CoSMeDis: A Distributed Social Media Platform with Formally Verified Confidentiality Guarantees

Thomas Bauereiß*, Armando Pesenti Gritti†, Andrei Popescu‡§, Franco Raimondi‡
*German Research Center for Artificial Intelligence (DFKI) Bremen, Germany
†Global NoticeBoard, UK
‡Department of Computer Science, Middlesex University London, UK
§Institute of Mathematics Simion Stoilow of the Romanian Academy

***Abstract*— We present the design, implementation and information flow verification of CoSMeDis, a distributed social media platform. The system consists of an arbitrary number of communicating nodes, deployable at different locations over the Internet. Its registered users can post content and establish intra-node and inter-node friendships, used to regulate access control over the posts. The system's kernel has been verified in the proof assistant Isabelle/HOL and automatically extracted as Scala code. We formalized a framework for composing a class of information flow security guarantees in a distributed system, applicable to input/output automata. We instantiated this framework to confidentiality properties for CoSMeDis's sources of information: posts, friendship requests, and friendship status.**

I. Introduction

Recent years have seen an explosion of web-based systems aimed at sharing information between multiple users in a convenient, but controlled fashion. Examples include enterprise systems, social networks, e-commerce sites and cloud services. These systems often deal with confidential information, such as credit card details, medical data, location information and sensitive documents. Unfortunately, most of these systems offer no guarantees concerning the prevention of *unintended flow* of information. Programming errors or ambiguous policy specifications leading to information leakage can have different degrees of severity and can affect many users [2,3].

Such errors are difficult to prevent, or even understand, because information flow security is a complex, global property of a program. The problem is aggravated by the heavy inter-connectivity of today's web applications. Sensitive user information in cloud storage systems, such as Amazon, may be shared with social media platforms, such as Twitter or Facebook, and web-based email systems, such as Gmail. While often the information sharing occurs via intuitively secure protocols, it is far more difficult to foresee the confidentiality issues arising synergistically in a network of applications. *What is needed is a solid, mathematically grounded understanding of how local confidentiality guarantees can be* composed *to deliver guarantees for the entire distributed system.*

In this paper, we develop such a theory of compositionality (i.e., the notion of composing local guarantees to form global guarantees) and apply it to a major case study: a distributed social media platform. We employ a notion of information flow security introduced by Kanav et al. [38], called Bounded Deducibility (BD) Security. The appeal of BD Security for dealing with web applications rests in its versatile mechanism for specifying rich security policies, inspired by epistemic logic. This notion has been formalized in the proof assistant Isabelle/HOL [53] and then utilized in confidentiality guarantees for two running web applications:

- CoCon [1] (introduced as a case study by the BD Security authors), a conference management system that has been deployed for two verification-friendly conferences [14,23];
- CoSMed ([7], Section II), a social media platform that we developed in previous work [12] as a prototype aimed at eventually serving the functionality and security needs of a charity organization [6].

The attacker models in these works (Section III) consider an arbitrary but fixed set of users, called *observers*, who interact with the system and from whom secret information should be protected. It is verified that the systems do not leak more information about the secret than specified by given policies.

In this paper, we design and implement a distributed version of CoSMed, which we call CoSMeDis, and extend its formal security guarantees. The extension consists of a communication infrastructure (Section IV), which introduces new security-critical aspects: (1) the ability to share secrets, namely, restricted-access post contents, between different nodes and (2) the ability to dynamically assign secret-sharing roles, by the designation of friends from remote nodes. The attacker model is lifted to sets of observing users at each node of the distributed system. Moreover, we assume that network traffic between the nodes is also observable, with the exception of the confidential content of communication. This is in line with assuming a Dolev-Yao-style network attacker [25], who can inspect communication without breaking encryption.

After a detailed analysis of how to cope compositionally with the security of CoSMeDis's features (Section V), we formulate a solution in abstract terms: as a framework for composing information flow security guarantees of input/output (I/O) automata (Section VI). At its heart lies a formal theorem for composing locally verified BD Security instances to a global BD Security property of the entire distributed system. The theorem is policy-agnostic, in that it composes policies without restrictions on their content, but requires a unique source node for each secret.

We instantiate the theorem for extending to CoSMeDis all the previously established CoSMed guarantees about the confidentiality of posts and friendship information and to prove a new guarantee about remote friendship (Section VII). CoSMeDis is a large piece of formal engineering, integrating

specification, code generation and verification. We discuss the scope of our verification (Section VIII) and compare our work with results from the literature (Section IX). The CoSMeDis homepage [5] has links to the formal proofs, the source code, documentation, and sample running nodes.

II. The Original CoSMed

This section recalls the original CoSMed system. It is a social media platform loosely inspired by platforms such as Facebook. It allows users to register and post information and to restrict access to this information based on friendship relationships established between users. For example, the user Alice can log into the web site and browse through posts made by other users. She can create new posts herself, e.g., a comment on a sports event. By default, this post is visible to her friends only. She can add new friends by looking up their profile, e.g., the profile of Bob, and requesting friendship by optionally entering a greeting text and clicking the submit button. When Bob approves the request, they become friends, and Bob can now see Alice's sports comment. Alice can also edit her posts and set their visibility level—either friends-only or public—at any time. The system has one user with special powers, the admin, who is responsible for approving the creation of users.

A. System Model

State. CoSMed has a mutable state, storing information on users, posts and friendships. For example: A user ID has an associated name, email address and info; a post ID has an associated title, text, image, owner ID and visibility; a friendship request is identified by two user IDs (the sender and the recipient) and has an associated greeting text created by the sender; the friendship status ("friend" or "not friend") is stored as a symmetric association between user IDs.

Actions. Users interact with the system via actions for creating, deleting, updating and reading items in the system, where an item can be a user, a post, a friendship request or a friendship status. There are also actions for listing items according to various filters or criteria—e.g., a user can list all posts, or all posts of a given friend, or all his friends, or all the friends of a given friend, etc. Each action a also contains the user ID of its issuer, denoted by $\mathsf{userOf}(a)$.

Outputs. When a user requests an action, the system first checks if the action is allowed, in which case the action is applied and the output is returned to the user; otherwise, an error message is emitted and the state remains unchanged.

B. Formalization and Implementation

The above behavior is formalized in Isabelle/HOL as an I/O automaton $\mathsf{Aut} = (\mathsf{State}, \mathsf{Act}, \mathsf{Out}, \sigma^0, \rightarrow)$, where $\sigma^0 \in \mathsf{State}$ is the (empty) *initial state* and $\rightarrow \subseteq \mathsf{State} \times \mathsf{Act} \times \mathsf{Out} \times \mathsf{State}$ is the transition relation. The inputs of the automaton are called *actions*. For example, an action that updates the content of a post has the form $(\mathsf{updatePost}, \mathit{uid}, \mathit{pid}, \mathit{pst})$. Here, updatePost is a label indicating the particular type of action, *uid* is the ID of the acting user, *pid* is the ID of the post, and *pst* is the new content. $\sigma \xrightarrow[o]{a} \sigma'$ has the following reading: if a user takes action a while the system is in state σ, the system responds producing output o and changing the state to σ'.

A system *transition* is a tuple $\mathit{trn} = (\sigma, a, o, \sigma')$ such that $\sigma \xrightarrow[o]{a} \sigma'$ holds. The states σ and σ' are called the transition's *source* and *target*. The transition's action a is also denoted by $\mathsf{actOf}(\mathit{trn})$. A system *trace* is a sequence $\mathit{tr} = \langle \mathit{trn}_1, \ldots, \mathit{trn}_n \rangle$ of transitions such that the source of the first transition (if any) is σ_0 and the target of each transition is the source of its successor in the sequence. The *end state* of a trace *tr*, written $\mathsf{endState}(\mathit{tr})$, is the target state of *tr*'s last transition if *tr* is non-empty, and the initial state σ^0 otherwise. Note that a system trace interleaves transitions containing actions from, and outputs to, different users. We let Trans denote the set of transitions and Trace the set of traces.

CoSMed's transition relation is deterministic, and in fact is represented as an executable function $\mathsf{Act} \times \mathsf{State} \rightarrow \mathsf{State} \times \mathsf{Out}$. This function is automatically extracted into Scala code using Isabelle's code generator [32], forming CoSMed's kernel. Around the extracted code, we implemented manually, in the Scalatra web framework [61], a layer of web-specific code that provides CoSMed's user interface, invoking kernel actions in response to user requests [12, §2.2].

III. Security Model: BD Security

Previously [12], we proved for CoSMed confidentiality properties of the form: Only under the circumstances specified by a given policy may users learn information on the system's documents. Our running example in this paper will be:

(P_1) A group of users can *learn* nothing about the updates to a post content *beyond* the existence of an update *unless* one of them is the admin or the post's owner, or becomes friends with the owner, or the post is marked as public.

Let us analyze what (P_1) expresses and how this is formalized. It focuses on the confidentiality of the content of a given post by a CoSMed user. It does not describe a policy for *access control* to this data, but rather a policy for *information flow control*. Indeed, it does not state that the post content cannot be accessed, but something stronger: that information about it cannot flow. Hence we write "learn" instead of "access."

The *(partial) secret* that (P_1) refers to is the content of a particular post, say, stored in the system under the ID PID. Since the content can be updated by the owner several times, we need to speak of a *sequence* of secrets: all the updates to the content, corresponding to all its versions held in the state during a system run. Formally, one defines a domain Sec of secrets, a filter $\mathsf{isSec} : \mathsf{Trans} \rightarrow \mathsf{Bool}$ and a secret-producing function $\mathsf{getSec} : \mathsf{Trans} \rightarrow \mathsf{Sec}$. This yields a function $\mathsf{S}^{\mathsf{getSec}}_{\mathsf{isSec}} : \mathsf{Trace} \rightarrow \mathsf{Sec}^*$ that extracts a sequence of secrets from a trace transition-wise, by filtering with isSec and then applying getSec to each of the trace's transitions. For example, consider the trace $\mathit{tr} = \langle \mathit{trn}_1, \mathit{trn}_2, \mathit{trn}_3 \rangle$ where isSec is only true for trn_1 and trn_3, then $\mathsf{S}^{\mathsf{getSec}}_{\mathsf{isSec}}(\mathit{tr}) = \langle \mathsf{getSec}(\mathit{trn}_1), \mathsf{getSec}(\mathit{trn}_3) \rangle$. For ($\mathsf{P}_1$), Sec is obviously taken to consist of post contents. Moreover, $\mathsf{isSec}\ (\sigma, a, o, \sigma')$ holds if a is a post-update action for PID, i.e., has the form $(\mathsf{updatePost}, \mathit{uid}, \mathsf{PID}, \mathit{pst})$ for some *uid*, *pst*, and the transition is successful, as signaled by an "OK" output. The function getSec for (P_1) extracts the post content from a: $\mathsf{getSec}\ (\sigma, a, o, \sigma') = \mathit{pst}$.

The *observers* (possible attackers) are here a group of users, say, with their IDs in a fixed (but arbitrary) set UIDs. Formally,

observations are managed similarly to the secrets. One defines the observation domain Obs, the filter $\mathsf{isObs} : \mathsf{Trans} \to \mathsf{Bool}$ and the production function $\mathsf{getObs} : \mathsf{Trans} \to \mathsf{Obs}$, and then defines $\mathsf{O}^{\mathsf{getObs}}_{\mathsf{isObs}} : \mathsf{Trace} \to \mathsf{Obs}^*$ by filtering with isObs and applying getObs transition-wise. For (P_1), Obs is taken to be the set of all action-output pairs. $\mathsf{isObs}\,(\sigma, a, o, \sigma')$ is true just in case the action is issued by one of the designated observers ($\mathsf{userOf}(a) \in \mathsf{UIDs}$), and getObs retrieves the action and output from the transition ($\mathsf{getObs}\,(\sigma, a, o, \sigma') = (a, o)$). Note that this includes failed actions, i.e., errors must not leak secrets.

So what can the observers learn about the secrets? According to (P_1), nothing *beyond* the (non-)existence of at least one update. This is formalized as a *bound*, i.e., a binary relation B between sequences of secrets. We require sequences of secrets related by B to be exchangeable, without interfering with the observations. Intuitively, given the sequence of secrets sl produced by a system trace, the set $\{sl' \mid \mathsf{B}(sl, sl')\}$ represents the amount of uncertainty of the observers: for what they know, sl could be any sl' in this set. Hence, B describes a lower bound on the uncertainty, i.e., an upper bound on the declassification (i.e., controlled release of designated secrets to designated observers). For (P_1), $\mathsf{B}(sl, sl')$ is defined as "sl empty implies sl' empty." Thus, the only piece of information that can flow to the observers is a harmless one: that no update has been performed yet (possibly because the post has not been created yet).

(P_1) also prescribes a legitimate way out of the declassification bound: if one of the observers has or acquires a role (system admin, post owner or owner's friend) or an intended declassification happens (the post is made public). This is formalized as a *trigger*, a unary predicate T on system transitions. The bound B is only imposed *unless* the trigger T occurs. Here, $\mathsf{T}\,(\sigma, a, o, \sigma')$ is defined as a property of the transition's target state σ': that the system's admin is in UIDs ($\mathsf{admin}(\sigma') \in \mathsf{UIDs}$), the registered owner of PID is in UIDs ($\mathsf{owner}(\sigma', \mathsf{PID}) \in \mathsf{UIDs}$) or has a user in UIDs as a registered friend ($\mathsf{UIDs} \cap \mathsf{friendIDs}(\sigma', \mathsf{owner}\ \sigma'\ \mathsf{PID}) \neq \emptyset$) or the visibility of PID is public ($\mathsf{visPost}(\sigma', \mathsf{PID}) = \mathsf{public}$). This formalizes the *unless* part of (P_1).

The above concepts apply generally, to any I/O automaton:

Definition 1. *An* attacker model *consists of*

- *a secrecy infrastructure* $(\mathsf{Secret}, \mathsf{isSec}, \mathsf{getSec})$ *and*
- *an observation infrastructure* $(\mathsf{Obs}, \mathsf{isObs}, \mathsf{getObs})$.

A security policy *is specified by*

- *a declassification trigger* T *and*
- *a declassification bound* B.

A BD security property [38] (P) *is specified by an attacker model and a security policy (as above).*

An I/O automaton Aut *satisfies the BD security property* (P) *if, for all traces tr in which* T *never holds for any transition and for all sequences of secrets sl′ such that* $\mathsf{B}(\mathsf{S}^{\mathsf{getSec}}_{\mathsf{isSec}}(tr), sl')$ *holds, there exists a system trace tr′ such that* $\mathsf{S}^{\mathsf{getSec}}_{\mathsf{isSec}}(tr') = sl'$ *and* $\mathsf{O}^{\mathsf{getObs}}_{\mathsf{isObs}}(tr') = \mathsf{O}^{\mathsf{getObs}}_{\mathsf{isObs}}(tr)$.

Intuitively, BD Security states that $\mathsf{O}^{\mathsf{getObs}}_{\mathsf{isObs}}$ *cannot learn anything about* $\mathsf{S}^{\mathsf{getSec}}_{\mathsf{isSec}}$ *beyond* B *unless* T *occurs*. It takes an epistemic logic perspective: Given any actual trace of events tr, an *alternative trace* tr' must exist that offers an equally valid explanation of the observations within the specified uncertainty/declassification bound.

IV. The Distributed System

Next, we describe the specification and implementation of the CoSMeDis extension of CoSMed.

We extend the original system with mechanisms for communicating and sharing data with other nodes located at different sites across the Internet. All nodes have identical behavior, i.e., will be CoSMeDis clones (but their internal states will of course be different due to their different interactions with users and among themselves). The resulting system is similar to existing, federated social networks, such as Diaspora* [8]. A user signs up at one of the nodes in the network and uses it to post and read content and maintain friendship links. Employing the new inter-node communication features, a user can also establish friendship links to users on other nodes and exchange content across nodes. Public posts are available to all users of connected nodes, whereas private posts are only accessible by local and remote friends of the post owner.

A. Communication Infrastructure

We consider networks with an arbitrary number of CoSMeDis nodes. A node is designated by a unique ID, its URL. We implement an asymmetric communication model. Any two nodes with IDs nid_1 and nid_2 can agree on a client-server relationship: The client nid_1 makes a request and the server nid_2 approves it (both actions being triggered by the admin users of the corresponding nodes). After that, nid_2 can share its posts with nid_1. In addition, users of nid_2 are allowed to mark as friends selected users of nid_1. Hence, the admins are responsible for setting up inter-node communication as well as approving local user creation. From a user perspective, the system is like CoSMed, except that users can see posts from other nodes if the owner has granted them access, and can add remote friends by selecting a node and entering a username.

To achieve the above, we extend the system's state with communication infrastructure (the IDs of the registered client and server nodes) and shared data (inter-node friendship and shared posts). We also add new types of actions to support the desired communication: sendServerReq and receiveClientReq, sendPost and receivePost, and sendUpdateRFriend and receiveUpdateRFriend. They come in pairs: There is an action on the receiving side to match that on the sending side. For successful communication, the parameters of these actions (consisting of user, node and post IDs, post content, etc.) must also match, in that what is being received must coincide with what is being sent. Here is the intended workflow and the matching patterns for these actions:

Request server connection. The admin uid_1 of a node nid_1 can issue a server request to another node nid_2, with the intention of establishing a client-server relationship. The corresponding action is $(\mathsf{sendServerReq}, uid_1, nid_2, request)$, where *request* is the content of the request message. When the request reaches nid_2, the action $(\mathsf{receiveClientReq}, nid_1, request)$ is triggered on nid_2, to the effect of recording in nid_2's state that nid_1 wishes to become a client.

Connect client with server. At a later time, the admin uid_2 of nid_2 can inspect and approve the request. This is done through the communication action $(\mathsf{connectClient}, uid_2, nid_1)$, which registers, in nid_2's state, the node nid_1 as a client. The matching action on the nid_1's side is $(\mathsf{connectServer}, nid_2)$, which registers, in nid_1's state, the node nid_2 as a server.

Share posts. After nid_1 and nid_2 have recognized each other as a client-server pair, other communication actions are possible. The admin uid_2 of the server nid_2 can send a local post *pid* at any time to the client nid_1, via $(\mathsf{sendPost}, uid_2, nid_1, pid)$. This action produces the output (pid, pst, uid_2', v), consisting of the post ID *pid*, the content *pst* of the post, the ID uid_2' of the post's owner, and information on the post's visibility, *v*. In this output, *pid* is copied from the action's parameter, whereas all the other components are retrieved from nid_2's state. The matching action on the nid_1 side is $(\mathsf{receivePost}, nid_2, pid, pst, uid_2', v)$. Sending an updated version of a previously shared post is possible—it has the effect of updating the remote version. A flag is stored in the server node's state for each shared post with each client, indicating whether the remote version is up to date.

Assign remote friends. Sharing a local post *pid* between the server nid_2 and the client nid_1 is at the discretion of nid_2's admin uid_2, which would typically send several posts in batch mode. However, the post owner uid_2' retains control on the remote access rights for his friend-only posts. Namely, the remote version of *pid* will only be accessible to users uid_1' of nid_1 which uid_2' designates as remote friends. Remote friend designation is done through the action $(\mathsf{sendUpdateRFriend}, uid_2', nid_1, uid_1', st)$, which sends an update to the friendship-like permission from user uid_2' of nid_2 to user uid_1' of nid_1; the flag *st* indicates if friendship is to be *granted* or *revoked*. The matching action on the nid_1 side is $(\mathsf{receiveUpdateRFriend}, nid_2, uid_2', uid_1', st)$, which updates the indicated permission.

B. Modeling the Distributed System

We will eventually model a network of an arbitrary number of nodes. But to keep the discussion simple, we initially assume only two nodes, represented by two I/O automata Aut_1 and Aut_2. In Section VI-B, we describe the *n*-ary case.

In our case study, Aut_1 and Aut_2 are identical—as CoSMeDis clones. However, this assumption will not be needed in our discussion. We shall use the indexes 1 and 2 to indicate the components of these automata, e.g., State_1, Act_1, State_2, Act_2, etc. An exception will be their transition relations, where we write $\rightarrow$ for both rather than $\rightarrow_1$ or $\rightarrow_2$.

As seen in Section IV-A, communication proceeds by matching certain transitions of the two components: each sending transition with a corresponding receiving transition. We model matching as a relation match between the transitions of Aut_1 and those of Aut_2, taking $\mathsf{match}((\sigma_1, a_1, o_1, \sigma_1'), (\sigma_2, a_2, o_2, \sigma_2'))$ to mean that either (a_1, o_1) is a sending action-output pair and (a_2, o_2) is the corresponding receiving action-output pair or vice versa. Thus, two transitions are matched if their actions are dual to each other and the parameters occurring in the sending action or in its output correspond to the input parameters of the receiving action. For example, the input parameters of a $\mathsf{receivePost}$ action have to match the output of the $\mathsf{sendPost}$ action.

Formally, $\mathsf{match}((\sigma_1, a_1, o_1, \sigma_1'), (\sigma_2, a_2, o_2, \sigma_2'))$ requires that the actions and outputs have one of the forms in Fig. 1's table, where NID_1 and NID_2 are the IDs of Aut_1 and Aut_2; or the symmetric forms, with NID_1 and NID_2 swapped. (It might be unclear why match should also depend on the transitions' states. Indeed, for CoSMeDis's communication, it does not: we instantiate it independently of the states, only considering the actions and outputs. However, state-dependent matching would make sense if the components had a shared part of the state, so we don't forbid it in our emerging framework.)

In our discussion, we distinguished separate (local) component actions from communication actions. We write $\mathsf{isCom}_i(a)$ (for $i \in \{1, 2\}$) to state that *a* is in the latter category for component *i*. This predicate can be derived from the matching predicate: $\mathsf{isCom}_i(a)$ holds whenever there exist trn_1 and trn_2 such that $\mathsf{match}(trn_1, trn_2)$ holds and *a* is the action of trn_i.

We define the distributed system as an I/O automaton, $\mathsf{Aut}_1 \times^{\mathsf{match}} \mathsf{Aut}_2$, representing the communicating product of the components:

- Its set of states is $\mathsf{State}_1 \times \mathsf{State}_2$, with the initial state being the pair of initial states, (σ_1^0, σ_2^0).
- Its set of actions is $\mathsf{Act}_1 + \mathsf{Act}_2 + \mathsf{Act}_1 \times \mathsf{Act}_2$, i.e., a disjoint union of Act_1 (representing separate actions of the first component), Act_2 (for separate actions of the second component), and $\mathsf{Act}_1 \times \mathsf{Act}_2$ (for joint communicating actions). We shall write $(1, a_1)$, $(2, a_2)$, and (a_1, a_2) for actions of the first, second, and third kind, respectively.
- Similarly, its set of outputs is $\mathsf{Out}_1 + \mathsf{Out}_2 + \mathsf{Out}_1 \times \mathsf{Out}_2$, and we use similar notations: $(1, o_1)$, or $(2, o_2)$, or (o_1, o_2).
- Its transition system is defined in Fig. 2. As can be seen, the $\textsc{Sep}_i$ rules allow each component to proceed separately, whereas the $\textsc{Com}$ rule allows matching communication transitions of any two components.

Note that a transition *trn* of $\mathsf{Aut}_1 \times^{\mathsf{match}} \mathsf{Aut}_2$ has one of the following three forms:

(1) $((\sigma_1, \sigma_2), (1, a_1), (1, o_1), (\sigma_1', \sigma_2))$
(2) $((\sigma_1, \sigma_2), (2, a_2), (2, o_2), (\sigma_1, \sigma_2'))$
(3) $((\sigma_1, \sigma_2), (a_1, a_2), (o_1, o_2), (\sigma_1', \sigma_2'))$

In the first case, *trn* is completely determined by the Aut_1-transition $trn_1 = (\sigma_1, a_1, o_1, \sigma_1')$ and by the Aut_2-state σ_2—we write $trn = \mathsf{sep}_1(trn_1, \sigma_2)$, marking that *trn* is given by the separate transition trn_1. Similarly, in the second case we write $trn = \mathsf{sep}_2(\sigma_1, trn_2)$, where $trn_2 = (\sigma_2, a_2, o_2, \sigma_2')$. In the third case, we write $trn = \mathsf{com}(trn_1, trn_2)$, marking that *trn* proceeds as a communication transition. Thus, any transition of $\mathsf{Aut}_1 \times^{\mathsf{match}} \mathsf{Aut}_2$ has either the form $\mathsf{sep}_1(trn_1, \sigma_2)$, or $\mathsf{sep}_2(\sigma_1, trn_2)$, or $\mathsf{com}(trn_1, trn_2)$.

Given an Aut_1-trace tr_1 and an Aut_2-trace tr_2, we define $tr_1 \,||^{\mathsf{match}}\, tr_2$, the *communicating shuffle* of tr_1 and tr_2, to be the set of all $(\mathsf{Aut}_1 \times^{\mathsf{match}} \mathsf{Aut}_2)$-traces obtained from shuffling (interleaving) tr_1 and tr_2—the inductive definition is shown in Fig. 3, where $\cdot$ is concatenation and $[]$ the empty trace.

C. Implementation

CoSMeDis is implemented similarly to its predecessor, CoSMed. It consists of three layers: the kernel (generated

a_1	o_1	a_2	o_2
$(\mathsf{sendServerReq}, uid_1, \mathsf{NID}_2, request)$	$request$	$(\mathsf{receiveClientReq}, \mathsf{NID}_1, request)$	outOK
$(\mathsf{connectClient}, uid_1, \mathsf{NID}_2)$	outOK	$(\mathsf{connectServer}, \mathsf{NID}_1)$	outOK
$(\mathsf{sendPost}, uid_1, \mathsf{NID}_2, pid)$	(pid, pst, uid'_1, v)	$(\mathsf{receivePost}, \mathsf{NID}_1, pid, pst, uid'_1, v)$	outOK
$(\mathsf{sendUpdateRFriend}, uid'_1, \mathsf{NID}_2, uid'_2, st)$	(uid'_1, uid'_2, st)	$(\mathsf{receiveUpdateRFriend}, \mathsf{NID}_1, uid'_1, uid'_2, st)$	outOK

Figure 1: Definition of match for CoSMeDis

$$\textsc{Sep}_1 \frac{\sigma_1 \xrightarrow[o_1]{a_1} \sigma'_1 \quad \neg\, \mathsf{isCom}_1(a_1)}{(\sigma_1, \sigma_2) \xrightarrow[(1,o_1)]{(1,a_1)} (\sigma'_1, \sigma_2)} \qquad \textsc{Sep}_2 \frac{\sigma_2 \xrightarrow[o_2]{a_2} \sigma'_2 \quad \neg\, \mathsf{isCom}_2(a_2)}{(\sigma_1, \sigma_2) \xrightarrow[(2,o_2)]{(2,a_2)} (\sigma_1, \sigma'_2)} \qquad \textsc{Com} \frac{\sigma_1 \xrightarrow[o_1]{a_1} \sigma'_1 \quad \sigma_2 \xrightarrow[o_2]{a_2} \sigma'_2 \quad \mathsf{match}((\sigma_1, a_1, o_1, \sigma'_1), (\sigma_2, a_2, o_2, \sigma'_2))}{(\sigma_1, \sigma_2) \xrightarrow[(o_1,o_2)]{(a_1,a_2)} (\sigma'_1, \sigma'_2)}$$

Figure 2: Transitions of a two-component distributed system

$$\textsc{Sep}_1 \frac{tr \in tr_1 \,||^{\mathsf{match}}\, tr_2 \quad \neg\, \mathsf{isCom}_1(\mathsf{actOf}_1(trn_1))}{(tr \cdot \mathsf{sep}_1(trn_1, \mathsf{endState}(tr_2))) \in (tr_1 \cdot trn_1) \,||^{\mathsf{match}}\, tr_2} \qquad \textsc{Sep}_2 \frac{tr \in tr_1 \,||^{\mathsf{match}}\, tr_2 \quad \neg\, \mathsf{isCom}_2(\mathsf{actOf}_2(trn_2))}{(tr \cdot \mathsf{sep}_2(\mathsf{endState}(tr_1), trn_2)) \in tr_1 \,||^{\mathsf{match}}\, (tr_2 \cdot trn_2)}$$

$$\textsc{Empty} \frac{\cdot}{[] \in [] \,||^{\mathsf{match}}\, []} \qquad \textsc{Com} \frac{tr \in tr_1 \,||^{\mathsf{match}}\, tr_2 \quad \mathsf{match}(trn_1, trn_2)}{(tr \cdot \mathsf{com}(trn_1, trn_2)) \in (tr_1 \cdot trn_1) \,||^{\mathsf{match}}\, (tr_2 \cdot trn_2)}$$

Figure 3: The communicating shuffle operator for CoSMeDis

automatically), the API layer, and the outer layer (the last two implemented manually).

The kernel consists of the I/O automaton extracted from the Isabelle specification to Scala code. In addition to regular data (on users, posts and friendship), the kernel state also contains identity checking data: passwords for users and keys for client and server nodes. Moreover, the kernel actions take passwords and/or keys as parameters—omitted in this paper to enhance readability.

The API layer forwards requests back and forth between the I/O automaton kernel and the outside world. It converts the payload of http(s) requests into elements of Act, which are passed to the automaton; the output retrieved from the automaton is then converted into JSON output, which is delivered as the API response. Special treatment is given to data that cannot be reasonably stored in memory, namely, the optional image files associated to posts. These are stored on the disk, while the kernel state stores the paths to their locations. They are all placed in a single directory, and the names of the files are the (guaranteed to be nonoverlapping) post IDs. When a user requests the read of a post image, the API layer invokes the corresponding reading action from the kernel to retrieve the path to that file—then the file is offered for download.

The outer layer handles the user interface and performs session management. It directs user requests to the API layer and displays the results back to the user or, if necessary, makes remote API requests to other nodes (for communication actions).

The main behavioral differences between CoSMeDis and CoSMed are the following:

- A user action could produce not only local effects, but possibly also a request to a different node. For example, the single action $(\mathsf{sendPost}, uid_2, nid_1, pid)$ triggers a local API call and a remote API call to nid_1.
- Not only users, but other nodes can issue actions, too. For example, $(\mathsf{receivePost}, nid_2, pid, pst, uid'_2, v)$ is an action issued by a remote node nid_2.

Making sure that send and receive actions are correctly matched is achieved by a transactional policy. E.g., when Aut_1's admin issues a sendPost request indicating the target node as Aut_2, the following happens:

- The sendPost action is run locally, producing the new state σ'_1 and the output o_1; but the new state is not yet committed.
- If sendPost was successful, a corresponding receivePost request is made remotely to Aut_2.
- If Aut_2 responds with output outOK, the new state σ'_1 is committed at Aut_1.

CoSMeDis is delivered as a bundle, installable at any location on the web to form a new node. The Scala code for the CoSMeDis I/O automaton is significantly larger than that of CoSMed: 2700 compared to 1650 LOC. As expected, the other parts of the application are also larger: 870 vs. 610 for the API layer and 900 vs. 720 for the outer layer.

Limited testing indicates that CoSMeDis performs well for small numbers of users. For up to 100 users in the system, each with 10 friends, 10 remote friends and 10 posts, the API could serve about 2600 requests/second for reading and sending actions and 1800 requests/second for writing and receiving actions. However, a major obstacle in the way of an efficient version of CoSMeDis is the parallelism bottleneck: To allow multi-threaded calls one currently needs to lock the whole state object (the full I/O automaton state), whereas in database-driven real-world applications one could lock single tables or rows, enabling parallel access. To capture this parallel model,

Isabelle's code generator would require a significant extension.

V. Composing Security for CoSMeDis

Each CoSMeDis node is an extension of CoSMed. Consequently, when trying to prove confidentiality for CoSMeDis, we look into how to extend the CoSMed confidentiality properties to properties of a single node, and then into how to compose node confidentiality to obtain guarantees for the entire system. As we shall see, these two steps are not independent, but we have to proceed in a feedback loop.

A. Security Models for the Components

Our running example, (P_1), limits the amount of information flowing from the content of a given secret: a post PID. In relation to this secret, we can distinguish two types of components: the node where it originates, say, Aut_1, and the other nodes which are possible receivers of the post.

For the originator, it is intuitive that we should be able to prove the same property (P_1), regardless of the fact that now the system is communicating on more channels. Indeed, as far as Aut_1's users are concerned, the notions of secret and observation are the same: the secrets are the updates to PID's content, while the observations are the actions and outputs of a given set of Aut_1 users, say, UIDs_1. The communication actions do not interfere with this local security model: it is irrelevant for the observation power of an Aut_1 user if the post is being sent to another node. Therefore, the proofs done for the original CoSMed work without essential modifications for the communication-updated version—with the same trigger and the same bound. In summary, we can easily (re)prove:

(P_1) A group of users UIDs_1 of Aut_1 can learn nothing about the updates to the content of Aut_1's post PID *beyond* the existence of an update *unless* one of them is the admin or PID's owner, or becomes friends with the owner, or PID is marked as public.

Now, consider a node Aut_2 that may potentially receive the content of the post PID from Aut_1. We can also prove a version of (P_1) for the receiving end with a (possibly overlapping) set of observers UIDs_2, *mutatis mutandis*:

(P_2) A group of users UIDs_2 of Aut_2 can learn nothing about the updates to the content of Aut_1's post PID *beyond* the existence of an update *unless* PID is being shared between Aut_1 and Aut_2 and [one of the users is the admin or becomes a remote friend of PID's owner, or PID is marked as public].

Formally, (P_1) and (P_2) are instances of BD Security, specified by the attacker models and security policies shown in Fig. 4, where NID_1 and NID_2 are the IDs of Aut_1 and Aut_2. The formalization of (P_1) is essentially the one sketched in Section III. In particular, the secrets are the updates to PID's content as produced by updatePost actions, and the trigger refers to one of the users in UIDs_1 being the admin, or PID's owner, or a friend. For (P_2), the secrets are also updates to PID's content, but they are produced differently: by receivePost actions having Aut_1 as sender. Any update of PID's content is received along with the owner's ID *uid* and with any possible update v of the visibility status, which is recorded as the "remote visibility" stored for $(\mathsf{NID}_1, \mathsf{PID})$. The trigger first requires that PID has been shared, which is stored in Aut_2's state as a remote post ID coming from Aut_1; then it makes requirements similar to (P_1)'s trigger, but referring to remote versions friendship and visibility. We write S_1 and O_1 instead of $\mathsf{S}^{\mathsf{isSec}_1}_{\mathsf{getSec}_1}$ and $\mathsf{O}^{\mathsf{isObs}_1}_{\mathsf{getObs}_1}$, and similarly for S_2 and O_2.

B. The Compositionality Challenge

Let us analyze how two properties, such as (P_1) and (P_2), can be composed into a property for $\mathsf{Aut} = \mathsf{Aut}_1 \times^{\mathsf{match}} \mathsf{Aut}_2$.

The compound attacker model should be a form of communication-aware "sum," or "union," of those for (P_1) and (P_2). Since the Aut-traces are obtained by the communicating shuffle of Aut_1- and Aut_2-traces, the observations produced by Aut are themselves shufflings of those of the components. So it is natural to take Obs, the compound observation domain, to be $\mathsf{Obs}_1 + \mathsf{Obs}_2 + \mathsf{Obs}_1 \times \mathsf{Obs}_2$—meaning, as usual, that an element of Obs will have either the form $(1, o_1)$ or $(2, o_2)$ or (o_1, o_2), where $o_i \in \mathsf{Obs}_i$. For an Aut-transition *trn*, we define $\mathsf{isObs}(trn)$ and $\mathsf{getObs}(trn)$ as follows. $\mathsf{isObs}(trn)$ is true iff:

- *trn* has the form $\mathsf{sep}_1(trn_1, \sigma_2)$ and $\mathsf{isObs}_1(trn_1)$ holds, in which case we define $\mathsf{getObs}(trn) = (1, \mathsf{getObs}_1(trn_1))$,
- or *trn* has the form $\mathsf{sep}_2(\sigma_1, trn_2)$ and $\mathsf{isObs}_2(trn_2)$ holds, in which case we define $\mathsf{getObs}(trn) = (2, \mathsf{getObs}_2(trn_2))$,
- or *trn* has the form $\mathsf{com}(trn_1, trn_2)$ and $\mathsf{isObs}_1(trn_1)$ or $\mathsf{isObs}_2(trn_2)$ hold, in which case we define $\mathsf{getObs}(trn) = (\mathsf{getObs}_1(trn_1), \mathsf{getObs}_2(trn_2))$.

Similar constructions are performed for secrets. The compound domain, Sec, is taken to be $\mathsf{Sec}_1 + \mathsf{Sec}_2 + \mathsf{Sec}_1 \times \mathsf{Sec}_2$, and isSec and getSec are defined correspondingly. This concludes the attacker model definition—we again write S and O instead of $\mathsf{S}^{\mathsf{getSec}}_{\mathsf{isSec}}$ and $\mathsf{O}^{\mathsf{getObs}}_{\mathsf{isObs}}$.

Before moving to the definition of the compound security policy, let us first contemplate some existing or missing symmetries in the matching of observations and secrets of the two components. The notion of matching transitions induces a notion of matching observations. In fact, the latter can be regarded as a stand-alone predicate $\mathsf{matchO} : \mathsf{Obs}_1 \times \mathsf{Obs}_2 \to \mathsf{Bool}$. For our case study, it is defined in the same way as match—since match does not actually depend on the states, but only on the action-output pairs, which also make up the observations. The corresponding communicating shuffle for observations, $||^{\mathsf{matchO}} : \mathsf{Obs}_1^* \times \mathsf{Obs}_2^* \to \mathsf{Pow}(\mathsf{Obs}^*)$ (where Pow is the powerset operator), is obtained from matchO similarly to how $||^{\mathsf{match}}$ is obtained from match.

When trying to define a similar notion of matching for secrets, $\mathsf{matchS} : \mathsf{Sec}_1 \times \mathsf{Sec}_2 \to \mathsf{Bool}$, we encounter an anomaly: There are secrets on the receiving end (where (P_2) holds), produced by receivePost actions, that are not matched at the sending end (where (P_1) holds). Indeed, (P_1)'s only secret-producing actions are updatePost actions, which are non-communicating. The anomaly is easy to repair by amending (P_1)'s attacker model to factor in sendPost actions as well:

Amendment 1. (P_1)'s secrecy infrastructure (from Fig. 4) is extended as highlighted below:

- $\mathsf{Sec}_1 = \mathsf{upd}\ \mathsf{Post} + \mathsf{snd}\ \mathsf{Post}$. We use this notation to mean that secrets are now post contents annotated with the labels upd or snd, in order to distinguish between posts produced

	(P_1)	(P_2)
Attacker Model	$\mathsf{Sec}_1 = \mathsf{Post}$	$\mathsf{Sec}_2 = \mathsf{Post}$
	$\mathsf{isSec}_1(\sigma_1, a_1, o_1, \sigma_1')$ iff $o_1 = \mathsf{outOK} \wedge$ $\exists uid, pst.\ a_1 = (\mathsf{updatePost}, uid, \mathsf{PID}, pst)$	$\mathsf{isSec}_2(\sigma_2, a_2, o_2, \sigma_2')$ iff $o_2 = \mathsf{outOK} \wedge$ $\exists pst, uid, v.\ a_2 = (\mathsf{receivePost}, \mathsf{NID}_1, \mathsf{PID}, pst, uid, v)$
	$\mathsf{getSec}_1(\sigma_1, a_1, o_1, \sigma_1') = pst$	$\mathsf{getSec}_2(\sigma_2, a_2, o_2, \sigma_2') = pst$
	$\mathsf{Obs}_1 = \mathsf{Act} \times \mathsf{Out}$	$\mathsf{Obs}_2 = \mathsf{Act} \times \mathsf{Out}$
	$\mathsf{isObs}_1(\sigma_1, a_1, o_1, \sigma_1')$ iff $\mathsf{userOf}(a_1) \in \mathsf{UIDs}_1$	$\mathsf{isObs}_2(\sigma_2, a_2, o_2, \sigma_2')$ iff $\mathsf{userOf}(a_2) \in \mathsf{UIDs}_2$
	$\mathsf{getObs}_1(\sigma_1, a_1, o_1, \sigma_1') = (a_1, o_1)$	$\mathsf{getObs}_2(\sigma_2, a_2, o_2, \sigma_2') = (a_2, o_2)$
Security Policy	$\mathsf{B}_1(sl, sl')$ iff $sl = [] \rightarrow sl' = []$	$\mathsf{B}_2(sl, sl')$ iff $sl = [] \rightarrow sl' = []$
	$\mathsf{T}_1\ (\sigma_1, a_1, o_1, \sigma_1')$ iff $\mathsf{admin}(\sigma') \in \mathsf{UIDs}_1 \vee \mathsf{owner}(\sigma_1', \mathsf{PID}) \in \mathsf{UIDs}_1 \vee$ $\mathsf{UIDs}_1 \cap \mathsf{friendIDs}(\sigma_1', \mathsf{owner}(\sigma_1', \mathsf{PID})) \neq \emptyset \vee$ $\mathsf{vis}(\sigma_1', \mathsf{PID}) = \mathsf{public}$	$\mathsf{T}_2\ (\sigma_2, a_2, o_2, \sigma_2')$ iff $\mathsf{PID} \in \mathsf{remotePostIDs}(\sigma_2', \mathsf{NID}_1) \wedge$ $(\mathsf{admin}(\sigma') \in \mathsf{UIDs}_2 \vee$ $\mathsf{UIDs}_2 \cap \mathsf{remoteFriendIDs}(\sigma_2', \mathsf{NID}_1,$ $\mathsf{remoteOwner}(\sigma_2', \mathsf{NID}_1, \mathsf{PID})) \neq \emptyset \vee$ $\mathsf{remoteVis}(\sigma_2', \mathsf{NID}_1, \mathsf{PID}) = \mathsf{public})$

Figure 4: Attacker model and security policy for (P_1) and (P_2)

by an update action, (upd, pst), and posts produced by a sending action, (snd, pst).

- $\mathsf{isSec}_1\ (\sigma, a, o, \sigma')$ iff
  $o = \mathsf{outOK} \wedge (\exists uid, pst.\ a = (\mathsf{updatePost}, uid, \mathsf{PID}, pst)) \vee$
  $\exists uid, nid, pst, uid', v.\ \begin{array}{l} o = (\mathsf{PID}, pst, uid', v) \wedge \\ a = (\mathsf{sendPost}, uid, nid, \mathsf{PID}) \end{array}$
- getSec_1 now extracts the content pst from both update actions and outputs for send actions:
  $\mathsf{getSec}_1(\_, (\mathsf{updatePost}, \_, \_, pst), \_, \_) = (\mathsf{upd}, pst)$
  $\mathsf{getSec}_1(\_, (\mathsf{sendPost}, \_, \_), (\_, pst, \_, \_), \_) = (\mathsf{snd}, pst)$

While restoring symmetry w.r.t. communication, this amendment creates disturbance in (P_1)'s security policy, namely, in its bound: It is no longer the case that the observers can learn nothing about a sequence of produced secrets beyond possible emptiness. Now, it is also known that the contents of updatePost and sendPost are correlated: What is being sent coincides with what was last updated. For example, (unless the trigger fires) from the sequence $\langle(\mathsf{upd}, pst_1), (\mathsf{upd}, pst_2), (\mathsf{snd}, pst_3), (\mathsf{upd}, pst_4), (\mathsf{snd}, pst_5)\rangle$, the observers would not know the values pst_i, but would know that $pst_2 = pst_3$ and $pst_4 = pst_5$. This public knowledge must be included in the bound in order for the amended property to hold.

Amendment 2. (P_1)'s bound is extended as follows:

$$\mathsf{B}_1(sl, sl')\ \text{iff}\ (sl = [] \rightarrow sl' = []) \wedge \mathsf{corr}(sl')$$

where $\mathsf{corr}(sl')$ states that sl' has its upd-labeled and snd-labeled post contents correlated in the above sense.

With these two amendments, (P_1) is still true—the proof is amended by keeping the correlation property as an invariant, but otherwise stays the same.

Now we can define a matching predicate for secrets: we take $\mathsf{matchS}((\mathsf{upd}, pst_1), pst_2)$ to be false, and $\mathsf{matchS}((\mathsf{snd}, pst_1), pst_2)$ to be true iff $pst_1 = pst_2$ (i.e., what is being sent coincides with what is being received). This gives a corresponding notion of communicating shuffle for secrets, $||^{\mathsf{matchS}}$. We shall simply write $||$ for any of the three shuffling operations—for transitions, observations or secrets.

In summary, we have identified a *communication infrastructure*, consisting of the predicates $\mathsf{match} : \mathsf{Trans}_1 \times \mathsf{Trans}_2 \rightarrow \mathsf{Bool}$, $\mathsf{matchO} : \mathsf{Obs}_1 \times \mathsf{Obs}_2 \rightarrow \mathsf{Bool}$ and $\mathsf{matchS} : \mathsf{Sec}_1 \times \mathsf{Sec}_2 \rightarrow \mathsf{Bool}$. As one would expect, this infrastructure is *compatible* with the secrets and the observations, in that, whenever $\mathsf{match}(trn_1, trn_2)$ holds, we have that $\mathsf{isSec}_1(trn_1)$ and $\mathsf{isSec}_2(trn_2)$ are equivalent, and if they hold then $\mathsf{matchS}(\mathsf{getSec}_1(trn_1), \mathsf{getSec}_2(trn_2))$ also holds; and similarly for observations.

An important consequence of having a compatible communication infrastructure is that $||$ commutes with the secret- and observation-producing functions:

Corollary 1. *$tr \in tr_1 \,||\, tr_2$ implies $\mathsf{S}(tr) \in \mathsf{S}_1(tr_1) \,||\, \mathsf{S}_2(tr_2)$ and $\mathsf{O}(tr) \in \mathsf{O}_1(tr_1) \,||\, \mathsf{O}_2(tr_2)$.*

The notion of shuffling secrets in a communication-aware fashion is essential for defining a compound bound B. We take $\mathsf{B}(sl, sl')$ to mean that, whenever $sl \in sl_1 \,||\, sl_2$ and $sl' \in sl_1' \,||\, sl_2'$ for some sl_1, sl_2, sl_1', sl_2', we have that $\mathsf{B}_1(sl_1, sl_1')$ and $\mathsf{B}_2(sl_2, sl_2')$ hold. This is the strongest bound we can hope for the composite. It performs a "shuffling intersection" of B_1 and B_2, stating that no matter how we decompose the secrets as a communicating shuffle of component secrets, both component bounds hold. It specifies an intersection of the amounts of uncertainty about the secret enforced by the components, i.e., a union of the amounts of information that is being declassified: if each component declassifies one particular aspect of the overall secret, then the composite declassifies *both*.

Finally, the natural composed trigger T is "T_1 or T_2": If the trigger of either component is fired, then the secrets are legitimately accessible to observers.

So far, so good: We have defined an attacker model and a security policy for the compound system—i.e., an instance of BD Security, which we denote by $(\mathsf{P}_1) \,||\, (\mathsf{P}_2)$. But can we prove that it indeed holds for $\mathsf{Aut} = \mathsf{Aut}_1 \times^{\mathsf{match}} \mathsf{Aut}_2$, assuming (P_1) and (P_2) hold for the components?

The challenge of proving $(\mathsf{P}_1) \,||\, (\mathsf{P}_2)$ is depicted in Fig. 5. Let tr be a trace of Aut. Suppose it produces the secrets sl and the observations ol, and let sl' be an alternative sequence of secrets such that $\mathsf{B}(sl, sl')$ holds. We know that tr is given by the communicating shuffle of some traces tr_1 of Aut_1 and tr_2 of Aut_2, i.e., $tr \in tr_1 \,||\, tr_2$. Say tr_1 and tr_2 produce the secrets sl_1 and sl_2 and the observations ol_1 and ol_2. In order to make a connection to the bounds of the components, and

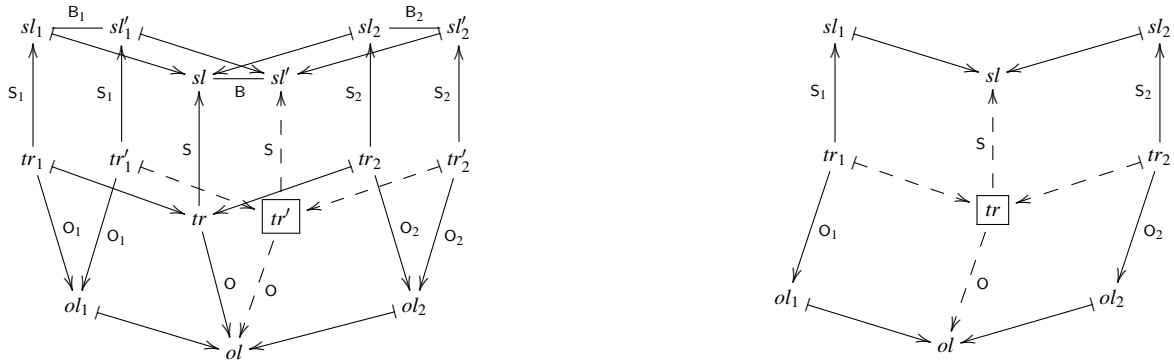

Figure 5: The problem of finding an alternative composed trace: in context (left) and in isolation (right)

therefore take advantage of (P_1) and (P_2), we invoke Corollary 1, which gives us $sl \in sl_1 \mid\mid sl_2$ and $ol \in ol_1 \mid\mid ol_2$. Then, by the definition of B, we have $\mathsf{B}_1(sl_1, sl'_1)$ and $\mathsf{B}_2(sl_2, sl'_2)$. By the security properties of the components, we obtain the alternative component traces tr'_1 and tr'_2 such that each tr'_i yields the secrets sl'_i and is observationally equivalent to tr_i.

So the original problem is reduced to the following: Can we use tr'_1 and tr'_2 to construct a suitable alternative trace for Aut, i.e., a trace tr' that produces sl' and is observationally equivalent to tr? The right side of Fig. 5 shows the problem in isolation, where we rename tr' to tr, tr'_i to tr_i and sl'_i to sl_i:

Problem. *We are given sl, ol, sl_i, ol_i and tr_i such that $\mathsf{S}_i(tr_i) = sl_i$, $\mathsf{O}_i(tr_i) = ol_i$, $sl \in sl_1 \mid\mid sl_2$ and $ol \in ol_1 \mid\mid ol_2$, and need to find tr such that $\mathsf{S}(tr) = sl$ and $\mathsf{O}(tr) = ol$.*

Thus, we know how to shuffle observations and secrets and are required to shuffle entire transitions in a compatible way. To this end, we clearly need the observation and secret matching to provide enough information for transition matching. For example, assume that each tr_i consists of a single transition, trn_i, with a communicating action. The only way to shuffle these into a solution tr is if they actually match, i.e., $\mathsf{match}(trn_1, trn_2)$ holds. And for proving this, we know that the observations and secrets of these transitions can be shuffled, hence they do match. Thus, what we would need is that the communication infrastructure is *strong* enough so that the matching infrastructure in the attacker model contains enough information for matching transitions: Whenever the actions of trn_1 and trn_2 are communicating, for $\mathsf{match}(trn_1, trn_2)$ to hold it is sufficient that (1) $\mathsf{matchO}(\mathsf{getObs}_1(trn_1), \mathsf{getObs}_2(trn_2))$ holds conditionally on $\mathsf{isObs}_1(trn_1)$ and $\mathsf{isObs}_2(trn_2)$ and (2) $\mathsf{matchS}(\mathsf{getSec}_1(trn_1), \mathsf{getSec}_2(trn_2))$ holds conditionally on $\mathsf{isSec}_1(trn_1)$ and $\mathsf{isSec}_2(trn_2)$.

Intuitively, this property requires that the observations and secrets of communication transitions fully capture their matching behavior. Unfortunately, the communication infrastructure from (P_1) and (P_2) in our case study is not strong enough. Consider a sending and a receiving action for *different* posts (other than the secret post PID), not involving any of the designated observer users (UIDs_1 or UIDs_2). These actions are neither observable nor secret. Hence, the preconditions of the above property are trivially satisfied, but still the actions fail to match, because they refer to different posts. In order to remedy this, we must strengthen the attackers. For our case study, a reasonable strategy would be to extend the observation power to communicating actions and their outputs, *provided the secret is not compromised*, as follows:

Amendment 3. (P_1)'s observations are extended as follows:

- $\mathsf{isObs}_1(\sigma_1, a_1, o_1, \sigma'_1)$ iff $\mathsf{userOf}(a_1) \in \mathsf{UIDs}_1 \vee \mathsf{isCom}_1(a_1)$
- $\mathsf{getObs}_1\ (\sigma_1, a_1, o_1, \sigma'_1) = (\mathsf{purgeA}_{\mathsf{PID}}(a_1), \mathsf{purgeO}_{\mathsf{PID}}(o_1))$

where $\mathsf{purgeA}_{\mathsf{PID}} : \mathsf{Act} \to \mathsf{Act}$ and $\mathsf{purgeO}_{\mathsf{PID}} : \mathsf{Out} \to \mathsf{Out}$ purge away from *communicating* actions and their outputs the content of PID's post (which constitutes the secret). The observations of (P_2) are extended analogously.

The parameters of actions and outputs that do not pertain to communication or do not manipulate PID's content are not affected by purging—this is the case for sendServerReq and receiveClientReq. By contrast, PID's content *pst* is replaced by the non-informative $\perp$ from everywhere it appears, e.g., $\mathsf{purgeA}_{\mathsf{PID}}(\mathsf{receivePost}, \mathsf{NID}_1, \mathsf{PID}, pst, uid_1, v) = (\mathsf{receivePost}, \mathsf{NID}_1, \mathsf{PID}, \perp, uid_1, v)$ and $\mathsf{purgeO}_{\mathsf{PID}}((\mathsf{PID}, pst, uid_1, v)) = (\mathsf{PID}, \perp, uid_1, v)$.

Since it was done in a secrecy-sensitive way, this increase in observation power keeps the properties (P_1) and (P_2) true. At the same time, it makes the communication infrastructure strong, as desired. The reason is that, for each pair of communication transitions trn_1 and trn_2:

- If they do not produce secrets, then purging does not affect their observations, so $\mathsf{match}(trn_1, trn_2)$ is equivalent to $\mathsf{matchO}(\mathsf{getObs}_1(trn_1), \mathsf{getObs}_2(trn_2))$.
- If they produce secrets, then $\mathsf{matchO}(\mathsf{getObs}_1(trn_1), \mathsf{getObs}_2(trn_2))$ caters for part of the condition required for $\mathsf{match}(trn_1, trn_2)$; the other part is ensured by $\mathsf{matchS}(\mathsf{getSec}_1(trn_1), \mathsf{getSec}_2(trn_2))$. For example, say the action of trn_1 is $(\mathsf{sendPost}, uid_1, \mathsf{NID}_2, \mathsf{PID})$ and its output is $(\mathsf{PID}, pst_1, uid_1, v_1)$, and the action of trn_2 is $(\mathsf{receivePost}, \mathsf{NID}_2, \mathsf{PID}, pst_2, uid_2, v_2)$. Then matching the observations yields $(\perp, uid_1, v_1) = (\perp, uid_2, v_2)$ and matching the secrets yields $pst_1 = pst_2$; together, they yield $(pst_1, uid_1, v_1) = (pst_2, uid_2, v_2)$, as required for $\mathsf{match}(trn_1, trn_2)$.

Note that Amendment 3 has made *all* communication actions observable (to various degrees): $\mathsf{isCom}_1(\mathsf{actOf}_1(trn_1))$ implies $\mathsf{isObs}_1(trn_1)$, and similarly for the second component.

This means *observable network traffic*—the common assumption of a network attacker with Dolev-Yao capabilities.

In general, having strong communication infrastructure and observable network traffic still does not entirely solve our problem. Part of it concerns not communication alone, but also the order of the individual component transitions. Consider the traces $tr_1 = \langle trn_1, trn_1' \rangle$ and $tr_2 = \langle trn_2', trn_2 \rangle$ where trn_1 and trn_2 match and produce local observations o_1 and o_2, respectively, and trn_1' and trn_2' are local transitions that produce local secrets s_1 and s_2, respectively. Then $sl = \langle (1, s_1), (2, s_2) \rangle$ and $ol = \langle (o_1, o_2) \rangle$ are valid shuffles of these secrets and observations. However, in tr_1, the secret comes *after* the (communicating) observation, while in tr_2 it comes *before*. Hence, it is not possible to shuffle tr_1 and tr_2 to a trace that produces sl (since the only possibility is $sl' = \langle (2, s_2), (1, s_1) \rangle$).

This difficulty resides at the heart of our security model: BD Security prescribes a relation, or rather a lack of (co)relation, between observations and secrets produced by a trace, but does not, in general, constrain the time ordering between the production of these observations and secrets. This loose coupling is useful for the local verification of individual systems: not having to worry about the time ordering between observations and secrets allows for flexibility in the proof strategy. However, when composing systems, it leads to the problem that, in general, the bound of the compound system cannot "foresee" which time orderings that arise from freely mixing the component secrets and observations are actually possible, potentially causing compositionality to fail.

Our case study suggests a way out of this conundrum: The above cannot occur because the second component never produces secrets by individual transitions, but only via communication: $\mathsf{isSec}_2(trn_2)$ always implies $\mathsf{isCom}_2(\mathsf{actOf}_2(trn_2))$.

Let us analyze this last property. In a system composed of several nodes, such as CoSMeDis, it is natural to think of a *source* of a secret as a node that produces the secret not by communication with other nodes, but by communication with the outside world. Note that, for simplicity, (P_1) considers the confidentiality of only *one* arbitrary but fixed post PID of Aut_1. (Appendix E shows an extension to *multiple* posts in arbitrary nodes across the network.) Hence, users can "upload" secrets in Aut_1 via updatePost actions, but the only contact of Aut_2 with this secret is via receivePost actions in pair with sendPost actions by Aut_1. In this context, Aut_2 is never the source of the secret, but can only receive it (as well as, possibly, send it back to the issuer or make it available to its users). Thus, we have *secret polarization*, with Aut_1 being the issuing pole.

VI. Abstract Framework for Composing Security

We distill the previous discussion into a framework for composing security in arbitrary communicating I/O automata.

A. Compositionality Theorem for Two Components

Let Aut_1 and Aut_2 be two I/O automata and let $\mathsf{match} : \mathsf{Trans}_1 \times \mathsf{Trans}_2 \to \mathsf{Bool}$ be a predicate for matching transitions. We let Aut be the match-product of Aut_1 and Aut_2, written $\mathsf{Aut}_1 \times^{\mathsf{match}} \mathsf{Aut}_2$, defined in Section IV-B. We use the notations and terminology from that section, e.g., isCom_i.

Furthermore, we fix the security property (P_1) for Aut_1, with its attacker model consisting of the secrecy infrastructure $(\mathsf{Secret}_1, \mathsf{isSec}_1, \mathsf{getSec}_1)$ and observation infrastructure $(\mathsf{Obs}_1, \mathsf{isObs}_1, \mathsf{getObs}_1)$ and its security policy consisting of the trigger T_1 and the bound B_1. We also fix the security policy (P_2) for Aut_2, with similar notations.

Finally, we fix two matching predicates for observations, $\mathsf{matchO} : \mathsf{Obs}_1 \times \mathsf{Obs}_2 \to \mathsf{Bool}$, and secrets, $\mathsf{match} : \mathsf{Sec}_1 \times \mathsf{Sec}_2 \to \mathsf{Bool}$. We call the triple $(\mathsf{match}, \mathsf{matchO}, \mathsf{matchS})$ *the communication infrastructure*.

In summary, the parameters of our compositionality theorems are two I/O automata, one security property for each, and a communication infrastructure. We now define the assumptions of the theorem, as discovered in Section V-B.

Definition 2. *(1) The communication infrastructure is called* compatible *if, for all trn_1 and trn_2,* $\mathsf{match}(trn_1, trn_2)$ *implies:*

- $\mathsf{isSec}_1(trn_1)$ *holds iff* $\mathsf{isSec}_2(trn_2)$ *holds, and in this case* $\mathsf{matchS}(\mathsf{getSec}_1(trn_1), \mathsf{getSec}_2(trn_2))$ *holds*
- $\mathsf{isObs}_1(trn_1)$ *holds iff* $\mathsf{isObs}_2(trn_2)$ *holds, and in this case* $\mathsf{matchO}(\mathsf{getObs}_1(trn_1), \mathsf{getObs}_2(trn_2))$ *holds*

(2) The communication infrastructure is called strong *if, for all trn_1 and trn_2, assuming*

- $\mathsf{isCom}_1(\mathsf{actOf}_1(trn_1)) \wedge \mathsf{isCom}_2(\mathsf{actOf}_2(trn_2))$
- $\mathsf{isObs}_1(trn_1) \wedge \mathsf{isObs}_2(trn_2) \to \mathsf{matchO}(\mathsf{getObs}_1(trn_1), \mathsf{getObs}_2(trn_2))$
- $\mathsf{isSec}_1(trn_1) \wedge \mathsf{isSec}_2(trn_2) \to \mathsf{matchS}(\mathsf{getSec}_1(trn_1), \mathsf{getSec}_2(trn_2))$

then $\mathsf{match}(trn_1, trn_2)$ *holds.*

(3) The attacker models of (P_1) *and* (P_2) *are said to have* observable network traffic *if, for all trn_1 and trn_2,* $\mathsf{isCom}_1(\mathsf{actOf}_1(trn_1))$ *implies* $\mathsf{isObs}_1(trn_1)$ *and* $\mathsf{isCom}_2(\mathsf{actOf}_2(trn_2))$ *implies* $\mathsf{isObs}_2(trn_2)$.

(4) The attacker models of (P_1) *and* (P_2) *are said to be* secret-polarized *if, for all trn_2,* $\mathsf{isSec}_2(trn_2)$ *implies* $\mathsf{isCom}_2(\mathsf{actOf}_2(trn_2))$.

The composed security property, $(\mathsf{P}) = (\mathsf{P}_1) \,||\, (\mathsf{P}_2)$, is defined as follows. Its attacker model $((\mathsf{Secret}, \mathsf{isSec}, \mathsf{getSec}), (\mathsf{Obs}, \mathsf{isObs}, \mathsf{getObs}))$ is defined as in Section V-B, e.g., $\mathsf{Obs} = \mathsf{Obs}_1 + \mathsf{Obs}_2 + \mathsf{Obs}_1 \times \mathsf{Obs}_2$. Moreover, its security policy (B, T) is defined as indicated in that section, namely, in more formal notation (where $||$ is the shuffle operator for secrets, defined using matchS):

$$\mathsf{B}(sl, sl') \text{ iff } \forall sl_1, sl_2, sl_1', sl_2'.\ sl \in sl_1 \,||\, sl_2 \wedge sl' \in sl_1' \,||\, sl_2' \to \mathsf{B}_1(sl_1, sl_1') \wedge \mathsf{B}_2(sl_2, sl_2')$$

$$\mathsf{T}(trn) \text{ iff } (\exists trn_1, \sigma_2.\ trn = \mathsf{sep}_1(trn_1, \sigma_2) \wedge \mathsf{T}_1(trn_1)) \vee (\exists \sigma_1, trn_2.\ trn = \mathsf{sep}_2(\sigma_1, trn_2) \wedge \mathsf{T}_2(trn_2)) \vee (\exists trn_1, trn_2.\ trn = \mathsf{com}(trn_1, trn_2) \wedge (\mathsf{T}_1(trn_1) \vee \mathsf{T}_2(trn_2)))$$

Theorem 1. *Assume*

- *the communication infrastructure is compatible and strong*
- *the attacker models of (P_1) and (P_2) have observable network traffic and are secret-polarized.*

If (P_1) *holds for* Aut_1 *and* (P_2) *holds for* Aut_2*, then* $(\mathsf{P}_1) \,||\, (\mathsf{P}_2)$ *holds for* $\mathsf{Aut}_1 \times^{\mathsf{match}} \mathsf{Aut}_2$.

Let us discuss the requirements of this theorem:

Compatible Communication Infrastructure, asking that not only the component transitions, but also their secrets and observations are composable. This is essential for being able to formulate, let alone prove, the composition of security guarantees—so this is a prerequisite to the very question about compositionality. As this paper's examples illustrate, producing a compatible communication infrastructure comes naturally from inspecting the interaction between communication on the one hand and secrets and observations on the other.

Strong Communication Infrastructure and Observable Network Traffic, asking that communication between the components be substantially exposed. This requirement is clearly an artifact for achieving compositionality. It may be argued that it is counter-intuitive to "allow" the attacker such power. However, the requirement needs to be regarded from the opposite angle: It is not about weakening the system by offering power to the attacker, but about showing that, even if the attacker could observe most of the communication, he would still not learn the secrets. In our case study, we achieved communication strength by letting the attacker observe everything in a communication except for sensitive information.

Secret Polarization, asking that only one of the components can issue secrets. For multi-user systems, this means that, once we agree on what the secrets are, only users of one component can upload secrets. Note that this does not prevent us from considering another notion of secret, where the other component is the issuer. For example, in our case study the secrets are the post contents for a post ID, which can belong to either component—the requirement only prevents us from considering two sources at the same time, e.g., in order to speak of the concatenation of secrets from two sources. The requirement seems to be fulfilled by a specific category of secrets, namely, those *produced and stored locally, on a single node* (and possibly communicated to other nodes). However, it is easy to imagine situations when it breaks—if we allowed users of different nodes to upload versions of the same (shared) post or Facebook-like photo album, or to jointly edit documents asynchronously. Consequently, secret polarization is *the* major limitation of our result. One workaround is to communicate each modification of a secret on a different node back to the source immediately, which is then responsible for merging and propagating such modifications. This is similar to a single-master (or master-slave) model in database replication [60], with a designated master node for each secret. In contrast, a multi-master model, where different nodes perform modifications of a secret concurrently and merge them asynchronously, is not supported by our framework. In Appendix E, we discuss another workaround for combining multiple sources of secrets *after* composition, provided these sources are independent of each other in a certain sense—this applies to the secret sources in CoSMeDis, but does not address the limitation in general.

The main strength of our result is its **policy agnosticism**: While the theorem requirements restrict the component attacker models, they say nothing about the security policies, i.e., their bounds and triggers. Hence, the theorem composes *any given security policies, no questions asked.* This "quantitative" flavor makes our theorem applicable in a variety of contexts, to seamlessly combine arbitrarily complex policies—as we illustrate in Section VII. These include *declassification during the process of inter-component communication*. Indeed, as our examples abundantly illustrate, communication transitions can influence both the attacker models and the security policies.

However, policy agnosticism has an inconvenience: Its general-purpose property composition may not be, in concrete cases, the most natural desired property for the compound system. For our running example, such a natural property is:

(P') A coalition consisting of two groups of users, UIDs_1 of Aut_1 and UIDs_2 of Aut_2, can learn nothing about the updates to the content the Aut_1's post PID *beyond* the existence of an update *unless* one of the following holds:

1) one of UIDs_1 is the admin or PID's owner, or becomes friends with the owner, or PID is marked as public
2) PID is being shared between Aut_1 and Aut_2 and [one of UIDs_2 is the admin or becomes a remote friend of PID's owner, or PID is marked as public]

This reads almost like $(\mathsf{P}_1) \,||\, (\mathsf{P}_2)$. In particular, the trigger T is clearly that formalized by $(\mathsf{P}_1) \,||\, (\mathsf{P}_2)$: the disjunction of the component triggers. However, (P')'s bound, phrased as "beyond the existence of an update," is not verbatim captured by $(\mathsf{P}_1) \,||\, (\mathsf{P}_2)$'s bound B.

Fortunately, we can easily derive (P') from $(\mathsf{P}_1) \,||\, (\mathsf{P}_2)$ using a general-purpose theorem for transfer between two security policies. Let Aut be an I/O automaton and (P) and (P') two security properties operating on it. (P) is said to have a *stronger security model* than (P') if there are two partial functions $f : \mathsf{Sec} \rightharpoonup \mathsf{Sec}'$ and $g : \mathsf{Obs} \rightharpoonup \mathsf{Obs}'$ from the secrets and observations of (P) to those of (P') that preserve the secrecy and observation infrastructures, the bounds and the triggers. (Details are given in Appendix A.)

Theorem 2. *Assume* (P) *has a stronger security model than* (P'). *If* (P) *holds for* Aut*, then so does* (P').

B. The N-ary Case

We aim to establish confidentiality for the entire distributed system, not just for two components. For our case study, we want to prove the following for every post PID belonging to a component Aut_i in a network of n components $\mathsf{Aut}_1, \ldots, \mathsf{Aut}_n$:

(P") A coalition of n groups of users, UIDs_k for each Aut_k, can learn nothing about the updates to PID's content *beyond* the existence of an update *unless* one of the following holds:

1) one of UIDs_i is the admin, or is PID's owner, or becomes friends with the owner, or PID is marked as public
2) the post is being shared by Aut_i with some Aut_k for $k \neq i$ and [one of UIDs_k is the admin or becomes a remote friend of PID's owner, or PID is marked as public]

To this end, we generalize the communicating product automaton construction from 2 to n mutually communicating components Aut_k. We fix, for each k, k' with $k \neq k'$, a matching predicate $\mathsf{match}_{k,k'} : \mathsf{Trans}_k \times \mathsf{Trans}_{k'} \to \mathsf{Bool}$ (between the transitions of Aut_k and $\mathsf{Aut}_{k'}$). We write match for the family $(\mathsf{match}_{k,k'})_{k,k'}$ and $\mathsf{isCom}_{k,k'} : \mathsf{Act}_k \to \mathsf{Bool}$ for the corresponding notion of communication action (belonging to Aut_k, and pertaining to communication with $\mathsf{Aut}_{k'}$). We assume that communication is *pairwise dedicated*, in that the predicates $\mathsf{isCom}_{k,k'}$ and $\mathsf{isCom}_{k,k''}$ are disjoint for $k' \neq k''$.

$$\textsc{Sep}_i\ \frac{\sigma_i \xrightarrow[o_i]{a_i} \sigma_i' \quad \forall j.\, \neg\, \mathsf{isCom}_{i,j}(a_i)}{(\sigma_k)_k \xrightarrow[(i,o_i)]{(i,a_i)} (\sigma_k)_k[i := \sigma_i']} \qquad \textsc{Com}\ \frac{\sigma_i \xrightarrow[o_i]{a_i} \sigma_i' \quad \sigma_j \xrightarrow[o_j]{a_j} \sigma_j' \quad i \neq j \quad \mathsf{match}_{i,j}((\sigma_i, a_i, o_i, \sigma_i'), (\sigma_j, a_j, o_j, \sigma_j'))}{(\sigma_k)_k \xrightarrow[((i,o_i),(j,o_j))]{((i,a_i),(j,a_j))} (\sigma_k)_k[i := \sigma_i', j := \sigma_j']}$$

Figure 6: Transitions of an n-component distributed system

The product is denoted by $\prod^{\mathsf{match}}_{k\in\{1,\ldots,n\}}\mathsf{Aut}_k$. Its states are families $(\sigma_k)_k \in \prod_{k\in\{1,\ldots,n\}} \mathsf{State}_k$. The transition relation is shown in Fig. 6, where, to avoid ambiguity, we use labeling to indicate the components not only for separate actions or outputs, but also for communicating actions or outputs, e.g., $((i,a_i),(j,a_j))$. We write $(\sigma_k)_k[i := \sigma_i']$ for the family of states that is the same as $(\sigma_k)_k$, except for the index i where it is updated from σ_i to σ_i'; and similarly for $(\sigma_k)_k[i := \sigma_i', j := \sigma_j']$.

Given BD Security properties (P_k) for Aut_k, their composition $||_{k\in\{1,\ldots,n\}}(\mathsf{P}_k)$ is defined as an immediate generalization of the binary case. E.g., the observation domain is

$$\mathsf{Obs} \;=\; \sum_{k\in\{1,\ldots,n\}} \mathsf{Obs}_k \;+\; \sum_{k,k'\in\{1,\ldots,n\}} \mathsf{Obs}_k \times \mathsf{Obs}_{k'}$$

so that it contains either separate observations (k, o_k) or joint observations $(k, o_k, k', o_{k'})$. Assuming that each pair of attacker models have compatible communication infrastructure consisting of $\mathsf{match}_{k,k'}$, $\mathsf{matchS}_{k,k'}$, and $\mathsf{matchO}_{k,k'}$, n-ary shuffle operators are defined:

- for traces, $||\,:\prod_{k\in\{1,\ldots,n\}} \mathsf{Trace}_k \to \mathsf{Pow}(\mathsf{Trace})$
- for secrets, $||\,:\prod_{k\in\{1,\ldots,n\}} \mathsf{Sec}_k^* \to \mathsf{Pow}(\mathsf{Sec}^*)$
- for observations, $||\,:\prod_{k\in\{1,\ldots,n\}} \mathsf{Obs}_k^* \to \mathsf{Pow}(\mathsf{Obs}^*)$

Similarly to the binary case, the composite trigger is defined as the disjunction of the component triggers and composite bound $\mathsf{B}(sl, sl')$ is defined from the component bounds:

$$\forall (sl_k)_k, (sl_k')_k.\; sl \in ||(sl_k)_k \;\wedge\; sl' \in ||(sl_k')_k \;\to\; \forall k.\; \mathsf{B}_k(sl_k, sl_k')$$

Now a generalization of Theorem 1 can be formulated. Most of the assumptions are those of Theorem 1 applied to all pairs Aut_k and $\mathsf{Aut}_{k'}$. An exception is secret polarization, which needs strengthening. We call the n component attacker models *uniquely secret-polarized* if there is a unique secret issuer, say, Aut_i, in the network. Formally: for all $k \neq i$ and Aut_k-transitions trn, $\mathsf{isSec}_k(trn)$ implies $\mathsf{isCom}_{k,i}(\mathsf{actOf}_k(trn))$.

Theorem 3. *Assume that the following properties hold for the I/O automata* $\mathsf{Aut}_1, \ldots, \mathsf{Aut}_n$ *with BD-security properties* $(\mathsf{P}_1), \ldots, (\mathsf{P}_n)$ *and communication infrastructure* match*:*

- *communication is pairwise dedicated*
- *the communication infrastructure between any two components is compatible and strong*
- *any two attacker models have observable network traffic*
- *the attacker models are uniquely secret-polarized*

If each (P_k) *holds for* Aut_k*, then* $||_{k\in\{1,\ldots,n\}}\ (\mathsf{P}_k)$ *holds for* $\prod^{\mathsf{match}}_{k\in\{1,\ldots,n\}}\mathsf{Aut}_k$.

Thus, the generalization to the n-ary case proceeds fairly smoothly, with the nuance that a single source of secrets is allowed in the whole network.

Back to CoSMeDis, to capture our concrete (P"), $\mathsf{match}_{k,k'}$ is defined just like in the binary case (shown in Fig. 1), but using the identifiers NID_k and $\mathsf{NID}_{k'}$ for Aut_k and $\mathsf{Aut}_{k'}$ instead of NID_1 and NID_2. Now (P") follows from this theorem along with the transfer theorem—the latter being used to customize the bound, similarly to the binary case.

VII. Verifying CoSMeDis's Confidentiality

We employ the last section's results to prove fine-grained declassification bounds for CoSMeDis's information sources.

A. Verified Properties

Our running example of a confidentiality property had the form: Nothing is inferable about a given secret (a post content) *unless* a trigger is being fired. The properties we had proved for the original CoSMed actually made stronger claims: Nothing is inferable about a given secret *beyond* the trace portions during which a trigger is active, i.e., when the observers' access to the secret is legitimate. This makes it possible to consider *dynamic* triggers, which can be repeatedly fired and canceled. For example, a user can become a friend of the post's owner, but later the friendship can be canceled by either user "unfriending" the other—only post updates performed outside the times of friendship should be protected from that user.

Technically, in the stronger properties the trigger is "swallowed" by the bound. So the price for the gained strength in confidentiality is a more complex bound, which operates on an enriched domain of secrets that include trigger information. For post confidentiality, the domain of secrets now consists not only of post contents, (psec, pst), but also of openness indicators, (osec, b), where b is a Boolean flag indicating whether the legitimate access window is open. The secret-producing function returns $\mathsf{osec}\ b$ only if the openness status changes, with b indicating the new status—e.g., b becomes True when an observer is marked as friend, and becomes False if the last observer is unfriended (and the other legitimate access conditions fail as well, e.g., the admin is not an observer). These indicators are used to formulate the bound in an access-window sensitive way. For example, if a trace produces the secrets $\langle(\mathsf{osec}, \mathsf{True}), (\mathsf{psec}, pst_1), (\mathsf{osec}, \mathsf{False}), (\mathsf{psec}, pst_2)\rangle$, then the bound protects pst_2, but not pst_1. This is because the update pst_1 occurred in a phase of the system execution where the observers had legitimate access to the post content, whereas the access window was closed before pst_2 occurred [12, §3.3].

Fortunately, our policy-agnostic theorems smoothly accommodate such complex bounds as well. In fact, the instantiation of the compositionality result works almost exactly the same. The only difference is that we work with a notion of secret that was already enriched with openness indicators. The amendment with communicating secrets is orthogonal—we partition the secrets of the form (psec, pst) in two categories, for update

Observers	Secrets	Bound
Group of users, UIDs	Content of a given post, say, PID	Updates performed while or last before one of the following holds: Some user in UIDs is the admin, is the post owner or is friend with the post owner PID is marked as public
	Friendship status between two given users, say, UID_1 and UID_2	Status changes performed while or last before the following holds: Some user in UIDs is the admin or is friend with UID_1 or UID_2
	Friendship requests between two given users, say, UID_1 and UID_2	Existence of accepted requests while or last before the following holds: Some user in UIDs is the admin or is friend with UID_1 or UID_2

Figure 7: Confidentiality properties for the original CoSMed

Observers	Secrets	Bound
n group of users, $\mathsf{UIDs}_1, \ldots, \mathsf{UIDs}_n$, one for each node	Content of a given post, say, PID of node NID_i	Updates performed while or last before one of the following holds: Some user in UIDs_i is NID_i's admin, is PID's owner or is friend with PID's owner PID is marked as public Some user in UIDs_j for $j \neq i$ is admin or remote friend with PID's owner
	Friendship status between two given users, say, UID_1 and UID_2 of NID_i	Status changes performed while or last before the following holds: Some user in UIDs_i is NID_i's admin or is friend with UID_1 or UID_2
	Friendship requests between two given users, say, UID_1 and UID_2 of NID_i	Existence of accepted requests while or last before the following holds: Some user in UIDs_i is NID_i's admin or is friend with UID_1 or UID_2

Figure 8: Confidentiality properties for CoSMeDis, lifted from CoSMed

and send: $(\mathsf{psec}, (\mathsf{upd}, pst))$ and $(\mathsf{psec}, (\mathsf{snd}, pst))$. Our bound extension (adding the correlation predicate) is also orthogonal.

The discussion leading to the compositionality theorems yields a heuristic for concretely achieving compositionality: Starting with a proved CoSMed instance of BD Security, split it in two CoSMeDis properties by identifying a secret issuer and a secret receiver; then strengthen the properties so that communication is acknowledged by both sides and has its non-confidential part observable; finally, adjust the bound to account for the new correlations introduced during strengthening. (Appendix B gives more details.)

We have used this heuristic to extend to CoSMeDis all the properties proved for the original CoSMed, summarized in Fig. 7—where the triggers are always vacuously false (since they are "swallowed" by the bounds) and the observers are always a given set of users. In Fig. 8, we summarize the end product after lifting these to CoSMeDis via our theorems.

Besides the already discussed post confidentiality, there are confidentiality properties regarding friendship status, i.e., the information on whether two users are friends, and friendship requests, i.e., the information on whether a user has issued a friendship request to another user. In both cases, the legitimate access windows are defined to mean that an observer is the admin or is currently friends with either of the involved users.

The application of our compositionality theorem to the confidentiality of friendship status and requests is easier than for post confidentiality. Unlike the latter, the former does not involve sharing secrets between nodes. Consequently, as seen in the corresponding entries in Figs. 7 and 8, the bound of the distributed system is the same as that of a single node. In addition to local friendship, we also verify confidentiality of *remote* friendship, as detailed in Appendix C.

B. Verification Technology Aspects

As discussed in Section IV-C, the functionality of CoSMeDis nodes has been specified in Isabelle as a particular I/O automaton, extending the previously formalized CoSMed I/O automaton—and this formed the basis of the nodes' implementation, via code extraction.

Our verification focused on the network formed of copies of this I/O automaton. First, we formalized Section VI's abstract theorems. The heart of Theorem 1, depicted in Fig. 5, is the construction of the alternative composed trace from two component traces tr_1 and tr_2 (whose observations and secrets have already been composed). The construction takes advantage of the assumed strength of communication and observations to define suitable compound-system transitions one at a time. It is guided by a tedious series of case distinctions: on whether the current compound-system observations and secrets were produced individually or by matching, on whether each of them is secret or observable, etc. Theorem 3 is proved inductively by iterating Theorem 1, the secret-issuer being composed with the other nodes, one at a time. The main difficulty with the transport Theorem 2 was its formulation as sufficiently expressive to capture our cases of interest. Once properly formulated, it followed by a straightforward manipulation of the quantifiers in the definition of BD security.

For the above theorems, we employed Isabelle's structured Isar proof language [63], which allowed us to document the nontrivial parts of the proofs (e.g., crucial case distinctions) in a mathematician's "pen-and-paper" style, while dismissing the trivial parts in an engineer's style, using automatic methods (cf. Appendix D). For example, the impossible cases in the proof of Theorem 1 (those that would contradict our assumptions) were discharged immediately without even spelling them out.

To facilitate the instantiation of these theorems, we proved

them within Isabelle modules called *locales* [37], which allow for the development of theorems parameterized by abstract data and assumptions. The locales automate the process of instantiating the theorems: The user provides concrete instances for the data and discharges the assumptions; in exchange, they obtain an unconditional version of the theorems for the given instance. For example, for Theorem 1 the data parameters were two I/O automata, two security properties, and a communication infrastructure. To anticipate our specific instantiations, we combined Theorems 3 (for n-ary composition) and 2 (for massaging the bound) into a single theorem, which is informed by a particular CoSMeDis aspect: The bound of the secret source is stronger than that of the other components, meaning it can be used (via Theorem 2) as the bound of the composite.

The compositionality framework took 5700 LOC and was built on top of a previously formalized framework for BD security (consisting of 1800 LOC). The system specification comprises 1500 LOC. As expected, the verification of the concrete instances (listed in Section VII-A) constituted the bulk of the development, 14400 LOC. The verification of each instance had two components: (1) proofs for the security of individual nodes and (2) verifications of the conditions for compositionality (leading to the corresponding instantiation of the locale).

For all but one case (remote friendship), we started with properties of CoSMed and split them into secret issuer and receiver properties for CoSMeDis, as prescribed by our heuristic. The original proofs for CoSMed were elaborate interactive proofs by unwinding [12, §4]. Their adaptation to CoSMeDis, with strengthening the attacker power, also went according to our heuristic. Nevertheless, this was laborious: The original proof scripts broke in places located deep inside nested case distinctions, hence to adapt them we needed to analyze large proof contexts. Due to the more complex bounds, the proofs for secret issuers were larger (and more time consuming) than those for the receivers. The average size of a proof for the former was 1500 LOC, about twice as large as for the latter.

In contrast to the unwinding proofs, the verification of the compositionality conditions was almost entirely automatic—thanks to the conditions being local (involving only actions and states, no traces or sequences of secrets). Indeed, a case distinction on all the CoSMeDis actions followed by "auto" (Isabelle's main automatic proof method) usually did the job, after instrumenting "auto" with the necessary simplification rules. The average size of an instantiation file, 150 LOC, mainly reflects the boilerplate for locale instantiation.

Overall, the verification of CoSMeDis consists of 21600 LOC, which required 4 person-months (including the formalization of the abstract framework, which involved trial and error). These followed as an extension of CoSMed's verification, consisting of 10000 LOC and having required 3 person-months. A large part of the CoSMeDis proofs were not developed from scratch, but as adaptations of previous CoSMed proofs. Together, these adapted proofs form 10000 LOC, of which roughly 8000 LOC are reused from CoSMed.

VIII. Discussion

Our formalized results are comprehensive with respect to CoSMeDis's high-level information flow, in that they cover all the secret sources of interest and establish fine-grained bounds for their flows. However, the formulation of confidentiality as BD security incurs the inherent limitations of possibilistic notions [38, §4.3]. For example, assume that the attacker can learn of the existence of close-in-time updates to posts belonging to two users. Then, also depending on other traffic information, he could infer a high probability of friendship between those users. This kind of attack cannot be captured by BD security, which would still proclaim that nothing is learned about the friendship status, since, as far as the attacker knows, both friendship and lack of friendship are *possible*.

Other attacks that are beyond the scope of BD security are those that occur *outside* the system. For example, we do not consider identity theft, and implicitly make the assumption that users only act on behalf of themselves. Another example is a user copying the content of his friend's post into his own public post, thus revealing it to an attacker who knows where this was copied from. In order for BD security to account for such a flow, it has to happen "officially," through the system—by a dedicated re-posting action that copies content from one post into another. To make an analogy with language-based security: If the content of a high variable influences that of a low variable via the programming language constructs, then the leak is typically caught; but if the user manually creates a correlation between the low and high console inputs to the program, the leak cannot be caught.

In addition to these theoretically inherent limitations, our verification work has practical limitations concerning code coverage, resilience under improvements or extensions, and the ability to cope with alternative social media architectures.

Coverage. First, our confidentiality guarantees refer to the *logic of the application layer*. They do not address network-level attacks, and implicitly count on communication being perfectly encrypted (even though, for achieving compositionality, we empower the attacker beyond the Dolev-Yao model).

Second, we only verified CoSMeDis's I/O automaton kernel, formalized in Isabelle and extracted into Scala (as discussed in Section IV-C). In particular, we trust the API layer to be a correct translator of http(s) requests to automaton actions and of automaton output to JSON output. Crucially, we trust that session management and the small piece of code for file management are implemented correctly. For example, mixing the identities of two users can entirely compromise confidentiality. Similarly, overwriting the image files between two users compromises post confidentiality. In summary, our security guarantees are valid for the overall system only if the code of the external layers does not introduce leaks. Extending the scope of verification to these layers would require us to step outside the reach of Isabelle. However, compared to the kernel, these layers require the establishment of much simpler, noninterference-like properties: that they transport the data back and forth between the end user and the verified kernel, without doing anything "interesting," like mixing identities. Such properties seem to be in the scope of static information-flow analyzers such as Joana [33]—so a hybrid verification scheme (as in, e.g., [39]) might be suitable here.

Future improvements. Our current implementation of CoSMeDis is only a prototype, still lacking some important features for production applications. In particular, (except for

the post image files) the system currently runs in memory, i.e., there is no database or other data persistence facility. To achieve data persistence on top of the current verification architecture, there are two possible approaches. We could use an in-memory database such as Redis [4], regularly storing snapshots for persistence. This worked excellently for CoCon, a system intended to be used for a single conference for a limited period of time, but will not scale to a full-fledged social media platform. A more realistic (but invasive) approach is to adapt the code generator to directly produce Scalatra code interacting with an SQL or NoSQL database (with NoSQL having the advantage that it maps naturally to the ML-style datatypes of Isabelle/HOL). Both solutions would introduce new information flow (from and to the persistent storage).

Future extensions. CoSMeDis currently offers a fairly restricted functionality. To become a practically usable platform, additional features must be added. Examples include searching for posts whose titles match a string, re-posting/re-tweeting a friend's post, offering lists of potential friends based on one's profile, and tagging a post by users who read it. Some of these features, e.g., searching, do not open new channels, hence can be added without affecting the security guarantees.

Other features would lead to a weakening of the guarantees. For example, if we want to allow re-posting of private posts, this would require amending the trigger to acknowledge a new legitimate flow. Yet other features, such as reader tagging, would not affect the existing guarantees, but would make nontrivial some previously trivial confidentiality questions: Under which conditions, and to which extent, can a group of users learn of a post's readers? In all these cases, BD security is flexible enough to express the new confidentiality situation, and it seems that our heuristic for establishing confidentiality would still work. However, the formal statements and proofs might need substantial revision. In order to still be able to instantiate our compositionality framework, we need to preserve its assumptions, notably secret polarization. The new features would need to be implemented in a way that respects the master-slave paradigm discussed in Section VI-A. For example, the re-posting of a remote post should proceed by notifying the original node.

Alternative architectures. CoSMeDis is a network of web applications following a client-server paradigm. This has the advantage that users can access the system from any device with a web browser. Moreover, the extension of CoSMed to CoSMeDis required only moderate changes to the code, leading to specification and proof reuse.

The disadvantage of this architecture is that users have to trust the operators of their node (unless they run one themselves) and of their friends' nodes, who have access to confidential information in the system. This can be alleviated to some degree by adding a layer of end-to-end encryption [9,10,30] on top. Moreover, there are distributed architectures for online social networks based on peer-to-peer (P2P) networks, avoiding the client-server paradigm [16,20,35]. They place the application logic on the clients instead of servers, and rely on data encryption to ensure privacy.

However, despite encryption and decentralization, we believe that security goals and challenges similar to those of our case study also arise when using such architectures—after all, confidential information is processed in plain text by the client software running on the end-points of the network. For example, a high-level security guarantee about the confidentiality of private posts would require each client in such a P2P network to preserve the confidentiality of posts received from the clients of friends. In principle, such a requirement could again be formulated as a BD Security property of an I/O automaton modeling the behavior of the client. However, instantiating our compositionality result to a P2P architecture appears to be problematic. This is due to the more complex communication topology, which goes beyond the single-master paradigm that we followed for CoSMeDis: In a P2P network, data is typically forwarded and stored along dynamic paths through the network, which collides with our assumption of unique secret polarization.

IX. Related Work

CoSMeDis belongs to a small, but expanding club of running systems proved to be secure using proof assistants, which includes an aircraft microprocessor [34] (in ACL2), a hardware architecture with information flow primitives [22] (in Coq), a separation kernel [21] (in HOL4), a noninterferent operating system kernel [51] (in Isabelle/HOL), a secure browser [36] (in Coq), and an e-voting system [39] (using the KeY theorem prover jointly with the Joana information flow analyzer).

In practice, security requirements in web applications are typically implemented using *access control*. In particular, for online social networks, relationship-based access control [27,54] supports policies depending on connections in the social graph, e.g., friendship links. Here, we aimed for more than access control: We wanted to protect secrets not only from direct illegitimate access, but also from leaking to unintended recipients who draw inferences based on observations of the system. Such inferences can easily evade access control.

Research in information flow security has a rich history, mostly unfolded from the pioneering notions of noninterference [28] and nondeducibility [62] (of which BD Security is a generalization). For the work we discuss below, we employ a well-known taxonomy of controlled information release (declassification) due to Sabelfeld and Sands [59], which is particularly relevant for the rich information exchange in web applications: Ideally, security policies should be able to describe precisely *what* information can be released (e.g., the content of a document), *by whom* (e.g., the owners or their designated "friends"), as well as *where* and/or *when* (e.g., only after the document is marked by the owner as public). We focus on the compositionality of the various approaches.

Language-based approaches [58] concentrate on specific programming languages. Since they operate on the level of language syntax, they often achieve an impressive degree of automation. For example, Jif [52] extends Java with security labels for data values and enforces security via a combination of static and run-time checking. It supports control over *who* may declassify information, but not *what* is declassified. Joana [33] checks noninterference of Java programs via static program analysis. Control of declassification is limited to *where* in the program declassification may occur. Jeeves [64] extends a core language for functionality with a language for flexible security policies. RF* [11] uses a relational Hoare logic to

reason about 2-safety properties of probabilistic programs, including language-based notions of information flow security. The What&Where security property [41] allows control over *what* is declassified by concurrent programs, but only in a non-interactive setting, not suitable for web applications. Secure multi-execution [24,57] is a run-time enforcement technique where multiple copies of a program are executed, one for each security level, controlling the information flows between the levels. Secure program partitioning [17,66] produces a distribution of the code and data in a program onto different hosts according to their different trust levels (e.g., trusted web server and untrusted client-side browser).

While each of the aforementioned approaches supports certain forms of declassification, they do not consider the setting and the kind of compositional reasoning that we have aimed for: *given* a network of communicating, multi-user systems, derive a global, complex information flow security property from the local security properties of the components. Instead, the preservation of security under composition (or, similarly challenging, *refinement* [50]) is often considered w.r.t. language constructs, such as sequential composition, if-then-else branches, or while loops, for example in [46].

DStar [67] and Fabric [40] do consider distributed systems, but they offer only coarse-grained control over what information may be declassified: an assignment of security labels to both data items and principals specifies which principals may declassify which data item. In contrast, the declassification bounds of BD Security specify what aspects of confidential information may be declassified on a semantic level, formulated in terms of arbitrary logical formulas. For example, the bound for remote friendship information (cf. Appendix C) specifies that an attacker who can observe network traffic may learn about the *existence* of a remote friend of a user UID on a given node, but not about the *identity* of that friend. The BD Security policy captures this in a declarative way, independently of the internal data structures used on the implementation level for storing and processing confidential information.

Our framework falls into the category of *system-based* approaches. They work with security properties expressed directly on the semantics, on variants of event systems or I/O automata. Early work in this category [47] has observed that even seemingly strong security properties are not preserved under composition in general. Subsequently, comprehensive frameworks have been developed for the composition of security properties in various settings, e.g., event systems [44], reactive systems [56] and process calculi [15,26]. Some of these focus on formulating classes of security properties that are always guaranteed to be preserved under a given notion of composition, such as McCullough's Restrictiveness [48]. Others, such as Mantel's MAKS framework [42,44], formulate side conditions on the local security properties that guarantee compositionality. Our compositionality result follows the latter tradition: Our assumptions of observable network traffic and strong and compatible communication infrastructures fall into the first case of Mantel's Generalized Zipping Lemma [44, Lemma 1]. The other cases of that lemma deal with scenarios where some communication transitions are neither observable nor confidential, which leads to additional requirements on the local security properties. The MAKS framework does not require secret polarization: Its security properties are fine-tuned so that matching problems due to the scheduling of secrets cannot occur. Hence, its compositionality result is less restrictive on the communication infrastructure than ours. The earlier security notions of McLean [49] and Zakinthinos and Lee [65] are captured as MAKS properties [45].

The above frameworks could not be used for our case study, because they focus on very strict, noninterference-like notions of security without support for controlled declassification (except for very specific notions, such as intransitive noninterference [43]). Chong and van der Meyden [18] discuss information flow policies (called *architectures*), where *filter functions* are used to restrict what information may flow between domains, together with an interpretation of the resulting security properties in terms of an epistemic logic. However, they do not consider compositional reasoning in our sense, i.e., composing the security properties of multiple components to a security property of the overall system. The same applies to the work on temporal logics and model checking approaches for hyperproperties [19], of which information flow security properties are an instance. Greiner and Grahl [29] present a compositionality result that supports *what*-declassification control, specified in terms of equivalence relations on communication events, but it cannot express *dynamically changing* confidentiality requirements—as needed for web applications in general and for CoSMeDis in particular. For example, whether a given post p by user u is confidential for an observer depends on the (dynamically changing) visibility setting of p and/or the friendship status between u and the observer.

The security notion from the literature that most closely resembles ours is a recent one by Guttman and Rowe [31], formulated on top of *blur operators*, which are similar to the declassification bounds used for BD Security. A compositionality result is presented that focuses on a different question than ours: Instead of composing the security guarantees of the individual nodes in a network, it considers a *partitioning* of the network into a core of nodes that may handle secret information and an outer part that is purely observing. It is proved that the security guarantees of the core are preserved under modification of the outer part. This is not what we needed in our case study: In CoSMeDis, *all* nodes of the network potentially handle secret information, so the "core" is the complete network. Our framework deals with the question of how the security properties of the nodes *inside* the core compose to an overall confidentiality guarantee.

ACKNOWLEDGMENTS

We are indebted to Jasmin Blanchette, Simon Greiner, Dieter Hutter, Nikhil Swamy, Dmitriy Traytel, and the anonymous reviewers for very useful comments and suggestions. We gratefully acknowledge support from:

- Innovate UK through the Knowledge Transfer Partnership 010041 between Caritas Anchor House and Middlesex University: "The Global Noticeboard (GNB): a verified social media platform with a charitable, humanitarian purpose",
- EPSRC through grants "VOWS" (EP/N019547/1) and "VRBMAS" (EP/K033921/1),
- DFG through grants "MORES" (Hu 737/5-2) and "SecDed" (Ni 491/13-3) in the priority program "RS^3 – Reliably Secure Software Systems" (SPP 1496).

REFERENCES

[1] "CoCon website," http://www21.in.tum.de/~popescua/rs3/CoCon.html.

[2] "Facebook may have leaked your personal information: Symantec," Reuters, May 2011, http://www.reuters.com/article/us-facebook-idUSTRE74A07R20110511.

[3] "The Heartbleed bug," http://heartbleed.com/.

[4] "The Redis website," https://redis.io/.

[5] "The CoSMeDis Homepage," http://andreipopescu.uk/CoSMeDis.html.

[6] "Caritas Anchor House," http://caritasanchorhouse.org.uk/, 2016.

[7] "CoSMed website," https://cosmed.globalnoticeboard.com, 2016.

[8] "The diaspora* project," https://diasporafoundation.org/, 2016.

[9] M. Backes, M. Maffei, and K. Pecina, "A security API for distributed social networks," in *NDSS*. The Internet Society, 2011.

[10] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," in *SIGCOMM*. ACM, 2009, pp. 135–146.

[11] G. Barthe, C. Fournet, B. Grégoire, P. Strub, N. Swamy, and S. Z. Béguelin, "Probabilistic relational verification for cryptographic implementations," in *POPL*, 2014, pp. 193–206.

[12] T. Bauereiß, A. P. Gritti, A. Popescu, and F. Raimondi, "CoSMed: A confidentiality-verified conference management system," in *ITP*, 2016.

[13] J. C. Blanchette, S. Böhme, A. Popescu, and N. Smallbone, "Encoding monomorphic and polymorphic types," *Logical Methods in Computer Science*, vol. 12, no. 4, 2016.

[14] J. C. Blanchette and S. Merz, Eds., *Interactive Theorem Proving - 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings*, vol. 9807, 2016.

[15] A. Bossi, D. Macedonio, C. Piazza, and S. Rossi, "Information flow in secure contexts," *Journal of Computer Security*, vol. 13, no. 3, pp. 391–422, 2005.

[16] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta, "PeerSoN: P2p Social Networking: Early Experiences and Insights," in *Second ACM EuroSys Workshop on Social Network Systems*. ACM, 2009, pp. 46–52.

[17] S. Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng, "Building secure web applications with automatic partitioning," *Commun. ACM*, vol. 52, no. 2, pp. 79–87, 2009.

[18] S. Chong and R. V. D. Meyden, "Using Architecture to Reason About Information Security," *ACM Trans. Inf. Syst. Secur.*, vol. 18, no. 2, pp. 8:1–8:30, Dec. 2015.

[19] M. R. Clarkson, B. Finkbeiner, M. Koleini, K. K. Micinski, M. N. Rabe, and C. Sánchez, "Temporal logics for hyperproperties," in *POST*, 2014, pp. 265–284.

[20] L. A. Cutillo, R. Molva, and T. Strufe, "Safebook: A privacy-preserving online social network leveraging on real-life trust," *IEEE Communications Magazine*, p. 95, 2009.

[21] M. Dam, R. Guanciale, N. Khakpour, H. Nemati, and O. Schwarz, "Formal verification of information flow security for a simple ARM-based separation kernel," in *CCS*, 2013, pp. 223–234.

[22] A. A. de Amorim, N. Collins, A. DeHon, D. Demange, C. Hritcu, D. Pichardie, B. C. Pierce, R. Pollack, and A. Tolmach, "A verified information-flow architecture," in *POPL*, 2014, pp. 165–178.

[23] H. de Nivelle, Ed., *Automated Reasoning with Analytic Tableaux and Related Methods - 24th International Conference, TABLEAUX 2015, Wrocław, Poland, September 21-24, 2015. Proceedings*, vol. 9323, 2015.

[24] D. Devriese and F. Piessens, "Noninterference through secure multi-execution," in *IEEE Symposium on Security and Privacy*, 2010, pp. 109–124.

[25] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Information Theory*, vol. 29, no. 2, pp. 198–207, 1983.

[26] R. Focardi and R. Gorrieri, "Classification of security properties (part i: Information flow)," in *FOSAD*, 2000, pp. 331–396.

[27] P. W. L. Fong, M. M. Anwar, and Z. Zhao, "A privacy preservation model for Facebook-style social network systems," in *ESORICS*, 2009, pp. 303–320.

[28] J. A. Goguen and J. Meseguer, "Unwinding and inference control," in *IEEE Symposium on Security and Privacy*, 1984, pp. 75–87.

[29] S. Greiner and D. Grahl, "Non-interference with what-declassification in component-based systems," in *CSF*, 2016, pp. 253–267.

[30] S. Guha, K. Tang, and P. Francis, "NOYB: Privacy in Online Social Networks," in *Workshop on Online Social Networks*. ACM, 2008, pp. 49–54.

[31] J. D. Guttman and P. D. Rowe, "A cut principle for information flow," in *CSF*, 2015, pp. 107–121.

[32] F. Haftmann and T. Nipkow, "Code generation via higher-order rewrite systems," in *FLOPS 2010*, 2010, pp. 103–117.

[33] C. Hammer and G. Snelting, "Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs," *Int. J. Inf. Sec.*, vol. 8, no. 6, pp. 399–422, 2009.

[34] D. S. Hardin, E. W. Smith, and W. D. Young, "A robust machine code proof framework for highly secure applications," in *ACL2*, 2006, pp. 11–20.

[35] S. Jahid, S. Nilizadeh, P. Mittal, N. Borisov, and A. Kapadia, "DECENT: A decentralized architecture for enforcing privacy in online social networks," in *PerCom*. IEEE, 2012, pp. 326–332.

[36] D. Jang, Z. Tatlock, and S. Lerner, "Establishing browser security guarantees through formal shim verification," in *USENIX Security*, 2012, pp. 113–128.

[37] F. Kammüller, M. Wenzel, and L. C. Paulson, "Locales—a sectioning concept for Isabelle," in *TPHOLs*, 1999, pp. 149–166.

[38] S. Kanav, P. Lammich, and A. Popescu, "A conference management system with verified document confidentiality," in *CAV*, 2014, pp. 167–183.

[39] R. Küsters, T. Truderung, B. Beckert, D. Bruns, M. Kirsten, and M. Mohr, "A hybrid approach for proving noninterference of java programs," in *CSF*, 2015, pp. 305–319.

[40] J. Liu, M. D. George, K. Vikram, X. Qi, L. Waye, and A. C. Myers, "Fabric: a platform for secure distributed computation and storage," in *SOSP*, 2009, pp. 321–334.

[41] A. Lux, H. Mantel, and M. Perner, "Scheduler-independent declassification," in *MPC*, 2012, pp. 25–47.

[42] H. Mantel, "Possibilistic definitions of security - an assembly kit," in *CSFW*, 2000, pp. 185–199.

[43] ——, "Information flow control and applications - bridging a gap," in *FME*, 2001, pp. 153–172.

[44] ——, "On the composition of secure systems," in *IEEE Symposium on Security and Privacy*, 2002, pp. 88–101.

[45] ——, "The framework of selective interleaving functions and the modular assembly kit," in *FMSE*, 2005, pp. 53–62.

[46] H. Mantel and A. Reinhard, "Controlling the what and where of declassification in language-based security," in *Programming Languages and Systems*, ser. LNCS, 2007, vol. 4421, pp. 141–156.

[47] D. McCullough, "Specifications for multi-level security and a hook-up property," in *IEEE Symposium on Security and Privacy*, 1987.

[48] ——, "A hookup theorem for multilevel security," *IEEE Trans. Software Eng.*, vol. 16, no. 6, pp. 563–568, 1990.

[49] J. McLean, "A general theory of composition for trace sets closed under selective interleaving functions," in *IEEE Symposium on Security and Privacy*, 1994, pp. 79–93.

[50] C. Morgan, "The shadow knows: Refinement and security in sequential programs," *Sci. Comput. Program.*, vol. 74, no. 8, pp. 629–653, 2009.

[51] T. C. Murray, D. Matichuk, M. Brassil, P. Gammie, T. Bourke, S. Seefried, C. Lewis, X. Gao, and G. Klein, "seL4: From general purpose to a proof of information flow enforcement," in *IEEE Symposium on Security and Privacy*, 2013, pp. 415–429.

[52] A. C. Myers, "Jflow: Practical mostly-static information flow control," in *POPL*, 1999, pp. 228–241.

[53] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer, 2002.

[54] R. Pardo and G. Schneider, "A formal privacy policy framework for social networks," in *SEFM*, 2014, pp. 378–392.

[55] L. C. Paulson and J. C. Blanchette, "Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers," in *IWIL*, 2010.

[56] W. Rafnsson and A. Sabelfeld, "Compositional information-flow security for interactive systems," in *CSF*, 2014, pp. 277–292.

[57] ——, "Secure multi-execution: Fine-grained, declassification-aware, and transparent," *Journal of Computer Security*, vol. 24, no. 1, pp. 39–90, 2016.

[58] A. Sabelfeld and A. C. Myers, "Language-based information-flow security," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 1, pp. 5–19, 2003.

[59] A. Sabelfeld and D. Sands, "Declassification: Dimensions and principles," *Journal of Computer Security*, vol. 17, no. 5, pp. 517–548, 2009.

[60] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Comput. Surv.*, vol. 37, no. 1, pp. 42–81, 2005.

[61] "The Scalatra Web Framework," 2016, http://scalatra.org/.

[62] D. Sutherland, "A model of information," in *9th National Security Conf.*, 1986, pp. 175–183.

[63] M. Wenzel, "Isar—a generic interpretative approach to readable formal proof documents," in *TPHOLs*, 1999, pp. 167–184.

[64] J. Yang, K. Yessenov, and A. Solar-Lezama, "A language for automatically enforcing privacy policies," in *POPL*, 2012, pp. 85–96.

[65] A. Zakinthinos and E. S. Lee, "A general theory of security properties," in *IEEE Symposium on Security and Privacy*, 1997, pp. 94–102.

[66] S. Zdancewic, L. Zheng, N. Nystrom, and A. C. Myers, "Secure program partitioning," *ACM Trans. Comput. Syst.*, vol. 20, no. 3, pp. 283–328, 2002.

[67] N. Zeldovich, S. Boyd-Wickizer, and D. Mazières, "Securing distributed systems with information flow control," in *NSDI*, 2008, pp. 293–308.

APPENDIX

A. More Details on the Security Transport Theorem

Here is the formal definition of the notion of "stronger security model," used in Theorem 2.

Definition 3. *Let* Aut *be an I/O automaton and* (P) *and* (P') *two security properties operating on it.* (P) *is said to have a* stronger security model *than* (P') *if there are two partial functions* $f : \mathsf{Sec} \rightharpoonup \mathsf{Sec}'$ *and* $g : \mathsf{Obs} \rightharpoonup \mathsf{Obs}'$ *from the secrets and observations of* (P) *to those of* (P') *that preserve the secrecy and observation infrastructures, the bounds and the triggers, i.e., such that the following are true:*

- $\mathsf{isSec}'(trn)$ *holds iff* $\mathsf{isSec}(trn)$ *and* $\mathsf{getSec}(trn) \in \mathrm{dom}(f)$, *and in this case* $f(\mathsf{getSec}(trn)) = \mathsf{getSec}'(trn)$
- $\mathsf{isObs}'(trn)$ *holds iff* $\mathsf{isObs}(trn)$ *and* $\mathsf{getObs}(trn) \in \mathrm{dom}(g)$, *and in this case* $g(\mathsf{getObs}(trn)) = \mathsf{getObs}'(trn)$
- $\mathsf{T}(trn)$ *implies* $\mathsf{T}'(trn)$
- $\mathsf{B}'(sl', tl')$ *and* $\mathsf{map}_f(sl) = sl'$ *imply that there is a* tl *such that* $\mathsf{map}_f(tl) = tl'$ *and* $\mathsf{B}(sl, tl)$

Above, $\mathsf{map}_f : \mathsf{Sec}^* \to \mathsf{Sec}'^*$ *denotes the secret-wise extension of the partial function* $f : \mathsf{Sec} \rightharpoonup \mathsf{Sec}'$ *to sequences, omitting any secrets* $s \notin \mathrm{dom}(f)$.

In case f is defined and injective on all secrets occurring in Aut, then it can be inverted and the last condition above simplifies to checking that $\mathsf{B}'(sl', tl')$ implies $\mathsf{B}(\mathsf{map}_{f^{-1}}(sl'), \mathsf{map}_{f^{-1}}(tl'))$.

Choosing *partial* functions f and g allows us to *weaken* the power of the observer and the notion of secrets by making the sets of observable and secret transitions smaller. In particular, observable transitions whose observations are not translated by g become *unobservable* in the new security model.

To prove that $(\mathsf{P}_1) \,||\, (\mathsf{P}_2)$ implies (P') for the main paper's running example, we use the theorem where $\mathsf{Obs}' = \mathsf{Obs}$, g is the identity, $\mathsf{Sec}' = \mathsf{Sec}_1$, and f extracts the secret s_1 out of $(1, s_1)$ or (s_1, s_2).

Note that Aut_2 does not produce local secrets, hence $(2, s_2)$ never occurs. Moreover, f is injective on valid secrets, because the sets of secrets occurring locally in Aut_1 and in communication with Aut_2 are disjoint, and, in a communication, the secret received by Aut_2 is uniquely determined by the secret sent by Aut_1. The inverse of f is

$$f^{-1}(s_1) = \begin{cases} (1, (\mathsf{upd}, pst)) & \text{if } s_1 = (\mathsf{upd}, pst) \\ ((\mathsf{snd}, pst), pst) & \text{if } s_1 = (\mathsf{snd}, pst) \end{cases}$$

The above assumptions are then immediate to check.

We also rely on Theorem 2 to prove that the various amendments of the component properties required to achieve compositionality (illustrated in Section V-B) are indeed strengthenings of the previous versions—which is important for guaranteeing that our quest for compositionality did not weaken any bit of the component properties we started with. Thus, to prove that the version of property (P_1) after amendments 1–3 implies the original, we define g as the identity on local observations and f as $f((\mathsf{upd}, pst)) = pst$.

B. Heuristic for Achieving Compositionality

Context. We have a monolithic system modeled as an I/O automaton Aut, delivered with a confidentiality guarantee (P) modeled as BD Security. We extend the system to a distributed system, consisting of several nodes able to communicate. Each node is an extension Aut' of Aut, with new actions for inter-node communication. The question is what confidentiality property can we prove for the distributed system.

Step 1. By its nature, (P) states something about a notion of secret and the way it is protected during interaction with the outside world, which takes place through actions and outputs of the original automaton Aut. We analyze what happens with the secret during inter-node communication in Aut', identifying the roles of *secret issuer* and *secret receiver* for the nodes. This leads to a split of (P) in two variants, (P_1) and (P_2).

Step 2. If necessary, we modify the secrecy infrastructure of (P_1) and/or (P_2) so that the communication of secret-correlated items is acknowledged by both as a secret-producing action. This may require a modification of the bound as well, to account for the correlation.

Step 3. We strengthen the observation power for (P_1) and (P_2) by allowing the observer to access any communication information that does not compromise the secrets.

Step 4. If we can identify a unique source for the secrets, we apply Theorem 3 for a distributed system consisting of one component satisfying (P_1) and $n-1$ components satisfying (P_2). Then we apply Theorem 2 to customize the compound bound into a property that uses the notion of secret from the secret-issuer component only.

C. More Details on the Verified Properties

Post Confidentiality. The post confidentiality property of the original CoSMed has a "dynamic" bound, incorporating

changes in the confidentiality of the post due to visibility updates and friending or unfriending of observers. We summarize the formal attacker models (AM) and security properties (SP) in Table I. For a detailed explanation, see [12]. The corresponding property of the issuer node of CoSMeDis extends that of the original CoSMed with the sending of posts: the additions are highlighted. On the receiver side, nothing is leaked to local observers beyond the number of updates (because communication traffic is observable). Only when an observer is added as a remote friend of the post owner, or the post is marked as public, the trigger fires and the post is declassified. The property for the entire distributed system has the observations built from all the component observations (as described in the main paper's *n*-ary compositionality theorem). However, the secrets, as well as the bound, are those of the secret issuer component—this is because we show the end product property, obtained after applying both the compositionality and the transfer theorems.

Friendship Confidentiality. For CoSMed, we had proved confidentiality properties for local friendship status of any (arbitrary but fixed) users UID_1 and UID_2, and friendship requests between them. Observers learn nothing about their friendship *status* beyond the status (and updates to it) during phases in which one of the observers is friends with either UID_1 or UID_2. In particular, if none of the observers ever become friends with UID_1 or UID_2, then they never learn anything about their friendship status. Moreover, observers learn nothing about the *content* of friendship requests. However, observers always have the (static) knowledge that some friendship request precedes the establishment of a friendship between UID_1 and UID_2. This is made explicit in the bound for friendship requests. We refer to [12] for details.

We have proved that these confidentiality properties of local friendship information still hold in CoSMeDis as they are. This is to be expected (and desired), since local friendship does not involve communication between nodes. However, *remote* friendship does. Hence, we prove an additional property for CoSMeDis, summarized in Table II. We consider the remote friends of an arbitrary, but fixed user UID of node Aut_i, who is not an observer. Since we assume communication traffic to be observable, we can't keep secret that a remote friendship action occurred and which node it targeted, but we assume that the secret content is unobservable, namely the *type* of remote friendship action (addition or deletion, formalized as a Boolean flag), as well as *who* was added or deleted as a remote friend. Indeed, we verify that this information is kept secret in CoSMeDis from observers who are not friends of UID. However, from observing the occurrence of remote friendship actions and combining it with the static knowledge that addition and deletion of any given remote friend can only occur alternatingly (first addition, then deletion, then addition again, and so on), an observer can deduce the *existence* of remote friends of UID on any given node Aut_j. The overall property we verify is

(P) A coalition of n groups of users, UIDs_k for each Aut_k, can learn nothing about the remote friends of the user UID of Aut_i beyond the existence of remote friends on any given node Aut_j unless one of UIDs_i is friends with UID.

Formally, the bound given in Table II states that it must be possible to *replace* the parameters of remote friendship actions arbitrarily (modulo alternation of addition and deletion). The bounds for issuer and receiver nodes are symmetric, because secrets are only generated during communication. Finally, the trigger makes explicit that the friend list of UID is legitimately declassified to local friends of UID (we have only implemented the listing of friends locally, not remotely, in CoSMeDis).

D. More Details on the Verification Technology

Besides the internal automation ("auto" and friends), in Isabelle there is the possibility to invoke external fully automatic provers via the Sledgehammer tool [55], It differs from internal automation in that it requires no instrumentation (of what facts to invoke in the proof, to add to the simplifier, etc.). Instead, the tool applies a relevance filter to identify facts that are likely to be useful for the stated goal; these facts are translated to first-order logic and handed over to the automatic provers; a possible positive answer from any of the provers (which also contains the much smaller set of actually used facts) is translated back into Isabelle/HOL's logic, where the original goal is discharged [13, §7].

Sledgehammer's hassle-free automation scheme was very handy when porting and strengthening unwinding proofs from CoSMed to CoSMeDis. We ended up with an average of 4 successful invocations of Sledgehammer per 100 LOC, which is quite large given the Isar language verbosity. By contrast, internal automation was sufficient for the more regular pattern needed for verifying the compositionality conditions.

E. Combining Independent Secret Sources

For simplicity, in this paper we have always considered the confidentiality of *one* secret source at a time, e.g., one given (arbitrary but fixed) post, or the friendship information between two given users. A legitimate question is therefore how to deal with *multiple sources simultaneously*.

E.g., consider the confidentiality of two different posts, PID_i in Aut_i and PID_j in Aut_j. We can instantiate the results of the paper for each post separately and obtain *two* security properties of the distributed system. We can easily *combine* these two properties, relying on two key assumptions:

1) The secrets are *independent* of each other. Indeed, updates to different posts are completely orthogonal in the system; there is no interference between different posts.
2) The *scheduling* of the different secrets is *not confidential*; e.g., the contents of PID_i and PID_j are considered confidential, but the relative timing of uploads is not.

The first assumption guarantees the soundness of our approach to first consider the secrets in isolation, not having to worry about possible inter-dependencies. The second assumption is important, because it allows us to ignore the scheduling of secrets—after composition. Before composing the system, this would not be possible, because scheduling information is still needed for the composition of traces.

We formalize these assumptions as follows. Let (P_i) and (P_j) be two security properties of the *same* system, where (P_i) comprises the observation producing function O_i, the secret producing function S_i, the trigger T_i, and the bound B_i, and analogously for (P_j).

Table I: Post confidentiality

For the original CoSMed	AM	$\mathsf{Sec} = \mathsf{psec}\ \mathsf{Post} + \mathsf{osec}\ \mathsf{Bool}$ $\mathsf{isSec}\ (\sigma, a, o, \sigma')$ iff (1) $(o = \mathsf{outOK} \wedge \exists uid, pst.\ a = (\mathsf{updatePost}, uid, \mathsf{PID}, pst)) \vee$ (2) $\mathsf{open}(\sigma') \neq \mathsf{open}(\sigma)$ where $\mathsf{open}(\sigma)$ iff $\mathsf{PID} \in \mathsf{postIDs}(\sigma) \wedge \mathsf{admin}(\sigma) \in \mathsf{UIDs}_i \vee \mathsf{owner}(\sigma, \mathsf{PID}) \in \mathsf{UIDs}_i \vee \mathsf{UIDs}_i \cap \mathsf{friendIDs}(\sigma, \mathsf{owner}(\sigma, \mathsf{PID})) \neq \emptyset \vee \mathsf{vis}(\sigma', \mathsf{PID}) = \mathsf{public}$ $\mathsf{getSec}\ (\sigma, a, o, \sigma') = (\mathsf{psec}, pst)$ in case (1) and $(\mathsf{osec}, \mathsf{open}\ \sigma)$ in case (2) $\mathsf{Obs} = \mathsf{Act} \times \mathsf{Out}$ $\mathsf{isObs}\ (\sigma, a, o, \sigma')$ iff $\mathsf{userOf}(a) \in \mathsf{UIDs}$ $\mathsf{getObs}\ (\sigma, a, o, \sigma') = (a, o)$
	SP	B is defined as follows, mutually inductively with another predicate BO: $\dfrac{pstl \neq [] \to pstl' \neq []}{\mathsf{B}\ (\mathsf{map}_{\mathsf{psec}}(pstl),\ \mathsf{map}_{\mathsf{psec}}(pstl'))}$ $\dfrac{\mathsf{B}(sl, sl')}{\mathsf{BO}\ (\mathsf{map}_{\mathsf{psec}}(pstl) \cdot (\mathsf{osec}, \mathsf{False}) \cdot sl,\ \mathsf{map}_{\mathsf{psec}}(pstl) \cdot (\mathsf{osec}, \mathsf{False}) \cdot sl')}$ $\mathsf{BO}\ (\mathsf{map}_{\mathsf{psec}}(pstl), \mathsf{map}_{\mathsf{psec}}(pstl))$ $\dfrac{\mathsf{BO}(sl, sl') \qquad pstl \neq [] \leftrightarrow pstl' \neq [] \qquad pstl \neq [] \to \mathsf{last}\ pstl = \mathsf{last}\ pstl'}{\mathsf{B}\ (\mathsf{map}_{\mathsf{psec}}(pstl) \cdot (\mathsf{osec}, \mathsf{True}) \cdot sl,\ \mathsf{map}_{\mathsf{psec}}(pstl') \cdot (\mathsf{osec}, \mathsf{True}) \cdot sl')}$ T is vacuously False
For CoSMeDis –secret issuer i–	AM_i	$\mathsf{Sec}_i = \mathsf{upd}\ (\mathsf{psec}\ \mathsf{Post} + \mathsf{osec}\ \mathsf{Bool}) + \mathsf{snd}\ \mathsf{Post}$ $\mathsf{isSec}_i(\sigma, a, o, \sigma')$ iff (1) $(o = \mathsf{outOK} \wedge \exists uid, pst.\ a = (\mathsf{updatePost}, uid, \mathsf{PID}, pst)) \vee$ (2) $\mathsf{open}(\sigma') \neq \mathsf{open}(\sigma) \vee$ (3) $(\exists uid, pst, uid', v.\ o = (pst, uid', v) \wedge a = (\mathsf{sendPost}, uid, nid, \mathsf{PID}))$ $\mathsf{getSec}_i(\sigma, a, o, \sigma') = \mathsf{upd}\ (\mathsf{psec}, pst)$ in case (1) and $(\mathsf{osec}, \mathsf{open}\ \sigma)$ in case (2) and $\mathsf{snd}\ pst$ in case (3) $\mathsf{Obs}_i = \mathsf{Act} \times \mathsf{Out}$ $\mathsf{isObs}_i\ (\sigma, a, o, \sigma')$ iff $\mathsf{userOf}(a) \in \mathsf{UIDs}_i \vee \exists k.\ \mathsf{isCom}_{i,k}(a)$ $\mathsf{getObs}_i\ (\sigma, a, o, \sigma') = (\mathsf{purgeA}_{\mathsf{PID}}(a),\ \mathsf{purgeO}_{\mathsf{PID}}(o))$
	SP_i	$\mathsf{B}_i(sl, sl')$ iff $\mathsf{B}(\mathsf{filter}_{\mathsf{upd}}(sl), \mathsf{filter}_{\mathsf{upd}}(sl')) \wedge \mathsf{corr}(sl')$ T_i is vacuously False
For CoSMeDis –other components j–	AM_j	$\mathsf{Sec}_j = \mathsf{Post}$ $\mathsf{isSec}_j\ (\sigma, a, o, \sigma')$ iff $o_j = \mathsf{outOK} \wedge \exists pst, uid, v.\ a = (\mathsf{receivePost}, \mathsf{NID}_i, \mathsf{PID}, pst, uid, v)$ $\mathsf{getSec}_j\ (\sigma, a, o, \sigma') = pst$ $\mathsf{Obs}_j = \mathsf{Act} \times \mathsf{Out}$ $\mathsf{isObs}_j\ (\sigma, a, o, \sigma')$ iff $\mathsf{userOf}(a) \in \mathsf{UIDs}_j \vee \exists k.\ \mathsf{isCom}_{j,k}(a)$ $\mathsf{getObs}_j\ (\sigma, a, o, \sigma') = (\mathsf{purgeA}_{\mathsf{PID}}(a), \mathsf{purgeO}_{\mathsf{PID}}(o))$
	SP_j	$\mathsf{B}_j(sl, sl')$ iff $\mathsf{length}\ sl = \mathsf{length}\ sl'$ $\mathsf{T}_j\ (\sigma, a, o, \sigma')$ iff $\begin{cases} \mathsf{admin}(\sigma') \in \mathsf{UIDs}_j \vee \\ \mathsf{UIDs}_j \cap \mathsf{remoteFriendIDs}(\sigma', \mathsf{NID}_i, \mathsf{remoteOwner}(\sigma', \mathsf{NID}_i, \mathsf{PID})) \neq \emptyset \vee \\ \mathsf{remoteVis}(\sigma', \mathsf{NID}_i, \mathsf{PID}) = \mathsf{public} \end{cases}$
For CoSMeDis –entire system–	AM	$\mathsf{Sec} = \mathsf{Sec}_i$ $\mathsf{isSec} = \mathsf{isSec}_i$ $\mathsf{getSec} = \mathsf{getSec}_i$ $\mathsf{Obs} = \mathsf{Obs} = \sum_{k \in \{1,\ldots,n\}} \mathsf{Obs}_k + \sum_{k,k' \in \{1,\ldots,n\}} \mathsf{Obs}_k \times \mathsf{Obs}_{k'}$ $\mathsf{isObs}(trn) = \begin{cases} \mathsf{isObs}_k(trn_k) \wedge \mathsf{isObs}_{k'}(trn_{k'}) & \text{if } trn = (k, trn_k, k', trn_{k'}) \\ \mathsf{isObs}_k(trn_k) & \text{if } trn = (k, trn_k) \end{cases}$ $\mathsf{getObs}(trn) = \begin{cases} (k, \mathsf{getObs}_k(trn_k), k', \mathsf{getObs}_{k'}(trn_{k'})) & \text{if } trn = (k, trn_k, k', trn_{k'}) \\ (k, \mathsf{getObs}_k(trn_k)) & \text{if } trn = (k, trn_k) \end{cases}$
	SP	$\mathsf{B} = \mathsf{B}_i$ $\mathsf{T}(trn) = \begin{cases} \mathsf{T}_k(trn_k) \vee \mathsf{T}_{k'}(trn_{k'}) & \text{if } trn = (k, trn_k, k', trn_{k'}) \\ \mathsf{T}_k(trn_k) & \text{if } trn = (k, trn_k) \end{cases}$

Table II: Remote friendship confidentiality

For CoSMeDis –secret issuer i–	AM_i	$\mathsf{Sec}_i = \mathsf{NodeID} \times \mathsf{UserID} \times \mathsf{Bool}$ $\mathsf{isSec}_i(\sigma, a, o, \sigma') = (o = \mathsf{outOK} \wedge \exists uid, nid, st.\ a = (\mathsf{sendUpdateRFriend}, \mathsf{UID}, nid, uid, st) \wedge uid \notin \mathsf{UIDs}_{nid})$ $\mathsf{getSec}_i(\sigma, (\mathsf{sendUpdateRFriend}, \mathsf{UID}, nid, uid, st), o, \sigma') = (nid, uid, st)$ $\mathsf{Obs}_i = \mathsf{Act} \times \mathsf{Out}$ $\mathsf{isObs}_i\ (\sigma, a, o, \sigma')$ iff $\mathsf{userOf}(a) \in \mathsf{UIDs}_i \vee \exists k.\ \mathsf{isCom}_{i,k}(a)$ $\mathsf{getObs}_i\ (\sigma, a, o, \sigma') = (\mathsf{purgeA}_{\mathsf{UID}}(a), \mathsf{purgeO}_{\mathsf{UID}}(o))$
	SP_i	$\mathsf{B}_i(sl, sl')$ iff $\mathsf{BC}(sl, sl') \wedge \mathsf{alter}(sl')$, where • $\mathsf{alter}(sl')$ states that friendship creation and deletion occurs alternatingly in sl for each remote user, and • BC is defined inductively by (1) $\mathsf{BC}([], [])$ and (2) $\mathsf{BC}((nid, uid, st) \cdot sl, (nid, uid', st') \cdot sl')$ iff $uid' \notin \mathsf{UIDs}_{nid} \wedge \mathsf{BC}(sl, sl')$ $\mathsf{T}_i(\sigma, a, o, \sigma') = (\exists uid \in \mathsf{UIDs}_i.\ uid \in \mathsf{friendIDs}(\sigma', \mathsf{UID}))$
For CoSMeDis –other components j–	AM_j	same as for the issuer, only replacing sendUpdateRFriend actions by receiveUpdateRFriend actions coming from the issuer i
	SP_j	$\mathsf{B}_j = \mathsf{B}_i$ T_j is vacuously False
For CoSMeDis –entire system–	AM	analogously to the attacker model for the network case of post confidentiality in Table I
	SP	$\mathsf{B} = \mathsf{B}_i$ $\mathsf{T} = \mathsf{T}_i$

We capture the first assumption by assuming that the secret of one security property is *observable* in the other attacker model. Formally, the observation function of one property is assumed to fully determine the secret and the trigger information of the other, i.e., $\mathsf{O}_i(tr) = \mathsf{O}_i(tr')$ implies that (1) $\mathsf{S}_j(tr) = \mathsf{S}_j(tr')$ and (2) the trigger T_j holds in tr iff it holds in tr'; and the same has to hold symmetrically for O_j, S_i, and T_i. This means that the variation of the secret S_i as required by (P_i) is possible without interfering with the secret information of (P_j): the latter stays fixed.

The second assumption is formalized by a combined secret producing function S that does not have the familiar shape of producing an interleaving of secrets, but it produces a *pair* of secret sequences. Namely, $\mathsf{S}(t) = (\mathsf{S}_i(t), \mathsf{S}_j(t))$. This combination captures the content of both secrets, but not their scheduling. Consequently, the combined bound is defined as $\mathsf{B}((sl_i, sl_j), (sl'_i, sl'_j)) = \mathsf{B}_i(sl_i, sl'_i) \wedge \mathsf{B}_j(sl_j, sl'_j)$.

The combined observation function O is assumed to correspond to an *intersection* of the observations of (P_i) and (P_j), i.e., either $\mathsf{O}_i(t) = \mathsf{O}_i(t')$ or $\mathsf{O}_j(t) = \mathsf{O}_j(t')$ implies $\mathsf{O}(t) = \mathsf{O}(t)$. The proof of the combined security property follows easily from the assumptions: Given a trace tr with $\mathsf{S}(tr) = (sl_i, sl_j)$ and an alternative secret pair (sl'_i, sl'_j) within B, we first invoke (P_i) to obtain tr' with $\mathsf{S}_i(tr') = sl'_i$, keeping sl_j and T_j constant, and then invoke (P_j) to obtain tr'' with $\mathsf{S}_j(tr'') = sl'_j$, keeping sl_i constant such that $\mathsf{S}(tr'') = (sl'_i, sl'_j)$. The combined observation $\mathsf{O}(tr)$ remains unchanged in every step.

This proof technique is applicable to arbitrary security properties, as long as the above assumptions are satisfied (and it is straightforward to lift it from pairs to *tuples* of multiple security properties). We have instantiated it in our Isabelle formalization for the above example in CoSMeDis: two posts PID_i and PID_j in arbitrary network nodes Aut_i and Aut_j. In order to satisfy the assumptions, we first had to strengthen the observation power of the security property discussed in the paper. In addition to the actions of observing users, we declare all actions that potentially contribute to *other* secret posts to be observable. This includes updating actions of other posts, but also trigger-relevant actions, such as the creation of friends of observers. The proof in Isabelle was automatic: after extending the observation function, there were hardly any changes necessary to the existing proof scripts. The original proof strategy still worked. In order to instantiate the above combination technique, it was necessary to add the (generic) infrastructure for the technique itself and a few helper lemmas for the concrete system, but the proofs were straightforward.