Computational Soundness for Interactive Primitives

Michael Backes, Esfandiar Mohammadi, and Tim Ruffing

CISPA, Saarland University, 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 [16, 30, 38, 39, 41, 43]. Numerous verification tools such as ProVerif [16] and APTE [30] 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, 9, 17, 24, 31, 33, 34, 54, 55]. 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 [46], forward-secure key exchange [40], and blind signatures [29], have gained tremendous attention in the scientific community and widespread deployment in real systems.

One example of their unique properties is that the prover in an interactive zero-knowledge proof can always deny that it has proved something to the verifier,

as used in off-the-record messaging [19]. Another example is forward security in modern key-exchange protocols such as TLS, i.e., after the communication ended, even compromising honest parties does not reveal the shared key.

The security of interactive primitives is often defined and established in the Universal Composability (UC) framework [21] or similar frameworks [10, 48, 52], which enable to prove strong security guarantees in a *composable* manner [27, 28, 44]. 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. Various interactive primitives have been proven to fulfill this strong UC-security [27, 28, 44].

For interactive primitives, ideal functionalities are a suitable abstraction, but for non-interactive primitives, DY-style abstractions have two significant advantages compared to a 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.

Backes, Hofheinz, and Unruh 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 [51] 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 [7].

In a case study, we apply our method to a recent CS result [7]. 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 [29] in ProVerif.

Remark on Supported Equivalence Properties. The aforementioned CS result [7] 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 [31, 32, 33].

2 Overview of the Paper

To facilitate understanding, we first give a high-level and then a technical overview of the proof strategy taken in the paper. *Computational soundness* of a *symbolic Dolev-Yao model* ensures that this symbolic model captures all security relevant aspects of the cryptographic model, which we call the *computational model*. Computational soundness is typically stated for a symbolic Dolev-Yao setting DY, a corresponding set of cryptographic schemes P.[1] Computational soundness of DY for cryptographic schemes P means the following: all attacks of a cryptographic attacker, using the cryptographic schemes P, can also be symbolically mounted in DY.

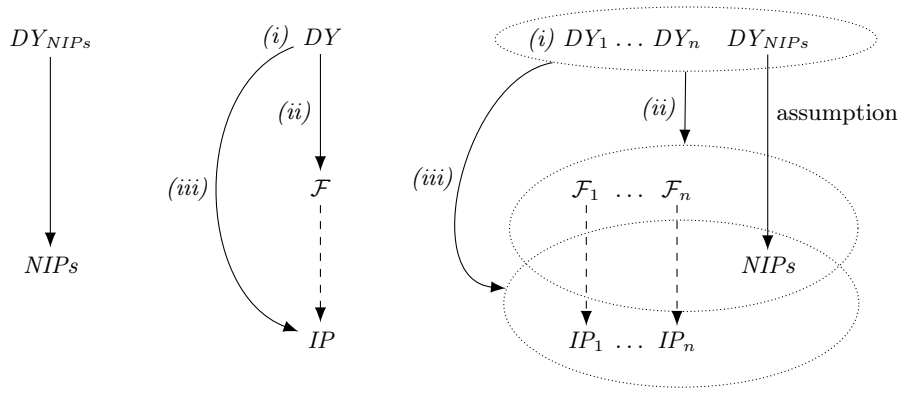
2.1 High-Level Overview

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 (Figure 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 (Figure 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.

[1] Technically, a soundness result also depends on a class of protocols and a set of security properties, which we omitted for the sake of illustration.



(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.

(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 (Figure 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 $DY_1, \ldots, DY_n$ (Section 6).

(ii) We then consider a unified model $(DY_1, \ldots, DY_n, DY_{NIPs})$ that consists of the Dolev-Yao models for the interactive primitives as well as a single Dolev-Yao model DY_{NIPs} that incorporates a set of non-interactive primitives. Under the assumption that DY_{NIPs} is computationally sound with respect to the cryptographic realizations $NIPs$, we show CS for the unified Dolev-Yao model with respect to the algorithms $(\mathcal{F}_1, \ldots, \mathcal{F}_n, NIPs)$, i.e., with respect to the ideal functionalities plus the cryptographic realizations for the non-interactive primitives (Section 7).

(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 $(IP_1, \ldots, IP_n, NIPs)$ (Section 9).

2.2 Technical Overview

This technical overview shall provide an orientation for the reader in the technical sections (Section 6 to Section 9).

Symbolic Model with Interactive Primitives For the symbolic representation, our goal is to abstract away from concrete implementations of interactive primitives by representing them in the symbolic model as ideal functionalities.

As a simple example, consider two parties A and B who run an interactive key exchange. In a symbolic calculus such as the applied π-calculus, this will be modeled as three parallel processes $A \mid B \mid P$, where P is a process that represents the ideal functionality for key exchange. Basically, the process P generates a fresh key and sends it to both parties on private channels. Since this abstracts away from the complex cryptographic details of the key exchange, the protocol $A \mid B \mid P$ is amenable to automated symbolic verification.

Integrating Non-Interactive Primitives Non-interactive primitives are typically abstracted using Dolev-Yao models with constructors (uninterpreted function symbols) and destructors (functions from terms to terms). For example, *dec* defined by $dec(k, enc(k, m)) = m$ is a destructor that destructs the term $enc(k, m)$, which is itself built by the constructor *enc*.

To combine ideal functionalities (for interactive primitives) with Dolev-Yao models (for non-interactive primitives), we formulate the process P in a way that it applies a complex destructor $D_{\mathcal{F}}$ of a Dolev-Yao model, which constitutes a reformulation of the ideal functionality $\mathcal{F}$.

As destructors are inherently non-interactive and stateless, $D_{\mathcal{F}}$ itself (as opposed to $\mathcal{F}$) cannot perform communication or keep state. As a remedy, $D_{\mathcal{F}}$ expects a state as arguments. Additionally, $D_{\mathcal{F}}$ expects the messages sent to the ideal functionality, e.g., $D_{\mathcal{F}}(state, A, \texttt{key exchange with B})$ denotes that party A sends a message `key exchange with B` to $\mathcal{F}$. Accordingly, $D_{\mathcal{F}}$ outputs an updated state and a message that should be forwarded to a protocol party, e.g., $D_{\mathcal{F}}(state, A, \texttt{key exchange with B}) = (state', B, k)$ for a generated key k that should be forwarded to B. The process P acts then merely as a wrapper around $D_{\mathcal{F}}$ that performs the communication and manages the state for $D_{\mathcal{F}}$.

2.3 Ideal Computational Execution

As a first step towards proving computational soundness, we explain how to leverage existing computational soundness results for non-interactive primitives.

The formulation of $\mathcal{F}$ as destructor $D_{\mathcal{F}}$ enables us to consider an *ideal computational execution*, in which $D_{\mathcal{F}}$ is implemented by a computational variant $A_{\mathcal{F}}$ of $\mathcal{F}$. Similarly, all constructors and destructors for non-interactive primitives are implemented by computational algorithms, e.g., the constructor *enc* is replaced by an algorithm A_{enc}, and the destructor *dec* is replaced by an algorithm A_{dec}.

Assume that 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 computationally sound. Then we can show that also the Dolev-Yao model *with* the destructor $D_{\mathcal{F}}$ is computationally sound as well.

Technically, this is done by inlining the code of $D_{\mathcal{F}}$ in the process $P_{\mathbf{D}}$. In this way, all applications of the destructor $D_{\mathcal{F}}$ are eliminated and the computational soundness result for the Dolev-Yao model without $D_{\mathcal{F}}$ can be applied.

2.4 Real Computational Execution

As a final step, we prove computational soundness for the interactive primitives. In the ideal computational execution, only non-interactive primitives have been implemented by their cryptographic realizations. While $A_{\mathcal{F}}$ is used in the computational model, it is merely an algorithmic representation of the ideal functionality $\mathcal{F}$.

To close the gap to the actual cryptographic interactive protocol, we assume that ϕ is a such an interactive protocol that is a UC-secure realization of $\mathcal{F}$. We encode this interactive protocol ϕ into an algorithm A_ϕ. At its core, A_ϕ has the same interface as $A_{\mathcal{F}}$, i.e., A_ϕ relies on P to perform communication and to manage state.

2.5 Computational Soundness for Interactive Primitives

Next, we leverage a composable security notion called *UC-secure realization* [21]. A protocol ϕ UC-securely realizes an ideal functionality $\mathcal{F}$ if there is a simulator that can simulate the behavior of all parties that are not under the control of the attacker in an indistinguishable manner.

We assume that the protocol ϕ is a UC-secure realization of the ideal functionality $\mathcal{F}$, and the UC framework allows for composability, a larger protocol that uses $\mathcal{F}$ internally can securely use ϕ instead of $\mathcal{F}$. To use the same example as above in the applied π-calculus, we can prove that the real computational execution of $A \mid B \mid P$ (using A_ϕ) is secure if the ideal execution of $A \mid B \mid P$ (using $A_{\mathcal{F}}$) is secure. In other words, if $A_{\mathcal{F}}$ is a computationally sound implementation of the interactive primitive $D_{\mathcal{F}}$ (which we have already established), then A_ϕ is a computational sound implementation of the interactive primitive $D_{\mathcal{F}}$. This constitutes our main result.

3 Related Work

There is a successful line of research for computational soundness of trace properties [13, 31, 39, 42] such as authentication and for static equivalence properties (i.e., against passive attackers) [2, 14, 49].

For equivalence properties against active attackers, however, there are only few previous results. The simulatable DY-style library of Backes, Pfitzmann, and Waidner [4, 9] 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, e.g., anonymity properties in protocols that encrypt signatures of different messages.

Cortier and Comon-Lundh [31] 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 [12, 13]. 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 [16] or Tarmarin [53].

With regard to the composability of computational soundness, Böhl, Cortier, and Warinschi [17] show how a computational soundness result that has been obtained via deduction soundness [35] 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.

UC Security and Computational Soundness. There is other work that leverages the strength of the UC framework. Backes, Maffei, and Mohammadi [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 [24], extended by Canetti and Gajek [23], 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 [36] 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. [50] and Küsters, Truderung, and Graf [51] 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, Kohlweiss, and Strub [45] 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 [56] 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, Kremer, and Pereira [37] and Böhl and Unruh [18] transfer simulation-based security completely into the symbolic setting, including symbolic composition theorems. However, these results do not guarantee computational soundness.

4 Review of the CoSP Framework for Equivalence

The abstraction and the computational soundness result put forward in this work are cast in an extension [7] of CoSP [3], a framework for symbolic protocol analyses and conceptually modular computational soundness proofs that hold for several languages, such as the applied π-calculus and RCF [6] (a core calculus of F#). In this section, we review the basic concepts underlying the CoSP framework for equivalence properties [7]. For technical details, we refer the reader to the previous work [3, 7].

4.1 Symbolic Indistinguishability

We define a symbolic notion of indistinguishability for a pair of protocols. First, we define a Dolev-Yao model, called *symbolic model* in CoSP. Then, we present how protocols are represented in CoSP. Thereafter, we present the capabilities of the symbolic attacker, and finally, we define the notion of symbolic indistinguishability.

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 a countably 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 a list of terms to a list of terms. We generally use underlines to denote lists, e.g., we write $\underline{t}$ for a list of terms.

Protocols. Protocols are represented as infinite trees. Each node in this tree represents an action in the protocol:*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 F on a term t, i.e., if $eval_F(t) = \bot$. Nodes have explicit *references* to other nodes whose terms they use. For example, a computation node that computes $C(t)$ references the node that produced t, e.g., an input node or another computation node.

Definition 1 (CoSP Bi-protocol). *A* CoSP bi-protocol Π *is defined like a protocol but uses bi-references instead of references. A* bi-reference *is a pair* $(\nu_{\text{left}}, \nu_{\text{right}})$ *of node identifiers of two (not necessarily distinct) nodes in the protocol tree. In the* left protocol $\text{left}(\Pi)$ *the bi-references are replaced by their left components; the* right protocol $\text{right}(\Pi)$ *is defined analogously.*

Symbolic Operations. As a next step, we model the capabilities of the symbolic attacker. We have to capture which protocol messages the attacker observes, in particular in which order an attacker observe these messages. Moreover, we have to capture which tests an attacker can perform in order to judge whether two protocols are distinguishable. These tests, called *symbolic operations*, capture the sequence of operations a symbolic attacker applies, including the used protocol messages.

Definition 2 (Symbolic Operation). *Let* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ *be a symbolic model. A* symbolic operation O/n (of arity n) on $\mathbf{M}$ *is a finite tree whose nodes are labeled with constructors from* $\mathbf{C}$*, destructors from* $\mathbf{D}$*, nonces from* $\mathbf{N}$*, and formal parameters* x_i *with* $i \in \{1, \ldots, n\}$*. For constructors and destructors, the children of a node represent its arguments (if any). Formal parameters* x_i *and nonces do not have children.*

To unify notation, we introduce $eval_F(\underline{t})$: if F is a constructor, $eval_F(\underline{t}) := F(\underline{t})$ for $F(\underline{t}) \in \mathbf{T}$, and $eval_F(\underline{t}) := \bot$ otherwise. If F is a nonce, $eval_F() := F$. If F is a destructor, $eval_F(\underline{t}) := F(\underline{t})$ if $F(\underline{t}) \neq \bot$ and $eval_F(\underline{t}) := \bot$ otherwise. If F is a symbolic operation, the evaluation function $eval_O : \mathbf{T}^n \to \mathbf{T}$ recursively evaluates the tree O starting at the root as follows: The formal parameter x_i evaluates to t_i. A node with $F \in \mathbf{C} \cup \mathbf{N}_E \cup \mathbf{D}$ evaluates according to $eval_F$, where $\mathbf{N}_E \subseteq \mathbf{N}$ are attacker nonces. If there is a node that evaluates to $\bot$, the whole tree evaluates to $\bot$.

Definition 3 (Symbolic Execution). *Let a symbolic model* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ *and a CoSP protocol* Π *be given. A* symbolic execution *is a path through a protocol tree. It induces a* view*, which contains the communication with the attacker. Together with the symbolic execution, we define an* attacker strategy *as the sequence of symbolic operations that the attacker performs. Formally, a symbolic execution of a protocol* Π *is a (finite) list of triples* (V_i, ν_i, f_i) *as follows.*

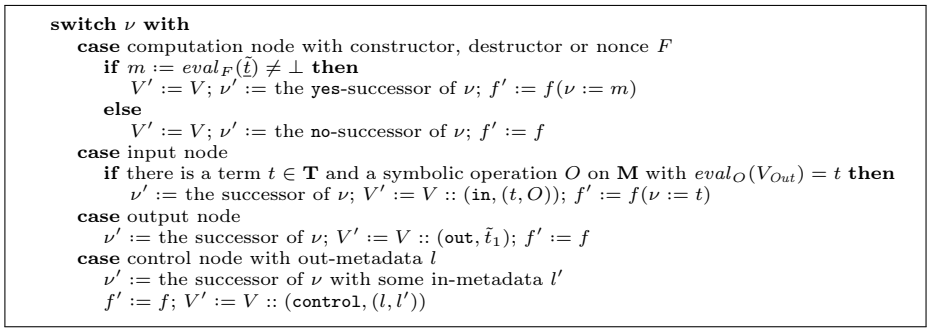


Fig. 2: Symbolic Execution

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 $\underline{\tilde{\nu}}$ be the nodes referenced by ν and define $\underline{\tilde{t}}$ 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.

SViews(Π) *is set of all symbolic views of Π. Given a view V, V_{Out} is the list of terms t contained in $(\mathtt{out}, t) \in V$. $V_{Out\text{-}Meta}$ is the list of terms l contained in $(\mathtt{control}, (l, l')) \in V$. V_{In} (the* attacker strategy*) is the list of terms that contains only entries of V of the form $(\mathtt{in}, (*, O))$ or $(\mathtt{control}, (*, l'))$, and the input term and the out-metadata has been masked with the symbol $*$. $[V_{In}]_{\mathrm{SViews}(\Pi)}$ is the equivalence class of all views $U \in \mathrm{SViews}(\Pi)$ with $U_{In} = V_{In}$.*

Symbolic Knowledge. 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. The definition captures that the attacker knows exactly which symbolic operation leads to which result.

Definition 4 (Symbolic Knowledge). *Let* $\mathbf{M}$ *be a symbolic model. Given a view V with $|V_{Out}| = n$, the* (full) symbolic knowledge *K_V is a function from symbolic operations on* $\mathbf{M}$ *(see Definition 2) of arity n to $\{\top, \bot\}$, defined by $K_V(O) := \bot$ if $eval_O(V_{Out}) = \bot$ and $K_V(O) := \top$ otherwise.*

Equivalent Views. Intuitively, we would like to consider two views *equivalent* if they look the same for a symbolic attacker. Despite the requirement that they have the same order of output, input and control nodes, this is the case if they agree on the out-metadata (the control data sent by the protocol) as well as the symbolic knowledge that can be gained out of the terms sent by the protocol.

Definition 5 (Equivalent Views). *Let two views V, V' of the same length be given. We denote their ith entry by V_i and V'_i, respectively. V and V' are* equivalent *($V \sim V'$), if the following three conditions hold:*

1. *(Same structure) V_i is of the form $(s, \cdot)$ if and only if V'_i is of the form $(s, \cdot)$ for some $s \in \{\mathtt{out}, \mathtt{in}, \mathtt{control}\}$.*

2. *(Same out-metadata)* $V_{\textit{Out-Meta}} = V'_{\textit{Out-Meta}}$.
3. *(Same symbolic knowledge)* $K_V = K_{V'}$.

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.

Definition 6 (Symbolic Indistinguishability). *Let* $\mathbf{M}$ *be a symbolic model and* P *be a class of protocols on* $\mathbf{M}$. *Given an attacker strategy* V_{In} *(in the sense of Definition 3), two protocols* $\Pi_1, \Pi_2 \in \mathrm{P}$ *are* symbolically indistinguishable under V_{In} *if for all views* $V_1 \in [V_{In}]_{\mathrm{SViews}(\Pi_1)}$ *of* Π_1 *under* V_{In}, *there is a view* $V_2 \in [V_{In}]_{\mathrm{SViews}(\Pi_2)}$ *of* Π_2 *under* V_{In} *such that* $V_1 \sim V_2$, *and vice versa.*

Two protocols $\Pi_1, \Pi_2 \in \mathrm{P}$ *are* symbolically indistinguishable *(*$\Pi_1 \approx_s \Pi_2$*), if* Π_1 *and* Π_2 *are indistinguishable under all attacker strategies. For a bi-protocol* Π, *we say that* Π *is symbolically indistinguishable if* $\mathrm{left}(\Pi) \approx_s \mathrm{right}(\Pi)$.

4.2 Computational Indistinguishability

On the computational side, the constructors and destructors in a symbolic model are realized with cryptographic algorithms, which we call computational implementations.

Computational Implementation. 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 $\perp$, 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. We rely on the notion of termination-insensitive computational indistinguishability (tic-indistinguishability) [57] to capture that two protocols are indistinguishable in the computational world. In comparison to standard computational indistinguishability, tic-indistinguishability does not require the the interactive machines to be polynomial-time, but it solely considers decisions that were made after a polynomially-bounded prefix of the interaction (where, both, the attacker's and the protocol's steps are counted). If after an activation, (say) the second protocol does not output anything within a polynomial numbers of steps, then the bi-protocol will be

considered indistinguishable no matter how the first protocol will behave in this case. We write the fact that a machine M terminates after n steps with the output a as $M \Downarrow_n a$.

Definition 7 (Tic-indistinguishability [57]). *Given two machines M, M' and a polynomial p, we write $\Pr[\langle M|M'\rangle \Downarrow_{p(k)} x]$ for the probability that the interaction between M and M' terminates within $p(k)$ steps and M' outputs x.*

We call two machines A and B termination-insensitively computationally indistinguishable for a machine $\mathcal{A}$ *($A \approx^{\mathcal{A}}_{tic} B$) if for all polynomials 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. Additionally, we call A and B termination-insensitively computationally indistinguishable *($A \approx_{tic} B$) if we have $A \approx^{\mathcal{A}}_{tic} B$ for all polynomial-time machines $\mathcal{A}$.*

With the notion of tic-indistinguishability, computational indistinguishability for bi-protocols is naturally defined. A bi-protocol is *computationally indistinguishable* if the corresponding challengers are tic-indistinguishable.

Definition 8 (Computational Indistinguishability). *Let Π be an efficient[2] CoSP bi-protocol and let $\mathbf{A}$ be a computational implementation of $\mathbf{M}$. Π is* (termination-insensitively) computationally indistinguishable *if for all ppt attackers $\mathcal{A}$ and for all polynomials p, $\mathrm{Exec}_{\mathbf{A},\mathbf{M},\mathrm{left}(\Pi)} \approx_{tic} \mathrm{Exec}_{\mathbf{A},\mathbf{M},\mathrm{right}(\Pi)}$.*

Computational Soundness The previous notions culminate in the definition of CS for equivalence properties. It states that the symbolic indistinguishability of a bi-protocol implies its computational indistinguishability.

Definition 9 (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.*

5 Review of the UC Framework

We briefly review the UC framework [21], 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

[2] A (bi-)protocol is *efficient* if the size of every node identifier ν is polynomially bounded in the length of the path to the root, and ν is computable in deterministic polynomial time given all node and edge identifiers on this path.

performs the desired protocol task locally. The ideal functionality $\mathcal{F}$ serves as an abstraction of this taskand as a constructive way to formalize security guarantees.

Security in the UC framework is defined as follows: 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; the written data is called *subroutine output* generated by M'. 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''. If a machine halts, the environment $\mathcal{Z}$ is activated, and chooses which machine to activate next.

In the proof, we use the so-called dummy attacker $\mathcal{A}_d$. $\mathcal{A}_d$ is an attacker that only follows the commands of the environment: $\mathcal{A}_d$ forwards all messages from the environment $\mathcal{Z}$ to a party's network channel and all messages that it receives from a party (i.e., all network messages) to the environment. Canetti shows [21] that it suffices to prove security against the dummy attacker $\mathcal{A}_d$ (and any ppt environment $\mathcal{Z}$).

6 Ideal Functionalities in the Symbolic Model

We abstract away from concrete implementations of interactive primitives by representing them 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 $A \mid B \mid P$, where P is the symbolic key exchange abstraction that generates a fresh key and sends it to both parties on private channels.

6.1 Conditions for the Underlying Model

First, in order to formalize our results, we need some standard symbolic and computational assumptions about the CS result for non-interactive primitives that we would like to extend by interactive primitives.

Symbolic Constraints. The symbolic model must fulfill the following properties, which will be necessary to formulate ideal functionalities in CoSP. First, it should be possible to construct pairs in the symbolic model. Second, we require that there is a distinguished dummy term that can be tested to be equal to other

terms. More formally, we say that a CoSP symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ is *standard* if *(i)* there are constructors $pair/2 \in \mathbf{C}$ and destructors $fst/1, snd/2 \in \mathbf{D}$, and *(ii)*, there is a constructor $null/0 \in \mathbf{C}$ and a destructor $equals/2 \in \mathbf{D}$.

Note that we only require the existence of certain constructors and destructors, but we do not impose explicit semantic restrictions symbolically.

Implementation Constraints. In contrast, semantic conditions are necessary for the algorithms that implement of these constructors and destructors. We call a CoSP computational implementation $\mathbf{A}$ for the standard symbolic model $\mathbf{M}$ *standard* if *(i)* for all $x, y \in \{0,1\}^*$, we have $A_{fst}(A_{pair}(x, y)) = x$ as well as $A_{snd}(A_{pair}(x, y)) = y$, and *(ii)* no algorithm A_C in $\mathbf{A}$ with $C \in \mathbf{C} \setminus \{null\}$ produces $A_{null}()$ on any input. The second condition ensures that $A_{null}()$ is of a unique type and can be achieved by a suitable tagging.

6.2 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.

State and Communication. 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 $\mathcal{F}$.

Only one message from one party can be sent to $\mathcal{F}$ per invocation. This form of communication is closely related to the sequential execution model in UC: Whenever the execution is handed over to a machine M, e.g., an ideal functionality, only one other machine M' may have written a message to a tape of M.

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. Note that there may be (infinite) paths through the protocol tree of $\mathcal{F}$, which do not contain any result nodes, however we will require that a symbolic execution of $\mathcal{F}$ reaches a result after finitely many steps.

Every result node μ_r and its second argument node μ'_r are computational nodes that are both annotated with the *pair* constructor. The term or bitstring constructed by the result node is a triple, encoded using two pairs.

Parameterized CoSP Protocols. For a bi-protocol Π, we formalize the ideal functionalities with the help of *parameterized CoSP protocols*, which have the following properties: Nodes in such protocols are not required to have successors and instead of other nodes, also *formal parameters* can be referenced. A parameterized CoSP protocol is intended to be plugged into another protocol; in that case the formal parameters references must be changed to references to

actual nodes. Given terms $\underline{t}$ that instantiate the parameters of a parameterized protocol Π, the symbolic execution of Π is defined canonically: Whenever a parameter reference to parameter i is resolved, the parameter t_i is used. This allows us to define an ideal functionality in CoSP as parameterized protocol with the parameters *state*, *sid*, *sender*, *input* and *rand* as described above.

Definition 10 (Ideal Functionality). *Suppose the symbolic model* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ *is standard (Section 6.1).A* CoSP ideal functionality *is an efficient probabilistic parameterized CoSP protocol on the symbolic model* $\mathbf{M}$ *such that:*

1. $\mathcal{F}$ references *parameters state, sid, sender, input and rand.*
2. $\mathcal{F}$ *contains no other nodes than computation nodes that are not be annotated by a nonce.*
3. *There is a subset result*$(\mathcal{F})$ *of the nodes in* $\mathcal{F}$*, such that* $\mu \in$ *result*$(\mathcal{F})$ *implies that there is no* $\nu' \in$ *result*$(\mathcal{F})$ *on the path from* μ *to the root. The nodes in result*$(\mathcal{F})$ *are called* result nodes of $\mathcal{F}$*. A result node has no successor.*
4. *Each result node* μ *is annotated with the constructor pair. The second referenced node of* μ *is another computation node* μ' *with the constructor pair.*
5. *The symbolic execution of* $\mathcal{F}$ *reaches a result nodes with all parameters* $t_{state}, t_{sid}, t_{sender}, t_{input}, t_{rand} \in \mathbf{T}$.

$\mathbf{F}$ *is an* ideal model *on* $\mathbf{M}$ *if each* $\mathcal{F} \in \mathbf{F}$ *is a CoSP ideal functionality on* $\mathbf{M}$.

Observe that the second condition ensures that the symbolic execution of $\mathcal{F}$ is deterministic, and thus $D_{\mathcal{F}}$ is deterministic and well-defined.

6.3 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).

A destructor is necessary, because CoSP constructors are just symbols with an arity, whereas destructors are partial functions that map terms to terms.

Definition 11 (Ideal Destructor). *Let* $\mathbf{F}$ *be an ideal model based on the symbolic model* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$*, 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 $\mathbf{D_F} := \mathbf{D} \cup \{D_{\mathcal{F}}/5 \mid \mathcal{F} \in \mathbf{F}\}$.

7 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 12 (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 that result node. The first argument of* $A_{\mathcal{F}}$ *represents the security parameter and the other arguments determine the inputs. If the result node produces* $\perp$ *as output, then the output of* $A_{\mathcal{F}}$ *is also* $\perp$.

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 $\mathbf{M_F}$.

Note that by definition, $\mathcal{F}$ is efficient (Section 4.2) and the algorithms in $\mathbf{A}$ are computable in polynomial time. Though, this does not imply that $A_{\mathcal{F}}$ is polynomial-time. In fact, it is possible that a result node is not reached and the execution may not terminate at all.[3] Thus we must require explicitly each $A_{\mathcal{F}}$ runs in polynomial time.

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 $A_{\mathcal{F}}$.

In the following, we state the computational soundness of the ideal functionalities, which are ideal implementations in the computational model. To leverage existing computational soundness results for non-interactive primitives, which hold for symbolic models without destructors $D_{\mathcal{F}}$, we inline the calls to all destructors $D_{\mathcal{F}}$.

[3] Even if the number of processed nodes is polynomially bounded, the running time can be super-polynomial, if the functions in $\mathbf{A}$ produce too large outputs. For instance, consider the function F which accepts an input of the form 1^x and outputs 1^{2^x}. Clearly, it is computable in deterministic polynomial time. A trace of length $\mathcal{O}(k)$ can implement a k-fold application of F on the input 1, which yields the exponentially long bitstring 1^{2^k}.

Definition 13 (Full Protocol). *Let Π an efficient CoSP (bi-)protocol. The corresponding* full protocol *$\hat{\Pi}$ is obtained from Π by inlining the calls to ideal functionalities: Each computation node ν with destructor $D_{\mathcal{F}}$ is replaced by the tree of the ideal functionality $\mathcal{F}$. The references in $\mathcal{F}$ are changed to references to the corresponding nodes referenced by ν and the subtree rooted at the yes-successor of ν is appended to every result node of $\mathcal{F}$.*

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*. Suppose that $\mathbf{M_F}$ has the ideal implementation $\mathbf{A_F}$. Let the protocol class* P′ *be defined as $\{\Pi \mid \hat{\Pi} \in \mathrm{P}\} =: \mathrm{P}'$, where $\hat{\Pi}$ is the full protocol corresponding to Π.*

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

Proof. Let Π be an efficient CoSP bi-protocol in P′ that symbolically satisfies indistinguishability. By definition, every $\mathcal{F} \in \mathbf{F}$ contains no input and output nodes, i.e., no communication with the attacker is carried out. Thus the views of Π and the full protocol $\hat{\Pi}$ do not differ. This holds for the symbolic views as well as for the computational views. We conclude that the symbolic indistinguishability of Π implies the symbolic indistinguishability of $\hat{\Pi}$.

Since $\mathbf{A}$ is a computationally sound implementation of the symbolic model $\mathbf{M}$ and $\hat{\Pi} \in \mathrm{P}$ is a protocol on $\mathbf{M}$ (in particular it does not use $D_{\mathcal{F}}$) $\hat{\Pi}$ with $\mathbf{A}$ is computationally indistinguishable. As the computational views of $\hat{\Pi}$ and Π are identical, the computational indistinguishability of Π follows.

8 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 $b_{state}, b_{sid}, b_{sender}, b_{input}, b_{rand}$ as input and produces a triple $(b'_{state}, (b_{receiver}, b_{output}))$, encoded as nested pair, 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 parties of a real protocol. We do not model that explicit here. Instead, we assume that A_ϕ enforces the separation of states on its own, e.g., by letting *state* be a list of protocol states of individual parties. If A_ϕ enforces that each party can only access its own state, then the other states can be dummy values in practice.

Since the algorithms can output a state, each UC protocol can be re-formulated 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.

9 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 some machine $\mu(\phi)$ is a UC-secure realization of some machine $\mu(\mathcal{F})$, 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 $D_\mathcal{F}$.

9.1 Protocol Conditions

To ensure that the CoSP computational execution corresponds to an exection in the UC framework, we require certain natural protocol conditions concerning the interactive primitives. The conditions ensure *(i)* that inputs and outputs of the interactive primitives 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 (the *rand* argument). Within a concrete symbolic calculus, syntactic criteria that imply the protocol conditions can be introduced. The wrapper in Section 10.3 will be an example for the applied π-calculus.

Definition 14 (Protocol Conditions for Interactive Primitives). *Given a CoSP protocol Π on an extended symbolic model $\mathbf{M}_\mathbf{F}$, 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'.

Π fulfills the protocol conditions for interactive primitives *if 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 computation nodes with the same destructor $D_\mathcal{F}$* 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 compuation nodes with the same destructor $D_\mathcal{F}$ 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 $(state, receiver, output)$ (encoded using pairs)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.*

9.2 Interfacing CoSP and UC

To simplify notation, we write A_θ to denote an interactive algorithm θ 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.

As we consider only UC protocol machines $\mu(\theta)$ as well as variants thereof, we leave the involved dummy parties in the definition of the ITM and in the remainder of the paper implicit, i.e., we face the protocol machine in an UC execution directly with the environment and annotate each message with an explicit party identifier. Recall that the dummy parties just relay messages from the environment to the UC ideal functionality and vice-versa. We stress that this treatment is only to simplify presentation; our model actually contains dummy parties as in UC.

Definition 15 (UC Machine for an Algorithm). *Let A_θ be an algorithm that uses the standard computational implementation $\mathbf{A}$ for the symbolic model $\mathbf{M}$. Let A_N be the algorithm that implements nonces. The interactive Turing machine (in the UC sense) $\mu(\theta)$ runs the following algorithm:*

- *At the beginning of the first activation, initialize the variable state $:= A_{null}()$.*
- *Whenever $\mu(A_\theta)$ is activated with a message input, let sender be the party identifier of the invoking party, or $A_{null}()$ if the message comes from the*

attacker. Let $rand := A_N(k)$ *and* $res := A_\theta(k, (state, sid, sender, input, rand))$*, where sid is the session ID of* $\mu(\theta)$*.*

- *If* $res = \bot$*, send* (`no`) *to the environment and block all further activations.*
- *Otherwise continue:*
  Let $state' := A_{fst}(res)$*, let* $receiver := A_{snd}(A_{fst}(res))$ *and let* $output := A_{snd}(A_{snd}(res))$*. Set* $state := state'$*.*
  - *If* $A_{equals}(receiver, A_{null}()) = \bot$*, send* (`yes` *receiver*, *output*) *to the environment.*
  - *Else pass output on the network to the attacker.*

9.3 Conditions for the Interactive Primitives

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 interactive nodes. 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 $\mathcal{A}_d$. By definition, $\mathcal{A}_d$ will only relay communication between the environment $\mathcal{Z}$ and the machine $\mu(\phi)$. 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 wrapper machines around $\mu(\theta)$ and $\mu(\phi)$. They report that communication took place between $\mu(\mathcal{F})$ or $\mu(\phi)$ and the attacker, but not what communication.

Definition 16 (Honest and CoSP Compatible Machine). *Given a machine* $\mu(\theta)$ *for an interactive algorithm* A_θ*, the corresponding* honest machine $\tilde{\mu}(\theta)$ *internally runs* $\mu(\theta)$ *and relays the communication with the following exception: If* $\mu(\theta)$ *generates output for the attacker, it is not forwarded, but stored. Instead, a subroutine output* (`output ready`) *is passed to the environment and all messages from the environment or the attacker are blocked*[4] *until the environment*

[4] Blocking can generally be realized by handing over the execution back to the activating party immediately.

sends a subroutine input (`deliver`). *Then the stored message is passed to the attacker.*

Moreover, we define for a given $\mu(\theta)$ *for an interactive algorithm* A_θ*, the corresponding* CoSP compatible machine $\hat{\mu}(\theta)$ *that internally runs* $\mu(\theta)$ *and relays the communication with the following two exceptions.*

1. *If* $\mu(\theta)$ *generates output* m *for the attacker, it is not forwarded, but stored. Instead, a subroutine output* $(\texttt{output ready}, m)$ *is passed to the environment.*
2. *If* $\mu(\theta)$ *receives a message* m *from the attacker, it stores this messages, informs the environment with* $(\texttt{input ready}, m)$*, waits for a* (`deliver`) *messages from the environment (and ignores all other messages), and only then forwards* m *to* $\mu(\theta)$*.*

If the honest machine is used, the environment is informed before giving output to the attacker. Then the environment is forced to let $\tilde{\mu}(\theta)$ deliver the output to the attacker explicitly. This is similar to a computational CoSP execution with A_θ where communication with the attacker can be observed in the views and the sent message is not available to the attacker until a special output node is reached.

To ensure that it is sound to use the wrapper machines instead of the original machines, 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.

Good Ideal Functionalities and Real Protocols. Before we explain the condition in full detail, we first review the situations that make them necessary in more detail.

Consider an UC execution in the real setting, i.e., a setting with a honest machine $\tilde{\mu}(\phi)$. Suppose that the attacker machine is the dummy attacker $\mathcal{A}_d$. The goal is to show that a simulator $\mathcal{S}$ can fake the same execution in the ideal setting with $\tilde{\mu}(\mathcal{F})$. Now assume that during an execution in the real setting, $\tilde{\mu}(\phi)$ produces *true* subroutine output, i.e. not (`output ready`), after it has been activated by $\mathcal{A}_d$. In an ideal setting with the standard machine $\mu(\mathcal{F})$, the simulator $\mathcal{S}$ is able to exchange several messages with $\mu(\mathcal{F})$ to make it produce subroutine output as well. However in an ideal setting with $\tilde{\mu}(\mathcal{F})$, the simulator $\mathcal{S}$ must ensure that the standard machine $\mu(\mathcal{F})$, internally run by the honest machine $\tilde{\mu}(\mathcal{F})$, does not reply back to $\mathcal{S}$. If that happened, the honest machine would tell the environment by sending (`output ready`), which could distinguish the real and the ideal setting trivially. In particular, if the environment runs a computational CoSP execution, an output node would eventually be reached in the ideal setting but not in the real setting. To circumvent this case, the condition in the following Definition 17 guarantees that $\mathcal{S}$ is able to force $A_\mathcal{F}$ and thus $\mu(A_\mathcal{F})$ to produce true subroutine output immediately. It allows the simulator $\mathcal{S}$ to check beforehand whether a particular message m to $\tilde{\mu}(\mathcal{F})$, which relays it to $\mu(\mathcal{F})$, would trigger subroutine output or a reply message back to $\mathcal{S}$. Furthermore, Definition 17 ensures that $\mu(\mathcal{F})$ must not change its internal state if it hands back to $\mathcal{S}$. That is, the simulator $\mathcal{S}$ is able to discard *dummy messages* that would lead to an immediate

reply, because they would be ignored anyway. Hence $\mathcal{S}$ is able to come up with a message that triggers subroutine output certainly.

Definition 17 (Condition for the Ideal Functionality). *Let $\mathcal{F}$ be an ideal functionality using the computational implementation $\mathbf{A}$, and let A_N be the algorithm that implements nonces. Consider an execution of $A_{\mathcal{F}}$ such that $res := A_{\mathcal{F}}(state, sid, sender, input, rand)$ with the following properties:*

- *$state, sid, input \in \{0,1\}^*$*
- *$sender = A_{null}()$, i.e. the execution is initiated by a message from the attacker machine*
- *$rand := A_N()$, i.e. rand is drawn according to A_N*

$\mathcal{F}$ is good for $\mathbf{A}$ *if the following condition holds for each such execution of $A_{\mathcal{F}}$:*

- *If and only if $input = A_{null}()$, we have $res \neq \bot$ and $A_{equals}(receiver, A_{null}())$. In that case, we say that $A_{\mathcal{F}}$ has received a* dummy message *from the attacker machine.*
- *For invocations by dummy messages, we additionally require $state' = state$ and $output = A_{null}()$, where $state' := A_{fst}(res)$ and $receiver := A_{snd}(A_{fst}(res))$. That is, $\mathbf{A}_{\mathcal{F}}$ does not fail but ignores the invocation completely and sends $A_{null}()$ to the attacker machine.*

Another problem is a converse situation: Suppose that during an execution in the real setting, the honest machine $\tilde{\mu}(\phi)$ reports (`output ready`) to the environment, because $\mu(\phi)$ has generated a message for the attacker, whereas $\mu(\mathcal{F})$ in the ideal setting generates true subroutine output s. If $\tilde{\mu}(\phi)$ has been activated by a message from the dummy attacker $\mathcal{A}_d$, then the simulator $\mathcal{S}$ has been instructed by the environment to relay this message to the protocol machine. Thus $\mathcal{S}$ has been activated and is able to send a dummy message to $\tilde{\mu}(\mathcal{F})$, which delays the subroutine output such that in both settings, (`output ready`) is reported to the environment. However, this is not possible in the case that $\tilde{\mu}(\phi)$ has not been invoked by the attacker. Consequently, Definition 18 excludes this case.

Definition 18 (Condition for the Real Protocol). *Let ϕ be a real protocol, and let $\mathbf{A}$ be a standard computational implementation. Moreover, let A_N be the algorithm that implements nonces. Consider an execution of ϕ such that $res := \phi(state_1, sid, sender, input, rand)$ with the following properties:*

- *$state_1, sid, input \in \{0,1\}^*$*
- *$sender \neq A_{null}()$, i.e. the execution is not initiated by a message from the attacker*
- *$rand := A_N()$, i.e. rand is drawn according to A_N*

ϕ is good for $\mathbf{A}$ *if for each such execution of ϕ, it holds that*

- *$res \neq \bot$ and*
- *$destination =$ `network`.*

The following lemma states that we can use the honest machines instead of the original machines given the ideal functionality and the real protocol are good. This captures that the conditions on the ideal functionality and the real protocol are indeed sufficient to overcome our problems.

Lemma 2. *Suppose that the real protocol ϕ is good and the ideal functionality $\mathcal{F}$ is good. Further suppose that $\mu(\phi)$ UC-realizes $\mu(\mathcal{F})$. Then the honest machine $\tilde{\mu}(\phi)$ UC-realizes the honest machine $\tilde{\mu}(\mathcal{F})$.*

Proof. Let $\tilde{\mathcal{Z}}$ an arbitrary polynomial-time UC environment. As $\mu(\phi)$ realizes $\mu(\mathcal{F})$, there is a valid simulator $\mathcal{S}$ for the dummy attacker $\mathcal{A}_d$. The main part of the proof compares the executions of $Exec_{\tilde{\mu}(\mathcal{F}),\mathcal{S},\tilde{\mathcal{Z}}}$, $Exec_{\mu(\mathcal{F}),\mathcal{S},\mathcal{Z}}$, $Exec_{\mu(\phi),\mathcal{A}_d,\mathcal{Z}}$ and $Exec_{\tilde{\mu}(\phi),\mathcal{A}_d,\tilde{\mathcal{Z}}}$, where $\mathcal{Z}$ is an environment which internally runs $\tilde{\mathcal{Z}}$. $\mathcal{Z}$ hides the syntactic differences between the honest machines $\tilde{\mu}$ and the standard machines μ, i.e. the messages (`output ready`) and (`deliver`), by acting as a wrapper for $\tilde{\mathcal{Z}}$. The considered executions are depicted in Figure 3. We prove that $\tilde{\mathcal{Z}}$ cannot distinguish these four executions. In particular, this environment cannot distinguish $Exec_{\tilde{\mu}(\mathcal{F}),\mathcal{S},\tilde{\mathcal{Z}}}$ and $Exec_{\tilde{\mu}(\phi),\mathcal{A}_d,\tilde{\mathcal{Z}}}$. In other words, $\mathcal{S}$ is also valid simulator for the executions with the honest machines, and $\tilde{\mu}(\phi)$ UC realizes $\tilde{\mu}(\mathcal{F})$.

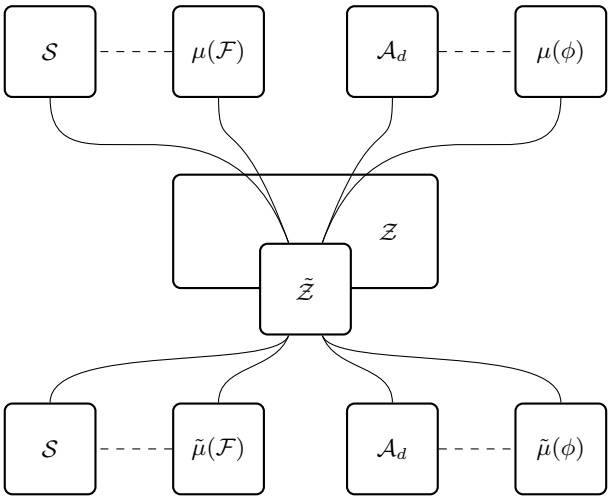


Fig. 3: The considered executions in the proof of Lemma 2

The conditions for the ideal functionality $\mathcal{F}$ (Definition 17) ensure that $\mathcal{F}$, and thus $\mu(\mathcal{F})$, produces subroutine output if and only it receives a dummy message. Dummy messages do not have any effect on the state of the machines in the network, since the activated machine immediately hands back to $\mathcal{S}$ without sending any information. That is, the validity of $\mathcal{S}$ does not depend on the dummy messages it sends to $\tilde{\mu}(\mathcal{F})$. Thus, without loss of generality, we may assume that $\mathcal{S}$ sends a dummy message to $\mu(\mathcal{F})$ if and only if $\mathcal{S}$ is instructed to send a message to the protocol but instead would send a message to the environment $\mathcal{Z}$ directly. This implies that $\mathcal{S}$ activates $\mu(\mathcal{F})$ whenever it is instructed by $\mathcal{Z}$ to relay a message to the protocol, i.e. whenever the dummy attacker $\mathcal{A}_d$ activates $\mu(\phi)$.

We distinguish cases on the possible actions of $\tilde{\mathcal{Z}}$ and the reaction of the machine activated after $\tilde{\mathcal{Z}}$. First, consider the case that $\tilde{\mathcal{Z}}$ instructs the respective attacker machine to deliver a message to the protocol, and that $\tilde{\mu}(\phi)$ as well as $\mu(\phi)$, respectively, generate subroutine output. In the ideal settings, assume for contradiction that $\mathcal{S}$ generates a message for the environment. Then the environment $\mathcal{Z}$ is able to distinguish $Exec_{\mu(\phi),\mathcal{A}_d,\mathcal{Z}}$ and $Exec_{\mu(\mathcal{F}),\mathcal{S},\mathcal{Z}}$; in the former setting $\mathcal{Z}$ has received a message from $\mu(\phi)$ whereas in the latter, the origin of the received the message is $\mathcal{S}$. This contradicts the validity of $\mathcal{S}$, which hence sends a message to $\mu(\mathcal{F})$. By construction, this is no dummy message. Thus Definition 17 guarantees that the internal instance of $\mu(\mathcal{F})$ does not directly reply to $\mathcal{S}$. Such a reply would be observable since $\tilde{\mu}(\mathcal{F})$ would report (`output ready`). Instead, $\mu(\mathcal{F})$ generates subroutine output, which is relayed to $\tilde{\mathcal{Z}}$ by $\tilde{\mu}(\mathcal{F})$. As $\mathcal{S}$ is a valid simulator for $\mu(\mathcal{F})$, this output is indistinguishable from the output given in the real settings.

Second, assume that we are in the case that $\mu(\phi)$, internally run by $\tilde{\mu}(\phi)$, generates a message back to $\mathcal{A}_d$, after this attacker has forwarded m to $\tilde{\mu}(\phi)$. The honest machine $\tilde{\mu}(\mathcal{F})$ informs the environment with a message containing (`output ready`). In this case, a standard message, i.e. not a dummy message, from $\mathcal{S}$ to $\mu(\mathcal{F})$ would lead to subroutine output. Again, $\mathcal{Z}$ could distinguish $Exec_{\mu(\phi),\mathcal{A}_d,\mathcal{Z}}$ and $Exec_{\mu(\mathcal{F}),\mathcal{S},\mathcal{Z}}$. Hence by construction, $\mathcal{S}$ generates a dummy message for $\mu(\mathcal{F})$ and $\tilde{\mu}(\mathcal{F})$ reports (`output ready`) to $\tilde{\mathcal{Z}}$.

Third, it remains to consider the case that $\tilde{\mathcal{Z}}$ sends subroutine input to the protocol machines. By Definition 18, we know that the protocol machine $\mu(\phi)$ in the real setting does not immediately reply to $\tilde{\mathcal{Z}}$; it sends a message to the attacker instead. Hence $\mu(\mathcal{F})$ does the same, otherwise $\mathcal{Z}$ could distinguish $Exec_{\mu(\phi),\mathcal{A}_d,\mathcal{Z}}$ and $Exec_{\mu(\mathcal{F}),\mathcal{S},\mathcal{Z}}$ trivially. The attacker machines are finally activated in all four settings. The rest of this case is analogous to the previous case.

Altogether, $\tilde{\mathcal{Z}}$ is informed about the communication between the protocol and the respective attacker in all four executions and especially cannot distinguish $Exec_{\tilde{\mu}(\mathcal{F}),\mathcal{S},\tilde{\mathcal{Z}}}$ and $Exec_{\tilde{\mu}(\phi),\mathcal{A}_d,\tilde{\mathcal{Z}}}$. That is, $\tilde{\mu}(\phi)$ UC realizes $\tilde{\mu}(\mathcal{F})$.

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 [44], zero-knowledge proofs [20], oblivious transfer [21], and secure function evaluation [21]. In some cases, a technical reformulation of the ideal functionality or the real protocol is necessary. The following paragraphs discuss these reformulations in more detail.

Immediate Outputs. Let ϕ be the real protocol. If ϕ gets inputs from a protocol party P_{in}, it is not able to pass a output immediately to a different party $P_{out} \neq P_{in}$ without having to communicate via the network (the attacker) in between. Thus a so-called *immediate output* is only possible back to P_{in}. That is, Definition 18 basically imposes the restriction that subroutine output cannot be given immediately back to P_{in} when ϕ received subroutine input from ϕ. This

essentially means that ϕ (and also $\mathcal{F}$ if ϕ realizes $\mathcal{F}$) must be formulated such that the results of the protocol are output whenever they are locally determined for a party. That is, the outputs of ϕ need not be requested through an interface; the reply to such a request would be an immediate output.

This is a natural assumption for interactive primitives, where cryptographic operations do not take place only locally as it is the case for encryption or digital signatures for instance. Indeed, the ideal functionalities for public-key encryption and signatures proposed by Canetti use immediate outputs, see [21] for a general discussion of immediate outputs.

Corruption. Our approach can be used with different corruption models. However, to be compatible with the conditions in Definitions 17 and 18, we treat adaptive corruption formally slightly different from the original UC framework.

Our convention is that the corruption messages are sent by the environment directly to the currupted party and not "via" the attacker. (This convenction is also used by Hofheinz and Shoup [47].) The corrupted party (or the ideal functionality on behalf of the currupted party) then informs the attacker about the corruption. This treatment avoids that the simulator in the ideal world has to deliver the corruption message to the ideal functionality, which would in turn activate the simulator, because this message flow is excluded by the condition for the ideal functionality.

The Price for Re-usability of Earlier Results. 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 [7]. 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.

9.4 Main Result

The main theorem 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. To establish the theorem, we need some natural *protocol conditions* (Section 9.1). 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.

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 (Section 9.1), and let the protocol class* P′ *be defined as* $\{\Pi \mid \hat{\Pi} \in \mathrm{P}\} =: \mathrm{P}'$*, where* $\hat{\Pi}$ *is the full protocol corresponding to* Π*. Suppose that every* $\mathcal{F} \in \mathbf{F}$ *is a good ideal functionality and every* $\phi \in \Phi$ *is a good real protocol (see Definitions 17 and 18). 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* $\mathbf{M}_\mathbf{F}$ *for* P′ *with respect to equivalence properties.*

Proof. For simplicity, we consider only one ideal functionality $\mathcal{F}$ and its implementation ϕ. Let $\Pi \in \mathrm{P}'$ be a symbolically indistinguishable bi-protocol using the destructor $D_\mathcal{F}$. Lemma 1 entails that the computational execution of Π with the canonical algorithm $A_\mathcal{F}$ is computationally indistinguishable.

Recall that for an interactive primitive θ (be it $\mathcal{F}$ or ϕ) with a computational implementation A_θ, there is a CoSP compatible UC machine $\hat{\mu}(\theta)$ (see Definition 16). Since Π fulfills the protocol conditions, the CoSP computational execution of Π can be formulated as a UC machine CoSPUC that calls $\hat{\mu}(\mathcal{F})$ (or $\hat{\mu}(\phi)$, respectively) instead of executing $\mathcal{F}$ (or ϕ) on its own.[5] Because the computational execution with $A_\mathcal{F}$ is computationally indistinguishable, we know that for all protocols $\Pi \in \mathrm{P}$ CoSPUC with left(Π) is tic-indistinguishable from CoSPUC with right(Π) when using $\hat{\mu}(\mathcal{F})$. Since $\tilde{\mu}(\mathcal{F})$ leaks less information than $\hat{\mu}(\mathcal{F})$, we can conclude that for all protocols $\Pi \in \mathrm{P}$ CoSPUC with left(Π) is tic-indistinguishable from CoSPUC with right(Π) when using $\tilde{\mu}(\mathcal{F})$.

Suppose that $\mu(\phi)$ UC-realizes $\mu(\mathcal{F})$, and let $\tilde{\mu}(\phi)$ and $\tilde{\mu}(\mathcal{F})$ be its corresponding honest machines, as described in Definition 16. Given that ϕ and $\mathcal{F}$ are good (see Definitions 17 and 18), Lemma 2 shows that $\tilde{\mu}(\phi)$ UC-realizes $\tilde{\mu}(\mathcal{F})$. Since $\tilde{\mu}(\mathcal{F})$ UC-realizes $\tilde{\mu}(\phi)$ and since tic-indistinguishability is transitive in a computationally indistinguishable part [57, Lemma 22], it follows that for all protocols $\Pi \in \mathrm{P}$ CoSPUC with left(Π) is tic-indistinguishable from CoSPUC with right(Π) when using $\tilde{\mu}(\phi)$. By the completeness of the dummy attacker, we can w.l.o.g. assume that the network attacker is the dummy attacker. Consequently, the environment learns the communication from the protocol to the network attacker. Recall that the only difference between $\hat{\mu}$ and $\tilde{\mu}$ is that $\hat{\mu}$ additionally leaks this communication from the protocol to the network attacker. Thus, it follows that for all protocols $\Pi \in \mathrm{P}$ CoSPUC with left(Π) is tic-indistinguishable from CoSPUC with right(Π) when using $\hat{\mu}(\phi)$.

[5] The UC machine CoSPUC is a formulation of the CoSP computational execution in the UC framework. In contrast to the CoSP computational execution, however, CoSPUC does not compute an interactive primitive θ itself but calls $\hat{\mu}(\theta)$ instead. In order to produce the same output as the CoSP computational execution, CoSPUC constructs the CoSP view accordingly. In particular CoSPUC maps the messages (`output ready, m`) and (`input ready,m`) to view entries that correspond to attacker communication.

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 [21, 26]. However, these primitives are possible if a common reference string (CRS) and authenticated message transfer (e.g., using a public-key infrastructure) is assumed [21, 22]. Another example is UC-secure key exchange, which is, depending on the formulation, strictly stronger than standard key exchange [25], and thus requires less efficient protocols. We refer to Canetti [21, 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 Section 3), 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,[6] 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 [21] 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.

10 Case Study: Untraceable Payments

Untraceable payments, proposed by Chaum [29], 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 [16] using a UC-secure abstraction of blind signatures by Fischlin [44]. Our computational soundness theorems entail that the result of ProVerif's verification carries over to the computational realization of untraceable payments.

Although Chaum's protocol is not a state-of-the-art e-cash protocol and much better protocols exist (e.g., [11, 15]), we chose it to illustrate the expressiveness of our result: the symbolic abstraction of (interactive) blind signatures, i.e., the

[6] 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.

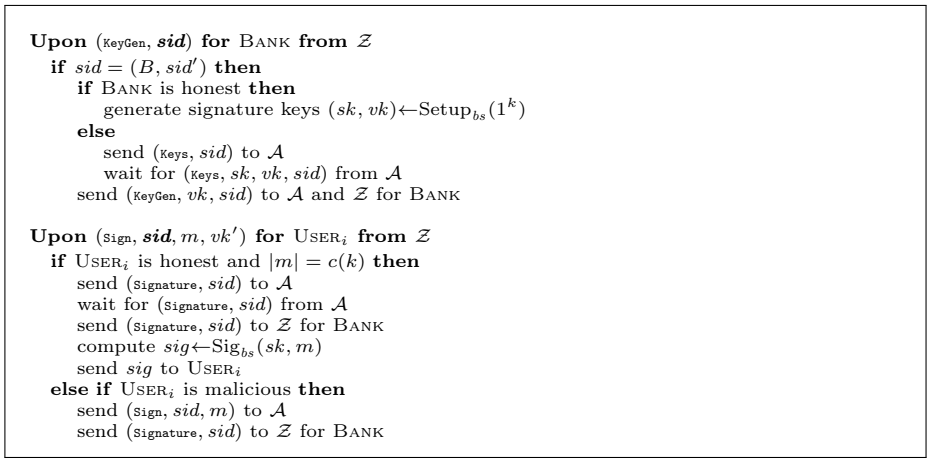


Fig. 4: The family of ideal functionalities $\mathcal{F}_{k,n}$ for blind signatures, where c is a function that specifies the length of a verification key, as detailed in the underlying computational soundness result for non-interactive primitives [7]. In our model, the tokens (i.e., the blindly signed messages) of the users are verification keys, which contain enough entropy to be unpredictable. If desired, they can optionally be used by the users to transfer a signed message along with the token, when the token is spent.

symbolic ideal functionality, internally uses (non-interactive) digital signatures. We show that we can naturally model this combination of interactive and non-interactive primitives.

10.1 Ideal Blind Signatures

Fischlin [44] introduced an ideal functionality and proved that there is a UC-secure realization for it under standard cryptographic assumptions. Our formulation is a slight variation that is better amenable to automated verification.

Our ideal functionality $\mathcal{F}$ (see Figure 4)for blind signatures models a scenario with one bank BANK and n users USER$_i$(for $i = 1$ to n). 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 (recall that ideal functionalities are incorruptible). Then, the functionality distributes the verification keys to the bank BANK and all users USER$_i$ (for $i = 1$ to n).

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 signing request and the message. Then, as in the honest user case, it informs the bank that a signature is being requested.

Fischlin [44] showed the existence of a protocol that UC-realizes an ideal functionality for blind signatures under standard cryptographic assumptions.

The protocol gives the attacker even more freedom than the functionality from Figure 4. Our functionality $\mathcal{F}$ has four differences to the ideal functionality of [44]. First, our functionality uses a fixed signature scheme. Second, our functionality performs a real key generation, locally stores the signing keys, and distributes the verification keys. Third, the ideal functionality does not fake the signatures but uses the signing key to produce real signatures. Fourth, our functionality does not have a verification phase, because we distribute the verification keys and every party can locally verify signatures. Since we only increase the power of the ideal functionality, the realization proof of our ideal functionality is very similar to the realization proof presented in [44]. Using Fischlin's construction, we can prove realization if we require that the signature scheme, used in our ideal functionality (Figure 4), is unforgeable. Details can be found in Appendix A.

10.2 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 (explained below) with respect to a computational implementation $\mathbf{A}_{sig}$ [7]. 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 [7, 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 [7] and fulfills the protocol conditions (Section 9.1). 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 [7], 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 [16] 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 [16].

Definition 19 (Uniform Bi-process). *A bi-process Q in the applied π-calculus is* uniform *if* $\mathrm{left}(Q) \rightarrow R_{\mathrm{left}}$ *implies that* $Q \rightarrow R$ *for some bi-process R with* $\mathrm{left}(R) \equiv R_{\mathrm{left}}$*, and symmetrically for* $\mathrm{right}(Q) \rightarrow R_{\mathrm{right}}$ *with* $\mathrm{right}(R) \equiv R_{\mathrm{right}}$*.*

10.3 Verifying Untraceability in ProVerif

ProVerif [16] is an automated verification tool that can prove the uniformity of bi-processes in the applied π-calculus [1]. We use a wrapper process (Figure 6) in the applied π-calculus that enforces the protocol conditions from Section 9.1.

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.It should not be able to distinguish who of the two payers purchased the coupon.

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.

Moreover, we do not include the length functions, since ProVerif typically does not terminate, once a symbolic model includes length functions. Below, we discuss why our verification is still sound.

Our code [8] 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.

10.4 Soundness of the Verification

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 [7] 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 [7]. 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 3. *Let* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ *be a symbolic model. Consider the model* $\mathbf{M}' = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}')$ *with* $\mathbf{D}' = \mathbf{D} \setminus \{d\}$*. Let* Π *be a bi-protocol on* $\mathbf{M}'$*(i.e.,* Π *does not use the destructor* d*).*

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* $\mathbf{M}'$*.*

Proof. We now prove the contrapositive. Assume Π is distinguishable in $\mathbf{M}$. More precisely, there is a symbolic operation O in $\mathbf{M}$ (that is not in $\mathbf{M}'$) that distinguishes Π. If O is available in $\mathbf{M}'$ as well, then there is nothing to show. Otherwise, if O is available in $\mathbf{M}$ but not in $\mathbf{M}$, then recall that $\mathbf{M}$ and $\mathbf{M}'$ only differ in the availability of the destructor *length*. Thus the distinguishing symbolic operation O contains a call to *length* that yields different results for Π_{left} and Π_{right}.

Because SimLength computes *length* by assumption correctly, the outputs of SimLength$(O_d, , V_{\text{left}}, K_{V_{\text{left}}}^{\mathbf{M}'})$ and SimLength$(O_d, V_{\text{right}}, K_{V_{\text{right}}}^{\mathbf{M}'})$ differ as well. These outputs only depend on O_d, the view and the oracle answers. m and the views are identical up to that point. Thus, the answers to oracle calls, i.e., the evaluations of symbolic operations in $\mathbf{M}'$ yield different results in the left and the right protocol. Thus, there is a symbolic operation $O'_{\mathbf{M}}$ in $\mathbf{M}'$ that distinguishes Π. □

Claim. SimLength defined as in Figure 5 is a function for the *length* destructor in [7] as required by Lemma 3.

Proof. The claim holds by code inspection of SimLength (see Figure 5). □

Plugging everything together, the successful ProVerif verification, Theorem 2, and Lemma 3 prove for our case study bi-process that any realization (adhering to our implementation conditions) is computational indistinguishability.

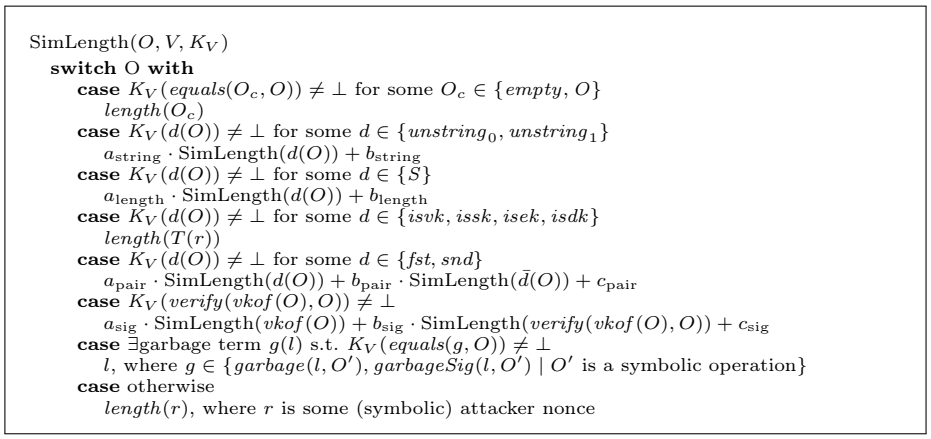


Fig. 5: The definition of SimLength. "·" and "+" denote symbolic multiplication and addition operations and a_f, b_f, c_f denote the coefficients for the length destructor for the message type f. Note that the length destructor is required to be linear [7].

```
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. 6: The wrapper for the ideal functionality

11 Conclusion

We presented the first general computational soundness that encompasses uniformity properties for interactive primitives, such as verifiable computation or blind signatures, for which equivalence properties are crucial. In a case study, we illustrated the applicability of our results by modeling and verifying untraceable payments and verifying in ProVerif.

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 Blind Signatures

We use the UC secure construction of blind signatures by Fischlin [44]. For an extended exposition of the construction, we refer to the work of Fischlin [44].[7]

In the construction, we use single theorem unique zero-knowledge proofs [44]. Such proof systems have two modes. In the mode `key`, a key pair (PK, SK) is generated, and, in mode `prove`, ZK proofs are generated with this PK. Uniqueness states that for each PK there each statement has exactly one valid proof per witness.

We use a ZK relation over (a sequence of) circuits $\mathsf{C}^{BS}_{crs,PK}$ indexed by a CRS crs and PK. Circuit $\mathsf{C}^{BS}_{crs,PK}$ expects as input a statement

$$x = C||ek||crs_{Com}||vk||H(PK)||m$$

of length $\chi(n) = c(n) + 2e(n) + h(n) + s(n) + n$ and a witness $w = u||v||B$ of length $\omega(n) = 2n + s(n)$, and returns an output bit which is determined as follows. In the construction, we will use the circuit of the algorithms $\mathrm{Enc}, \mathrm{Com}, \mathrm{Ver}_{sig}$ for checking that the signature verification succeeds,

$$\mathrm{Ver}_{sig}(vk, \mathrm{Com}(crs_{Com}, m||H(PK)||v; u), B) = 1$$

and that the value C equals the ciphertext, i.e.,

$$\mathrm{Enc}(ek, \mathrm{Com}(crs_{Com}, m||H(PK)||v; u)||B; v).$$

The corresponding relation is defined as follows:

$$R^{BS}_{crs,PK} = \{(x, w) \in \{0,1\}^{\chi(n)} \times \{0,1\}^{\omega(n)} \mid \mathsf{C}^{BS}_{crs,PK}(x, w) = 1\}.$$

We use a second zero-knowledge proof with an NP relation R^{ss} that is defined by a sequence of circuits C^{ss}_n that evaluates to 1 if and only if for statement $x = U||E||ek||crs_{Com}||crs_{uni}$ the following three conditions hold:

[7] For the sake of readability, we omit the session id in our descriptions, since we only consider a single-session functionality.

Upon (KeyGen) **for** BANK **from** $\mathcal{Z}$
send (GetCRS) to $\mathcal{F}_{\text{CRS}}$
wait for crs from $\mathcal{F}_{\text{CRS}}$
send (GetCRSSS) to $\mathcal{F}_{\text{CRS-SS}}$
wait for crs_{ss} from $\mathcal{F}_{\text{CRS-SS}}$
send (GetCRSCom) to $\mathcal{F}_{\text{CRS-COM}}$
wait for crs_{Com} from $\mathcal{F}_{\text{CRS-COM}}$
send (GetEncKey) to $\mathcal{F}_{\text{PKI-ENC}}$
wait for ek from $\mathcal{F}_{\text{PKI-ENC}}$
generate signature keys $(sk, vk) \leftarrow \text{Setup}_{sig}(1^\eta)$
send $(\texttt{Register}, vk)$ to $\mathcal{Z}$

Upon $(\texttt{Sign}, \pi_{ss}, (\texttt{User}, i))$ **for** BANK **from** $\mathcal{A}$
extract the public statement from π_{ss} as $(U||E||ek||crs_{Com}||crs_{uni})$
let $x_{ss} \leftarrow (U||E||ek||crs)$
compute $b \leftarrow \text{Ver}_{zk\text{-}ss}(crs_{ss}, x_{ss}, \pi_{ss})$
if $b = 1$ **then**
compute $B \leftarrow \text{Sig}(sk, U)$
send $(\texttt{Signed}, B)$ to USER_i via $\mathcal{A}$

Fig. 7: The protocol for blind signatures for the bank

1. $E = \text{Enc}(ek, m||PK||u||SK||v; u')$
2. $U = \text{Com}(crs_{Com}, m||H(PK)||v; u)$
3. There exists a randomness string ρ with $(PK, SK) \leftarrow \text{Prove}_{uni}(\texttt{key}, crs_{uni}; \rho)$.

Construction 1 (UC Blind Signature Scheme [44]) *Let* $(\text{Setup}_{sig}, \text{Sig}, \text{Ver}_{sig})$ *be a signature scheme,* $(\text{Setup}_{enc}, \text{Enc}, \text{Dec})$ *be an encryption scheme, and* $(\text{Setup}_{com}, \text{Com})$ *be a commitment scheme. Let* $(\text{Setup}_{zk}, \text{Prove}, \text{Ver}_{zk})$ *be a non-interactive zero-knowledge proof system for* R^{BS} *and let* $(\text{Setup}_{zk\text{-}ss}, \text{Prove}_{ss}, \text{Ver}_{zk\text{-}ss})$ *be a non-interactive zero-knowledge proof system for* R^{ss}.

The UC protocol for blind signature consists of one party that executes the bank protocol, defined in Figure 7, and several parties that execute the user protocol, defined in Figure 8. As setup assumptions, we use public key infrastructures and common reference strings, formalized as the ideal functionalities, i.e., the protocol is formulated in the $\mathcal{F}_{\text{PKI-SIG}}$, $\mathcal{F}_{\text{PKI-ENC}}$, $\mathcal{F}_{\text{CRS}}$, $\mathcal{F}_{\text{CRS-SS}}$, $\mathcal{F}_{\text{CRS-COM}}$ *hybrid model.*

Let $vk_{BS} = (crs, vk)$ *and* $sk_{BS} = sk$. *Verification is non-interactive. Thus, we give the construction of the verification algorithm directly:*
$\text{Ver}_{sig\text{-}bs}(vk_{BS}, m, S)$: *Parse* $S = C||\pi$ *and extract the statement* x *from* π *as* $x = C||ek||crs_{Com}||vk||m$, *check whether* $vk = vk'$ *from* $vk_{BS} = (crs, vk)$. *Then, output* $\leftarrow \text{Ver}_{sig}(crs, x, \pi)$ *for* $x = C||ek||crs_{Com}||vk||m$.

Like the construction of Fischlin [44], we need the property that the used encryption scheme, the commitment scheme and the signature scheme do not leak information through the length of their outputs. This ensures that a malicious bank cannot just identify a particular blind signature by its length. We briefly

Upon first invokation

send $\texttt{requestCRS}$ to $\mathcal{F}_{\text{CRS}}$
store crs
send $\texttt{requestCRS}$ to $\mathcal{F}_{\text{CRS-SS}}$
wait for a response crs_{ss} store it
send $\texttt{requestCRSCom}$ to $\mathcal{F}_{\text{CRS-COM}}$
wait for a response crs_{Com} store it
generate $(dk, ek) \leftarrow \text{Setup}_{enc}(1^\eta)$
send $(\texttt{Register}, ek)$ to $\mathcal{F}_{\text{PKI-ENC}}$
generate $(sk, vk) \leftarrow \text{Setup}_{sig}(1^\eta)$
store vk

Upon $(\texttt{Sign}, m, vk')$ **for** USER_i **from** $\mathcal{Z}$

choose $(u, v, u', \rho) \leftarrow \{0,1\}^n$
let $(PK, SK) \leftarrow \text{Prove}_{uni}(\texttt{key}, crs_{uni}; \rho)$
compute $U \leftarrow \text{Com}(crs, m||H(PK)||v; u)$
compute $E \leftarrow \text{Enc}(ek, m||PK||v||SK||u; u')$
let $x_{ss} \leftarrow U||E||ek||crs_{Com}||crs_{uni}$
let $w_{ss} \leftarrow m||v||u||u'||\rho$
compute $\pi_{ss} \leftarrow \text{Prove}(crs, x_{ss}, w_{ss})$
send $(\texttt{Sign}, \pi_{ss})$ to BANK

Upon $(\texttt{Signed}, B, (\texttt{User}, i))$ **for** USER_i **from** $\mathcal{A}$

compute $b \leftarrow \text{Ver}_{sig}(vk, U, B)$
if $b = 1$ **then**
  compute $C \leftarrow \text{Enc}(ek, U||B; v)$
  set $x \leftarrow C||ek||crs_{Com}||vk||H(PK)||m$
  set $w \leftarrow u||v||B$
  let $\pi \leftarrow \text{Prove}_{uni}(\texttt{prove}, crs, x, w, SK)$
  output $S \leftarrow C||PK||\pi$ to $\mathcal{Z}$

Fig. 8: The protocol for blind signatures for USER_i

restate the exact notions used in [44]: A signature scheme is *length-invariant* if all its verification keys vk and signatures have the same length $s(\eta)$ where η is the security parameter. A commitment scheme that commits on strings of length η using η bits of randomness is length-invariant if all commitments are of length $c(\eta)$. Finally, we say that an encryption scheme is length-invariant if all its encryption keys ek as well as ciphertexts (for plaintexts of length $c(\eta) + s(\eta)$) have the length $e(\eta)$.

Moreover, we assume that the commitment scheme has *unique openings*, .i.e., there do not exists $(z, r) \neq (z', r')$ with $\text{Com}(crs_{Com}, z; r) = \text{Com}(crs_{Com}, z'; r')$ with overwhelming probability.

Construction 2 *Let* $(\text{Setup}_{bs}, \text{Sig}'_{bs}, \text{Ver}_{sig\text{-}bs})$ *be the following a signature scheme, where* $\text{Ver}_{sig\text{-}bs}$ *is defined as in Construction 1.*

$\mathrm{Setup}_{bs}(1^\eta)$: *Let* $crs \leftarrow \mathrm{Setup}_{zk\text{-}uni}(1^\eta)$, $crs_{ss} \leftarrow \mathrm{Setup}_{zk\text{-}ss}(1^\eta)$, $crs_{Com} \leftarrow \mathrm{Setup}_{com}(1^\eta)$, $(dk, ek) \leftarrow \mathrm{Setup}_{enc}(1^\eta)$, *and* $(sk, vk) \leftarrow \mathrm{Setup}_{sig}(1^\eta)$. *Let* $sk_{BS} \leftarrow (crs, crs_{ss}, crs_{Com}, dk, sk)$ *and* $vk_{BS} \leftarrow (crs, crs_{ss}, crs_{Com}, ek, vk)$. *Output* (sk_{BS}, vk_{BS}).

$\mathrm{Sig}'_{bs}(sk_{BS}, m)$: *Run the user and the signer protocol from Construction 1 and output the output of the user.*

From Theorem 2 in the work of Fischlin [44] it immediately follows that the Construction 2 is strongly existentially unforgeable.[8]

Theorem 3 ([44]). *Suppose that the signature scheme* $(\mathrm{Setup}_{sig}, \mathrm{Sig}, \mathrm{Ver}_{sig})$ *is length-invariant, unforgeable against adaptive chosen-message attacks, and that* Sig *is deterministic. Further suppose that the encryption scheme* $(\mathrm{Setup}_{enc}, \mathrm{Enc}, \mathrm{Dec})$ *is length-invariant and IND-CPA secure. Additionally, let* $(\mathrm{Setup}_{com}, \mathrm{Com})$ *be a length-invariant non-interactive commitment scheme in the common reference string model, which is statistically binding and computationally hiding and has unique openings. Also let* H *be a collision-intractable hash function family and* $(\mathrm{Setup}_{zk\text{-}uni}, \mathrm{Prove}_{uni}, \mathrm{Ver}_{zk\text{-}uni})$ *be a single-theorem unique non-interactive zero-knowledge proof system for* R^{BS} *with deterministic verifier* $\mathrm{Ver}_{zk\text{-}uni}$. *Let* $(\mathrm{Setup}_{zk\text{-}ss}, \mathrm{Prove}_{ss}, \mathrm{Ver}_{zk\text{-}ss})$ *be a (regular) non-interactive zero-knowledge proof system for* R^{ss}.

Construction 2 is an strongly existentially unforgeable signature scheme.

A.1 UC Realization Proof for Blind Signatures

Theorem 4. *Suppose that the signature scheme* $(\mathrm{Setup}_{sig}, \mathrm{Sig}, \mathrm{Ver}_{sig})$ *is length-invariant, unforgeable against adaptive chosen-message attacks, and that* Sig *is deterministic. Further suppose that the encryption scheme* $(\mathrm{Setup}_{enc}, \mathrm{Enc}, \mathrm{Dec})$ *is length-invariant and IND-CPA secure. Additionally, let* $(\mathrm{Setup}_{com}, \mathrm{Com})$ *be a length-invariant non-interactive commitment scheme in the common reference string model, which is statistically binding and computationally hiding and has unique openings. Also let* H *be a collision-intractable hash function family and* $(\mathrm{Setup}_{zk\text{-}uni}, \mathrm{Prove}_{uni}, \mathrm{Ver}_{zk\text{-}uni})$ *be a single-theorem unique non-interactive zero-knowledge proof system for* R^{BS} *with deterministic verifier* $\mathrm{Ver}_{zk\text{-}uni}$. *Let* $(\mathrm{Setup}_{zk\text{-}ss}, \mathrm{Prove}_{ss}, \mathrm{Ver}_{zk\text{-}ss})$ *be a (regular) non-interactive zero-knowledge proof system for* R^{ss}.

Then Construction 1 realizes the ideal functionality in Figure 4 in the $\mathcal{F}_{\text{PKI-SIG}}$, $\mathcal{F}_{\text{PKI-ENC}}$, $\mathcal{F}_{\text{CRS}}$, $\mathcal{F}_{\text{CRS-SS}}$, $\mathcal{F}_{\text{CRS-COM}}$ *hybrid model.*

[8] Fischlin showed the result for a variant of the construction in which $\mathcal{Z}$ and π_{ss} are not send in the first round. However, these two messages are not used in the final output; hence his result immediately carries over to Construction 2. Moreover, Fischlin formulated unforgeability for blind signatures, i.e., for interactive protocols. An attacker against the strong existential unforgeability game can be used to break the unforgeability of the underlying blind signature scheme by simulating the signing oracle by computing the protocol of User_i.

Proof. The proof follows along the lines of the work of Fischlin [44]. We refer the interested reader to [44, Theorem 4] for an extended exposition. We only present a proof outline and elaborate on the parts of the proof that differ from [44, Theorem 4].

We have to show that for each attacker $\mathcal{A}$ attacking the real-world protocol from Construction 1 there exists a simulator $\mathcal{S}$ against the ideal-functionality such that for all environments $\mathcal{Z}$.

We describe a sequence of games indistinguishable that is indistinguishable for all ppt environments $\mathcal{Z}$. Let game 0 be the ideal setting in which the ideal functionality communicates with the simulator $\mathcal{S}$.

In game 1, the simulator $\mathcal{S}_1$ computes the setup functionalities $\mathcal{F}_{\text{PKI-SIG}}$, $\mathcal{F}_{\text{PKI-ENC}}$, $\mathcal{F}_{\text{CRS}}$, $\mathcal{F}_{\text{CRS-SS}}$, and $\mathcal{F}_{\text{CRS-COM}}$. As $\mathcal{S}_1$ honestly computes the setup functionalities, game 0 and game 1 are perfectly indistinguishable.

In game 2, all ZK proofs of honest parties are simulated (for both ZK proof schemes). By the zero-knowledge property game 2 is indistinguishable from game 1.

In game 3, all encryption operations are replaced by encryptions of the constant 0 string. By the IND-CPA security of the encryption scheme, game 3 is indistinguishable from game 2.

In game 4, all commitment operations are replaced by commitments of the constant 0 string. By the hiding property of the commitment scheme, game 4 is indistinguishable from game 3.

In game 5, the simulator $\mathcal{S}_5$ instead of the protocol computes all cryptographic operations because the real messages are not needed anymore. Moreover, in game 5 the ideal functionality replaced the (by now modified) protocol. Game 5 is perfectly indistinguishable from game 4 because the simulator receives all information that it needs to simulate game 4.

Hence, there is a simulator such that game 5 and game 0 are computationally indistinguishable.

References

1. Abadi, M., Fournet, C.: Mobile Values, New Names, and Secure Communication. In: POPL'01, pp. 104–115. ACM (2001)
2. Abadi, M., Baudet, M., Warinschi, B.: Guessing Attacks and the Computational Soundness of Static Equivalence. In: FOSSACS'06, pp. 398–412. Springer (2006)
3. Backes, M., Hofheinz, D., Unruh, D.: CoSP: A General Framework for Computational Soundness Proofs. In: CCS'09, 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'10, pp. 352–363. Schloss Dagstuhl (2010)
6. Backes, M., Maffei, M., Unruh, D.: Computationally Sound Verification of Source Code. In: CCS, pp. 387–398. ACM (2010)
7. Backes, M., Mohammadi, E., Ruffing, T.: Computational Soundness Results for ProVerif. In: POST'14, pp. 42–62. Springer (2014)

8. Backes, M., Mohammadi, E., Ruffing, T.: ProVerif code of the case study. URL: https://www.infsec.cs.uni-saarland.de/~mohammadi/paper/case_study_untraceable_payments.zip
9. Backes, M., Pfitzmann, B., Waidner, M.: A Composable Cryptographic Library with Nested Operations (Extended Abstract). In: CCS'03, pp. 220–230. ACM (2003)
10. Backes, M., Pfitzmann, B., Waidner, M.: The Reactive Simulatability (RSIM) Framework for Asynchronous Systems. Inf. Comput. 205(12), 1685–1720 (2007)
11. Baldimtsi, F., Chase, M., Fuchsbauer, G., Kohlweiss, M.: Anonymous Transferable E-cash. In: PKC'15, pp. 101–124. Springer (2015)
12. Bana, G., Comon-Lundh, H.: A Computationally Complete Symbolic Attacker for Equivalence Properties. In: CCS'14, pp. 609–620 (2014)
13. Bana, G., Comon-Lundh, H.: Towards Unconditional Soundness: Computationally Complete Symbolic Attacker. In: POST'12, pp. 189–208. Springer (2012)
14. Baudet, M., Cortier, V., Kremer, S.: Computationally Sound Implementations of Equational Theories Against Passive Adversaries. In: ICALP'05, pp. 652–663. Springer (2005)
15. Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized Anonymous Payments from Bitcoin. In: S&P'14, pp. 459–474. IEEE (2014)
16. Blanchet, B., Abadi, M., Fournet, C.: Automated Verification of Selected Equivalences for Security Protocols. In: LICS'05, pp. 331–340 (2005)
17. Böhl, F., Cortier, V., Warinschi, B.: Deduction Soundness: Prove One, Get Five for Free. In: CCS'13, pp. 1261–1272. ACM (2013)
18. Böhl, F., Unruh, D.: Symbolic Universal Composability. In: CSF'13, pp. 257–271. IEEE (2013)
19. Borisov, N., Goldberg, I., Brewer, E.: Off-the-record Communication, or, Why Not to Use PGP. In: pp. 77–84. ACM (2004)
20. Camenisch, J., Krenn, S., Shoup, V.: A Framework for Practical Universally Composable Zero-Knowledge Protocols. In: ASIACRYPT'11, pp. 449–467. Springer (2011)
21. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. Full and revised version of FOCS'01 paper. IACR Cryptology ePrint Archive: 2000/067/20130717:020004 (2013)
22. Canetti, R., Fischlin, M.: Universally Composable Commitments. In: CRYPTO'01, pp. 19–40. Springer (2001)
23. Canetti, R., Gajek, S.: Universally Composable Symbolic Analysis of Diffie-Hellman based Key Exchange. IACR Cryptology ePrint Archive: 2010/303 (2010)
24. Canetti, R., Herzog, J.: Universally Composable Symbolic Security Analysis. J. of Crypt. 24(1), 83–147 (2011)
25. Canetti, R., Krawczyk, H.: Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In: CRYPTO'02, pp. 143–161. Springer (2002)
26. Canetti, R., Kushilevitz, E., Lindell, Y.: On the Limitations of Universally Composable Two-Party Computation without Set-up Assumptions. J. of Crypt. 19(2), 68–86 (2003)
27. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally Composable Two-party and Multi-party Secure Computation. In: STOC'02, pp. 494–503. ACM (2002)
28. Chandran, N., Goyal, V., Sahai, A.: New Constructions for UC Secure Computation Using Tamper-Proof Hardware. In: EUROCRYPT'08, pp. 545–562. Springer (2008)
29. Chaum, D.: Blind Signatures for Untraceable Payments. In: CRYPTO'82, pp. 199–203. Plenum Press (1982)

30. Cheval, V.: APTE: An Algorithm for Proving Trace Equivalence. In: TACAS'14, pp. 587–592. Springer (2014)
31. Comon-Lundh, H., Cortier, V.: Computational Soundness of Observational Equivalence. In: CCS'08, pp. 109–118. ACM (2008)
32. Comon-Lundh, H., Cortier, V., Scerri, G.: Security Proof with Dishonest Keys. In: POST, pp. 149–168. Springer (2012)
33. Comon-Lundh, H., Hagiya, M., Kawamoto, Y., Sakurada, H.: Computational Soundness of Indistinguishability Properties Without Computable Parsing. In: ISPEC'12, pp. 63–79. Springer (2012)
34. Cortier, V., Kremer, S., Küsters, R., Warinschi, B.: Computationally Sound Symbolic Secrecy in the Presence of Hash Functions. In: FSTTCS'06, pp. 176–187. Springer (2006)
35. Cortier, V., Warinschi, B.: A Composable Computational Soundness Notion. In: CCS'11, pp. 63–74. ACM (2011)
36. Dahl, M., Damgård, I.: Universally Composable Symbolic Analysis for Two-Party Protocols Based on Homomorphic Encryption. In: EUROCRYPT'14, pp. 695–712. Springer (2014)
37. Delaune, S., Kremer, S., Pereira, O.: Simulation Based Security in the Applied Pi Calculus. In: FSTTCS'09, pp. 169–180. Schloss Dagstuhl (2009)
38. Delaune, S., Kremer, S., Ryan, M.: Verifying Privacy-Type Properties of Electronic Voting Protocols. J. Comput. Secur. 17(4), 435–487 (2009)
39. 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)
40. Diffie, W., Van Oorschot, P.C., Wiener, M.J.: Authentication and Authenticated Key Exchanges. Designs, Codes and Cryptography 2(2), 107–125 (1992)
41. Dolev, D., Yao, A.C.: On the Security of Public Key Protocols. IEEE Transactions on Information Theory 29(2), 198–208 (1983)
42. Dougherty, D.J., Guttman, J.D.: Symbolic Protocol Analysis for Diffie-Hellman. In: TGC'13, Springer (2013)
43. Even, S., Goldreich, O.: On the Security of Multi-Party Ping-Pong Protocols. In: FOCS'83, pp. 34–39. IEEE (1983)
44. Fischlin, M.: Round-Optimal Composable Blind Signatures in the Common Reference String Model. In: CRYPTO'06, Springer (2006)
45. Fournet, C., Kohlweiss, M., Strub, P.-Y.: Modular Code-based Cryptographic Verification. In: CCS'11, pp. 341–350. ACM (2011)
46. Goldwasser, S., Micali, S., Rackoff, C.: The Knowledge Complexity of Interactive Proof Systems. SIAM J. Comp. 18(1), 186–207 (1989)
47. Hofheinz, D., Shoup, V.: GNUC: A New Universal Composability Framework. IACR Cryptology ePrint Archive: `2011/303` (2011)
48. Hofheinz, D., Shoup, V.: GNUC: A New Universal Composability Framework. J. of Crypt. 28(3), 423–508 (2013)
49. Kremer, S., Mazaré, L.: Adaptive Soundness of Static Equivalence. In: ESORICS'07, pp. 610–625. Springer (2007)
50. Küsters, R., Scapin, E., Truderung, T., Graf, J.: Extending and Applying a Framework for the Cryptographic Verification of Java Programs. In: POST'14, pp. 220–239. Springer (2014)
51. Küsters, R., Truderung, T., Graf, J.: A Framework for the Cryptographic Verification of Java-like Programs. In: CSF'12, pp. 198–212. IEEE (2012)
52. Küsters, R., Tuengerthal, M.: The IITM Model: a Simple and Expressive Model for Universal Composability. IACR Cryptology ePrint Archive: `2013/025` (2013)

53. Meier, S., Schmidt, B., Cremers, C., Basin, D.A.: The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In: CAV'13, pp. 696–701. Springer (2013)
54. Micciancio, D., Warinschi, B.: Soundness of Formal Encryption in the Presence of Active Adversaries. In: TCC'04, pp. 133–151. Springer (2004)
55. Sprenger, C., Backes, M., Basin, D., Pfitzmann, B., Waidner, M.: Cryptographically Sound Theorem Proving. In: CSFW'06, pp. 153–166. IEEE (2006)
56. Swamy, N., Weinberger, J., Schlesinger, C., Chen, J., Livshits, B.: Verifying Higher-order Programs with the Dijkstra Monad. In: PLDI'13, pp. 387–398. ACM (2013)
57. Unruh, D.: Termination-Insensitive Computational Indistinguishability (and Applications to Computational Soundness). In: CSF'11, pp. 251–265. IEEE (2011)