* ∈ *UIDs* ∧ *uid* ≠ *UID1* ∧ *uid* ≠ *UID2*)
                ∨ *a = COMact com-act* ∨ *a = Sact sa*
    using *γ UID1-UID2-UIDs* **by** *fastforce*
  { fix *uid*
    assume *uid* ∈∈ *friendIDs s UID1* ∨ *uid* ∈∈ *friendIDs s UID2* **and** *uid* ∈ *UIDs*
    with *os* **have** *friendIDs s = friendIDs s1* **unfolding** *open-def openByF-def* **by** *auto*
  } **note** *fIDs = this*
  { fix *uid uid*′
    assume *uid*: *uid* ≠ *UID1 uid* ≠ *UID2*
    have *friendIDs s uid = friendIDs s1 uid* (**is** *?f-eq*)
     and *pendingFReqs s uid = pendingFReqs s1 uid* (**is** *?pFR-eq*)
     and *uid* ∈∈ *friendIDs s uid*′ ⟷ *uid* ∈∈ *friendIDs s1 uid*′ (**is** *?f-iff*)

and *uid* ∈∈ *pendingFReqs s uid'* ⟷ *uid* ∈∈ *pendingFReqs s1 uid'* (**is** *?pFR-iff*)
and *friendReq s uid uid'* = *friendReq s1 uid uid'* (**is** *?FR-eq*)
and *friendReq s uid' uid* = *friendReq s1 uid' uid* (**is** *?FR-eq'*)
proof −
show *?f-eq ?pFR-eq* **using** *uid ss1 UID1-UID2-UIDs* **unfolding** *eqButUID-def*
by (*auto intro*!: *eqButUIDf-not-UID*)
show *?f-iff ?pFR-iff* **using** *uid ss1 UID1-UID2-UIDs* **unfolding** *eqButUID-def*
by (*auto intro*!: *eqButUIDf-not-UID'*)
from *uid* **have** ¬ (*uid*,*uid'*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)} **by** *auto*
then show *?FR-eq ?FR-eq'* **using** *ss1 UID1-UID2-UIDs* **unfolding** *eqButUID-def*
by (*auto intro*!: *eqButUID12-not-UID*)
qed
} **note** *simps* = *this eqButUID-stateSelectors r-defs s-defs c-defs com-defs l-defs u-defs d-defs*
note *facts* = *ss1 step step1 uid-a*
show *?thesis*
proof (*cases a*)
case (*Ract ra*) **then show** *?thesis* **using** *facts* **by** (*cases ra*) (*auto simp add*: *simps*)
next
case (*Sact sa*) **then show** *?thesis* **using** *facts* **by** (*cases sa*) (*auto simp add*: *simps*)
next
case (*Cact ca*) **then show** *?thesis* **using** *facts* **by** (*cases ca*) (*auto simp add*: *simps*)
next
case (*COMact ca*) **then show** *?thesis* **using** *facts* **by** (*cases ca*) (*auto simp add*: *simps*)
next
case (*Lact la*)
then show *?thesis* **using** *facts* **proof** (*cases la*)
case (*lFriends uid p uid'*)
with γ **have** *uid*: *uid* ∈ *UIDs* **using** *Lact* **by** *auto*
then have *uid-uid'*: *uid* ∈∈ *friendIDs s uid'* ⟷ *uid* ∈∈ *friendIDs s1 uid'*
using *ss1 UID1-UID2-UIDs* **unfolding** *eqButUID-def* **by** (*intro eqButUIDf-not-UID'*) *auto*
show *?thesis*
proof (*cases* (*uid'* = *UID1* ∨ *uid'* = *UID2*) ∧ *uid* ∈∈ *friendIDs s uid'*)
case *True*
with *uid* **have** *friendIDs s* = *friendIDs s1* **by** (*intro fIDs*) *auto*
then show *?thesis* **using** *lFriends facts Lact* **by** (*auto simp*: *simps*)
next
case *False*
then show *?thesis* **using** *lFriends facts Lact simps*(*1*) *uid-uid'* **by** (*auto simp*: *simps*)
qed
next
case (*lInnerPosts uid p*)

then have *o*: $\bigwedge$*nid. owner s nid = owner s1 nid*
and *n*: $\bigwedge$*nid. post s nid = post s1 nid*
and *nids*: *postIDs s = postIDs s1*
and *vis*: *vis s = vis s1*
and *fu*: $\bigwedge$*uid′. uid* ∈∈ *friendIDs s uid′* ⟷ *uid* ∈∈ *friendIDs s1 uid′*
and *e*: *e-listInnerPosts s uid p* ⟷ *e-listInnerPosts s1 uid p*
using *ss1 uid-a Lact* **unfolding** *eqButUID-def l-defs* **by** (*auto simp add*: *simps*(*3*))
have *listInnerPosts s uid p = listInnerPosts s1 uid p*
unfolding *listInnerPosts-def o n nids vis fu* **..**
with *e* **show** *?thesis* **using** *Lact lInnerPosts step step1* **by** *auto*
qed (*auto simp add*: *simps*)
next
case (*Uact ua*) **then show** *?thesis* **using** *facts* **by** (*cases ua*) (*auto simp add*: *simps*)
next
case (*Dact da*) **then show** *?thesis* **using** *facts* **by** (*cases da*) (*auto simp add*: *simps*)
qed
qed

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 φ*E*:
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*
using φ **proof** (*elim* φ*.elims disjE conjE*)

fix *s1 uid p uid′ ou1 s1′*
assume (*uid*,*uid′*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)} **and** *ou*: *ou1* = *outOK*
and *?trn* = *Trans s1* (*Cact* (*cFriend uid p uid′*)) *ou1 s1′*
then have *trn*: *a* = *Cact* (*cFriend uid p uid′*) *s* = *s1 s′* = *s1′ ou* = *ou1*
and *uids*: *uid* = *UID1* ∧ *uid′* = *UID2* ∨ *uid* = *UID2* ∧ *uid′* = *UID1* **using** *UID1-UID2* **by** *auto*
then show *thesis* **using** *ou uids trn step UID1-UID2-UIDs UID1-UID2 reach-distinct-friends-reqs*[*OF rs*]
by (*intro Friend*[*of uid p uid′*]) (*auto simp add*: *c-defs friends12-def*)
next
fix *s1 uid p uid′ ou1 s1′*
assume *op*: *open s1* ≠ *open s1′*
and *?trn* = *Trans s1* (*Cact* (*cFriend uid p uid′*)) *ou1 s1′*
then have *trn*: *open s* ≠ *open s′ s* = *s1 s′* = *s1′ ou* = *ou1*
and *a*: *a* = *Cact* (*cFriend uid p uid′*)
by *auto*
with *step* **have** *uids*: (*uid* ∈ *UIDs* ∧ *uid′* ∈ {*UID1*, *UID2*} ∨ *uid* ∈ {*UID1*, *UID2*} ∧ *uid′* ∈ *UIDs*) ∧
ou = *outOK* ∧ ¬*openByF s* ∧ *openByF s′* ∧ ¬*openByA s* ∧ ¬*openByA s′*
by (*cases rule*: *step-open-cases*) *auto*
then show *thesis* **using** *a UID1-UID2-UIDs* **by** (*intro OpenF*) *auto*
next
fix *s1 uid p uid′ ou1 s1′*
assume (*uid*,*uid′*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)} **and** *ou*: *ou1* = *outOK*
and *?trn* = *Trans s1* (*Dact* (*dFriend uid p uid′*)) *ou1 s1′*
then have *trn*: *a* = *Dact* (*dFriend uid p uid′*) *s* = *s1 s′* = *s1′ ou* = *ou1*
and *uids*: *uid* = *UID1* ∧ *uid′* = *UID2* ∨ *uid* = *UID2* ∧ *uid′* = *UID1* **using** *UID1-UID2* **by** *auto*
then show *thesis* **using** *step ou reach-friendIDs-symmetric*[*OF rs*]
by (*intro Unfriend*) (*auto simp*: *d-defs friends12-def*)
next
fix *s1 uid p uid′ ou1 s1′*
assume *op*: *open s1* ≠ *open s1′*
and *?trn* = *Trans s1* (*Dact* (*dFriend uid p uid′*)) *ou1 s1′*
then have *trn*: *open s* ≠ *open s′ s* = *s1 s′* = *s1′ ou* = *ou1*
and *a*: *a* = *Dact* (*dFriend uid p uid′*)
by *auto*
with *step* **have** *uids*: (*uid* ∈ *UIDs* ∧ *uid′* ∈ {*UID1*, *UID2*} ∨ *uid* ∈ {*UID1*, *UID2*} ∧ *uid′* ∈ *UIDs*) ∧
ou = *outOK* ∧ *openByF s* ∧ (¬*openByF s′*) ∧ (¬*openByA s*) ∧ (¬*openByA s′*)
by (*cases rule*: *step-open-cases*) *auto*
then show *thesis* **using** *a UID1-UID2-UIDs* **by** (*intro CloseF*) *auto*
next
fix *s1 uid p uid′ p′ ou1 s1′*
assume *op*: *open s1* ≠ *open s1′*
and *?trn* = *Trans s1* (*Cact* (*cUser uid p uid′ p′*)) *ou1 s1′*
then have *trn*: *open s* ≠ *open s′ s* = *s1 s′* = *s1′ ou* = *ou1*

and *a*: $a = Cact\ (cUser\ uid\ p\ uid'\ p')$
by *auto*
with *step* **have** *uids*: $(uid' = UID2 \lor uid' = UID1) \land ou = outOK \land$
$(\neg openByF\ s) \land (\neg openByF\ s') \land openByA\ s \land (\neg openByA\ s')$
by (*cases rule*: *step-open-cases*) *auto*
then show *thesis* **using** *a UID1-UID2-UIDs* **by** (*intro CloseA*) *auto*
qed

lemma *step-open-φ*:
assumes $step\ s\ a = (ou,\ s')$
and $open\ s \neq open\ s'$
shows $\varphi\ (Trans\ s\ a\ ou\ s')$
using *assms* **by** (*cases rule*: *step-open-cases*) (*auto simp*: *open-def*)

lemma *step-friends12-φ*:
assumes $step\ s\ a = (ou,\ s')$
and $friends12\ s \neq friends12\ s'$
shows $\varphi\ (Trans\ s\ a\ ou\ s')$
proof –
have $a = Cact\ (cFriend\ UID1\ (pass\ s\ UID1)\ UID2) \lor a = Cact\ (cFriend\ UID2\ (pass\ s\ UID2)\ UID1) \lor$
$a = Dact\ (dFriend\ UID1\ (pass\ s\ UID1)\ UID2) \lor a = Dact\ (dFriend\ UID2\ (pass\ s\ UID2)\ UID1)$
using *assms step-friends12* **by** (*cases* $ou = outOK$) *auto*
moreover then have $ou = outOK$ **using** *assms* **by** *auto*
ultimately show $\varphi\ (Trans\ s\ a\ ou\ s')$ **by** *auto*
qed

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*: $a \neq Cact\ (cFriend\ UID1\ (pass\ s\ UID1)\ UID2) \land a \neq Cact\ (cFriend\ UID2\ (pass\ s\ UID2)\ UID1) \land$
$a \neq Dact\ (dFriend\ UID1\ (pass\ s\ UID1)\ UID2) \land a \neq Dact\ (dFriend\ UID2\ (pass\ s\ UID2)\ UID1)$
and *φ*: $\varphi\ (Trans\ s\ a\ ou\ s')$
shows $\varphi\ (Trans\ s1\ a\ ou1\ s1')$
proof –
have $eqButUID\ s'\ s1'$ **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*] **.**
then have $open\ s = open\ s1$ **and** $open\ s' = open\ s1'$
and $openByA\ s = openByA\ s1$ **and** $openByA\ s' = openByA\ s1'$
and $openByF\ s = openByF\ s1$ **and** $openByF\ s' = openByF\ s1'$
using *ss1* **by** (*auto simp*: *eqButUID-open-eq eqButUID-openByA-eq eqButUID-openByF-eq*)
with *φ a step step1* **show** $\varphi\ (Trans\ s1\ a\ ou1\ s1')$ **using** *UID1-UID2-UIDs*
by (*elim φ.elims*) (*auto simp*: *c-defs d-defs*)
qed

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
assume φ (*Trans s a ou s'*)
with *assms* **show** φ (*Trans s1 a ou1 s1'*) **by** (*rule eqButUID-step-φ-imp*)
next
assume φ (*Trans s1 a ou1 s1'*)
moreover have *eqButUID s1 s* **using** *ss1* **by** (*rule eqButUID-sym*)
moreover have *a* ≠ *Cact* (*cFriend UID1* (*pass s1 UID1*) *UID2*) ∧
a ≠ *Cact* (*cFriend UID2* (*pass s1 UID2*) *UID1*) ∧
a ≠ *Dact* (*dFriend UID1* (*pass s1 UID1*) *UID2*) ∧
a ≠ *Dact* (*dFriend UID2* (*pass s1 UID2*) *UID1*)
using *a ss1* **by** (*auto simp*: *eqButUID-stateSelectors*)
ultimately show φ (*Trans s a ou s'*) **using** *rs rs1 step step1*
by (*intro eqButUID-step-φ-imp*[*of s1 s*])
qed

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* ⇒ *value list* ⇒ *bool*
where
BO-FrVal[*simp*,*intro*!]:
BO (*map FrVal fs*) (*map FrVal fs*)
|*BO-BC*[*intro*]:
BC vl vl1 ⟹
BO (*map FrVal fs* @ *OVal False* # *vl*) (*map FrVal fs* @ *OVal False* # *vl1*)

|*BC-FrVal*[*simp*,*intro*!]:
BC (*map FrVal fs*) (*map FrVal fs1*)
|*BC-BO*[*intro*]:
BO vl vl1 ⟹ (*fs* = [] ⟷ *fs1* = []) ⟹ (*fs* ≠ [] ⟹ *last fs* = *last fs1*) ⟹
BC (*map FrVal fs* @ *OVal True* # *vl*)
(*map FrVal fs1* @ *OVal True* # *vl1*)

definition *B vl vl1* ≡ *BO vl vl1* ∧ *alternatingFriends vl1 False*

lemma *BO-Nil-Nil*: *BO vl vl1* ⟹ *vl* = [] ⟹ *vl1* = []
by (*cases rule*: *BO.cases*) *auto*

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** $B = B$
done

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* ≠ [] **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 *cases*
assume *UID1* ∈∈ *pendingFReqs s UID2* ∨ *UID2* ∈∈ *pendingFReqs s UID1*

then show *thesis* **proof**
assume *pFR*: *UID1* ∈∈ *pendingFReqs s UID2*
let *?a* = *Cact* (*cFriend UID2* (*pass s UID2*) *UID1*)
let *?s'* = *createFriend s UID1* (*pass s UID1*) *UID2*
let *?trn* = *Trans s ?a outOK ?s'*
have *step*: *step s ?a* = (*outOK*, *?s'*) **using** *IDs pFR UID1-UID2*
unfolding *createFriend-sym*[*of s UID1 pass s UID1 UID2 pass s UID2*]
by (*auto simp add*: *c-defs*)
moreover then have *φ ?trn* **and** *f ?trn* = *FrVal True* **and** *friends12 ?s'*
by (*auto simp*: *c-defs friends12-def*)
moreover have ¬*γ ?trn* **using** *UID1-UID2-UIDs* **by** *auto*
ultimately show *thesis* **using** *nf12 rs*
by (*intro that*[*of* [*?a*] [*outOK*]]) (*auto intro*: *Cact-cFriend-step-eqButUID*)
next
assume *pFR*: *UID2* ∈∈ *pendingFReqs s UID1*
let *?a* = *Cact* (*cFriend UID1* (*pass s UID1*) *UID2*)
let *?s'* = *createFriend s UID1* (*pass s UID1*) *UID2*
let *?trn* = *Trans s ?a outOK ?s'*
have *step*: *step s ?a* = (*outOK*, *?s'*) **using** *IDs pFR UID1-UID2* **by** (*auto simp add*: *c-defs*)
moreover then have *φ ?trn* **and** *f ?trn* = *FrVal True* **and** *friends12 ?s'*
by (*auto simp*: *c-defs friends12-def*)
moreover have ¬*γ ?trn* **using** *UID1-UID2-UIDs* **by** *auto*
ultimately show *thesis* **using** *nf12 rs*
by (*intro that*[*of* [*?a*] [*outOK*]]) (*auto intro*: *Cact-cFriend-step-eqButUID*)
qed
next
assume *pFR*: ¬(*UID1* ∈∈ *pendingFReqs s UID2* ∨ *UID2* ∈∈ *pendingFReqs s UID1*)
let *?a1* = *Cact* (*cFriendReq UID2* (*pass s UID2*) *UID1 emptyRequestInfo*)
let *?s1* = *createFriendReq s UID2* (*pass s UID2*) *UID1 emptyRequestInfo*
let *?trn1* = *Trans s ?a1 outOK ?s1*
let *?a2* = *Cact* (*cFriend UID1* (*pass ?s1 UID1*) *UID2*)
let *?s2* = *createFriend ?s1 UID1* (*pass ?s1 UID1*) *UID2*
let *?trn2* = *Trans ?s1 ?a2 outOK ?s2*
have *eFR*: *e-createFriendReq s UID2* (*pass s UID2*) *UID1 emptyRequestInfo* **using** *IDs pFR nf12*
using *reach-friendIDs-symmetric*[*OF rs*]
by (*auto simp add*: *c-defs friends12-def*)
then have *step1*: *step s ?a1* = (*outOK*, *?s1*) **by** *auto*
moreover then have ¬*φ ?trn1* **and** ¬*γ ?trn1* **using** *UID1-UID2-UIDs* **by** *auto*
moreover have *eqButUID s ?s1* **by** (*intro Cact-cFriendReq-step-eqButUID*[*OF step1*]) *auto*
moreover have *rs1*: *reach ?s1* **using** *step1* **by** (*intro reach-PairI*[*OF rs*])
moreover have *step2*: *step ?s1 ?a2* = (*outOK*, *?s2*) **using** *IDs* **by** (*auto simp*: *c-defs*)
moreover then have *φ ?trn2* **and** *f ?trn2* = *FrVal True* **and** *friends12 ?s2*
by (*auto simp*: *c-defs friends12-def*)
moreover have ¬*γ ?trn2* **using** *UID1-UID2-UIDs* **by** *auto*

moreover have *eqButUID ?s1 ?s2* **by** (*intro Cact-cFriend-step-eqButUID*[*OF step2 rs1*]) *auto*
moreover have *?s2* = *createFriend s UID1* (*pass s UID1*) *UID2*
using *eFR* **by** (*intro createFriendReq-createFriend-absorb*)
ultimately show *thesis* **using** *nf12 rs*
by (*intro that*[*of* [*?a1*, *?a2*] [*outOK*, *outOK*]]) (*auto intro*: *eqButUID-trans*)
qed

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* ≠ [] **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 −
let *?a* = *Dact* (*dFriend UID1* (*pass s UID1*) *UID2*)
let *?s'* = *deleteFriend s UID1* (*pass s UID1*) *UID2*
let *?trn* = *Trans s ?a outOK ?s'*
have *step*: *step s ?a* = (*outOK*, *?s'*) **using** *IDs f12 UID1-UID2*
by (*auto simp add*: *d-defs friends12-def*)
moreover then have *φ ?trn* **and** *f ?trn* = *FrVal False* **and** ¬*friends12 ?s'*
using *reach-friendIDs-symmetric*[*OF rs*] **by** (*auto simp*: *d-defs friends12-def*)
moreover have ¬*γ ?trn* **using** *UID1-UID2-UIDs* **by** *auto*
ultimately show *thesis* **using** *f12 rs*
by (*intro that*[*of* [*?a*] [*outOK*]]) (*auto intro*: *Dact-dFriend-step-eqButUID*)
qed

definition Δ*0* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*0 s vl s1 vl1* ≡
eqButUID s s1 ∧ *friendIDs s* = *friendIDs s1* ∧ *open s* ∧
BO vl vl1 ∧ *alternatingFriends vl1* (*friends12 s1*)

definition Δ*1* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*1 s vl s1 vl1* ≡ (∃ *fs fs1*.
eqButUID s s1 ∧ ¬*open s* ∧
alternatingFriends vl1 (*friends12 s1*) ∧
vl = *map FrVal fs* ∧ *vl1* = *map FrVal fs1*)

definition Δ*2* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
Δ*2 s vl s1 vl1* ≡ (∃ *fs fs1 vlr vlr1*.
eqButUID s s1 ∧ ¬*open s* ∧ *BO vlr vlr1* ∧
alternatingFriends vl1 (*friends12 s1*) ∧
(*fs* = [] ⟷ *fs1* = []) ∧
(*fs* ≠ [] ⟶ *last fs* = *last fs1*) ∧
(*fs* = [] ⟶ *friendIDs s* = *friendIDs s1*) ∧

vl = *map FrVal fs* @ *OVal True* # *vlr* ∧
vl1 = *map FrVal fs1* @ *OVal True* # *vlr1*)

lemma Δ*2-I*:
assumes *eqButUID s s1* ¬*open s BO vlr vlr1 alternatingFriends vl1* (*friends12 s1*)
fs = [] ⟷ *fs1* = [] *fs* ≠ [] ⟶ *last fs* = *last fs1*
fs = [] ⟶ *friendIDs s* = *friendIDs s1*
vl = *map FrVal fs* @ *OVal True* # *vlr*
vl1 = *map FrVal fs1* @ *OVal True* # *vlr1*
shows Δ*2 s vl s1 vl1*
using *assms* **unfolding** Δ*2-def* **by** *blast*

lemma *istate*-Δ*0*:
assumes *B*: *B vl vl1*
shows Δ*0 istate vl istate vl1*
using *assms* **unfolding** Δ*0-def istate-def B-def open-def openByA-def openByF-def friends12-def*
by *auto*

lemma *unwind-cont*-Δ*0*: *unwind-cont* Δ*0* {Δ*0*,Δ*1*,Δ*2*}
proof(*rule*, *simp*)
  let *?*Δ = λ*s vl s1 vl1*. Δ*0 s vl s1 vl1* ∨
                Δ*1 s vl s1 vl1* ∨
                Δ*2 s vl s1 vl1*
  fix *s s1* :: *state* **and** *vl vl1* :: *value list*
  assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** Δ*0*: Δ*0 s vl s1 vl1*
  then have *rs*: *reach s* **and** *ss1*: *eqButUID s s1* **and** *fIDs*: *friendIDs s* = *friendIDs s1*
    and *os*: *open s* **and** *BO*: *BO vl vl1* **and** *aF1*: *alternatingFriends vl1* (*friends12 s1*)
    using *reachNT-reach* **unfolding** Δ*0-def* **by** *auto*
  show *iaction ?*Δ *s vl s1 vl1* ∨
      ((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
  proof−
    have *?react* **proof**
      fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
      let *?trn* = *Trans s a ou s'*
      assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
      show *match ?*Δ *s s1 vl1 a ou s' vl'* ∨ *ignore ?*Δ *s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
      proof *cases*
        assume φ: φ *?trn*
        then have *vl*: *vl* = *f ?trn* # *vl'* **using** *c* **by** (*auto simp*: *consume-def*)
        from *BO* **have** *?match* **proof** (*cases f ?trn*)
          case (*FrVal fv*)
            with *BO vl* **obtain** *vl1'* **where** *vl1'*: *vl1* = *f ?trn* # *vl1'* **and** *BO'*: *BO vl' vl1'*
            proof (*cases rule*: *BO.cases*)

case (*BO-BC vl'' vl1'' fs*)
moreover with *vl FrVal* **obtain** *fs'* **where** *fs = fv # fs'* **by** (*cases fs*) *auto*
ultimately show *?thesis* **using** *FrVal BO-BC vl*
by (*intro that*[*of map FrVal fs' @ OVal False # vl1''*]) *auto*
qed *auto*
from *fIDs* **have** *f12*: *friends12 s = friends12 s1* **unfolding** *friends12-def* **by** *auto*
show *?match* **using** *φ step rs FrVal* **proof** (*cases rule*: *φE*)
case (*Friend uid p uid'*)
then have *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] []
using *ss1* **unfolding** *eqButUID-def* **by** *auto*
let *?s1' = createFriend s1 UID1* (*pass s1 UID1*) *UID2*
have *s'*: *s' = createFriend s UID1 p UID2*
using *Friend step* **by** (*auto simp*: *createFriend-sym*)
have *ss'*: *eqButUID s s'* **using** *rs step Friend*
by (*auto intro*: *Cact-cFriend-step-eqButUID*)
moreover then have *os'*: *open s'* **using** *os eqButUID-open-eq* **by** *auto*
moreover obtain *al oul* **where** *al*: *sstep s1 al* = (*oul*, *?s1'*) *al* ≠ []
and *tr1*: *O* (*traceOf s1 al*) = []
V (*traceOf s1 al*) = [*FrVal True*]
and *f12s1'*: *friends12 ?s1'*
and *s1s1'*: *eqButUID s1 ?s1'*
using *rs1 IDs1 Friend* **unfolding** *f12* **by** (*auto elim*: *toggle-friends12-True*)
moreover have *friendIDs s' = friendIDs ?s1'*
using *Friend*(*6*) *f12* **unfolding** *s'*
by (*intro eqButUID-createFriend12-friendIDs-eq*[*OF ss1 rs rs1*]) *auto*
ultimately have *Δ0 s' vl' ?s1' vl1'*
using *ss1 BO' aF1* **unfolding** *Δ0-def vl1' Friend*(*3*)
by (*auto intro*: *eqButUID-trans eqButUID-sym*)
then show *?match* **using** *tr1 vl1' Friend UID1-UID2-UIDs*
by (*intro matchI-ms*[*OF al*]) (*auto simp*: *consumeList-def*)
next
case (*Unfriend uid p uid'*)
then have *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] []
using *ss1* **unfolding** *eqButUID-def* **by** *auto*
let *?s1' = deleteFriend s1 UID1* (*pass s1 UID1*) *UID2*
have *s'*: *s' = deleteFriend s UID1 p UID2*
using *Unfriend step* **by** (*auto simp*: *deleteFriend-sym*)
have *ss'*: *eqButUID s s'* **using** *rs step Unfriend*
by (*auto intro*: *Dact-dFriend-step-eqButUID*)
moreover then have *os'*: *open s'* **using** *os eqButUID-open-eq* **by** *auto*
moreover obtain *al oul* **where** *al*: *sstep s1 al* = (*oul*, *?s1'*) *al* ≠ []
and *tr1*: *O* (*traceOf s1 al*) = []
V (*traceOf s1 al*) = [*FrVal False*]
and *f12s1'*: ¬*friends12 ?s1'*

and *s1s1'*: *eqButUID s1 ?s1'*
using *rs1 IDs1 Unfriend* **unfolding** *f12* **by** (*auto elim*: *toggle-friends12-False*)
moreover have *friendIDs s' = friendIDs ?s1'*
using *fIDs* **unfolding** *s'* **by** (*auto simp*: *d-defs*)
ultimately have Δ*0 s' vl' ?s1' vl1'*
using *ss1 BO' aF1* **unfolding** Δ*0-def vl1' Unfriend*(*3*)
by (*auto intro*: *eqButUID-trans eqButUID-sym*)
then show *?match* **using** *tr1 vl1' Unfriend UID1-UID2-UIDs*
by (*intro matchI-ms*[*OF al*]) (*auto simp*: *consumeList-def*)
qed *auto*
next
case (*OVal ov*)
with *BO vl* **obtain** *vl1'* **where** *vl1'*: *vl1 = OVal False # vl1'*
and *vl'*: *vl = OVal False # vl'*
and *BC*: *BC vl' vl1'*
proof (*cases rule*: *BO.cases*)
case (*BO-BC vl'' vl1'' fs*)
moreover then have *fs* = [] **using** *vl* **unfolding** *OVal* **by** (*cases fs*) *auto*
ultimately show *thesis* **using** *vl* **by** (*intro that*[*of vl1''*]) *auto*
qed *auto*
then have *f ?trn = OVal False* **using** *vl* **by** *auto*
with φ *step rs* **show** *?match* **proof** (*cases rule*: φ*E*)
case (*CloseF uid p uid'*)
let *?s1' = deleteFriend s1 uid p uid'*
let *?trn1 = Trans s1 a outOK ?s1'*
have *s'*: *s' = deleteFriend s uid p uid'* **using** *CloseF step* **by** *auto*
have *step1*: *step s1 a* = (*outOK*, *?s1'*)
using *CloseF step ss1 fIDs* **unfolding** *eqButUID-def* **by** (*auto simp*: *d-defs*)
have *s's1'*: *eqButUID s' ?s1'* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*] **.**
moreover have *os'*: ¬*open s'* **using** *CloseF os* **unfolding** *open-def* **by** *auto*
moreover have *fIDs'*: *friendIDs s' = friendIDs ?s1'*
using *fIDs* **unfolding** *s'* **by** (*auto simp*: *d-defs*)
moreover have *f12s1*: *friends12 s1 = friends12 ?s1'*
using *CloseF*(*2*) *UID1-UID2-UIDs* **unfolding** *friends12-def d-defs* **by** *auto*
from *BC* **have** Δ*1 s' vl' ?s1' vl1'* ∨ Δ*2 s' vl' ?s1' vl1'*
proof (*cases rule*: *BC.cases*)
case (*BC-FrVal fs fs1*)
then show *?thesis* **using** *aF1 os' fIDs' f12s1 s's1'* **unfolding** Δ*1-def vl1'* **by** *auto*
next
case (*BC-BO vlr vlr1 fs fs1*)
then have Δ*2 s' vl' ?s1' vl1'* **using** *s's1' os' aF1 f12s1 fIDs'* **unfolding** *vl1'*

```
        by (intro Δ2-I[of - - - - - fs fs1]) auto
      then show ?thesis ..
    qed
    moreover have open s1 ¬open ?s1'
      using ss1 os s's1' os' by (auto simp: eqButUID-open-eq)
    moreover then have φ ?trn1 unfolding CloseF by auto
      ultimately show ?match using step1 vl1' CloseF UID1-UID2
UID1-UID2-UIDs
        by (intro matchI[of s1 a outOK ?s1' vl1 vl1']) (auto simp:
consume-def)
  next
    case (CloseA uid p uid' p')
    let ?s1' = createUser s1 uid p uid' p'
    let ?trn1 = Trans s1 a outOK ?s1'
    have s': s' = createUser s uid p uid' p' using CloseA step by auto
    have step1: step s1 a = (outOK, ?s1')
      using CloseA step ss1 unfolding eqButUID-def by (auto simp:
c-defs)
    have s's1': eqButUID s' ?s1' using eqButUID-step[OF ss1 step step1
rs rs1] .
    moreover have os': ¬open s' using CloseA os unfolding open-def
by auto
    moreover have fIDs': friendIDs s' = friendIDs ?s1'
      using fIDs unfolding s' by (auto simp: c-defs)
    moreover have f12s1: friends12 s1 = friends12 ?s1'
      unfolding friends12-def by (auto simp: c-defs)
    from BC have Δ1 s' vl' ?s1' vl1' ∨ Δ2 s' vl' ?s1' vl1'
    proof (cases rule: BC.cases)
      case (BC-FrVal fs fs1)
        then show ?thesis using aF1 os' fIDs' f12s1 s's1' unfolding
Δ1-def vl1' by auto
    next
      case (BC-BO vlr vlr1 fs fs1)
        then have Δ2 s' vl' ?s1' vl1' using s's1' os' aF1 f12s1 fIDs'
unfolding vl1'
        by (intro Δ2-I[of - - - - - fs fs1]) auto
      then show ?thesis ..
    qed
    moreover have open s1 ¬open ?s1'
      using ss1 os s's1' os' by (auto simp: eqButUID-open-eq)
    moreover then have φ ?trn1 unfolding CloseA by auto
      ultimately show ?match using step1 vl1' CloseA UID1-UID2
UID1-UID2-UIDs
        by (intro matchI[of s1 a outOK ?s1' vl1 vl1']) (auto simp:
consume-def)
  qed auto
qed
then show ?match ∨ ?ignore ..
next
```

assume *n*φ: $\neg\varphi$ *?trn*
then have *os'*: *open s* = *open s'* **and** *f12s'*: *friends12 s* = *friends12 s'*
using *step-open-*φ[*OF step*] *step-friends12-*φ[*OF step*] **by** *auto*
have *vl'*: *vl'* = *vl* **using** *n*φ *c* **by** (*auto simp*: *consume-def*)
show *?thesis* **proof** (*cases a* $\neq$ *Cact* (*cFriend UID1* (*pass s UID1*) *UID2*) $\wedge$
a $\neq$ *Cact* (*cFriend UID2* (*pass s UID2*) *UID1*) $\wedge$
a $\neq$ *Dact* (*dFriend UID1* (*pass s UID1*) *UID2*) $\wedge$
a $\neq$ *Dact* (*dFriend UID2* (*pass s UID2*) *UID1*))
case *True*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by** (*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
have *fIDs'*: *friendIDs s'* = *friendIDs s1'*
using *eqButUID-step-friendIDs-eq*[*OF ss1 rs rs1 step step1 True fIDs*] **.**
from *True n*φ **have** *n*φ*'*: $\neg\varphi$ *?trn1* **using** *eqButUID-step-*φ[*OF ss1 rs rs1 step step1*] **by** *auto*
then have *f12s1'*: *friends12 s1* = *friends12 s1'*
using *step-friends12-*φ[*OF step1*] **by** *auto*
have *eqButUID s' s1'* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*] **.**
then have Δ*0 s' vl' s1' vl1* **using** *os fIDs' aF1 BO*
unfolding Δ*0-def os' f12s1' vl'* **by** *auto*
then have *?match*
using *step1 n*φ*' fIDs eqButUID-step-*γ*-out*[*OF ss1 step step1*]
by (*intro matchI*[*of s1 a ou1 s1' vl1 vl1*]) (*auto simp*: *consume-def*)
then show *?match* $\vee$ *?ignore* **..**
next
case *False*
with *n*φ **have** *ou* $\neq$ *outOK* **by** *auto*
then have *s'* = *s* **using** *step False* **by** *auto*
then have *?ignore* **using** Δ*0 False UID1-UID2-UIDs* **unfolding** *vl'* **by** (*intro ignoreI*) *auto*
then show *?match* $\vee$ *?ignore* **..**
qed
qed
qed
then show *?thesis* **using** *BO BO-Nil-Nil* **by** *auto*
qed
qed

lemma *unwind-cont-*Δ*1*: *unwind-cont* Δ*1* {Δ*1*, Δ*0*}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. Δ*1 s vl s1 vl1* $\vee$ Δ*0 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *1*: Δ*1 s vl s1 vl1*
from *rsT* **have** *rs*: *reach s* **by** (*intro reachNT-reach*)
from *1* **obtain** *fs fs1*
where *ss1*: *eqButUID s s1* **and** *os*: $\neg$*open s*
and *aF1*: *alternatingFriends vl1* (*friends12 s1*)

and *vl*: *vl* = *map FrVal fs* **and** *vl1*: *vl1* = *map FrVal fs1*
unfolding Δ*1-def* **by** *auto*
from *os* **have** *IDs*: *IDsOK s* [*UID1*, *UID2*] [] [] [] **unfolding** *open-defs* **by** *auto*
then have *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] [] **using** *ss1* **unfolding** *eqButUID-def* **by** *auto*
show *iaction ?*Δ *s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof *cases*
assume *fs1*: *fs1* = []
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
show *match ?*Δ *s s1 vl1 a ou s' vl'* ∨ *ignore ?*Δ *s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
proof *cases*
assume *φ*: *φ ?trn*
with *vl c* **obtain** *fv fs'* **where** *vl'*: *vl'* = *map FrVal fs'* **and** *fv*: *f ?trn* = *FrVal fv*
by (*cases fs*) (*auto simp*: *consume-def*)
from *φ step rs fv* **have** *ss'*: *eqButUID s s'*
by (*elim φE*) (*auto intro*: *Cact-cFriend-step-eqButUID Dact-dFriend-step-eqButUID*)
then have ¬*open s'* **using** *os* **by** (*auto simp*: *eqButUID-open-eq*)
moreover have *eqButUID s' s1* **using** *ss1 ss'* **by** (*auto intro*: *eqButUID-sym eqButUID-trans*)
ultimately have Δ*1 s' vl' s1 vl1* **using** *aF1* **unfolding** Δ*1-def vl' vl1* **by** *auto*
moreover have ¬*γ ?trn* **using** *φ step rs fv UID1-UID2-UIDs* **by** (*elim φE*) *auto*
ultimately have *?ignore* **by** (*intro ignoreI*) *auto*
then show *?match* ∨ *?ignore* **..**
next
assume *nφ*: ¬*φ ?trn*
then have *os'*: *open s* = *open s'* **and** *f12s'*: *friends12 s* = *friends12 s'*
using *step-open-φ*[*OF step*] *step-friends12-φ*[*OF step*] **by** *auto*
have *vl'*: *vl'* = *vl* **using** *nφ c* **by** (*auto simp*: *consume-def*)
show *?thesis* **proof** (*cases 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*))
case *True*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by** (*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
from *True nφ* **have** *nφ'*: ¬*φ ?trn1* **using** *eqButUID-step-φ*[*OF ss1 rs rs1 step step1*] **by** *auto*
then have *f12s1'*: *friends12 s1* = *friends12 s1'*
using *step-friends12-φ*[*OF step1*] **by** *auto*

```
          have eqButUID s' s1' using eqButUID-step[OF ss1 step step1 rs rs1] .
          then have Δ1 s' vl' s1' vl1 using os aF1 vl vl1
            unfolding Δ1-def os' vl' f12s1' by auto
          then have ?match
            using step1 nφ' os eqButUID-step-γ-out[OF ss1 step step1]
            by (intro matchI[of s1 a ou1 s1' vl1 vl1]) (auto simp: consume-def)
          then show ?match ∨ ?ignore ..
        next
          case False
          with nφ have ou ≠ outOK by auto
          then have s' = s using step False by auto
           then have ?ignore using 1 False UID1-UID2-UIDs unfolding vl' by
(intro ignoreI) auto
          then show ?match ∨ ?ignore ..
        qed
      qed
    qed
    then show ?thesis using fs1 unfolding vl1 by auto
  next
    assume fs1 ≠ []
    then obtain fs1' where fs1: fs1 = (¬friends12 s1) # fs1'
                     and aF1': alternatingFriends (map FrVal fs1') (¬friends12 s1)
      using aF1 unfolding vl1 by (cases fs1) auto
    obtain al oul s1' where sstep s1 al = (oul, s1') al ≠ [] eqButUID s1 s1'
                            friends12 s1' = (¬friends12 s1)
                          O (traceOf s1 al) = [] V (traceOf s1 al) = [FrVal (¬friends12
s1)]
      using rs1 IDs1
    by (cases friends12 s1) (auto intro: toggle-friends12-True toggle-friends12-False)
    moreover then have Δ1 s vl s1' (map FrVal fs1')
        using os aF1' vl ss1 unfolding Δ1-def by (auto intro: eqButUID-sym
eqButUID-trans)
    ultimately have ?iact using vl1 unfolding fs1
      by (intro iactionI-ms[of s1 al oul s1'])
         (auto simp: consumeList-def O-Nil-never list-ex-iff-length-V)
    then show ?thesis ..
  qed
qed

lemma unwind-cont-Δ2: unwind-cont Δ2 {Δ2,Δ0}
proof(rule, simp)
  let ?Δ = λs vl s1 vl1. Δ2 s vl s1 vl1 ∨ Δ0 s vl s1 vl1
  fix s s1 :: state and vl vl1 :: value list
  assume rsT: reachNT s and rs1: reach s1 and 2: Δ2 s vl s1 vl1
  from rsT have rs: reach s by (intro reachNT-reach)
  obtain fs fs1 vlr vlr1
  where ss1: eqButUID s s1 and os: ¬open s and BO: BO vlr vlr1
    and aF1: alternatingFriends vl1 (friends12 s1)
    and vl: vl = map FrVal fs @ OVal True # vlr
```

and *vl1*: *vl1* = *map FrVal fs1* @ *OVal True* # *vlr1*
and *fs-fs1*: *fs* = [] ⟷ *fs1* = []
and *last-fs*: *fs* ≠ [] ⟶ *last fs* = *last fs1*
and *fs-fIDs*: *fs* = [] ⟶ *friendIDs s* = *friendIDs s1*
using *2* **unfolding** Δ*2-def* **by** *auto*
from *os* **have** *IDs*: *IDsOK s* [*UID1*, *UID2*] [] [] [] **unfolding** *open-defs* **by** *auto*
then have *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] [] **using** *ss1* **unfolding** *eqButUID-def* **by** *auto*
show *iaction ?*Δ *s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof *cases*
assume *length fs1* > *1*
then obtain *fs1′*
where *fs1*: *fs1* = (¬*friends12 s1*) # *fs1′* **and** *fs1′*: *fs1′* ≠ []
and *last-fs′*: *last fs1* = *last fs1′*
and *aF1′*: *alternatingFriends* (*map FrVal fs1′* @ *OVal True* # *vlr1*) (¬*friends12 s1*)
using *vl1 aF1* **by** (*cases fs1*) *auto*
obtain *al oul s1′* **where** *sstep s1 al* = (*oul*, *s1′*) *al* ≠ [] *eqButUID s1 s1′*
friends12 s1′ = (¬*friends12 s1*)
O (*traceOf s1 al*) = [] *V* (*traceOf s1 al*) = [*FrVal* (¬*friends12 s1*)]
using *rs1 IDs1*
by (*cases friends12 s1*) (*auto intro*: *toggle-friends12-True toggle-friends12-False*)
moreover then have Δ*2 s vl s1′* (*map FrVal fs1′* @ *OVal True* # *vlr1*)
using *os aF1′ vl ss1 fs1′ last-fs′ fs-fs1 last-fs BO* **unfolding** *fs1*
by (*intro* Δ*2-I*[*of - - vlr vlr1 - fs fs1′*])
(*auto intro*: *eqButUID-sym eqButUID-trans*)
ultimately have *?iact* **using** *vl1* **unfolding** *fs1*
by (*intro iactionI-ms*[*of s1 al oul s1′*])
(*auto simp*: *consumeList-def O-Nil-never list-ex-iff-length-V*)
then show *?thesis* **..**
next
assume *len1-leq-1*: ¬ *length fs1* > *1*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn* = *Trans s a ou s′* **let** *?trn1* = *Trans s1 a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl′*
show *match ?*Δ *s s1 vl1 a ou s′ vl′* ∨ *ignore ?*Δ *s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
proof *cases*
assume φ: φ *?trn*
show *?thesis* **proof** *cases*
assume *length fs* > *1*
then obtain *fv fs′*
where *fs1*: *fs* = *fv* # *fs′* **and** *fs1′*: *fs′* ≠ []
and *last-fs′*: *last fs* = *last fs′*
using *vl* **by** (*cases fs*) *auto*
with φ *c* **have** *fv*: *f ?trn* = *FrVal fv* **and** *vl′*: *vl′* = *map FrVal fs′* @ *OVal*

True # *vlr*
unfolding *vl consume-def* **by** *auto*
from φ *step rs fv* **have** *ss'*: *eqButUID s s'*
by (*elim* φE) (*auto intro*: *Cact-cFriend-step-eqButUID Dact-dFriend-step-eqButUID*)
then have $\neg$*open s'* **using** *os* **by** (*auto simp*: *eqButUID-open-eq*)
moreover have *eqButUID s' s1* **using** *ss1 ss'* **by** (*auto intro*: *eqButUID-sym eqButUID-trans*)
ultimately have $\Delta 2$ *s' vl' s1 vl1*
using *aF1 vl' fs1' fs-fs1 last-fs BO* **unfolding** *fs1 vl1*
by (*intro* $\Delta 2$*-I*[*of - - vlr vlr1 - fs' fs1*])
(*auto intro*: *eqButUID-sym eqButUID-trans*)
moreover have $\neg\gamma$ *?trn* **using** φ *step rs fv UID1-UID2-UIDs* **by** (*elim* φE) *auto*
ultimately have *?ignore* **by** (*intro ignoreI*) *auto*
then show *?match* $\vee$ *?ignore* **..**
next
assume *len-leq-1*: $\neg$ *length fs* > *1*
show *?thesis* **proof** *cases*
assume *fs*: *fs* = []
then have *fs1*: *fs1* = [] **and** *fIDs*: *friendIDs s* = *friendIDs s1*
using *fs-fs1 fs-fIDs* **by** *auto*
from *fs* φ *c* **have** *ov*: *f ?trn* = *OVal True* **and** *vl'*: *vl'* = *vlr*
unfolding *vl consume-def* **by** *auto*
with φ *step rs* **have** *?match* **proof** (*cases rule*: φE)
case (*OpenF uid p uid'*)
let *?s1'* = *createFriend s1 uid p uid'*
let *?trn1* = *Trans s1 a outOK ?s1'*
have *s'*: *s'* = *createFriend s uid p uid'* **using** *OpenF step* **by** *auto*
have *eqButUIDf* (*pendingFReqs s*) (*pendingFReqs s1*)
using *ss1* **unfolding** *eqButUID-def* **by** *auto*
then have *uid'* $\in\in$ *pendingFReqs s uid* $\longleftrightarrow$ *uid'* $\in\in$ *pendingFReqs s1 uid*
using *OpenF* **by** (*intro eqButUIDf-not-UID'*) *auto*
then have *step1*: *step s1 a* = (*outOK*, *?s1'*)
using *OpenF step ss1 fIDs* **unfolding** *eqButUID-def* **by** (*auto simp*: *c-defs*)
have *s's1'*: *eqButUID s' ?s1'* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*] **.**
moreover have *os'*: *open s'* **using** *OpenF* **unfolding** *open-def* **by** *auto*
moreover have *fIDs'*: *friendIDs s'* = *friendIDs ?s1'*
using *fIDs* **unfolding** *s'* **by** (*auto simp*: *c-defs*)
moreover have *f12s1*: *friends12 s1* = *friends12 ?s1'*
using *OpenF*(*2*) *UID1-UID2-UIDs* **unfolding** *friends12-def c-defs* **by** *auto*
ultimately have $\Delta 0$ *s' vl' ?s1' vlr1*
using *BO aF1* **unfolding** $\Delta 0$*-def vl' vl1 fs1* **by** *auto*
moreover have $\neg$*open s1 open ?s1'*
using *ss1 os s's1' os'* **by** (*auto simp*: *eqButUID-open-eq*)

moreover then have φ *?trn1* **unfolding** *OpenF* **by** *auto*
ultimately show *?match* **using** *step1 vl1 fs1 OpenF UID1-UID2 UID1-UID2-UIDs*
by (*intro matchI*[*of s1 a outOK ?s1' vl1 vlr1*]) (*auto simp*: *consume-def*)
qed *auto*
then show *?thesis* **..**
next
assume *fs* $\neq$ []
then obtain *fv* **where** *fs*: *fs* = [*fv*] **using** *len-leq-1* **by** (*cases fs*) *auto*
then have *fs1*: *fs1* = [*fv*] **using** *len1-leq-1 fs-fs1 last-fs* **by** (*cases fs1*) *auto*
with *aF1* **have** *f12s1*: *friends12 s1* = ($\neg$*fv*) **unfolding** *vl1* **by** *auto*
have *fv*: *f ?trn* = *FrVal fv* **and** *vl'*: *vl'* = *OVal True* # *vlr*
using *c* φ **unfolding** *vl fs* **by** (*auto simp*: *consume-def*)
with φ *step rs* **have** *?match* **proof** (*cases rule*: φ*E*)
case (*Friend uid p uid'*)
then have *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] []
using *ss1* **unfolding** *eqButUID-def* **by** *auto*
have *fv*: *fv* = *True* **using** *fv Friend* **by** *auto*
let *?s1'* = *createFriend s1 UID1* (*pass s1 UID1*) *UID2*
have *s'*: *s'* = *createFriend s UID1 p UID2*
using *Friend step* **by** (*auto simp*: *createFriend-sym*)
have *ss'*: *eqButUID s s'* **using** *rs step Friend*
by (*auto intro*: *Cact-cFriend-step-eqButUID*)
moreover then have *os'*: $\neg$*open s'* **using** *os eqButUID-open-eq* **by** *auto*
moreover obtain *al oul* **where** *al*: *sstep s1 al* = (*oul*, *?s1'*) *al* $\neq$ []
and *tr1*: *O* (*traceOf s1 al*) = []
V (*traceOf s1 al*) = [*FrVal True*]
and *f12s1'*: *friends12 ?s1'*
and *s1s1'*: *eqButUID s1 ?s1'*
using *rs1 IDs1 Friend f12s1* **unfolding** *fv* **by** (*auto elim*: *toggle-friends12-True*)
moreover have *friendIDs s'* = *friendIDs ?s1'*
using *Friend*(*6*) *f12s1* **unfolding** *s' fv*
by (*intro eqButUID-createFriend12-friendIDs-eq*[*OF ss1 rs rs1*]) *auto*
ultimately have Δ*2 s' vl' ?s1'* (*OVal True* # *vlr1*)
using *BO ss1 aF1* **unfolding** *vl' vl1 fs1 f12s1 fv*
by (*intro* Δ*2-I*[*of - - - - -* [] []])
(*auto intro*: *eqButUID-trans eqButUID-sym*)
then show *?match* **using** *tr1 vl1 Friend UID1-UID2-UIDs* **unfolding** *fs1 fv*
by (*intro matchI-ms*[*OF al*]) (*auto simp*: *consumeList-def*)
next
case (*Unfriend uid p uid'*)
then have *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] []
using *ss1* **unfolding** *eqButUID-def* **by** *auto*
have *fv*: *fv* = *False* **using** *fv Unfriend* **by** *auto*

let *?s1′* = *deleteFriend s1 UID1* (*pass s1 UID1*) *UID2*
have *s′*: *s′* = *deleteFriend s UID1 p UID2*
using *Unfriend step* **by** (*auto simp*: *deleteFriend-sym*)
have *ss′*: *eqButUID s s′* **using** *rs step Unfriend*
by (*auto intro*: *Dact-dFriend-step-eqButUID*)
moreover then have *os′*: ¬*open s′* **using** *os eqButUID-open-eq* **by** *auto*
moreover obtain *al oul* **where** *al*: *sstep s1 al* = (*oul*, *?s1′*) *al* ≠ []
and *tr1*: *O* (*traceOf s1 al*) = []
V (*traceOf s1 al*) = [*FrVal False*]
and *f12s1′*: ¬*friends12 ?s1′*
and *s1s1′*: *eqButUID s1 ?s1′*
using *rs1 IDs1 Unfriend f12s1* **unfolding** *fv* **by** (*auto elim*: *toggle-friends12-False*)
moreover have *friendIDs s′* = *friendIDs ?s1′*
using *Unfriend*(*6*) *f12s1* **unfolding** *s′ fv*
by (*intro eqButUID-deleteFriend12-friendIDs-eq*[*OF ss1 rs rs1*])
ultimately have Δ*2 s′ vl′ ?s1′* (*OVal True # vlr1*)
using *BO ss1 aF1* **unfolding** *vl′ vl1 fs1 f12s1 fv*
by (*intro* Δ*2-I*[*of - - - - -* [] []])
(*auto intro*: *eqButUID-trans eqButUID-sym*)
then show *?match* **using** *tr1 vl1 Unfriend UID1-UID2-UIDs* **unfolding** *fs1 fv*
by (*intro matchI-ms*[*OF al*]) (*auto simp*: *consumeList-def*)
qed *auto*
then show *?thesis* **..**
qed
qed
next
assume *nφ*: ¬*φ ?trn*
then have *os′*: *open s* = *open s′* **and** *f12s′*: *friends12 s* = *friends12 s′*
using *step-open-φ*[*OF step*] *step-friends12-φ*[*OF step*] **by** *auto*
have *vl′*: *vl′* = *vl* **using** *nφ c* **by** (*auto simp*: *consume-def*)
show *?thesis* **proof** (*cases 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*))
case *True*
obtain *ou1 s1′* **where** *step1*: *step s1 a* = (*ou1*, *s1′*) **by** (*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1′*
from *True nφ* **have** *nφ′*: ¬*φ ?trn1* **using** *eqButUID-step-φ*[*OF ss1 rs rs1 step step1*] **by** *auto*
then have *f12s1′*: *friends12 s1* = *friends12 s1′*
using *step-friends12-φ*[*OF step1*] **by** *auto*
have *eqButUID s′ s1′* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*] **.**
moreover have *friendIDs s* = *friendIDs s1* ⟶ *friendIDs s′* = *friendIDs s1′*

```
          using eqButUID-step-friendIDs-eq[OF ss1 rs rs1 step step1 True] ..
        ultimately have Δ2 s' vl' s1' vl1
          using os' os aF1 BO fs-fs1 last-fs fs-fIDs unfolding f12s1' vl' vl vl1
          by (intro Δ2-I) auto
        then have ?match
          using step1 nφ' os eqButUID-step-γ-out[OF ss1 step step1]
          by (intro matchI[of s1 a ou1 s1' vl1 vl1]) (auto simp: consume-def)
        then show ?match ∨ ?ignore ..
    next
      case False
        with nφ have ou ≠ outOK by auto
        then have s' = s using step False by auto
         then have ?ignore using 2 False UID1-UID2-UIDs unfolding vl' by
(intro ignoreI) auto
        then show ?match ∨ ?ignore ..
    qed
   qed
  qed
  then show ?thesis unfolding vl by auto
 qed
qed


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


theorem secure: secure
apply (rule unwind-decomp-secure-graph[of Gr Δ0])
unfolding Gr-def
apply (simp, smt insert-subset order-refl)
using
istate-Δ0 unwind-cont-Δ0 unwind-cont-Δ1 unwind-cont-Δ2
unfolding Gr-def by (auto intro: unwind-cont-mono)

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* **using** *UID1-UID2-UIDs UID1-UID2*
by *unfold-locales*

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*
by (*cases trn*) (*auto elim*: *Issuer.φE*)

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 (*unfold-locales*, *goal-cases*)

case (*1 aid trn*) **then show** *?case* **by** *auto* **next**
case (*2 aid trn*) **then show** *?case* **by** *auto* **next**
case (*3 aid trn*) **then show** *?case* **by** (*cases trn*) *auto* **next**
case (*4 aid trn*) **then show** *?case* **by** (*cases* (*aid*,*trn*) *rule*: *tgtNodeOf.cases*) *auto* **next**
case (*5 aid1 trn1 aid2 trn2*) **then show** *?case* **by** *auto* **next**
case (*6 aid1 trn1 aid2 trn2*) **then show** *?case* **by** (*cases trn1*; *cases trn2*; *auto*) **next**
case (*7 aid1 trn1 aid2 trn2*) **then show** *?case* **by** *auto* **next**
case (*8 aid1 trn1 aid2 trn2*) **then show** *?case* **by** (*cases trn1*; *cases trn2*; *auto*) **next**
case (*9 aid trn*) **then show** *?case* **by** (*cases* (*aid*,*trn*) *rule*: *tgtNodeOf.cases*) (*auto simp*: *FriendObservationSetup.γ.simps*) **next**
case (*10 aid trn*) **then show** *?case* **by** *auto*
qed *auto*

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*
using *AID-AIDs Issuer.secure*
by *unfold-locales auto*

theorem *secure*: *secure*
proof (*intro preserve-source-secure ballI*)
fix *aid*
assume *aid* ∈ *AIDs* − {*AID*}
then show *Net.lsecure aid* **by** (*intro Abstract-BD-Security.B-id-secure*) (*auto simp*: *B-id-def*)
qed

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* ⇒ *bool* **where**
φ (*Trans s* (*Cact* (*cFriendReq uid p uid' req*)) *ou s'*) =
((*uid*,*uid'*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)} ∧ *ou* = *outOK*)
|
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* (*cFriendReq uid p uid' req*)) *ou s'*) =
(*if uid* = *UID1* ∧ *uid'* = *UID2 then FRVal U1 req*
else if uid = *UID2* ∧ *uid'* = *UID1 then FRVal U2 req*
else OVal True)
|
f (*Trans s* (*Cact* (*cFriend uid p uid'*)) *ou s'*) =
(*if* (*uid*,*uid'*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)} *then FVal True*
else OVal True)
|
f (*Trans s* (*Dact* (*dFriend uid p uid'*)) *ou s'*) =
(*if* (*uid*,*uid'*) ∈ {(*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 φ*E*:
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 ∈∈ *pendingFReqs s' UID2 UID1* ∉ *set* (*pendingFReqs s UID2*)
UID2 ∈∈ *pendingFReqs s' UID1* ⟷ *UID2* ∈∈ *pendingFReqs s UID1*
| (*FReq2*) *u p req* **where** *a* = *Cact* (*cFriendReq UID2 p UID1 req*) *ou* = *outOK*

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

friends12 s′ = friends12 s
UID1 ∈∈ pendingFReqs s′ UID2 ⟷ UID1 ∈∈ pendingFReqs s UID2
UID2 ∈∈ pendingFReqs s′ UID1 ⟷ UID2 ∈∈ pendingFReqs s UID1
using *φ* **proof** (*elim φ.elims disjE conjE*)
  fix *s1 uid p uid′ req ou1 s1′*
  assume (*uid*,*uid′*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)} **and** *ou*: *ou1* = *outOK*
    and *?trn* = *Trans s1* (*Cact* (*cFriendReq uid p uid′ req*)) *ou1 s1′*
  then have *trn*: *a* = *Cact* (*cFriendReq uid p uid′ req*) *s* = *s1 s′* = *s1′ ou* = *ou1*
      and *uids*: *uid* = *UID1* ∧ *uid′* = *UID2* ∨ *uid* = *UID2* ∧ *uid′* = *UID1* **using** *UID1-UID2* **by** *auto*
  from *uids* **show** *thesis* **proof**
    assume *uid* = *UID1* ∧ *uid′* = *UID2*
    then show *thesis* **using** *ou uids trn step UID1-UID2-UIDs UID1-UID2 reach-distinct-friends-reqs*[*OF rs*]
      by (*intro FReq1*[*of p req*]) (*auto simp add*: *c-defs friends12-def open-defs*)
  next
    assume *uid* = *UID2* ∧ *uid′* = *UID1*
    then show *thesis* **using** *ou uids trn step UID1-UID2-UIDs UID1-UID2 reach-distinct-friends-reqs*[*OF rs*]
      by (*intro FReq2*[*of p req*]) (*auto simp add*: *c-defs friends12-def open-defs*)
  qed
next
  fix *s1 uid p uid′ ou1 s1′*
  assume (*uid*,*uid′*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)} **and** *ou*: *ou1* = *outOK*
    and *?trn* = *Trans s1* (*Cact* (*cFriend uid p uid′*)) *ou1 s1′*
  then have *trn*: *a* = *Cact* (*cFriend uid p uid′*) *s* = *s1 s′* = *s1′ ou* = *ou1*
      and *uids*: *uid* = *UID1* ∧ *uid′* = *UID2* ∨ *uid* = *UID2* ∧ *uid′* = *UID1* **using** *UID1-UID2* **by** *auto*
  then show *thesis* **using** *ou uids trn step UID1-UID2-UIDs UID1-UID2 reach-distinct-friends-reqs*[*OF rs*]
    by (*intro Friend*[*of uid p uid′*]) (*auto simp add*: *c-defs friends12-def*)
next
  fix *s1 uid p uid′ ou1 s1′*
  assume *op*: *open s1* ≠ *open s1′*
    and *?trn* = *Trans s1* (*Cact* (*cFriend uid p uid′*)) *ou1 s1′*
  then have *trn*: *open s* ≠ *open s′ a* = *Cact* (*cFriend uid p uid′*) *s* = *s1 s′* = *s1′ ou* = *ou1*
    by *auto*
  with *step* **have** *uids*: (*uid* ∈ *UIDs* ∧ *uid′* ∈ {*UID1*, *UID2*} ∨ *uid* ∈ {*UID1*, *UID2*} ∧ *uid′* ∈ *UIDs*) ∧
    ou = *outOK* ∧ *p* = *pass s uid* ∧
    ¬*openByF s* ∧ *openByF s′* ∧ ¬*openByA s* ∧ ¬*openByA s′*
    by (*cases rule*: *step-open-cases*) *auto*
  moreover have *friends12 s′* ⟷ *friends12 s*
    using *step-friendIDs*[*OF step*, *of UID1 UID2*] *trn uids UID1-UID2 UID1-UID2-UIDs*
    by (*auto simp add*: *friends12-def*)
  moreover have (*UID1* ∈∈ *pendingFReqs s′ UID2* ⟷ *UID1* ∈∈ *pendingFReqs*

s UID2) ∧
(*UID2* ∈∈ *pendingFReqs s′ UID1* ⟷ *UID2* ∈∈ *pendingFReqs s UID1*)
using *step-pendingFReqs*[*OF step*, *of UID1 UID2*] *trn uids UID1-UID2 UID1-UID2-UIDs*
by *auto*
ultimately show *thesis* **using** *trn*(*2*) *step UID1-UID2-UIDs UID1-UID2* **by** (*intro OpenF*) *auto*
next
fix *s1 uid p uid′ ou1 s1′*
assume (*uid*,*uid′*) ∈ {(*UID1*,*UID2*), (*UID2*,*UID1*)} **and** *ou*: *ou1* = *outOK*
and *?trn* = *Trans s1* (*Dact* (*dFriend uid p uid′*)) *ou1 s1′*
then have *trn*: *a* = *Dact* (*dFriend uid p uid′*) *s* = *s1 s′* = *s1′ ou* = *ou1*
and *uids*: *uid* = *UID1* ∧ *uid′* = *UID2* ∨ *uid* = *UID2* ∧ *uid′* = *UID1* **using** *UID1-UID2* **by** *auto*
then show *thesis* **using** *step ou reach-friendIDs-symmetric*[*OF rs*] *reach-distinct-friends-reqs*[*OF rs*]
by (*intro Unfriend*; *auto simp*: *d-defs friends12-def*) *blast*+
next
fix *s1 uid p uid′ ou1 s1′*
assume *op*: *open s1* ≠ *open s1′*
and *?trn* = *Trans s1* (*Dact* (*dFriend uid p uid′*)) *ou1 s1′*
then have *trn*: *open s* ≠ *open s′ a* = *Dact* (*dFriend uid p uid′*) *s* = *s1 s′* = *s1′ ou* = *ou1*
by *auto*
with *step* **have** *uids*: (*uid* ∈ *UIDs* ∧ *uid′* ∈ {*UID1*, *UID2*} ∨ *uid* ∈ {*UID1*, *UID2*} ∧ *uid′* ∈ *UIDs*) ∧
ou = *outOK* ∧ *openByF s* ∧ ¬*openByF s′* ∧ ¬*openByA s* ∧ ¬*openByA s′*
by (*cases rule*: *step-open-cases*) *auto*
moreover have *friends12 s′* ⟷ *friends12 s*
using *step-friendIDs*[*OF step*, *of UID1 UID2*] *trn uids UID1-UID2 UID1-UID2-UIDs*
by (*auto simp add*: *friends12-def*)
moreover have (*UID1* ∈∈ *pendingFReqs s′ UID2* ⟷ *UID1* ∈∈ *pendingFReqs s UID2*) ∧
(*UID2* ∈∈ *pendingFReqs s′ UID1* ⟷ *UID2* ∈∈ *pendingFReqs s UID1*)
using *step-pendingFReqs*[*OF step*, *of UID1 UID2*] *trn uids UID1-UID2 UID1-UID2-UIDs*
by *auto*
ultimately show *thesis* **using** *trn*(*1*,*2*) *step UID1-UID2 UID1-UID2-UIDs*
by (*intro CloseF*) *auto*
next
fix *s1 uid p uid′ p′ ou1 s1′*
assume *op*: *open s1* ≠ *open s1′*
and *?trn* = *Trans s1* (*Cact* (*cUser uid p uid′ p′*)) *ou1 s1′*
then have *trn*: *open s* ≠ *open s′ a* = *Cact* (*cUser uid p uid′ p′*) *s* = *s1 s′* = *s1′ ou* = *ou1*
by *auto*
with *step* **have** *uids*: (*uid′* = *UID2* ∨ *uid′* = *UID1*) ∧ *ou* = *outOK* ∧
¬*openByF s1* ∧ ¬*openByF s1′* ∧ *openByA s1* ∧ ¬*openByA s1′*

by (*cases rule*: *step-open-cases*) *auto*
moreover have *friends12 s1* ′ ⟷ *friends12 s1*
using *step-friendIDs*[*OF step*, *of UID1 UID2*] *trn uids UID1-UID2 UID1-UID2-UIDs*
by (*auto simp add*: *friends12-def*)
moreover have (*UID1* ∈∈ *pendingFReqs s*′ *UID2* ⟷ *UID1* ∈∈ *pendingFReqs s UID2*) ∧
(*UID2* ∈∈ *pendingFReqs s*′ *UID1* ⟷ *UID2* ∈∈ *pendingFReqs s UID1*)
using *step-pendingFReqs*[*OF step*, *of UID1 UID2*] *trn uids UID1-UID2 UID1-UID2-UIDs*
by *auto*
ultimately show *thesis* **using** *trn step UID1-UID2-UIDs UID1-UID2* **by** (*intro CloseA*) *auto*
qed

lemma *step-open-φ*:
assumes *step s a* = (*ou*, *s*′)
and *open s* ≠ *open s*′
shows φ (*Trans s a ou s*′)
using *assms* **by** (*cases rule*: *step-open-cases*) (*auto simp*: *open-def*)

lemma *step-friends12-φ*:
assumes *step s a* = (*ou*, *s*′)
and *friends12 s* ≠ *friends12 s*′
shows φ (*Trans s a ou s*′)
proof −
have *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*)
using *assms step-friends12* **by** *auto*
moreover then have *ou* = *outOK* **using** *assms* **by** *auto*
ultimately show φ (*Trans s a ou s*′) **by** *auto*
qed

lemma *step-pendingFReqs-φ*:
assumes *step s a* = (*ou*, *s*′)
and (*UID1* ∈∈ *pendingFReqs s UID2*) ≠ (*UID1* ∈∈ *pendingFReqs s*′ *UID2*)
∨ (*UID2* ∈∈ *pendingFReqs s UID1*) ≠ (*UID2* ∈∈ *pendingFReqs s*′ *UID1*)
shows φ (*Trans s a ou s*′)
proof −
have ∃ *req*. *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*) ∨
a = *Cact* (*cFriendReq UID1* (*pass s UID1*) *UID2 req*) ∨
a = *Cact* (*cFriendReq UID2* (*pass s UID2*) *UID1 req*)
by (*rule ccontr*, *insert assms step-pendingFReqs*) *auto*
moreover then have *ou* = *outOK* **using** *assms* **by** *auto*
ultimately show φ (*Trans s a ou s*′) **by** *auto*

qed

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*: ∀ *req*. *a* ≠ *Cact* (*cFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Cact* (*cFriend UID2* (*pass s UID2*) *UID1*) ∧
a ≠ *Cact* (*cFriendReq UID1* (*pass s UID1*) *UID2 req*) ∧
a ≠ *Cact* (*cFriendReq UID2* (*pass s UID2*) *UID1 req*) ∧
a ≠ *Dact* (*dFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Dact* (*dFriend UID2* (*pass s UID2*) *UID1*)
and *φ*: *φ* (*Trans s a ou s'*)
shows *φ* (*Trans s1 a ou1 s1'*)
proof −
have *eqButUID s' s1'* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*] **.**
then have *open s* = *open s1* **and** *open s'* = *open s1'*
and *openByA s* = *openByA s1* **and** *openByA s'* = *openByA s1'*
and *openByF s* = *openByF s1* **and** *openByF s'* = *openByF s1'*
using *ss1* **by** (*auto simp*: *eqButUID-open-eq eqButUID-openByA-eq eqButUID-openByF-eq*)
with *φ a step step1* **show** *φ* (*Trans s1 a ou1 s1'*) **using** *UID1-UID2-UIDs*
by (*elim φ.elims*) (*auto simp*: *c-defs d-defs*)
qed

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*: ∀ *req*. *a* ≠ *Cact* (*cFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Cact* (*cFriend UID2* (*pass s UID2*) *UID1*) ∧
a ≠ *Cact* (*cFriendReq UID1* (*pass s UID1*) *UID2 req*) ∧
a ≠ *Cact* (*cFriendReq UID2* (*pass s UID2*) *UID1 req*) ∧
a ≠ *Dact* (*dFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Dact* (*dFriend UID2* (*pass s UID2*) *UID1*)
shows *φ* (*Trans s a ou s'*) = *φ* (*Trans s1 a ou1 s1'*)
proof
assume *φ* (*Trans s a ou s'*)
with *assms* **show** *φ* (*Trans s1 a ou1 s1'*) **by** (*rule eqButUID-step-φ-imp*)
next
assume *φ* (*Trans s1 a ou1 s1'*)
moreover have *eqButUID s1 s* **using** *ss1* **by** (*rule eqButUID-sym*)
moreover have ∀ *req*. *a* ≠ *Cact* (*cFriend UID1* (*pass s1 UID1*) *UID2*) ∧
a ≠ *Cact* (*cFriend UID2* (*pass s1 UID2*) *UID1*) ∧
a ≠ *Cact* (*cFriendReq UID1* (*pass s1 UID1*) *UID2 req*) ∧
a ≠ *Cact* (*cFriendReq UID2* (*pass s1 UID2*) *UID1 req*) ∧
a ≠ *Dact* (*dFriend UID1* (*pass s1 UID1*) *UID2*) ∧
a ≠ *Dact* (*dFriend UID2* (*pass s1 UID2*) *UID1*)
using *a ss1* **unfolding** *eqButUID-def* **by** *auto*

ultimately show φ (*Trans s a ou s'*) **using** *rs rs1 step step1*
by (*intro eqButUID-step-*φ*-imp*[*of s1 s*])
qed

end

end
theory *Friend-Request*
imports
Friend-Request-Value-Setup
Bounded-Deducibility-Security.Compositional-Reasoning
begin

8.2 Declassification bound

context *Friend*
begin

fun *T* :: (*state*,*act*,*out*) *trans* ⇒ *bool*
where *T trn* = *False*

Friendship updates form an alternating sequence of friending and unfriending, and every successful friend creation is preceded by one or two friendship requests.

fun *validValSeq* :: *value list* ⇒ *bool* ⇒ *bool* ⇒ *bool* ⇒ *bool* **where**
validValSeq [] - - - = *True*
| *validValSeq* (*FRVal U1 req* # *vl*) *st r1 r2* ⟷ (¬*st*) ∧ (¬*r1*) ∧ *validValSeq vl st True r2*
| *validValSeq* (*FRVal U2 req* # *vl*) *st r1 r2* ⟷ (¬*st*) ∧ (¬*r2*) ∧ *validValSeq vl st r1 True*
| *validValSeq* (*FVal True* # *vl*) *st r1 r2* ⟷ (¬*st*) ∧ (*r1* ∨ *r2*) ∧ *validValSeq vl True False False*
| *validValSeq* (*FVal False* # *vl*) *st r1 r2* ⟷ *st* ∧ (¬*r1*) ∧ (¬*r2*) ∧ *validValSeq vl False False False*
| *validValSeq* (*OVal True* # *vl*) *st r1 r2* ⟷ *validValSeq vl st r1 r2*
| *validValSeq* (*OVal False* # *vl*) *st r1 r2* ⟷ *validValSeq vl st r1 r2*

abbreviation *validValSeqFrom* :: *value list* ⇒ *state* ⇒ *bool*
where *validValSeqFrom vl s*
≡ *validValSeq vl* (*friends12 s*) (*UID1* ∈∈ *pendingFReqs s UID2*) (*UID2* ∈∈ *pendingFReqs s UID1*)

With respect to the friendship status updates, we use the same "while-or-last-before" bound as for friendship status confidentiality.

inductive *BO* :: *value list* ⇒ *value list* ⇒ *bool*
and *BC* :: *value list* ⇒ *value list* ⇒ *bool*
where
BO-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* = []
by (*cases rule*: *BO.cases*) *auto*

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** $B = B$
done

8.3 Unwinding proof

lemma *validFrom-validValSeq*:
assumes *validFrom s tr*
and *reach s*
shows *validValSeqFrom* (*V tr*) *s*
using *assms* **proof** (*induction tr arbitrary*: *s*)

case (*Cons trn tr s*)
then obtain *a ou s'* **where** *trn*: *trn* = *Trans s a ou s'*
and *step*: *step s a* = (*ou*, *s'*)
and *tr*: *validFrom s' tr*
and *s'*: *reach s'*
by (*cases trn*) (*auto iff*: *validFrom-Cons intro*: *reach-PairI*)
then have *vVS-tr*: *validValSeqFrom* (*V tr*) *s'* **by** (*intro Cons.IH*)
show *?case* **proof** *cases*
assume φ: φ (*Trans s a ou s'*)
then have *V*: *V* (*Trans s a ou s'* # *tr*) = *f* (*Trans s a ou s'*) # *V tr* **by** *auto*
from φ *vVS-tr Cons.prems step* **show** *?thesis* **unfolding** *trn V* **by** (*elim* φ*E*) *auto*
next
assume ¬φ (*Trans s a ou s'*)
then have *V* (*Trans s a ou s'* # *tr*) = *V tr* **and** *friends12 s'* = *friends12 s*
and *UID1* ∈∈ *pendingFReqs s' UID2* ⟷ *UID1* ∈∈ *pendingFReqs s UID2*
and *UID2* ∈∈ *pendingFReqs s' UID1* ⟷ *UID2* ∈∈ *pendingFReqs s UID1*
using *step-friends12-*φ[*OF step*] *step-pendingFReqs-*φ[*OF step*] **by** *auto*
with *vVS-tr* **show** *?thesis* **unfolding** *trn* **by** *auto*
qed
qed *auto*

lemma *validFrom istate tr* ⟹ *validValSeqFrom* (*V tr*) *istate*
using *validFrom-validValSeq*[*of istate*] *reach.Istate* **unfolding** *istate-def friends12-def*
by *auto*

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 (*cases u*)
case *U1*
then have *step s* (*Cact* (*cFriendReq UID1* (*pass s UID1*) *UID2 req*)) =
(*outOK*, *createFriendReq s UID1* (*pass s UID1*) *UID2 req*)
and ¬*friends12* (*createFriendReq s UID1* (*pass s UID1*) *UID2 req*)
using *IDs vVS reach-friendIDs-symmetric*[*OF rs*] **by** (*auto simp*: *c-defs friends12-def*)
then show *thesis* **using** *U1 vVS UID1-UID2* **by** (*intro that*[*of - - UID1 UID2*]) (*auto simp*: *c-defs*)

next
case *U2*
then have *step s* (*Cact* (*cFriendReq UID2* (*pass s UID2*) *UID1 req*)) =
(*outOK*, *createFriendReq s UID2* (*pass s UID2*) *UID1 req*)
and ¬*friends12* (*createFriendReq s UID2* (*pass s UID2*) *UID1 req*)
using *IDs vVS reach-friendIDs-symmetric*[*OF rs*] **by** (*auto simp*: *c-defs friends12-def*)
then show *thesis* **using** *U2 vVS UID1-UID2* **by** (*intro that*[*of - - UID2 UID1*]) (*auto simp*: *c-defs*)
qed

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 −
from *vVS* **have** *UID1* ∈∈ *pendingFReqs s UID2* ∨ *UID2* ∈∈ *pendingFReqs s UID1* **by** *auto*
then show *thesis* **proof**
assume *pFR*: *UID1* ∈∈ *pendingFReqs s UID2*
let *?a* = *Cact* (*cFriend UID2* (*pass s UID2*) *UID1*)
let *?s′* = *createFriend s UID1* (*pass s UID1*) *UID2*
let *?trn* = *Trans s ?a outOK ?s′*
have *step*: *step s ?a* = (*outOK*, *?s′*) **using** *IDs pFR UID1-UID2*
unfolding *createFriend-sym*[*of s UID1 pass s UID1 UID2 pass s UID2*]
by (*auto simp add*: *c-defs*)
moreover then have φ *?trn* **and** *f ?trn* = *FVal True* **and** *friends12 ?s′*
and *UID1* ∉ *set* (*pendingFReqs ?s′ UID2*)
and *UID2* ∉ *set* (*pendingFReqs ?s′ UID1*)
using *reach-distinct-friends-reqs*[*OF rs*] **by** (*auto simp*: *c-defs friends12-def*)
moreover have ¬γ *?trn* **using** *UID1-UID2-UIDs* **by** *auto*
ultimately show *thesis* **using** *nf12 rs vVS*
by (*intro that*[*of ?a ?s′ UID2 UID1*]) (*auto intro*: *Cact-cFriend-step-eqButUID*)
next
assume *pFR*: *UID2* ∈∈ *pendingFReqs s UID1*
let *?a* = *Cact* (*cFriend UID1* (*pass s UID1*) *UID2*)
let *?s′* = *createFriend s UID1* (*pass s UID1*) *UID2*

  let *?trn* = *Trans s ?a outOK ?s'*
  have *step*: *step s ?a* = (*outOK*, *?s'*) **using** *IDs pFR UID1-UID2* **by** (*auto simp add*: *c-defs*)
  moreover then have φ *?trn* **and** *f ?trn* = *FVal True* **and** *friends12 ?s'*
      and *UID1* $\notin$ *set* (*pendingFReqs ?s' UID2*)
      and *UID2* $\notin$ *set* (*pendingFReqs ?s' UID1*)
   using *reach-distinct-friends-reqs*[*OF rs*] **by** (*auto simp*: *c-defs friends12-def*)
  moreover have $\neg\gamma$ *?trn* **using** *UID1-UID2-UIDs* **by** *auto*
  ultimately show *thesis* **using** *nf12 rs vVS*
   by (*intro that*[*of ?a ?s' UID1 UID2*]) (*auto intro*: *Cact-cFriend-step-eqButUID*)
 qed
qed

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 $\neg$*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 –
 let *?a* = *Dact* (*dFriend UID1* (*pass s UID1*) *UID2*)
 let *?s'* = *deleteFriend s UID1* (*pass s UID1*) *UID2*
 let *?trn* = *Trans s ?a outOK ?s'*
 have *UID1* $\notin$ *set* (*pendingFReqs s UID2*) *UID2* $\notin$ *set* (*pendingFReqs s UID1*)
  using *f12 reach-distinct-friends-reqs*[*OF rs*] **unfolding** *friends12-def* **by** *auto*
 then have *step*: *step s ?a* = (*outOK*, *?s'*)
    and *UID1* $\notin$ *set* (*pendingFReqs ?s' UID2*) *UID2* $\notin$ *set* (*pendingFReqs ?s' UID1*)
  using *IDs f12 UID1-UID2* **by** (*auto simp add*: *d-defs friends12-def*)
 moreover then have φ *?trn* **and** *f ?trn* = *FVal False* **and** $\neg$*friends12 ?s'*
  using *reach-friendIDs-symmetric*[*OF rs*] **by** (*auto simp*: *d-defs friends12-def*)
 moreover have $\neg\gamma$ *?trn* **using** *UID1-UID2-UIDs* **by** *auto*
 ultimately show *thesis* **using** *f12 rs vVS*
  by (*intro that*[*of ?a ?s'*]) (*auto intro*: *Dact-dFriend-step-eqButUID*)
qed

lemma *toggle-friends12*:
assumes *rs*: *reach s*
  and *IDs*: *IDsOK s* [*UID1*, *UID2*] [] [] []
  and *f12*: *friends12 s* $\neq$ *fv*

and *vVS*: *validValSeqFrom* (*FVal fv* # *vl*) *s*
obtains *a s'*
where *step s a* = (*outOK*, *s'*)
and *friends12 s'* = *fv*
and *eqButUID s s'*
and *φ* (*Trans s a outOK s'*)
and *f* (*Trans s a outOK s'*) = *FVal fv*
and ¬*γ* (*Trans s a outOK s'*)
and *validValSeqFrom vl s'*
proof (*cases friends12 s*)
case *True*
moreover then have *UID1* ∉ *set* (*pendingFReqs s UID2*) *UID2* ∉ *set* (*pendingFReqs s UID1*)
and *fv* = *False*
and *vVS*: *validValSeqFrom* (*FVal False* # *vl*) *s*
using *reach-distinct-friends-reqs*[*OF rs*] *vVS f12* **unfolding** *friends12-def* **by** *auto*
moreover then have *UID1* ∉ *set* (*pendingFReqs* (*deleteFriend s UID1* (*pass s UID1*) *UID2*) *UID2*)
UID2 ∉ *set* (*pendingFReqs* (*deleteFriend s UID1* (*pass s UID1*) *UID2*) *UID1*)
by (*auto simp*: *d-defs*)
ultimately show *thesis* **using** *assms*
by (*elim toggle-friends12-False*, *blast*, *blast*, *blast*) (*elim that*, *blast*+)
next
case *False*
moreover then have *fv* = *True*
and *vVS*: *validValSeqFrom* (*FVal True* # *vl*) *s*
using *vVS f12* **by** *auto*
moreover have *UID1* ∉ *set* (*pendingFReqs* (*createFriend s UID1* (*pass s UID1*) *UID2*) *UID2*)
UID2 ∉ *set* (*pendingFReqs* (*createFriend s UID1* (*pass s UID1*) *UID2*) *UID1*)
using *reach-distinct-friends-reqs*[*OF rs*] **by** (*auto simp*: *c-defs*)
ultimately show *thesis* **using** *assms*
by (*elim toggle-friends12-True*, *blast*, *blast*, *blast*) (*elim that*, *blast*+)
qed

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'*
using *assms* **proof** (*cases rule*: *BO.cases*)
case (*BO-FVal fs*) **then show** *thesis* **by** (*cases fs*) (*auto intro*: *Nil FVal*) **next**
case (*BO-BC vl'' vl1'' fs*) **then show** *thesis* **by** (*cases fs*) (*auto intro*: *FVal*

OVal)
qed

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′*
using *assms* **proof** (*cases rule*: *BC.cases*)
  case (*BC-FVal fs fs1*)
    then show *?thesis* **proof** (*induction fs1*)
      case *Nil* **then show** *?case* **by** (*induction fs*) (*auto intro*: *that*(*1*,*2*)) **next**
      case (*Cons fv fs1′*) **then show** *?case* **by** (*intro that*(*3*)) *auto*
    qed
next
  case (*BC-BO vl′ vl1′ fs fs1*)
    then show *?thesis* **proof** (*cases fs1 rule*: *rev-cases*)
      case *Nil* **then show** *?thesis* **using** *BC-BO* **by** (*intro that*(*7*)) *auto* **next**
      case (*snoc fs1′ fv′*)
        moreover then obtain *fs′* **where** *fs* = *fs′* ## *fv′* **using** *BC-BO*
          by (*induction fs rule*: *rev-induct*) *auto*
        ultimately show *?thesis* **using** *BC-BO* **proof** (*induction fs1′*)
          case *Nil*
            then show *?thesis* **proof** (*induction fs′*)
              case *Nil* **then show** *?thesis* **by** (*intro that*(*6*)) *auto* **next**
              case (*Cons fv″ fs″*) **then show** *?thesis* **by** (*intro that*(*4*)) *auto*
            qed
        next
          case (*Cons fv″ fs1″*) **then show** *?thesis* **by** (*intro that*(*5*)) *auto*
        qed
    qed
qed

definition *Δ0* :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**

$\Delta 0$ *s vl s1 vl1* ≡
s = *s1* ∧ *B vl vl1* ∧ *open s* ∧ (¬*IDsOK s* [*UID1*, *UID2*] [] [] [])

definition $\Delta 1$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 1$ *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 $\Delta 2$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 2$ *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 $\Delta 3$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 3$ *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 $\Delta 2$ *s vl s1 vl1*
using *assms* **unfolding** Δ*2-def* **by** *blast*

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 $\Delta 3$ *s vl s1 vl1*
using *assms* **unfolding** Δ*3-def* **by** *blast*

lemma *istate-*$\Delta 0$:
assumes *B*: *B vl vl1*

shows *Δ0 istate vl istate vl1*
using *assms* **unfolding** *Δ0-def istate-def B-def open-def openByA-def openByF-def friends12-def*
by *auto*

lemma *unwind-cont-Δ0*: *unwind-cont Δ0* {*Δ0*,*Δ1*,*Δ2*,*Δ3*}
proof(*rule*, *simp*)
  let *?Δ* = *λs vl s1 vl1*. *Δ0 s vl s1 vl1* ∨
                         Δ1 s vl s1 vl1 ∨
                         Δ2 s vl s1 vl1 ∨
                         Δ3 s vl s1 vl1
  fix *s s1* :: *state* **and** *vl vl1* :: *value list*
  assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *Δ0*: *Δ0 s vl s1 vl1*
  then have *rs*: *reach s* **and** *ss1*: *s1* = *s* **and** *B*: *B vl vl1* **and** *os*: *open s*
      and *IDs*: ¬*IDsOK s* [*UID1*, *UID2*] [] [] []
    using *reachNT-reach* **unfolding** *Δ0-def* **by** *auto*
  from *IDs* **have** *UID1* ∉ *set* (*pendingFReqs s UID2*) **and** ¬*friends12 s*
        and *UID2* ∉ *set* (*pendingFReqs s UID1*)
    using *reach-IDs-used-IDsOK*[*OF rs*] **unfolding** *friends12-def* **by** *auto*
  with *B* **have** *BO*: *BO* (*filter* (*Not* ∘ *isFRVal*) *vl*) (*filter* (*Not* ∘ *isFRVal*) *vl1*)
        and *vl-vl1*: *vl* = [] ⟶ *vl1* = []
        and *vl-OVal*: *vl* ≠ [] ∧ *hd vl* = *OVal False* ⟶ *vl1* ≠ [] ∧ *hd vl1* = *OVal False*
        and *vVS*: *validValSeqFrom vl1 s*
    unfolding *B-def* **by** (*auto simp*: *istate-def friends12-def*)
  show *iaction ?Δ s vl s1 vl1* ∨
      ((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
  proof −
    have *?react* **proof**
      fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
      let *?trn* = *Trans s a ou s'*
     assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
     show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
      proof *cases*
        assume *φ*: *φ ?trn*
        then obtain *uid p uid' p'* **where** *a*: *a* = *Cact* (*cUser uid p uid' p'*)
                          ¬*openByA s'* ¬*openByF s'*
                          ou = *outOK f ?trn* = *OVal False*
                          friends12 s' = *friends12 s*
                            UID1 ∈∈ *pendingFReqs s' UID2* ⟷ *UID1* ∈∈ *pendingFReqs s UID2*
                            UID2 ∈∈ *pendingFReqs s' UID1* ⟷ *UID2* ∈∈ *pendingFReqs s UID1*
          using *step rs IDs* **by** (*elim φE*) (*auto simp*: *openByA-def*)
        with *c φ* **have** *vl*: *vl* = *OVal False* # *vl'* **unfolding** *consume-def* **by** *auto*
        with *vl-OVal* **obtain** *vl1'* **where** *vl1*: *vl1* = *OVal False* # *vl1'* **by** (*cases vl1*) *auto*
        from *BO vl vl1* **have** *BC'*: *BC* (*filter* (*Not* ∘ *isFRVal*) *vl'*) (*filter* (*Not* ∘

isFRVal) *vl1'*)
by (*cases rule*: *BO-cases*) *auto*
then have $\Delta 2$ *s' vl' s' vl1'* ∨ $\Delta 3$ *s' vl' s' vl1'* **using** *vVS a* **unfolding** *vl1*
proof (*cases rule*: *BC.cases*)
case *BC-FVal*
then show *?thesis* **using** *vVS a* **unfolding** *vl1*
by (*intro disjI1* $\Delta 2$*-I*) (*auto simp*: *open-def*)
next
case *BC-BO*
then show *?thesis* **using** *vVS a* **unfolding** *vl1*
by (*intro disjI2* $\Delta 3$*-I*) (*auto simp*: *open-def*)
qed
then have *?match* **using** *step a* φ **unfolding** *ss1 vl1*
by (*intro matchI*[*of s a ou s'*]) (*auto simp*: *consume-def*)
then show *?thesis* **..**
next
assume *n*φ: ¬φ *?trn*
then have *s'*: *open s' friends12 s' = friends12 s*
UID1 ∈∈ *pendingFReqs s' UID2* ⟷ *UID1* ∈∈ *pendingFReqs s UID2*
UID2 ∈∈ *pendingFReqs s' UID1* ⟷ *UID2* ∈∈ *pendingFReqs s UID1*
using *os step-open-*φ[*OF step*] *step-friends12-*φ[*OF step*] *step-pendingFReqs-*φ[*OF step*]
by *auto*
moreover have *vl' = vl* **using** *n*φ *c* **by** (*auto simp*: *consume-def*)
ultimately have $\Delta 0$ *s' vl' s' vl1* ∨ $\Delta 1$ *s' vl' s' vl1*
using *vVS B BO* **unfolding** $\Delta 0$*-def* $\Delta 1$*-def*
by (*cases IDsOK s'* [*UID1*, *UID2*] [] [] []) *auto*
then have *?match* **using** *step c n*φ **unfolding** *ss1*
by (*intro matchI*[*of s a ou s'*]) (*auto simp*: *consume-def*)
then show *?thesis* **..**
qed
qed
then show *?thesis* **using** *vl-vl1* **by** *auto*
qed
qed

lemma *unwind-cont-*$\Delta 1$: *unwind-cont* $\Delta 1$ {$\Delta 1$,$\Delta 2$,$\Delta 3$}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. $\Delta 1$ *s vl s1 vl1* ∨
$\Delta 2$ *s vl s1 vl1* ∨
$\Delta 3$ *s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** $\Delta 1$: $\Delta 1$ *s vl s1 vl1*
then have *rs*: *reach s* **and** *ss1*: *eqButUID s s1* **and** *fIDs*: *friendIDs s = friendIDs s1*
and *os*: *open s* **and** *BO*: *BO* (*filter* (*Not o isFRVal*) *vl*) (*filter* (*Not o isFRVal*) *vl1*)

and *vVS1*: *validValSeq vl1* (*friends12 s1*)
(*UID1* ∈∈ *pendingFReqs s1 UID2*)
(*UID2* ∈∈ *pendingFReqs s1 UID1*) (**is** *?vVS vl1 s1*)
and *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] []
using *reachNT-reach* **unfolding** *Δ1-def* **by** *auto*
show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof *cases*
assume ∃ *u req vl1'*. *vl1* = *FRVal u req* # *vl1'*
then obtain *u req vl1'* **where** *vl1*: *vl1* = *FRVal u req* # *vl1'* **by** *auto*
obtain *a uid uid' s1'* **where** *step1*: *step s1 a* = (*outOK*, *s1'*) **and** φ (*Trans s1 a outOK s1'*)
and *a*: *a* = *Cact* (*cFriendReq uid* (*pass s1 uid*) *uid' req*)
and *uid*: *uid* = *UID1* ∧ *uid'* = *UID2* ∨ *uid* = *UID2* ∧ *uid'* = *UID1*
and *f* (*Trans s1 a outOK s1'*) = *FRVal u req* **and** *?vVS vl1' s1'*
using *rs1 IDs1 vVS1 UID1-UID2-UIDs* **unfolding** *vl1* **by** (*blast intro*: *produce-FRVal*)
moreover then have ¬γ (*Trans s1 a outOK s1'*) **using** *UID1-UID2-UIDs* **by** *auto*
moreover have *eqButUID s1 s1'* **using** *step1 a uid* **by** (*auto intro*: *Cact-cFriendReq-step-eqButUID*)
moreover have *friendIDs s1'* = *friendIDs s1* **and** *IDsOK s1'* [*UID1*, *UID2*] [] [] []
using *step1 a uid* **by** (*auto simp*: *c-defs*)
ultimately have *?iact* **using** *ss1 fIDs os BO* **unfolding** *vl1*
by (*intro iactionI*[*of s1 a outOK s1'*]) (*auto simp*: *consume-def Δ1-def intro*: *eqButUID-trans*)
then show *?thesis* **..**
next
assume *nFRVal1*: ¬ (∃ *u req vl1'*. *vl1* = *FRVal u req* # *vl1'*)
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
show *match ?Δ s s1 vl1 a ou s' vl'* ∨ *ignore ?Δ s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
proof *cases*
assume φ: φ *?trn*
then have *vl*: *vl* = *f ?trn* # *vl'* **using** *c* **by** (*auto simp*: *consume-def*)
from *BO* **show** *?thesis* **proof** (*cases f ?trn*)
case (*FVal fv*)
with *BO* **obtain** *vl1'* **where** *vl1*: *vl1* = *f ?trn* # *vl1'*
using *BO-Nil-iff*[*OF BO*] *FVal vl nFRVal1*
by (*cases rule*: *BO-cases*; *cases vl1*; *cases hd vl1*) *auto*
with *BO* **have** *BO'*: *BO* (*filter* (*Not o isFRVal*) *vl'*) (*filter* (*Not o isFRVal*) *vl1'*)
using *FVal vl* **by** (*cases rule*: *BO-cases*) *auto*
from *fIDs* **have** *f12*: *friends12 s* = *friends12 s1* **unfolding** *friends12-def*

by *auto*
have *?match* **using** *φ step rs FVal* **proof** (*cases rule*: *φE*)
case (*Friend uid p uid′*)
then have *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] []
using *ss1* **unfolding** *eqButUID-def* **by** *auto*
let *?s1′* = *createFriend s1 UID1* (*pass s1 UID1*) *UID2*
have *s′*: *s′* = *createFriend s UID1 p UID2*
using *Friend step* **by** (*auto simp*: *createFriend-sym*)
have *ss′*: *eqButUID s s′* **using** *rs step Friend*
by (*auto intro*: *Cact-cFriend-step-eqButUID*)
moreover then have *os′*: *open s′* **using** *os eqButUID-open-eq* **by** *auto*
moreover obtain *a1 uid1 uid1′ p1*
where *step s1 a1* = (*outOK*, *?s1′*) *friends12 ?s1′*
a1 = *Cact* (*cFriend uid1 p1 uid1′*)
uid1 = *UID1* ∧ *uid1′* = *UID2* ∨ *uid1* = *UID2* ∧ *uid1′* = *UID1*
φ (*Trans s1 a1 outOK ?s1′*)
f (*Trans s1 a1 outOK ?s1′*) = *FVal True*
eqButUID s1 ?s1′ ?vVS vl1′ ?s1′
using *rs1 IDs1 Friend vVS1* **unfolding** *vl1 f12 Friend*(*3*)
by (*elim toggle-friends12-True*) *blast+*
moreover then have *IDsOK ?s1′* [*UID1*, *UID2*] [] [] [] **by** (*auto simp*: *c-defs*)
moreover have *friendIDs s′* = *friendIDs ?s1′*
using *Friend*(*6*) *f12* **unfolding** *s′*
by (*intro eqButUID-createFriend12-friendIDs-eq*[*OF ss1 rs rs1*]) *auto*
ultimately show *?match* **using** *ss1 BO′ Friend UID1-UID2-UIDs* **unfolding** *vl1 Δ1-def*
by (*intro matchI*[*of s1 a1 outOK ?s1′*])
(*auto simp*: *consume-def intro*: *eqButUID-trans eqButUID-sym*)
next
case (*Unfriend uid p uid′*)
then have *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] []
using *ss1* **unfolding** *eqButUID-def* **by** *auto*
let *?s1′* = *deleteFriend s1 UID1* (*pass s1 UID1*) *UID2*
have *s′*: *s′* = *deleteFriend s UID1 p UID2*
using *Unfriend step* **by** (*auto simp*: *deleteFriend-sym*)
have *ss′*: *eqButUID s s′* **using** *rs step Unfriend*
by (*auto intro*: *Dact-dFriend-step-eqButUID*)
moreover then have *os′*: *open s′* **using** *os eqButUID-open-eq* **by** *auto*
moreover obtain *a1 uid1 uid1′ p1*
where *step s1 a1* = (*outOK*, *?s1′*) ¬*friends12 ?s1′*
a1 = *Dact* (*dFriend uid1 p1 uid1′*)
uid1 = *UID1* ∧ *uid1′* = *UID2* ∨ *uid1* = *UID2* ∧ *uid1′* = *UID1*
φ (*Trans s1 a1 outOK ?s1′*)
f (*Trans s1 a1 outOK ?s1′*) = *FVal False*
eqButUID s1 ?s1′ ?vVS vl1′ ?s1′
using *rs1 IDs1 Unfriend vVS1* **unfolding** *vl1 f12 Unfriend*(*3*)

by (*elim toggle-friends12-False*) *blast+*
moreover have *friendIDs s′ = friendIDs ?s1′ IDsOK ?s1′* [*UID1*, *UID2*] [] [] []
using *fIDs IDs1* **unfolding** *s′* **by** (*auto simp*: *d-defs*)
ultimately show *?match* **using** *ss1 BO′ Unfriend UID1-UID2-UIDs* **unfolding** *vl1 Δ1-def*
by (*intro matchI*[*of s1 a1 outOK ?s1′*])
(*auto simp*: *consume-def intro*: *eqButUID-trans eqButUID-sym*)
qed *auto*
then show *?thesis* **..**
next
case (*OVal ov*)
with *BO* **obtain** *vl1′* **where** *vl1′*: *vl1 = OVal False # vl1′*
using *BO-Nil-iff*[*OF BO*] *OVal vl nFRVal1*
by (*cases rule*: *BO-cases*; *cases vl1*; *cases hd vl1*) *auto*
with *BO* **have** *BC′*: *BC* (*filter* (*Not o isFRVal*) *vl′*) (*filter* (*Not o isFRVal*) *vl1′*)
using *OVal vl* **by** (*cases rule*: *BO-cases*) *auto*
from *BO vl OVal* **have** *f ?trn = OVal False* **by** (*cases rule*: *BO-cases*) *auto*
with *φ step rs* **have** *?match* **proof** (*cases rule*: *φE*)
case (*CloseF uid p uid′*)
let *?s1′ = deleteFriend s1 uid p uid′*
let *?trn1 = Trans s1 a outOK ?s1′*
have *s′*: *s′ = deleteFriend s uid p uid′* **using** *CloseF step* **by** *auto*
have *step1*: *step s1 a =* (*outOK*, *?s1′*)
and *pFR1′*: *pendingFReqs ?s1′ = pendingFReqs s1*
using *CloseF step ss1 fIDs* **unfolding** *eqButUID-def* **by** (*auto simp*: *d-defs*)
have *s′s1′*: *eqButUID s′ ?s1′* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*] **.**
moreover have *os′*: *¬open s′* **using** *CloseF os* **unfolding** *open-def* **by** *auto*
moreover have *fIDs′*: *friendIDs s′ = friendIDs ?s1′*
using *fIDs* **unfolding** *s′* **by** (*auto simp*: *d-defs*)
moreover have *f12s1*: *friends12 s1 = friends12 ?s1′*
using *CloseF*(*2*) *UID1-UID2-UIDs* **unfolding** *friends12-def d-defs* **by** *auto*
from *BC′* **have** *Δ2 s′ vl′ ?s1′ vl1′ ∨ Δ3 s′ vl′ ?s1′ vl1′*
proof (*cases rule*: *BC.cases*)
case (*BC-FVal fs fs1*)
then show *?thesis* **using** *vVS1 os′ fIDs′ f12s1 s′s1′ pFR1′*
unfolding *Δ2-def vl1′* **by** *auto*
next
case (*BC-BO vlr vlr1 fs fs1*)
then have *Δ3 s′ vl′ ?s1′ vl1′* **using** *s′s1′ os′ vVS1 f12s1 fIDs′ pFR1′*
unfolding *vl1′* **by** (*intro Δ3-I*[*of - - - - - fs fs1*]) *auto*
then show *?thesis* **..**

qed
moreover have *open s1* ¬*open ?s1'*
using *ss1 os s's1' os'* **by** (*auto simp*: *eqButUID-open-eq*)
moreover then have φ *?trn1* **unfolding** *CloseF* **by** *auto*
ultimately show *?match* **using** *step1 vl1' CloseF UID1-UID2 UID1-UID2-UIDs*
by (*intro matchI*[*of s1 a outOK ?s1' vl1 vl1'*]) (*auto simp*: *consume-def*)
next
case (*CloseA uid p uid' p'*)
let *?s1'* = *createUser s1 uid p uid' p'*
let *?trn1* = *Trans s1 a outOK ?s1'*
have *s'*: *s'* = *createUser s uid p uid' p'* **using** *CloseA step* **by** *auto*
have *step1*: *step s1 a* = (*outOK*, *?s1'*)
and *pFR1'*: *pendingFReqs ?s1'* = *pendingFReqs s1*
using *CloseA step ss1* **unfolding** *eqButUID-def* **by** (*auto simp*: *c-defs*)
have *s's1'*: *eqButUID s' ?s1'* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*] **.**
moreover have *os'*: ¬*open s'* **using** *CloseA os* **unfolding** *open-def* **by** *auto*
moreover have *fIDs'*: *friendIDs s'* = *friendIDs ?s1'*
using *fIDs* **unfolding** *s'* **by** (*auto simp*: *c-defs*)
moreover have *f12s1*: *friends12 s1* = *friends12 ?s1'*
unfolding *friends12-def* **by** (*auto simp*: *c-defs*)
from *BC'* **have** Δ*2 s' vl' ?s1' vl1'* ∨ Δ*3 s' vl' ?s1' vl1'*
proof (*cases rule*: *BC.cases*)
case (*BC-FVal fs fs1*)
then show *?thesis* **using** *vVS1 os' fIDs' f12s1 s's1' pFR1'*
unfolding Δ*2-def vl1'* **by** *auto*
next
case (*BC-BO vlr vlr1 fs fs1*)
then have Δ*3 s' vl' ?s1' vl1'* **using** *s's1' os' vVS1 f12s1 fIDs' pFR1'*
unfolding *vl1'* **by** (*intro* Δ*3-I*[*of - - - - - fs fs1*]) *auto*
then show *?thesis* **..**
qed
moreover have *open s1* ¬*open ?s1'*
using *ss1 os s's1' os'* **by** (*auto simp*: *eqButUID-open-eq*)
moreover then have φ *?trn1* **unfolding** *CloseA* **by** *auto*
ultimately show *?match* **using** *step1 vl1' CloseA UID1-UID2 UID1-UID2-UIDs*
by (*intro matchI*[*of s1 a outOK ?s1' vl1 vl1'*]) (*auto simp*: *consume-def*)
qed *auto*
then show *?thesis* **..**
next
case (*FRVal u req*)
obtain *p*

where *a*: (*a* = *Cact* (*cFriendReq UID1 p UID2 req*) ∧ *UID1* ∈∈ *pendingFReqs s′ UID2* ∧
UID1 ∉ *set* (*pendingFReqs s UID2*) ∧
(*UID2* ∈∈ *pendingFReqs s′ UID1* ⟷ *UID2* ∈∈ *pendingFReqs s UID1*)) ∨
(*a* = *Cact* (*cFriendReq UID2 p UID1 req*) ∧ *UID2* ∈∈ *pendingFReqs s′ UID1* ∧
UID2 ∉ *set* (*pendingFReqs s UID1*) ∧
(*UID1* ∈∈ *pendingFReqs s′ UID2* ⟷ *UID1* ∈∈ *pendingFReqs s UID2*))
ou = *outOK* ¬*friends12 s* ¬*friends12 s′ open s′* = *open s*
using *φ step rs FRVal* **by** (*cases rule*: *φE*) *fastforce*+
then have *fIDs′*: *friendIDs s′* = *friendIDs s* **using** *step* **by** (*auto simp*: *c-defs*)
have *eqButUID s s′* **using** *a step*
by (*auto intro*: *Cact-cFriendReq-step-eqButUID*)
then have Δ*1 s′ vl′ s1 vl1*
unfolding Δ*1-def* **using** *ss1 fIDs′ fIDs os a*(*5*) *vVS1 IDs1 BO vl FRVal*
by (*auto intro*: *eqButUID-trans eqButUID-sym*)
moreover from *φ step rs a* **have** ¬*γ* (*Trans s a ou s′*)
using *UID1-UID2-UIDs* **by** (*cases rule*: *φE*) *auto*
ultimately have *?ignore* **by** (*intro ignoreI*) *auto*
then show *?thesis* **..**
qed
next
assume *nφ*: ¬*φ ?trn*
then have *os′*: *open s* = *open s′* **and** *f12s′*: *friends12 s* = *friends12 s′*
using *step-open-φ*[*OF step*] *step-friends12-φ*[*OF step*] **by** *auto*
have *vl′*: *vl′* = *vl* **using** *nφ c* **by** (*auto simp*: *consume-def*)
show *?thesis* **proof** (*cases* ∀ *req. a* ≠ *Cact* (*cFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Cact* (*cFriend UID2* (*pass s UID2*) *UID1*) ∧
a ≠ *Cact* (*cFriendReq UID2* (*pass s UID2*) *UID1 req*) ∧
a ≠ *Cact* (*cFriendReq UID1* (*pass s UID1*) *UID2 req*) ∧
a ≠ *Dact* (*dFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Dact* (*dFriend UID2* (*pass s UID2*) *UID1*))
case *True*
obtain *ou1 s1′* **where** *step1*: *step s1 a* = (*ou1*, *s1′*) **by** (*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1′*
have *fIDs′*: *friendIDs s′* = *friendIDs s1′* **using** *True*
by (*intro eqButUID-step-friendIDs-eq*[*OF ss1 rs rs1 step step1 - fIDs*]) *auto*
from *True nφ* **have** *nφ′*: ¬*φ ?trn1* **using** *eqButUID-step-φ*[*OF ss1 rs rs1 step step1*] **by** *auto*
then have *f12s1′*: *friends12 s1* = *friends12 s1′*
and *pFRs′*: *UID1* ∈∈ *pendingFReqs s1 UID2* ⟷ *UID1* ∈∈

pendingFReqs s1′ UID2
UID2 ∈∈ *pendingFReqs s1 UID1* ⟷ *UID2* ∈∈ *pendingFReqs s1′ UID1*
using *step-friends12-φ*[*OF step1*] *step-pendingFReqs-φ*[*OF step1*]
by *auto*
have *eqButUID s′ s1′* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*] **.**
then have Δ*1 s′ vl′ s1′ vl1* **using** *os fIDs′ vVS1 BO IDsOK-mono*[*OF step1 IDs1*]
unfolding Δ*1-def os′ f12s1′ pFRs′ vl′* **by** *auto*
then have *?match*
using *step1 nφ′ fIDs eqButUID-step-γ-out*[*OF ss1 step step1*]
by (*intro matchI*[*of s1 a ou1 s1′ vl1 vl1*]) (*auto simp*: *consume-def*)
then show *?match* ∨ *?ignore* **..**
next
case *False*
with *nφ* **have** *ou* ≠ *outOK* **by** *auto*
then have *s′* = *s* **using** *step False* **by** *auto*
then have *?ignore* **using** Δ*1 False UID1-UID2-UIDs* **unfolding** *vl′* **by** (*intro ignoreI*) *auto*
then show *?match* ∨ *?ignore* **..**
qed
qed
qed
moreover have *vl* = [] ⟶ *vl1* = [] **proof**
assume *vl* = []
with *BO* **have** *filter* (*Not* ∘ *isFRVal*) *vl1* = [] **using** *BO-Nil-iff*[*OF BO*] **by** *auto*
with *nFRVal1* **show** *vl1* = [] **by** (*cases vl1*; *cases hd vl1*) *auto*
qed
ultimately show *?thesis* **by** *auto*
qed
qed

lemma *unwind-cont-*Δ*2*: *unwind-cont* Δ*2* {Δ*2*, Δ*1*}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. Δ*2 s vl s1 vl1* ∨ Δ*1 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *2*: Δ*2 s vl s1 vl1*
from *rsT* **have** *rs*: *reach s* **by** (*intro reachNT-reach*)
from *2* **obtain** *fs fs1*
where *ss1*: *eqButUID s s1* **and** *os*: ¬*open s*
and *vVS1*: *validValSeqFrom vl1 s1*
and *fs*: *filter* (*Not o isFRVal*) *vl* = *map FVal fs*
and *fs1*: *filter* (*Not o isFRVal*) *vl1* = *map FVal fs1*
unfolding Δ*2-def* **by** *auto*
from *os* **have** *IDs*: *IDsOK s* [*UID1*, *UID2*] [] [] [] **unfolding** *open-defs* **by** *auto*
then have *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] [] **using** *ss1* **unfolding** *eqButUID-def* **by** *auto*
show *iaction ?*Δ *s vl s1 vl1* ∨

((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof *cases*
assume *vl1*: *vl1* = []
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn* = *Trans s a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl′*
show *match ?Δ s s1 vl1 a ou s′ vl′* ∨ *ignore ?Δ s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
proof *cases*
assume *φ*: *φ ?trn*
with *c* **have** *vl*: *vl* = *f ?trn* # *vl′* **by** (*auto simp*: *consume-def*)
with *fs* **have** *?ignore* **proof** (*cases f ?trn*)
case (*FRVal u req*)
obtain *p*
where *a*: (*a* = *Cact* (*cFriendReq UID1 p UID2 req*) ∧ *UID1* ∈∈ *pendingFReqs s′ UID2* ∧
UID1 ∉ *set* (*pendingFReqs s UID2*) ∧
(*UID2* ∈∈ *pendingFReqs s′ UID1* ⟷ *UID2* ∈∈ *pendingFReqs s UID1*)) ∨
(*a* = *Cact* (*cFriendReq UID2 p UID1 req*) ∧ *UID2* ∈∈ *pendingFReqs s′ UID1* ∧
UID2 ∉ *set* (*pendingFReqs s UID1*) ∧
(*UID1* ∈∈ *pendingFReqs s′ UID2* ⟷ *UID1* ∈∈ *pendingFReqs s UID2*))
ou = *outOK* ¬*friends12 s* ¬*friends12 s′ open s′* = *open s*
using *φ step rs FRVal* **by** (*cases rule*: *φE*) *fastforce+*
then have *fIDs′*: *friendIDs s′* = *friendIDs s* **using** *step* **by** (*auto simp*: *c-defs*)
have *eqButUID s s′* **using** *a step*
by (*auto intro*: *Cact-cFriendReq-step-eqButUID*)
then have Δ*2 s′ vl′ s1 vl1*
unfolding Δ*2-def* **using** *ss1 os a*(*5*) *vVS1 vl fs fs1*
by (*auto intro*: *eqButUID-trans eqButUID-sym*)
moreover from *φ step rs a* **have** ¬*γ* (*Trans s a ou s′*)
using *UID1-UID2-UIDs* **by** (*cases rule*: *φE*) *auto*
ultimately show *?ignore* **by** (*intro ignoreI*) *auto*
next
case (*FVal fv*)
with *fs vl* **obtain** *fs′* **where** *fs′*: *fs* = *fv* # *fs′* **by** (*cases fs*) *auto*
from *φ step rs FVal* **have** *ss′*: *eqButUID s s′*
by (*elim φE*) (*auto intro*: *Cact-cFriend-step-eqButUID Dact-dFriend-step-eqButUID*)
then have ¬*open s′* **using** *os* **by** (*auto simp*: *eqButUID-open-eq*)
moreover have *eqButUID s′ s1* **using** *ss1 ss′* **by** (*auto intro*: *eqButUID-sym eqButUID-trans*)
ultimately have Δ*2 s′ vl′ s1 vl1*
using *vVS1 fs′ fs* **unfolding** Δ*2-def vl vl1 FVal* **by** *auto*
moreover have ¬*γ ?trn* **using** *φ step rs FVal UID1-UID2-UIDs* **by** (*elim φE*) *auto*

ultimately show *?ignore* **by** (*intro ignoreI*) *auto*
qed *auto*
then show *?thesis* **..**
next
assume *nφ*: ¬*φ ?trn*
then have *os'*: *open s = open s'* **and** *f12s'*: *friends12 s = friends12 s'*
using *step-open-φ*[*OF step*] *step-friends12-φ*[*OF step*] **by** *auto*
have *vl'*: *vl' = vl* **using** *nφ c* **by** (*auto simp*: *consume-def*)
show *?thesis* **proof** (*cases* ∀ *req. a* ≠ *Cact* (*cFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Cact* (*cFriend UID2* (*pass s UID2*) *UID1*) ∧
a ≠ *Cact* (*cFriendReq UID2* (*pass s UID2*) *UID1 req*) ∧
a ≠ *Cact* (*cFriendReq UID1* (*pass s UID1*) *UID2 req*) ∧
a ≠ *Dact* (*dFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Dact* (*dFriend UID2* (*pass s UID2*) *UID1*))
case *True*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by** (*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
from *True nφ* **have** *nφ'*: ¬*φ ?trn1*
using *eqButUID-step-φ*[*OF ss1 rs rs1 step step1*] **by** *auto*
then have *f12s1'*: *friends12 s1* = *friends12 s1'*
and *pFRs'*: *UID1* ∈∈ *pendingFReqs s1 UID2* ⟷ *UID1* ∈∈ *pendingFReqs s1' UID2*
UID2 ∈∈ *pendingFReqs s1 UID1* ⟷ *UID2* ∈∈ *pendingFReqs s1' UID1*
using *step-friends12-φ*[*OF step1*] *step-pendingFReqs-φ*[*OF step1*]
by *auto*
have *eqButUID s' s1'* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*] **.**
then have Δ*2 s' vl' s1' vl1* **using** *os vVS1 fs fs1*
unfolding Δ*2-def os' f12s1' pFRs' vl'* **by** *auto*
then have *?match*
using *step1 nφ' os eqButUID-step-γ-out*[*OF ss1 step step1*]
by (*intro matchI*[*of s1 a ou1 s1' vl1 vl1*]) (*auto simp*: *consume-def*)
then show *?match* ∨ *?ignore* **..**
next
case *False*
with *nφ* **have** *ou* ≠ *outOK* **by** *auto*
then have *s'* = *s* **using** *step False* **by** *auto*
then have *?ignore* **using** *2 False UID1-UID2-UIDs* **unfolding** *vl'* **by** (*intro ignoreI*) *auto*
then show *?match* ∨ *?ignore* **..**
qed
qed
qed
then show *?thesis* **using** *vl1* **by** *auto*
next

assume *vl1* ≠ []
then obtain *v vl1′* **where** *vl1*: *vl1* = *v* # *vl1′* **by** (*cases vl1*) *auto*
with *fs1* **have** *?iact* **proof** (*cases v*)
case (*FRVal u req*)
obtain *a uid uid′ s1′* **where** *step1*: *step s1 a* = (*outOK*, *s1′*) **and** φ (*Trans s1 a outOK s1′*)
and *a*: *a* = *Cact* (*cFriendReq uid* (*pass s1 uid*) *uid′ req*)
and *uid*: *uid* = *UID1* ∧ *uid′* = *UID2* ∨ *uid* = *UID2* ∧ *uid′* = *UID1*
and *f* (*Trans s1 a outOK s1′*) = *FRVal u req*
and *vVS1′*: *validValSeqFrom vl1′ s1′*
using *rs1 IDs1 vVS1 UID1-UID2-UIDs* **unfolding** *vl1 FRVal* **by** (*blast intro*: *produce-FRVal*)
moreover then have ¬γ (*Trans s1 a outOK s1′*) **using** *UID1-UID2-UIDs* **by** *auto*
moreover have *eqButUID s1 s1′* **using** *step1 a uid*
by (*auto intro*: *Cact-cFriendReq-step-eqButUID*)
moreover then have Δ*2 s vl s1′ vl1′* **using** *ss1 os vVS1′ fs fs1* **unfolding** *vl1 FRVal*
by (*intro* Δ*2-I*[*of s s1′ vl1′ vl fs fs1*]) (*auto intro*: *eqButUID-trans*)
ultimately show *?iact* **using** *ss1 os* **unfolding** *vl1 FRVal*
by (*intro iactionI*[*of s1 a outOK s1′*]) (*auto simp*: *consume-def intro*: *eqButUID-trans*)
next
case (*FVal fv*)
then obtain *fs1′* **where** *fs1′*: *fs1* = *fv* # *fs1′*
using *vl1 fs1* **by** (*cases fs1*) *auto*
from *FVal vVS1 vl1* **have** *f12*: *friends12 s1* ≠ *fv*
and *vVS1*: *validValSeqFrom* (*FVal fv* # *vl1′*) *s1* **by** *auto*
then show *?iact* **using** *rs1 IDs1 vl1 FVal ss1 os fs fs1 fs1′ vl1 FVal*
by (*elim toggle-friends12*[*of s1 fv vl1′*], *blast*, *blast*, *blast*)
(*intro iactionI*[*of s1 - - - vl1 vl1′*],
auto simp: *consume-def intro*: Δ*2-I*[*of s - vl1′ vl fs fs1′*] *eqButUID-trans*)
qed *auto*
then show *?thesis* **..**
qed
qed

lemma *unwind-cont-*Δ*3*: *unwind-cont* Δ*3* {Δ*3*,Δ*1*}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. Δ*3 s vl s1 vl1* ∨ Δ*1 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *3*: Δ*3 s vl s1 vl1*
from *rsT* **have** *rs*: *reach s* **by** (*intro reachNT-reach*)
obtain *fs fs1 vlr vlr1*
where *ss1*: *eqButUID s s1* **and** *os*: ¬*open s* **and** *BO*: *BO vlr vlr1*
and *vVS1*: *validValSeqFrom vl1 s1*

and *fs*: *filter* (*Not o isFRVal*) *vl* = *map FVal fs* @ *OVal True* # *vlr*
and *fs1*: *filter* (*Not o isFRVal*) *vl1* = *map FVal fs1* @ *OVal True* # *vlr1*
and *fs-fs1*: *fs* = [] ⟷ *fs1* = []
and *last-fs*: *fs* ≠ [] ⟶ *last fs* = *last fs1*
and *fs-fIDs*: *fs* = [] ⟶ *friendIDs s* = *friendIDs s1*
using *3* **unfolding** Δ*3-def* **by** *auto*
have *BC*: *BC* (*map FVal fs* @ *OVal True* # *vlr*) (*map FVal fs1* @ *OVal True* # *vlr1*)
using *fs fs1 fs-fs1 last-fs BO* **by** *auto*
from *os* **have** *IDs*: *IDsOK s* [*UID1*, *UID2*] [] [] [] **unfolding** *open-defs* **by** *auto*
then have *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] [] **using** *ss1* **unfolding** *eqButUID-def* **by** *auto*
show *iaction ?*Δ *s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof *cases*
assume ∃ *u req vl1'*. *vl1* = *FRVal u req* # *vl1'*
then obtain *u req vl1'* **where** *vl1*: *vl1* = *FRVal u req* # *vl1'* **by** *auto*
obtain *a uid uid' s1'* **where** *step1*: *step s1 a* = (*outOK*, *s1'*) **and** φ: φ (*Trans s1 a outOK s1'*)
and *a*: *a* = *Cact* (*cFriendReq uid* (*pass s1 uid*) *uid' req*)
and *uid*: *uid* = *UID1* ∧ *uid'* = *UID2* ∨ *uid* = *UID2* ∧ *uid'* = *UID1*
and *f*: *f* (*Trans s1 a outOK s1'*) = *FRVal u req*
and *validValSeqFrom vl1' s1'*
using *rs1 IDs1 vVS1 UID1-UID2-UIDs* **unfolding** *vl1* **by** (*blast intro*: *produce-FRVal*)
moreover have *eqButUID s1 s1'* **using** *step1 a uid* **by** (*auto intro*: *Cact-cFriendReq-step-eqButUID*)
moreover have *friendIDs s1'* = *friendIDs s1* **and** *IDsOK s1'* [*UID1*, *UID2*] [] [] []
using *step1 a uid* **by** (*auto simp*: *c-defs*)
ultimately have Δ*3 s vl s1' vl1'* **using** *ss1 os BO fs-fs1 last-fs fs-fIDs fs fs1* **unfolding** *vl1*
by (*intro* Δ*3-I*[*of - - vlr vlr1 vl1' fs fs1 vl*])
(*auto simp*: *consume-def intro*: *eqButUID-trans*)
moreover have ¬γ (*Trans s1 a outOK s1'*) **using** *a uid UID1-UID2-UIDs* **by** *auto*
ultimately have *?iact* **using** *step1* φ *f* **unfolding** *vl1*
by (*intro iactionI*[*of s1 a outOK s1'*]) (*auto simp*: *consume-def*)
then show *?thesis* **..**
next
assume *nFRVal1*: ¬(∃ *u req vl1'*. *vl1* = *FRVal u req* # *vl1'*)
from *BC* **show** *?thesis* **proof** (*cases rule*: *BC-cases*)
case (*BO-FVal fv fv' fs' vl'' vl1''*)
then have *fs'*: *filter* (*Not o isFRVal*) *vl* = *map FVal* (*fv* # *fs'* ## *fv'*) @ *OVal True* # *vl''*
and *fs1'*: *filter* (*Not o isFRVal*) *vl1* = *FVal fv'* # *OVal True* # *vl1''*
using *fs fs1* **by** *auto*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*

let *?trn* = *Trans s a ou s'* **let** *?trn1* = *Trans s1 a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: $\neg$ *T ?trn* **and** *c*: *consume ?trn vl vl'*
show *match ?*Δ *s s1 vl1 a ou s' vl'* $\vee$ *ignore ?*Δ *s s1 vl1 a ou s' vl'* (**is** *?match* $\vee$ *?ignore*)
proof *cases*
assume φ: φ *?trn*
with *c* **have** *vl*: *vl* = *f ?trn* # *vl'* **by** (*auto simp*: *consume-def*)
with *fs'* **have** *?ignore* **proof** (*cases f ?trn*)
case (*FRVal u req*)
obtain *p*
where *a*: (*a* = *Cact* (*cFriendReq UID1 p UID2 req*) $\wedge$ *UID1* $\in\in$ *pendingFReqs s' UID2* $\wedge$
UID1 $\notin$ *set* (*pendingFReqs s UID2*) $\wedge$
(*UID2* $\in\in$ *pendingFReqs s' UID1* $\longleftrightarrow$ *UID2* $\in\in$ *pendingFReqs s UID1*)) $\vee$
(*a* = *Cact* (*cFriendReq UID2 p UID1 req*) $\wedge$ *UID2* $\in\in$ *pendingFReqs s' UID1* $\wedge$
UID2 $\notin$ *set* (*pendingFReqs s UID1*) $\wedge$
(*UID1* $\in\in$ *pendingFReqs s' UID2* $\longleftrightarrow$ *UID1* $\in\in$ *pendingFReqs s UID2*))
ou = *outOK* $\neg$*friends12 s* $\neg$*friends12 s' open s'* = *open s*
using φ *step rs FRVal* **by** (*cases rule*: φ*E*) *fastforce+*
then have *fIDs'*: *friendIDs s'* = *friendIDs s* **using** *step* **by** (*auto simp*: *c-defs*)
have *eqButUID s s'* **using** *a step*
by (*auto intro*: *Cact-cFriendReq-step-eqButUID*)
then have Δ*3 s' vl' s1 vl1*
using *ss1 a os BO vVS1 fs-fs1 last-fs fs-fIDs fs fs1 fIDs' vl FRVal*
by (*intro* Δ*3-I*[*of s' s1 vlr vlr1 vl1 fs fs1 vl'*])
(*auto intro*: *eqButUID-trans eqButUID-sym*)
moreover from φ *step rs a* **have** $\neg\gamma$ (*Trans s a ou s'*)
using *UID1-UID2-UIDs* **by** (*cases rule*: φ*E*) *auto*
ultimately show *?ignore* **by** (*intro ignoreI*) *auto*
next
case (*FVal fv''*)
with *vl fs'* **have** *FVal*: *f ?trn* = *FVal fv*
and *vl'*: *filter* (*Not* $\circ$ *isFRVal*) *vl'* = *map FVal* (*fs'* ## *fv'*) @ *OVal True* # *vl''*
by *auto*
from φ *step rs FVal* **have** *ss'*: *eqButUID s s'*
by (*elim* φ*E*) (*auto intro*: *Cact-cFriend-step-eqButUID Dact-dFriend-step-eqButUID*)
then have $\neg$*open s'* **using** *os* **by** (*auto simp*: *eqButUID-open-eq*)
moreover have *eqButUID s' s1* **using** *ss1 ss'* **by** (*auto intro*: *eqButUID-sym eqButUID-trans*)
ultimately have Δ*3 s' vl' s1 vl1* **using** *BO-FVal*(*3*) *vVS1 vl' fs1'*
by (*intro* Δ*3-I*[*of s' s1 vl'' vl1'' vl1 fs'* ## *fv'* [*fv'*] *vl'*]) *auto*
moreover have $\neg\gamma$ *?trn* **using** φ *step rs FVal UID1-UID2-UIDs* **by** (*elim* φ*E*) *auto*

ultimately show *?ignore* **by** (*intro ignoreI*) *auto*
qed *auto*
then show *?thesis* **..**
next
assume $n\varphi$: $\neg\varphi$ *?trn*
then have *os'*: *open s = open s'* **and** *f12s'*: *friends12 s = friends12 s'*
using *step-open-*φ[*OF step*] *step-friends12-*φ[*OF step*] **by** *auto*
have *vl'*: *vl' = vl* **using** $n\varphi$ *c* **by** (*auto simp*: *consume-def*)
show *?thesis* **proof** (*cases* ∀ *req. a* ≠ *Cact* (*cFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Cact* (*cFriend UID2* (*pass s UID2*) *UID1*) ∧
a ≠ *Cact* (*cFriendReq UID2* (*pass s UID2*) *UID1 req*) ∧
a ≠ *Cact* (*cFriendReq UID1* (*pass s UID1*) *UID2 req*) ∧
a ≠ *Dact* (*dFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Dact* (*dFriend UID2* (*pass s UID2*) *UID1*))
case *True*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by** (*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
from *True* $n\varphi$ **have** $n\varphi'$: $\neg\varphi$ *?trn1*
using *eqButUID-step-*φ[*OF ss1 rs rs1 step step1*] **by** *auto*
then have *f12s1'*: *friends12 s1* = *friends12 s1'*
and *pFRs'*: *UID1* ∈∈ *pendingFReqs s1 UID2* ⟷ *UID1* ∈∈ *pendingFReqs s1' UID2*
UID2 ∈∈ *pendingFReqs s1 UID1* ⟷ *UID2* ∈∈ *pendingFReqs s1' UID1*
using *step-friends12-*φ[*OF step1*] *step-pendingFReqs-*φ[*OF step1*]
by *auto*
have *eqButUID s' s1'* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*]
.
thm Δ*3-I*[*of s' s1' vl'' vl1'' vl1 fv # fs' ## fv'* [*fv'*] *vl'*]
then have Δ*3 s' vl' s1' vl1* **using** *os vVS1 fs' fs1' BO-FVal*
unfolding *os' f12s1' pFRs' vl'*
by (*intro* Δ*3-I*[*of s' s1' vl'' vl1'' vl1 fv # fs' ## fv'* [*fv'*] *vl*]) *auto*
then have *?match*
using *step1* $n\varphi'$ *os eqButUID-step-*γ*-out*[*OF ss1 step step1*]
by (*intro matchI*[*of s1 a ou1 s1' vl1 vl1*]) (*auto simp*: *consume-def*)
then show *?match* ∨ *?ignore* **..**
next
case *False*
with $n\varphi$ **have** *ou* ≠ *outOK* **by** *auto*
then have *s' = s* **using** *step False* **by** *auto*
then have *?ignore* **using** *3 False UID1-UID2-UIDs* **unfolding** *vl'* **by** (*intro ignoreI*) *auto*
then show *?match* ∨ *?ignore* **..**

qed
qed
qed
then show *?thesis* **using** *fs′* **by** *auto*
next
case (*BO-FVal1 fv fv′ fs′ fs1′ vl′′ vl1′′*)
then have *fs′*: *filter* (*Not o isFRVal*) *vl* = *map FVal* (*fs′ ## fv′*) @ *OVal True # vl′′*
and *fs1′*: *filter* (*Not o isFRVal*) *vl1* = *map FVal* (*fv # fs1′ ## fv′*) @ *OVal True # vl1′′*
using *fs fs1* **by** *auto*
with *nFRVal1* **obtain** *vl1′*
where *vl1*: *vl1* = *FVal fv # vl1′*
and *vl1′*: *filter* (*Not o isFRVal*) *vl1′* = *map FVal* (*fs1′ ## fv′*) @ *OVal True # vl1′′*
by (*cases vl1*; *cases hd vl1*) *auto*
with *vVS1* **have** *f12*: *friends12 s1* ≠ *fv*
and *vVS1*: *validValSeqFrom* (*FVal fv # vl1′*) *s1* **by** *auto*
then have *?iact* **using** *rs1 IDs1 vl1 ss1 os BO-FVal1*(*3*) *fs′ vl1′*
by (*elim toggle-friends12*[*of s1 fv vl1′*], *blast*, *blast*, *blast*)
(*intro iactionI*[*of s1 - - - vl1 vl1′*],
auto simp: *consume-def*
intro: Δ*3-I*[*of s - vl′′ vl1′′ vl1′ fs′ ## fv′ fs1′ ## fv′ vl*]
eqButUID-trans)
then show *?thesis* **..**
next
case (*FVal-BO fv vl′′ vl1′′*)
then have *fs′*: *filter* (*Not o isFRVal*) *vl* = *FVal fv # OVal True # vl′′*
and *fs1′*: *filter* (*Not o isFRVal*) *vl1* = *FVal fv # OVal True # vl1′′*
using *fs fs1* **by** *auto*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn* = *Trans s a ou s′* **let** *?trn1* = *Trans s1 a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl′*
show *match ?*Δ *s s1 vl1 a ou s′ vl′* ∨ *ignore ?*Δ *s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
proof *cases*
assume *φ*: *φ ?trn*
with *c* **have** *vl*: *vl* = *f ?trn # vl′* **by** (*auto simp*: *consume-def*)
with *fs′* **show** *?thesis* **proof** (*cases f ?trn*)
case (*FRVal u req*)
obtain *p*
where *a*: (*a* = *Cact* (*cFriendReq UID1 p UID2 req*) ∧ *UID1* ∈∈ *pendingFReqs s′ UID2* ∧
UID1 ∉ *set* (*pendingFReqs s UID2*) ∧
(*UID2* ∈∈ *pendingFReqs s′ UID1* ⟷ *UID2* ∈∈ *pendingFReqs s UID1*)) ∨
(*a* = *Cact* (*cFriendReq UID2 p UID1 req*) ∧ *UID2* ∈∈

pendingFReqs s' UID1 ∧
UID2 ∉ *set* (*pendingFReqs s UID1*) ∧
(*UID1* ∈∈ *pendingFReqs s' UID2* ⟷ *UID1* ∈∈ *pendingFReqs s UID2*))
ou = *outOK* ¬*friends12 s* ¬*friends12 s' open s'* = *open s*
using *φ step rs FRVal* **by** (*cases rule*: *φE*) *fastforce+*
then have *fIDs'*: *friendIDs s'* = *friendIDs s* **using** *step* **by** (*auto simp*: *c-defs*)
have *eqButUID s s'* **using** *a step*
by (*auto intro*: *Cact-cFriendReq-step-eqButUID*)
then have Δ*3 s' vl' s1 vl1*
using *ss1 a os BO vVS1 fs-fs1 last-fs fs-fIDs fs fs1 fIDs' vl FRVal*
by (*intro* Δ*3-I*[*of s' s1 vlr vlr1 vl1 fs fs1 vl'*])
(*auto intro*: *eqButUID-trans eqButUID-sym*)
moreover from *φ step rs a* **have** ¬γ (*Trans s a ou s'*)
using *UID1-UID2-UIDs* **by** (*cases rule*: *φE*) *auto*
ultimately have *?ignore* **by** (*intro ignoreI*) *auto*
then show *?thesis* **..**
next
case (*FVal fv''*)
with *vl fs'* **have** *FVal*: *f ?trn* = *FVal fv*
and *vl'*: *filter* (*Not* ∘ *isFRVal*) *vl'* = *OVal True* # *vl''*
by *auto*
from *fs1' nFRVal1* **obtain** *vl1'*
where *vl1*: *vl1* = *FVal fv* # *vl1'*
and *vl1'*: *filter* (*Not* ∘ *isFRVal*) *vl1'* = *OVal True* # *vl1''*
by (*cases vl1*; *cases hd vl1*) *auto*
have *?match* **using** *φ step rs FVal* **proof** (*cases rule*: *φE*)
case (*Friend uid p uid'*)
then have *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] []
and *f12s1*: ¬*friends12 s1*
and *fv*: *fv* = *True*
using *ss1 vVS1 FVal* **unfolding** *eqButUID-def vl1* **by** *auto*
let *?s1'* = *createFriend s1 UID1* (*pass s1 UID1*) *UID2*
have *s'*: *s'* = *createFriend s UID1 p UID2*
using *Friend step* **by** (*auto simp*: *createFriend-sym*)
have *ss'*: *eqButUID s s'* **using** *rs step Friend*
by (*auto intro*: *Cact-cFriend-step-eqButUID*)
moreover then have *os'*: ¬*open s'* **using** *os eqButUID-open-eq* **by** *auto*
moreover obtain *a1 uid1 uid1' p1*
where *step s1 a1* = (*outOK*, *?s1'*) *friends12 ?s1'*
a1 = *Cact* (*cFriend uid1 p1 uid1'*)
uid1 = *UID1* ∧ *uid1'* = *UID2* ∨ *uid1* = *UID2* ∧ *uid1'* = *UID1*
φ (*Trans s1 a1 outOK ?s1'*)
f (*Trans s1 a1 outOK ?s1'*) = *FVal True*
eqButUID s1 ?s1' validValSeqFrom vl1' ?s1'
using *rs1 IDs1 Friend vVS1 f12s1* **unfolding** *vl1 FVal*

by (*elim toggle-friends12-True*; *blast*)
moreover then have *IDsOK ?s1′* [*UID1*, *UID2*] [] [] [] **by** (*auto simp*: *c-defs*)
moreover have *friendIDs s′* = *friendIDs ?s1′*
using *Friend*(*6*) *f12s1* **unfolding** *s′*
by (*intro eqButUID-createFriend12-friendIDs-eq*[*OF ss1 rs rs1*]) *auto*
ultimately show *?match*
using *ss1 FVal-BO Friend UID1-UID2-UIDs vl′ vl1′* **unfolding** *vl1 fv*
by (*intro matchI*[*of s1 a1 outOK ?s1′*])
(*auto simp*: *consume-def intro*: *eqButUID-trans eqButUID-sym*
intro!: Δ*3-I*[*of s′ ?s1′ vl″ vl1″ vl1′* [] [] *vl′*])
next
case (*Unfriend uid p uid′*)
then have *IDs1*: *IDsOK s1* [*UID1*, *UID2*] [] [] []
and *f12s1*: *friends12 s1*
and *fv*: *fv* = *False*
using *ss1 vVS1 FVal* **unfolding** *eqButUID-def vl1* **by** *auto*
let *?s1′* = *deleteFriend s1 UID1* (*pass s1 UID1*) *UID2*
have *s′*: *s′* = *deleteFriend s UID1 p UID2*
using *Unfriend step* **by** (*auto simp*: *deleteFriend-sym*)
have *ss′*: *eqButUID s s′* **using** *rs step Unfriend*
by (*auto intro*: *Dact-dFriend-step-eqButUID*)
moreover then have *os′*: ¬*open s′* **using** *os eqButUID-open-eq* **by** *auto*
moreover obtain *a1 uid1 uid1′ p1*
where *step s1 a1* = (*outOK*, *?s1′*) ¬*friends12 ?s1′*
a1 = *Dact* (*dFriend uid1 p1 uid1′*)
uid1 = *UID1* ∧ *uid1′* = *UID2* ∨ *uid1* = *UID2* ∧ *uid1′* = *UID1*
φ (*Trans s1 a1 outOK ?s1′*)
f (*Trans s1 a1 outOK ?s1′*) = *FVal False*
eqButUID s1 ?s1′ validValSeqFrom vl1′ ?s1′
using *rs1 IDs1 Unfriend vVS1 f12s1* **unfolding** *vl1 FVal*
by (*elim toggle-friends12-False*; *blast*)
moreover then have *IDsOK ?s1′* [*UID1*, *UID2*] [] [] [] **by** (*auto simp*: *d-defs*)
moreover have *friendIDs s′* = *friendIDs ?s1′*
using *Unfriend*(*6*) *f12s1* **unfolding** *s′*
by (*intro eqButUID-deleteFriend12-friendIDs-eq*[*OF ss1 rs rs1*])
ultimately show *?match*
using *ss1 FVal-BO Unfriend UID1-UID2-UIDs vl′ vl1′* **unfolding** *vl1 fv*
by (*intro matchI*[*of s1 a1 outOK ?s1′*])
(*auto simp*: *consume-def intro*: *eqButUID-trans eqButUID-sym*
intro!: Δ*3-I*[*of s′ ?s1′ vl″ vl1″ vl1′* [] [] *vl′*])
qed *auto*
then show *?thesis* **..**

qed *auto*
next
assume *nφ*: ¬φ *?trn*
then have *os'*: *open s* = *open s'* **and** *f12s'*: *friends12 s* = *friends12 s'*
using *step-open-φ*[*OF step*] *step-friends12-φ*[*OF step*] **by** *auto*
have *vl'*: *vl'* = *vl* **using** *nφ c* **by** (*auto simp*: *consume-def*)
show *?thesis* **proof** (*cases* ∀ *req*. *a* ≠ *Cact* (*cFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Cact* (*cFriend UID2* (*pass s UID2*) *UID1*) ∧
a ≠ *Cact* (*cFriendReq UID2* (*pass s UID2*) *UID1 req*) ∧
a ≠ *Cact* (*cFriendReq UID1* (*pass s UID1*) *UID2 req*) ∧
a ≠ *Dact* (*dFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Dact* (*dFriend UID2* (*pass s UID2*) *UID1*))
case *True*
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by** (*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1'*
from *True nφ* **have** *nφ'*: ¬φ *?trn1*
using *eqButUID-step-φ*[*OF ss1 rs rs1 step step1*] **by** *auto*
then have *f12s1'*: *friends12 s1* = *friends12 s1'*
and *pFRs'*: *UID1* ∈∈ *pendingFReqs s1 UID2* ⟷ *UID1* ∈∈ *pendingFReqs s1' UID2*
UID2 ∈∈ *pendingFReqs s1 UID1* ⟷ *UID2* ∈∈ *pendingFReqs s1' UID1*
using *step-friends12-φ*[*OF step1*] *step-pendingFReqs-φ*[*OF step1*]
by *auto*
have *eqButUID s' s1'* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*]
.
thm Δ*3-I*[*of s' s1' vl'' vl1'' vl1* [*fv*] [*fv*] *vl'*]
then have Δ*3 s' vl' s1' vl1* **using** *os vVS1 fs' fs1' FVal-BO*
unfolding *os' f12s1' pFRs' vl'*
by (*intro* Δ*3-I*[*of s' s1' vl'' vl1'' vl1* [*fv*] [*fv*] *vl*]) *auto*
then have *?match*
using *step1 nφ' os eqButUID-step-γ-out*[*OF ss1 step step1*]
by (*intro matchI*[*of s1 a ou1 s1' vl1 vl1*]) (*auto simp*: *consume-def*)
then show *?match* ∨ *?ignore* **..**
next
case *False*
with *nφ* **have** *ou* ≠ *outOK* **by** *auto*
then have *s'* = *s* **using** *step False* **by** *auto*
then have *?ignore* **using** *3 False UID1-UID2-UIDs* **unfolding** *vl'* **by** (*intro ignoreI*) *auto*
then show *?match* ∨ *?ignore* **..**
qed
qed

qed
then show *?thesis* **using** *fs′* **by** *auto*
next
case (*OVal vl″ vl1″*)
then have *fs′*: *filter* (*Not o isFRVal*) *vl* = *OVal True* # *vl″*
and *fs1′*: *filter* (*Not o isFRVal*) *vl1* = *OVal True* # *vl1″*
and *BO″*: *BO vl″ vl1″*
using *fs fs1* **by** *auto*
from *fs fs′* **have** *fs*: *fs* = [] **by** (*cases fs*) *auto*
with *fs-fIDs* **have** *fIDs*: *friendIDs s* = *friendIDs s1* **by** *auto*
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn* = *Trans s a ou s′* **let** *?trn1* = *Trans s1 a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl′*
show *match ?Δ s s1 vl1 a ou s′ vl′* ∨ *ignore ?Δ s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
proof *cases*
assume *φ*: *φ ?trn*
with *c* **have** *vl*: *vl* = *f ?trn* # *vl′* **by** (*auto simp*: *consume-def*)
with *fs′* **show** *?thesis* **proof** (*cases f ?trn*)
case (*FRVal u req*)
obtain *p*
where *a*: (*a* = *Cact* (*cFriendReq UID1 p UID2 req*) ∧ *UID1* ∈∈ *pendingFReqs s′ UID2* ∧
UID1 ∉ *set* (*pendingFReqs s UID2*) ∧
(*UID2* ∈∈ *pendingFReqs s′ UID1* ⟷ *UID2* ∈∈ *pendingFReqs s UID1*)) ∨
(*a* = *Cact* (*cFriendReq UID2 p UID1 req*) ∧ *UID2* ∈∈ *pendingFReqs s′ UID1* ∧
UID2 ∉ *set* (*pendingFReqs s UID1*) ∧
(*UID1* ∈∈ *pendingFReqs s′ UID2* ⟷ *UID1* ∈∈ *pendingFReqs s UID2*))
ou = *outOK* ¬*friends12 s* ¬*friends12 s′ open s′* = *open s*
using *φ step rs FRVal* **by** (*cases rule*: *φE*) *fastforce+*
then have *fIDs′*: *friendIDs s′* = *friendIDs s* **using** *step* **by** (*auto simp*: *c-defs*)
have *eqButUID s s′* **using** *a step*
by (*auto intro*: *Cact-cFriendReq-step-eqButUID*)
then have *Δ3 s′ vl′ s1 vl1*
using *ss1 a os OVal*(*3*) *vVS1 fs′ fs1′ fs fs-fs1 fIDs′ fIDs* **unfolding** *vl FRVal*
by (*intro Δ3-I*[*of s′ s1 vl″ vl1″ vl1 fs fs1 vl′*])
(*auto intro*: *eqButUID-trans eqButUID-sym*)
moreover from *φ step rs a* **have** ¬*γ* (*Trans s a ou s′*)
using *UID1-UID2-UIDs* **by** (*cases rule*: *φE*) *auto*
ultimately have *?ignore* **by** (*intro ignoreI*) *auto*
then show *?thesis* **..**
next

case (*OVal ov'*)
with *vl fs'* **have** *OVal*: *f ?trn = OVal True*
and *vl'*: *filter (Not ∘ isFRVal) vl' = vl''*
by *auto*
from *fs1' nFRVal1* **obtain** *vl1'*
where *vl1*: *vl1 = OVal True # vl1'*
and *vl1'*: *filter (Not ∘ isFRVal) vl1' = vl1''*
by (*cases vl1*; *cases hd vl1*) *auto*
have *?match* **using** *φ step rs OVal* **proof** (*cases rule*: *φE*)
case (*OpenF uid p uid'*)
let *?s1' = createFriend s1 uid p uid'*
have *s'*: *s' = createFriend s uid p uid'*
using *OpenF step* **by** *auto*
from *OpenF(2)* **have** *uids*: *uid ≠ UID1 ∧ uid ≠ UID2 ∧ uid' = UID1 ∨*
uid ≠ UID1 ∧ uid ≠ UID2 ∧ uid' = UID2 ∨
uid' ≠ UID1 ∧ uid' ≠ UID2 ∧ uid = UID1 ∨
uid' ≠ UID1 ∧ uid' ≠ UID2 ∧ uid = UID2
using *UID1-UID2-UIDs* **by** *auto*
have *eqButUIDf* (*pendingFReqs s*) (*pendingFReqs s1*)
using *ss1* **unfolding** *eqButUID-def* **by** *auto*
then have *uid' ∈∈ pendingFReqs s uid ⟷ uid' ∈∈ pendingFReqs s1 uid*
using *OpenF* **by** (*intro eqButUIDf-not-UID'*) *auto*
then have *step1*: *step s1 a = (outOK, ?s1')*
using *OpenF step ss1 fIDs* **unfolding** *eqButUID-def* **by** (*auto simp*: *c-defs*)
have *s's1'*: *eqButUID s' ?s1'* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*] **.**
moreover have *os'*: *open s'* **using** *OpenF* **unfolding** *open-def* **by** *auto*
moreover have *fIDs'*: *friendIDs s' = friendIDs ?s1'*
using *fIDs* **unfolding** *s'* **by** (*auto simp*: *c-defs*)
moreover have *f12s1*: *friends12 s1 = friends12 ?s1'*
UID1 ∈∈ pendingFReqs s1 UID2 ⟷ UID1 ∈∈ pendingFReqs ?s1' UID2
UID2 ∈∈ pendingFReqs s1 UID1 ⟷ UID2 ∈∈ pendingFReqs ?s1' UID1
using *uids* **unfolding** *friends12-def c-defs* **by** *auto*
moreover then have *validValSeqFrom vl1' ?s1'* **using** *vVS1* **unfolding** *vl1* **by** *auto*
ultimately have *Δ1 s' vl' ?s1' vl1'*
using *BO'' IDsOK-mono*[*OF step1 IDs1*] **unfolding** *Δ1-def vl' vl1'* **by** *auto*
moreover have *φ ?trn ⟷ φ* (*Trans s1 a outOK ?s1'*)
using *OpenF(1) uids* **by** (*intro eqButUID-step-φ*[*OF ss1 rs rs1 step step1*]) *auto*
ultimately show *?match* **using** *step1 φ OpenF(1,3,4)* **unfolding** *vl1*

by (*intro matchI*[*of s1 a outOK ?s1′ - vl1′*]) (*auto simp*: *consume-def*)
qed *auto*
then show *?thesis* **..**
qed *auto*
next
assume *nφ*: ¬*φ ?trn*
then have *os′*: *open s* = *open s′* **and** *f12s′*: *friends12 s* = *friends12 s′*
using *step-open-φ*[*OF step*] *step-friends12-φ*[*OF step*] **by** *auto*
have *vl′*: *vl′* = *vl* **using** *nφ c* **by** (*auto simp*: *consume-def*)
show *?thesis* **proof** (*cases* ∀ *req. a* ≠ *Cact* (*cFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Cact* (*cFriend UID2* (*pass s UID2*) *UID1*) ∧
a ≠ *Cact* (*cFriendReq UID2* (*pass s UID2*) *UID1 req*) ∧
a ≠ *Cact* (*cFriendReq UID1* (*pass s UID1*) *UID2 req*) ∧
a ≠ *Dact* (*dFriend UID1* (*pass s UID1*) *UID2*) ∧
a ≠ *Dact* (*dFriend UID2* (*pass s UID2*) *UID1*))
case *True*
obtain *ou1 s1′* **where** *step1*: *step s1 a* = (*ou1*, *s1′*) **by** (*cases step s1 a*) *auto*
let *?trn1* = *Trans s1 a ou1 s1′*
from *True nφ* **have** *nφ′*: ¬*φ ?trn1*
using *eqButUID-step-φ*[*OF ss1 rs rs1 step step1*] **by** *auto*
then have *f12s1′*: *friends12 s1* = *friends12 s1′*
and *pFRs′*: *UID1* ∈∈ *pendingFReqs s1 UID2* ⟷ *UID1* ∈∈ *pendingFReqs s1′ UID2*
UID2 ∈∈ *pendingFReqs s1 UID1* ⟷ *UID2* ∈∈ *pendingFReqs s1′ UID1*
using *step-friends12-φ*[*OF step1*] *step-pendingFReqs-φ*[*OF step1*]
by *auto*
have *eqButUID s′ s1′* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*]
.
moreover have *friendIDs s′* = *friendIDs s1′*
using *eqButUID-step-friendIDs-eq*[*OF ss1 rs rs1 step step1 - fIDs*] *True*
by *auto*
ultimately have Δ*3 s′ vl′ s1′ vl1* **using** *os vVS1 fs′ fs1′ OVal*
unfolding *os′ f12s1′ pFRs′ vl′*
by (*intro* Δ*3-I*[*of s′ s1′ vl′′ vl1′′ vl1* [] [] *vl*]) *auto*
then have *?match*
using *step1 nφ′ os eqButUID-step-γ-out*[*OF ss1 step step1*]
by (*intro matchI*[*of s1 a ou1 s1′ vl1 vl1*]) (*auto simp*: *consume-def*)
then show *?match* ∨ *?ignore* **..**
next
case *False*

```
              with nφ have ou ≠ outOK by auto
              then have s' = s using step False by auto
               then have ?ignore using 3 False UID1-UID2-UIDs unfolding vl'
by (intro ignoreI) auto
              then show ?match ∨ ?ignore ..
            qed
          qed
        qed
        then show ?thesis using fs' by auto
    next
      case (FVal1 fv fs' fs1')
        from this(1) have False proof (induction fs' arbitrary: fs)
          case (Cons fv'' fs'')
            then obtain fs''' where map FVal (fv'' # fs''') @ OVal True # vlr =
map FVal (fv'' # fs'')
              by (cases fs) auto
            with Cons.IH[of fs''] show False by auto
        qed auto
        then show ?thesis ..
    next
      case (FVal) then show ?thesis by (induction fs) auto next
      case (Nil) then show ?thesis by auto
    qed
  qed
qed



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


theorem secure: secure
apply (rule unwind-decomp-secure-graph[of Gr Δ0])
unfolding Gr-def
apply (simp, smt insert-subset order-refl)
using
istate-Δ0 unwind-cont-Δ0 unwind-cont-Δ1 unwind-cont-Δ2 unwind-cont-Δ3
unfolding Gr-def by (auto intro: unwind-cont-mono)

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* **using** *UID1-UID2-UIDs UID1-UID2*
by *unfold-locales*

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*
by (*cases trn*) (*auto elim*: *Issuer*.φ*E*)

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 (*unfold-locales*, *goal-cases*)
  case (*1 aid trn*) **then show** *?case* **by** *auto* **next**
  case (*2 aid trn*) **then show** *?case* **by** *auto* **next**
  case (*3 aid trn*) **then show** *?case* **by** (*cases trn*) *auto* **next**
  case (*4 aid trn*) **then show** *?case* **by** (*cases* (*aid*,*trn*) *rule*: *tgtNodeOf.cases*) *auto* **next**
  case (*5 aid1 trn1 aid2 trn2*) **then show** *?case* **by** *auto* **next**
  case (*6 aid1 trn1 aid2 trn2*) **then show** *?case* **by** (*cases trn1*; *cases trn2*; *auto*) **next**
  case (*7 aid1 trn1 aid2 trn2*) **then show** *?case* **by** *auto* **next**
  case (*8 aid1 trn1 aid2 trn2*) **then show** *?case* **by** (*cases trn1*; *cases trn2*; *auto*) **next**
  case (*9 aid trn*) **then show** *?case* **by** (*cases* (*aid*,*trn*) *rule*: *tgtNodeOf.cases*) (*auto simp*: *FriendObservationSetup.γ.simps*) **next**
  case (*10 aid trn*) **then show** *?case* **by** *auto*
qed *auto*

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*
using *AID-AIDs Issuer.secure*
by *unfold-locales auto*

theorem *secure*: *secure*
proof (*intro preserve-source-secure ballI*)
  fix *aid*
  assume *aid* ∈ *AIDs* − {*AID*}
  then show *Net.lsecure aid* **by** (*intro Abstract-BD-Security.B-id-secure*) (*auto simp*: *B-id-def*)
qed

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 symmetric. 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'*
by (*cases ca*; *auto*)+

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*))
by (*cases ou*; *cases uID* = *UID*; *auto*)+

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*)
⟷ *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 osn*)
⟷ (∃ *p′*. *a* = *COMact* (*comSendPost uID p′ aID nID*) ∧ *p* = *emptyPass* ∧ *ou* = *O-sendPost osn*)
g (*Trans s a ou s′*) = (*COMact* (*comSendCreateOFriend uID p aID uID′*), *O-sendCreateOFriend* (*aid*, *sp*, *uid*, *uid′*))
⟷ ((∃ *p′ uid″*. (*a* = *COMact* (*comSendCreateOFriend uID p′ aID uid″*) ∨ *a* = *COMact* (*comSendDeleteOFriend uID p′ aID uid″*)) ∧ *uID* = *UID* ∧ *uid″* ∉ *UIDs aID* ∧ *uID′* = *emptyUserID* ∧ *p* = *emptyPass*)
∨ (∃ *p′*. *a* = *COMact* (*comSendCreateOFriend uID p′ aID uID′*) ∧ ¬(*uID* = *UID* ∧ *uID′* ∉ *UIDs aID*) ∧ *p* = *emptyPass*))
∧ ((∃ *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*)))

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'*))
⟷ (∃ *p'*. *a* = *COMact* (*comSendDeleteOFriend uID p' aID uID'*) ∧ ¬(*uID* = *UID* ∧ *uID'* ∉ *UIDs aID*) ∧ *p* = *emptyPass*)
∧ (*ou* = *O-sendDeleteOFriend* (*aid*, *sp*, *uid*, *uid'*) ∧ ¬(*uid* = *UID* ∧ *uid'* ∉ *UIDs aid*))
g (*Trans s a ou s'*) = (*COMact* (*comReceiveDeleteOFriend aID cp uID uID'*), *outOK*)
⟷ *a* = *COMact* (*comReceiveDeleteOFriend aID cp uID uID'*) ∧ *ou* = *outOK*
by (*cases a*; *auto simp*: *comPurge-simps*)+

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* × *userID*) *list* ⇒ (*apiID* × *userID*) *list* **where**
filterUIDs auidl = *filter* (λ*auid*. (*snd auid*) ∈ *UIDs* (*fst auid*)) *auidl*

fun *removeUIDs* :: (*apiID* × *userID*) *list* ⇒ (*apiID* × *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*
by *auto*

lemma *eqButUIDs-sym*:
assumes *eqButUIDs uidl uidl1*
shows *eqButUIDs uidl1 uidl*
using *assms* **by** *auto*

lemma *eqButUIDs-trans*:
assumes *eqButUIDs uidl uidl1* **and** *eqButUIDs uidl1 uidl2*
shows *eqButUIDs uidl uidl2*

using *assms* **by** *auto*

lemma *eqButUIDs-remove1-cong*:
assumes *eqButUIDs uidl uidl1*
shows *eqButUIDs* (*remove1 auid uidl*) (*remove1 auid uidl1*)
using *assms* **by** (*auto simp*: *filter-remove1*)

lemma *eqButUIDs-snoc-cong*:
assumes *eqButUIDs uidl uidl1*

shows *eqButUIDs* (*uidl ## auid'*) (*uidl1 ## auid'*)
using *assms* **by** *auto*

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*
unfolding *eqButUIDf-def* **by** *auto*

lemma *eqButUIDf-sym*:
assumes *eqButUIDf frds frds1* **shows** *eqButUIDf frds1 frds*
using *assms* **unfolding** *eqButUIDf-def*
by *auto*

lemma *eqButUIDf-trans*:
assumes *eqButUIDf frds frds1* **and** *eqButUIDf frds1 frds2*
shows *eqButUIDf frds frds2*
using *assms* **unfolding** *eqButUIDf-def* **by** *auto*

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*))
using *assms* **unfolding** *eqButUIDf-def* **by** *auto*

lemma *eqButUIDf-not-UID*:
⟦*eqButUIDf frds frds1*; *uid* ≠ *UID*⟧ ⟹ *frds uid* = *frds1 uid*
unfolding *eqButUIDf-def* **by** (*auto split*: *if-splits*)

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*
unfolding *eqButUID-def* **by** *auto*

lemma *eqButUID-sym*[*sym*]:
assumes *eqButUID s s1* **shows** *eqButUID s1 s*
using *assms eqButUIDf-sym* **unfolding** *eqButUID-def* **by** *auto*

lemma *eqButUID-trans*[*trans*]:
assumes *eqButUID s s1* **and** *eqButUID s1 s2* **shows** *eqButUID s s2*
using *assms eqButUIDf-trans* **unfolding** *eqButUID-def* **by** *metis*

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
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
using *assms* **unfolding** *eqButUID-def IDsOK-def[abs-def]* **by** *auto*

lemmas *eqButUID-eqButUIDf = eqButUID-stateSelectors(22)*

lemma *eqButUID-eqButUIDs*:
eqButUID s s1 ⟹ *eqButUIDs (sentOuterFriendIDs s UID) (sentOuterFriendIDs s1 UID)*
unfolding *eqButUID-def eqButUIDf-def* **by** *auto*

lemma *eqButUID-not-UID*:
eqButUID s s1 ⟹ *uid* ≠ *UID* ⟹ *sentOuterFriendIDs s uid = sentOuterFriendIDs s1 uid*
unfolding *eqButUID-def eqButUIDf-def* **by** *auto*

lemma *eqButUID-sentOuterFriends-UIDs*:
assumes *eqButUID s s1*
and *uid′* ∈ *UIDs aid*
shows *(aid, uid′)* ∈∈ *sentOuterFriendIDs s UID* ⟷ *(aid, uid′)* ∈∈ *sentOuterFriendIDs s1 UID*
proof –
  have *(aid, uid′)* ∈∈ *filterUIDs (sentOuterFriendIDs s UID)*
    ⟷ *(aid, uid′)* ∈∈ *filterUIDs (sentOuterFriendIDs s1 UID)*
    using *assms* **unfolding** *eqButUID-def eqButUIDf-def* **by** *auto*
  then show *?thesis* **using** *assms* **by** *auto*
qed

lemma *eqButUID-sentOuterFriendIDs-cong*:
assumes *eqButUID s s1*
and *uid′* ∉ *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*
using *assms* **unfolding** *eqButUID-def eqButUIDf-def* **by** (*auto simp*: *filter-remove1*)

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*⦈)
⋀ *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* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* ⦇*pendingFReqs* := *uu1*⦈) (*s1* ⦇*pendingFReqs* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *uu1* = *uu2* ⟹ *eqButUID* (*s* ⦇*friendReq* := *uu1*⦈) (*s1* ⦇*friendReq* := *uu2*⦈)
⋀ *uu1 uu2*. *eqButUID s s1* ⟹ *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* ⟹ *eqButUIDf 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*⦈)
unfolding *eqButUID-def* **by** *auto*

lemma *distinct-remove1-idem*: *distinct xs* ⟹ *remove1 y* (*remove1 y xs*) = *remove1 y xs*
by (*induction xs*) (*auto simp add*: *remove1-idem*)

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 −
note *simps* = *eqButUID-stateSelectors s-defs c-defs d-defs u-defs r-defs l-defs com-defs*
eqButUID-sentOuterFriends-UIDs eqButUID-not-UID
from *assms* **show** *?thesis* **proof** (*cases a*)
case (*Sact sa*) **with** *assms* **show** *?thesis* **by** (*cases sa*) (*auto simp add*: *simps intro*!: *eqButUID-cong*)
next
case (*Cact ca*) **with** *assms* **show** *?thesis* **by** (*cases ca*) (*auto simp add*: *simps intro*!: *eqButUID-cong*)
next
case (*Uact ua*) **with** *assms* **show** *?thesis* **by** (*cases ua*) (*auto simp add*: *simps intro*!: *eqButUID-cong*)
next
case (*Ract ra*) **with** *assms* **show** *?thesis* **by** (*cases ra*) (*auto simp add*: *simps intro*!: *eqButUID-cong*)

next
  case (*Lact la*) **with** *assms* **show** *?thesis* **by** (*cases la*) (*auto simp add*: *simps intro*!: *eqButUID-cong*)
next
  case (*COMact ca*)
    with *assms* **show** *?thesis* **proof** (*cases ca*)
      case (*comSendCreateOFriend uid p aid uid'*)
        then show *?thesis*
       using *COMact assms eqButUID-eqButUIDf*[*OF ss1*] *eqButUID-eqButUIDs*[*OF ss1*]
          by (*cases uid* = *UID*; *cases uid'* ∈ *UIDs aid*)
                (*auto simp*: *simps intro*!: *eqButUID-cong eqButUIDf-cong intro*: *eqButUID-sentOuterFriendIDs-cong*)
    next
      case (*comSendDeleteOFriend uid p aid uid'*)
        then show *?thesis*
       using *COMact assms eqButUID-eqButUIDf*[*OF ss1*] *eqButUID-eqButUIDs*[*OF ss1*]
          by (*cases uid* = *UID*; *cases uid'* ∈ *UIDs aid*)
           (*auto simp*: *simps filter-remove1 intro*!: *eqButUID-cong eqButUIDf-cong intro*: *eqButUID-sentOuterFriendIDs-cong*)
    qed (*auto simp*: *simps intro*!: *eqButUID-cong*)
next
  case (*Dact da*) **with** *assms* **show** *?thesis* **by** (*cases da*) (*auto simp add*: *simps intro*!: *eqButUID-cong*)
qed
qed

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** $a = Cact\ (cFriend\ uid\ p\ uid')\ ou = outOK\ p = pass\ s\ uid$
$uid \in UIDs\ AID \wedge uid' = UID \vee uid = UID \wedge uid' \in UIDs\ AID$
$open\ s'\ \neg open\ s$
| ($CloseF$) $uid\ p\ uid'$ **where** $a = Dact\ (dFriend\ uid\ p\ uid')\ ou = outOK\ p = pass\ s\ uid$
$uid \in UIDs\ AID \wedge uid' = UID \vee uid = UID \wedge uid' \in UIDs\ AID$
$open\ s\ \neg open\ s'$
using *assms* **proof** (*cases a*)
case (*Uact ua*) **then show** *?thesis* **using** *assms* **by** (*cases ua*) (*auto simp*: *u-defs open-def*) **next**
case (*COMact ca*) **then show** *?thesis* **using** *assms* **by** (*cases ca*) (*auto simp*: *com-defs open-def*) **next**
case (*Sact sa*)
then show *?thesis* **using** *assms* **by** (*cases sa*) (*auto simp*: *s-defs open-def*)
next
case (*Cact ca*)
then show *?thesis* **using** *assms* **proof** (*cases ca*)
case (*cFriend uid p uid′*)
then show *?thesis* **using** *Cact assms* **by** (*intro OpenF*) (*auto simp*: *c-defs open-def*)
qed (*auto simp*: *c-defs open-def*)
next
case (*Dact da*)
then show *?thesis* **using** *assms* **proof** (*cases da*)
case (*dFriend uid p uid′*)
then show *?thesis* **using** *Dact assms* **by** (*intro CloseF*) (*auto simp*: *d-defs open-def*)
qed
qed *auto*

lemma *COMact-open*:
assumes $step\ s\ a = (ou, s')$
and $a = COMact\ ca$
shows $open\ s = open\ s'$
by (*rule ccontr*, *insert assms*, *elim open-step-cases*, *auto*)

lemma *eqButUID-open-eq*: $eqButUID\ s\ s1 \Longrightarrow open\ s = open\ s1$
using *open-def eqButUID-def* **by** *auto*

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*

proof *cases*
assume *open s* = *open s'*
with φ **show** *thesis* **by** (*elim* φ.*elims*) (*auto intro*: *Friend Unfriend*)
next
assume *open s* ≠ *open s'*
then show *thesis* **proof** (*elim open-step-cases*[*OF* - *step*], *goal-cases*)
case *1* **then show** *?case* **by** (*intro OpenF*) *auto* **next**
case *2* **then show** *?case* **by** (*intro CloseF*) *auto*
qed
qed

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 −
obtain *uid sa com-act* **where** *uid-a*: (*userOfA a* = *Some uid* ∧ *uid* ∈ *UIDs AID* ∧ *uid* ≠ *UID*)
∨ *a* = *COMact com-act* ∨ *a* = *Sact sa*
using γ *UID-UIDs* **by** *fastforce*
note *simps* = *eqButUID-not-UID eqButUID-stateSelectors r-defs s-defs c-defs com-defs l-defs u-defs d-defs*
note *facts* = *ss1 step step1 uid-a*
show *?thesis*
proof (*cases a*)
case (*Ract ra*) **then show** *?thesis* **using** *facts* **by** (*cases ra*) (*auto simp add*: *simps*)
next
case (*Sact sa*) **then show** *?thesis* **using** *facts* **by** (*cases sa*) (*auto simp add*: *simps*)
next
case (*Cact ca*) **then show** *?thesis* **using** *facts* **by** (*cases ca*) (*auto simp add*: *simps*)
next
case (*COMact ca*)
then show *?thesis* **using** *facts* **proof** (*cases ca*)
case (*comSendCreateOFriend uID p aID uID'*)
with *facts* φ **show** *?thesis* **using** *COMact eqButUID-sentOuterFriends-UIDs*[*OF ss1*]
by (*cases uID* = *UID*) (*auto simp*: *simps*)
next
case (*comSendDeleteOFriend uID p aID uID'*)
with *facts* φ **show** *?thesis* **using** *COMact eqButUID-sentOuterFriends-UIDs*[*OF ss1*]

        by (*cases uID* = *UID*) (*auto simp*: *simps*)
      qed (*auto simp*: *simps*)
  next
    case (*Lact la*)
      then show *?thesis* **using** *facts* **proof** (*cases la*)
        case (*lSentOuterFriends uID p uID'*)
          with *Lact facts os1* **show** *?thesis* **by** (*cases uID'* = *UID*) (*auto simp*: *simps open-def*)
      next
        case (*lInnerPosts uid p*)
          then have *o*: $\bigwedge$*nid. owner s nid* = *owner s1 nid*
              and *n*: $\bigwedge$*nid. post s nid* = *post s1 nid*
              and *nids*: *postIDs s* = *postIDs s1*
              and *vis*: *vis s* = *vis s1*
              and *fu*: $\bigwedge$*uid'. friendIDs s uid'* = *friendIDs s1 uid'*
              and *e*: *e-listInnerPosts s uid p* $\longleftrightarrow$ *e-listInnerPosts s1 uid p*
            using *ss1* **unfolding** *eqButUID-def l-defs* **by** *auto*
          have *listInnerPosts s uid p* = *listInnerPosts s1 uid p*
            unfolding *listInnerPosts-def o n nids vis fu* **..**
          with *e* **show** *?thesis* **using** *Lact lInnerPosts step step1* **by** *auto*
      qed (*auto simp add*: *simps*)
  next
    case (*Uact ua*) **then show** *?thesis* **using** *facts* **by** (*cases ua*) (*auto simp add*: *simps*)
  next
    case (*Dact da*) **then show** *?thesis* **using** *facts* **by** (*cases da*) (*auto simp add*: *simps*)
  qed
qed

lemma *step-open-φ*:
assumes *step s a* = (*ou*, *s'*)
and *open s* ≠ *open s'*
shows *φ* (*Trans s a ou s'*)
using *assms* **by** (*elim open-step-cases*) (*auto simp*: *open-def*)

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'*
using *assms* **proof** *cases*
  assume *φ* (*Trans s a ou s'*)
  then show *eqButUID s s'* **using** *assms* **proof** (*cases rule*: *φE*)
    case (*Friend p aid uid'*)

then show *?thesis*
using *assms eqButUID-sentOuterFriendIDs-cong*[*of s s*]
by (*auto split*: *prod.splits simp*: *com-defs*)
next
case (*Unfriend p aid uid′*)
then show *?thesis*
using *assms eqButUID-sentOuterFriendIDs-cong*[*of s s*]
by (*auto split*: *prod.splits simp*: *com-defs*)
qed *auto*
qed (*auto split*: *prod.splits*)

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 −
have *eqButUID s′ s1′* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*] **.**
then have *open s* = *open s1* **and** *open s′* = *open s1′*
using *ss1* **by** (*auto simp*: *eqButUID-open-eq*)
with *φ step step1* **show** *φ* (*Trans s1 a ou1 s1′*)
using *rs ss1 a* **by** (*elim φE*) (*auto simp*: *com-defs*)
qed

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
assume *φ* (*Trans s a ou s′*)
with *assms* **show** *φ* (*Trans s1 a ou1 s1′*) **by** (*rule eqButUID-step-φ-imp*)
next
assume *φ* (*Trans s1 a ou1 s1′*)
moreover have *eqButUID s1 s* **using** *ss1* **by** (*rule eqButUID-sym*)
moreover have ∀ *aID uID′*. *uID′* ∉ *UIDs aID* ⟶
a ≠ *COMact* (*comSendCreateOFriend UID* (*pass s1 UID*) *aID uID′*)
∧
a ≠ *COMact* (*comSendDeleteOFriend UID* (*pass s1 UID*) *aID uID′*)
using *a ss1* **by** (*auto simp*: *eqButUID-stateSelectors*)

ultimately show φ (*Trans s a ou s'*) **using** *rs rs1 step step1*
by (*intro eqButUID-step-*φ*-imp*[*of s1 s*])
qed

lemma *eqButUID-step-*γ:
assumes *ss1*: *eqButUID s s1*
and *rs*: *reach s* **and** *rs1*: *reach s1*
and *step*: *step s a* = (*ou*,*s'*) **and** *step1*: *step s1 a* = (*ou1*,*s1'*)
and *a*: $\forall$ *aID uID'*. *uID'* $\notin$ *UIDs aID* $\longrightarrow$
a $\neq$ *COMact* (*comSendCreateOFriend UID* (*pass s UID*) *aID uID'*)
$\wedge$
a $\neq$ *COMact* (*comSendDeleteOFriend UID* (*pass s UID*) *aID uID'*)
shows γ (*Trans s a ou s'*) = γ (*Trans s1 a ou1 s1'*)
proof −
{ **fix** *ca*
assume *a*: *a* = *COMact ca*
then have *ou* = *ou1* **using** *assms* **proof** (*cases ca*)
case (*comSendCreateOFriend uid p aid uid'*)
with *assms a* **show** *?thesis*
by (*cases uid* = *UID*; *cases uid'* $\in$ *UIDs aid*)
(*auto simp*: *com-defs eqButUID-def eqButUID-sentOuterFriends-UIDs eqButUID-not-UID*)
next
case (*comSendDeleteOFriend uid p aid uid'*)
with *assms a* **show** *?thesis*
by (*cases uid* = *UID*; *cases uid'* $\in$ *UIDs aid*)
(*auto simp*: *com-defs eqButUID-def eqButUID-sentOuterFriends-UIDs eqButUID-not-UID*)
qed (*auto simp*: *com-defs eqButUID-def*)
}
with *assms* **show** *?thesis* **by** *auto*
qed

end

end
theory *Outer-Friend-Issuer*
imports
Outer-Friend-Issuer-Value-Setup
Bounded-Deducibility-Security.Compositional-Reasoning
begin

9.1.5 Declassification bound

context *OuterFriendIssuer*
begin

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

where $T\ trn = False$

For each user *uid* at a node *aid*, the remote friendship updates with the fixed user *UID* at node *AID* form an alternating sequence of friending and unfriending.

Note that actions involving remote users who are observers do not produce secret values; instead, those actions are observable, and the property we verify does not protect their confidentiality.

fun *validValSeq* :: *value list* ⇒ (*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* = []
unfolding *B-def BO-def* **by** (*auto elim*: *BC.cases*)

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** $B = B$
done

9.1.6 Unwinding proof

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

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

definition $\Delta 2$ :: *state* ⇒ *value list* ⇒ *state* ⇒ *value list* ⇒ *bool* **where**
$\Delta 2$ *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*
by (*induction vl arbitrary*: *auidl*) (*auto elim*: *validValSeq.elims*)

lemma *filter-removeAll*: *filter P* (*removeAll x xs*) = *removeAll x* (*filter P xs*)
unfolding *removeAll-filter-not-eq* **by** (*auto intro*: *filter-cong*)

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 *cases*
  assume *φ ?trn*
  moreover then obtain *v* **where** *vl* = *v* # *vl′* **using** *c* **by** (*cases vl*, *auto simp*:
consume-def)
  moreover have *distinct* (*sentOuterFriendIDs s UID*) **using** *rs* **by** (*intro reach-distinct-friends-reqs*)
  ultimately show *?thesis* **using** *assms*
    by (*elim φE*)
      (*auto simp*: *com-defs c-defs d-defs consume-def distinct-remove1-removeAll*
filter-removeAll)
next
  assume *nφ*: ¬*φ ?trn*
  then have *vl′*: *vl′* = *vl* **using** *c* **by** (*auto simp*: *consume-def*)
  then show *?thesis* **using** *vVS step* **proof** (*cases a*)

case (*Sact sa*) **then show** *?thesis* **using** *assms vl′* **by** (*cases sa*) (*auto simp*: *s-defs*) **next**
case (*Cact ca*) **then show** *?thesis* **using** *assms vl′* **by** (*cases ca*) (*auto simp*: *c-defs*) **next**
case (*Dact da*) **then show** *?thesis* **using** *assms vl′* **by** (*cases da*) (*auto simp*: *d-defs*) **next**
case (*Uact ua*) **then show** *?thesis* **using** *assms vl′* **by** (*cases ua*) (*auto simp*: *u-defs*) **next**
case (*COMact ca*) **then show** *?thesis* **using** *assms vl′ nφ* **by** (*cases ca*) (*auto simp*: *com-defs filter-remove1*)
qed *auto*
qed

lemma *istate-Δ0*:
assumes *B*: *B vl vl1*
shows *Δ0 istate vl istate vl1*
using *assms* **unfolding** *Δ0-def*
by *auto*

lemma *unwind-cont-Δ0*: *unwind-cont Δ0* {*Δ1*,*Δ2*}
proof(*rule*, *simp*)
let *?Δ* = *λs vl s1 vl1*. *Δ1 s vl s1 vl1* ∨
Δ2 s vl s1 vl1
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *Δ0*: *Δ0 s vl s1 vl1*
then have *rs*: *reach s* **and** *s*: *s* = *istate* **and** *s1*: *s1* = *istate* **and** *B*: *B vl vl1*
using *reachNT-reach* **unfolding** *Δ0-def* **by** *auto*
show *iaction ?Δ s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?Δ s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn* = *Trans s a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl′*
show *match ?Δ s s1 vl1 a ou s′ vl′* ∨ *ignore ?Δ s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
proof (*intro disjI1*)
obtain *uid p* **where** *a*: *a* = *Sact* (*sSys uid p*) ∨ *s′* = *s*
using *step* **unfolding** *s* **by** (*elim istate-sSys*) *auto*
have ¬*open s′* **using** *step a s* **by** (*auto simp*: *istate-def s-defs open-def*)
moreover then have ¬*φ ?trn* **using** *step rs a* **by** (*auto elim*!: *φE simp*: *s istate-def com-defs*)
moreover have *sentOuterFriendIDs s′ UID* = *sentOuterFriendIDs s UID*
using *s a step* **by** (*auto simp*: *s-defs*)
ultimately show *?match* **using** *s s1 step B c* **unfolding** *Δ1-def Δ2-def B-def*
by (*intro matchI*[*of s1 a ou s′ vl1 vl1*]) (*auto simp*: *consume-def*)
qed
qed

with *B-Nil-Nil*[*OF B*] **show** *?thesis* **by** *auto*
qed
qed

lemma *unwind-cont-Δ1*: *unwind-cont* Δ*1* {Δ*1*,Δ*2*}
proof(*rule*, *simp*)
let *?*Δ = λ*s vl s1 vl1*. Δ*1 s vl s1 vl1* ∨
Δ*2 s vl s1 vl1*
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** Δ*0*: Δ*1 s vl s1 vl1*
then have *rs*: *reach s* **and** *s*: *s1* = *s* **and** *BO*: *BO vl vl1*
and *vVS1*: *validValSeqFrom vl1 s1*
using *reachNT-reach* **unfolding** Δ*1-def* **by** *auto*
show *iaction ?*Δ *s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction ?*Δ *s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s'* :: *state* **and** *vl'*
let *?trn* = *Trans s a ou s'*
assume *step*: *step s a* = (*ou*, *s'*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl'*
show *match ?*Δ *s s1 vl1 a ou s' vl'* ∨ *ignore ?*Δ *s s1 vl1 a ou s' vl'* (**is** *?match* ∨ *?ignore*)
proof *cases*
assume φ: φ *?trn*
consider (*Eq*) *vl1* = *vl*
| (*BC*) *vl0 vl'' vl1''* **where** *vl* = *vl0* @ *OVal False* # *vl''*
and *vl1* = *vl0* @ *OVal False* # *vl1''*
and *BC vl'' vl1''*
using *BO*
by (*auto simp*: *BO-def*)
then have *?match*
proof *cases*
case *Eq*
then show *?thesis*
using *step s c vVS1 step-validValSeqFrom*[*OF step rs c*]
by (*intro matchI*[*of s1 a ou s' vl1 vl'*]) (*auto simp*: Δ*1-def BO-def*)
next
case *BC*
show *?match* **proof** (*cases vl0*)
case *Nil*
then have *consume ?trn vl1 vl1''* **and** *vl'* = *vl''* **and** *f*: *f ?trn* = *OVal False*
using φ *c BC* **by** (*auto simp*: *consume-def*)
moreover then have *validValSeqFrom vl1'' s'*
using *s rs vVS1* **by** (*intro step-validValSeqFrom*[*OF step*]) *auto*
moreover have ¬*open s'* **using** φ *step rs f* **by** (*auto elim*: φ*E*)
ultimately show *?thesis*
using *step s BC* **by** (*intro matchI*[*of s1 a ou s' vl1 vl1''*]) (*auto simp*:

Δ*2-def*)
next
case (*Cons v vl0′*)
then have *consume ?trn vl1* (*vl0′* @ *OVal False* # *vl1″*) and *vl′* = *vl0′* @ *OVal False* # *vl″*
using φ *c BC* by (*auto simp*: *consume-def*)
moreover then have *validValSeqFrom* (*vl0′* @ *OVal False* # *vl1″*) *s′*
using *s rs vVS1* by (*intro step-validValSeqFrom*[*OF step*]) *auto*
ultimately show *?thesis*
using *step s BC*
by (*intro matchI*[*of s1 a ou s′ vl1* (*vl0′* @ *OVal False* # *vl1″*)]) (*auto simp*: Δ*1-def BO-def*)
qed
qed
then show *?match* ∨ *?ignore* ..
next
assume *n*φ: ¬φ *?trn*
then have *consume ?trn vl1 vl1* and *vl′* = *vl* using *c* by (*auto simp*: *consume-def*)
moreover then have *validValSeqFrom vl1 s′*
using *s rs vVS1* by (*intro step-validValSeqFrom*[*OF step*]) *auto*
ultimately have *?match*
using *step s BO* by (*intro matchI*[*of s1 a ou s′ vl1 vl1*]) (*auto simp*: Δ*1-def*)
then show *?match* ∨ *?ignore* ..
qed
qed
with *BO* show *?thesis* by (*auto simp*: *BO-def*)
qed
qed

lemma *unwind-cont-*Δ*2*: *unwind-cont* Δ*2* {Δ*2*}
proof(*rule*, *simp*)
fix *s s1* :: *state* **and** *vl vl1* :: *value list*
assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** Δ*2*: Δ*2 s vl s1 vl1*
then have *rs*: *reach s* **and** *ss1*: *eqButUID s s1* **and** *BC*: *BC vl vl1*
and *os*: ¬*open s1* **and** *vVS1*: *validValSeqFrom vl1 s1*
using *reachNT-reach* **unfolding** Δ*2-def* **by** *auto*
show *iaction* Δ*2 s vl s1 vl1* ∨
((*vl* = [] ⟶ *vl1* = []) ∧ *reaction* Δ*2 s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
proof−
have *?react* **proof**
fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
let *?trn* = *Trans s a ou s′*
assume *step*: *step s a* = (*ou*, *s′*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl′*
show *match* Δ*2 s s1 vl1 a ou s′ vl′* ∨ *ignore* Δ*2 s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
proof *cases*
assume φ: φ *?trn*

with *BC c* **have** *?match* **proof** (*cases rule*: *BC.cases*)
case (*BC-FrVal vl'' vl1'' uid' aid uid st st'*)
then show *?thesis* **proof** (*cases st'*)
case *True*
let *?a1* = *COMact* (*comSendCreateOFriend UID* (*pass s1 UID*) *aid uid'*)
let *?ou1* = *O-sendCreateOFriend* (*aid*, *clientPass s aid*, *UID*, *uid'*)
let *?s1'* = *snd* (*sendCreateOFriend s1 UID* (*pass s1 UID*) *aid uid'*)
let *?trn1* = *Trans s1 ?a1 ?ou1 ?s1'*
have *c1*: *consume ?trn1 vl1 vl1''* **and** *vl'* = *vl''* **and** *f ?trn* = *FrVal aid uid st*
using φ *c BC-FrVal True* **by** (*auto simp*: *consume-def*)
moreover then have *a*: (*a* = *COMact* (*comSendCreateOFriend UID* (*pass s UID*) *aid uid*)
∧ *ou* = *O-sendCreateOFriend* (*aid*, *clientPass s aid*, *UID*, *uid*))
∨ (*a* = *COMact* (*comSendDeleteOFriend UID* (*pass s UID*) *aid uid*)
∧ *ou* = *O-sendDeleteOFriend* (*aid*, *clientPass s aid*, *UID*, *uid*))
and *IDs*: *IDsOK s* [*UID*] [] [] [*aid*]
and *uid*: *uid* ∉ *UIDs aid*
using φ *step rs* **by** (*auto elim*!: φ*E split*: *prod.splits simp*: *com-defs*)
moreover have *step1*: *step s1 ?a1* = (*?ou1*, *?s1'*)
using *IDs vVS1 BC-FrVal True ss1* **by** (*auto simp*: *com-defs eqButUID-def*)
moreover then have *validValSeqFrom vl1'' ?s1'*
using *vVS1 rs1 c1* **by** (*intro step-validValSeqFrom*[*OF step1*]) *auto*
moreover have ¬*open ?s1'* **using** *os* **by** (*auto simp*: *open-def com-defs*)
moreover have *eqButUID s' ?s1'*
using *ss1 step a uid BC-FrVal*(*4*) *eqButUID-eqButUIDf*[*OF ss1*] *eqButUID-eqButUIDs*[*OF ss1*]
by (*auto split*: *prod.splits simp*: *com-defs filter-remove1 intro*!: *eqButUID-cong eqButUIDf-cong*)
moreover have γ *?trn* = γ *?trn1* **and** *g ?trn* = *g ?trn1*
using *BC-FrVal a uid* **by** (*auto simp*: *com-defs*)
ultimately show *?match*
using *BC-FrVal* **by** (*intro matchI*[*of s1 ?a1 ?ou1 ?s1' vl1 vl1''*]) (*auto simp*: Δ*2-def*)
next
case *False*
let *?a1* = *COMact* (*comSendDeleteOFriend UID* (*pass s1 UID*) *aid uid'*)
let *?ou1* = *O-sendDeleteOFriend* (*aid*, *clientPass s aid*, *UID*, *uid'*)
let *?s1'* = *snd* (*sendDeleteOFriend s1 UID* (*pass s1 UID*) *aid uid'*)
let *?trn1* = *Trans s1 ?a1 ?ou1 ?s1'*
have *c1*: *consume ?trn1 vl1 vl1''* **and** *vl'* = *vl''* **and** *f ?trn* = *FrVal aid uid st*

using φ *c BC-FrVal False* **by** (*auto simp*: *consume-def*)
moreover then have *a*: (*a* = *COMact* (*comSendCreateOFriend UID* (*pass s UID*) *aid uid*)
∧ *ou* = *O-sendCreateOFriend* (*aid*, *clientPass s aid*, *UID*, *uid*))
∨ (*a* = *COMact* (*comSendDeleteOFriend UID* (*pass s UID*) *aid uid*)
∧ *ou* = *O-sendDeleteOFriend* (*aid*, *clientPass s aid*, *UID*, *uid*))
and *IDs*: *IDsOK s* [*UID*] [] [] [*aid*]
and *uid*: *uid* ∉ *UIDs aid*
using φ *step rs* **by** (*auto elim*!: φ*E split*: *prod.splits simp*: *com-defs*)
moreover have *step1*: *step s1 ?a1* = (*?ou1*, *?s1'*)
using *IDs vVS1 BC-FrVal False ss1* **by** (*auto simp*: *com-defs eqButUID-def*)
moreover then have *validValSeqFrom vl1'' ?s1'*
using *vVS1 rs1 c1* **by** (*intro step-validValSeqFrom*[*OF step1*]) *auto*
moreover have ¬*open ?s1'* **using** *os* **by** (*auto simp*: *open-def com-defs*)
moreover have *eqButUID s' ?s1'*
using *ss1 step a uid BC-FrVal*(*4*) *eqButUID-eqButUIDf*[*OF ss1*] *eqButUID-eqButUIDs*[*OF ss1*]
by (*auto split*: *prod.splits simp*: *com-defs filter-remove1 intro*!: *eqButUID-cong eqButUIDf-cong*)
moreover have γ *?trn* = γ *?trn1* **and** *g ?trn* = *g ?trn1*
using *BC-FrVal a uid* **by** (*auto simp*: *com-defs*)
ultimately show *?match*
using *BC-FrVal* **by** (*intro matchI*[*of s1 ?a1 ?ou1 ?s1' vl1 vl1''*]) (*auto simp*: Δ*2-def*)
qed
qed (*auto simp*: *consume-def*)
then show *?match* ∨ *?ignore* **..**
next
assume *n*φ: ¬φ *?trn*
then have *vl'*: *vl'* = *vl* **using** *c* **by** (*auto simp*: *consume-def*)
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by** (*cases step s1 a*)
let *?trn1* = *Trans s1 a ou1 s1'*
show *?match* ∨ *?ignore*
proof (*cases* ∀ *aID uID'*. *uID'* ∉ *UIDs aID* ⟶
a ≠ *COMact* (*comSendCreateOFriend UID* (*pass s UID*) *aID uID'*) ∧
a ≠ *COMact* (*comSendDeleteOFriend UID* (*pass s UID*) *aID uID'*))
case *True*
then have *n*φ*1*: ¬φ *?trn1*
using *n*φ *ss1 rs rs1 step step1* **by** (*auto simp*: *eqButUID-step-*φ)
have *?match* **using** *step1* **unfolding** *vl'* **proof** (*intro matchI*[*of s1 a ou1 s1' vl1 vl1*])
show *c1*: *consume ?trn1 vl1 vl1* **using** *n*φ*1* **by** (*auto simp*: *consume-def*)

show Δ*2 s′ vl s1′ vl1* **using** *BC* **unfolding** Δ*2-def* **proof** (*intro conjI*)
show *eqButUID s′ s1′* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*]
.
show ¬*open s1′* **proof**
assume *open s1′*
with *os* **have** *open s1* ≠ *open s1′* **by** *auto*
then show *False* **using** *step1 nφ1* **by** (*elim open-step-cases*[*of s1 s1′*]) *auto*
qed
show *validValSeqFrom vl1 s1′*
using *c1 rs1 vVS1* **by** (*intro step-validValSeqFrom*[*OF step1*]) *auto*
qed *auto*
show γ *?trn* = γ *?trn1* **using** *ss1 rs rs1 step step1 True* **by** (*intro eqButUID-step-*γ) *auto*
next
assume γ *?trn*
then have *ou* = *ou1* **using** *os nφ nφ1* **by** (*intro eqButUID-step-γ-out*[*OF ss1 step step1*]) *auto*
then show *g ?trn* = *g ?trn1* **by** (*cases a*) *auto*
qed *auto*
then show *?match* ∨ *?ignore* **..**
next
case *False*
with *nφ* **have** *?ignore*
using *UID-UIDs BC step ss1 os vVS1* **unfolding** *vl′*
by (*intro ignoreI*) (*auto simp*: Δ*2-def split*: *prod.splits*)
then show *?match* ∨ *?ignore* **..**
qed
qed
qed
with *BC* **show** *?thesis* **by** (*cases rule*: *BC.cases*) *auto*
qed
qed

definition *Gr* **where**
Gr =
{
(Δ*0*, {Δ*1*,Δ*2*}),
(Δ*1*, {Δ*1*,Δ*2*}),
(Δ*2*, {Δ*2*})
}

theorem *secure*: *secure*
apply (*rule unwind-decomp-secure-graph*[*of Gr* Δ*0*])
unfolding *Gr-def*
apply (*simp*, *smt insert-subset order-refl*)
using
*istate-*Δ*0 unwind-cont-*Δ*0 unwind-cont-*Δ*1 unwind-cont-*Δ*2*

unfolding *Gr-def* **by** (*auto intro*: *unwind-cont-mono*)

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* ⇒ *bool* **where**
γ (*Trans - a ou -*) ⟷ (∃ *uid*. *userOfA a* = *Some uid* ∧ *uid* ∈ *UIDs AID′*) ∨
(∃ *ca*. *a* = *COMact ca* ∧ *ou* ≠ *outErr*)

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* (*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* ∧ *uID* = *UID* ∧ *uID′* ∉ *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* ∧ *uID* = *UID* ∧ *uID*′ ∉ *UIDs AID*′
then comReceiveCreateOFriend aID cp uID emptyUserID
else 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*′. *ca* = *comSendCreateOFriend uID p*′ *aID uID*′ ∧ *p* = *emptyPass*)
comPurge ca = *comReceiveCreateOFriend aID cp uID uID*′
⟷ (∃ *uid*′′. (*ca* = *comReceiveCreateOFriend aID cp uID uid*′′ ∨ *ca* = *comReceiveDeleteOFriend aID cp uID uid*′′) ∧ *aID* = *AID* ∧ *uID* = *UID* ∧ *uid*′′ ∉ *UIDs AID*′ ∧ *uID*′ = *emptyUserID*)
∨ (*ca* = *comReceiveCreateOFriend aID cp uID uID*′ ∧ ¬(*aID* = *AID* ∧ *uID* = *UID* ∧ *uID*′ ∉ *UIDs AID*′))
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*′ ∧ ¬(*aID* = *AID* ∧ *uID* = *UID* ∧ *uID*′ ∉ *UIDs AID*′)
by (*cases ca*; *auto*)+

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 *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*′)

$\longleftrightarrow$ *a* = *COMact* (*comConnectServer aID sp*) ∧ *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*) ∧ *ou* = *ou′*
g (*Trans s a ou s′*) = (*COMact* (*comSendPost uID p aID nID*), *ou′*)
$\longleftrightarrow$ (∃ *p′*. *a* = *COMact* (*comSendPost uID p′ aID nID*) ∧ *p* = *emptyPass* ∧ *ou* = *ou′*)
g (*Trans s a ou s′*) = (*COMact* (*comSendCreateOFriend uID p aID uID′*), *ou′*)
$\longleftrightarrow$ (∃ *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′*)
$\longleftrightarrow$ (((∃ *uid″*. (*a* = *COMact* (*comReceiveCreateOFriend aID cp uID uid″*) ∨ *a* = *COMact* (*comReceiveDeleteOFriend aID cp uID uid″*)) ∧ *aID* = *AID* ∧ *uID* = *UID* ∧ *uid″* ∉ *UIDs AID′* ∧ *uID′* = *emptyUserID*)
∨ (*a* = *COMact* (*comReceiveCreateOFriend aID cp uID uID′*) ∧ ¬(*aID* = *AID* ∧ *uID* = *UID* ∧ *uID′* ∉ *UIDs AID′*)))
∧ *ou* = *ou′*)
g (*Trans s a ou s′*) = (*COMact* (*comSendDeleteOFriend uID p aID uID′*), *ou′*)
$\longleftrightarrow$ (∃ *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′*)
$\longleftrightarrow$ *a* = *COMact* (*comReceiveDeleteOFriend aID cp uID uID′*) ∧ ¬(*aID* = *AID* ∧ *uID* = *UID* ∧ *uID′* ∉ *UIDs AID′*) ∧ *ou* = *ou′*
by (*cases a*; *auto simp*: *comPurge-simps*)+

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* × *userID*) *list* ⇒ (*apiID* × *userID*) *list* ⇒ *bool* **where**
eqButUIDl auidl auidl1 = (*remove1* (*AID*,*UID*) *auidl* = *remove1* (*AID*,*UID*) *auidl1*)

lemma *eqButUIDl-eq*[*simp*,*intro*!]: *eqButUIDl auidl auidl*
by *auto*

lemma *eqButUIDl-sym*:
assumes *eqButUIDl auidl auidl1*
shows *eqButUIDl auidl1 auidl*
using *assms* **by** *auto*

lemma *eqButUIDl-trans*:
assumes *eqButUIDl auidl auidl1* **and** *eqButUIDl auidl1 auidl2*
shows *eqButUIDl auidl auidl2*
using *assms* **by** *auto*

lemma *eqButUIDl-remove1-cong*:
assumes *eqButUIDl auidl auidl1*
shows *eqButUIDl* (*remove1 auid auidl*) (*remove1 auid auidl1*)
using *assms* **by** (*auto simp*: *remove1-commute*)

lemma *eqButUIDl-snoc-cong*:
assumes *eqButUIDl auidl auidl1*
and $auid' \in\in auidl \longleftrightarrow auid' \in\in auidl1$
shows *eqButUIDl* ($auidl\ \#\#\ auid'$) ($auidl1\ \#\#\ auid'$)
using *assms* **by** (*auto simp*: *remove1-append remove1-idem*)

definition *eqButUIDf* **where**
eqButUIDf frds frds1 $\equiv$
($\forall$ *uid. if uid* $\in$ *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*
unfolding *eqButUIDf-def* **by** *auto*

lemma *eqButUIDf-sym*:
assumes *eqButUIDf frds frds1* **shows** *eqButUIDf frds1 frds*
using *assms* **unfolding** *eqButUIDf-def*
by *auto*

lemma *eqButUIDf-trans*:
assumes *eqButUIDf frds frds1* **and** *eqButUIDf frds1 frds2*
shows *eqButUIDf frds frds2*
using *assms* **unfolding** *eqButUIDf-def* **by** *fastforce*

lemma *eqButUIDf-cong*:
assumes *eqButUIDf frds frds1*
and *uid* $\in$ *UIDs AID'* $\Longrightarrow$ *uu* = *uu1*
and *uid* $\notin$ *UIDs AID'* $\Longrightarrow$ *eqButUIDl uu uu1*
shows *eqButUIDf* (*frds* (*uid* := *uu*)) (*frds1*(*uid* := *uu1*))

using *assms* **unfolding** *eqButUIDf-def* **by** *auto*

lemma *eqButUIDf-UIDs*:
⟦*eqButUIDf frds frds1*; *uid* ∈ *UIDs AID'*⟧ ⟹ *frds uid* = *frds1 uid*
unfolding *eqButUIDf-def* **by** (*auto split*: *if-splits*)

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*
unfolding *eqButUID-def* **by** *auto*

lemma *eqButUID-sym*[*sym*]:
assumes *eqButUID s s1* **shows** *eqButUID s1 s*
using *assms eqButUIDf-sym* **unfolding** *eqButUID-def* **by** *auto*

lemma *eqButUID-trans*[*trans*]:
assumes *eqButUID s s1* **and** *eqButUID s1 s2* **shows** *eqButUID s s2*
using *assms eqButUIDf-trans* **unfolding** *eqButUID-def* **by** *metis*

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
using *assms* **unfolding** *eqButUID-def IDsOK-def*[*abs-def*] **by** *auto*

lemma *eqButUID-UIDs*:
eqButUID s s1 $\Longrightarrow$ *uid* $\in$ *UIDs AID'* $\Longrightarrow$ *recvOuterFriendIDs s uid = recvOuterFriendIDs s1 uid*
unfolding *eqButUID-def eqButUIDf-def* **by** *auto*

lemma *eqButUID-recvOuterFriends-UIDs*:
assumes *eqButUID s s1*
and *uid* $\neq$ *UID* $\vee$ *aid* $\neq$ *AID*
shows (*aid*, *uid*) $\in\in$ *recvOuterFriendIDs s uid'* $\longleftrightarrow$ (*aid*, *uid*) $\in\in$ *recvOuterFriendIDs s1 uid'*
using *assms* **unfolding** *eqButUID-def eqButUIDf-def*
proof –
  have (*aid*, *uid*) $\in\in$ *remove1* (*AID*,*UID*) (*recvOuterFriendIDs s uid'*)
    $\longleftrightarrow$ (*aid*, *uid*) $\in\in$ *remove1* (*AID*,*UID*) (*recvOuterFriendIDs s1 uid'*)
    using *assms* **unfolding** *eqButUID-def eqButUIDf-def* **by** (*cases uid'* $\in$ *UIDs AID'*) *auto*
  then show *?thesis* **using** *assms* **by** *auto*
qed

lemma *eqButUID-remove1-UID-recvOuterFriends*:
assumes *eqButUID s s1*

shows *remove1* (*AID*,*UID*) (*recvOuterFriendIDs s uid*) = *remove1* (*AID*,*UID*) (*recvOuterFriendIDs s1 uid*)
using *assms* **unfolding** *eqButUID-def eqButUIDf-def* **by** (*cases uid* ∈ *UIDs AID*′) *auto*

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

$\bigwedge$ *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|))*
unfolding *eqButUID-def* **by** *auto*

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* $\Rightarrow$ *bool* **where**
φ *(Trans s (COMact (comReceiveCreateOFriend aID cp uID uID')) ou s')* =
(aID = AID $\wedge$ *uID = UID* $\wedge$ *uID'* $\notin$ *UIDs AID'* $\wedge$ *ou = outOK)*
|
φ *(Trans s (COMact (comReceiveDeleteOFriend aID cp uID uID')) ou s')* =
(aID = AID $\wedge$ *uID = UID* $\wedge$ *uID'* $\notin$ *UIDs AID'* $\wedge$ *ou = outOK)*
|
φ - = *False*

fun *f* :: *(state,act,out) trans* $\Rightarrow$ *value* **where**
f (Trans s (COMact (comReceiveCreateOFriend aID cp uID uID')) ou s') = FrVal AID' uID' True
|
f (Trans s (COMact (comReceiveDeleteOFriend aID cp uID uID')) ou s') = FrVal AID' uID' False

|
f - = *undefined*

lemma *recvOFriend-eqButUID*:
assumes *step s a* = (*ou*, *s'*)
and *reach s*
and *a* = *COMact* (*comReceiveCreateOFriend AID cp UID uID'*) ∨ *a* = *COMact* (*comReceiveDeleteOFriend AID cp UID uID'*)
and *uID'* ∉ *UIDs AID'*
shows *eqButUID s s'*
using *assms reach-distinct-friends-reqs*(*4*) **unfolding** *eqButUID-def eqButUIDf-def*
by (*auto simp*: *com-defs remove1-idem remove1-append*)

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 −
note *facts* = *eqButUID-recvOuterFriends-UIDs*[*OF ss1*] *eqButUID-UIDs*[*OF ss1*] *eqButUID-remove1-UID-recvOuterFriends*[*OF ss1*]
note *simps* = *eqButUID-stateSelectors s-defs c-defs d-defs u-defs r-defs l-defs com-defs*
note *congs* = *eqButUID-cong eqButUIDf-cong eqButUIDl-snoc-cong eqButUIDl-remove1-cong*
from *assms* **show** *?thesis* **proof** (*cases a*)
case (*Sact sa*) **with** *assms* **show** *?thesis* **by** (*cases sa*) (*auto simp*: *simps congs*)
next
case (*Cact ca*) **with** *assms* **show** *?thesis* **by** (*cases ca*) (*auto simp*: *simps congs*)
next
case (*Uact ua*) **with** *assms* **show** *?thesis* **by** (*cases ua*) (*auto simp*: *simps congs*)
next
case (*Ract ra*) **with** *assms* **show** *?thesis* **by** (*cases ra*) (*auto simp*: *simps congs*)
next
case (*Lact la*) **with** *assms* **show** *?thesis* **by** (*cases la*) (*auto simp*: *simps congs*)
next
case (*COMact ca*)
with *assms* **show** *?thesis* **proof** (*cases* φ (*Trans s a ou s'*) ∨ φ (*Trans s1 a ou1 s1'*))
case *True*
then have *eqButUID s s'* **and** *eqButUID s1 s1'*
using *COMact rs rs1 recvOFriend-eqButUID*[*OF step*] *recvOFriend-eqButUID*[*OF step1*]
by (*cases ca*; *auto*)+

then show *eqButUID s′ s1′* **using** *ss1* **by** (*auto intro*: *eqButUID-sym eqButUID-trans*)
next
case *False*
then show *?thesis* **using** *assms facts COMact* **by** (*cases ca*) (*auto simp*: *simps intro*!: *congs*)
qed
next
case (*Dact da*) **with** *assms* **show** *?thesis* **by** (*cases da*) (*auto simp*: *simps congs*)
qed
qed

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 −
obtain *uid com-act* **where** *uid-a*: (*userOfA a* = *Some uid* ∧ *uid* ∈ *UIDs AID′*)
∨ (*a* = *COMact com-act* ∧ *ou* ≠ *outErr*)
using γ *UID-UIDs* **by** *fastforce*
note *simps* = *eqButUID-stateSelectors eqButUID-UIDs*[*OF ss1*] *r-defs s-defs c-defs com-defs l-defs u-defs d-defs*
note *facts* = *ss1 step step1 uid-a*
show *?thesis*
proof (*cases a*)
case (*Ract ra*) **then show** *?thesis* **using** *facts* **by** (*cases ra*) (*auto simp add*: *simps*)
next
case (*Sact sa*) **then show** *?thesis* **using** *facts* **by** (*cases sa*) (*auto simp add*: *simps*)
next
case (*Cact ca*) **then show** *?thesis* **using** *facts* **by** (*cases ca*) (*auto simp add*: *simps*)
next
case (*COMact ca*)
then show *?thesis* **using** *facts* **proof** (*cases ca*)
case (*comReceiveCreateOFriend aid sp uid uid′*)
with *facts* φ **show** *?thesis* **using** *COMact eqButUID-recvOuterFriends-UIDs*[*OF ss1*]
by (*auto simp*: *simps*)
next
case (*comReceiveDeleteOFriend aid sp uid uid′*)
with *facts* φ **show** *?thesis* **using** *COMact eqButUID-recvOuterFriends-UIDs*[*OF ss1*]

by (*auto simp*: *simps*)
qed (*auto simp*: *simps*)
next
case (*Lact la*)
then show *?thesis* **using** *facts* **proof** (*cases la*)
case (*lInnerPosts uid p*)
then have *o*: $\bigwedge$*nid. owner s nid = owner s1 nid*
and *n*: $\bigwedge$*nid. post s nid = post s1 nid*
and *nids*: *postIDs s = postIDs s1*
and *vis*: *vis s = vis s1*
and *fu*: $\bigwedge$*uid'. friendIDs s uid' = friendIDs s1 uid'*
and *e*: *e-listInnerPosts s uid p* $\longleftrightarrow$ *e-listInnerPosts s1 uid p*
using *ss1* **unfolding** *eqButUID-def l-defs* **by** *auto*
have *listInnerPosts s uid p = listInnerPosts s1 uid p*
unfolding *listInnerPosts-def o n nids vis fu* **..**
with *e* **show** *?thesis* **using** *Lact lInnerPosts step step1* **by** *auto*
qed (*auto simp add*: *simps*)
next
case (*Uact ua*) **then show** *?thesis* **using** *facts* **by** (*cases ua*) (*auto simp add*: *simps*)
next
case (*Dact da*) **then show** *?thesis* **using** *facts* **by** (*cases da*) (*auto simp add*: *simps*)
qed
qed

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'*) $\longleftrightarrow$ φ (*Trans s a ou s'*)
shows γ (*Trans s a ou s'*) = γ (*Trans s1 a ou1 s1'*)
using *assms eqButUID-step-*γ*-out*[*OF assms*] *eqButUID-step-*γ*-out*[*OF - step1 step*]
by (*auto intro*: *eqButUID-sym*)

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) $trans \Rightarrow bool$
where $T\ trn = False$

For each user uid at this receiver node AID', the remote friendship updates with the fixed user UID at the issuer node AID form an alternating sequence of friending and unfriending.

Note that actions involving remote users who are observers do not produce secret values; instead, those actions are observable, and the property we verify does not protect their confidentiality.

Moreover, there is no declassification trigger on the receiver side, so $OVal$ values are never produced by receiver nodes, only by the issuer node.

definition $friendsOfUID :: state \Rightarrow userID\ set$ **where**
$friendsOfUID\ s = \{uid.\ (AID,UID) \in\in recvOuterFriendIDs\ s\ uid \wedge uid \notin UIDs\ AID'\}$

fun $validValSeq :: value\ list \Rightarrow userID\ set \Rightarrow bool$ **where**
$validValSeq\ []\ \_ = True$
$|\ validValSeq\ (FrVal\ aid\ uid\ True\ \#\ vl)\ uids \longleftrightarrow uid \notin uids \wedge aid = AID' \wedge uid \notin UIDs\ AID' \wedge validValSeq\ vl\ (insert\ uid\ uids)$
$|\ validValSeq\ (FrVal\ aid\ uid\ False\ \#\ vl)\ uids \longleftrightarrow uid \in uids \wedge aid = AID' \wedge uid \notin UIDs\ AID' \wedge validValSeq\ vl\ (uids - \{uid\})$
$|\ validValSeq\ (OVal\ ov\ \#\ vl)\ uids \longleftrightarrow False$

abbreviation $validValSeqFrom\ vl\ s \equiv validValSeq\ vl\ (friendsOfUID\ s)$

Observers may learn about the occurrence of remote friendship actions (by observing network traffic), but not their content; remote friendship actions at a receiver node AID' can be replaced by different actions involving different users of that node (who are not observers) without affecting the observations.

inductive $BC :: value\ list \Rightarrow value\ list \Rightarrow bool$
where
$BC\text{-}Nil[simp,intro]$: $BC\ []\ []$
$|\ BC\text{-}FrVal[intro]$:
$BC\ vl\ vl1 \Longrightarrow uid' \notin UIDs\ AID' \Longrightarrow BC\ (FrVal\ aid\ uid\ st\ \#\ vl)\ (FrVal\ AID'\ uid'\ st'\ \#\ vl1)$

definition $B\ vl\ vl1 \equiv BC\ vl\ vl1 \wedge validValSeqFrom\ vl1\ istate$

lemma $BC\text{-}Nil\text{-}Nil$: $BC\ vl\ vl1 \Longrightarrow vl1 = [] \longleftrightarrow vl = []$
by ($induction\ rule$: $BC.induct$) $auto$

lemma $BC\text{-}id$: $validValSeq\ vl\ uids \Longrightarrow BC\ vl\ vl$
by ($induction\ rule$: $validValSeq.induct$) $auto$

lemma $BC\text{-}append$: $BC\ vl\ vl1 \Longrightarrow BC\ vl'\ vl1' \Longrightarrow BC\ (vl\ @\ vl')\ (vl1\ @\ vl1')$

by (*induction rule*: *BC.induct*) *auto*

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** $B = B$
done

9.2.5 Unwinding proof

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

lemma *istate-*$\Delta 0$:
assumes *B*: *B vl vl1*
shows $\Delta 0$ *istate vl istate vl1*
using *assms* **unfolding** $\Delta 0$*-def B-def*
by *auto*

lemma *friendsOfUID-cong*:
assumes *recvOuterFriendIDs s* = *recvOuterFriendIDs s'*
shows *friendsOfUID s* = *friendsOfUID s'*
using *assms* **unfolding** *friendsOfUID-def* **by** *auto*

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*
using *assms* **unfolding** *friendsOfUID-def* **by** (*auto simp*: *com-defs*)

lemma *friendsOfUID-step-Create-UID*:
assumes *uid'* ∉ *UIDs AID'*
shows *friendsOfUID* (*receiveCreateOFriend s AID sp UID uid'*) = *insert uid'* (*friendsOfUID s*)
using *assms* **unfolding** *friendsOfUID-def* **by** (*auto simp*: *com-defs*)

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'*}
using *assms reach-distinct-friends-reqs*(*4*) **unfolding** *friendsOfUID-def* **by** (*auto simp*: *com-defs*)

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 *cases*
  assume *φ ?trn*
  moreover then obtain *v* **where** *vl* = *v* # *vl′* **using** *c* **by** (*cases vl*, *auto simp*: *consume-def*)
  ultimately show *?thesis* **using** *assms*
   by (*elim φ.elims*) (*auto simp*: *consume-def friendsOfUID-step-Create-UID friendsOfUID-step-Delete-UID*)
next
  assume *nφ*: ¬*φ ?trn*
  then have *vl′*: *vl′* = *vl* **using** *c* **by** (*auto simp*: *consume-def*)
  then show *?thesis* **using** *vVS step* **proof** (*cases a*)
    case (*Sact sa*) **then show** *?thesis* **using** *assms vl′* **by** (*cases sa*) (*auto simp*: *s-defs cong*: *friendsOfUID-cong*) **next**
    case (*Cact ca*) **then show** *?thesis* **using** *assms vl′* **by** (*cases ca*) (*auto simp*: *c-defs cong*: *friendsOfUID-cong*) **next**
    case (*Dact da*) **then show** *?thesis* **using** *assms vl′* **by** (*cases da*) (*auto simp*: *d-defs cong*: *friendsOfUID-cong*) **next**
    case (*Uact ua*) **then show** *?thesis* **using** *assms vl′* **by** (*cases ua*) (*auto simp*: *u-defs cong*: *friendsOfUID-cong*) **next**
    case (*COMact ca*)
      then show *?thesis* **using** *assms vl′ nφ* **proof** (*cases ca*)
        case (*comReceiveCreateOFriend aid sp uid uid′*)
          then show *?thesis* **using** *COMact assms nφ* **by** (*auto simp*: *friendsOfUID-step-not-UID consume-def*)
      next
        case (*comReceiveDeleteOFriend aid sp uid uid′*)
          then show *?thesis* **using** *COMact assms nφ* **by** (*auto simp*: *friendsOfUID-step-not-UID consume-def*)
      qed (*auto simp*: *com-defs cong*: *friendsOfUID-cong*)
  qed *auto*
qed

lemma *unwind-cont-Δ0*: *unwind-cont Δ0* {*Δ0*}
proof(*rule*, *simp*)
  fix *s s1* :: *state* **and** *vl vl1* :: *value list*
  assume *rsT*: *reachNT s* **and** *rs1*: *reach s1* **and** *Δ0*: *Δ0 s vl s1 vl1*
  then have *rs*: *reach s* **and** *ss1*: *eqButUID s s1* **and** *BC*: *BC vl vl1*
      and *vVS1*: *validValSeqFrom vl1 s1*
    using *reachNT-reach* **unfolding** *Δ0-def* **by** *auto*
  show *iaction Δ0 s vl s1 vl1* ∨
      ((*vl* = [] ⟶ *vl1* = []) ∧ *reaction Δ0 s vl s1 vl1*) (**is** *?iact* ∨ (- ∧ *?react*))
  proof−
    have *?react* **proof**
      fix *a* :: *act* **and** *ou* :: *out* **and** *s′* :: *state* **and** *vl′*
      let *?trn* = *Trans s a ou s′*
     assume *step*: *step s a* = (*ou*, *s′*) **and** *T*: ¬ *T ?trn* **and** *c*: *consume ?trn vl vl′*

show *match* $\Delta 0$ *s s1 vl1 a ou s′ vl′* ∨ *ignore* $\Delta 0$ *s s1 vl1 a ou s′ vl′* (**is** *?match* ∨ *?ignore*)
proof *cases*
assume φ: φ *?trn*
with *BC c* **have** *?match* **proof** (*cases rule*: *BC.cases*)
case (*BC-FrVal vl′′ vl1′′ uid′ aid uid st st′*)
then show *?thesis* **proof** (*cases st′*)
case *True*
let *?a1* = *COMact* (*comReceiveCreateOFriend AID* (*serverPass s AID*) *UID uid′*)
let *?ou1* = *outOK*
let *?s1′* = *receiveCreateOFriend s1 AID* (*serverPass s AID*) *UID uid′*
let *?trn1* = *Trans s1 ?a1 ?ou1 ?s1′*
have *c1*: *consume ?trn1 vl1 vl1′′* **and** *vl′* = *vl′′* **and** *f ?trn* = *FrVal AID′ uid st*
using φ *c BC-FrVal True* **by** (*auto elim*: φ*.elims simp*: *consume-def*)
moreover then have *a*: *a* = *COMact* (*comReceiveCreateOFriend AID* (*serverPass s AID*) *UID uid*)
∨ *a* = *COMact* (*comReceiveDeleteOFriend AID* (*serverPass s AID*) *UID uid*)
and *ou*: *ou* = *outOK*
and *IDs*: *IDsOK s* [] [] [(*AID*,[])] []
and *uid*: *uid* ∉ *UIDs AID′*
using φ *step rs* **by** (*auto elim*!: φ*.elims split*: *prod.splits simp*: *com-defs*)
moreover have *step1*: *step s1 ?a1* = (*?ou1*, *?s1′*)
using *IDs vVS1 BC-FrVal True ss1* **by** (*auto simp*: *com-defs eqButUID-def friendsOfUID-def*)
moreover then have *validValSeqFrom vl1′′ ?s1′*
using *vVS1 rs1 c1* **by** (*intro step-validValSeqFrom*[*OF step1*]) *auto*
moreover have *eqButUID s′ ?s1′*
using *ss1 recvOFriend-eqButUID*[*OF step rs a uid*]
using *recvOFriend-eqButUID*[*OF step1 rs1*, *of serverPass s AID uid′*] *BC-FrVal*(*4*)
by (*auto intro*: *eqButUID-sym eqButUID-trans*)
moreover have γ *?trn* = γ *?trn1* **and** *g ?trn* = *g ?trn1*
using *BC-FrVal a ou uid* **by** (*auto simp*: *com-defs*)
ultimately show *?match*
using *BC-FrVal* **by** (*intro matchI*[*of s1 ?a1 ?ou1 ?s1′ vl1 vl1′′*]) (*auto simp*: $\Delta 0$*-def*)
next
case *False*
let *?a1* = *COMact* (*comReceiveDeleteOFriend AID* (*serverPass s AID*) *UID uid′*)
let *?ou1* = *outOK*
let *?s1′* = *receiveDeleteOFriend s1 AID* (*serverPass s AID*) *UID uid′*
let *?trn1* = *Trans s1 ?a1 ?ou1 ?s1′*
have *c1*: *consume ?trn1 vl1 vl1′′* **and** *vl′* = *vl′′* **and** *f ?trn* = *FrVal AID′ uid st*

using φ *c BC-FrVal False* **by** (*auto elim*: φ*.elims simp*: *consume-def*)
moreover then have *a*: *a* = *COMact* (*comReceiveCreateOFriend AID* (*serverPass s AID*) *UID uid*)
∨ *a* = *COMact* (*comReceiveDeleteOFriend AID* (*serverPass s AID*) *UID uid*)
and *ou*: *ou* = *outOK*
and *IDs*: *IDsOK s* [] [] [(*AID*,[])] []
and *uid*: *uid* ∉ *UIDs AID'*
using φ *step rs* **by** (*auto elim*!: φ*.elims split*: *prod.splits simp*: *com-defs*)
moreover have *step1*: *step s1 ?a1* = (*?ou1*, *?s1'*)
using *IDs vVS1 BC-FrVal False ss1* **by** (*auto simp*: *com-defs eqButUID-def friendsOfUID-def*)
moreover then have *validValSeqFrom vl1'' ?s1'*
using *vVS1 rs1 c1* **by** (*intro step-validValSeqFrom*[*OF step1*]) *auto*
moreover have *eqButUID s' ?s1'*
using *ss1 recvOFriend-eqButUID*[*OF step rs a uid*]
using *recvOFriend-eqButUID*[*OF step1 rs1*, *of serverPass s AID uid'*] *BC-FrVal*(*4*)
by (*auto intro*: *eqButUID-sym eqButUID-trans*)
moreover have γ *?trn* = γ *?trn1* **and** *g ?trn* = *g ?trn1*
using *BC-FrVal a ou uid* **by** (*auto simp*: *com-defs*)
ultimately show *?match*
using *BC-FrVal* **by** (*intro matchI*[*of s1 ?a1 ?ou1 ?s1' vl1 vl1''*]) (*auto simp*: Δ*0-def*)
qed
qed (*auto simp*: *consume-def*)
then show *?match* ∨ *?ignore* **..**
next
assume *n*φ: ¬φ *?trn*
then have *vl'*: *vl'* = *vl* **using** *c* **by** (*auto simp*: *consume-def*)
obtain *ou1 s1'* **where** *step1*: *step s1 a* = (*ou1*, *s1'*) **by** (*cases step s1 a*)
let *?trn1* = *Trans s1 a ou1 s1'*
show *?match* ∨ *?ignore*
proof (*cases* ∀ *uID'*. *uID'* ∉ *UIDs AID'* ⟶
a ≠ *COMact* (*comReceiveCreateOFriend AID* (*serverPass s1 AID*) *UID uID'*) ∧
a ≠ *COMact* (*comReceiveDeleteOFriend AID* (*serverPass s1 AID*) *UID uID'*))
case *True*
then have *n*φ*1*: ¬φ *?trn1* **using** *step1* **by** (*auto elim*!: φ*.elims simp*: *com-defs*)
have *?match* **using** *step1* **unfolding** *vl'* **proof** (*intro matchI*[*of s1 a ou1 s1' vl1 vl1*])
show *c1*: *consume ?trn1 vl1 vl1* **using** *n*φ*1* **by** (*auto simp*: *consume-def*)
show Δ*0 s' vl s1' vl1* **using** *BC* **unfolding** Δ*0-def* **proof** (*intro conjI*)
show *eqButUID s' s1'* **using** *eqButUID-step*[*OF ss1 step step1 rs rs1*]
.
show *validValSeqFrom vl1 s1'*

```
        using c1 rs1 vVS1 by (intro step-validValSeqFrom[OF step1]) auto
      qed auto
      show γ ?trn = γ ?trn1 using ss1 rs rs1 step step1 True nφ nφ1
        by (intro eqButUID-step-γ) auto
    next
      assume γ ?trn
     then have ou = ou1 using nφ nφ1 by (intro eqButUID-step-γ-out[OF
ss1 step step1]) auto
      then show g ?trn = g ?trn1 by (cases a) auto
    qed auto
    then show ?match ∨ ?ignore ..
  next
    case False
    with nφ have ?ignore
      using UID-UIDs BC step ss1 vVS1 unfolding vl'
      by (intro ignoreI) (auto simp: Δ0-def split: prod.splits)
    then show ?match ∨ ?ignore ..
  qed
 qed
qed
with BC show ?thesis by (cases rule: BC.cases) auto
 qed
qed


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


theorem secure: secure
apply (rule unwind-decomp-secure-graph[of Gr Δ0])
unfolding Gr-def

using
istate-Δ0 unwind-cont-Δ0
unfolding Gr-def by (auto intro: unwind-cont-mono)


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* **using** *UID-UIDs* **by** *unfold-locales*

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 (*unfold-locales*, *goal-cases*)
case (*1 aid trn*)
interpret *Receiver*: *OuterFriendReceiver UIDs AID UID aid* **by** *unfold-locales*
from *1* **show** *?case* **by** (*cases trn*) (*auto elim*!: *Issuer*.φ*E Receiver*.φ.*elims split*: *prod.splits*)
next
case (*2 aid trn*)
interpret *Receiver*: *OuterFriendReceiver UIDs AID UID aid* **by** *unfold-locales*
from *2* **show** *?case* **by** (*cases trn*) (*auto elim*!: *Issuer*.φ*E Receiver*.φ.*elims*)
next
case (*3 aid trn*)
interpret *Receiver*: *OuterFriendReceiver UIDs AID UID aid* **by** *unfold-locales*
from *3* **show** *?case* **by** (*cases* (*aid*,*trn*) *rule*: *tgtNodeOf.cases*) (*auto*)
next
case (*4 aid trn*)
interpret *Receiver*: *OuterFriendReceiver UIDs AID UID aid* **by** *unfold-locales*
from *4* **show** *?case* **by** (*cases* (*aid*,*trn*) *rule*: *tgtNodeOf.cases*) *auto*
next
case (*5 aid1 trn1 aid2 trn2*)
interpret *Receiver1*: *OuterFriendReceiver UIDs AID UID aid1* **by** *unfold-locales*
interpret *Receiver2*: *OuterFriendReceiver UIDs AID UID aid2* **by** *unfold-locales*
from *5* **show** *?case* **by** (*elim sync-cases*) (*auto simp*: *com-defs*)
next
case (*6 aid1 trn1 aid2 trn2*)
interpret *Receiver1*: *OuterFriendReceiver UIDs AID UID aid1* **by** *unfold-locales*
interpret *Receiver2*: *OuterFriendReceiver UIDs AID UID aid2* **by** *unfold-locales*
from *6* **show** *?case* **by** (*elim sync-cases*) (*auto*)
next
case (*7 aid1 trn1 aid2 trn2*)
interpret *Receiver1*: *OuterFriendReceiver UIDs AID UID aid1* **by** *unfold-locales*
interpret *Receiver2*: *OuterFriendReceiver UIDs AID UID aid2* **by** *unfold-locales*
from *7* **show** *?case*
using *Issuer.COMact-open*[*of srcOf trn1 actOf trn1 outOf trn1 tgtOf trn1*]
using *Issuer.COMact-open*[*of srcOf trn2 actOf trn2 outOf trn2 tgtOf trn2*]
by (*elim sync-cases*) *auto*
next
case (*8 aid1 trn1 aid2 trn2*)
interpret *Receiver1*: *OuterFriendReceiver UIDs AID UID aid1* **by** *unfold-locales*
interpret *Receiver2*: *OuterFriendReceiver UIDs AID UID aid2* **by** *unfold-locales*
assume *comOf aid1 trn1* = *Send comOf aid2 trn2* = *Recv syncO aid1* (*g aid1 trn1*) *aid2* (*g aid2 trn2*)
φ *aid1 trn1* ⟹ φ *aid2 trn2* ⟹ *f aid1 trn1* = *f aid2 trn2*

validTrans aid1 trn1 validTrans aid2 trn2
then show *?case* **using** *emptyUserID-not-UIDs*
by (*elim syncO-cases*; *cases trn1*; *cases trn2*)
(*auto simp*: *Issuer.g-simps Receiver1.g-simps Receiver2.g-simps simp*: *com-defs*)
next
case (*9 aid trn*)
interpret *Receiver*: *OuterFriendReceiver UIDs AID UID aid* **by** *unfold-locales*
from *9* **show** *?case* **by** (*cases* (*aid*,*trn*) *rule*: *tgtNodeOf.cases*) (*auto*)
next
case (*10 aid trn*)
interpret *Receiver*: *OuterFriendReceiver UIDs AID UID aid* **by** *unfold-locales*
from *10* **show** *?case* **using** *AID-AIDs* **by** (*auto elim*!: *Receiver.φ.elims*)
next
case (*11 vSrc nid vn*) **then show** *?case* **by** (*cases vSrc*) *auto*
next
case (*12 vSrc nid vn*) **then show** *?case* **by** (*cases vSrc*) *auto*
qed

context
fixes *AID′* :: *apiID*
assumes *AID′*: $AID' \in AIDs - \{AID\}$
begin

interpretation *Receiver*: *OuterFriendReceiver UIDs AID UID AID′* **by** *unfold-locales*

lemma *Issuer-BC-Receiver-BC*:
assumes *Issuer.BC vl vl1*
shows *Receiver.BC* (*Net.projectSrcV AID′ vl*) (*Net.projectSrcV AID′ vl1*)
using *assms* **by** (*induction rule*: *Issuer.BC.induct*) *auto*

lemma *Collect-setminus*: $Collect\ P - A = \{u.\ u \notin A \wedge P\ u\}$
by *auto*

lemma *Issuer-vVS-Receiver-vVS*:
assumes *Issuer.validValSeq vl auidl*
shows *Receiver.validValSeq* (*Net.projectSrcV AID′ vl*) {*uid.* (*AID′*,*uid*) ∈∈ *auidl*}
using *assms AID′*
proof (*induction vl auidl rule*: *Issuer.validValSeq.induct*)
case (*2 aid uid vl auidl*)
then show *?case* **by** (*auto simp*: *insert-Collect Collect-setminus*, *linarith*, *smt Collect-cong*)
next
case (*3 aid uid vl auidl*)
then show *?case* **by** (*auto simp*: *insert-Collect Collect-setminus*; *smt Collect-cong*)
qed *auto*

lemma *Issuer-B-Receiver-B*:
assumes *Issuer.B vl vl1*

shows *Receiver.B* (*Net.projectSrcV AID′ vl*) (*Net.projectSrcV AID′ vl1*)
using *assms Issuer-BC-Receiver-BC Issuer-vVS-Receiver-vVS*[*of* - []]
unfolding *Issuer.B-def Issuer.BO-def Receiver.B-def Receiver.friendsOfUID-def*
by (*auto simp*: *istate-def intro*!: *Receiver.BC-append Receiver.BC-id*, *blast dest*: *Issuer.validValSeq-prefix*)

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** 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*
using *AID-AIDs Issuer-B-Receiver-B Issuer.secure*
by *unfold-locales auto*

theorem *secure*: *secure*
proof (*intro preserve-source-secure ballI*)
fix *aid*
interpret *Receiver*: *OuterFriendReceiver UIDs AID UID aid* **by** *unfold-locales*
assume *aid* ∈ *AIDs* − {*AID*}
then show *Net.lsecure aid* **using** *Receiver.secure* **by** *auto*
qed

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*, vol-

ume 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.