Bounded-Deducibility Security

Andrei Popescu
Department of Computer Science, University of Sheffield, UK

Thomas Bauereiss
Department of Computer Science and Technology, University of Cambridge, UK

Peter Lammich
Department of Computer Science, University of Twente, The Netherlands

Abstract

We describe Bounded-Deducibility (BD) security, an expressive framework for the specification and verification of information-flow security. The framework grew by confronting concrete challenges of specifying and verifying fine-grained confidentiality properties in some realistic web-based systems. The concepts and theorems that constitute this framework have an eventful history of such "confrontations", often involving trial and error, which are reported in previous papers. This paper is the first to focus on the framework itself rather than the case studies, gathering in one place all the abstract results about BD security.

2012 ACM Subject Classification Security and privacy → Formal security models; Security and privacy → Logic and verification; Security and privacy → Security requirements

Keywords and phrases Information-flow security, Unwinding proof method, Compositionality, Verification

Digital Object Identifier 10.4230/LIPIcs.ITP.2021.3

Category Invited Paper

Funding The work presented here has been supported by: EPSRC through the grant "Verification of Web-based Systems (VOWS)" (EP/N019547/1); DFG through the grants "Security Type Systems and Deduction" (Ni 491/13-2) and "MORES – Modelling and Refinement of Security Requirements on Data and Processes (Hu 737/5-2)", part of "RS^3 – Reliably Secure Software Systems" (SPP 1496); VeTSS through the grant "Formal Verification of Information Flow Security for Relational Databases"; 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".

Acknowledgements We are fortunate to have collaborated with some excellent researchers and developers on various parts of the implementation and verification work based on BD security: Sergey Grebenshchikov, Ping Hou, Sudeep Kanav and Ondřej Kunčar have contributed to CoCon, while Armando Pesenti Gritti and Franco Raimondi have contributed to CoSMed and CoSMeDis. We thank this paper's reviewers for their helpful comments and suggestions.

1 Introduction

Bounded-Deducibility (BD) security is a framework we have developed recently for the specification and verification of information-flow security. It is applicable widely, to systems described as nondeterminisic I/O automata, and caters for the fine-grained specification of restrictions on their flows of information. We formalized the framework in the proof assistant Isabelle/HOL [31, 32] and used it in the verification of confidentiality properties of some web applications.

Information-flow security has a rich history, with many formal definitions having been proposed, differing in how systems, attackers, and flow policies are modeled [18, 26, 29, 30, 33, 34, 42–44, 46, 47]. Nevertheless, a new notion seemed necessary because the existing notions (Section 4) were not expressive enough for our case studies: multi-user web-based systems with flows of information requiring fine-grained control. For example, about a multi-user multi-conference management system, we wanted to prove a property such as the following, which refers to the series of uploads of a document's versions in a system: "A group of users learn nothing about a paper beyond the absence of any upload unless one of them becomes an author of that paper or a PC member at the paper's conference." (Importantly, the property is about not only what can be *directly accessed*, but also what can be *learned* by interacting with the system – this distinguishes information-flow control from mere access control.) Every abstract definition and theorem in the BD security framework was inspired from, and refined based on, the needs of concrete interactive systems. This ended up contributing to the area of information-flow security an increased level of precision in specification and proof, of the kind that we believe can make a difference in practical system verification.

In previous papers [7–9, 23, 37], BD security has only been discussed in the context of verifying these concrete systems. This helps with intuition and motivation, but makes it easy to miss the forest from the trees, i.e., miss the abstract level of the development. The current paper is the first to collect in one place all our abstract results, and to present them independently of any case studies (Section 2). They include the BD unwinding proof method (Section 2.5), as well as theorems on proof (Section 2.6) and system compositionality (Section 2.7). We hope that this paper will better demonstrate the scope of the framework and help identify potential new applications. The framework is open-ended and open-source [10, 35], and new contributions are welcome.

Three major verification case studies will also be briefly described while recalling their contribution to the framework's design (Section 3). These are the CoCon conference management system (Section 3.1, [23, 37]), the CoSMed social media platform (Section 3.2, [7, 9]), and the CoSMeDis distributed extension of CoSMed (Section 3.3, [8]).

Notations

We write function application by juxtaposition, without placing the argument in parentheses, as in $f\ a$, unless required for disambiguation, e.g., $f\ (g\ a)$. Multiple-argument functions will usually be considered in curried form – e.g., we think of $f : A \to B \to C$ as a two-argument function, and $f\ a\ b$ denotes its application to a and b. We write "$\circ$" for function composition. Bool denotes the two-element set of Booleans, $\{\mathsf{true}, \mathsf{false}\}$. Predicates and relations will be modeled as functions to Bool. For example, $P : A \to \mathsf{Bool}$ is a (unary) predicate on A and $Q : A \to A \to \mathsf{Bool}$ is a binary relation on A. Given $a \in A$, we write "$P\ a$ holds", or simply "$P\ a$", to mean that $P\ a = \mathsf{true}$; and similarly for binary relations.

Given a set A, we write $\mathsf{Set}(A)$ for the powerset (i.e., set of all subsets) of A, and $\mathsf{List}(A)$ for the set of lists with elements in A. We write $[a_1, \ldots, a_n]$ for the list consisting of the indicated elements; in particular, $[]$ is the empty list and $[a]$ is a singleton list. As a general convention, if a, b denote elements in A, then al, bl will denote elements in $\mathsf{List}(A)$. An exception will be the system traces – even though they are lists of transitions t, for them we will use the customized notation tr. We write "$\cdot$" for list concatenation. Applied to a non-empty list $[a_1, \ldots, a_n]$, the function head returns its first element a_1. Given a function $f : A \to B$ and $[a_1, \ldots, a_n] \in \mathsf{List}(A)$, $\mathsf{map}\ f\ [a_1, \ldots, a_n]$ returns $[f\ a_1, \ldots, f\ a_n]$. Given a partial function $f : A \rightharpoonup B$ and $[a_1, \ldots, a_n] \in \mathsf{List}(A)$, let $[a_{i_1}, a_{i_2}, \ldots, a_{i_k}]$ be the sublist of $[a_1, \ldots, a_n]$ that keeps only elements on which f is defined (where $1 \leq i_1 < i_2 < \cdots < i_k \leq n$); then $\mathsf{map}\ f\ [a_1, \ldots, a_n]$

returns $[f\ a_{i_1}, \ldots, f\ a_{i_k}]$. In other words, partial functions are mapped while omitting the elements on which they are not defined. Given a predicate P, $\mathsf{filter}\ P\ [a_1, \ldots, a_n]$ returns the sublist of $[a_1, \ldots, a_n]$ that keeps only the elements satisfying P.

2 Specification and Reasoning Framework

Our framework is developed around a simple and general notion of system: nondeterministic I/O automata. It also provides a notion of policy to describe the (dis)allowed flows of information in these systems. A policy has several parameters that regulate the tension between observations (what can be seen) and secrets (what needs to be protected). The judicious use of these parameters allows fine-tuning not only *what*, but also *how much* needs to be protected, and *when*, or even *for how long*. The framework offers methods to prove that the policies are satisfied by systems, and to manage proof and system complexity via compositionality results.

2.1 System model

The systems whose information-flow security properties will be studied are nondeterministic I/O automata. Namely, we call *system* a tuple $\mathcal{A} = (\mathsf{State}, \mathsf{Act}, \mathsf{Out}, \mathsf{istate}, \mathsf{Trans})$, where:

- State, ranged over by σ, σ' etc., is the set of states;
- Act, ranged over by a, b etc., is the set of actions;
- Out, ranged over by ou, ou' etc., is the set of outputs;
- $\mathsf{istate} \in \mathsf{State}$ is the initial state;
- $\mathsf{Trans} \subseteq \mathsf{State} \times \mathsf{Act} \times \mathsf{Out} \times \mathsf{State}$ is the set of transitions.

(Note that we call "action" what is usually called "input" for I/O automata.) A transition $t = (\sigma, a, ou, \sigma') \in \mathsf{Trans}$ has the following interpretation: If action a is taken while the system is in state σ, the system may respond by producing output ou and changing the state to σ'. We call σ the source, a the action, ou the output, and σ' the target of t. The transition's action a is also denoted by $\mathsf{actOf}\ t$. We will write $\sigma \stackrel{t}{\Longrightarrow} \sigma'$ to express that $t \in \mathsf{Trans}$, σ is the source of t and σ' is the target of t.

A *trace* is any non-empty list of transitions $[t_1, \ldots, t_n]$ such that the source of t_1 is istate and, for all $i \in \{2, \ldots, n\}$, the source of t_i is the target of t_{i-1}. We let Trace, ranged over by tr, be the set of traces. A *trace fragment* has the form $[t_i, \ldots, t_j]$ with $1 \leq i < j \leq n$, where $[t_1, \ldots, t_n]$ is a trace. We write TraceF_σ for the set of trace fragments that start in σ, i.e., have σ as the source of their first transition. Note that all these concepts are relative to a system $\mathcal{A}$. When we want to emphasize the underlying system, we may write $\mathsf{Trace}_{\mathcal{A}}$ instead of Trace, $\mathsf{TraceF}_{\mathcal{A},\sigma}$ instead of TraceF_σ, etc.

2.2 Flow policies

Given a system $\mathcal{A} = (\mathsf{State}, \mathsf{Act}, \mathsf{Out}, \mathsf{istate}, \mathsf{Trans})$, our goal is to express its information-flow security via policies that are capable of fine-grained distinctions between desirable flows (which are important for the system's functionality) and undesirable flows (which constitute information leaks possibly exploitable by attackers). To achieve such surgical precision, a policy should accurately identify the following: (1) What observations can be made on the system, (2) Which data constitute secrets that need protection, (3) How much of these secrets should be protected (and how much can be revealed), and (4) Under which conditions protection is required.

For accommodating these requirements, we define a *flow policy* $\mathcal{F}$ to consist of:

(1) an *observation infrastructure* (Obs, isObs, getObs), where
- Obs, ranged over by o, o' etc., is a chosen domain of observations,
- isObs : Trans $\rightarrow$ Bool is a predicate identifying observation-producing transitions,
- getObs : Trans $\rightarrow$ Obs is a function for producing observations from transitions;

(2) a *secrecy infrastructure* (Sec, isSec, getSec), where
- Sec, ranged over by s, s' etc., is a chosen domain of secrets,
- isSec : Trans $\rightarrow$ Bool is a predicate identifying secret-producing transitions,
- getSec : Trans $\rightarrow$ Sec is a function for producing secrets from transitions;

(3) a *declassification bound*, i.e., a relation on lists of secrets, B : List(Sec) $\rightarrow$ List(Sec) $\rightarrow$ Bool;

(4) a *declassification trigger*, i.e., a predicate on transitions, T : Trans $\rightarrow$ Bool.

Note that the observation and secrecy infrastructures have the same form. We define O : Trace $\rightarrow$ List(Obs) by O = map getObs $\circ$ filter isObs, and S : Trace $\rightarrow$ List (Sec) by S = map getSec $\circ$ filter isSec. Thus, O uses filter to select the transitions in a trace that are observable according to isObs, and then applies getObs to each selected transition. Similarly, S produces lists of secrets by filtering with isSec and applying getSec. Thus, when applied to a trace tr, O and S give the lists of observations and respectively secrets produced by tr.

2.3 Bounded-Deducibility security

For the rest of Section 2, let us fix a system $\mathcal{A}$ = (State, Act, Out, istate, Trans) and a flow policy $\mathcal{F}$, where (Obs, isObs, getObs) is its observation infrastructure, (Sec, isSec, getSec) its secrecy infrastructure, B its declassification bound and T its declassification trigger. Furthermore, let O : Trace $\rightarrow$ List(Obs) and S : Trace $\rightarrow$ List(Sec) be the functions on traces induced by these observation and secrecy infrastructures.

A system $\mathcal{A}$ is said to be *Bounded-Deducibility (BD) secure with respect to the flow policy* $\mathcal{F}$, written $\mathcal{A} \models \mathcal{F}$, provided that for all $tr_1 \in$ Trace and $sl_1, sl_2 \in$ List(Sec),
- if never T tr_1, S $tr_1 = sl_1$ and B sl_1 sl_2,
- then there exists $tr_2 \in$ Trace such that O tr_2 = O tr_1 and S $tr_2 = sl_2$.

The predicate never T tr_1 says that T holds for no transition in tr_1.

Here is how to interpret the above definition: tr_1 is a trace that occurs when running the system, and sl_1 is the list of secrets that it produces. BD security says that, if the trigger T is never fired during tr_1, it is impossible for an observer (potential attacker) to distinguish tr_1 from any other trace tr_2 that produces some secrets sl_2 that are B-related to (i.e., located within bound B from) sl_1. Hence, for all the observer knows (via the observation function O), the trace tr_1 might as well have been tr_2.

When referring to the items in this definition, we will call tr_1 "the original trace" and tr_2 "the alternative trace". We will also apply the qualifiers "original" and "alternative" to the produced lists of observations and secrets. Note that BD security is a $\forall\exists$-statement: quantified universally over the original trace tr_1 and the alternative secrets sl_2, and then existentially over the alternative trace tr_2. (The universal quantification over sl_1 is done only for clarity; it can be avoided, since sl_1 = S tr_1.)

We can think of B negatively, as a lower bound for uncertainty, or positively, as an upper bound for the amount of information release, also known as *declassification*. For example, if B is an equivalence, then the observers learn the equivalence class of the secret, but nothing more. On the other hand, T is a trigger removing the bound B: As soon as T becomes true, the containment of declassification is no longer guaranteed. In summary, BD security says: An observer O cannot learn about the secrets anything *beyond* B *unless* T occurs.

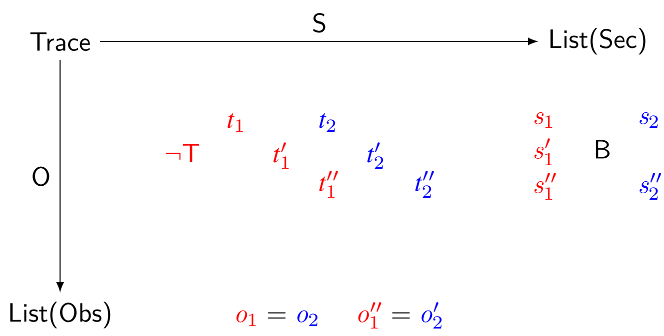

Figure 1 BD security illustrated.

Fig. 1 contains a visual illustration of BD security's two-dimensional nature: The system traces (displayed on the top left corner) produce observations (on the bottom left), as well as secrets (on the top right). The figure also includes an abstract example of traces and their observation and secret projections. The original trace tr_1 consists of three transitions, $tr_1 = [t_1, t_1', t_1'']$, of which all produce secrets, $[s_1, s_1', s_1'']$, and only the first and the third produce observations, $[o_1, o_1'']$ – all these are depicted in red. The alternative trace tr_2 also consists of three transitions, $tr_2 = [t_2, t_2', t_2'']$, of which the first and the third produce secrets, $[s_2, s_2'']$, and the first two produce observations, $[o_2, o_2']$ – all these are depicted in blue. Thus, the figure's functions O and S are given by filters and producers behaving as follows:

	isObs	getObs	isSec	getSec
t_1	true	o_1	true	s_1
t_1'	false		true	s_1'
t_1''	true	o_1''	true	s_1''

	isObs	getObs	isSec	getSec
t_2	true	o_2	true	s_2
t_2'	true	o_2'	false	
t_2''	false		true	s_2''

The empty slots in the tables correspond to values of getObs and getSec that are irrelevant, since the corresponding values of isObs and isSec are false. The ∀∃ statement expressing BD security is illustrated on the figure by making a choice of the ∀-quantified entities and the ∃-quantified entities: Given the original trace, here $[t_1, t_1', t_1'']$ (which produces the shown observations and secrets and has all its transitions satisfying ¬T) and given some alternative secrets, here $[s_2, s_2'']$, located within the bound B of the original secrets, BD security requires the existence of the alternative trace, here $[t_2, t_2', t_2'']$, producing the same observations and producing the alternative secrets.

2.4 From nondeducibility to bounded deducibility

BD security is a natural evolution of the idea of *nondeducibility* introduced in pioneering work by Sutherland [46]: by refining the notion of "nothing being deducible" to that of "nothing being deducible beyond a certain bound and unless a certain trigger occurs".

Indeed, nondeducibility can be expressed in terms of operators $\mathsf{O} : \mathsf{Trace} \to \mathsf{List(Obs)}$ and $\mathsf{S} : \mathsf{Trace} \to \mathsf{List(Sec)}$ by requiring that, for all $tr_1 \in \mathsf{Trace}$ and $sl_1, sl_2 \in \mathsf{List(Sec)}$, if $\mathsf{S}\ tr_1 = sl_1$ then there exists $tr_2 \in \mathsf{Trace}$ such that $\mathsf{O}\ tr_2 = \mathsf{O}\ tr_1$ and $\mathsf{S}\ tr_2 = sl_2$. Thus, BD security becomes nondeducibility when B is everywhere true and T everywhere false – meaning no declassification, i.e., maximum uncertainty.

2.5 Unwinding proof method

To prove that the system is BD secure with respect to the flow policy, $\mathcal{A} \models \mathcal{F}$, one needs to do the following: Given

- the original trace tr_1 for which never T holds and which produces the list of secrets sl_1,
- and an alternative list of secrets sl_2 such that B sl_1 sl_2 holds,

one should provide an alternative trace tr_2 whose produced list of secrets is exactly sl_2 and whose produced list of observations is the same as that of tr_1.

Following the tradition of unwinding for noninterference-like properties [19,26,41], we want to construct tr_2 from tr_1 incrementally: As tr_1 grows, tr_2 should grow nearly synchronously. Unwindings are traditionally binary relations Δ on State that bookkeep the states reached by tr_1 and tr_2, say σ_1 and σ_2, and show how these can evolve transition by transition in the process of constructing tr_2 from tr_1; they guarantee that any Δ-related states σ_1 and σ_2 evolve via transitions $\sigma_1 \xRightarrow{t_1} \sigma_1'$ and $\sigma_2 \xRightarrow{t_2} \sigma_2'$ to Δ-related states σ_1' and σ_2'. In our case, unlike in the traditional case, we have a significantly more complex infrastructure to deal with: Since the produced observations of tr_1 and tr_2 will have to be equal, it is reasonable to track them synchronously; but the produced secrets are regulated by arbitrary bounds B, hence they will have to track them more flexibly.

To address the above, an unwinding for BD security will be not just a binary relation between states, but a binary relation between pairs consisting of a state and a list of secrets. Let us introduce some convenient notation to describe this. For any pairs (σ, sl) and (σ', sl') in $\mathsf{State} \times \mathsf{List}(\mathsf{Sec})$ and any transition t, we will write $(\sigma, sl) \xRightarrow{t} (\sigma', sl')$ as a shorthand for the following two statements: (1) $\sigma \xRightarrow{t} \sigma'$, and (2) either $\neg\,\mathsf{isSec}\ t$ and $sl' = sl$, or $\mathsf{isSec}\ t$ and there exists s such that $sl = [s] \cdot sl'$. The second statement means that the transition t either does not produce a secret thus leaving sl unchanged ($sl' = sl$), or produces the secret from the beginning of sl thus reducing it to sl'; we can think of this as a transition between lists of secrets that are still to be produced. Moreover, for any two transitions t_1 and t_2, we will write $t_1 =_{\mathsf{Obs}} t_2$ as a shorthand for the following two statements: (1) $\mathsf{isObs}\ t_1$ if and only if $\mathsf{isObs}\ t_2$, and (2) if $\mathsf{isObs}\ t_1$ then $\mathsf{getObs}\ t_1 = \mathsf{getObs}\ t_2$. In other words, t_1 and t_2 produce either the same observation or no observation.

A relation $\Delta : (\mathsf{State} \times \mathsf{List}(\mathsf{Sec})) \to (\mathsf{State} \times \mathsf{List}(\mathsf{Sec})) \to \mathsf{Bool}$ is said to be a *BD unwinding* if, for all (σ_1, sl_1), $(\sigma_2, sl_2) \in \mathsf{State} \times \mathsf{List}(\mathsf{Sec})$ such that σ_1 is $(\neg\mathsf{T})$-reachable, σ_2 is reachable and $\Delta\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$, we have that one of the following three cases holds:

(1) $sl_1 \neq []$ or $sl_2 = []$, and $\mathsf{reaction}\ \Delta\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$; or

(2) $\mathsf{iaction}\ \Delta\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$; or

(3) $sl_1 \neq []$ and $\mathsf{exit}\ \sigma_1\ (\mathsf{head}\ sl_1)$.

Above, a state being reachable means that there exists a trace tr leading to it; and $(\neg\mathsf{T})$-reachability additionally requires that all transitions in tr satisfy $\neg\mathsf{T}$.

The predicates reaction, iaction (read "independent action") and exit will be defined below. The first two describe possible evolution patterns for the pairs (σ_1, sl_1) and (σ_2, sl_2) so that the result is still in Δ. By contrast, the exit predicate provides a shortcut for an early finish during a proof by unwinding. When reading the definitions of these predicates, the reader should keep in mind what we want from a BD unwinding: to manage the incremental growth of an alternative trace (that has currently reached state σ_2), in response to the growth of an original trace (that has currently reached state σ_1), while considering the list of secrets sl_1 that the remainder of the original trace *is assumed to produce* and the list of secrets sl_2 that the remainder of the alternative trace *will have to produce*.

$\mathsf{reaction}\ \Delta\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$ is defined to mean that, for all $t_1 \in \mathsf{Trans}$ and $(\sigma'_1, sl'_1) \in \mathsf{State} \times \mathsf{List}(\mathsf{Sec})$ such that $(\sigma_1, sl_1) \overset{t_1}{\Longrightarrow} (\sigma'_1, sl'_1)$, one of the following two cases holds:

(1) $\neg\ \mathsf{isObs}\ t_1$ and $\Delta\ (\sigma'_1, sl'_1)\ (\sigma_2, sl_2)$; or

(2) there exist $t_2 \in \mathsf{Trans}$ and $(\sigma'_2, sl'_2) \in \mathsf{State} \times \mathsf{List}(\mathsf{Sec})$ such that $(\sigma_2, sl_2) \overset{t_2}{\Longrightarrow} (\sigma'_2, sl'_2)$, $t_1 =_{\mathsf{Obs}} t_2$ and $\Delta\ (\sigma'_1, sl'_1)\ (\sigma'_2, sl'_2)$.

Thus, $\mathsf{reaction}\ \Delta\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$ describes two ways in which one can "react" to a transition t_1 taken by the original trace: (1) either ignoring it (if it is unobservable), or (2) matching it with a transition t_2 of the alternative trace. In both cases, we must stay in Δ.

$\mathsf{iaction}\ \Delta\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$ is defined to mean that there exist $t_2 \in \mathsf{Trans}$ and $(\sigma'_2, sl'_2) \in \mathsf{State} \times \mathsf{List}(\mathsf{Sec})$ such that $(\sigma_2, sl_2) \overset{t_2}{\Longrightarrow} (\sigma'_2, sl'_2)$, $\neg\ \mathsf{isObs}\ t_2$, $\mathsf{isSec}\ t_2$ and $\Delta\ (\sigma_1, sl_1)\ (\sigma'_2, sl'_2)$.

Thus, $\mathsf{iaction}$ describes the possibility of an "independent" (i.e., non-reactive) action by taking an unobservable secret-producing transition in the alternative trace. While the unobservability requirement ($\neg\ \mathsf{isObs}\ t_2$) is justified by the desire to keep the observations synchronized, the reason for the secret-producing requirement ($\mathsf{isSec}\ t_2$) is more subtle: Repeating unobservable *and non-secret-producing* independent actions could indefinitely delay the growth of the original trace while making no progress with the alternative list of secrets, rendering unwinding reasoning unsound.

$\mathsf{exit}\ \sigma\ s$ is defined to mean that, for all states σ' that are $(\neg\mathsf{T})$-reachable from σ and all transitions t with source σ' such that $\neg\ \mathsf{T}\ t$, if $\mathsf{isSec}\ t$ then $\mathsf{getSec}\ t \neq s$.

The idea behind exit is that BD security holds trivially for original traces that are unable to produce their due list of secrets sl_1; and exit detects this (thus closing that branch of the unwinding proof) by noticing that not even the first secret in sl_1 can be produced starting from the current state σ_1 – indeed, in the definition of unwinding, exit is invoked with σ_1 and $\mathsf{head}\ sl_1$.

Left unexplained so far are the (non)emptiness conditions guarding the invocations of the $\mathsf{reaction}$ and exit predicates in the definition of BD unwinding. For exit, it is obvious that we need $sl_1 \neq []$ for talking about the first element in sl_1. But for $\mathsf{reaction}$, why require that $sl_1 \neq []$ or $sl_2 = []$? Again, this decision has to do with the soundness of BD unwinding as a proof method: If the negation of this condition is true, it means that the original trace is done with producing its secrets ($sl_1 = []$) and the alternative trace still has some secrets to produce ($sl_2 \neq []$). In that case, we want to enforce an $\mathsf{iaction}$ move which, being secret-producing, would make progress through the remaining alternative list of secrets sl_2; this is achieved by preventing a $\mathsf{reaction}$ move, which would be the only alternative (since an exit move needs $sl_1 \neq []$). With these definitions, BD unwinding fulfills its goal:

▶ **Lemma 1.** [23, 37] Assume Δ is a BD unwinding and let $\sigma_1, \sigma_2 \in \mathsf{State}$ such that $\mathsf{reach}_{\neg\mathsf{T}}\ \sigma_1$ and $\mathsf{reach}\ \sigma_2$. Then, for all $tr_1 \in \mathsf{TraceF}_{\sigma_1}$ and $sl_1, sl_2 \in \mathsf{List}(\mathsf{Sec})$,

- if $\mathsf{never}\ \mathsf{T}\ tr_1$, $\mathsf{S}\ tr_1 = sl_1$ and $\Delta\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$,
- then there exists $tr_2 \in \mathsf{TraceF}_{\sigma_2}$ such that $\mathsf{O}\ tr_2 = \mathsf{O}\ tr_1$ and $\mathsf{S}\ tr_2 = sl_2$.

In other words, assuming $\Delta\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$ holds and given the remaining part tr_1 of the original trace (starting in σ_1) which produces secrets sl_1, there exists a trace tr_2 that produces the same observations and produces the desired secrets sl_2. The lemma's proof goes by induction on the sum of the lengths of tr_1 and sl_2. The induction step either reaches a contradiction (if exit is invoked), or consumes a transition from tr_1 (if $\mathsf{reaction}$ is invoked) or a secret from sl_2 (if $\mathsf{iaction}$ is invoked).

To connect this result to BD security, in particular to factor in the bound B as well, we additionally require that a BD unwinding Δ includes the bound B in the initial state. So we can think of Δ as generalizing and strengthening the bound, and then maintaining it all the way to the successful production of the alternative trace required by BD security. We are closing in on the main result about BD unwinding, a consequence of the lemma taking $\sigma_1 = \sigma_2 = \mathsf{istate}$. It states that BD unwinding is a *sound proof method* for BD security.

▶ **Theorem 2.** (Unwinding Theorem [23,37]) Assume that the following hold:
(1) For all $sl_1, sl_2 \in \mathsf{List(Sec)}$, if $\mathsf{B}\ sl_1\ sl_2$ then $\Delta\ (\mathsf{istate}, sl_1)\ (\mathsf{istate}, sl_2)$.
(2) Δ is a BD unwinding.
Then $\mathcal{A} \models \mathcal{F}$.

According to this theorem, to prove BD security of a system, it suffices to define a relation Δ and show that (1) it includes the bound B in the initial state and (2) it is a BD unwinding.

2.6 Proof compositionality

When verifying a BD security policy for a large system, defining a single monolithic BD unwinding could be daunting. We can alleviate this by working not with a single unwinding relation, but with a network of relations, such that any relation may "unwind" into any number of relations in the network.

To this end, we refine the notion of BD unwinding. Given a relation Δ and a set of relations $\Delta\mathsf{s}$, Δ is a said to be a *BD unwinding into* $\Delta\mathsf{s}$ if it satisfies the same conditions as in the definition of BD unwinding, just that $\mathsf{iaction}\ \Delta$ and $\mathsf{reaction}\ \Delta$ are replaced by $\mathsf{iaction}\ (\bigvee \Delta\mathsf{s})$ and $\mathsf{reaction}\ (\bigvee \Delta\mathsf{s})$, where $\bigvee \Delta\mathsf{s}$ is the disjunction (i.e., union) of all the relations in $\Delta\mathsf{s}$. Namely, for all (σ_1, sl_1), $(\sigma_2, sl_2) \in \mathsf{State} \times \mathsf{List(Sec)}$ such that σ_1 is $(\neg\mathsf{T})$-reachable, σ_2 is reachable and $\Delta\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$, one of the following three cases holds:
(1) $sl_1 \neq []$ or $sl_2 = []$, and $\mathsf{reaction}\ (\bigvee \Delta\mathsf{s})\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$; or
(2) $\mathsf{iaction}\ (\bigvee \Delta\mathsf{s})\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$; or
(3) $sl_1 \neq []$ and $\mathsf{exit}\ \sigma_1\ (\mathsf{head}\ sl_1)$.

This enables a form of sound compositional reasoning: If we verify a condition as above for each component relation, we obtain an overall secure system.

▶ **Theorem 3.** (Multiplex Unwinding Theorem [37]) Let $\Delta\mathsf{s}$ be a set of relations. For each $\Delta \in \Delta\mathsf{s}$, let $\mathsf{next}_\Delta \subseteq \Delta\mathsf{s}$ be a (possibly empty) set of "successors" of Δ, and let $\Delta_{\mathsf{init}} \in \Delta\mathsf{s}$ be a chosen "initial" relation. Assume the following hold:
(1) For all $sl_1, sl_2 \in \mathsf{List(Sec)}$, if $\mathsf{B}\ sl_1\ sl_2$ then $\Delta_{\mathsf{init}}\ (\mathsf{istate}, sl_1)\ (\mathsf{istate}, sl_2)$.
(2) Each $\Delta \in \Delta\mathsf{s}$ is a BD unwinding into next_Δ.
Then $\mathcal{A} \models \mathcal{F}$.

The network of components can form any directed graph − Fig. 2 shows an example. However, when doing concrete proofs by unwinding, we found that the following essentially linear network often suffices (Fig. 3): Each Δ_i unwinds either into itself, or into Δ_{i+1} (if $i \neq n$), or into an exit component Δ_{e} that always chooses the "exit" unwinding condition. (In practice, Δ_{e} will collect "error" situations that break invariants, hence preventing the original trace from producing its due secrets.) To express this, we define the notion of Δ being a *BD continuation-unwinding into* $\Delta\mathsf{s}$ similarly to that of "BD unwinding into" but excluding the exit case, i.e., requiring that either (1) $sl_1 \neq []$ or $sl_2 = []$, and $\mathsf{reaction}\ (\bigvee \Delta\mathsf{s})\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$, or (2) $\mathsf{iaction}\ (\bigvee \Delta\mathsf{s})\ (\sigma_1, sl_1)\ (\sigma_2, sl_2)$ hold. And Δ is said to be a *BD exit-unwinding* if the exit case, (3) $sl_1 \neq []$ and $\mathsf{exit}\ \sigma_1\ (\mathsf{head}\ sl_1)$, holds. We obtain:

▶ **Theorem 4.** (Sequential Multiplex Unwinding Theorem [37]) Consider the indexed set of relations $\{\Delta_1, \ldots, \Delta_n\}$ and the relation Δ_{e} such that the following hold:

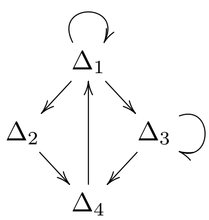

Figure 2 A network of unwinding components.

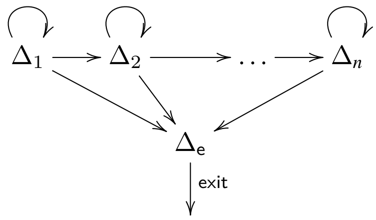

Figure 3 A linear network with exit.

(1) For all $sl_1, sl_2 \in \mathsf{List}(\mathsf{Sec})$, if $\mathsf{B}\ sl_1\ sl_2$ then $\Delta_1\ (\mathsf{istate}, sl_1)\ (\mathsf{istate}, sl_2)$.
(2) Δ_i is a BD continuation-unwinding into $\{\Delta_i, \Delta_{i+1}, \Delta_{\mathsf{e}}\}$.
(3) Δ_{e} is a BD exit-unwinding.
Then $\mathcal{A} \models \mathcal{F}$.

Although the Multiplex Unwinding Theorems are easy consequences of the (plain) Unwinding Theorem, we found them to be very useful tools for managing proof complexity.

2.7 System compositionality

A complexity management desideratum equally important to proof compositionality is system compositionality: the possibility to infer BD security for a compound system from BD security of the components. Next, we will describe a compositionality result for a communicating network of systems. We start with two, then we generalize to n systems.

2.7.1 Product systems

Let $\mathcal{A}_1 = (\mathsf{State}_1, \mathsf{Act}_1, \mathsf{Out}_1, \mathsf{istate}_1, \mathsf{Trans}_1)$ and $\mathcal{A}_2 = (\mathsf{State}_2, \mathsf{Act}_2, \mathsf{Out}_2, \mathsf{istate}_2, \mathsf{Trans}_2)$ be two systems. We want to model communication between $\mathcal{A}_1$ and $\mathcal{A}_2$ by matching certain transitions that these systems must take synchronously while exchanging data. This is captured by a relation $\mathsf{match} : \mathsf{Trans}_1 \to \mathsf{Trans}_2 \to \mathsf{Bool}$. Transition matching gives a very flexible communication scheme: It can model message-passing communication using the transitions' actions and outputs, but also shared-state communication using the transitions' source and target states.

We will distinguish between separate (local) component actions and communication actions. We write $\mathsf{isCom}_i\ a$ (for $i \in \{1, 2\}$) to indicate that an action a is in the latter category for $\mathcal{A}_i$. Namely, $\mathsf{isCom}_i\ a$ holds whenever there exist t_1 and t_2 such that $\mathsf{match}\ t_1\ t_2$ holds and a is the action of t_i.

We define the match-*communicating product of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$, written $\mathcal{A}_1 \times^{\mathsf{match}} \mathcal{A}_2$, as the following system $(\mathsf{State}, \mathsf{Act}, \mathsf{Out}, \mathsf{istate}, \mathsf{Trans})$:

- $\mathsf{State} = \mathsf{State}_1 \times \mathsf{State}_2$;
- $\mathsf{Act} = \mathsf{Act}_1 + \mathsf{Act}_2 + \mathsf{Act}_1 \times \mathsf{Act}_2$; thus, Act is 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 write $(1, a_1)$, $(2, a_2)$, and (a_1, a_2) for actions of the first, second and third kind, respectively;
- $\mathsf{Out} = \mathsf{Out}_1 + \mathsf{Out}_2 + \mathsf{Out}_1 \times \mathsf{Out}_2$; thus, like Act, Out is a disjoint union, and we use similar notations for its elements: $(1, ou_1)$, $(2, ou_2)$ and (ou_1, ou_2);
- $\mathsf{istate} = (\mathsf{istate}_1, \mathsf{istate}_2)$;
- Trans contains three kinds of transitions:

- separate $\mathcal{A}_1$-transitions $((\sigma_1, \sigma_2), (1, a_1), (1, ou_1), (\sigma_1', \sigma_2))$, where $(\sigma_1, a_1, ou_1, \sigma_1') \in \mathsf{Trans}_1$ and $\neg\ \mathsf{isCom}_1\ a_1$;
- separate $\mathcal{A}_2$-transitions $((\sigma_1, \sigma_2), (2, a_2), (2, ou_2), (\sigma_1, \sigma_2'))$, where $(\sigma_2, a_2, ou_2, \sigma_2') \in \mathsf{Trans}_2$ and $\neg\ \mathsf{isCom}_2\ a_2$;
- communication transitions $((\sigma_1, \sigma_2), (a_1, a_2), (ou_1, ou_2), (\sigma_1', \sigma_2'))$, where $(\sigma_1, a_1, ou_1, \sigma_1') \in \mathsf{Trans}_1$, $(\sigma_2, a_2, ou_2, \sigma_2') \in \mathsf{Trans}_2$ and $\mathsf{match}\ (\sigma_1, a_1, ou_1, \sigma_1')\ (\sigma_2, a_2, ou_2, \sigma_2')$.

Thus, a transition t of $\mathcal{A}_1 \times^{\mathsf{match}} \mathcal{A}_2$ has exactly one of the following three forms shown above. In the first case, t is completely determined by an $\mathcal{A}_1$-transition $t_1 = (\sigma_1, a_1, ou_1, \sigma_1')$ and an $\mathcal{A}_2$-state σ_2 – we write $t = \mathsf{sep}_1\ t_1\ \sigma_2$, marking that t is given by the separate transition t_1. Similarly, in the second case we write $t = \mathsf{sep}_2\ \sigma_1\ t_2$, where $t_2 = (\sigma_2, a_2, ou_2, \sigma_2')$. In the third case, we write $t = \mathsf{com}\ t_1\ t_2$, marking that t proceeds as a communication transition. Thus, in our new notation, any transition of $\mathcal{A}_1 \times^{\mathsf{match}} \mathcal{A}_2$ has either the form $\mathsf{sep}_1\ t_1\ \sigma_2$, or $\mathsf{sep}_2\ \sigma_1\ t_2$, or $\mathsf{com}(t_1, t_2)$.

2.7.2 Product flow policies

Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be flow policies for $\mathcal{A}_1$ and $\mathcal{A}_2$. Given $i \in \{1, 2\}$, we write $(\mathsf{Obs}_i, \mathsf{isObs}_i, \mathsf{getObs}_i)$ for the observation infrastructure, $(\mathsf{Sec}_i, \mathsf{isSec}_i, \mathsf{getSec}_i)$ for the secrecy infrastructure, B_i for the declassification bound and T_i for the declassification trigger of $\mathcal{F}_i$. We want to compose the policies $\mathcal{F}_1$ and $\mathcal{F}_2$ in a natural way, forming a policy for the product $\mathcal{A}_1 \times^{\mathsf{match}} \mathcal{A}_2$. To achieve this, we need observation and secret counterparts of the transition-matching predicate match, in the form of predicates $\mathsf{matchO} : \mathsf{Obs}_1 \to \mathsf{Obs}_2 \to \mathsf{bool}$ and $\mathsf{matchS} : \mathsf{Sec}_1 \to \mathsf{Sec}_2 \to \mathsf{bool}$. Triples $(\mathsf{match}, \mathsf{matchO}, \mathsf{matchS})$ will be called *communication infrastructures*.

A sanity property that we will assume about our communication infrastructures is that its matching operators are compatible with (i.e., preserved by) the secrecy and observation infrastructure operators.

Compatible Communication: For all $t_1 \in \mathsf{Trans}_1$ and $t_2 \in \mathsf{Trans}_2$, if $\mathsf{match}\ t_1\ t_2$ then:
- $\mathsf{isSec}_1\ t_1$ if and only if $\mathsf{isSec}_2\ t_2$, and in this case we have $\mathsf{matchS}\ (\mathsf{getSec}_1\ t_1)\ (\mathsf{getSec}_2\ t_2)$;
- $\mathsf{isObs}_1\ t_1$ if and only if $\mathsf{isObs}_2\ t_2$, and in this case we have $\mathsf{matchO}\ (\mathsf{getObs}_1\ t_1)\ (\mathsf{getObs}_2\ t_2)$.

The *product of $\mathcal{F}_1$ and $\mathcal{F}_2$ along a communication infrastructure* $(\mathsf{match}, \mathsf{matchO}, \mathsf{matchS})$, written $\mathcal{F}_1 \times^{(\mathsf{match},\mathsf{matchO},\mathsf{matchS})} \mathcal{F}_2$, is defined as the following flow policy for $\mathcal{A}_1 \times^{\mathsf{match}} \mathcal{A}_2$.

We start with its observation and secrecy infrastructures, which are naturally defined considering that observations and secrets can be produced either separately or in communication steps. The observation infrastructure $(\mathsf{Obs}, \mathsf{isObs}, \mathsf{getObs})$ is the following:
- $\mathsf{Obs}_1 + \mathsf{Obs}_2 + \mathsf{Obs}_1 \times \mathsf{Obs}_2$; thus, 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 any $t \in \mathsf{Trans}$, $\mathsf{isObs}\ t$ and $\mathsf{getObs}\ t$ are defined as follows:
  - if t has the form $\mathsf{sep}_1\ t_1\ \sigma_2$, then $\mathsf{isObs}\ t = \mathsf{isObs}_1\ t_1$ and $\mathsf{getObs}\ t = (1, \mathsf{getObs}_1\ t_1)$;
  - if t has the form $\mathsf{sep}_2\ \sigma_1\ t_2$, then $\mathsf{isObs}\ t = \mathsf{isObs}_2\ t_2$ and $\mathsf{getObs}\ t = (2, \mathsf{getObs}_2\ t_2)$;
  - if t has the form $\mathsf{com}\ t_1\ t_2$, then $\mathsf{isObs}\ t = (\mathsf{isObs}_1\ t_1$ and $\mathsf{isObs}_2\ t_2)$ and $\mathsf{getObs}\ t = (\mathsf{getObs}_1\ t_1, \mathsf{getObs}_2\ t_2)$.

One could argue that, when t has the form $\mathsf{com}\ t_1\ t_2$, $\mathsf{isObs}\ t$ should be defined not as (1) $\mathsf{isObs}_1\ t_1$ and $\mathsf{isObs}_2\ t_2$, but as (2) $\mathsf{isObs}_1\ t_1$ or $\mathsf{isObs}_2\ t_2$, thus making the compound transition observable if either component transition is observable. However, we will only work under the assumption of Compatible Communication (introduced above), which makes (1) and (2) equivalent.

$$\textsc{Sep}_1\,\frac{sl \in sl_1 \times^{\mathsf{matchS}} sl_2 \qquad \neg\, \mathsf{isComS}_1\ s_1}{sl \cdot [(1, s_1)] \ \in\ (sl_1 \cdot [s_1]) \times^{\mathsf{matchS}} sl_2} \qquad \textsc{Sep}_2\,\frac{sl \in sl_1 \times^{\mathsf{matchS}} sl_2 \qquad \neg\, \mathsf{isComS}_2\ s_2}{sl \cdot [(2, s_2)] \ \in\ sl_1 \times^{\mathsf{matchS}} (sl_2 \cdot [s_2])}$$

$$\textsc{Empty}\,\frac{\cdot}{[] \in [] \times^{\mathsf{matchS}} []} \qquad \textsc{Com}\,\frac{sl \in sl_1 \times^{\mathsf{matchS}} sl_2 \qquad \mathsf{matchS}\ s_1\ s_2}{sl \cdot [(s_1, s_2)] \ \in\ (sl_1 \cdot [s_1]) \times^{\mathsf{matchS}} (sl_2 \cdot [s_2])}$$

Figure 4 Shuffle product for lists of secrets.

The secrecy infrastructure $(\mathsf{Sec}, \mathsf{isSec}, \mathsf{getSec})$ is defined similarly to the observation infrastructure: 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.

The trigger T of the product flow policy is also the natural one: Any firing of the trigger on either side, either separately or during communication, will fire the composite trigger. Formally, we take $\mathsf{T}\ t$ to mean the following: (1) if t has the form $\mathsf{sep}_1\ t_1\ \sigma_2$, then $\mathsf{T}_1\ t_1$ holds; (2) if t has the form $\mathsf{sep}_2\ \sigma_1\ t_2$, then $\mathsf{T}_2\ t_2$ holds; (3) if t has the form $\mathsf{com}\ t_1\ t_2$, then $\mathsf{T}_1\ t_1$ holds or $\mathsf{T}_2\ t_2$ holds.

It remains to define the bound B of the product flow policy. Let $sl \in \mathsf{List}(\mathsf{Sec})$ be a list of secrets in the composite secret domain. Intuitively, the most restrictive bound B we can hope for will forbid the declassification, for any lists of secrets $sl_1 \in \mathsf{List}(\mathsf{Sec}_1)$ and $sl_2 \in \mathsf{List}(\mathsf{Sec}_2)$ into which sl can be decomposed (i.e., which can be combined to make up sl), of anything beyond what can be declassified about sl_1 and sl_2 within the components' bounds B_1 and B_2.

To capture this, we collect all valid ways of combining sl_1 and sl_2, via the matchS-shuffle product operator $\times^{\mathsf{matchS}} : \mathsf{List}(\mathsf{Sec}_1) \to \mathsf{List}(\mathsf{Sec}_2) \to \mathsf{Set}(\mathsf{List}(\mathsf{Sec}))$ whose inductive definition is shown in Fig. 4. The set $sl_1 \times^{\mathsf{matchS}} sl_2$ contains all possible interleavings of sl_1 and sl_2, achieved by separate individual steps (rule $\textsc{Sep}_1$ and $\textsc{Sep}_2$) and communication steps (rule $\textsc{Com}$). For $i \in \{1, 2\}$, $\mathsf{isComS}_i\ s$ is the secret counterpart of the predicate isCom_i, expressing that the secret s participates in a matchS-relationship. We define $\mathsf{B}\ sl\ sl'$ to mean that, for all $sl_1, sl'_1 \in \mathsf{List}(\mathsf{Sec}_1)$ and $sl_2, sl'_2 \in \mathsf{List}(\mathsf{Sec}_2)$, if $sl \in sl_1 \times^{\mathsf{matchS}} sl_2$ and $sl' \in sl'_1 \times^{\mathsf{matchS}} sl'_2$, then $\mathsf{B}_1(sl_1, sl'_1)$ and $\mathsf{B}_2(sl_2, sl'_2)$ hold.

2.7.3 Compositionality result

We next introduce some properties that refer to the flow policies $\mathcal{F}_1$ and $\mathcal{F}_2$ and the communication infrastructure $(\mathsf{match}, \mathsf{matchO}, \mathsf{matchS})$. Together with Compatible Communication, they will be sufficient for compositionality.

Strong Communication: For all $t_1 \in \mathsf{Trans}_1$ and $t_2 \in \mathsf{Trans}_2$, if the following hold:

- $\mathsf{isCom}_1(\mathsf{actOf}_1\ t_1)$ and $\mathsf{isCom}_2\ (\mathsf{actOf}_2\ t_2)$,
- $\mathsf{isObs}_1\ t_1$, $\mathsf{isObs}_2\ t_2$ and $\mathsf{matchO}\ (\mathsf{getObs}_1\ t_1)\ (\mathsf{getObs}_2\ t_2)$,
- $\mathsf{isSec}_1\ t_1$ and $\mathsf{isSec}_2\ t_2$ imply $\mathsf{matchS}\ (\mathsf{getSec}_1\ t_1)\ (\mathsf{getSec}_2\ t_2)$,

then $\mathsf{match}\ t_1\ t_2$ holds.

The property says that, for observable communicating transitions, observation matching together with secret matching (the latter conditional on secrecy) causes the matching of the entire transitions.

Observable Communication: For all $t_1 \in \mathsf{Trans}_1$, $\mathsf{isCom}_1\ (\mathsf{actOf}_1\ t_1)$ implies $\mathsf{isObs}_1\ t_1$; and for all $t_2 \in \mathsf{Trans}_2$, $\mathsf{isCom}_2\ (\mathsf{actOf}_2\ t_2)$ implies $\mathsf{isObs}_2\ t_2$.

The property says that all communicating transitions are observable (i.e., isObs is true for them), although it does not say anything about what can actually be observed about them (via getObs).

Secret Polarization: For all $t_2 \in \mathsf{Trans}_2$, $\mathsf{isSec}_2\ t_2$ implies $\mathsf{isCom}_2\ (\mathsf{actOf}_2\ t_2)$.

The property says that any $\mathcal{A}_2$-transition that is secret-producing must be a communicating transition, which means that only $\mathcal{A}_1$ is able to produce secrets independently.

We are now ready to state our system compositionality result about BD security:

▶ **Theorem 5.** (System Compositionality Theorem [8]) Assume that the flow policies $\mathcal{F}_1$ and $\mathcal{F}_2$ and the communication infrastructure $(\mathsf{match}, \mathsf{matchO}, \mathsf{matchS})$ satisfy all the above properties, namely Compatible, Strong and Observable Communication, and Secret Polarization. Moreover, assume $\mathcal{A}_1 \models \mathcal{F}_1$ and $\mathcal{A}_2 \models \mathcal{F}_2$. Then $\mathcal{A}_1 \times^{\mathsf{match}} \mathcal{A}_2 \ \models\ \mathcal{F}_1 \times^{(\mathsf{match},\mathsf{matchO},\mathsf{matchS})} \mathcal{F}_2$.

In [8], we discuss in great detail this theorem's assumptions in the context of verifying a concrete distributed system. The main strength of the theorem is that it allows composing general bounds and triggers. For this to work, we put restrictions on the observation and secrecy infrastructures. Among these, Compatible Communication seems to occur naturally in communicating systems – at least in our case studies of interest, which are multi-user web-based systems. When targeting such systems, Strong and Observable Communication seem to be achievable for a given desired policy via a uniform process of strengthening the observation and secrecy infrastructures: allowing one to observe as much non-sensitive information as possible, and making minor adjustments to the bounds and triggers to accommodate the additional harmless information unblocked [8, App. B].

On the other hand, Secret Polarization is the major limitation of the theorem.[1] For multi-user systems, this means that, for the notion of secret defined by the flow policies $\mathcal{F}_1$ and $\mathcal{F}_2$, only users of one of the two component systems, $\mathcal{A}_1$, can be allowed to upload secrets. However, this does not prevent us from considering another notion of secret, where the other component is the issuer, as part of a different pair of flow policies $\mathcal{F}_1'$ and $\mathcal{F}_2'$.[2]

Finally, an inconvenience of applying the theorem is the somewhat artificial nature of the composite bound. While by design the composite bound is as restrictive as possible (which is good for accuracy), in practice we would prefer a less restrictive but more readable bound, referring to secrets of a simpler nature than the composite secrets. To obtain this, we can perform an adjustment using a general-purpose theorem that transports a BD security property between different observation and secret domains, possibly loosening the bound and weakening the trigger, i.e., overall weakening the flow policy.

This works as follows. Let $\mathcal{F}$ and $\mathcal{F}'$ be two flow policies for a system $\mathcal{A}$, where we write $(\mathsf{Obs}, \mathsf{isObs}, \mathsf{getObs})$ and $(\mathsf{Obs}', \mathsf{isObs}', \mathsf{getObs}')$ for their observation infrastructures, and similarly for their secrecy infrastructures, bounds and triggers. $\mathcal{F}'$ is said to be *weaker* than $\mathcal{F}$, written $\mathcal{F}' \leq \mathcal{F}$, if there exist two partial functions $f : \mathsf{Sec} \rightharpoonup \mathsf{Sec}'$ and $g : \mathsf{Obs} \rightharpoonup \mathsf{Obs}'$ that preserve the secrecy and observation infrastructures, the bounds and the triggers, i.e., such that the following hold:

- $\mathsf{isSec}'\ t$ if and only if $\mathsf{isSec}\ t$ and f is defined on $\mathsf{getSec}\ t$, and in this case $f\ (\mathsf{getSec}\ t) = \mathsf{getSec}'\ t$;
- $\mathsf{isObs}'\ t$ if and only if $\mathsf{isObs}\ t$ and g is defined on $\mathsf{getObs}\ t$, and in this case $g\ (\mathsf{getObs}\ t) = \mathsf{getObs}'\ t$;
- $\mathsf{T}\ t$ implies $\mathsf{T}'\ t$;
- $\mathsf{B}'\ sl'\ tl'$ and $\mathsf{map}\ f\ sl = sl'$ imply that there exists tl such that $\mathsf{map}\ f\ tl = tl'$ and $\mathsf{B}\ sl\ tl$.

[1] In [8, Sec. V.8], we discuss in great detail the technical reasons for requiring Secret Polarization, which have to do with BD security favoring the under-specification of the time ordering between observations and secrets.

[2] See also [8, App. E] for a discussion on combining independent secret sources for more holistic multi-policy security guarantees.

▶ **Theorem 6.** (Transport Theorem [8]) If $\mathcal{A} \models \mathcal{F}$ and $\mathcal{F}' \leq \mathcal{F}$, then $\mathcal{A} \models \mathcal{F}'$.

In conclusion, one can use the System Compositionality Theorem to obtain for the composite system $\mathcal{A}_1 \times^{\mathsf{match}} \mathcal{A}_2$ a flow policy $\mathcal{F} = \mathcal{F}_1 \times^{(\mathsf{match},\mathsf{matchO},\mathsf{matchS})} \mathcal{F}_2$ with a strong bound, and the Transport Theorem to produce from this a perhaps weaker but more natural flow policy $\mathcal{F}'$ (for the same system $\mathcal{A}_1 \times^{\mathsf{match}} \mathcal{A}_2$). [8, App.A] gives more intuition on using the two theorems in tandem.

2.7.4 The *n*-ary case

The System Compositionality Theorem generalizes quite smoothly from the binary to the n-ary case. Let $(\mathcal{A}_k = (\mathsf{State}_k, \mathsf{Act}_k, \mathsf{Out}_k, \mathsf{istate}_k, \mathsf{Trans}_k))_{k \in \{1,\ldots,n\}}$ be a family of n systems. We fix, for each k, k' with $k \neq k'$, a matching predicate $\mathsf{match}_{k,k'} : \mathsf{Trans}_k \times \mathsf{Trans}_{k'} \to \mathsf{Bool}$. 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 $\mathcal{A}_k$ and pertaining to communication with $\mathcal{A}_{k'}$). We will make the sanity assumption that a system cannot use the same action to communicate with different systems.

Pairwise-Dedicated Communication: If $k' \neq k''$, then for all k the predicates $\mathsf{isCom}_{k,k'}$ and $\mathsf{isCom}_{k,k''}$ are disjoint, in that there exists no $a \in \mathsf{Act}_k$ such that $\mathsf{isCom}_{k,k'}\ a$ and $\mathsf{isCom}_{k,k''}\ a$. The match-communicating product of the family of systems $(\mathcal{A}_k)_{k \in \{1,\ldots,n\}}$, written $\prod^{\mathsf{match}}_{k \in \{1,\ldots,n\}} \mathcal{A}_k$, generalizes of the binary case. Namely, it is the following system $(\mathsf{State}, \mathsf{Act}, \mathsf{Out}, \mathsf{istate}, \mathsf{Trans})$:

- $\mathsf{State} = \prod_{k \in \{1,\ldots,n\}} \mathsf{State}_k$; so the states are families $(\sigma_k)_{k \in \{1,\ldots,n\}}$, or $(\sigma_k)_k$ for short;
- $\mathsf{Act} = \sum_{k \in \{1,\ldots,n\}} \mathsf{Act}_k + \sum_{k,k' \in \{1,\ldots,n\}, k \neq k'} \mathsf{Act}_k \times \mathsf{Act}_{k'}$; we write (i, a_i) for elements of the i'th summand on the left (separate actions by component $\mathcal{A}_i$), and $((i, a_i), (j, a_j))$ for elements of the (i, j)'th summand on the right (joint communicating actions by components $\mathcal{A}_i$ and $\mathcal{A}_j$);
- $\mathsf{Out} = \sum_{k \in \{1,\ldots,n\}} \mathsf{Out}_k + \sum_{k,k' \in \{1,\ldots,n\}, k \neq k'} \mathsf{Out}_k \times \mathsf{Out}_{k'}$ (similarly to Act);
- $\mathsf{istate} = (\mathsf{istate}_k)_{k \in \{1,\ldots,n\}}$;
- Trans contains two kinds of transitions:
  - for $i \in \{1, \ldots, n\}$, separate $\mathcal{A}_i$-transitions $((\sigma_k)_k, (i, a_i), (i, ou_i), (\sigma_k)_k[i := \sigma'_i])$, where $(\sigma_i, a_i, ou_i, \sigma'_i) \in \mathsf{Trans}_i$ and $\neg\, \mathsf{isCom}_i\ a_i$;
  - for $i, j \in \{1, \ldots, n\}$ such that $i \neq j$, communication transitions (between $\mathcal{A}_i$ and $\mathcal{A}_j$) $((\sigma_k)_k, ((i, a_i), (j, a_j)), ((i, ou_i), (j, ou_j)), (\sigma_k)_k[i := \sigma'_i, j := \sigma'_j])$, where $(\sigma_i, a_i, ou_i, \sigma'_i) \in \mathsf{Trans}_i$, $(\sigma_j, a_j, ou_j, \sigma'_j) \in \mathsf{Trans}_j$ and $\mathsf{match}_{i,j}(\sigma_i, a_i, ou_i, \sigma'_i)(\sigma_j, a_j, ou_j, \sigma'_j)$.

Above, we wrote $(\sigma_k)_k[i := \sigma'_i]$ 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 the flow policies $\mathcal{F}_k$ for the component systems $\mathcal{A}_k$ and the families of matching predicates for transitions, $\mathsf{match} = (\mathsf{match}_{k,k'})_{k,k'}$, observations, $\mathsf{matchO} = (\mathsf{matchO}_{k,k'})_{k,k'}$, and secrets, $\mathsf{matchS} = (\mathsf{matchS}_{k,k'})_{k,k'}$, the product flow policy $\prod^{(\mathsf{match},\mathsf{matchO},\mathsf{matchS})}_{k \in \{1,\ldots,n\}} \mathcal{F}_k$ is defined as a straightforward generalization of the binary case. For example, its observation domain is $\sum_{k \in \{1,\ldots,n\}} \mathsf{Obs}_k + \sum_{k,k' \in \{1,\ldots,n\}, k \neq k'} \mathsf{Obs}_k \times \mathsf{Obs}_{k'}$, so that it contains either separate observations (k, o_k) or joint observations $((k, o_k), (k', o_{k'}))$. Its trigger T is defined on separate i-transitions to be the trigger of the i component, and on (i, j)-communication transitions to be the disjunction of the triggers of the i and j component. And its bound $\mathsf{B}\ sl\ sl'$ is defined from the component bounds: For all $(sl_k)_k, (sl'_k)_k \in \prod_{k \in \{1,\ldots,n\}} \mathsf{List}(\mathsf{Sec}_k)$, if $sl \in \bigtimes^{\mathsf{matchS}}(sl_k)_k$ and $sl' \in \bigtimes^{\mathsf{matchS}}(sl'_k)_k$, then, for all k, $\mathsf{B}_k\ sl_k\ sl'_k$ holds – where $\bigtimes^{\mathsf{matchS}}$ is the n-ary matchS-shuffle product operator, which applied to a family of lists of secrets $(sl_k)_k$ gives all possible interleavings of these lists achieved by separate individual steps and communication steps.

Now we can formulate an n-ary generalization of the System Compositionality Theorem. Most of its assumptions will be those of the binary version, applied to all pairs of components (k, k') for $k, k' \in \{1, \ldots, n\}$ and $k \neq k'$. The only exception is Secret Polarization, which must be strengthened. It is not sufficient to have a single secret issuer for every pair (k, k'), but we need a unique secret issuer for the entire system of n components.

Unique Secret Polarization: There exists $i \in \{1, \ldots, n\}$ such that for all $k \in \{1, \ldots, n\}$ with $k \neq i$ and for all $t \in \mathsf{Trans}_k$, $\mathsf{isSec}_k\ t$ implies $\mathsf{isCom}_{k,i}\ (\mathsf{actOf}_k\ t)$.

▶ **Theorem 7.** (System Compositionality Theorem, n-ary case [8]) Assume the following:
- For all $k, k' \in \{1, \ldots, n\}$ such that $k \neq k'$, the flow policies $\mathcal{F}_k$ and $\mathcal{F}_{k'}$ and their communication infrastructure $(\mathsf{match}_{k,k'}, \mathsf{matchO}_{k,k'}, \mathsf{matchS}_{k,k'})$ satisfy the properties of Pairwise-Dedicated, Compatible, Strong and Observable Communication.
- The families $(\mathcal{F}_k)_{k\in\{1,\ldots,n\}}$ and $(\mathsf{match}_{k,k'}, \mathsf{matchO}_{k,k'}, \mathsf{matchS}_{k,k'})_{k,k'\in\{1,\ldots,n\},k\neq k'}$ (as a whole) satisfy Unique Secret Polarization.
- $\mathcal{A}_k \models \mathcal{F}_k$ for all $k \in \{1, \ldots, n\}$.

Then $\prod^{\mathsf{match}}_{k\in\{1,\ldots,n\}} \mathcal{A}_k \models \prod^{(\mathsf{match},\mathsf{matchO},\mathsf{matchS})}_{k\in\{1,\ldots,n\}} \mathcal{F}_k$.

In conclusion, the generalization of the System Compositionality Theorem to the n-ary case proceeds almost pairwise, but with an additional sanity assumption (Pairwise-Dedicated Communication) and a strengthened assumption (Unique Secret Polarization).

3 Verified Systems

We have formalized in Isabelle/HOL the BD security framework (consisting of Section 2's concepts and theorems) [10, 35]. Recall that the framework operates on nondeterministic I/O automata. We have instantiated it to particular (deterministic) automata representing the functional kernels of some web-based systems. Fig. 5 shows the high-level architecture of these systems, which follows a paradigm of security by design:
- The kernel is formalized and verified in Isabelle.
- The formalization is automatically translated into a functional programming language – which in all our case studies was Scala, one of the target languages of Isabelle's code generator [20, 21].
- The translated program is wrapped in a user-friendly web application.

3.1 CoCon

CoCon [23, 36, 37] is an EasyChair-like conference management system, which was deployed to two international conferences: TABLEAUX 2015 and ITP 2016 [37, §5]. The web application

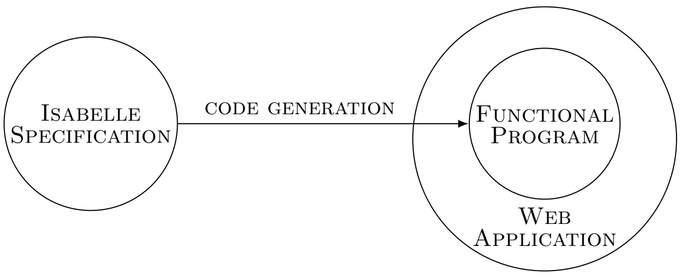

Figure 5 High-level architecture of the verified systems.

Table 1 Confidentiality properties for CoCon. The observations are made by a group of users G.

Secrets	**Declassification Trigger**	**Declassification Bound**
Paper	Some user in G is one of the paper's authors	Last uploaded version
	Some user in G is one of the paper's authors or a PC member$^{\mathsf{B}}$	Absence of any upload
Review	Some user in G is the review's author	Last edited version before Discussion and all the later versions
	Some user in G is the review's author or a non-conflicted PC member$^{\mathsf{D}}$	Last edited version before Notification
	Some user in G is the review's author or a non-conflicted PC member$^{\mathsf{D}}$ or the reviewed paper's author$^{\mathsf{N}}$	Absence of any edit
Discussion	Some user in G is a non-conflicted PC member	Absence of any edit
Decision	Some user in G is a non-conflicted PC member	Last edited version
	Some user in G is a non-conflicted PC member or a PC member$^{\mathsf{N}}$ or the decided paper's author$^{\mathsf{N}}$	Absence of any edit
Reviewer assignment	Some user in G is a non-conflicted PC member$^{\mathsf{R}}$	Reviewers being non-conflicted PC members, and number of reviewers
	Some user in G is a non-conflicted PC member$^{\mathsf{R}}$ or one of the reviewed paper's authors$^{\mathsf{N}}$	Reviewers being non-conflicted PC members

Phase Stamps: B = Bidding, D = Discussion, N = Notification, R = Review

layer of Fig. 5 was realized as a thin REST API implemented in Scalatra [45] wrapped around the verified kernel together with a stateless GUI written in AngularJS [2] that communicates with the API.

CoCon was our first case study, which motivated the initial design and formalization of the BD security framework. Our goal to express, let alone verify, fine-grained policies concerning the flow of information in CoCon between users and documents, could not be supported by the existing concepts in the literature. (See [23, §4.1] for a discussion.) Examples of properties we wanted to express are:

(1) A group of users learn nothing about a paper *beyond* the last uploaded version *unless* one of them becomes an author of that paper.
(2) A group of users learn nothing about a paper *beyond* the absence of any upload *unless* one of them becomes an author of that paper or a PC member at the paper's conference.
(3) A group of users learn nothing about the content of a review *beyond* the last edited version before Discussion phase and the later versions *unless* one of them is that review's author.

The BD security trigger and bound were born out of the need to formally capture the "unless" and "beyond" components of such properties. Tab. 1 summarizes informally the CoCon properties we have expressed in our framework as flow policies. The observation

Table 2 Confidentiality properties for the original CoSMed. The observations are made by a group of users G. The trigger is vacuously false.

Secrets	**Declassification Bound**
Content of a given post	Updates performed while or last before one of the following holds: – Some user in G is the admin, is the post owner or is friends with its owner – The post is marked as public
Friendship status between two given users U and V	Status changes performed while or last before the following holds: – Some user in G is the admin or is friends with U or V
Friendship requests between two given users U and V	Existence of accepted requests while or last before the following holds: – Some user in G is the admin or is friends with U or V

Table 3 Confidentiality properties for CoSMeDis, lifted from CoSMed. The observations are made by n groups of users – one group G_i at each node i. The declassification trigger is vacuously false.

Secrets	**Declassification Bound**
Content of a given post at node i	Updates performed while or last before one of the following holds: – Some user in G_i is the node's admin, is the post owner or is friends with its owner – The post is marked as public – Some user in G_j for $j \neq i$ is the admin at node j or is remote friends with the post's owner
Friendship status between two given users U and V at node i	Status changes performed while or last before the following holds: – Some user at node i is the node's admin or is friends with U or V
Friendship requests between two given users U and V at node i	Existence of accepted requests while or last before the following holds: – Some user at node i is the node's admin or is friends with U or V

infrastructure is always the same, given by the actions and outputs of a fixed group G of users. The secrecy infrastructures are given by the various documents managed by the system (paper content, review, discussion, decision) but also, in the table's last two rows, by information about the reviewers assigned to a paper. These properties should be read as follows: A group of users learns nothing about the given secret (more precisely, about all the uploads or edits performed on a document in the indicated "secret" category) beyond the indicated bound, unless the indicated trigger becomes true. For example, the above properties (1)–(3) are the first three shown in the table, with slightly stronger triggers factoring in the conference phase as well, which we indicate succinctly via "phase stamps" – e.g., the presence of the phase stamp "D" indicates the requirement that the conference must have moved into the Discussion phase. For each type of secret, we have a range of increasingly restrictive bounds matched by increasingly weaker triggers – indeed, the more we tighten the bound (meaning we allow less information to flow), the weaker the trigger becomes (since there are more events that could break the bound). This bound–trigger dynamics exhaustively characterizes the possible flows in the system.

The notion of BD unwinding was developed and refined during the verification of CoCon's policies. The opportunity to take proof shortcuts (via the exit predicate) was discovered during practical "proof hacking" sessions, and led to major simplifications in the development. The different unwinding components in the Sequential Multiplex Unwinding Theorem were naturally mapped to the different phases of a conference's workflow.

3.2 CoSMed

CoSMed [9, 11] is a simple Facebook-style social media platform, where users can register, create posts and establish friendship relationships. It was implemented following the same high-level architecture as CoCon. But unlike CoCon, CoSMed is only a research prototype, not intended for practical use.

CoSMed's confidentiality properties raised new challenges and inspired a more expressive way of modeling flows. In the style of CoCon, we could have specified and proved properties such as:

> A group of users learn nothing about a post *unless* one of them is the admin, or is the post's owner, or becomes friends with the owner, or the post gets marked as public.

Remember that the trigger introduced via "unless" expresses a condition in whose presence the property stops guaranteeing anything – in other words, a trigger opens an access window indefinitely. While true, such a property is not strong enough to be useful for CoSMed, where both friendship and public visibility can be freely switched on and off by the owner at any time (e.g., by "unfriending" a user, and later "friending" them again). Instead, we wanted to prove more dynamic flow policies, reflecting any number of successive openings and closings of the access windows during system execution.

Tab. 2 summarizes informally the BD security properties that we ended up proving for CoSMed. The observation infrastructure is again given by a group G of users, and the secrecy infrastructure refers to either the content of a given post, or to information on the friendship status between two users or on the issued friendship requests. For example, the property on the first row is the dynamic-flow refinement of the coarser property discussed above:

> A group of users learn nothing about a post beyond the updates performed while (or last before) one of them is the admin, or is the post's owner, or becomes friends with the owner, or the post is marked as public.

Thus, the "beyond–unless" bound-trigger combination we had employed for CoCon gave way to a "beyond–while" scheme for CoSMed, where "while" refers to the allowed access windows. To achieve this formally, we made the triggers vacuously false (i.e., deactivated them completely) and incorporated the opening and closing of access windows in inductively defined bounds. [9] discusses in detail this paradigm shift, which however did not require adjustments to the framework itself.

3.3 CoSMeDis

CoSMeDis [8, 12] is a multi-node distributed extension of CoSMed that follows a Diaspora-style scheme [1]: Different nodes can be deployed independently at different internet locations. The admins of any two nodes can initiate a protocol to connect these nodes, after which the users of one node can establish friendship relationships and share data with users of the other. Thus, a node of CoSMeDis consists of CoSMed plus actions for connecting nodes and cross-node post sharing and friending.

Our goal was to extend the confidentiality properties we had verified for CoSMed first to one CoSMeDis node, then to the multi-node CoSMeDis network. [8] describes in great detail this verification extension effort, which led to the discovery of the System Compositionality Theorems. The outcome was the properties shown in Tab. 3, which are natural multi-node generalizations of CoSMed's properties (from Tab. 2). They were obtained by applying the n-ary System Compositionality Theorem, then the Transport Theorem to switch to more readable secrets and bounds.

4 Related Work

We only discuss briefly the most related work, focusing on the general framework rather than the verification case studies. For more comprehensive literature comparisons (which also cover verification), we refer to our earlier papers [8, 9, 37].

Since we aimed for high expressiveness and precision, we defined BD security by quantifying over execution traces of general systems. This "heavy duty" approach, sometimes called *system-based security* [26], can be contrasted with *language-based security* [42], concerned with coarser-grained but tractable notions that can be automatically analyzed on programming language syntax.

BD security provides an expressive realization of Sabelfeld and Sands's dimensions of declassification [44] in a system-based setting. It descends from the epistemic logic [40] inspired tradition of modeling information-flow security, pioneered by Sutherland with Nondeducibility [46] and continued with Halpern and O'Neill's Secrecy Maintenance [22] and with Askarov et al.'s Gradual Release [3–6], the latter developed in a language-based setting. Our BD unwinding is a non-trivial generalization of unwinding proof methods going back to Goguen and Meseguer [19] and Rushby [41], which have been extensively studied as part of Mantel's MAKS framework [24, 26]. Unlike these predecessors which use safety-like unwinding conditions, BD unwinding combines safety with liveness: In the BD unwinding game, the "defender", who builds the alternative trace tr_2, must

- not only be able to *always* stay in the game – a safety-like property,
- but also be able to *eventually* produce the alternative secrets sl_2 (provided the "attacker", who controls the original trace tr_1, has produced all the original secrets sl_1) – a liveness-like property.

Because of the restrictive way of handling the liveness part of the aforementioned game, BD unwinding is not a complete proof method, in that it cannot prove every instance of BD security. We leave a complete extension of BD unwinding as future work.

Our system compositionality result joins a body of technically delicate work in system-based security, where the difficult terrain was recognized early on [27]. Several frameworks have been developed in various settings, e.g., event systems [25], reactive systems [39] and process calculi [13, 17]. Some of these focus on formulating very restricted classes of security properties that are always guaranteed to be preserved under a given notion of composition, such as McCullough's Restrictiveness [28]. Others, such as Mantel's MAKS framework [24, 25], formulate side conditions on the components' security properties that guarantee compositionality. Our result is in the latter category, and refers to a significantly more expressive notion of information-flow security than its predecessors (which is not to say that our result subsumes these previous results).

Temporal logics designed for information-flow security, such as SecLTL [15] and HyperCTL* [14, 16, 38], can express similar-looking properties to the instances of BD security we verified for CoCon – though semantically they differ by interpreting trace quantification synchronously.

References

1 The Diaspora project. `https://diasporafoundation.org/`, 2016.
2 The AngularJS Web Framework, 2021. URL: `https://angularjs.org/`.
3 Aslan Askarov and Stephen Chong. Learning is change in knowledge: Knowledge-based security for dynamic policies. In *CSF*, pages 308–322, 2012.
4 Aslan Askarov and Andrew C. Myers. Attacker control and impact for confidentiality and integrity. *Logical Methods in Computer Science*, 7(3), 2011.
5 Aslan Askarov and Andrei Sabelfeld. Gradual release: Unifying declassification, encryption and key release policies. In *IEEE Symposium on Security and Privacy*, pages 207–221, 2007.
6 Aslan Askarov and Andrei Sabelfeld. Tight enforcement of information-release policies for dynamic languages. In *CSF*, pages 43–59, 2009.
7 Thomas Bauereiss, Armando Pesenti Gritti, Andrei Popescu, and Franco Raimondi. CoSMed: A Confidentiality-Verified Social Media Platform. In *ITP*, 2016.
8 Thomas Bauereiss, Armando Pesenti Gritti, Andrei Popescu, and Franco Raimondi. CoSMeDis: A distributed social media platform with formally verified confidentiality guarantees. In *IEEE Symposium on Security and Privacy*, pages 729–748, 2017.
9 Thomas Bauereiss, Armando Pesenti Gritti, Andrei Popescu, and Franco Raimondi. CoSMed: A Confidentiality-Verified Social Media Platform. *J. Autom. Reasoning*, 61(1-4):113–139, 2018.
10 Thomas Bauereiss and Andrei Popescu. Compositional BD Security. *Archive of Formal Proofs*, 2021. URL: `https://www.isa-afp.org/entries/Compositional_BD_Security.html`.
11 Thomas Bauereiss and Andrei Popescu. CoSMed: A confidentiality-verified social media platform. *Archive of Formal Proofs*, 2021. URL: `https://www.isa-afp.org/entries/CoSMed.html`.
12 Thomas Bauereiss and Andrei Popescu. CoSMeDis: A confidentiality-verified distributed social media platform. *Archive of Formal Proofs*, 2021. URL: `https://www.isa-afp.org/entries/CoSMeDis.html`.
13 Annalisa Bossi, Damiano Macedonio, Carla Piazza, and Sabina Rossi. Information flow in secure contexts. *Journal of Computer Security*, 13(3):391–422, 2005. URL: `http://content.iospress.com/articles/journal-of-computer-security/jcs235`.
14 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *POST*, pages 265–284, 2014.
15 Rayna Dimitrova, Bernd Finkbeiner, Máté Kovács, Markus N. Rabe, and Helmut Seidl. Model checking information flow in reactive systems. In *VMCAI*, pages 169–185, 2012.
16 Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL ˆ*. In *International Conference on Computer Aided Verification*, pages 30–48. Springer, 2015.
17 Riccardo Focardi and Roberto Gorrieri. Classification of security properties (Part I: Information flow). In *FOSAD*, pages 331–396, 2000.
18 Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.
19 Joseph A. Goguen and José Meseguer. Unwinding and inference control. In *IEEE Symposium on Security and Privacy*, pages 75–87, 1984.
20 Florian Haftmann. *Code Generation from Specifications in Higher-Order Logic*. Ph.D. thesis, Technische Universität München, 2009.
21 Florian Haftmann and Tobias Nipkow. Code generation via higher-order rewrite systems. In *FLOPS 2010*, pages 103–117, 2010.
22 Joseph Y. Halpern and Kevin R. O'Neill. Secrecy in multiagent systems. *ACM Trans. Inf. Syst. Secur.*, 12(1), 2008.
23 Sudeep Kanav, Peter Lammich, and Andrei Popescu. A conference management system with verified document confidentiality. In *CAV*, pages 167–183, 2014.

24 Heiko Mantel. Possibilistic definitions of security - an assembly kit. In *CSFW*, pages 185–199, 2000.

25 Heiko Mantel. On the composition of secure systems. In *IEEE Symposium on Security and Privacy*, pages 88–101, 2002.

26 Heiko Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security.* PhD thesis, University of Saarbrücken, 2003.

27 Daryl McCullough. Specifications for multi-level security and a hook-up property. In *IEEE Symposium on Security and Privacy*, 1987.

28 Daryl McCullough. A hookup theorem for multilevel security. *IEEE Trans. Software Eng.*, 16(6):563–568, 1990.

29 John McLean. Security models. In *Encyclopedia of Software Engineering*, 1994.

30 Toby C. Murray, Andrei Sabelfeld, and Lujo Bauer. Special issue on verified information flow security. *Journal of Computer Security*, 25(4-5):319–321, 2017.

31 Tobias Nipkow and Gerwin Klein. *Concrete Semantics: With Isabelle/HOL*. Springer, 2014.

32 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

33 Andrei Popescu, Johannes Hölzl, and Tobias Nipkow. Proving concurrent noninterference. In *CPP*, pages 109–125, 2012.

34 Andrei Popescu, Johannes Hölzl, and Tobias Nipkow. Formalizing probabilistic noninterference. In *CPP*, pages 259–275. Springer, 2013.

35 Andrei Popescu and Peter Lammich. Bounded-deducibility security. *Archive of Formal Proofs*, 2014. URL: `https://www.isa-afp.org/entries/Bounded_Deducibility_Security.html`.

36 Andrei Popescu and Peter Lammich. CoCon: A confidentiality-verified conference management system. *Archive of Formal Proofs*, 2021. URL: `https://www.isa-afp.org/entries/CoCon.html`.

37 Andrei Popescu, Peter Lammich, and Ping Hou. CoCon: A conference management system with formally verified document confidentiality. *J. Autom. Reason.*, 65(2):321–356, 2021.

38 Markus N. Rabe, Peter Lammich, and Andrei Popescu. A shallow embedding of HyperCTL. *Archive of Formal Proofs*, 2014, 2014.

39 Willard Rafnsson and Andrei Sabelfeld. Compositional information-flow security for interactive systems. In *CSF*, pages 277–292, 2014.

40 Yoram Moses Ronald Fagin, Joseph Y. Halpern and Moshe Vardi. *Reasoning about knowledge.* MIT Press, 2003.

41 John Rushby. Noninterference, transitivity, and channel-control security policies. Technical report, Computer Science Laboratory SRI International, December 1992. URL: `http://www.csl.sri.com/papers/csl-92-2/`.

42 Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

43 Andrei Sabelfeld and David Sands. Probabilistic noninterference for multi-threaded programs. In *CSFW*, pages 200–214, 2000.

44 Andrei Sabelfeld and David Sands. Declassification: Dimensions and principles. *Journal of Computer Security*, 17(5):517–548, 2009.

45 The Scalatra Web Framework, 2021. URL: `http://scalatra.org/`.

46 D. Sutherland. A model of information. In *9th National Security Conf.*, pages 175–183, 1986.

47 Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2,3):167–187, 1996.