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” (Ni491/13-2) and “MORES – Modelling and Refinement of Security Requirements on Data and Processes (Hu737/5-2)”, part of “RS³ – 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 nondeterministic 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.
3:2  Bounded-Deducibility Security

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.” (Important, 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.

\( \text{Bool} \) denotes the two-element set of Booleans, \( \{ \text{true}, \text{false} \} \). Predicates and relations will be modeled as functions to \( \text{Bool} \). For example, \( P : A \to \text{Bool} \) is a (unary) predicate on \( A \) and \( Q : A \to A \to \text{Bool} \) is a binary relation on \( A \). Given \( a \in A \), we write “\( P a \) holds”, or simply “\( P a \)”, to mean that \( P a = \text{true} \); and similarly for binary relations.

Given a set \( A \), we write \( \text{Set}(A) \) for the powerset (i.e., set of all subsets) of \( A \), and \( \text{List}(A) \) 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 \( a, b \) will denote elements in \( \text{List}(A) \). An exception will be the system traces – even though they are lists of transitions \( t \), for them we will use the customized notation \( t r \). We write “\( . \)’’ for list concatenation. Applied to a non-empty list \( [a_1, \ldots, a_n] \), the function \( \text{head} \) returns its first element \( a_1 \). Given a function \( f : A \to B \) and \( [a_1, \ldots, a_n] \in \text{List}(A) \), map \( f \) \( [a_1, \ldots, a_n] \) returns \( [f a_1, \ldots, f a_n] \). Given a partial function \( f : A \to B \) and \( [a_1, \ldots, a_n] \in \text{List}(A) \), let \( [a_1, a_2, \ldots, a_k] \) 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 map \( f \) \( [a_1, \ldots, a_n] \)
returns \( [f a_1, \ldots, f a_n] \). In other words, partial functions are mapped while omitting the elements on which they are not defined. Given a predicate \( P \), \( \text{filter} \, P \, [a_1, \ldots, a_n] \) 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 \( \mathcal{A} = (\text{State}, \text{Act}, \text{Out}, \text{istate}, \text{Trans}) \), where:

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

(Note that we call “action” what is usually called “input” for I/O automata.) A transition \( t = (\sigma, a, ou, \sigma') \in \text{Trans} \) has the following interpretation: If action \( a \) is taken while the system is in state \( \sigma \), the system may respond by producing output \( ou \) and changing the state to \( \sigma' \). We call \( \sigma \) the source, \( a \) the action, \( ou \) the output, and \( \sigma' \) the target of \( t \). The transition’s action \( a \) is also denoted by \( \text{actOf} \, t \). We will write \( \sigma \xrightarrow{t} \sigma' \) to express that \( t \in \text{Trans} \), \( \sigma \) is the source of \( t \) and \( \sigma' \) 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 \text{istate} and, for all \( i \in \{2, \ldots, n\} \), the source of \( t_i \) is the target of \( t_{i-1} \). We let \text{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 \( \text{TraceF}_\sigma \) for the set of trace fragments that start in \( \sigma \), i.e., have \( \sigma \) 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 \( \text{Trace}_{\mathcal{A}} \) instead of \( \text{Trace} \), \( \text{TraceF}_{\mathcal{A}, \sigma} \) instead of \( \text{TraceF}_\sigma \), etc.

### 2.2 Flow policies

Given a system \( \mathcal{A} = (\text{State}, \text{Act}, \text{Out}, \text{istate}, \text{Trans}) \), 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.
3.4 Bounded-Deducibility Security

For accommodating these requirements, we define a flow policy \( F \) to consist of:

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

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

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

4. a declassification trigger, i.e., a predicate on transitions, \( T : \text{Trans} \rightarrow \text{Bool} \).

Note that the observation and secrecy infrastructures have the same form. We define \( O : \text{Trace} \rightarrow \text{List(Obs)} \) by \( O = \text{map getObs} \circ \text{filter isObs} \), and \( S : \text{Trace} \rightarrow \text{List(Sec)} \) by \( S = \text{map getSec} \circ \text{filter isSec} \). Thus, \( O \) uses filter to select the transitions in a trace that are observable according to \( \text{isObs} \), and then applies \( \text{getObs} \) to each selected transition. Similarly, \( S \) produces lists of secrets by filtering with \( \text{isSec} \) and applying \( \text{getSec} \). 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 \( \mathcal{A} = (\text{State}, \text{Act}, \text{Out}, \text{istate}, \text{Trans}) \) and a flow policy \( F \), where \((\text{Obs}, \text{isObs}, \text{getObs})\) is its observation infrastructure, \((\text{Sec}, \text{isSec}, \text{getSec})\) its secrecy infrastructure, \( B \) its declassification bound and \( T \) its declassification trigger. Furthermore, let \( O : \text{Trace} \rightarrow \text{List(Obs)} \) and \( S : \text{Trace} \rightarrow \text{List(Sec)} \) 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 \( F \), written \( \mathcal{A} \models F \), provided that for all \( tr_1 \in \text{Trace} \) and \( sl_1, sl_2 \in \text{List(Sec)} \),

- if \( \text{never } T \) \( tr_1 \), \( S \) \( tr_1 = sl_1 \) and \( B \) \( sl_1 \) \( sl_2 \),

  then there exists \( tr_2 \in \text{Trace} \) such that \( O \) \( tr_2 = O \) \( tr_1 \) and \( S \) \( tr_2 = sl_2 \).

The predicate \( \text{never } T \) \( tr_1 \) 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.
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 $t_1$ consists of three transitions, $t_1 = [t_1, t'_1, t''_1]$, 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 $t_2$ also consists of three transitions, $t_2 = [t_2, t'_2, t''_2]$, 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:

$$
\begin{array}{c|c|c|c|c|c|c|c|c}
\hline
& isObs & getObs & isSec & getSec & isObs & getObs & isSec & getSec \\
\hline
\neg T & o_1 & s_1 & t_1 & o_2 & s_2 & t_2 & o'_2 & s'_2 \\
\end{array}
$$

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 $\forall \exists$ statement expressing BD security is illustrated on the figure by making a choice of the $\forall$-quantified entities and the $\exists$-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 $\neg T$) 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 $O : \text{Trace} \to \text{List(Obs)}$ and $S : \text{Trace} \to \text{List(Sec)}$ by requiring that, for all $t_1 \in \text{Trace}$ and $s_1, s_2 \in \text{List(Sec)}$, if $S t_1 = s_1$, then there exists $t_2 \in \text{Trace}$ such that $O t_2 = O t_1$ and $S t_2 = s_2$. 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, $A \models \mathcal{F}$, one needs to do the following: Given

- the original trace $tr_1$ for which $\neg T$ holds and which produces the list of secrets $sl_1$,
- and an alternative list of secrets $sl_2$ such that $B \subseteq sl_1 \cup sl_2$ 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 $\Delta$ on State that bookkeep the states reached by $tr_1$ and $tr_2$, say $\sigma_1$ and $\sigma_2$, and show how these can evolve transition by transition in the process of constructing $tr_2$ from $tr_1$; they guarantee that any $\Delta$-related states $\sigma_1$ and $\sigma_2$ evolve via transitions $\sigma_1 \xrightarrow{\Delta} \sigma_1'$ and $\sigma_2 \xrightarrow{\Delta} \sigma_2'$ to $\Delta$-related states $\sigma_1'$ and $\sigma_2'$. 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 $(\sigma, sl)$ and $(\sigma', sl')$ in State $\times$ List(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 $\neg \text{isSec} t$ and $sl' = sl$, or $\text{isSec} t$ 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 $t_1 =_{\text{obs}} t_2$ as a shorthand for the following two statements: (1) $\text{isObs} t_1$ if and only if $\text{isObs} t_2$, and (2) if $\text{isObs} t_1$ then $\text{getObs} t_1 = \text{getObs} t_2$. In other words, $t_1$ and $t_2$ produce either the same observation or no observation.

A relation $\Delta : (\text{State} \times \text{List(Sec)}) \rightarrow (\text{State} \times \text{List(Sec)}) \rightarrow \text{Bool}$ is said to be a BD unwinding if, for all $(\sigma_1, sl_1)$, $(\sigma_2, sl_2) \in \text{State} \times \text{List(Sec)}$ such that $\sigma_1$ is $\neg T$-reachable, $\sigma_2$ 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 reaction $\Delta (\sigma_1, sl_1) (\sigma_2, sl_2)$; or
2. action $\Delta (\sigma_1, sl_1) (\sigma_2, sl_2)$; or
3. $sl_1 \neq []$ and exit $\sigma_1$ (head $sl_1$).

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

The predicates reaction, action (read “independent action”) and exit will be defined below. The first two describe possible evolution patterns for the pairs $(\sigma_1, sl_1)$ and $(\sigma_2, sl_2)$ so that the result is still in $\Delta$. 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 $\sigma_2$), in response to the growth of an original trace (that has currently reached state $\sigma_1$), 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.
there exist

we can think of

with producing its secrets (\(\text{State} \times \text{List(Sec)}\)) such that \((\sigma_1, s_1) \xrightarrow{\Delta} (\sigma'_1, s'_1)\), one of the following two cases holds:

1. \(\neg \text{isObs } t_1 \) and \(\Delta (\sigma'_1, s'_1) (\sigma_2, s_l)\); or

2. there exist \(t_2 \in \text{Trans}\) and \((\sigma'_2, s'_2) \in \text{State} \times \text{List(Sec)}\) such that \((\sigma_2, s_2) \xrightarrow{\Delta} (\sigma'_2, s'_2)\), \(t_1 = \text{Obs } t_2\) and \(\Delta (\sigma'_1, s'_1) (\sigma'_2, s'_2)\).

Thus, \(\Delta (\sigma_1, s_1) (\sigma_2, s_2)\) 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 \(\Delta\).

\(\text{ reaction } \Delta (\sigma_1, s_1) (\sigma_2, s_2)\) is defined to mean that there exist \(t_2 \in \text{Trans}\) and \((\sigma'_2, s'_2) \in \text{State} \times \text{List(Sec)}\) such that \((\sigma_2, s_2) \xrightarrow{\Delta} (\sigma'_2, s'_2)\), \(\neg \text{isObs } t_2\), \(\text{isSec } t_2\) and \(\Delta (\sigma_1, s_1)(\sigma'_2, s'_2)\).

Thus, \(\text{ reaction}\) 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 (\(\neg \text{isObs } t_2\)) is justified by the desire to keep the observations synchronized, the reason for the secret-producing requirement (\(\text{isSec } t_2\)) 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.

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

The idea behind \(\text{exit}\) is that BD security holds trivially for original traces that are unable to produce their due list of secrets \(s_l\); and \(\text{exit}\) detects this (thus closing that branch of the unwinding proof) by noticing that not even the first secret in \(s_l\) can be produced starting from the current state \(\sigma_1\) – indeed, in the definition of unwinding, \(\text{exit}\) is invoked with \(\sigma_1\) and \(\text{head } s_l\).

Left unexplained so far are the (non)emptiness conditions guarding the invocations of the \(\text{reaction}\) and \(\text{exit}\) predicates in the definition of BD unwinding. For \(\text{exit}\), it is obvious that we need \(s_l \neq \emptyset\) for talking about the first element in \(s_l\). But for \(\text{reaction}\), why require that \(s_l \neq \emptyset\) or \(s_l = \emptyset\)? 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 (\(s_l = \emptyset\)) and the alternative trace still has some secrets to produce (\(s_l \neq \emptyset\)). In that case, we want to enforce an \(\text{iaction}\) move which, being secret-producing, would make progress through the remaining alternative list of secrets \(s_l\); this is achieved by preventing a \(\text{reaction}\) move, which would be the only alternative (since an \(\text{exit}\) move needs \(s_l \neq \emptyset\)). With these definitions, BD unwinding fulfills its goal:

\(\textbf{Lemma 1.}\) [23, 37] Assume \(\Delta\) is a BD unwinding and let \(\sigma_1, \sigma_2 \in \text{State}\) such that \(\text{reach } \neg \text{T } \sigma_1\) and \(\text{reach } \sigma_2\). Then, for all \(t_{1} \in \text{TraceF}_{\sigma_1}\) and \(s_{1}, s_{2} \in \text{List(Sec)}\),

\(\text{ if } \text{never } \text{T } t_{1}, S\ t_{1} = s_{1} \text{ and } \Delta (\sigma_1, s_1) (\sigma_2, s_2),\)

\(\text{ then there exists } t_{2} \in \text{TraceF}_{\sigma_2} \text{ such that } O\ t_{2} = O\ t_{1} \text{ and } S\ t_{2} = s_{2}.\)

In other words, assuming \(\Delta (\sigma_1, s_1) (\sigma_2, s_2)\) holds and given the remaining part \(t_{1}\) of the original trace (starting in \(\sigma_1\)) which produces secrets \(s_{1}\), there exists a trace \(t_{2}\) that produces the same observations and produces the desired secrets \(s_{2}\). The lemma’s proof goes by induction on the sum of the lengths of \(t_{1}\) and \(s_{2}\). The induction step either reaches a contradiction (if \(\text{exit}\) is invoked), or consumes a transition from \(t_{1}\) (if \(\text{reaction}\) is invoked) or a secret from \(s_{2}\) (if \(\text{iaction}\) 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 \(\Delta\) includes the bound \(B\) in the initial state. So we can think of \(\Delta\) 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 $\sigma_1 = \sigma_2 = \text{istate}$. 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 $sl_1, sl_2 \in \text{List(Sec)}$, if $B \cdot sl_1 \cdot sl_2$ then $\Delta$ (istate, $sl_1$) (istate, $sl_2$).
(2) $\Delta$ is a BD unwinding.
Then $A \vdash F$.

According to this theorem, to prove BD security of a system, it suffices to define a relation $\Delta$ 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 $\Delta$ and a set of relations $\Delta s$, $\Delta$ is said to be a BD unwinding into $\Delta s$ if it satisfies the same conditions as in the definition of BD unwinding, just that iaction $\Delta$ and reaction $\Delta$ are replaced by iaction ($\bigvee \Delta s$) and reaction ($\bigvee \Delta s$), where $\bigvee \Delta s$ is the disjunction (i.e., union) of all the relations in $\Delta s$. Namely, for all $(\sigma_1, sl_1), (\sigma_2, sl_2) \in \text{State} \times \text{List(Sec)}$ such that $\sigma_1$ is ($\neg T$)-reachable, $\sigma_2$ 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 reaction ($\bigvee \Delta s$) ($\sigma_1, sl_1$) ($\sigma_2, sl_2$); or
(2) iaction ($\bigvee \Delta s$) ($\sigma_1, sl_1$) ($\sigma_2, sl_2$); or
(3) $sl_1 \neq []$ and exit $\sigma_1$ (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 $\Delta s$ be a set of relations. For each $\Delta \in \Delta s$, let next$\Delta \subseteq \Delta s$ be a (possibly empty) set of “successors” of $\Delta$, and let $\Delta_{\text{ init}} \in \Delta s$ be a chosen “initial” relation. Assume the following hold:
(1) For all $sl_1, sl_2 \in \text{List(Sec)}$, if $B \cdot sl_1 \cdot sl_2$ then $\Delta_{\text{ init}}$ (istate, $sl_1$) (istate, $sl_2$).
(2) Each $\Delta \in \Delta s$ is a BD unwinding into next$\Delta$.
Then $A \vdash F$.

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 $\Delta_i$, unwinds either into itself, or into $\Delta_{i+1}$ (if $i \neq n$), or into an exit component $\Delta_e$ that always chooses the “exit” unwinding condition. (In practice, $\Delta_e$ 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 $\Delta$ being a BD continuation-unwinding into $\Delta s$ 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 reaction ($\bigvee \Delta s$) ($\sigma_1, sl_1$) ($\sigma_2, sl_2$), or (2) iaction ($\bigvee \Delta s$) ($\sigma_1, sl_1$) ($\sigma_2, sl_2$) hold. And $\Delta$ is said to be a BD exit-unwinding if the exit case, (3) $sl_1 \neq []$ and exit $\sigma_1$ (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 $\Delta_e$ such that the following hold:
(1) For all \( sl_1, sl_2 \in \text{List}(\text{Sec}) \), if \( B \cdot sl_1 \cdot sl_2 \) then \( \Delta_1 (\text{istate}, sl_1) (\text{istate}, sl_2) \).

(2) \( \Delta_i \) is a BD continuation-unwinding into \( \{ \Delta_i, \Delta_{i+1}, \Delta_e \} \).

(3) \( \Delta_e \) is a BD exit-unwinding.

Then \( A \models F \).

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 \( A_1 = (\text{State}_1, \text{Act}_1, \text{Out}_1, \text{istate}_1, \text{Trans}_1) \) and \( A_2 = (\text{State}_2, \text{Act}_2, \text{Out}_2, \text{istate}_2, \text{Trans}_2) \) be two systems. We want to model communication between \( A_1 \) and \( A_2 \) by matching certain transitions that these systems must take synchronously while exchanging data. This is captured by a relation \( \text{match} : \text{Trans}_1 \rightarrow \text{Trans}_2 \rightarrow \text{Bool} \). 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 \( \text{isCom}_i a \) (for \( i \in \{1, 2\} \)) to indicate that an action \( a \) is in the latter category for \( A_i \). Namely, \( \text{isCom}_i a \) holds whenever there exist \( t_1 \) and \( t_2 \) such that \( \text{match} t_1 t_2 \) holds and \( a \) is the action of \( t_i \).

We define the **match-communicating product of \( A_1 \) and \( A_2 \)**, written \( A_1 \times^{\text{match}} A_2 \), as the following system \( (\text{State}, \text{Act}, \text{Out}, \text{istate}, \text{Trans}) \):

- \( \text{State} = \text{State}_1 \times \text{State}_2 \);
- \( \text{Act} = \text{Act}_1 + \text{Act}_2 + \text{Act}_1 \times \text{Act}_2 \); thus, \( \text{Act} \) is a disjoint union of \( \text{Act}_1 \) (representing separate actions of the first component), \( \text{Act}_2 \) (for separate actions of the second component), and \( \text{Act}_1 \times \text{Act}_2 \) (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;
- \( \text{Out} = \text{Out}_1 + \text{Out}_2 + \text{Out}_1 \times \text{Out}_2 \); thus, like \( \text{Act} \), \( \text{Out} \) is a disjoint union, and we use similar notations for its elements: \( (1, ou_1), (2, ou_2) \) and \( (ou_1, ou_2) \);
- \( \text{istate} = (\text{istate}_1, \text{istate}_2) \);
- \( \text{Trans} \) contains three kinds of transitions:
Bounded-Deducibility Security

3:10

separate \( A_1 \)-transitions \(((\sigma_1, \sigma_2), (1, a_1), (1, ou_1), (\sigma'_1, \sigma'_2))\),
where \( (\sigma_1, a_1, ou_1, \sigma'_1) \in Trans_1 \) and \( \neg isCom_1 a_1 \);

separate \( A_2 \)-transitions \(((\sigma_1, \sigma_2), (2, a_2), (2, ou_2), (\sigma_1, \sigma'_2))\),
where \( (\sigma_2, a_2, ou_2, \sigma'_2) \in Trans_2 \) and \( \neg isCom_2 a_2 \);

communication transitions \(((\sigma_1, \sigma_2), (a_1, a_2), (ou_1, ou_2), (\sigma'_1, \sigma'_2))\),
where \( (\sigma_1, a_1, ou_1, \sigma'_1) \in Trans_1 \), \( (\sigma_2, a_2, ou_2, \sigma'_2) \in Trans_2 \)
and match \(((\sigma_1, a_1, ou_1, \sigma'_1)) (\sigma_2, a_2, ou_2, \sigma'_2))\).

Thus, a transition \( t \) of \( A_1 \times^{match} A_2 \) has exactly one of the following three forms shown above.
In the first case, \( t \) is completely determined by an \( A_1 \)-transition \( t_1 = (\sigma_1, a_1, ou_1, \sigma'_1) \) and an \( A_2 \)-state \( \sigma_2 \) - we write \( t = isCom_1 t_1 \sigma_2 \), marking that \( t \) is given by the separate transition \( t_1 \).
Similarly, in the second case we write \( t = isCom_2 t_1 \sigma_2 \), where \( t_2 = (\sigma_2, a_2, ou_2, \sigma'_2) \). In the third case, we write \( t = com t_1 t_2 \), marking that \( t \) proceeds as a communication transition.

Thus, in our new notation, any transition of \( A_1 \times^{match} A_2 \) has either the form \( isCom_1 t_1 \sigma_2 \), or \( isCom_2 t_1 \sigma_2 \), or \( com(t_1, t_2) \).

2.7.2 Product flow policies

Let \( F_1 \) and \( F_2 \) be flow policies for \( A_1 \) and \( A_2 \). Given \( t \in \{1, 2\} \), we write \((obs, isObs, getObs)\)
for the observation infrastructure, \((Sec, isSec, getSec)\) for the secrecy infrastructure, \( B \) for the
declassification bound and \( T \) for the declassification trigger of \( F_i \).
We want to compose the policies \( F_1 \) and \( F_2 \) in a natural way, forming a policy for the product \( A_1 \times^{match} A_2 \).
To achieve this, we need observation and secret counterparts of the transition-matching predicate \( match \),
in the form of predicates \( matchO : Obs_1 \rightarrow Obs_2 \rightarrow bool \) and \( matchS : Sec_1 \rightarrow Sec_2 \rightarrow bool \).
Triples \((match, matchO, matchS)\) will be called \emph{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.

\underline{Compatible Communication:}
For all \( t_1 \in Trans_1 \) and \( t_2 \in Trans_2 \), if \( match t_1 t_2 \) then:
\begin{itemize}
  \item \( isSec_1 t_1 \) if and only if \( isSec_2 t_2 \), and in this case we have \( match (getSec_1 t_1) (getSec_2 t_2) \);
  \item \( isObs_1 t_1 \) if and only if \( isObs_2 t_2 \), and in this case we have \( matchO (getObs_1 t_1) (getObs_2 t_2) \).
\end{itemize}

The \emph{product} of \( F_1 \) and \( F_2 \) along a \emph{communication infrastructure} \((match, matchO, matchS)\),
written \( F_1 \times^{(match,matchO,matchS)} F_2 \), is defined as the following flow policy for \( A_1 \times^{match} A_2 \).

We start with its observation and secrecy infrastructures, which are naturally defined considering that observations and secrets can be produced either separated or in communication steps. The observation infrastructure \((Obs, isObs, getObs)\) is the following:
\begin{itemize}
  \item \( Obs_1 + Obs_2 + Obs_1 \times Obs_2 \); thus, an element of \( Obs \) will have either the form \((1, o_1)\), or \((2, o_2)\), or \((o_1, o_2)\), where \( o_1 \in Obs_1 \).
  \item For any \( t \in Trans \), \( isObs t \) and \( getObs t \) are defined as follows:
    \begin{itemize}
      \item if \( t \) has the form \( isCom_1 t_1 \sigma_2 \), then \( isObs t = isObs_1 t_1 \) and \( getObs t = (1, getObs_1 t_1) \);
      \item if \( t \) has the form \( isCom_2 t_1 \sigma_2 \), then \( isObs t = isObs_2 t_2 \) and \( getObs t = (2, getObs_2 t_2) \);
      \item if \( t \) has the form \( com t_1 t_2 \), then \( isObs t = (isObs_1 t_1 \) and \( isObs_2 t_2) \) and \( getObs t = (getObs_1 t_1, getObs_2 t_2) \).
    \end{itemize}
\end{itemize}

One could argue that, when \( t \) has the form \( com t_1 t_2 \), \( isObs t \) should be defined not as \( (1) \) \( isObs_1 t_1 \) and \( isObs_2 t_2 \), but as \( (2) \) \( isObs_1 t_1 \) or \( isObs_2 t_2 \), 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.
The secrecy infrastructure \((\text{Sec}, \text{isSec}, \text{getSec})\) is defined similarly to the observation infrastructure: \(\text{Sec}\) is taken to be \(\text{Sec}_1 + \text{Sec}_2 + \text{Sec}_1 \times \text{Sec}_2\), and \(\text{isSec}\) and \(\text{getSec}\) 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 \(T\) to mean the following: (1) if \(t\) has the form \(\text{sep} 1 t_1 t_2\), then \(T 1 t_1\) holds; (2) if \(t\) has the form \(\text{sep} 2 t_1 t_2\), then \(T 2 t_2\) holds; (3) if \(t\) has the form \(\text{com} t_1 t_2\), then \(T 1 t_1\) holds or \(T 2 t_2\) holds.

It remains to define the bound \(B\) of the product flow policy. Let \(sl 1 \in \text{List} (\text{Sec})\) 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 \(sl_1 \in \text{List} (\text{Sec}_1)\) and \(sl_2 \in \text{List} (\text{Sec}_2)\) 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 \text{matchS}-shuffle product operator \(\times \text{matchS} : \text{List} (\text{Sec}_1) \rightarrow \text{List} (\text{Sec}_2) \rightarrow \text{Set} (\text{List} (\text{Sec}))\) whose inductive definition is shown in Fig. 4. The set \(sl_1 \times \text{matchS} sl_2\) contains all possible interleavings of \(sl_1\) and \(sl_2\), achieved by separate individual steps (rule \(\text{SEP}_1\) and \(\text{SEP}_2\)) and communication steps (rule \(\text{COM}\)). For \(i \in \{1, 2\}\), \(\text{isComS}_s\) is the secret counterpart of the predicate \(\text{isCom}_i\), expressing that the secret \(s\) participates in a \text{matchS}-relationship. We define \(B \ times sl'\) to mean that, for all \(sl_1, sl'_1 \in \text{List} (\text{Sec}_1)\) and \(sl_2, sl'_2 \in \text{List} (\text{Sec}_2)\), if \(sl \in sl_1 \times \text{matchS} sl_2\) and \(sl' \in sl'_1 \times \text{matchS} sl'_2\), then \(B_1 (sl_1, sl'_1)\) and \(B_2 (sl_2, sl'_2)\) hold.

2.7.3 Compositionality result

We next introduce some properties that refer to the flow policies \(F_1\) and \(F_2\) and the communication infrastructure \((\text{match}, \text{matchO}, \text{matchS})\). Together with Compatible Communication, they will be sufficient for compositionality.

**Strong Communication:** For all \(t_1 \in \text{Trans}_1\) and \(t_2 \in \text{Trans}_2\), if the following hold:
- \(\text{isCom}_1 (\text{actOf}_1 t_1)\) and \(\text{isCom}_2 (\text{actOf}_2 t_2)\),
- \(\text{isObs}_1 t_1, \text{isObs}_2 t_2\) and \(\text{matchO} (\text{getObs}_1 t_1) (\text{getObs}_2 t_2)\),
- \(\text{isSec}_1 t_1\) and \(\text{isSec}_2 t_2\) imply \(\text{matchS} (\text{getSec}_1 t_1) (\text{getSec}_2 t_2)\),
- then \(\text{match} t_1 t_2\) 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 \(t_1 \in \text{Trans}_1\), \(\text{isCom}_1 (\text{actOf}_1 t_1)\) implies \(\text{isObs}_1 t_1\); and for all \(t_2 \in \text{Trans}_2\), \(\text{isCom}_2 (\text{actOf}_2 t_2)\) implies \(\text{isObs}_2 t_2\).

The property says that all communicating transitions are observable (i.e., \(\text{isObs}\) is true for them), although it does not say anything about what can actually be observed about them (via \(\text{getObs}\)).
Secret Polarization: For all \( t_2 \in \text{Trans}_2 \), isSec\( _2 \) \( t_2 \) implies isCom\( _2 \) (actOf\( _2 \) \( t_2 \)).

The property says that any \( A_2 \)-transition that is secret-producing must be a communicating transition, which means that only \( A_1 \) is able to produce secrets independently.

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

\[ \text{System Compositionality Theorem [8]} \]

Assume that the flow policies \( F_1 \) and \( F_2 \) and the communication infrastructure \( (\text{match}, \text{match}_0, \text{match}_5) \) satisfy all the above properties, namely Compatible, Strong and Observable Communication, and Secret Polarization. Moreover, assume \( A_1 \models F_1 \) and \( A_2 \models F_2 \). Then \( A_1 \times_{\text{match}} A_2 \models F_1 \times_{(\text{match}, \text{match}_0, \text{match}_5)} F_2 \).

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 \( F_1 \) and \( F_2 \), only users of one of the two component systems, \( A_1 \), 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 \( F'_1 \) and \( F'_2 \).

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

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

\(^1\) 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 \( A \models F \) and \( F' \leq F \), then \( A \models F' \).

In conclusion, one can use the System Compositional Theorem to obtain for the composite system \( A_1 \times^{\text{match}} A_2 \) a flow policy \( F = F_1 \times^{(\text{match}, \text{matchO}, \text{matchS})} F_2 \) with a strong bound, and the Transport Theorem to produce from this a perhaps weaker but more natural flow policy \( F' \) (for the same system \( A_1 \times^{\text{match}} A_2 \)). [8, App.A] gives more intuition on using the two theorems in tandem.

2.7.4 The \( n \)-ary case

The System Compositional Theorem generalizes quite smoothly from the binary to the \( n \)-ary case. Let \( A_k = (\text{State}_k, \text{Act}_k, \text{Out}_k, \text{istate}_k, \text{Trans}_k)_{k \in \{1, \ldots, n\}} \) be a family of \( n \) systems.

We fix, for each \( k, k' \) with \( k \neq k' \), a matching predicate \( \text{match}_{k,k'} : \text{Trans}_k \times \text{Trans}_{k'} \to \text{Bool} \). We write \( \text{match} \) for the family \( (\text{match}_{k,k'})_{k,k'} \) and \( \text{isCom}_{k,k'} : \text{Act}_k \to \text{Bool} \) for the corresponding notion of communication action (belonging to \( A_k \) and pertaining to communicating with \( A_{k'} \)). 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 \( \text{isCom}_{k,k''} \) and \( \text{isCom}_{k',k''} \) are disjoint, in that there exists no \( a \in \text{Act}_k \) such that \( \text{isCom}_{k,k''} a \) and \( \text{isCom}_{k',k''} a \). The match-communicating product of the family of systems \( (A_k)_{k \in \{1, \ldots, n\}} \), written \( \prod_{k \in \{1, \ldots, n\}}^{\text{match}} A_k \), generalizes of the binary case. Namely, it is the following system \( (\text{State}, \text{Act}, \text{Out}, \text{istate}, \text{Trans}) \):

\[
\begin{align*}
\text{State} &= \prod_{k \in \{1, \ldots, n\}} \text{State}_k; \text{so the states are families } (\sigma_k)_{k \in \{1, \ldots, n\}}, \text{or } (\sigma_k)_k \text{ for short}; \\
\text{Act} &= \sum_{k \in \{1, \ldots, n\}} \text{Act}_k + \sum_{k,k' \in \{1, \ldots, n\}, k \neq k'} \text{Act}_k \times \text{Act}_k; \text{we write } (i, a_i) \text{ for elements of the } i \text{th summand on the left (separate actions by component } A_i) \text{, and } ((i, a_i), (j, a_j)) \text{ for elements of the } (i, j) \text{th summand on the right (joint communicating actions by components } A_i \text{ and } A_j); \\
\text{Out} &= \sum_{k \in \{1, \ldots, n\}} \text{Out}_k + \sum_{k,k' \in \{1, \ldots, n\}, k \neq k'} \text{Out}_k \times \text{Out}_{k'} \text{ (similarly to } \text{Act}); \\
\text{istate} &= (\text{istate}_k)_{k \in \{1, \ldots, n\}}; \\
\text{Trans} &= \text{contains two kinds of transitions:}
\begin{itemize}
  \item for \( i \in \{1, \ldots, n\} \), separate } A_i \text{ transitions } ((\sigma_k)_k, (i, a_i), (i, o_{i,u_i}), (\sigma_k)_k[i := \sigma'_{i}]), \text{ where } (\sigma_i, a_i, o_{i,u_i}, \sigma'_i) \in \text{Trans}, \text{ and } \text{isCom}_{i,a_i}; \\
  \item for } i, j \in \{1, \ldots, n\} \text{ such that } i \neq j, \text{ communication transitions (between } A_i \text{ and } A_j) \\
((\sigma_k)_k, (i, a_i), (j, a_j)), ((i, o_{i,u_i}), (j, o_{j,u_j})), (\sigma_k)_k[i := \sigma'_{i}, j := \sigma'_{j}]), \text{ where } (\sigma_i, a_i, o_{i,u_i}, \sigma'_i) \in \text{Trans}, \text{ and } \text{match}_{i,j}(\sigma_i, a_i, o_{i,u_i}, \sigma'_i)(\sigma_j, a_j, o_{j,u_j}, \sigma'_j).
\end{itemize}
\]

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 \( \sigma_i \) to \( \sigma'_i \); and similarly for \( (\sigma_k)_k[i := \sigma'_i, j := \sigma'_j] \).

Given the flow policies \( F_k \) for the component systems \( A_k \) and the families of matching predicates for transitions, \( \text{match} = (\text{match}_{k,k'})_{k,k'} \), observations, \( \text{matchO} = (\text{matchO}_{k,k'})_{k,k'} \), and secrets, \( \text{matchS} = (\text{matchS}_{k,k'})_{k,k'} \), the product flow policy \( \prod_{k \in \{1, \ldots, n\}}^{\text{match}, \text{matchO}, \text{matchS}} F_k \) is defined as a straightforward generalization of the binary case. For example, its observation domain is \( \sum_{k \in \{1, \ldots, n\}} \text{Obs}_k + \sum_{k,k' \in \{1, \ldots, n\}, k \neq k'} \text{Obs}_k \times \text{Obs}_{k'} \), 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 \( B \) is defined from the component bounds: for all \((s_{k})_k, (s'_{k'})_k \in \prod_{k \in \{1, \ldots, n\}} \text{List} (\text{Sec}_k) \), if \( s \in \prod_{k \in \{1, \ldots, n\}}^{\text{matchS}} (s_k)_k \) and \( s' \in \prod_{k \in \{1, \ldots, n\}}^{\text{matchS}} (s'_{k'})_k \), then, for all \( k, \) \( B_k s_s s' \) holds - where \( \prod_{k \in \{1, \ldots, n\}}^{\text{matchS}} \) is the \( n \)-ary \( \text{matchS} \)-shuffle product operator, which applied to a family of lists of secrets \((s_k)_k \) 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 $k, k' \in \{1, \ldots, n\}$ with $k \neq i$ and for all $t \in \text{Trans}_k$, $\text{isSec}_k t$ implies $\text{isCom}_k(i, \text{actOf}_k t)$.

\[ \bigwedge_{k \in \{1, \ldots, n\}} A_k \models F_k \]

3.14 Bounded-Deducibility Security

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

<table>
<thead>
<tr>
<th>Secrets</th>
<th>Declassification Trigger</th>
<th>Declassification Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper</td>
<td>Some user in $G$ is one of the paper’s authors</td>
<td>Last uploaded version</td>
</tr>
<tr>
<td></td>
<td>Some user in $G$ is one of the paper’s authors or a PC member$^B$</td>
<td>Absence of any upload</td>
</tr>
<tr>
<td>Review</td>
<td>Some user in $G$ is the review’s author</td>
<td>Last edited version before Discussion and all the later versions</td>
</tr>
<tr>
<td></td>
<td>Some user in $G$ is the review’s author or a non-conflicted PC member$^D$</td>
<td>Last edited version before Notification</td>
</tr>
<tr>
<td></td>
<td>Some user in $G$ is the review’s author or a non-conflicted PC member$^D$ or the reviewed paper’s author$^N$</td>
<td>Absence of any edit</td>
</tr>
<tr>
<td>Discussion</td>
<td>Some user in $G$ is a non-conflicted PC member</td>
<td>Absence of any edit</td>
</tr>
<tr>
<td>Decision</td>
<td>Some user in $G$ is a non-conflicted PC member</td>
<td>Last edited version</td>
</tr>
<tr>
<td></td>
<td>Some user in $G$ is a non-conflicted PC member or a PC member$^N$ or the decided paper’s author$^N$</td>
<td>Absence of any edit</td>
</tr>
<tr>
<td>Reviewer assignment</td>
<td>Some user in $G$ is a non-conflicted PC member$^R$</td>
<td>Reviewers being non-conflicted PC members, and number of reviewers</td>
</tr>
<tr>
<td></td>
<td>Some user in $G$ is a non-conflicted PC member$^R$ or one of the reviewed paper’s authors$^N$</td>
<td>Reviewers being non-conflicted PC members</td>
</tr>
</tbody>
</table>

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 the 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.

<table>
<thead>
<tr>
<th>Secrets</th>
<th>Declassification Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content of a given post</td>
<td>Updates performed while or last before one of the following holds:</td>
</tr>
<tr>
<td></td>
<td>- Some user in $G$ is the admin, is the post owner or is friends with its owner</td>
</tr>
<tr>
<td></td>
<td>- The post is marked as public</td>
</tr>
<tr>
<td>Friendship status between two given users $U$ and $V$</td>
<td>Status changes performed while or last before the following holds:</td>
</tr>
<tr>
<td></td>
<td>- Some user in $G$ is the admin or is friends with $U$ or $V$</td>
</tr>
<tr>
<td>Friendship requests between two given users $U$ and $V$</td>
<td>Existence of accepted requests while or last before the following holds:</td>
</tr>
<tr>
<td></td>
<td>- Some user in $G$ is the admin or is friends with $U$ or $V$</td>
</tr>
</tbody>
</table>

### 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.

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

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 infrastructure is always the same, given by the actions and outputs of a fixed group $G$ of users.
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. \cite{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 \cite{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 \cite{26}, can be contrasted with language-based security \cite{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 \cite{44} in a system-based setting. It descends from the epistemic logic \cite{40} inspired tradition of modeling information-flow security, pioneered by Sutherland with Nondeducibility \cite{46} and continued with Halpern and O’Neill’s Secrecy Maintenance \cite{22} and with Askarov et al.’s Gradual Release \cite{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 \cite{19} and Rushby \cite{41}, which have been extensively studied as part of Mantel’s MAKS framework \cite{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 $s_2$ (provided the “attacker”, who controls the original trace $tr_1$, has produced all the original secrets $s_1$) – 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 \cite{27}. Several frameworks have been developed in various settings, e.g., event systems \cite{25}, reactive systems \cite{39} and process calculi \cite{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 \cite{28}. Others, such as Mantel’s MAKS framework \cite{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 \cite{15} and HyperCTL$^*$ \cite{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

3:20 Bounded-Deducibility Security