Computational Soundness for Interactive Primitives

Michael Backes, Esfandiar Mohammadi, and Tim Ruffing(✉)

CISPA, Saarland University, Saarbrücken, Germany
{backes,mohammadi}@cs.uni-saarland.de, tim.ruffing@mmci.uni-saarland.de

Abstract. We present a generic computational soundness result for interactive cryptographic primitives. Our abstraction of interactive primitives leverages the Universal Composability (UC) framework, and thereby offers strong composability properties for our computational soundness result: given a computationally sound Dolev-Yao model for non-interactive primitives, and given UC-secure interactive primitives, we obtain computational soundness for the combined model that encompasses both the non-interactive and the interactive primitives. Our generic result is formulated in the CoSP framework for computational soundness proofs and supports any equivalence property expressible in CoSP such as strong secrecy and anonymity.

In a case study, we extend an existing computational soundness result by UC-secure blind signatures. We obtain computational soundness for blind signatures in uniform bi-processes in the applied π-calculus. This enables us to verify the untraceability of Chaum's payment protocol in ProVerif in a computationally sound manner.

1 Introduction

Manual security analyses of cryptographic protocols are complex and error-prone. As a result, various automated verification techniques have been developed based on so-called Dolev-Yao models, which abstract cryptographic operations as symbolic terms obeying simple cancellation rules [12,26,35,36,38,40]. Numerous verification tools such as ProVerif [12] and APTE [26] are capable of reasoning about equivalence properties, e.g., strong secrecy and anonymity.

A wide range of these Dolev-Yao models is computationally sound, i.e., the security of a symbolically abstracted protocol entails the security of a suitable cryptographic realization [3,7,14,20,27,29,31,50,52]. However, virtually all of these computational soundness results are inherently restricted to non-interactive primitives such as encryption and signatures.

In contrast, *interactive cryptographic primitives* such as interactive zero-knowledge proofs [43], forward-secure key exchange [37], and blind signatures [25], have gained tremendous attention in the scientific community and widespread deployment in real systems.

The security of interactive primitives is often defined and established in the Universal Composability (UC) framework [17] or similar frameworks [8,44,48],

G. Pernul et al. (Eds.): ESORICS 2015, Part I, LNCS 9326, pp. 125–145, 2015.
DOI: 10.1007/978-3-319-24174-6_7

which allow to prove strong security guarantees in a *composable* manner [23,24,41]. In such frameworks, a primitive is secure if its execution is indistinguishable from a setting in which all parties have a private connection to an imaginary trusted machine, called *ideal functionality*, which performs the desired task locally and in a trustworthy manner.

For interactive primitives, ideal functionalities are a suitable abstraction, but for non-interactive primitives, DY-style abstractions have two significant advantages compared to a corresponding abstraction as an ideal functionality (e.g., for encryption schemes or digital signatures): first, as Dolev-Yao models do not incorporate shared memory, the verification of concurrent processes that use Dolev-Yao models is far more efficient, and second, the attacker is purely defined by symbolic rules and is thus much better suited for automatically deriving desired properties such as invariants. There is a rich literature on computationally sound DY-style abstractions. For example, Backes et al. introduced CoSP, a general framework for computational soundness proofs [3], which decouples the treatment of the Dolev-Yao model from the treatment of the language, e.g., the applied π-calculus or RCF. Proving x cryptographic Dolev-Yao models sound for y languages only requires $x + y$ proofs (instead of $x \cdot y$).

Previous work on computational soundness of verification tools for ideal functionalities [47] does not apply to protocols that combine interactive and non-interactive primitives with such computationally sound DY-style abstractions. In this work, we address this gap.

Contribution. We present a generic computational soundness (CS) result for UC-secure interactive primitives. Given a computationally sound Dolev-Yao model for non-interactive primitives and given UC-secure interactive primitives, we show the combined CS for the non-interactive *and* the interactive primitives. This allows us to handle protocols that combine interactive primitives with non-interactive primitives, e.g., protocols that encrypt blind signatures, or protocols that use interactive zero-knowledge proofs about ciphertexts. Our generic method is compatible with any CS result for non-interactive primitives that is cast in the CoSP framework for equivalence properties [6].

In a case study, we apply our method to a recent CS result [6]. We obtain the combined CS for (non-interactive) ordinary signatures and (interactive) blind signatures. The underlying CS result for non-interactive primitives supports uniform bi-protocols, i.e., protocol pairs that always take the same branches and differ only in the messages that they operate on. Consequently, our case study supports uniform bi-processes in the applied π-calculus. Finally, we conduct a computationally sound verification of the untraceability of Chaum's payment protocol [25] in ProVerif.

Remark on Supported Equivalence Properties. The aforementioned CS result [6] is so far the only result established in the CoSP framework for equivalence properties, and is limited to uniform bi-processes. As a result, it is unclear whether a larger class of equivalence properties can be expressed within the existing CoSP framework at all. Thus it is unclear whether our generic result could possibly

apply to a larger class of equivalence properties, even though we believe that our core ideas do not fundamentally rely on the specifics of the CoSP framework. The underlying problem is caused by the current embeddings of languages (such as the applied π-calculus) into CoSP. These embeddings do not provide a satisfying solution for concurrency, because they give the attacker full control over the scheduling of even internal scheduling decisions such as the scheduling of concurrent processes. Yet, CS results established with our generic method cover any equivalence properties covered by the underlying CS result for non-interactive primitives. Our work shares this limitation with other state-of-the-art CS results for equivalence properties [27–29].

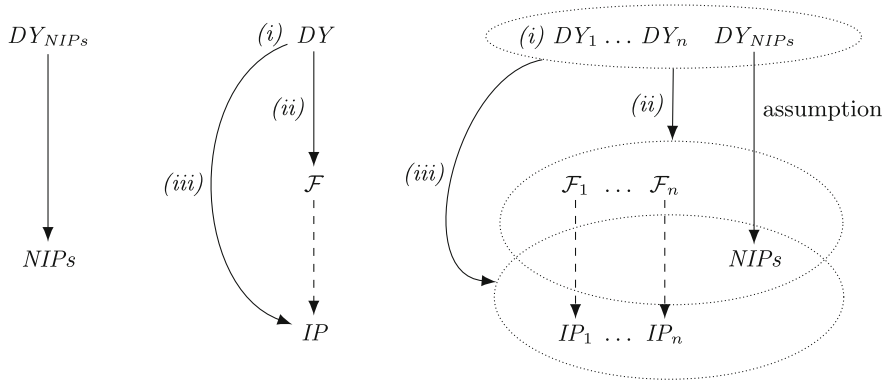
Overview. To facilitate understanding, we give a brief overview of the proof strategy taken in the paper. Typical CS results for non-interactive primitives ($NIPs$) state that the security of a protocol in a symbolic Dolev-Yao setting DY implies the security of the protocol in a computational setting, where real cryptographic algorithms are used instead of DY-style constructors and destructors (Fig. 1a).

Our proof strategy contains *two* computational settings: one setting with a computational ideal functionality $\mathcal{F}$ and one setting with its UC-secure cryptographic realization IP. For the sake of illustration, we start by explaining our approach with only a single interactive primitive (Fig. 1b).

(i) We transform the computational ideal functionality $\mathcal{F}$ to the symbolic setting by incorporating it into a Dolev-Yao model DY.
(ii) We show CS for the Dolev-Yao model with respect to the ideal functionality $\mathcal{F}$, which lives in the computational setting.
(iii) Under the assumption that IP is a UC-secure cryptographic realization of $\mathcal{F}$, we show CS for the Dolev-Yao model DY with respect to the cryptographic realization IP of the interactive primitive.

Next, we consider the setting of the paper (Fig. 1c). It consists of cryptographic realizations $IP_1, \ldots, IP_n$ of several interactive primitives and additionally of a set of cryptographic realizations $NIPs$ of several non-interactive primitives.

(i) We transform the computational ideal functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_n$ to the symbolic setting by incorporating them into Dolev-Yao models $DY_1, \ldots, DY_n$ (Sect. 5).
(ii) We then consider a unified model $(DY_1, \ldots, DY_n, DY_{NIP_s})$ that consists of the Dolev-Yao models for the interactive primitives as well as a single Dolev-Yao model DY_{NIP_s} that incorporates a set of non-interactive primitives. Under the assumption that DY_{NIP_s} is computationally sound with respect to the cryptographic realizations NIP_s, we show CS for the unified Dolev-Yao model with respect to the algorithms $(\mathcal{F}_1, \ldots, \mathcal{F}_n, NIP_s)$, i.e., with respect to the ideal functionalities plus the cryptographic realizations for the non-interactive primitives (Sect. 6).
(iii) Under the assumption that $IP_1, \ldots, IP_n$ are UC-secure realizations of $\mathcal{F}_1, \ldots, \mathcal{F}_n$, we show CS for the unified Dolev-Yao model with respect to the cryptographic realizations $(IP_1, \ldots, IP_n, NIP_s)$ (Sect. 8).



(a) CS for NIPs (b) CS for an IP (our approach) (c) Combined CS result for NIPs and IPs (detailed overview of our approach)

Fig. 1. An overview over different types of CS results for non-interactive primitives (NIPs) and interactive primitives (IPs). Solid arrows represent computational soundness. Dashed arrows represent UC-security.

2 Related Work

There is a successful line of research for computational soundness of trace properties [10,27,36,39] such as authentication and for static equivalence properties (i.e., against passive attackers) [2,11,45].

For equivalence properties against active attackers, however, there are only few previous results. The simulatable DY-style library of Backes et al. [4,7] was the first result to show computational soundness against active attackers and for equivalence properties on payloads. For this DY-style library it is not known how to formalize more properties than the secrecy of payloads.

Cortier and Comon-Lundh [27] show computational soundness for observational equivalence for symmetric encryption in the applied π-calculus. The scope of their work is incomparable to our work: their result is restricted to processes that do not contain private channels and abort if a conditional fails, whereas our result is restricted to uniform bi-processes.

An alternative approach to secure abstractions has recently been proposed by Bana and Comon-Lundh [9,10]. Instead of prescribing what an attacker can do and showing that no deviating computational behavior is possible, they pursue the approach to define what is impossible for an attacker (e.g., break the encryption) as first-order logic formulas over symbolic representations. Then, they specify the protocol in question and the existence of a potential attack in the same symbolic model. In their framework, inconsistency of a set of axioms implies security of the protocol. An inherent problem with this style of abstraction is the verification: it is not amenable to general-purpose DY-style verification tools, e.g., ProVerif [12] or Tarmarin [49].

With regard to the composability of computational soundness, Böhl et al. [14] show how a computational soundness result that has been obtained via deduction soundness [32] can be extended to hash functions, MACs, signatures, and symmetric and asymmetric encryption. While they add a set of non-interactive primitives to a given computational soundness result, we add a set of interactive primitives to a given computational soundness result.

There is other work that leverages the strength of the UC framework. Backes et al. [5] prove a computational soundness result for SMPC that is parametric in the same way as our result. However, their result considers only trace properties and is specific to SMPC. Canetti and Herzog [20], extended by Canetti and Gajek [19], show computational soundness for UC-secure key exchange protocols and signatures. There are two major differences to our work. First, their result is specific to the used primitives, while our result can be used for a large class of UC-secure interactive primitives. Second, even though their result holds for equivalence properties, the authors—in contrast to our work—do not show that their result can be combined with computationally sound Dolev-Yao models for non-interactive primitives.

Dahl and Damgård [33] show the computational soundness of a certain class of two-party protocols with respect to UC security, i.e., symbolic security implies computational UC security. While they use the UC framework to obtain strong, composable computational security for protocols that use certain non-interactive primitives, we use the UC framework to obtain ordinary, non-composable computational security for protocols that use UC-secure interactive primitives.

Küsters et al. [46] and Küsters et al. [47] leverage non-interference techniques for ideal functionalities in Java programs. While their method is capable of covering a large class of protocols and interactive primitives, it does not encompass DY-style abstractions of non-interactive primitives such as encryption. Thus, they have to represent all non-interactive primitives as ideal functionalities. Since the abstraction that uses ideal functionalities inherently contains shared memory between protocol parties, automated verification techniques are forced to deal with numerous interleaving runs and the verification costs significantly increase with the number of ideal functionalities. We show that UC-secure ideal functionalities of interactive primitives can be combined with computationally sound DY-style abstractions of non-interactive primitives, thereby minimizing the amount of ideal functionalities.

Fournet et al. [42] show computational soundness for the refinement type system F7 (and later F*) by relying on ideal functionalities as abstraction. The required type annotations serve as local invariants and make the verification feasible, even with shared memory and many interleaving runs. First steps have been undertaken towards automated type inference [53] for the type annotations; however, the automation is incomplete and still requires a significant amount of human interaction. As the type system is for the computational setting (against a computational attacker), automated type derivation is inherently harder than in a symbolic setting (against a symbolic attacker).

Delaune et al. [34] and Böhl and Unruh [15] transfer simulation-based security completely into the symbolic setting, including symbolic composition theorems. However, these results do not guarantee computational soundness.

3 Review of the CoSP Framework for Equivalence

We review the CoSP framework for equivalence properties [6], in which we cast our computational soundness result.

Symbolic Model. In CoSP, symbolic abstractions of protocols and of the attacker are formulated in a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$: a set of free functions $\mathbf{C}$, an infinite set $\mathbf{N}$ of nonces, a set $\mathbf{T}$ of terms (formed by constructors and nonces), and a set $\mathbf{D}$ of destructors, i.e., partial functions from terms to terms.

Protocols. Protocols are represented as infinite trees with the following nodes: *computation nodes* are used for drawing fresh nonces and applying constructors and destructors; *input nodes* and *output nodes* are used for sending and receiving terms; *control nodes* are used for allowing the attacker to schedule the protocol. A computation node is annotated with its arguments and has two outgoing edges: a yes-edge, used for the application of constructors, for drawing a nonce, and for the successful application of a constructor or destructor, and a no-edge, used for the failed application of a constructor or destructor. Nodes have explicit *references* to other nodes whose terms they use.

Symbolic Operations. We model the capabilities of the symbolic attacker as operations that the attacker can perform on protocol messages. A *symbolic operation* is a finite tree, whose nodes are labeled with constructors, destructors, nonces from the symbolic model $\mathbf{M}$, or pointers to messages that the protocol has sent to the attacker. There is a natural evaluation function $eval_O$ that evaluates a symbolic operation O in a bottom-up fashion on a list of terms, resulting in a term or the error symbol $\perp$.

Symbolic Execution. A *symbolic execution* is a path through a protocol tree. Formally, a symbolic execution of a protocol Π is a (finite) list of triples (V_i, ν_i, f_i) as follows. Initially, we have $V_1 = \varepsilon$, ν_1 is the root of Π, and f_1 is an empty partial function mapping node identifiers to terms. For every two consecutive tuples (V, ν, f) and (V', ν', f') in the list, let $\underline{\nu}$ be the nodes referenced by ν and define $\tilde{t}$ through $\tilde{t}_j := f(\tilde{\nu}_j)$. Figure 2 depicts a case distinction over ν for defining valid successors V', ν', and f'. Each V_i is called *symbolic view*.

Given a view V, V_{Out} is the list of terms t contained in $(\texttt{out}, t) \in V$. $V_{Out\text{-}Meta}$ is the list of terms l contained in $(\texttt{control}, (l, l')) \in V$. V_{In} (the *attacker strategy*) is the list of terms that contains only entries of V of the form $(\texttt{in}, (*, O))$ or $(\texttt{control}, (*, l'))$, and the first term has been masked with the symbol $*$.

switch ν **with**
  case computation node with constructor, destructor or nonce F
    if $m := F(\tilde{t}) \neq \bot$ **then**
      $V' := V$; $\nu' :=$ the `yes`-successor of ν; $f' := f(\nu := m)$
    else
      $V' := V$; $\nu' :=$ the `no`-successor of ν; $f' := f$
  case input node
    if there is a term $t \in \mathbf{T}$ and a symbolic operation O on $\mathbf{M}$ with $eval_O(V_{Out}) = t$ **then**
      $\nu' :=$ the successor of ν; $V' := V :: (\texttt{in}, (t, O))$; $f' := f(\nu := t)$
  case output node
    $\nu' :=$ the successor of ν; $V' := V :: (\texttt{out}, \tilde{t}_1)$; $f' := f$
  case control node with out-metadata l
    $\nu' :=$ the successor of ν with some in-metadata l'
    $f' := f$; $V' := V :: (\texttt{control}, (l, l'))$

Fig. 2. Symbolic execution

Symbolic Knowledge and Equivalent Views. The *symbolic knowledge* of the attacker comprises the results of all the symbolic operations that the attacker can perform on messages output by the protocol. Given a view V, the *symbolic knowledge* K_V is a function from symbolic operations on $\mathbf{M}$ of arity $|V_{Out}|$ to $\{\top, \bot\}$, where $\top$ unifies all results of $eval_O(V_{Out})$ that are not $\bot$.

Two views are *equivalent* if they *(i)* have the same structure (i.e., the same order of `out`, `in`, and `control` entries), *(ii)* have the same out-metadata (i.e., $V_{Out\text{-}Meta} = V'_{Out\text{-}Meta}$), and (*iii*) lead to the same knowledge (i.e., $K_V = K_{V'}$).

Symbolic Indistinguishability. Finally, we define two protocols to be *symbolically indistinguishable* if the two protocols lead to equivalent views when faced with the same attacker strategy.

Computational Implementation. On the computational side, the constructors and destructors in a symbolic model are realized with cryptographic algorithms, which we call *computational implementations*. A computational implementation is a family $\mathbf{A} = (A_x)_{x \in \mathbf{C} \cup \mathbf{D} \cup \mathbf{N_P}}$ of deterministic polynomial-time algorithms $\mathbf{A}_F$ for each constructor or destructor $F \in \mathbf{C} \cup \mathbf{D}$ well as a probabilistic polynomial-time (ppt) algorithm A_N for drawing protocol nonces $N \in \mathbf{N}$.

Computational Execution. The *computational execution* of a protocol is the interaction between a ppt machine called the *computational challenger* and a ppt attacker $\mathcal{A}$. The transcript of the execution contains the computational counterparts of a symbolic execution. The computational challenger traverses the protocol tree and interacts with the attacker: at a computation node the corresponding algorithm is run and depending on whether the algorithm succeeds or outputs $\bot$, either the yes-branch or the no-branch is taken; at an output node, the message is sent to the attacker; at an input node a message is received by the attacker; and at a control node the attacker is asked which edge to take.

Computational Indistinguishability. The CoSP framework for indistinguishability properties [6] uses *termination-insensitive computational indistinguishability* [54]

(tic-indistinguishability) to capture that two protocols are computationally indistinguishable. In comparison to the standard notion of indistinguishability, tic-indistinguishability does not require the interactive machines to be polynomial-time; instead, it only considers decisions that were made for polynomially-bounded prefixes of the interaction.

Given two machines A, B and a polynomial p, we write $\Pr[\langle A|B\rangle \Downarrow_{p(k)} x]$ for the probability that the interaction between A and B terminates within $p(k)$ steps and B outputs x.

Two machines A and B are *tic-indistinguishable* [54] *for a machine* $\mathcal{A}$ ($A \approx^{\mathcal{A}}_{tic} B$) if for all p, there is a negligible function μ such that for all $z, a, b \in \{0,1\}^*$ with $a \neq b$, $\Pr[\langle A(k)|\mathcal{A}(k,z)\rangle \Downarrow_{p(k)} a] + \Pr[\langle B(k)|\mathcal{A}(k,z)\rangle \Downarrow_{p(k)} b] \leq 1 + \mu(k)$. Here, z represents an auxiliary string. We call A and B *tic-indistinguishable* ($A \approx_{tic} B$) if $A \approx^{\mathcal{A}}_{tic} B$ for all ppt machines $\mathcal{A}$.

We define a pair of protocols to be *computationally indistinguishable* if the corresponding challengers are tic-indistinguishable. With the previously introduced notions, we define *computational soundness*, which states that symbolic indistinguishability implies computational indistinguishability.

Definition 1 (Computational Soundness). *Let a symbolic model* $\mathbf{M}$ *and a class* P *of efficient protocols be given. A computational implementation* $\mathbf{A}$ *of* $\mathbf{M}$ *is* computationally sound *for* $\mathbf{M}$ *if every pair of protocols in* P *is computationally indistinguishable whenever it is symbolically indistinguishable.*

4 Review of the UC Framework

We briefly review the UC framework [17], as we use it to establish our computational soundness result. The UC framework is designed to enable a modular analysis of security protocols. In this framework, the security of a protocol ϕ is defined by comparing the protocol with a setting in which all parties have a private connection to a trusted machine $\mathcal{F}$, called *ideal functionality*, which performs the desired protocol task locally. The ideal functionality $\mathcal{F}$ serves as an abstraction of this task. A protocol ϕ *UC-realizes* an ideal functionality $\mathcal{F}$ if for all ppt machines $\mathcal{A}$ (the *attacker*) there is a ppt machine $\mathcal{S}$ (the *simulator*) such that no ppt machine $\mathcal{Z}$ (the *environment*) can distinguish an interaction with ϕ and $\mathcal{A}$ from an interaction with $\mathcal{F}$ and $\mathcal{S}$. The environment is connected to the protocol and the attacker in the real setting or to the functionality and the simulator in the ideal setting.

Each machine M has two different input tapes. First, it has a *subroutine input tape*, which is used when another machine M', e.g., the environment $\mathcal{Z}$, calls them as a local subroutine. Second, each machine has a *network tape*, which is connected to the attacker $\mathcal{A}$ or the simulator $\mathcal{S}$.

The order in which computations are performed in UC is as follows. The execution starts with the environment $\mathcal{Z}$. Its execution pauses whenever it writes a message to an input tape of another machine M'. At this point, M' is activated and runs until M', in turn, writes a message to a tape of another machine M''.

5 Ideal Functionalities in the Symbolic Model

We abstract interactive primitives in the symbolic model as ideal functionalities. As a simple example, consider two parties A and B running an interactive key exchange. For example in the applied π-calculus, this is modeled as three parallel processes $A \mid P \mid B$, where P is the symbolic key exchange abstraction that generates a fresh key and sends it to both parties on private channels.

Formalizing Ideal Functionalities. An ideal functionality $\mathcal{F}$ in CoSP is symbolically abstracted as a CoSP protocol with only computation nodes; it will serve as a subroutine in another protocol. Technically, $\mathcal{F}$ excepts five parameters *state*, *sid*, *sender*, *input*, and *rand* as input. Since destructors and algorithms in CoSP are stateless as opposed to machines in UC, we model the state explicitly by the first parameter. A message sent to $\mathcal{F}$ is modeled by the parameters *sender* and *input*, where *sender* represents an identifier of the sending party and *input* the contents. If the message comes from the attacker, *sender* is *null*(). The *sid* parameter gives $\mathcal{F}$ access to its session id. The last parameter *rand* is a fresh randomness for $\mathcal{F}$.

For the output, $\mathcal{F}$ contains *result nodes*. They indicate the end of an invocation of $\mathcal{F}$, and the messages computed by the reached result nodes encode $\mathcal{F}$'s output.

Ideal Functionalities in the Symbolic Model. An ideal functionality yields a potentially complex destructor $D_{\mathcal{F}}$ with the same behavior as the symbolic operation. To combine ideal functionalities for interactive primitives with Dolev-Yao models for non-interactive primitives, we formulate the aforementioned process P, which models the ideal task, essentially as an application of the destructor $D_{\mathcal{F}}$.

An application of the destructor corresponds to a message sent to the UC machine implementing the ideal functionality. This allows a CoSP protocol to use the ideal functionality like a subroutine (as in the UC framework).

Definition 2 (Ideal Destructor). *Let* $\mathbf{F}$ *be an ideal model (a set of ideal functionalities) based on the symbolic model* $\mathbf{M} = (\boldsymbol{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$*, and let* $\mathcal{F} \in \mathbf{F}$*.*

The ideal destructor of $\mathcal{F}$ *is a destructor* $D_{\mathcal{F}} : \mathbf{T}^5 \to \mathbf{T}$ *with* $(t_{state}, t_{sid}, t_{sender}, t_{input}, t_{rand}) \mapsto t_{res}$*. Here* t_{res} *is the term produced by the reached result node in the symbolic execution of* $\mathcal{F}$ *with parameters* $t_{state}, t_{sid}, t_{sender}, t_{input}, t_{rand}$*.*

Extended Symbolic Model. Given destructors $D_{\mathcal{F}}$ for $\mathcal{F} \in \mathbf{F}$ and a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ (for non-interaction primitives), the *extended symbolic model* is $\mathbf{M_F} := (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D_F})$ where $\mathbf{D_F} := \mathbf{D} \cup \{D_{\mathcal{F}}/5 \mid \mathcal{F} \in \mathbf{F}\}$.

6 Ideal Functionalities in the Computational Model

As a first step to prove computational soundness, we explain how to leverage existing computational soundness results for non-interactive primitives. The formulation of $\mathcal{F}$ as a destructor $D_{\mathcal{F}}$ enables us to consider an ideal computational

execution, in which $D_{\mathcal{F}}$ is implemented by a computational variant (called *the canonical algorithm*) $A_{\mathcal{F}}$ of $\mathcal{F}$.

Definition 3 (Canonical Algorithm). *Let an extended symbolic model* $\mathbf{M_F}$ *based on* $\mathbf{M}$ *and a computational implementation* $\mathbf{A}$ *of* $\mathbf{M}$ *be given. The* canonical algorithm of $\mathcal{F}$ *is the algorithm* $A_{\mathcal{F}} : \mathbb{N} \times (\{0,1\}^*)^5 \to \{0,1\}^*$ *with* $(b_{state}, b_{sid}, b_{sender}, b_{input}, b_{rand}) \mapsto b_{res}$. *It runs the an unbounded variant of the computational execution of* $\mathcal{F}$ *and stops if the first reached result node is reached. (An attacker is not involved, because* $\mathcal{F}$ *contains only computation nodes.) The output* b_{res} *is the bitstring computed by the that node. The first argument of* $A_{\mathcal{F}}$ *represents the security parameter and the other arguments determine the inputs.*

Ideal Implementations. Recall that we extend a symbolic model $\mathbf{M}$ by ideal destructors $D_{\mathcal{F}}$, resulting in a new symbolic model $\mathbf{M_F}$. Analogously, we extend a computational implementation $\mathbf{A}$ for $\mathbf{M}$ by the canonical algorithms $A_{\mathcal{F}}$, given that each $A_{\mathcal{F}}$ is computable in polynomial-time. Writing $A_{\mathcal{F}}$ instead of $A_{D_{\mathcal{F}}}$, the resulting *ideal implementation* $\mathbf{A_F} := (A_x)_{x \in \mathbf{C} \cup \mathbf{D_F} \cup \mathbf{N}}$ implements $\mathbf{M_F}$.

Computational Soundness for the Ideal Functionalities. Assume we have a computational soundness result for the implementations of non-interactive primitives (e.g., A_{enc} and A_{dec}). That is, the Dolev-Yao model without the special destructor $D_{\mathcal{F}}$ (only consisting of *enc* and *dec*) is computational sound. Then we can show that also the Dolev-Yao model *with* the destructor $D_{\mathcal{F}}$ is computationally sound given that $D_{\mathcal{F}}$ is implemented by $A_{\mathcal{F}}$.

The following lemma states the computational soundness of the ideal functionalities, which are ideal implementations in the computational model. To establish the lemma, we need some natural *protocol conditions* (Appendix A). They ensure *(i)* that inputs and outputs of the ideal functionalities are actually plugged to input and output nodes, *(ii)* that sessions and state are handled correctly and *(iii)*, that fresh randomness is provided for each call of the ideal functionality (the *rand* argument). Within a concrete symbolic calculus, syntactic criteria that imply the protocol conditions can be introduced.

Lemma 1 (Soundness of Ideal Implementations). *Let* $\mathbf{M_F}$ *be an extended symbolic model based on* $\mathbf{M}$*, and let* $\mathbf{A}$ *be a computationally sound implementation of* $\mathbf{M}$ *for protocols* Π *in a class of protocols* P *that fulfills the protocol conditions (Appendix A). Suppose that* $\mathbf{M_F}$ *has the ideal implementation* $\mathbf{A_F}$*. Suppose that for every* $\Pi \in P$*, we have that the full protocol* $\hat{\Pi}$ *is in* P*.*

Then the ideal implementation $\mathbf{A_F}$ *is computationally sound for* $\mathbf{M_F}$ *and* P*.*

For the proof of the lemma (see the full version [30]), we construct a *full* protocol $\hat{\Pi}$ from Π by inlining the calls to ideal implementations: Each computation node ν with destructor $D_{\mathcal{F}}$ is replaced by the tree of the ideal functionality $\mathcal{F}$. The parameters of $\mathcal{F}$ are connected to the nodes referenced by ν and the subtree rooted at the yes-successor of ν is appended to every result node of $\mathcal{F}$. The proof basically uses the fact that the full protocol $\hat{\Pi}$ does not use any of the ideal destructors $D_{\mathcal{F}}$. Thus the computational soundness of $\mathbf{M}$ applies.

7 Real Protocols in CoSP

In the ideal computational execution, the interactive primitives are not implemented by their actual cryptographic realizations: while $A_{\mathcal{F}}$ is computational, it is merely an algorithmic representation of the ideal functionality $\mathcal{F}$. To close the gap to a real interactive protocol, we assume that there is a an interactive protocol ϕ that is a UC-secure realization of $\mathcal{F}$.

Formally, we define a *real algorithm* A_ϕ, which has the same interface as an algorithm $A_{\mathcal{F}}$, i.e., it takes bitstrings $b_{state_1}, b_{sid}, b_{sender}, b_{input}, b_{rand}$ as input and produces a triple $(b'_{state_1}, b_{receiver}, b_{output})$ of bitstrings as output.

The arguments directly correspond to the arguments of canonical algorithms of ideal functionalities, and the same intuition should be applied in general. In contrast to an ideal functionality however, there is no "joint state" between the participants of a real protocol. To enforce this statically, the state argument $state_1$ only represents the state of one single protocol party P.

Since the algorithms can output a state, each UC protocol can be reformulated as a real algorithm in our model. If we have a cryptographic realization for every $\mathcal{F}$ in an ideal model $\mathbf{F}$, we can extend a computational implementation $\mathbf{A}$ to a *real implementation* $\mathbf{A}_\Phi$. $\mathbf{A}_\mathbf{F}$ and $\mathbf{A}_\Phi$ allow us to compare an ideal implementation of the interactive primitives with a real one, as in the UC framework.

To simplify notation, we write A_θ to denote an interactive algorithm that is either the canonical algorithm for an ideal functionality $\theta = \mathcal{F}$ or the algorithm for a real protocol $\theta = \phi$.

To make use of the UC framework, we first bring interactive algorithms to the UC setting by constructing machines in the UC sense from them. We write $\mu(\theta)$ for the machine that runs A_θ internally. It basically provides an interface to a computational CoSP execution that activates $\mu(\theta)$ whenever A_θ should be executed. In case that $\theta = \phi$ is a real algorithm, we require that $\mu(\theta)$ separates the state of distinct protocol parties. This models a real protocol execution as the parties can only communicate via the attacker.

8 Computational Soundness for Interactive Primitives

As a final step, we prove computational soundness for the interactive primitives. We leverage the composability of UC security: If the real protocol ϕ is a UC-secure realization of the ideal functionality $\mathcal{F}$, then instances of $\mathcal{F}$ used in a larger protocol can be replaced securely by instances of ϕ.

Using the UC framework, we would like to show an analogous result in our model: if the machine $\mu(\phi)$ is a UC-secure realization of $\mu(\mathcal{F})$, then instances of the canonical algorithm $A_{\mathcal{F}}$ used in a larger protocol can be replaced securely by instances of the real algorithm A_ϕ. Consequently, if $A_{\mathcal{F}}$ is a computational sound implementation of the destructor $D_{\mathcal{F}}$, then A_ϕ is a computational sound implementation of the destructor $D_{\mathcal{F}}$.

We require that the ideal functionality $\mathcal{F}$ and the real protocol ϕ adhere to few technical conditions. We explain why these conditions are necessary, what they exactly are, and why they do not constitute fundamental restrictions.

Problems. Our goal is to consider a UC environment $\mathcal{Z}$ that runs a computational CoSP execution but does not handle computation nodes with the destructor $D_{\mathcal{F}}$. Instead, this task should be delegated to a UC machine. For a interactive algorithm A_θ however, the standard machine $\mu(\theta)$ does not suffice for this purpose:

One problem stems from the fact that in the CoSP execution run by $\mathcal{Z}$, communication with the attacker happens only when an input or an output node is reached in the CoSP protocol. However, the machine $\mu(\theta)$ could just not adhere to this restriction and exchange messages with the attacker machine even if the CoSP execution run by $\mathcal{Z}$ does not currently process an input or an output node.

The second problem concerns only the ideal setting, and consists of a lack of information of the environment $\mathcal{Z}$. The CoSP view output by the environment must contain the communication between $\mathcal{F}$ and the simulator $\mathcal{S}$, but this communication is not visible for $\mathcal{Z}$ in UC. In fact, $\mu(\mathcal{F})$ and $\mathcal{S}$ can exchange arbitrary messages without even noticed by $\mathcal{Z}$.

To understand why this second problem does not arise in the real setting, consider w.l.o.g. the dummy attacker $\mathcal{A}_d$ that will only relay communication between the environment $\mathcal{Z}$ and the machine $\mu(\phi)$.[1] Thus $\mathcal{Z}$ is informed about all communication between $\mu(\phi)$ and $\mathcal{A}_d$.

Technical Remedy. In the proof of our main theorem, we build a wrapper machine $\tilde{\mu}(\theta)$ around every machine $\mu(\theta)$. It reports to the environment $\mathcal{Z}$ that communication took place between $\mu(\theta)$ and the attacker, but not what communication. To ensure that the wrapper machine can be used instead, we assume that the ideal functionality $\mathcal{F}$ and the real protocol ϕ are *good*, i.e. we require them to adhere to one technical condition each. We describe the conditions here only informal. Exact definitions can be found in the full version [30].

Condition on the Ideal Functionality. The condition on the ideal functionality basically states that the simulator can force $\mu(\mathcal{F})$ to produce output to the environment. This helps in a situation where the real attacker sends a message to $\mu(\phi)$, which sends in turn a message m to the environment. In the ideal setting, the simulator must force $\mu(\mathcal{F})$ to send a message indistinguishable from m to the environment immediately, without replying to the simulator first, because such a reply would be reported to the environment by the wrapper machine $\tilde{\mu}(\mathcal{F})$.

Condition on the Real Protocol. The condition on the real protocol ensures that a message from the environment to $\mu(\phi)$ leads to a output message to the attacker immediately. Here the excluded situation is that the real protocol machine $\mu(\phi)$ answers a request from the environment immediately, whereas the ideal machine

[1] Canetti shows [17] that it suffices to prove security against a dummy attacker $\mathcal{A}_d$, which acts as proxy for the environment $\mathcal{Z}$.

$\mu(\mathcal{F})$ would have to talk to the simulator first, which is not possible without being reported to the environment by the wrapper machine $\tilde{\mu}(\mathcal{F})$.

Discussion. We stress that both the conditions for the ideal functionality and the conditions for the real protocol are rather technical requirements instead of severe restrictions. The conditions are fulfilled by virtually all natural interactive primitives such as blind signatures [41], zero-knowledge proofs [16], oblivious transfer [17], and secure function evaluation [17]. In some cases, a technical reformulation of the ideal functionality or the real protocol is necessary. For instance, a real protocol that provides access to its results via an request interface would violate our condition; however it can be formulated such that it reports the results to the environment without being asked.

Furthermore, the condition for the ideal functionality seems to exclude adaptive corruption models. The reason is that these models typically require the ideal functionality to report parts of its internal state corresponding to a corrupted party to the simulator, after the simulator decides to corrupt that party. Still, by modeling corruption in a slightly different but still natural manner, a reformulation is possible. We refer to the full version [30] for a detailed discussion.

The main cause for the two technical conditions is a discrepancy between the UC framework and the CoSP framework. We use the latter in order to leverage existing results [6]. As a result, we inherit the restrictions that stem from the way previous embeddings resolved non-deterministic choices, e.g., concurrent computations: the distinguisher has full control over all scheduling decisions of concurrent computations and is fully aware of the execution state with respect to control flow. As a consequence the distinguisher can observe that communication between the simulator and the ideal functionality takes place. This is in contrast to the UC framework, where the distinguisher (the environment) cannot observe this communication.

Main Result. The main theorem, which is proven in the full version [30], states that we can extend a computational soundness result for equivalence properties to a computational soundness result for interactive primitives that are soundly abstracted by ideal functionalities.

Theorem 1. *Let* $\mathbf{M_F}$ *be an extended symbolic model based on* $\mathbf{M}$*, and let* $\mathbf{A}_\Phi$ *be a computational implementation of* $\mathbf{M_F}$ *based on* $\mathbf{A}$*. Let* P *be a class of CoSP protocols such that every protocol in* P *fulfills the protocol conditions for interactive primitives (Appendix A). Suppose that every* $\mathcal{F} \in \mathbf{F}$ *is a good ideal functionality and every* $\phi \in \Phi$ *is a good real protocol (see the full version [30]). Suppose that for every ideal functionality* $\mathcal{F} \in \mathbf{F}$ *and the corresponding real protocol* $\phi \in \Phi$*, we have that* $\mu(\phi)$ *UC-realizes* $\mu(\mathcal{F})$*.*

If $\mathbf{A}$ *is a computationally sound implementation of* $\mathbf{M}$ *for* P *with respect to equivalence properties, then* $\mathbf{A}_\Phi$ *is a computationally sound implementation of* $\mathbf{M_F}$ *for* P *with respect to equivalence properties.*

Limitations. While our result can be used with a wide range of natural two-party and multi-party primitives in the UC framework, it comes with several limitations.

First, since UC security is a very strong notion, some interactive primitives cannot be achieved in the UC framework, or they can only achieved under additional assumptions, or they require less efficient protocols than under ordinary security definitions. For instance, zero-knowledge proofs and oblivious-transfer are impossible without additional assumptions [17,22]. However, these primitives are possible if a common reference string (CRS) and authenticated message transfer (e.g., using a public-key infrastructure) is assumed [17,18]. Another example is UC-secure key exchange, which is, depending on the formulation, strictly stronger than standard key exchange [21], and thus requires less efficient protocols. We refer to Canetti [17, 2005 revision] for a comprehensive overview over different primitives in the UC framework.

Second, our result cannot be used to abstract *non-interactive* primitives using the UC framework. (While such abstractions are not desirable for automated verification (see Sect. 2), they might be desirable to achieve composability.) The culprit is the condition for the real protocol. Recall that it imposes that the protocol does not immediately reply to the environment, i.e., to the caller. While this is a natural assumption for interactive primitives,[2] it is very unnatural for non-interactive primitives. Indeed, all meaningful "protocols" that realize ideal functionalities for public-key encryption and signatures proposed by Canetti [17] violate the condition that we impose upon real protocols, because they perform the cryptographic operation locally without network communication involving the attacker. However, we are not aware of any natural *interactive* protocol, which cannot be reformulated to adhere to the technical conditions outlined above.

9 Case Study: Untraceable Payments

Untraceable payments, proposed by Chaum [25], allow a payer to perform a payment to a payee, say a shop, via a bank. In Chaum's protocol, a payer basically buys a coupon, i.e., a signed random bitstring, such that the bank does not know the coupon. Then, the user can pay with this coupon at a shop, and the shop will check the validity of the coupon with the bank. As the main cryptographic tool for untraceable payments Chaum suggests *blind signatures*, which guarantee that the bank neither learns the message nor the signature while signing the message.

We verify the untraceability of the payments with the verification tool ProVerif [12] using a UC-secure abstraction of blind signatures by Fischlin [41]. Our computational soundness theorems entail that the result of ProVerif's verification carries over to the computational realization of untraceable payments.

[2] It is the very nature of interactive protocols that a message is sent on the network, i.e., the protocol activates the attacker, before it reports results to the caller.

Ideal Blind Signatures and Their Realization. Our ideal functionality $\mathcal{F}$ for blind signatures models a scenario with one bank BANK and n users USER_i. It consists of a setup phase and offers a signing oracle to the users. In the setup phase, the bank generates signature keys or receives them from the attacker. Then, the functionality distributes the verification keys to the bank BANK and all users.

Upon a signing request $(\texttt{Sign}, sid, m, vk')$ from USER_i, the functionality for an honest USER_i waits for the attacker to deliver the message, signs the message m using the stored signing key sk, and sends the result to USER_i. For a malicious USER_i, the ideal functionality $\mathcal{F}$ informs $\mathcal{A}$ about the message. Then it informs the bank that a signature is being requested.

Fischlin [41] showed the existence of a protocol that UC-realizes an ideal functionality for blind signatures under standard cryptographic assumptions. Our functionality differs in details from the one in [41]. Using Fischlin's construction ϕ, we can prove realization if we require that the signature scheme, used by the ideal functionality is unforgeable. The proof is essentially only a modification of the proof in [41], and can be found in the full version [30].

Computational Soundness of Signatures and Blind Signatures. We rely on a symbolic model $\mathbf{M}_{sig}$ for digital signatures. (It contains also public-key encryption, which we do not use). The model is computationally sound in CoSP for uniform bi-protocols with respect to a computational implementation $\mathbf{A}_{sig}$ [6]. The aforementioned ideal functionality $\mathcal{F}$ for blind signatures and its UC-secure realization ϕ yields a CoSP destructor $D_{\mathcal{F}}$ and a real implementation A_{ϕ}, respectively. Symbolically, we extend $\mathbf{M}_{sig}$ by $D_{\mathcal{F}}$, resulting in $\mathbf{M}_{sig,bsig}$. Computationally, we extend $\mathbf{A}_{sig}$ by A_{ϕ}, resulting in $\mathbf{A}_{sig,bsig}$. Finally, Theorem 1 and the computational soundness for signatures in uniform bi-processes in the applied π-calculus [6, Theorem 3] yield the computational soundness of our case study.

Theorem 2. *Let Q be an applied-π bi-process on the symbolic model $\mathbf{M}_{sig,bsig}$ that is randomness-safe [6] and fulfills the protocol conditions (Appendix A). If Q is uniform, then the computational bi-protocol corresponding to Q, which uses the computational implementation $\mathbf{A}_{sig,bsig}$, is computationally indistinguishable.*

Uniform Bi-protocols. We leverage a computational soundness result [6], which is restricted to uniform bi-protocols. Bi-protocols are pairs of protocols that always take the same branches and differ only in the messages that they operate on.

Uniform bi-protocols cannot express equivalence between protocols with processes of different structure. For example, consider a protocol Π_1 with a client process that sends some request to a server twice. If the requests are unlinkable to each other, then formally, the client process is equivalent to a protocol Π_2 with the parallel composition of two client processes that send one request each. However, Π_1 and Π_2 have different structure, i.e., they differ in more than the terms they operate on. Thus a uniform bi-protocol cannot model this unlinkability.

A uniform bi-process [13] in the applied π-calculus is the counterpart of a uniform bi-protocol in CoSP. A bi-process is a pair of processes that only differ in the terms they operate on. Formally, they contain expressions of the form

choice$[a, b]$, where a is used in the left process and b is used in the right one. A bi-process Q can only reduce if both its processes can reduce in the same way. We consider the variant of the applied π-calculus used for the original CoSP embedding [3]. The operational semantics is defined in terms of *structural equivalence* ($\equiv$) and *internal reduction* ($\rightarrow$); for a precise definition of the applied π-calculus, we refer to [12]. Formally, a bi-process Q in the applied π-calculus is *uniform* if $\text{left}(Q) \rightarrow R_{\text{left}}$ implies that $Q \rightarrow R$ for some bi-process R with $\text{left}(R) \equiv R_{\text{left}}$, and symmetrically for $\text{right}(Q) \rightarrow R_{\text{right}}$ with $\text{right}(R) \equiv R_{\text{right}}$.

Verifying Untraceability in ProVerif. ProVerif [12] is an automated verification tool that can prove the uniformity of bi-processes in the applied π-calculus [1]. We use a wrapper process (Fig. 3) in the applied π-calculus that enforces the protocol conditions from Appendix A.

This wrapper maintains the session identifier in a way that is compatible with UC, maintains the state of the ideal functionality, and offers an interface that is compatible with our computational soundness result for interactive primitives.

Model in ProVerif. We used ProVerif to model a small untraceable payment system with two payers and one payee, say a shop owner. We modeled the scenario in which the bank is compromised and two honest payers purchase coupons. Then, one of the payers uses the coupon, and the shop owner leaks the coupon to the bank by cashing it. We modeled the scenario as a process for the ideal functionality of blind signatures and one bi-process that models both the payers and the shop owner. Since we consider untraceability, the bank is not modeled explicitly, it is the attacker.

To help ProVerif terminate, we replaced the process that executes the very complex destructor $D_{\mathcal{F}}$ by an equivalent process consisting of a series of *let* and *if* commands. As there is no communication in the equivalent process, the modified protocol differs only in the fact that it offers more scheduling possibilities: the attacker can schedule other processes in the middle of the computation, which is not possible in the unmodified process with the atomic destructor $D_{\mathcal{F}}$. Thus any attack possible on the unmodified process is also possible on the modified one.

Our code [51] has about 200 lines of code. ProVerif proves uniformity within under a second on a machine with an Intel i7 CPU (2 GHz) and 4 GB RAM.

Even though the symbolic model $\mathbf{M}_{sig}$ includes a length function, we did not include the corresponding length destructor in the case study, because ProVerif does otherwise not terminate. Nevertheless, our verification is computationally sound, because the length functions in the underlying result [6] are only necessary to handle public-key encryption, which is not part of $\mathbf{M}_{sig}$ in our case study.

Formally, we present the following lemma, which can be useful beyond our case study when applying the result of [6]. The lemma states that we can ignore a destructor d in the symbolic analysis of a bi-protocol, if *(i)* d is not used in the bi-protocol and *(ii)* d can be simulated using other destructors and constructors.

Lemma 2. *Let* $\mathbf{M} = (\boldsymbol{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ *be a symbolic model. Consider the model* $\mathbf{M}' = (\boldsymbol{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}')$ *with* $\mathbf{D}' = \mathbf{D} \setminus \{d\}$. *Let* Π *be a bi-protocol on* $\mathbf{M}'$.

Assume there is a function simD with the following property: given any symbolic operation O_d in $\mathbf{M}$*, and any view* V*, but only the symbolic knowledge* $K_V^{\mathbf{M}'}$ *of* $\mathbf{M}'$*, simD outputs a symbolic operation* O_{simD} *on* $\mathbf{M}'$ *that simulates* d*, i.e.,* $O_{simD}(\underline{t}) = d(O_d(\underline{t}))$ *for all sequences of terms* $\underline{t} \in \mathbf{T}^*$*.*

Then Π *is indistinguishable in the symbolic model* $\mathbf{M}$ *if it is indistinguishable in the symbolic model* $\mathbf{M}'$*.*

In the full version [30], we prove the lemma and give a function *SimLength* that simulates the destructor "*length*" used in [6].

Plugging everything together, the successful ProVerif verification, Theorem 2, and Lemma 2 prove for our case study bi-process that any realization adhering to the implementation conditions of $\mathbf{A}_{sig,bsig}$ [6] is computationally indistinguishable.

```
let functionalityWrapper_F =
  (* initialize *)
  in(initInputC_F, any_value);
  new attSessC; new protSessInC; new commonSessC;
  out(attC, attSessC);
  out(initOutputC_F, protSessInC);
  new stateC; new resC; new sid;
  (
    (* initialize state *)
    out(stateC, null()
  )
  |
  !(
    (
      (* receive from attacker *)
      in(attSessC, attInput);
      out(commonSessC, (attInput, attSessC))
    ) | (
      (* receive from protocol party *)
      in(protSessInC, (protInput, protParty));
      out(commonSessC, (protInput, protParty))
    ) | (
      (* handle both types of input *)
      in(commonSessC, (input, sender));
      in(stateC, state);
      new rand;
      (* execute ideal functionality *)
      let (state', (receiver, output)) =
        D_F(state, sid, input, sender, rand) in
      out(resC, (state',(receiver, output)))
    ) | (
      (* process outputs *)
      in(resC, (state', (receiver, output)));
      out(receiver, output);
      out(stateC, state')
    )
  )
).
```

Fig. 3. The wrapper for the ideal functionality

Acknowledgments. We thank the reviewers for their helpful and valuable comments. This work was supported by the German Ministry for Education and Research (BMBF)

through funding for the Center for IT-Security, Privacy and Accountability (CISPA) and the German Universities Excellence Initiative.

A Protocol Conditions

Given a CoSP protocol Π, consider the directed graph $ref(\Pi)$ which has the property that a node ν_s is successor of a node ν_p if and only if ν_p references ν_s in its annotations. It is a tree because nodes may only reference nodes which are on the path to the root in the protocol tree. For a node ν of Π, the *reference tree of* ν is the subtree of $ref(\Pi)$ which is rooted at ν and reachable from there. We say that a node ν *is determined* by a node ν' if on the path (through $ref(\Pi)$) from ν to ν' exclusive, every node has exactly one successor. The corresponding path is called *reference path to* ν'.

We require that the following criteria are met for for all ideal functionalities $\mathcal{F}$ and all computation nodes ν with a destructor $D_{\mathcal{F}}$.

1. We say that two interactive nodes *belong to the same session* if and only if one of them is contained in the reference tree of the *state* argument node of the other. Two interactive nodes with destructor $D_{\mathcal{F}} \in \mathbf{F}$ are required to be part of the same session if and only if they have the same *sid* argument node.
2. Let ν' be the bottom-most predecessor of ν that belongs to the same session, if any. Let be the output computed by ν' in a computational execution of the protocol. On the path from ν' to ν, there are the following nodes:
   - Three computation nodes ν_{state}, $\nu_{receiver}$ and ν_{output} which produce the bitstrings *state*, *receiver* and *output*, respectively. They are determined by ν'. Their reference paths to ν' contain only computation nodes and ν is in the yes-subtree of all these computation nodes.
   - If and only if in a computational execution of the protocol, the bitstring produced by $\nu_{receiver}$ is $A_{null}()$, an output node referencing ν_{output}.
3. The *state* argument of ν is ν_{state} or a computation node with constructor *null*().
4. ν_{state} is not referenced by other nodes than ν.
5. The *sender* argument is a computation node with constructor *null* if and only if the *input* argument is an input node.
6. The *rand* argument of ν is a computation node ν_{rand} with nonce $N \in \mathbf{N}$. On a path trough ν_{rand}, there is no other computation node with nonce N. ν_{rand} is not referenced by other nodes than ν.

References

1. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: POPL 2001, pp. 104–115. ACM (2001)
2. Abadi, M., Baudet, M., Warinschi, B.: Guessing attacks and the computational soundness of static equivalence. In: Aceto, L., Ingólfsdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 398–412. Springer, Heidelberg (2006)

3. Backes, M., Hofheinz, D., Unruh, D.: CoSP: a general framework for computational soundness proofs. In: CCS 2009, pp. 66–78. ACM (2009)
4. Backes, M., Laud, P.: Computationally sound secrecy proofs by mechanized flow analysis. In: CCS, pp. 370–379. ACM (2006)
5. Backes, M., Maffei, M., Mohammadi, E.: Computationally sound abstraction and verification of secure multi-party computations. In: FSTTCS 2010, pp. 352–363. Schloss Dagstuhl (2010)
6. Backes, M., Mohammadi, E., Ruffing, T.: Computational soundness results for ProVerif. In: Abadi, M., Kremer, S. (eds.) POST 2014. LNCS, vol. 8414, pp. 42–62. Springer, Heidelberg (2014)
7. Backes, M., Pfitzmann, B., Waidner, M.: A composable cryptographic library with nested operations (extended abstract). In: CCS 2003, pp. 220–230. ACM (2003)
8. Backes, M., Pfitzmann, B., Waidner, M.: The reactive simulatability (RSIM) framework for asynchronous systems. Inf. Comput. **205**(12), 1685–1720 (2007)
9. Bana, G., Comon-Lundh, H.: A computationally complete symbolic attacker for equivalence properties. In: CCS 2014, pp. 609–620 (2014)
10. Bana, G., Comon-Lundh, H.: Towards unconditional soundness: computationally complete symbolic attacker. In: Degano, P., Guttman, J.D. (eds.) POST 2012. LNCS, vol. 7215, pp. 189–208. Springer, Heidelberg (2012)
11. Baudet, M., Cortier, V., Kremer, S.: Computationally sound implementations of equational theories against passive adversaries. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 652–663. Springer, Heidelberg (2005)
12. Blanchet, B., Abadi, M., Fournet, C.: Automated verification of selected equivalences for security protocols. In: LICS, pp. 331–340 (2005)
13. Blanchet, B., Fournet, C.: Automated verification of selected equivalences for security protocols. In: LICS 2005, pp. 331–340. IEEE (2005)
14. Böhl, F., Cortier, V., Warinschi, B.: Deduction soundness: prove one, get five for free. In: CCS 2013, pp. 1261–1272. ACM (2013)
15. Böhl, F., Unruh, D.: Symbolic universal composability. In: CSF 2013. IEEE (2013)
16. Camenisch, J., Krenn, S., Shoup, V.: A framework for practical universally composable zero-knowledge protocols. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 449–467. Springer, Heidelberg (2011)
17. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. Full and revised version of FOCS 2001 paper. IACR ePrint Archive: 2000/067/20130717:020004 (2013)
18. Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–23. Springer, Heidelberg (2001)
19. Canetti, R., Gajek, S.: Universally Composable Symbolic Analysis of Diffie-Hellman based Key Exchange. IACR ePrint Archive: 2010/303 (2010)
20. Canetti, R., Herzog, J.: Universally composable symbolic security analysis. J. Cryptol. **24**(1), 83–147 (2011)
21. Canetti, R., Krawczyk, H.: Security analysis of IKE's signature-based key-exchange protocol. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 143–161. Springer, Heidelberg (2002)
22. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. J. Cryptol. **19**(2), 68–86 (2003)
23. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: STOC 2002, pp. 494–503. ACM (2002)

24. Chandran, N., Goyal, V., Sahai, A.: New constructions for UC secure computation using tamper-proof hardware. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 545–562. Springer, Heidelberg (2008)
25. Chaum, D.: Blind Signatures for Untraceable Payments. In: CRYPTO 1982, pp. 199–203. Plenum Press (1982)
26. Cheval, V.: APTE: an algorithm for proving trace equivalence. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 587–592. Springer, Heidelberg (2014)
27. Comon-Lundh, H., Cortier, V.: Computational Soundness of Observational Equivalence. In: CCS 2008, pp. 109–118. ACM (2008)
28. Comon-Lundh, H., Cortier, V., Scerri, G.: Security proof with dishonest keys. In: Degano, P., Guttman, J.D. (eds.) POST 2012. LNCS, vol. 7215, pp. 149–168. Springer, Heidelberg (2012)
29. Comon-Lundh, H., Hagiya, M., Kawamoto, Y., Sakurada, H.: Computational soundness of indistinguishability properties without computable parsing. In: Ryan, M.D., Smyth, B., Wang, G. (eds.) ISPEC 2012. LNCS, vol. 7232, pp. 63–79. Springer, Heidelberg (2012)
30. Computational Soundness for Interactive Primitives (full version of this paper). https://www.infsec.cs.uni-saarland.de/~mohammadi/interactive.html
31. Cortier, V., Kremer, S., Küsters, R., Warinschi, B.: Computationally sound symbolic secrecy in the presence of hash functions. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 176–187. Springer, Heidelberg (2006)
32. Cortier, V., Warinschi, B.: A Composable Computational Soundness Notion. In: CCS 2011, pp. 63–74. ACM (2011)
33. Dahl, M., Damgård, I.: Universally composable symbolic analysis for two-party protocols based on homomorphic encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 695–712. Springer, Heidelberg (2014)
34. Delaune, S., Kremer, S., Pereira, O.: Simulation based security in the applied Pi calculus. In: FSTTCS 2009, pp. 169–180. Schloss Dagstuhl (2009)
35. Delaune, S., Kremer, S., Ryan, M.: Verifying privacy-type properties of electronic voting protocols. J. Comput. Secur. **17**(4), 435–487 (2009)
36. Delaune, S., Kremer, S., Ryan, M.D., Steel, G.: Formal analysis of protocols based on TPM state registers. In: CSF, pp. 66–80. IEEE (2011)
37. Diffie, W., Van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Des. Codes Crypt. **2**(2), 107–125 (1992)
38. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Trans. Inf. Theory **29**(2), 198–208 (1983)
39. Dougherty, D.J., Guttman, J.D.: An algebra for symbolic Diffie-Hellman protocol analysis. In: Palamidessi, C., Ryan, M.D. (eds.) TGC 2012. LNCS, vol. 8191, pp. 164–181. Springer, Heidelberg (2013)
40. Even, S., Goldreich, O.: On the security of multi-party ping-pong protocols. In: FOCS 1983, pp. 34–39. IEEE (1983)
41. Fischlin, M.: Round-optimal composable blind signatures in the common reference string model. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 60–77. Springer, Heidelberg (2006)
42. Fournet, C., Kohlweiss, M., Strub, P.-Y.: Modular code-based cryptographic verification. In: CCS 2011, pp. 341–350. ACM (2011)
43. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comp. **18**(1), 186–207 (1989)

44. Hofheinz, D., Shoup, V.: GNUC: a new universal composability framework. J. Cryptol. **28**(3), 423–508 (2013)
45. Kremer, S., Mazaré, L.: Adaptive soundness of static equivalence. In: Biskup, S., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 610–625. Springer, Heidelberg (2007)
46. Küsters, R., Scapin, E., Truderung, T., Graf, J.: Extending and applying a framework for the cryptographic verification of java programs. In: Abadi, M., Kremer, S. (eds.) POST 2014. LNCS, vol. 8414, pp. 220–239. Springer, Heidelberg (2014)
47. Küsters, R., Truderung, T., Graf, J.: A framework for the cryptographic verification of java-like programs. In: CSF 2012, pp. 198–212. IEEE (2012)
48. Küsters, R., Tuengerthal, M.: The IITM Model: a Simple and Expressive Model for Universal Composability. IACR ePrint Archive: 2013/025 (2013)
49. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 696–701. Springer, Heidelberg (2013)
50. Micciancio, D., Warinschi, B.: Soundness of formal encryption in the presence of active adversaries. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 133–151. Springer, Heidelberg (2004)
51. ProVerif code of the case study. https://www.infsec.cs.uni-saarland.de/~mohammadi/paper/case_study_untraceable_payments.zip
52. Sprenger, C., Backes, M., Basin, D., Pfitzmann, B., Waidner, M.: Cryptographically sound theorem proving. In: CSFW 2006, pp. 153–166. IEEE (2006)
53. Swamy, N., Weinberger, J., Schlesinger, C., Chen, J., Livshits, B.: Verifying higher-order programs with the Dijkstra Monad. In: PLDI 2013, pp. 387–398. ACM (2013)
54. Unruh, D.: Termination-insensitive computational indistinguishability (and applications to computational soundness). In: CSF 2011, pp. 251–265. IEEE (2011)