2021 IEEE 34th Computer Security Foundations Symposium (CSF)

On Compositional Information Flow Aware Refinement

Christoph Baumann*, Mads Dam†, Roberto Guanciale†, Hamed Nemati‡

*Ericsson Research Security
Kista, Sweden
christoph.baumann@ericsson.com

†Department of Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
{mfd, robertog}@kth.se

‡Helmholtz Center for
Information Security (CISPA)
Saarbrücken, Germany
hnnemati@cispa.saarland

***Abstract*—The concepts of *information flow security* and *refinement* are known to have had a troubled relationship ever since the seminal work of McLean. In this work we study refinements that support changes in data representation and semantics, including the addition of state variables that may induce new observational power or side channels. We propose a new epistemic approach to ignorance-preserving refinement where an abstract model is used as a specification of a system's permitted information flows, that may include the declassification of secret information. The core idea is to require that refinement steps must not induce observer knowledge that is not already available in the abstract model. Our study is set in the context of a class of shared variable multi-agent models similar to interpreted systems in epistemic logic. We demonstrate the expressiveness of our framework through a series of small examples and compare our approach to existing, stricter notions of information-flow secure refinement based on bisimulations and noninterference preservation. Interestingly, noninterference preservation is not supported "out of the box" in our setting, because refinement steps may introduce new secrets that are independent of secrets already present at abstract level. To support verification, we first introduce a "cube-shaped" unwinding condition related to conditions recently studied in the context of value-dependent noninterference, kernel verification, and secure compilation. A fundamental problem with ignorance-preserving refinement, caused by the support for general data and observation refinement, is that sequential composability is lost. We propose a solution based on relational pre- and post-conditions and illustrate its use together with unwinding on the oblivious RAM construction of Chung and Pass.**

I. Introduction

As is well known, refinement and information flow security is an uneasy combination [1], the so-called "refinement paradox". An abstract system specification of, e.g., a processor pipeline will leave many implementation details unspecified. A series of refinement steps will introduce many design decisions regarding the pipeline structure, for instance, concerning the number and ordering of pipeline stages and the use of speculation and out-of-order mechanisms. Each refinement step will eliminate underspecification and nondeterminism, but may inadvertently introduce new types of observation, for instance through caches, that can cause undesired information leaks via side channels, as seen in the new Spectre family of attacks [2].

Formally, the phenomenon manifests itself in that the different forms of trace inclusion or simulation ordering used in refinement to account for functional correctness will in general cause the information content of each observation to increase as the system becomes more precisely determined.

Yet for the formal analysis of real systems and architectures a modular approach based on some form of refinement seems essential, since it would allow to handle different security threats at different abstraction layers and then to compose the analysis results to provide security at system level. Among the minimum set of features we would expect from such an approach are the following:

REQ1 Support for various forms of structural refinement including data representation and the addition of new state variables that may introduce discriminating power not available at high abstraction level.

REQ2 Allowing reduction of underspecification and nondeterminism, where this obeys information flow constraints.

REQ3 Covering the intentional leakage of secret information.

REQ4 The ability to compose refinements component-wise and in a top-down manner, guaranteeing the desired information flow constraints.

REQ5 A clear semantic justification in terms of the knowledge an observer gains from monitoring system runs.

REQ6 Support of effective reasoning methods.

A number of authors have attempted to develop approaches to information flow aware refinement. In Sect.XI we review some of the key contributions w.r.t. these requirements.

An essential problem in information flow preserving refinement is that observers learn, i.e., accumulate knowledge, during execution. The requirement is that observers do not learn more at refined level than what they learn when executing at abstract level, but what is learned is highly context-dependent. Consider the well-known example of credit card numbers. A security requirement may be that no more than four digits of the credit card number may be revealed in sessions involving third parties. Any implementation revealing no more than four digits will meet the requirements and be secure when executed in isolation. On the other hand, implementations that choose to reveal different digits are insecure when executed in sequence. This illustrates the challenges posed by underspecification and nondeterminism (REQ2), but similar phenomena arise in the context of secret-dependent data and observation refinement. For instance, consider a one time pad, where a uniformly random key provides information-theoretically secure protection of a secret when used once, but not when used twice.

In this work, we tackle these challenges by identifying and preserving an attacker's *ignorance* of confidential data,

DOI 10.1109/CSF51468.2021.00010

i.e., their uncertainty about the possible values of secret data, which refinement must preserve or increase (but not reduce) to be confidentiality-preserving. While the concept of ignorance has been studied in the context of information flow [3] and refinement [4] before, our account of ignorance-preserving refinement, IPR, is to our knowledge the first to achieve the goals stated above, using a modular approach that is agnostic to the information flow policies to be preserved. In particular, we cover both classical noninterference of sensitive and public data and the intentional leakage of confidential information.

The key idea is to use observer ignorance in the abstract model as a specification of the permitted information flow, and then to ensure that any allowed refinement induces an upper bound on the corresponding flow in the refined model. We achieve this using epistemic logic. Our study is cast in a framework of synchronous multi-agent systems, akin to a shared-memory version of interpreted systems in the sense of [5]. We present our modeling framework in Sect. II and our account of refinement and IPR in Sects. III and IV. We illustrate the usage of IPR and our modeling framework in Sect. V through a series of small examples, using a small modeling language that may have independent interest. Interestingly, IPR does not imply noninterference preservation "out of the box". Ignorance-preserving refinement only addresses secrets that are already represented at abstract level, but a concrete model may introduce new types of unrelated secrets that hence are not protected by IPR, as discussed in Sect. VI. Turning to verification, in Sect. VII we present an unwinding condition for IPR. The condition is related to "cube-shaped" unwindings that have been studied in the recent past in the context of value-dependent noninterference [6], kernel verification [7, p. 186], and secure compilation [8] and are discussed thoroughly in Sect. XI. In Sect. VIII we discuss the problem of compositionality in more detail, and in Sect. IX we present our solution based on a kind of relational Hoare logic [9] lifted to refinements. In Sect. X we apply our methodology to a simplified ORAM design to illustrate the combination of unwinding and relational verification in an example that requires extensive use of composition. Finally, we survey related work in the area (Sect. XI) and conclude (Sect. XII).

II. Basics, Models, Epistemics

The models we consider in this work are extended transition systems equipped with a synchronous multi-process structure, a shared store, and an observation function. A *model* $\mathcal{M} = (S_0, S, \rightarrow, Obs_p)$ is a transition system with S the set of states, $S_0 \subseteq S$ the set of initial states, $\rightarrow \subseteq S \times S$ the transition relation, and Obs_p an observation function, where the components are structured as follows:

- States $s \in S$ are pairs $(c, d) \in C \times D$ of a *control state* c and a *store*, d. We generally use the notation $s.c$ and $s.d$ for the left and right projections on s.
- A *control state* is a mapping $c \in C = PId \rightarrow PC$ of process (thread) identifiers $p \in PId$ to thread control states $pc \in PC$.
- A *store* is a mapping $d \in D = Var \rightarrow Val$ of variables from set Var to values in set Val.
- An *initial state* $s_0 \in S_0 = \{(init, d_0) \mid d_0 \in D\}$ is a pair where $init \in C$ is a designated, i.e., unique, starting control state. Computations may start with any store.
- A *final state* is any state s in which no transition starts, i.e., for which no s' exists such that $s \rightarrow s'$.
- An *observation function* $Obs_p : S \rightarrow \mathcal{O}_p$ returns the observations $\mathcal{O}_p$ a process p can make in a given state.

In what follows we restrict our attention to scenarios where a process p may observe a subset of variables $Var_{Obs}(p) \subseteq Var$ along with its current control state, i.e., $\mathcal{O}_p = (Var_{Obs}(p) \rightarrow D) \times PC$. Thus, p's *observation* in state s is a pair $Obs_p(s) = (\lambda x \in Var_{Obs}(p).\ s.d(x), s.c(p))$. We let α, β range over observations and write $s \xrightarrow[p]{\alpha} s'$, if $Obs_p(s') = \alpha$ and $s \rightarrow s'$.

Models as defined above describe synchronously evolving collections of processes that operate on a single shared state. Each transition represents a synchronous round of computation, with each such round a collection of atomic state transitions, one for each process. Besides the processes' local control, transitions are conditioned on, and affect, a shared global state. This provides a very general setting in which to model a wide variety of computational phenomena including time, asynchrony, and multi-way synchronization at different levels of abstraction, cf. the use of three-way synchronization in the abstract model of Example 1 below. By decoupling observations from state transitions we also make it easy to, for instance, add a passive process to represent an external synchronous observer of a selected set of state variables.

For model description we use a process pseudo-language with `|>` as a round separator and ordinary multi-assignments separated by `;` to build the component atomic transitions. These multi-assignments are required to be terminating, and write-write and read-write race free, in order to preserve the per-round atomicity abstraction. The synchronous processes are composed in parallel via `||`. In our examples we restrict attention to static process configurations, however our underlying semantics allows to model dynamic process networks as well, e.g., through an explicit thread management process.

This model is a variant of the interpreted systems model of Fagin et al [5]. Interpreted systems have agents (threads) with local states that can be affected by other agents through a global transition relation, as here. As for interpreted systems, adversaries can in our framework be modeled as nondeterministic environment processes with prescribed observability properties. Models as used in this paper adds to the interpreted systems model a shared store, and an explicit observation function. The former is convenient for the types of applications we address. Observation functions Obs_p allow to encode both statically determined sets of variables observed by process p and dynamically varying notions of observability.

Example 1 (Three Party Protocol Specification)**.** In Fig. 1 we show an abstract specification of a three-party additive shared secret protocol. The protocol operates over natural numbers modulo some n. Each process receives as parameter the sum of the other processes local variable `z` and uses this to compute the sum of the `z`'s while keeping the value of the process's local `z` private. The initial value of each `z` (and `res`) is an unknown element of `Val` $= \{0, \ldots, n-1\}$.

```
system three_party_spec
  type Val = [0..n-1]
  process proc(sum : Val)
    var z, res : Val
    do res := z + sum |>
  configuration
    pid0 = proc(pid1.z + pid2.z) ||
    pid1 = proc(pid2.z + pid0.z) ||
    pid2 = proc(pid0.z + pid1.z)
```

Fig. 1. Abstract specification of three-party additive secret sharing protocol.

Local variables are always observable to the process in which they are declared. In general, observability of variables is regulated by a special "observability" declaration. Unobservability, however, does not necessarily mean that the variable cannot be read. This is commonly used in specifications to implement ideal functionality such as here where each process reads (here: receives as parameter) the sum of the other processes `z` variables without actually observing them individually. Evidently, one key goal of refinement is to produce concrete models that do not appeal to such ideal functionality for their realization.

The system operates in a standard synchronous fashion: At the start of each round the state of each variable used as input for the new round is recorded, then the new values of all state variables to be updated are computed, and after this the actual update takes place. Thus, the model in Fig. 1 executes exactly one round after which the processes terminate.

We explain how to convert the specification in Fig. 1 to a model. Formally, the protocol has processes p_0, p_1, p_2, each with two local control states, say pc_0, pc_{end} where pc_0 is initial and pc_{end} is final. The initial store for each process has `res` and `z` initialized to some value in `Val`, say $s_0.d(p_0.\mathtt{z}) = 4$, $s_0.d(p_1.\mathtt{z}) = 7$, and $s_0.d(p_2.\mathtt{z}) = 5$. These variables cannot be directly observed by the other processes, as explained above. For the transition relation the p_i execute the assignments synchronously in parallel and terminate. For the sake of the example let modulus $\mathtt{n} = 9$ and observation $\alpha = ([p_2.\mathtt{z} \mapsto 5][p_2.\mathtt{res} \mapsto 7], pc_{end})$. Since p_2's local variables `z` and `res` are the only variables observable to p_2 we obtain that $s_0 \xrightarrow{\alpha}_{p_2} s_1$ with $s_1.d(p_i.\mathtt{res}) = 7$ for all i.

In the context of a given transition system, a *run* $\rho \in \mathcal{R}(\mathcal{M})$ is a finite sequence $s_0 \cdots s_n$ such that $s_0 \in S_0$ and $s_{i-1} \to s_i$ for all $i > 0$ for which s_i is defined. The i'th state of ρ, $\rho(i)$, is s_i, and $|\rho| \in \mathbb{N}$ is the length of ρ. A *complete* run is one that cannot be extended, i.e. there is no s such that $\rho(|\rho|-1) \to s$. In that case $lst(\rho) = \rho(|\rho| - 1)$ is final.

The notions of observation trace, observation equivalence, and the epistemic notions of knowledge and its dual, ignorance, are standard. First, two states s_1, s_2 are observationally equivalent as seen by process p, $s_1 \sim_p s_2$, if p has the same observations in the two states, $Obs_p(s_1) = Obs_p(s_2)$. A p-observation trace is the sequence of observation of some run ρ, i.e. $Obs_p(\rho) = Obs_p(\rho(0)) \cdots Obs_p(\rho(|\rho| - 1))$. A complete trace is a trace of a complete run, and the runs ρ_1 and ρ_2 are p-observation equivalent $\rho_1 \sim_p \rho_2$, if p's observations in ρ_1 are the same as p's observations in ρ_2, i.e. $Obs_p(\rho_1) = Obs_p(\rho_2)$.

In the context of a given model $\mathcal{M}$ we view a *property* as a set $\phi \subseteq \mathcal{R}(\mathcal{M})$ representing the set of runs that are consistent with the observations made so far in the computation. This allows to define the standard epistemic modality $K_p\phi$ of perfect recall knowledge and its De Morgan dual $I_p\phi$ of "ignorance" on properties ϕ in the following way:

- $\rho \in K_p\phi$, if for all $\rho' \in \mathcal{R}(\mathcal{M})$, if $\rho' \sim_p \rho$ then $\rho' \in \phi$.
- $\rho \in I_p\phi$, if there is $\rho' \in \phi$ such that $\rho' \sim_p \rho$.

The set $I_p\phi$ is the set of runs ρ that are "compatible" with some ρ' in ϕ in the sense that p cannot tell ρ' from ρ. Accordingly, we call a set ϕ a *p-ignorance set*, or just ignorance set if p is understood from the context, if ϕ is closed under $\sim_p$. Dually, if $\rho \in K_p\phi$ then ϕ includes all runs ρ' for which $\rho \sim_p \rho'$, i.e. it is inconceivable for p as a perfect recall observer that ρ does not occur in ϕ. The K_p operator is the standard S5 epistemic modality of knowledge. From the definition we immediately obtain standard modal K and S5 properties like $\phi \subseteq I_p\phi$, $I_p(\phi \cup \psi) = I_p\phi \cup I_p\psi$, $\mathrm{T} : K_p\phi \subseteq \phi$, $\mathrm{B} : \phi \subseteq K_p(I_p\phi)$, $4 : K_p\phi \subseteq K_p(K_p\phi)$, as well as the monotonicity of ignorance: $\phi \subseteq \psi \Rightarrow I_p\phi \subseteq I_p\psi$.

Since I_p is closed under $\sim_p$ and distributive w.r.t. $\cup$, to compute the ignorance of an observer p, it suffices to consider only the cases $\phi = \{\rho\}$ for some representative ρ of a given observation trace. Still, we stick to the general case below.

Example 2. Let ρ be the run s_0s_1 of Example 1. Then $I_{p_2}\{\rho\}$ is the set of all runs ρ' that agree with ρ on the initial state assignments to $p_2.\mathtt{z}$ and $p_2.\mathtt{res}$, and the final state assignment to $p_2.\mathtt{res}$. On the other hand, $K_{p_2}\{\rho\}$ is empty, and $K_{p_1}\phi$ is nonempty only if ϕ is closed under any substitution that preserves the initial and final state assignments of the local variables of p_1, and in particular keeps the sum $p_0.\mathtt{z} + p_1.\mathtt{z} + p_2.\mathtt{z}$ constant (modulo n).

III. Refinement

We now compare two models, an abstract model $\mathcal{M}_a$ with states $s \in S$, runs $\mathcal{R}_a$, and transition relation $\to_a$, typically used to predicate the desired behavior, and a concrete, or implementation model $\mathcal{M}_c$, with states $t \in T$, runs $\mathcal{R}_c$, and relation $\to_c$, that is used to describe how the abstract behavior is realized. We write ρ, ϕ for (sets of) abstract runs and σ, ψ for concrete ones. The two models are connected by a *refinement relation* $s \Downarrow t$, or function $\lceil t \rceil = s$, which for each concrete state t produces one or more abstract states s, which t is intended to refine. We refer to refinement relations of the latter form as *functional*. In this section we set out the basic properties we assume of refinements before, in Section IV, we turn to information flow preservation.

A large section of work in the refinement domain is based on the notion of simulation, cf. [10, 11, 12], which in the present synchronous setting can be cast as follows.

Definition 1 (Simulation, Observation Preservation)**.**

1) *The refinement relation $\Downarrow$ is a* simulation *of $\to_c$ by $\to_a$, if $s \Downarrow t \to_c t'$ implies $s \to_a s' \Downarrow t'$ for some s'. Moreover, if $s \Downarrow t$, then s is initial or final if and only if t is.*
2) *The relation $\Downarrow$ preserves p-observations, if whenever $s \Downarrow t$ it holds that $Obs_p(t) = Obs_p(s)$.*

In the functional case the equivalent condition to 1.1 is that $t \to t'$ implies $\lceil t \rceil \to \lceil t' \rceil$. The simulation property 1.1 allows abstractions to be point-wise extended to runs by $\rho = s_0 \cdots s_n \Downarrow t_0 \cdots t_n = \sigma$ if $s_i \Downarrow t_i$ for all $i : 0 \leq i \leq |\rho| = |\sigma| = n + 1$ and for sets ϕ, ψ, $\phi \Downarrow \psi$, if for all $\rho \in \phi$ there is $\sigma \in \psi$ such that $\rho \Downarrow \sigma$, and vice versa, for all $\sigma \in \psi$ there is $\rho \in \phi$ such that $\rho \Downarrow \sigma$. In the functional case we define $\lceil s_0 \cdots s_n \rceil = \lceil s_0 \rceil \cdots \lceil s_n \rceil$ for abstracting concrete runs. Moreover, we denote by $\Uparrow$ the direct image of $\Downarrow^{-1}$, i.e., $\psi\Uparrow = \{\rho \mid \exists \sigma \in \psi.\ \rho \Downarrow \sigma\}$ and obtain:

Corollary 1. *If $\Downarrow$ is a simulation of the concrete model, then 1) $\phi \Downarrow \psi \Rightarrow \phi \subseteq \psi\Uparrow$ and 2) $(\psi\Uparrow) \Downarrow \psi$.* □

For functional correctness, refinement usually requires both simulation and observation preservation. In this work we rely on the simulation condition as the crucial hook needed to relate computations at abstract and concrete level. Preservation of observations in the sense of 1.2 is used, e.g., in [13] but its necessity appears less clear. For ignorance preservation the key issue is preservation of observable distinctions and not necessarily the observations themselves. Indeed, as we show in this paper it is perfectly possible to conceive of meaningful refinement-like relations that preserve observation distinctions but not the observations themselves.

The key is to shift attention from preservation of observations to preservation of distinctions. In particular we distinguish observational equivalence relation $\sim_p$ on the abstract model from its counterpart (written $\approx_p$) on the concrete model. This motivates the following well-formedness condition:

Definition 2 (Well-formedness)**.** *The refinement relation $\Downarrow$ is well-formed, if $s_1 \Downarrow t_1 \approx_p t_2$ and $s_2 \Downarrow t_2$ implies $s_1 \sim_p s_2$.*

For functional refinement relations this becomes the condition that $t_1 \approx_p t_2$ implies $\lceil t_1 \rceil \sim_p \lceil t_2 \rceil$.

Well-formedness reflects the expectation that information content of models should generally increase under refinement. Then, if two abstract states are observationally distinct, we should expect this discriminating power to be preserved to concrete level. On the other hand, reflecting the increasing information content we should not necessarily expect indistinguishability to be preserved. We obtain:

Proposition 1. *Suppose that the simulation $\Downarrow$ is well-formed. Then:*

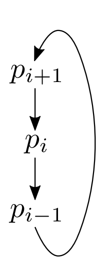
1) If $\rho_1 \Downarrow \sigma_1 \approx_p \sigma_2$ and $\rho_2 \Downarrow \sigma_2$ then $\rho_1 \sim_p \rho_2$.
2) Suppose $\phi \Downarrow \psi$, then $I_p\phi \supseteq (I_p\psi)\Uparrow$.

Proof. 1) Follows immediately from Def. 2. 2) If $\rho \in (I_p\psi)\Uparrow$ then we find $\sigma \in I_p\psi$ such that $\rho \Downarrow \sigma$ and a $\sigma' \in \psi$ such that $\sigma \approx_p \sigma'$. By $\phi \Downarrow \psi$ there is $\rho' \in \phi$ with $\rho' \Downarrow \sigma'$ and by well-formedness $\rho \sim_p \rho'$. But then $\rho \in I_p\phi$. □

In other words it follows directly from well-formedness and the simulation property that ignorance is preserved from concrete to abstract level. We define:

Definition 3 (Refinement)**.** *The refinement relation $\Downarrow$ is a* refinement, *if $\Downarrow$ is well-formed and a simulation.*

```
system three_party_impl {
  type Val = [0..n-1]
  process proc(in : Val)
    var x, y, out, res: Val
    do
      out := y |>
      out := in + x + y |>
      out := in + x + y |>
      res := in + x |>
  configuration
    pid0 = proc(pid1.out) ||
    pid1 = proc(pid2.out) ||
    pid2 = proc(pid0.out)
```



j	$in_{j,i}$
1	$y_{0,i+1}$
2	$y_{0,i-1}+z_{i+1}$
3	$y_{0,i}+z_{i-1}+z_{i+1}$

Fig. 2. Three-party protocol implementation.

IV. Ignorance Preservation

While Prop. 1.2 is useful, our interest, however, is in preservation of ignorance in the opposite direction. To arrive at a suitable definition we first introduce a Chaum-style additive secret sharing implementation [14] of the three party protocol.

Example 3 (Three Party Protocol Implementation)**.** The concrete version of Example 1 is shown in Fig. 2. At concrete level each p_i splits z into shares x, y such that $\mathrm{z} = \mathrm{x}+\mathrm{y}$ modulo n. In addition the configuration of the processes causes the local variable `out` for each p_i to be bound "counterclockwise" to the local parameter `in` of p_{i-1}. The protocol starts with p_0, p_1, p_2 in their initial control states, and with arbitrary data state assignments. Each process then executes its program in synchronous rounds, separated by `|>`.

We show that there is a refinement according to Def. 3 relating the abstract and concrete models. The refinement is as should be expected. The abstraction represents each z as $\mathrm{x} + \mathrm{y}$ for some choice of x and y, and `res` at abstract level as itself at concrete level. Initial states at concrete level are mapped to initial states at abstract level. For the transitions, each round at abstract level corresponds to four rounds at concrete level, achieved simply by deriving from the concrete transition relation $\to_c$ the relation $\to^4 = \to_c \circ \to_c \circ \to_c \circ \to_c$. Observations of process p_i on concrete level are the concatenation of p_i's local variable assignments for each of the four transitions, including the inputs.

Well-formedness is easy to check. For instance for initial states s_1, s_2, t_1, t_2 and $i \in \{0, 1, 2\}$ we get:

$$\begin{aligned} s_1(p_i.\mathrm{z}) &= t_1(p_i.\mathrm{x}) + t_1(p_i.\mathrm{y}) && (\text{if } s_1 \Downarrow t_1) \\ &= t_2(p_i.\mathrm{x}) + t_2(p_i.\mathrm{y}) && (\text{if } t_1 \approx_{p_i} t_2) \\ &= s_2(p_i.\mathrm{z}) && (\text{if } s_2 \Downarrow t_2) \end{aligned}$$

and similarly for `res`. The same argument applies for the final states, since x, y, z are assigned at neither abstract nor concrete level. For the simulation property let $s \Downarrow t$, $s \to s'$, and $t = t_0 \to \cdots t_4$. Let also $in_{j,i} = t_j(p_i.\mathtt{in})$ and similarly for $x_{j,i}$, $y_{j,i}$, $res_{j,i}$. Similarly, let $z_i = s(p_i.\mathrm{z})$ and $res'_i = s'(p_i.\mathtt{res})$. To establish the simulation, it suffices to show that $res'_i = res_{4,i}$ for all $i \in \{0, 1, 2\}$ (cf. Fig. 2).

$$\begin{aligned} res_{4,i} &= in_{3,i} + x_{3,i} = in_{3,i} + x_{0,i} \\ &= y_{0,i} + z_{i-1} + z_{i+1} + x_{0,i} = z_i + z_{i+1} + z_{i-1} = res'_i \end{aligned}$$

From the point of view of confidentiality preservation the problem with the above argument is that it does not address information flow. In particular, it does not show if some information is leaked in some intermediate state. One key idea for confidentiality preservation, proposed by Morgan [4], is to compare ignorance at abstract with ignorance at concrete level: If the ignorance at concrete level is "at least as high as" (in [4]: a superset of) the ignorance at abstract level, no more information is learned by executing the protocol at concrete level than what is learned by executing the ideal functionality.

However, ignorance at abstract and concrete levels is not readily comparable, as in our setting (as opposed to [4]) the state spaces related by the refinement are different. In general, refinement will reduce nondeterminism and add observational power by implementation choices, e.g., for data representation. For instance, in Example 3, p_i learns the value of $p_{i+1}.\mathtt{y}$ in round 1. Nevertheless, reflecting our view of the abstract model as specifying the desired information flow properties, all information relevant for the analysis of information flow preservation is available already at abstract level. Thus we can use the refinement relation to push epistemic properties between the abstract and concrete levels, as follows:

Definition 4 (Ignorance-Preserving Refinement, IPR)**.** *The refinement $\Downarrow$ is p-ignorance-preserving, if $\Downarrow$ is a well-formed simulation such that $\phi \Downarrow \psi$ implies $I_p\phi \Downarrow I_p\psi$.*

We relativize ignorance preservation to the processes p since this allows to use different abstraction functions for each p, reflecting the potentially different views each process may have of the refinement. It becomes clear that Def. 4 is the desired property if we consider an equivalent formulation:

Proposition 2. *The refinement $\Downarrow$ is p-ignorance-preserving, iff $\phi \Downarrow \psi$ implies $I_p\phi = (I_p\psi)\Uparrow$.*

Proof. By Cor. 1.1, $I_p\phi \Downarrow I_p\psi$ implies $I_p\phi \subseteq (I_p\psi)\Uparrow$ and by well-formedness (Prop. 1.2) $I_p\phi = (I_p\psi)\Uparrow$. The other direction follows directly via $((I_p\psi)\Uparrow) \Downarrow I_p\psi$ by Cor. 1.2. □

Thus, IPR means that we have the same ignorance for observer p on both levels, when viewed in terms of the abstract model. In particular, a concrete model observer p cannot distinguish more behaviors than possible on the abstract model, when "re-abstracting" the set of indistinguishable concrete runs. The following is a useful sufficient and necessary condition for ignorance-preserving refinement:

Definition 5 (Paired Refinement)**.** *The refinement $\Downarrow$ is paired, if for all ρ, ρ', σ':*

$$\textit{If } \rho \sim_p \rho' \Downarrow \sigma' \textit{ then exists } \sigma \textit{ s.t. } \rho \Downarrow \sigma \approx_p \sigma'. \qquad (*)$$

Proposition 3. *The paired refinement condition $(*)$ holds for refinement $\Downarrow$ if, and only if, $\Downarrow$ is p-ignorance-preserving.*

Proof. The implication IPR$\Rightarrow$ $(*)$ follows directly from $I_p\phi \Downarrow I_p\psi$ for $\phi = \{\rho'\}$ and $\psi = \{\sigma'\}$. For direction $(*) \Rightarrow$ IPR, assume that $\phi \Downarrow \psi$ for the refinement $\Downarrow$. For any $\rho \in I_p\phi$ we find $\rho' \in \phi$ such that $\rho \sim_p \rho'$ and a $\sigma' \in \psi$ such that $\rho' \Downarrow \sigma'$. By $(*)$ we find σ s.t. $\rho \Downarrow \sigma$ and $\sigma \approx_p \sigma'$, i.e. $\sigma \in I_p\psi$. Conversely, if $\sigma \in I_p\psi$ then we find $\sigma' \in \psi$ such that $\sigma \approx_p \sigma'$ and then a $\rho' \in \phi$ such that $\rho' \Downarrow \sigma'$. By the simulation property we find ρ such that $\rho \Downarrow \sigma$ and then by well-formedness, $\rho \sim_p \rho'$, i.e. $\rho \in I_p\phi$, as desired. □

```
system abstract  // concrete
  observable observe : nat
  process p
    var h : nat
    do observe := random(nat) |>  // ... := h
  configuration p
```

Fig. 3. Example: Confidentiality nonpreservation.

Intuitively, $(*)$ requires that for each pair of indistinguishable abstract runs, if one of them is implemented, so is the other one and the corresponding concrete runs are indistinguishable as well.

For Example 3, $s \Downarrow t$ if the abstract state s and the concrete state t for each p_i agree on their assignments to x, y, z (such that $\mathtt{z} = \mathtt{x} + \mathtt{y}$), and res, and that z, resp. x and y are constant in either system. At abstract level, $\rho \sim_{p_i} \rho'$ if for p_i, at each state the local assignments to z and res agree, and $p_{i-1}.\mathtt{z} + p_{i+1}.\mathtt{z}$ agree. At concrete level, $\sigma \approx_{p_i} \sigma'$ if, for p_i, the initial assignment to the local variables agree, and the values of $p_{i+1}.\mathtt{y}$, $p_{i+1}.\mathtt{x} + p_{i-1}.\mathtt{y}$, and $p_{i-1}.\mathtt{x}$ are the same in both runs, as the latter terms can be observed from (subtracting) the inputs received during the different rounds.

We see that if $\rho \sim_p \rho' \Downarrow \sigma'$ then by suitably adjusting $p_{i+1}.\mathtt{x}$ and $p_{i-1}.\mathtt{y}$ we can construct σ such that $\rho \Downarrow \sigma \approx_{p_i} \sigma'$. But then it follows directly from Prop. 3 that:

Theorem 1. *The refinement for the three-party secret sharing protocol is p_i-ignorance-preserving.*

Finally, for the sake of completeness, we point out that not all refinements are ignorance-preserving. Consider the classic insecure refinement in Fig. 3 where a random choice is replaced with leaking a secret variable h. The observability declaration in Fig. 3 can be viewed as shorthand for an additional process env containing the local variable observe, which in the single round of the example is assigned a random number in the abstract specification and h in the concrete. Let the refinement $\Downarrow$ be the identity on h and observe. It is trivial to see that $\Downarrow$ satisfies the simulation and well-formedness conditions and that it fails $(*)$. This does not mean that no IPR from abstract to concrete exists: If $\Downarrow$ only relates observe by identity, this is a valid IPR but not a very useful one, as it does not implement the abstract secret h.

V. Examples

We show some basic examples to illustrate different aspects of ignorance-preserving refinement.

Example 4 (A trivial top specification)**.** Figure 4 shows an abstract specification of a trivial system topSpec with a single control state and no local variables. The system topSpec lacks secrets altogether. From the point of view of ignorance-preservation it is therefore degenerate: There is no ignorance to preserve under refinement. The fact that for implLeak, x is private and leaked to the environment through the variable

```
system topSpec
  process p
    repeat skip |>
  configuration p

system implLeak
  observable observe : nat
  process p
    var x : nat
    repeat observe := x |>
  configuration p
```

Fig. 4. Example: A trivial top specification.

observe is irrelevant, as x is not correlated with any information present at abstract level. But then *any* system, including for instance implLeak, should be considered a legitimate refinement. To show that implLeak is an IPR of topSpec let $\lceil t \rceil = s$ where s is the unique state of the abstract model. The abstraction is trivially well-formed and the simulation condition holds (but not observation preservation). Finally, if $\rho_1 \sim_p \rho_2 = \lceil \sigma_2 \rceil$ then also $\rho_1 = \lceil \sigma_2 \rceil$, which shows ignorance preservation, i.e., IPR allows to pick an indistinguishable run that leaks the same value of x.

In other words, for an information flow constraint to be guaranteed by IPR at concrete level, the constraint must be represented in some form already in the abstract model. Nevertheless, it may be necessary for a refinement step to introduce additional secrets in the concrete model. Whether IPR guarantees the confidentiality of such secrets depends on their relationship to the secrets on the abstract level and the observations. We will come back to this issue in Sect. VI.

Example 5 (Scheduler)**.** The next example illustrates reduction of nondeterminism. An abstract nondeterministic scheduler of processes $p(0)$, $p(1)$, $p(2)$ is given in Fig. 5. The processes take turns updating a shared (observable) variable v, using a global random seed k that should be hidden from the environment which observes the sequence of v's produced. The concrete version (not shown) is a round-robin scheduler obtained from Fig. 5 by replacing the random assignment in sched with active := (active + 1)%3. As in the earlier examples the processes execute synchronously in parallel, either an assignment or a guard, and the only observable variable is v. Thus, an abstract observation trace for the abstract scheduler will be a trace of the form $v_0 v_1 \ldots v_n \ldots$ where $v_{i+1} = \texttt{work}(v_i, k_i)$ and k_i is the i'th random number generated globally.

The refinement $\Downarrow$ relates variables s.v to t.v, s.k to t.k, and s.active to any t.active. It is clear that the simulation and well-formedness conditions hold for $\Downarrow$, and for ignorance preservation it suffices to see that a trace of v's can be produced at abstract iff it can be produced at concrete level.

```
system scheduler(range : nat)
  var active, k : nat
  fun work() : nat =
    (do_something(v, k), random(range))
  observable v : nat

  process p(i : nat)
    repeat
      if active = i
      then (v, k) := work(v, k) |>
      else skip |>

  process sched(n : nat) =
    repeat active := random(n) |>

  configuration
    p(0) || p(1) || p(2) || sched(3)
```

Fig. 5. Abstract scheduler.

```
system asynchIdealProtocol
  observable count : nat init 0

  process sender
    var s : byte stream
    var put : byte
    repeat
      put := hd(s) ; s := tl(s) ; count++ |>>

  process receiver
    var get : byte
    repeat get := sender.put |>>

  configuration sender >> receiver
```

Fig. 6. Asynchronous ideal protocol model.

Example 6 (Asynchronous Communication with One Time Pad)**.** We consider a simple asynchronous producer-consumer system with communication protected using a one time pad. The ideal protocol specification (Fig. 6) uses the asynchronous sequencing operator |>> defined using the delay operator:

- delay(c ; p) = delay(p)
- delay(c |>>) = delay(c |>) = skip |>
- delay(c |> p) = skip |> delay(p)
- p >> q = (p ; delay(q)) || (delay(p) ; q)

The sender contains an unbounded byte stream s, elementwise passed to the receiver (using variable put), where an external observer may count the number of transactions (represented explicitly in the ideal model by history variable count). The expression sender >> receiver signifies that both processes take turns executing a single synchronous step each. The abstract delay operation can be implemented using a simple binary semaphore, which is toggled by the sender and reset by the receiver for flow control.

At concrete level (Fig. 7) the input bytes are encrypted (XOR-ed) using a random key r fetched from a key generator key_gen in each round by both sender and receiver. The observable state at concrete level is the round count along with the transmitted ciphertext. The refinement $s \Downarrow t$ relates the receivers with each other, and sender at abstract level with key_gen >> sender at concrete level. For the variables, relation $\Downarrow$ acts as the identity on sender.s, sender.count, receiver.get, and requires t.sender.put $\oplus$ t.key_gen.r to be identical to s.put, where $\oplus$ is XOR. Then well-formedness holds trivially and preservation of ignorance follows from the security of one time pad. Given a sequence of values v, encrypted on the concrete level using key sequence r, then for any other sequence v' we can always find r' such that $v \oplus r = v' \oplus r'$.

```
system asynchOnetimepadProtocol
  observable put : byte, count : nat = 0

  process key_gen
    var r : byte = 0
    repeat r := randomByte() |>>

  process sender
    var s : byte stream
    repeat
      put := key_gen.r xor hd(s) ;
      s := tl(s) ; count++ |>>

  process receiver
    var get : byte
    repeat get := key_gen.r xor sender.put |>>

  configuration
    key_gen >> sender >> receiver
```

Fig. 7. Asynchronous one time pad protocol.

VI. Conservativity

Viewed from afar there are two established approaches in the literature to prove ignorance preservation, either by appealing to some sort of behavioral equivalence or preorder in the sense of process algebra, or by restricting attention to prior established confidentiality properties.

In this section we cover examples of both approaches, namely bisimulation and noninterference, NI. We first address bisimulation, adapted to the present setting by assuming that the variables at abstract and concrete levels are the same.

Definition 6 (Bisimulation)**.** *The refinement $\Downarrow$ is a bisimulation (for p) if whenever $s \Downarrow t$ then*

1) *$Obs_p(s) = Obs_p(t)$ (denoted $s \simeq_p t$).*
2) *If $s \xrightarrow{\alpha_p} s'$, then there is t' such that $t \xrightarrow{\alpha_p} t'$ and $s' \Downarrow t'$.*
3) *if $t \xrightarrow{\alpha_p} t'$, then there is s' such that $s \xrightarrow{\alpha_p} s'$ and $s' \Downarrow t'$.*

Proposition 4. *If $\Downarrow$ is a bisimulation then $\Downarrow$ is an IPR.*

Proof. Using Prop. 3, if $\rho_0 \sim_p \rho_1 \Downarrow \sigma_1$, then $\rho_1 \simeq_p \sigma_1$ by 6.1, hence also $\rho_0 \simeq_p \sigma_1$. By 6.2 we find σ_0 with $\rho_0 \Downarrow \sigma_0$, hence $\rho_0 \simeq_p \sigma_0$ by 6.1. Thus $\sigma_0 \approx_p \sigma_1$ and $(*)$ holds. □

For noninterference, a reasonable account of NI in the present state-based setting is the following:

Definition 7 (Noninterference)**.** *A model $\mathcal{M}$ is* noninterferent *for process p if for all runs $\rho_0 \in \mathcal{R}(\mathcal{M})$, if $\rho_0(0) \sim_p s_1$ then there exists a $\rho_1 \in \mathcal{R}(\mathcal{M})$ such that $\rho_1(0) = s_1$ and $\rho_0 \sim_p \rho_1$.*

In other words, in a noninterferent model any two indistinguishable initial states have the same observation traces. A suitable interpretation of conservativity in this context is that NI is preserved under ignorance-preserving refinement:

Definition 8 (NI Preservation)**.** *Suppose $\mathcal{M}_a \Downarrow \mathcal{M}_c$ is ignorance preserving and $\mathcal{M}_a$ is noninterferent. Then $\Downarrow$* preserves noninterference *if $\mathcal{M}_c$ is noninterferent as well.*

Unfortunately, NI preservation does not hold in general. This can be easily seen for Example 4, where the addition of an unrelated secret variable breaks noninterference on the concrete level. Moreover, certain IPRs that reduce nondeterminism by increasing the state space do not preserve noninterference in the sense of Def. 8.

Example 7. Assume an abstract model with runs $s_1 \xrightarrow{\alpha} s_3$ and $s_2 \xrightarrow{\alpha} s_4$ for $\alpha \in \{a, b\}$. An ignorance-preserving refinement is given by runs $t_1^a \xrightarrow{a} t_3^a$, $t_1^b \xrightarrow{b} t_3^b$, $t_2^a \xrightarrow{a} t_4^a$, and $t_2^b \xrightarrow{b} t_4^b$, where $s_i \Downarrow t_i^\alpha$ for all i and α. Let all initial states be indistinguishable, then clearly the abstract model is noninterferent. However, for $t_1^a \approx t_2^b$ no indistinguishable runs exist, i.e., NI does not hold on the concrete level unless we refine the observation equivalence relation to distinguish t_i^a from t_j^b.

Note however, that the example is still secure, in the sense that a concrete observer cannot tell whether s_1 or s_3 were the initial state of a corresponding abstract computation. This is exactly the sense of confidentiality that IPR is preserving, as it is intentionally defined to prevent an observer from gaining knowledge about confidential information in terms of the abstract model.

More precisely, consider the following alternative definition of noninterference as a closure property on ignorance sets I.

$$P_{NI,p}(I) \equiv \forall \rho_0 \in I, s_1.\ \rho_0(0) \sim_p s_1 \Rightarrow \exists \rho_1 \in I.\ \rho_1(0) = s_1$$

Obviously, requiring $P_{NI,p}(I_p\{\rho_0\})$ for all $\rho_0 \in \mathcal{R}(\mathcal{M})$ is equivalent to Def. 7. In fact, it is well known that a plethora of information flow policies can be expressed as closure properties of the set of indistinguishable runs of a system [15]. As IPR guarantees $I_p\phi = (I_p\psi)\Uparrow$ if $\phi \Downarrow \psi$, for any closure property P that holds on $I_p\phi$, we immediately get $P((I_p\psi)\Uparrow)$, i.e., the information policy is preserved by the indistinguishable runs of the concrete model, when "re-abstracting" them w.r.t. $\Downarrow$. Thus a concrete observer does not gain any more information about the confidential abstract behaviors and values than allowed by the abstract specification.

We argue that this sense of confidentiality preservation is sufficient in most cases. Still, for certain refinements IPR can also preserve information flow policies in the traditional sense.

Proposition 5. *Consider a system that is partitioned into disjoint sets H of secret and L of observable variables. Moreover assume that the refinement relation induces a bijection $\leftrightarrow$ on secret variables such that $s \Downarrow t$ implies $s.d|_H \leftrightarrow t.d|_H$. For such systems, IPR preserves NI in the sense of Def. 8.*

Proof. For any run σ_0 let $\sigma_0(0) \approx_p t_1$. By the simulation property and well-formedness, we find $\rho_0 \Downarrow \sigma_0$ and $s_1 \Downarrow t_1$ such that $\rho_0(0) \sim_p s_1$. By abstract level noninterference we find a run $\rho_1 \in \mathcal{R}(\mathcal{M}_a)$ such that $\rho_1(0) = s_1$ and $\rho_1 \sim_p \rho_0$. Then by $(*)$, we find σ_1 such that $\rho_1 \Downarrow \sigma_1$ and $\sigma_1 \approx_p \sigma_0$, in particular $\sigma_1(0).d|_L = \sigma_0(0).d|_L = t_1.d|_L$. For the secret variables, we have $\sigma_1(0).d|_H \leftrightarrow^{-1} s_1.d|_H \leftrightarrow t_1.d|_H$. Thus $t_1 = \sigma_1(0)$ and σ_1 is the desired noninterferent run. □

A special case of the above type of system is when we have the same secret variables in abstract and concrete model (cf. Example 5). Clearly the proof above fails if the concrete model introduces unrelated secret variables H' in the initial states such that no bijection can be built, because then we cannot show that run σ_1 obtained from IPR starts in t_1. Noninterference then has to be proved separately for the

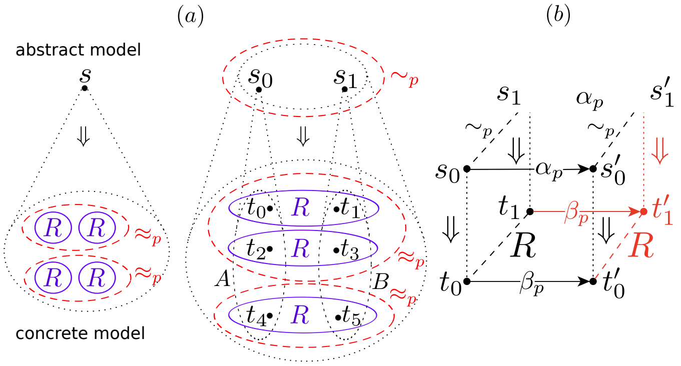


Fig. 8. (a) Partitioning of concrete states. For simplicity, we only consider the functional case of $\Downarrow^{-1}$, i.e., $A \cap B = \emptyset$. (b) Simulation cube. Black relations are assumed, red ones must be shown.

additional variables H', but the proof can be simplified by assuming that the original secrets H are observable.

Example 8. (Secure Compiler) Assume we have proved IPR for a secure compiler such that a binary implementation does not leak the values of secret variables H when the source program does not. If the compiler represents variables in a unique fashion, by Prop. 5 we obtain NI w.r.t. concrete relation $\approx_H$ which keeps H unobservable. Additionally the compiler randomizes the binary's memory layout to protect against ROP attacks. Naturally, we would like to keep seed H' of the random layout secret, i.e., establish NI w.r.t. a corresponding relation $\approx_{H'}$, however there is no notion of this implementation detail at the source code level. Hence, IPR holds for a relation $\approx_{H,H'}$, covering the intersection of observable variables for $\approx_H$ and $\approx_{H'}$, even if the compiler introduces a vulnerability that leaks H'. However, if we prove NI w.r.t. $\approx_{H'}$ separately (disregarding H) and rule out such leakage, then adding NI for $\approx_H$ from IPR yields NI for $\approx_{H,H'}$ via a simple transitive argument: Assume two states $t_1 \approx_{H,H'} t_2$ with secret values $t_1.d(H,H') = (h_1, h'_1)$ and $t_2.d(H,H') = (h_2, h'_2)$. We can construct a state t_3 with $t_3.d(H,H') = (h_2, h'_1)$ such that $t_1 \approx_H t_3 \approx_{H'} t_2$. Using the NI results for $\approx_H$ and $\approx_{H'}$ we obtain NI for $\approx_{H,H'}$.

VII. An Unwinding Condition for IPR

We present an unwinding condition for ignorance preservation. The idea is similar to standard unwinding or bisimulation equivalences used in literature to establish certain information flow policies (e.g., [16]), except that the focus here is instead on preserving the given indistinguishable behavior of the abstract level when refining to the concrete one.

Definition 9 (IPR Unwinding Relation). *A symmetric relation $R \subseteq T \times T$ is a p-*unwinding relation, *if $s_0 \Downarrow t_0$, $s_1 \Downarrow t_1$, $s_0 \sim_p s_1$ and $t_0 \; R \; t_1$ implies:*

1) $t_0 \approx_p t_1$.

2) If $t_0 \xrightarrow{\beta_p} t'_0$, $s_0 \xrightarrow{\alpha_p} s'_0$, $s_1 \xrightarrow{\alpha_p} s'_1$, and $s'_0 \Downarrow t'_0$ then exists t'_1 with $t_1 \xrightarrow{\beta_p} t'_1$, $s'_1 \Downarrow t'_1$, and $t'_0 \; R \; t'_1$ (thus also $t'_0 \approx_p t'_1$).

Figure 8.a illustrates the idea of relation R partitioning the concrete states. On the left side we see that for one abstract state the refinement may introduce distinguishable representatives. Relation R now partitions the $\approx_p$-equivalence classes into indistinguishable subsets that may R-relate to concrete representatives of other indistinguishable abstract states (right side). The unwinding condition requires that such pairs of concrete states stay in R after each step (and thus remain indistinguishable) if their abstract counterparts do. A corresponding simulation "cube" [6, 8] is shown in Fig. 8.b.

Definition 10 (Partition Preserving Refinement)**.** *The refinement relation $\mathcal{M}_1 \Downarrow \mathcal{M}_2$ is p-*partition preserving, *if there is a p-unwinding relation R such that, if s_0, s_1, t_0 are start states such that $s_0 \Downarrow t_0$ and $s_0 \sim_p s_1$ then there is t_1 such that $s_1 \Downarrow t_1$ and $t_0 \; R \; t_1$.*

In the simplest case, we have $R \equiv \approx_p$. It is easy to show that such a p-partition preserving refinement holds for Examples 5 and 6 with $p = env$. However, there are more complex cases:

Example 9 (Three Party Unwinding)**.** We show that R has to be more restrictive to be p-partition preserving for the refinement of Example 3. Given a process p_i, $i \in \{0,1,2\}$, we choose R such that for the states in relation it ensures observational equivalence w.r.t p_i ($\approx_{p_i}$) as well as equality for the values of $\mathtt{y}_{i+1}$, $\mathtt{y}_{i-1} + \mathtt{x}_{i+1}$, and $\mathtt{x}_{i-1}$ which p_i can distinguish as noted before. Let $s_0 \Downarrow t_0 \; R \; t_1$, and $s_0 \sim_{p_i} s_1 \Downarrow t_1$ be initial states. We get $s_j.\mathtt{z}_k = t_j.\mathtt{x}_k + t_j.\mathtt{y}_k$, for $j \in \{0,1\}$ and $k \in \{i-1, i, i+1\}$ as well as $s_0.\mathtt{z}_i = s_1.\mathtt{z}_i$ and the equivalences on t_0 and t_1 given by R.

Condition 9.1 is trivial to show. For 9.2 we know that p_i observes the terms $\mathtt{x}_i$, $\mathtt{y}_i$, and $[\mathtt{y}_{i+1}, \mathtt{y}_{i-1} + \mathtt{z}_{i+1}, \mathtt{y}_i + \mathtt{z}_{i-1} + \mathtt{z}_{i+1}]$ where the list contains the inputs p_i receives during the protocol. We assume $t_0 \rightarrow t'_0$, $s_0 \rightarrow s'_0$, $s_1 \rightarrow s'_1$, and $s'_0 \sim_{p_i} s'_1$ as well as $s'_0 \Downarrow t'_0$. Therefore, we obtain $t'_0.res = s'_0.res = s'_1.res$, thus $s_0.\mathtt{z}_{i-1} + s_0.\mathtt{z}_{i+1} = s_1.\mathtt{z}_{i-1} + s_1.\mathtt{z}_{i+1}$. It is easy to check that then any step $t_1 \rightarrow t'_1$ will produce the same observations as $t_0 \rightarrow t'_0$ for p_i, if the initial states also agree on $\mathtt{x}_i$, $\mathtt{y}_i$, $\mathtt{y}_{i+1}$, $\mathtt{y}_{i-1} + \mathtt{x}_{i+1}$, and $\mathtt{x}_{i-1}$ as assumed. Since the system is deterministic, simulation gives $s'_1 \Downarrow t'_1$. As the $\mathtt{x}$ and $\mathtt{y}$ do not change, and the same result is computed, i.e., $t'_1.res = s'_1.res = t'_0.res$, we have $t'_0 \; R \; t'_1$. □

We show that partition preservation is a sufficient condition for ignorance preservation as it implies paired refinement (∗).

Proposition 6. *If refinement $\Downarrow$ is p-partition preserving then $\Downarrow$ is p-ignorance preserving.*

The converse of Prop. 6, i.e., that ignorance preservation entails the existence of a p-partition preserving refinement, does not hold. This is due to the linear time nature of ignorance preservation vs. the branching time nature of unwinding. Figure 9 adapts the classical example: Clearly, the refinement is ignorance-preserving, but not p-partition preserving. If it were, we would need to find an IPR unwinding relation R such that $t_1 \; R \; t_5$, hence also $t_2 \; R \; t_6$. But then, since $t_2 \xrightarrow{c} t_4$, $\lceil t_2 \rceil \xrightarrow{b} \lceil t_4 \rceil$, and $\lceil t_6 \rceil \xrightarrow{b} s_6$, we need to find some t such that $t_6 \xrightarrow{c} t$, $\lceil t \rceil = s_6$, and $t_4 \; R \; t$. But such a t does not exist.

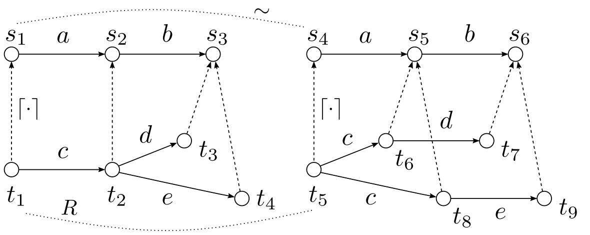


Fig. 9. IPR does not require p-partition preservation.

VIII. Compositionality

In this section we begin to study ways of combining refinements. At a minimum, two types of composition, vertical and sequential, must be considered. Vertical composition is essential for stepwise refinement, and sequential composition is a prerequisite for scalability.

For vertical composition, first, assume models $\mathcal{M}_i$, $0 \leq i \leq 2$, and assume we have p-ignorance-preserving refinements $\Downarrow_i$, $i \in \{1, 2\}$ from $\mathcal{M}_{i-1}$ to $\mathcal{M}_i$. Then the relational composition $\Downarrow_1 \circ \Downarrow_2$ from $\mathcal{M}_0$ to $\mathcal{M}_2$ should be p-ignorance-preserving, too. The simulation property and well-formedness properties are easily checked, it remains to show that a vertically composed refinement is an IPR if its component refinements are:

Proposition 7. *If the refinements $\Downarrow_1$ and $\Downarrow_2$ are p-ignorance-preserving then so is $\Downarrow = \Downarrow_1 \circ \Downarrow_2$.*

Proof. Let $\phi \Downarrow_1 \psi \Downarrow_2 \xi$. Using $(*)$, assume $\rho_0 \sim_p \rho_1 \Downarrow \tau_1$. We find σ_1 such that $\rho_1 \Downarrow_1 \sigma_1 \Downarrow_2 \tau_1$. By $(*)$ there is σ_0 with $\rho_0 \Downarrow_1 \sigma_0 \approx_p \sigma_1$. Applying $(*)$ again for $\sigma_0 \sim_p \sigma_1 \Downarrow_2 \tau_1$, we obtain τ_0 with $\rho_0 \Downarrow \tau_0 \approx_p \tau_1$ and conclude via Prop. 3. □

We present a version of sequential composition, adapted to the multi-threaded models introduced in Sect. II. Given models $\mathcal{M}_1$, $\mathcal{M}_2$ we define their sequential composition $\mathcal{M}_1 \rhd \mathcal{M}_2$ with parameters S, C, D, etc., using indices i for those of $\mathcal{M}_i$, and we assume that $PId_1 = PId_2$, $D_1 = D_2$, and (for simplicity) $\mathcal{O}_{p,1} = \mathcal{O}_{p,2}$. In other words, models $\mathcal{M}_1$ and $\mathcal{M}_2$ are required to have the same threads and identically shaped stores and observations. We let $PC = PC_1 + PC_2$, the disjoint sum of PC_1 and PC_2 with corresponding injection functions PC_1, PC_2, and . For $s = (c, d) \in S_1$ we define:

$$\begin{aligned} \mathit{before}(c, d) &= (\lambda p.\, \mathsf{PC}_1\ c(p), d) \\ \mathit{after}(c, d) &= (\lambda p.\, \mathsf{PC}_2\ c(p), d) \end{aligned}$$

injecting a state in $\mathcal{M}_1$, resp. $\mathcal{M}_2$, into the state space of $\mathcal{M}_1 \rhd \mathcal{M}_2$. Then $S_0 = \{\mathit{before}(s_0) \mid s_0 \in S_{1_0}\}$ and $s \to s'$ if:

- $s = \mathit{before}(s_1)$, $s' = \mathit{before}(s_1')$, and $s_1 \to_1 s_1'$, or
- $s = \mathit{after}(s_2)$, $s' = \mathit{after}(s_2')$, and $s_2 \to_2 s_2'$, or
- $s = \mathit{before}(s_1)$, $s' = \mathit{after}(s_2)$, s_1 is final in $\mathcal{M}_1$, s_2 is initial in $\mathcal{M}_2$ such that $s_2.c = \mathit{init}_2$ and $s_2.d = s_1.d$.

For single process models this construction acts as a sequential composition in the expected way. For concurrent models the effect is to impose a global barrier, or synchronization point in the sense that in $\mathcal{M}_1 \rhd \mathcal{M}_2$ no observation of a thread in $\mathcal{M}_2$ can precede an observation of a thread in $\mathcal{M}_1$.

We can define the composition of runs $\rho_1 \rhd \rho_2$ as the partial operation for which the final store of a complete run ρ_1 is identical to the first store of ρ_2 (and control suitably injected into $PC_1 + PC_2$). Then each run ρ of $\mathcal{M}_1 \rhd \mathcal{M}_2$ can be written as either ρ_1 or $\rho_1 \rhd \rho_2$ with $\rho_1 \in \mathcal{R}(\mathcal{M}_1)$ and $\rho_2 \in \mathcal{R}(\mathcal{M}_2)$.

Example 10. Let $\mathcal{M}_1$ be the model of Example 3, and let $\mathcal{M}_2$ be identical to $\mathcal{M}_1$ except that $p_i.\mathtt{x}$ is exchanged in round 1 instead of $p_i.\mathtt{y}$. The composition $\mathcal{M}_1 \rhd \mathcal{M}_2$ has the same processes executing 8 rounds synchronously in parallel. In each run of $\mathcal{M}_1 \rhd \mathcal{M}_2$, $p_i.\mathtt{res} = p_0.\mathtt{x} + p_0.\mathtt{y} + p_1.\mathtt{x} + p_1.\mathtt{y} + p_2.\mathtt{x} + p_2.\mathtt{y}$ for rounds 4 to 8.

We extend sequential composition to sets of runs in the obvious way: $\phi_1 \rhd \phi_2 = \{\rho_1 \rhd \rho_2 \mid \rho_1 \in \phi_1, \rho_2 \in \phi_2\}$. However, this definition leads to a sobering observation about the composition of ignorance sets:

Proposition 8. $I_p(\phi_1 \rhd \phi_2) \subsetneq I_p\phi_1 \rhd I_p\phi_2$

Proof. For containment we calculate:

$$\begin{aligned} I_p(\phi_1 \rhd \phi_2) &= \{\rho' \mid \exists \rho \in \phi_1 \rhd \phi_2.\ \rho \sim_p \rho'\} \\ &= \{\rho' \mid \exists \rho_1 \in \phi_1, \rho_2 \in \phi_2.\ \rho_1 \rhd \rho_2 \sim_p \rho'\} \\ &\subseteq \{\rho_1' \rhd \rho_2' \mid \exists \rho_1 {\in} \phi_1, \rho_2 {\in} \phi_2.\ \rho_1 \sim_p \rho_1', \rho_2 \sim_p \rho_2'\} \\ &= \{\rho_1' \rhd \rho_2' \mid \rho_1' \in I_p\phi_1, \rho_2' \in I_p\phi_2\} \\ &= I_p\phi_1 \rhd I_p\phi_2 \end{aligned}$$

For strict containment see the proof of Theorem 2 below. □

Prop. 8 shows that when observing components in isolation, composition of runs that are indistinguishable in isolation may produce runs that can be distinguished when observing the components in sequence. Specifically, the independent choice of ρ_1, ρ_2 in line 3 of the proof does not entail a dependent choice such that $\rho_1 \rhd \rho_2 \sim_p \rho'$ exists, as the proof of Theorem 2 demonstrates.

In order to define the sequential composition of refinements, let refinement relations $\mathcal{M}_{a,1} \Downarrow_1 \mathcal{M}_{c,1}$ and $\mathcal{M}_{a,2} \Downarrow_2 \mathcal{M}_{c,2}$ be given. The composed relation $s \Downarrow_1 \rhd \Downarrow_2 t$ holds if either $s = \mathit{before}(s_1)$, $t = \mathit{before}(t_1)$ and $s_1 \Downarrow_1 t_1$, or $s = \mathit{after}(s_2)$, $t = \mathit{after}(t_2)$ and $s_2 \Downarrow_2 t_2$.

There is, however, a complication to resolve for this definition to make sense. In general, there is no guarantee that the composability condition at the model boundary holds across abstraction levels, i.e., there may be states $s_1 \Downarrow_1 t_1$ and t_2 such that t_1 is final, t_2 is initial, and $\mathit{before}(t_1) \to_c \mathit{after}(t_2)$ in the composed model (i.e. such that $t_1.d = t_2.d$), but whenever $s_2 \Downarrow_2 t_2$ then $s_1.d \neq s_2.d$. This could happen for instance if some information about a data item at abstract level is carried by the final control state of $\mathcal{M}_{c,1}$ at concrete level. Thus the simulation property may fail for $\Downarrow_1 \rhd \Downarrow_2$. Accordingly, we say that $\Downarrow_1$, $\Downarrow_2$ are *compatible*, if for every final s_1 in $\mathcal{M}_{a,1}$, there exists a unique initial s_2 in $\mathcal{M}_{a,2}$ such that $s_1.d = s_2.d$ and whenever t_1 is final in $\mathcal{M}_{c,1}$ with $s_1 \Downarrow t_1$, and there is an initial t_2 in $\mathcal{M}_{c,2}$ with $t_1.d = t_2.d$, then it holds that $s_2 \Downarrow_2 t_2$. Since there is only one initial control state in $\mathcal{M}_{a,2}$, this requires that the initial stores of $\mathcal{M}_{a,2}$ are a superset of the final stores of $\mathcal{M}_{a,1}$ and that $\Downarrow_1$ in final states agrees on the mapping of stores with $\Downarrow_2$ in initial states.

Proposition 9. *If $\Downarrow_1$, $\Downarrow_2$ are compatible well-formed simulations then $\Downarrow_1 \triangleright \Downarrow_2$ is a well-formed simulation.*

Unfortunately, compatibility does not give compositionality:

Theorem 2. *Ignorance-preserving refinement is not compositional w.r.t. sequential composition.*

Proof. Let $\mathcal{M}$ be the abstract three-party secret sharing model of Example 1. As noted in Section IV we obtain a p_i-ignorance preserving refinement for any concrete $\mathcal{M}_1$ executing the program in Example 3. Symmetrically, we obtain a p_i-ignorance preserving refinement for $\mathcal{M}_2$ of Example 10. Note that the two refinements are indeed compatible, but the composed refinement $\mathcal{M} \triangleright \mathcal{M} \Downarrow \mathcal{M}_1 \triangleright \mathcal{M}_2$ is not p_i-ignorance preserving. To see this, pick runs σ_1 in $\mathcal{M}_1$ and σ_2 in $\mathcal{M}_2$ such that $\sigma_1 \triangleright \sigma_2$ is defined. In the abstract run $\rho_1 \triangleright \rho_2 = \lceil \sigma_1 \triangleright \sigma_2 \rceil$ the $p_i.\mathtt{z}$ and $p_i.\mathtt{res}$ are determined. By, say, decrementing $p_{i-1}.z$ and incrementing $p_{i+1}.\mathtt{z}$ we obtain an abstract run $\rho_1' \triangleright \rho_2' \sim_{p_i} \rho_1 \triangleright \rho_2$. But $\rho_1' \triangleright \rho_2' \neq \lceil \sigma' \rceil$ for any run $\sigma' \approx_p \sigma_1 \triangleright \sigma_2$. Otherwise we would find $\sigma_1' \approx_p \sigma_1$ and $\sigma_2' \approx_p \sigma_2$ such that $\sigma' = \sigma_1' \triangleright \sigma_2'$, but σ_1 determines $p_{i+1}.\mathtt{y}$ and $p_{i-1}.\mathtt{x}$ for σ_1' and σ_2 determines $p_{i+1}.\mathtt{x}$ and $p_{i-1}.\mathtt{y}$ for σ_2', whence the composed run $\sigma_1 \triangleright \sigma_2$ determines $p_{i-1}.\mathtt{z}$ and $p_{i+1}.\mathtt{z}$ for $\sigma_1' \triangleright \sigma_2'$. But then it cannot be the case that $\lceil \sigma_1' \triangleright \sigma_2' \rceil = \rho_1' \triangleright \rho_2'$. □

A correlate of Theorem 2 is the non-compositionality result of [17], proved in a stochastic setting for a different example.

The root cause of the phenomenon is that an observer is able to combine knowledge accumulated over composed protocol runs, to induce knowledge not attainable over separate runs only. While the IPR condition remains useful as a semantic condition at the system (end-to-end) level, Theorem 2 shows that it is not by itself a practical tool to prove ignorance preservation for sequentially composed systems.

While we have found sufficient and necessary conditions under which refinements $I_p\phi_i \Downarrow_i I_p\psi_i$ are sequentially composable, these consider pairs of refinements instead of constraining each refinement separately. Avoiding this modularity issue, we pursue a more tractable approach below.

IX. Relational Verification

The problem for sequential compositionality is that refinements have too little control over start and end states to ensure that runs can be composed. In this section we use ideas from relational Hoare logic [9] to address this.

Definition 11 (Relational Refinement). *Given symmetric relations $R_{pre}, R_{post} \subseteq T \times T$, we call the triple $\{R_{pre}\} \Downarrow \{R_{post}\}$ a* p-relational refinement, *if $\Downarrow$ is a well-formed refinement from $\mathcal{M}_a$ to $\mathcal{M}_c$ and the following conditions hold:*

1) *$R_{pre} \subseteq \approx_p$.*
2) *If $s_1 \Downarrow t_1$ are initial states, then given any $s_1 \sim_p s_2$, we can find a t_2 such that $s_2 \Downarrow t_2$ and $t_1 \; R_{pre} \; t_2$.*
3) *If $t_1 \; R_{pre} \; t_2$, $\sigma_1(0) = t_1$, $\rho_1 \Downarrow \sigma_1$, $\rho_1 \sim_p \rho_2$, $\rho_2(0) \Downarrow t_2$, then a run σ_2 exists with $\sigma_2(0) = t_2$, $\rho_2 \Downarrow \sigma_2$, $\sigma_1 \approx_p \sigma_2$, and if σ_1 is complete, so is σ_2 and $lst(\sigma_1) \; R_{post} \; lst(\sigma_2)$.*

Roughly, the triple $\{R_{pre}\} \Downarrow \{R_{post}\}$ expresses that whenever there is a terminating run from a concrete state t_1, which is a refinement of an abstract state indistinguishable from s_2, then it is possible to find a terminating run from some other concrete state t_2, which is a refinement of s_2, and such that the two runs are indistinguishable, R_{pre} holds on the initial states of the runs, and R_{post} on the final states of the two runs.

By conditioning R_{post} on whether σ_1 and σ_2 are complete, the definition covers both terminating and diverging programs. Clearly, the second and third condition imply ($*$), thus:

Corollary 2. *Any p-relational refinement is an IPR.* □

The main result of this section is to show that relational refinements are compositional. To that end let $\mathit{before}(t_1)$ $\mathit{before}(R)$ $\mathit{before}(t_2)$ if and only if $t_1 \; R \; t_2$ and similarly for after. Moreover we say that relation R_1 *can proceed as* relation R_2 if whenever $t_1 \; R_1 \; t_2$ for final states t_1, t_2 of $\mathcal{M}_{c,1}$ and $t_1.d = t_1'.d$, $t_2.d = t_2'.d$ for initial states t_1', t_2' of $\mathcal{M}_{c,2}$, we have $t_1' \; R_2 \; t_2'$. Intuitively, this ensures that R_1 constrains all memory contents on which R_2 depends and that the possible observations of p encoded in R_2 are a subset of those encoded in R_1. If $\{R_2\} \Downarrow_2 \{R_{post}\}$ is a p-relational refinement, then $\mathcal{M}_{c,2}$ does not reveal any information that $\mathcal{M}_{c,1}$ keeps secret. We obtain:

Theorem 3. *Suppose $\{R_{pre}\} \Downarrow_1 \{R_1\}$ and $\{R_2\} \Downarrow_2 \{R_{post}\}$ are p-relational refinements such that R_1 can proceed as R_2 and $\Downarrow_1$, $\Downarrow_2$ are compatible. Then $\{\mathit{before}(R_{pre})\} \Downarrow_1 \triangleright \Downarrow_2$ $\{\mathit{after}(R_{post})\}$ is a p-relational refinement.*

It follows by Cor. 2 that if $\{R_{pre}\} \Downarrow_1 \{R_1\}$ and $\{R_2\} \Downarrow_2$ $\{R_{post}\}$ are p-relational refinements, and R_1 can proceed as R_2, then the refinement $\Downarrow_1 \triangleright \Downarrow_2$ is p-ignorance preserving.

Note that the unwinding from Sec. VII is a special case of a p-relational refinement, i.e., it establishes a relational invariant on the complete runs of a concrete model.

Theorem 4. *If $\Downarrow$ is p-partition preserving for a relation R that relates final states only to final states, then $\{R\} \Downarrow \{R\}$.*

This allows to use unwinding as a powerful tool for sequential composition. If unwinding relations can be found for the different finite components of a system, where one can proceed as the next one subsequently, we obtain IPR for the composition via Theorem 3. Indeed we can show: Using Theorem 4 and the unwinding relation R of Example 9 it now follows that $\{R\} \Downarrow \triangleright \Downarrow \{R\}$ where $\Downarrow$ is the three party refinement of Example 3. Similarly for single rounds of Examples 5 and 6 we obtain $\{\sim_{env}\} \Downarrow \{\sim_{env}\}$. Below we also show how Theorems 3 and 4 allow to apply our approach inductively, splitting the (potentially diverging) runs of a complex system into complete runs of its components.

X. Example: Oblivious RAM

An Oblivious RAM (ORAM) is a program transformation which preserves functional behavior and can secure a program against adversaries capable of observing the accessed memory addresses; e.g., an attacker which can mount trace-driven attacks via shared data cache, or inspect the disk sectors accessed by a database, or observe the accesses performed to a network storage in a cloud infrastructure. The abstract model

```
system oramSpec
 observable round : nat init 0
 process p
   var mem[0..n], add[0..z], res : nat
   fun read(a:nat) : nat =
     if random(1) then ret mem[a] else abort
   repeat res := read(add[round]); round++ |>
```

Fig. 10. ORAM model.

```
 1  system oramImpl {
 2   observable rds, wts: nat list init []
 3   observable round : nat init 0
 4   process p
 5    var mem[0..2^m−2] : block
 6    var tbl[0..n] : nat
 7    var add[0..z], r_l[0..z], r_p[0..z], res: nat
 8    fun fetch(i:nat):nat = rds:=rds++[i];ret mem[i]
 9    fun store(i:nat, b:block) =
10      if |b| > k then abort
11      else wts := wts++[i]; mem[i] := b
12    fun read(a:nat) : nat =
13      for i in path-from-root(tbl[a]) do
14        tmp := fetch(i);
15        if a↦(l,v) in tmp then
16          (res',tmp) := (v, tmp \ [a↦(l,v)]);
17        store(i, tmp);
18      tbl[a] := r_l[round];
19      store(0, fetch(0) + [a↦(tbl[a],res')]);
20      ret res'
21    fun push-back() =
22      (leaf, push) := (r_p[round], []);
23      for i in path-from-root(leaf) do
24        keep := fetch(i);
25        (push, keep) :=
26          ([a↦(l,v) in push ∪ keep | i ↛[l,leaf]],
27           [a↦(l,v) in push ∪ keep | i →[l,leaf]]);
28        store(i, keep)
29    repeat res := read(add[round]);
30           push-back(); round++ |>
```

Fig. 11. Implementation of ORAM.

of ORAM (Fig. 10) is secure against these types of attacks by construction, due to indistinguishability of accessed memory addresses. Notice that the specification has $n{+}1$ memory cells, accepts random failures of the memory, and that the array of $z+1$ elements add represents the list of addresses accessed by the process.

The implementation of the ORAM, presented in Fig. 11, is a simplified version of Chung and Pass's construction [18]. It uses $2^m{-}1$ memory blocks that are logically organized in a balanced binary tree with depth m and 2^{m-1} leafs. Each memory block contains up to k entries of the shape $a \mapsto (l, v)$, where $a \in \{0 \dots n\}$ is an address in the address space of the ORAM, $l \in \{0 \dots 2^{m-1}{-}1\}$ is a leaf identifier, and $v \in \mathbb{N}$ is a value. The maximum block size depends on a security parameter and must be small enough to store temporary copies of a block and the additional temporary variables in hardware registers. The implementation maps, via the table tbl, each address of the specification to a single leaf, i.e., the value of the address is stored in one of the blocks in the path that connect the tree root to the address's leaf. Additionally to the list of addresses accessed by the process, the ORAM implementation uses two lists of random numbers r_l and r_p.

Reading from an ORAM address involves reading and writing the entire tree path that connect the tree's root to the address's leaf. For every node in the path (lines 13-17): the algorithm reads the corresponding block (line 14), checks if the block contains an entry for the queried address and eventually removes it (lines 15-16), and writes back the updated block (line 17). Finally, the algorithm associates the address to a new random leaf (line 18), and appends the entry for the address to the root block (line 19), which is indexed by 0. In order to reduce contention in the root block, the process performs a push-back procedure, chooses a random *leaf* and pushes every entry $a \mapsto (l, v)$ in the path of the leaf to the lowest common ancestor i of l and *leaf* (written $i \rightarrow [l, \mathit{leaf}]$). All variables, except memory locations, are stored in hardware registers, hence the corresponding operations are not visible to the attacker. Accesses to the memory (i.e., fetch and store) extend the adversary's observations with the index of the accessed block. These observations are collected in the history variables rds and wts. The mechanism to write into the ORAM is analogous, with the exception of the usage of a new value in place of res in line 19.

The following Lemma shows that the ORAM construction is correct.

Lemma 1. *$\Downarrow^r_{ORAM}$ is a refinement, where $s \Downarrow^r_{ORAM} t$ iff:*

1) *$(s.add, s.res, s.round) = (t.add, t.res, t.round)$;*
2) *if $a \mapsto (l, v) \in t.mem[i]$ then $v = s.mem[a]$, $l = t.tbl[a]$, and $i \in$ path-from-root(l);*
3) *for every address a exists a unique block i such that $a \mapsto (l, v) \in t.mem[i]$*
4) *$s.round = r$*

Proof. Let $t \rightarrow t'$ and $a = t.add[t.oramRound]$. Property (3) ensures that there is a unique block i_a that contains an entry for a and (2) guarantees that $i_a \in$ *path-from-root*$(t.tbl[a])$. For these reasons lines (13-17) only modify block i_a, removing from this block the entry for a. Property (2) also guarantees that $res' = s.mem[a]$. Therefore lines (13-17) preserves properties (2) and (3) for all addresses except a. These properties are re-established for a by lines (18) and (19), which adds the entry for a to the root. In fact, the new position of the entry (i.e., the root) is in the path of all possible leafs. The result of $read$ is assigned to res, hence $t'.res = s.mem[a]$.

For the push-back operation, note that it does not modify values of the addresses. Properties (2) and (3) are preserved by lines (23-28) if we consider $push$ an additional block of the ORAM. Moreover, during the push procedure $push$ only contains entries for addresses mapped to descendant of i and after the last iteration $push$ is empty. Therefore properties (2) and (3) are re-established at the end of the push-back procedure.

For property (1) it is sufficient to note that add is never changed, $t'.res = s.mem[a]$, and $t'.round = t.round + 1$. Hence this property can be re-established by the transition $s \rightarrow s'$ that does not fail. □

To show that the ORAM is an ignorance-preserving refinement we use relational verification, which allows us to compose sequences of ORAM and other operations. To define the relation, we introduce the following auxiliary function:

$$\begin{aligned} \mathit{leaf}(t,a,i) = & \begin{cases} t.\mathit{tbl}[a] & \text{if } A = \emptyset \\ t.r_l[\mathit{max}\ A] & \mathit{otherwise} \end{cases} \\ \text{where } A = & \{ j \mid t.\mathit{round} \leq j < i \land t.\mathit{add}[j] = a \} \end{aligned}$$

Intuitively, $\mathit{leaf}(t,a,i)$ identifies to which leaf the address a is mapped after $i - t.\mathit{round}$ steps from t. In fact:

Lemma 2. *If* $t_0 \rightarrow t_1 \rightarrow^{j-1} t_j$ *then* $\mathit{leaf}(t_0, a, j + t_0.\mathit{round}) = \mathit{leaf}(t_1, a, j + t_1.\mathit{round} - 1) = t_n.\mathit{tbl}[a]$.

Proof. Notice that

$$j + t_0.\mathit{round} = j + t_1.\mathit{round} - 1 = t_j.\mathit{round}\,.$$

We first prove $\mathit{leaf}(t_0, a, j + t_0.\mathit{round}) = \mathit{leaf}(t_1, a, j + t_1.\mathit{round} - 1)$ by cases: (1) $a = t_0.\mathit{add}[t_0.\mathit{round}]$ and (2) $a \neq t_0.\mathit{add}[t_0.\mathit{round}]$.
Case (1) It is enough to show that A_0 for t_0 is equal to A_1 for t_1 and that $t_0.\mathit{tbl}[a] = t_1.\mathit{tbl}[a]$.
Case (2) We notice that $\mathit{max}(A_1)$ for t_1 is equal to $\mathit{max}(A_0)$ of t_0 for any $i \neq t_0.\mathit{round} + 1$ (i.e., $j \neq 1$) and $A_0 \neq \emptyset$. For $i = t_0.\mathit{round}{+}1$ holds $A_0 = A_1 \cup \{t_0.\mathit{round}\}$ and $t_0.\mathit{round} = \mathit{max}(A_0)$. Hence, $\mathit{leaf}(t_0, a, t_0.\mathit{round} + 1) = t.r_l[t_0.\mathit{round}]$ and $\mathit{leaf}(t_1, a, t_1.\mathit{round}) = t_1.\mathit{tbl}[a]$. Line (18) ensures that $t_1.\mathit{tbl}[a] = t.r_l[t_0.\mathit{round}]$, hence *leaf* is unchanged.

The proof that $\mathit{leaf}(t_0, a, j + t_0.\mathit{round}) = t_j.\mathit{tbl}[a]$ is by induction over j, where we use the invariant property for the inductive argument and the fact that $\mathit{leaf}(t, a, t.\mathit{round}) = t.\mathit{tbl}[a]$ for the base case. □

We use the same pre/post-relations for relational verification: $t_1\ R\ t_2$ iff (1) $t_1 \approx t_2$ (i.e., rds, wts, and $round$ are the same), (2) $t_1.r_p = t_2.r_p$, (3) for every i holds $\mathit{leaf}(t_1, t_1.\mathit{add}[i], i) = \mathit{leaf}(t_2, t_2.\mathit{add}[i], i)$, and (4) let dom be the set of addresses of the entries of a block, $A_h = \{a \in t_h.\mathit{add}\}$ the addresses accessed by t_h for $h \in \{1, 2\}$, and L be the leaf nodes then (4a) for every node $b \notin L$ holds $\mathit{dom}(t_h.\mathit{mem}[b]) \subseteq A_h$ and (4b) for every $b \in L$ holds $|\{a \in \mathit{dom}(t_h.\mathit{mem}[b]) \setminus A_h\}| + |A_h| \leq k$.

Lemma 3. $\{R\} \Downarrow^r_{ORAM} \{R\}$ *is a relational refinement.*

Proof. Theorem 4 guarantees that $\{R\} \Downarrow^r_{ORAM} \{R\}$ is a relational refinement if $\Downarrow^r_{ORAM}$ is p-partition preserving. The latter is proven by showing that R is a proper unwinding relation according to Def. 9.

The proof relies on Lemma 1 and 2 The invariant of the *leaf* function and condition (3) ensure that the path of nodes accessed while reading is the same. Condition (2) guarantees that the path of nodes accessed while pushing back is the same. Property (4) is a sufficient condition to guarantee that the ORAM implementation does not fail: it entails that $n \leq 2^{m-1} * (k - |A_j|)$. If one of the two states, say t_2, does not satisfy (4) then it is possible to find an execution of t_1 that succeeds and that forces t_2 to fail by choosing unfortunate sequences for $t_1.r_l$ and $t_1.r_p$. For example, let t_2 access $|A_2| > k$ different addresses and let t_1 always access the same address. The sequences r_l and r_p may be the repetition of a single leaf l. While t_1 may succeed, since it always moves the same entry to the root and pushes it back to leaf l, t_2 will fail independently of the initial allocation of the ORAM entries. In fact, t_2 is forced within $k+1$ steps to push $k+1$ entries into the same leaf l. □

In practice, Chung and Pass show that there exists a negligible function $\mu(n)$ such that the probability of overflow is bounded by $\mu(n)$. This allows to relax property (4) for practical applications.

Theorem 5. $\Downarrow^0_{ORAM} \triangleright \cdots \triangleright \Downarrow^n_{ORAM}$ *is an IPR.*

Proof. The proof shows $\{R\} \Downarrow^0_{ORAM} \triangleright \cdots \triangleright \Downarrow^j_{ORAM} \{R\}$ by using induction on j. For ORAM it is straightforward to show that R can proceed as R, $\Downarrow^r_{ORAM}$ and $\Downarrow^{r-1}_{ORAM}$ are compatible, and $\mathit{before}(R) = R = \mathit{after}(R)$. Therefore for the inductive case we can use Lemma 3 to show that $\{R\} \Downarrow^r_{ORAM} \{R\}$ and Theorem 3 to obtain $\{R\} \Downarrow^0_{ORAM} \triangleright \cdots \triangleright \Downarrow^r_{ORAM} \{R\}$ from $\{R\} \Downarrow^0_{ORAM} \triangleright \cdots \triangleright \Downarrow^{r-1}_{ORAM} \{R\}$. Finally Corollary 2 guarantees that $\Downarrow^0_{ORAM} \triangleright \cdots \triangleright \Downarrow^j_{ORAM}$ is an IPR. □

XI. Related Work

Starting with Goguen and Meseguer [24] there is a large body of work on specifying information flow security policies using different variations of noninterference. The need to support communication beyond the multilevel security model has driven generalizations of noninterference based on some form of declassification [25, 26, 3, 27, 28, 29, 26], with the goal of defining specific circumstances under which new flows of information are allowed. Our objective is fundamentally different: Rather than devising a specific mechanism for declassification we are interested in ways of transferring arbitrary information flow properties from specification (the abstract model) to implementation (the refined model). Therefore we provide a verification strategy that is oblivious to the mechanism used to analyze declassification in the abstract model.

Knowledge-based formulations of information flows have recently gained popularity [27, 3, 30], often modeling observer knowledge by equivalence relations on input states. A precursor to our work is the work of Cohen et al. [31] on abstraction in multi-agent systems. They introduce an epistemic simulation relation that is essentially a state-based version of IPR, and use this to show preservation of formulas in the epistemic temporal logic ACTLK.

Table I compares previous work addressing the problem of confidentiality-preserving refinement in some form. It summarizes the key features and evaluates the results with respect to the requirements laid out in our introduction: support for nondeterminism (REQ2); preservation of noninterference; support for intentional information leakage (REQ3); type of refinement and composition (REQ4); support for probabilistic models; and identification of unwinding verification condition (REQ6). Additionally, in this table we use Santen's [32] classification of refinements for a better comparison:

- *process refinement* (REQ2): reducing nondeterminism,
- *data / observational refinement* (REQ1): transformation of data and introduction of observations, and

Papers	Model	Runs	Observations	Preserves	Refinement	Nondeterminism	Noninterference	Intent. leakage	Process Ref	Data / Obs Ref	Action Ref	Unwinding	S Composition	V Composition	Probabilistic
Graham-Cumming and Sanders [10]	State machine	Events	uninterpreted	Indisting.	Simulation	●	●	○	●	◐	●	◐	−	●	○
Roscoe et al. [19]	CSP	Events	Low events	Obs Det	CSP refinement	●	●	○	●	○	○	○	−	●	○
Mantel [16]	Event system	Events	Low events	Unwinding	Trace inclusion	●	●	○	●	○	○	●	−	●	○
Heisel et al. [20]	CSP	Events	Window channel	Prob Indisting.	Trace inclusion	●	●	●	●	●	◐	○	○	●	●
Santen et al. [17]	CSP	Events	Window channel	Prob Indisting.	CSP refinement	●	●	●	●	●	◐	○	●	●	●
Alur et al. [13]	LTS	States, labels	uninterpreted	Secrecy	Trace inclusion	●	●	○	●	◐	◐	●	−	●	○
Seehusen and Stølen [21]	Processes	Events	Low events	Policy	Trace projection	●	●	○	●	●	●	○	−	●	○
Morgan [4]	Programs	States	Low Vars, CF	Ignorance	Global Var Eq	●	●	●	●	◐	●	○	◐	●	○
Van der Meyden and Zhang [11]	LTS	States, labels	Obs function, labels	Policy	Simulation	●	●	○	●	○	◐	●	−	●	○
Costanzo et al. [12]	State machine	States	Obs function	Indisting.	Simulation	○	●	○	−	◐	●	●	−	●	−
Murray, Sison, et al. [6, 22]	Programs	States	Low Vars, Timing	Noninterference	Simulation	○	●	○	−	○	●	●	−	●	−
Barthe et al. [8, 23]	LTS	States, labels	CF, Mem Trace	Constant-Time	Simulation	○	●	○	−	◐	●	●	−	●	−
This work	State machine	States	Obs function	Ignorance	Simulation	●	●	●	●	●	●	●	●	●	○

CF=Control Flow, Conf=Confidentiality, Det=Determinism, Eq=Equivalence, Indisting.=Indistinguishability, Intent.=Intentional, Mem=Memory, Obs=Observation, Prob=Probabilistic, Ref=Refinement, S=Sequential, V=Vertical, Var=Variable, ●=yes, ◐=partially, ○=no, −=not applicable

TABLE I
COMPARISON OF RELATED WORK ON REFINEMENT WITH INFORMATION FLOW PRESERVATION.

- *action refinement* (REQ1): transformations of events or observations that are atomic at one level into composite ones at the other (cf. Examples 2, 6 and Section X).

Many works, e.g., [19, 33], recognize that noninterference can be preserved by refinement in the presence of observation determinism. However, limiting the scope to observation determinism makes it impossible to support intentional information leakage without explicit treatment of declassification. Other results, e.g., [21, 11], focus on scenarios where intentional communication of secret information is not allowed. Several approaches do not support full data and observation refinement. For instance, Mantel [16] presents a set of refinement operators which preserve specific information flow properties, but refinements can only reduce nondeterminism. In Graham-Cumming and Sanders [10] and Costanzo et al. [12] states that are indistinguishable on the abstract level must remain indistinguishable when refined, while we only assume the converse, i.e., well-formedness. Others allow data refinement but not refinement of observations, e.g., [13]. These limitations prevent the analysis of scenarios where the adversary gains observations at the implementation level (e.g., when the refinement must deal with potential cache based side-channels).

Three strands of work are most relevant to our development. First, Heisel et al. [20] consider information flow preservation for a probabilistic version of CSP. While the treatment of probabilistic information flow is out of scope for this work, their security condition in fact reduces to ($*$) when considering only the possibilistic case. Since they allow secret-dependent observation refinement, they encounter compositionality-related issues [17] that appear to be related to the ones identified in this paper. However, they do not provide an unwinding condition, nor a general method to regain compositionality such as our relational refinement, and generally leave action refinement out of scope.

Morgan and McIver [4, 34] propose an instrumented *shadow* semantics for ignorance-preserving program refinement, constructing the ignorance set explicitly for the final values of hidden variables. Thus they allow to conceal intermediate leakage of secrets which our approach detects. Moreover, their refinement requires equality for all global variables, allowing the introduction of new observations only as local variables within the scope of the refined program. These local variables cease to exist after the scope of the program has ended, hence the approach does not allow for persistent implementation variables that can carry data between invocations of different program segments. On one hand this makes the information-flow-preserving refinement compositional by construction (a.k.a. monotonicity [35]). However, and more seriously, it also rules out many realistic and important scenarios, e.g., caches clearly are persistent implementation variables of machine code programs.

Unwinding conditions similar to our Def. 9 have appeared in the literature before. For instance, Murray et al. [6] introduced a coupling invariant to preserve timing-sensitive value-dependent noninterference in a refinement "cube“ that expands on the common "square“ formed by abstract and concrete transition relations and the simulation relation. The methodology was later applied to verify information flow preservation for a compiler for simple concurrent programs [22]. In [7, p. 186] an unwinding cube similar to [6] is used to discharge low equivalence at cache-aware level against low equivalence at cacheless level for a given ARMv7 kernel routine; a precursory study to the general theory presented here. Barthe et al. [8] study information flow preservation in the context of compilation and demonstrate how information leaks due to control flow and memory accesses do not increase under several compiler optimizations. A follow-up paper [23] applies the technique to show that a somewhat modified version of the CompCert compiler is constant-time preserving. Our unwinding condition related to their constant-time simulations, but more permissive. There are important differences, however, in that 1) their model is deterministic, 2) the definition of leakage does not change in the refinement, 3) their condition lacks an epistemic justification and is "hard-coded" to enforce constant-time.

One of the features that differentiate the work presented here from other works referred to above is that IPR supports introducing secret dependent observations together with sequential compositionality. The one-time pad Example 6 illustrates this. There we obtain an IPR even though the attacker gains observability of `r xor hd(s)`. Similarly, a compiler may shuffle an array using a random key `r` and introduce a memory look-up dependent on `r xor hd(s)` without compromising the secrecy of `s`. This type of refinement is not considered in [8, 10, 12] because it violates the assumption requiring that all refined runs of the same abstract state have the same leakage. Similarly, this is not allowed in [6], since modified variables must either be private, which is not the case for `put`, or have their classification decreased, which would prevent the direct flow of `r xor hd(s)` to `put`. It *is* allowed in e.g. [4], but there the scoping of local variables prevents additional information learned during the execution of one implementation from being passed to the continuation. This makes IPR a more permissive condition justifying information-theoretically secure refinement. Therefore it allows us to raise the question when it is possible to infer that an entire implementation is information flow secure in terms of the information flow security of the specification and the information flow preservation of each step.

Finally, several other works focus on secure refinement for a certain class of programs, e.g., stream processing functions [36], schedulers [11], cryptographic algorithms [37, 38], or security protocols [39, 40]. These approaches typically employ models tailored to their use case and the specific desired information flow properties. In particular, reasoning about cryptography is with few exceptions [41] incompatible with a formal epistemic analysis. Generally, our methodology allows to show possibilistic information flow preservation for state-based models, independent of the application scenario.

XII. Concluding Remarks

We have proposed a new compositional approach to information flow preserving refinement, based on observer ignorance, explored its properties and limitations, and exposed it to several examples, some of them non-trivial. Our approach supports the refinement of nondeterministic synchronous multi-agent models using well-formed refinement relations. The synchronicity requirement is no major hurdle, as we showed how to embed asynchronous models. Refinements that are not well-formed are benign: they also allow the observer knowledge of abstract secret behavior to *decrease*. We ruled out the phenomenon to simplify our proofs, but it seems not to be a crucial necessity.

A trickier issue is the requirement that observations occur with the same granularity at all levels of abstraction, as changing the atomicity of observable behaviors is challenging [35], especially for nondeterministic systems. In Example 3 and Sect. X, we avoided the problem by fixing the number of concrete steps for each abstract step and by accumulating intermediate observations, respectively. The same trick could also be applied on the abstract level if a refinement reduces the number of steps, but a more flexible approach is desirable.

Another topic that needs more work is the application of our approach in automated relational verification (cf. [42]). While the theory seems amenable to it, questions remain how to efficiently build and make use of the abstract information flow specification when verifying its implementation.

The paper covers sequential composition, but not explicit concurrency. We have in fact defined a well-behaved synchronous parallel composition that respects IPR and closely mirrors the "internal" multi-agent structure of our models. This is left out for lack of space.

A possible extension of this work concerns encryption, since we do not restrict attackers to be computationally bound. Reconciling epistemics and cryptography in a computationally sound manner has been a difficult challenge since the inception of BAN logic [43], see [41, 44] for some possible approaches.

Nevertheless, the theory presented here is relevant in many other practical applications including secure compilation, program synthesis, and the treatment of hardware side channels.

Acknowledgment

This work partially was supported by grants from the Swedish Foundation for Strategic Research and the Swedish Civil Contingencies Agency, as well as the German Federal Ministry of Education and Research (BMBF) through funding for the CISPA-Stanford Center for Cybersecurity (FKZ: 13N1S0762).

References

[1] J. McLean, "A general theory of composition for a class of "possibilistic" properties," *Transaction on Software Engineering*, vol. 22, no. 1, pp. 53–67, 1996.

[2] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," *ArXiv e-prints*, Jan. 2018.

[3] A. Askarov and S. Chong, "Learning is change in knowledge: Knowledge-based security for dynamic policies," in *Computer Security Foundations Symposium (CSF)*. IEEE, 2012, pp. 308–322.

[4] C. Morgan, "*The Shadow Knows*: Refinement and security in sequential programs," *Science of Computer Programming*, vol. 74, no. 8, pp. 629–653, 2009.

[5] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi, *Reasoning about knowledge*. MIT Press, 1995.

[6] T. C. Murray, R. Sison, E. Pierzchalski, and C. Rizkallah, "Compositional verification and refinement of concurrent

value-dependent noninterference," in *IEEE 29th Computer Security Foundations Symposium, (CSF)*, 2016, pp. 417–431.

[7] H. Nemati, "Secure system virtualization: End-to-end verification of memory isolation," Ph.D. dissertation, Royal Institute of Technology, Stockholm, Sweden, 2017. [Online]. Available: https://nbn-resolving.org/urn:nbn:se:kth:diva-213030

[8] G. Barthe, B. Grégoire, and V. Laporte, "Secure compilation of side-channel countermeasures: The case of cryptographic constant-time," in *IEEE 31st Computer Security Foundations Symposium (CSF)*, July 2018, pp. 328–343.

[9] N. Benton, "Simple relational correctness proofs for static analyses and program transformations," *SIGPLAN Not.*, vol. 39, no. 1, p. 1425, Jan. 2004.

[10] J. Graham-Cumming and J. W. Sanders, "On the refinement of non-interference," in *Proceedings Computer Security Foundations Workshop IV (CSFW)*, 1991, pp. 35–42.

[11] R. Van der Meyden and C. Zhang, "Information flow in systems with schedulers, Part II: Refinement," *Theor. Comput. Sci.*, vol. 484, p. 7092, May 2013.

[12] D. Costanzo, Z. Shao, and R. Gu, "End-to-end verification of information-flow security for C and assembly programs," in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2016, pp. 648–664.

[13] R. Alur, P. Cerný, and S. Zdancewic, "Preserving secrecy under refinement," in *International Colloquium on Automata, Languages and Programming*. Springer, 2006, pp. 107–118.

[14] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *J. Cryptol.*, vol. 1, no. 1, p. 6575, Mar. 1988.

[15] H. Mantel, "Possibilistic definitions of security - An assembly kit," in *Proc. 13th IEEE Computer Security Foundations Workshop, CSFW*, 2000, pp. 185–199.

[16] ——, "Preserving information flow properties under refinement," in *Symposium on Security and Privacy*. IEEE, 2001, pp. 78–91.

[17] T. Santen, M. Heisel, and A. Pfitzmann, "Confidentiality-preserving refinement is compositional - Sometimes," in *Proceedings of the 7th European Symposium on Research in Computer Security (ESORICS)*. Springer-Verlag, 2002, p. 194211.

[18] K. Chung and R. Pass, "A simple ORAM," *IACR Cryptology ePrint Archive*, vol. 2013, p. 243, 2013.

[19] A. W. Roscoe, J. C. P. Woodcock, and L. Wulf, "Non-interference through determinism," in *Proceedings of the Third European Symposium on Research in Computer Security*, ser. ESORICS 94. Berlin, Heidelberg: Springer-Verlag, 1994, p. 3353.

[20] M. Heisel, A. Pfitzmann, and T. Santen, "Confidentiality-preserving refinement," in *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, ser. CSFW 01. USA: IEEE Computer Society, 2001, p. 295.

[21] F. Seehusen and K. Stølen, "Information flow security, abstraction and composition," *IET Information Security*, vol. 3, no. 1, pp. 9–33, March 2009.

[22] R. Sison and T. Murray, "Verifying that a compiler preserves concurrent value-dependent information-flow security," in *10th International Conference on Interactive Theorem Proving, ITP*, ser. LIPIcs, vol. 141, 2019, pp. 27:1–27:19.

[23] G. Barthe, S. Blazy, B. Grégoire, R. Hutin, V. Laporte, D. Pichardie, and A. Trieu, "Formal verification of a constant-time preserving C compiler," *Proc. ACM Program. Lang.*, vol. 4, no. POPL, Dec. 2019.

[24] J. A. Goguen and J. Meseguer, "Security policies and security models," in *Symposium on Security and Privacy*. IEEE, 1982, pp. 11–20.

[25] J. Rushby, "Noninterference, transitivity and channel-control security policies," SRI International, Tech. Rep., 1992.

[26] R. van der Meyden, "What, indeed, is intransitive noninterference?" in *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2007, pp. 235–250.

[27] M. Balliu, M. Dam, and G. Le Guernic, "Epistemic temporal logic for information flow security," in *Workshop on Programming Languages and Analysis for Security*. ACM, 2011, pp. 6:1–6:12.

[28] A. Askarov and A. Sabelfeld, "Gradual release: Unifying declassification, encryption and key release policies," in *Symposium on Security and Privacy*. IEEE, 2007, pp. 207–221.

[29] S. Chong and A. C. Myers, "Security policies for downgrading," in *Conference on Computer and Communications Security*. ACM, 2004, pp. 198–209.

[30] F. Besson, N. Bielova, and T. Jensen, "Hybrid monitoring of attacker knowledge," in *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, 2016, pp. 225–238.

[31] M. Cohen, M. Dam, A. Lomuscio, and F. Russo, "Abstraction in model checking multi-agent systems," in *Proc. 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Vol, 2*, 2009, pp. 945–952.

[32] T. Santen, "Preservation of probabilistic information flow under refinement," *Inf. Comput.*, vol. 206, no. 24, p. 213249, Feb. 2008.

[33] M. R. Clarkson and F. B. Schneider, "Hyperproperties," *Journal of Computer Security*, vol. 18, pp. 1157–1210, 2010.

[34] A. McIver and C. C. Morgan, "*Sums and Lovers:* Case studies in security, compositionality and refinement," in *Formal Methods*. Springer, 2009, pp. 289–304.

[35] C. Morgan, "Compositional noninterference from first principles," *Formal Aspects of Computing*, vol. 24, no. 1, pp. 3–26, 2012.

[36] J. Jürjens, "Secrecy-preserving refinement," in *Symposium of Formal Methods Europe*. Springer, 2001, pp. 135–152.

[37] C. Fournet and T. Rezk, "Cryptographically sound implementations for typed information-flow security," in

SIGPLAN Notices, vol. 43. ACM, 2008, pp. 323–335.

[38] J. B. Almeida, M. Barbosa, G. Barthe, and F. Dupressoir, "Certified computer-aided cryptography: efficient provably secure machine code from high-level implementations," in *Computer & Communications Security*. ACM, 2013, pp. 1217–1230.

[39] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," in *Symposium on Logic in Computer Science*. IEEE, 2005, pp. 331–340.

[40] C. Sprenger and D. A. Basin, "Refining security protocols," *Journal of Computer Security*, vol. 26, no. 1, pp. 71–120, 2018.

[41] M. Cohen and M. Dam, "A complete axiomatization of knowledge and cryptography," in *Symposium on Logic in Computer Science*. IEEE, July 2007, pp. 77–88.

[42] M. Balliu, M. Dam, and G. L. Guernic, "Encover: Symbolic exploration for information flow security," in *25th IEEE Computer Security Foundations Symposium, CSF*, 2012, pp. 30–44.

[43] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *Transactions on Computer Systems*, vol. 8, no. 1, pp. 18–36, 1990.

[44] J. Y. Halpern, R. van der Meyden, and R. Pucella, "An epistemic foundation for authentication logics (extended abstract)," in *TARK*, ser. EPTCS, vol. 251, 2017, pp. 306–323.

Appendix

A. Proofs

Proposition 6. If refinement $\Downarrow$ is p-partition preserving then $\Downarrow$ is p-ignorance preserving.

Proof. Let an unwinding relation R be given satisfying the conditions of Defs. 9 and 10. Following $(*)$ we assume runs ρ_0, ρ_1, σ_1 such that $\rho_0 \sim_p \rho_1 \Downarrow \sigma_1$. We construct, by induction on $n \geq 1$, a run σ_0^n of length n that in the limit meets the requirements of $(*)$, i.e., such that for length n prefixes ρ_0^n, ρ_1^n, σ_1^n of ρ_0, ρ_1, σ_1, respectively, $\rho_0^n \Downarrow \sigma_0^n$, $\rho_1^n \Downarrow \sigma_1^n$, and $\sigma_0^n \approx_p \sigma_1^n$, but additionally $\sigma_0^n(n)\ R\ \sigma_1^n(n)$.

For the base case ($n = 1$) let ρ_0^n, ρ_1^n, σ_1^n be singleton runs (i.e., start states) s_0, s_1, t_1 with $s_0 \sim_p s_1 \Downarrow t_1$. By Def. 10 we find $\sigma_0^n = t_0$ such that $s_0 \Downarrow t_0$ and $t_0\ R\ t_1$. Then by Def. 9.1, $t_0 \approx_p t_1$.

For the induction step assume we have determined runs ρ_0^{n+1}, ρ_1^{n+1}, σ_0^{n+1} such that $\rho_0^{n+1} \sim_p \rho_1^{n+1} \Downarrow \sigma_1^{n+1}$. By the induction hypothesis we find σ_0^n such that $\rho_0^n \Downarrow \sigma_0^n \approx_p \sigma_1^n$, and also $\sigma_0^n\ R\ \sigma_1^n(n)$. In that case, since $\sigma_1^n(n) = \sigma_1^{n+1}(n) \xrightarrow{\beta_p} \sigma_1^{n+1}(n{+}1)$, and similarly $\rho_1^{n+1}(n) \xrightarrow{\alpha_p} \rho_1^{n+1}(n{+}1)$, $\rho_0^{n+1}(n) \xrightarrow{\alpha_p} \rho_0^{n+1}(n{+}1)$, and $\rho_1^{n+1}(n{+}1) \Downarrow \sigma_1^{n+1}(n{+}1)$, we are, by partition preservation (Def. 9.2), able to extend σ_0^n to σ_0^{n+1} such that $\rho_0^{n+1} \Downarrow \sigma_0^{n+1}$ and $\sigma_0^{n+1}(n{+}1)\ R\ \sigma_1^{n+1}(n{+}1)$. But then by Def. 9.1, $\sigma_0^{n+1} \approx_p \sigma_1^{n+1}$, as desired. □

Proposition 9. If $\Downarrow_1$, $\Downarrow_2$ are compatible well-formed simulations then $\Downarrow_1 \triangleright \Downarrow_2$ is a well-formed simulation.

Proof. Given a run $\sigma = \sigma_1 \triangleright \sigma_2$, with $\sigma_i \in \mathcal{R}_{c,i}$ for $i \in \{1, 2\}$. By the refinement we get $\rho_1 \Downarrow_1 \sigma_1$. As σ_1 is complete, so is ρ_1 and $lst(\rho_1)$ is final. By compatibility we know therefore that $lst(\rho_1)$ determines an abstract initial state s_2 with $s_2 \Downarrow_2 \sigma_2(0)$. By inductive application of the simulation we obtain an abstract run $\rho_2 \Downarrow_2 \sigma_2$ with $\rho_2(0) = s_2$. Hence it exists $\rho = \rho_1 \triangleright \rho_2$ such that $\rho \Downarrow \sigma$. Well-formedness follows directly from the well-formedness of $\Downarrow_1$ and $\Downarrow_2$, i.e., if $\sigma_1 \triangleright \sigma_2 \approx_p \sigma_1' \triangleright \sigma_2'$ and $\rho_1 \triangleright \rho_2 \Downarrow \sigma_1 \triangleright \sigma_2$ as well as $\rho_1' \triangleright \rho_2' \Downarrow \sigma_1' \triangleright \sigma_2'$, then clearly $\sigma_1 \approx_p \sigma_1'$ and $\sigma_2 \approx_p \sigma_2'$. By well-formedness, $\rho_1 \sim_p \rho_1'$ and $\rho_2 \sim_p \rho_2'$, hence $\rho_1 \triangleright \rho_2 \sim_p \rho_1' \triangleright \rho_2'$. □

Theorem 3. Suppose $\{R_{pre}\} \Downarrow_1 \{R_1\}$ and $\{R_2\} \Downarrow_2 \{R_{post}\}$ are p-relational refinements such that R_1 composes with R_2 and $\Downarrow_1$, $\Downarrow_2$ are compatible. Then $\{\mathit{before}(R_{pre})\} \Downarrow_1 \triangleright \Downarrow_2 \{\mathit{after}(R_{post})\}$ is a p-relational refinement.

Proof. Let the two relational refinements be given. Clearly $\mathit{before}(R_{pre}) \subseteq \approx_p$. Condition 2) follows directly from 2) for $\{R_{pre}\} \Downarrow_1 \{R_1\}$. As per 3) above, suppose $t_1\ \mathit{before}(R_{pre})\ t_2$, $\sigma_1(0) = \mathit{before}(t_1)$, $\rho_1 \Downarrow \sigma_1$, $\rho_1 \sim_p \rho_2$, $\rho_2(0) \Downarrow \mathit{before}(t_2)$.

Assume that σ_1 completes in $\mathcal{M}_{c,2}$. Otherwise, the argument below is easily adapted. Then $\sigma_1 = \sigma_{1,1} \triangleright \sigma_{1,2}$, $\sigma_{1,1}(0) = \mathit{before}(t_1)$, $\rho_1 = \rho_{1,1} \triangleright \rho_{1,2}$, $\rho_2 = \rho_{2,1} \triangleright \rho_{2,2}$, $\rho_{1,1} \sim_p \rho_{2,1}$, $\rho_{1,2} \sim_p \rho_{2,2}$, $\rho_{1,1} \Downarrow_1 \sigma_{1,1}$, $\rho_{1,2} \Downarrow_2 \sigma_{1,2}$, and $\rho_{2,1}(0) \Downarrow_1 t_2$.

Let $\mathit{before}(t_1') = lst(\sigma_{1,1})$, a final state. By the relational refinement for $\Downarrow_1$ we find a run $\sigma_{2,1}$ such that $\sigma_{2,1}(0) = t_2$, $\rho_{2,1} \Downarrow_1 \sigma_{2,1}$, $\sigma_{1,1} \approx_p \sigma_{2,1}$, and $t_1'\ R_1\ t_2'$, for a final t_2' such that $\mathit{before}(t_2') = lst(\sigma_{2,1})$.

Let $t_1'' = \sigma_{1,2}(0)$ and $t_2'' = (init_2, d_2')$ where $init_2$ is the initial control state of $\mathcal{M}_{c,2}$ and d_2' is the store of t_2'.

By the definition of sequential composition, we obtain that $\mathit{before}(t_2') \to_c \mathit{after}(t_2'')$ in $\mathcal{M}_{c,1} \triangleright \mathcal{M}_{c,2}$ and $t_1''\ R_2\ t_2''$, as the relations compose. Also, since $\rho_{2,1} \Downarrow_1 \sigma_{2,1}$ we get $lst(\rho_{2,1}) \Downarrow_1 t_2'$, but then, by compatibility of $\Downarrow_1$ and $\Downarrow_2$, since $\rho_{2,2}(0)$ is uniquely determined by $lst(\rho_{2,1})$ we obtain $\rho_{2,2}(0) \Downarrow_2 t_2''$.

This allows to deploy the relational refinement property for $\Downarrow_2$. This yields a complete $\sigma_{2,2}$ such that $\sigma_{2,2}(0) = t_2''$, $\rho_{2,2} \Downarrow_2 \sigma_{2,2} \approx_p \sigma_{1,2}$, and $lst(\sigma_{1,2})\ R_{post}\ lst(\sigma_{2,2})$, since run σ_1 completes in $\mathcal{M}_{c,2}$ and thus also $\sigma_{1,2}$ is complete.

We have already established that $\sigma_2 = \sigma_{2,1} \triangleright \sigma_{2,2}$ composes, and that $\sigma_2(0) = t_2$. We also have $\rho_2 \Downarrow \sigma_2$, $\sigma_1 \approx_p \sigma_2$, and $lst(\sigma_1)\ \mathit{after}(R_{post})\ lst(\sigma_2)$. This concludes the proof. □

Theorem 4. If $\Downarrow$ is p-partition preserving for a relation R that relates final states only to final states, then $\{R\} \Downarrow \{R\}$.

Proof. Condition 11.1 is immediate by 9.1. Condition 11.2 is immediate by partition preservation. For 11.3, assume that $t_1\ R\ t_2$, $\sigma_1(0) = t_1$, $\rho_1 \Downarrow \sigma_1$, $\rho_1 \sim_p \rho_2$, $\rho_2(0) \Downarrow t_2$, and σ_1 is complete. If σ_1 is a singleton the conclusion follows directly, noting that if t_1 is final, also t_2 is by hypothesis. In the inductive case, let $\sigma_1(0) \xrightarrow{\beta_p} \sigma_1(1)$ and σ_1' be the first suffix of σ_1. Since $\rho_1 \Downarrow \sigma_1$ we find corresponding suffix ρ_1' and α such that $\rho_1(0) \xrightarrow{\alpha_p} \rho_1'(0)$ and $\rho_1' \Downarrow \sigma_1'$ and $\rho_1' \sim_p \rho_2'$ where ρ_2' is the first suffix of ρ_2. By IPR unwinding we then find t_2' such that $t_2 \xrightarrow{\beta_p} t_2'$, $\rho_2'(0) \Downarrow t_2'$, and $t_1'\ R\ t_2'$. By induction hypothesis we can then construct the run σ_2 called for by Def. 11.3 and we conclude. □