CoSP: A General Framework For Computational Soundness Proofs

Michael Backes  
Saarland University and MPI-SWS  
backes@mpi-sws.mpg.de

Dennis Hofheinz*  
CWI, Amsterdam  
Dennis.Hofheinz@cwi.nl

Dominique Unruh†  
Saarland University  
Saarbrücken, Germany  
unruh@mccm.uni-saarland.de

ABSTRACT
We describe CoSP, a general framework for conducting computational soundness proofs of symbolic models and for embedding these proofs into formal calculi. CoSP considers arbitrary equational theories and computational implementations, and it abstracts away many details that are not crucial for proving computational soundness, such as message scheduling, corruption models, and even the internal structure of a protocol. CoSP enables soundness results, in the sense of preservation of trace properties, to be proven in a conceptually modular and generic way: proving \( x \) cryptographic primitives sound for \( y \) calculus only requires \( x + y \) proofs (instead of \( x \cdot y \) proofs without this framework), and the process of embedding \( x \) is conceptually decoupled from computational soundness proofs of cryptographic primitives. We exemplify the usefulness of CoSP by proving the first computational soundness result for the full-fledged applied \( \pi \)-calculus under active attacks. Concretely, we embed the applied \( \pi \)-calculus into CoSP and give a sound implementation of public-key encryption and digital signatures.

Categories and Subject Descriptors
C.2.2 [Computer-Communication Networks]: Network Protocols—Protocol Verification

General Terms
Security, theory, verification

1. INTRODUCTION
Proofs of security protocols are known to be error-prone and, owing to the distributed-system aspects of multiple interleaved protocol runs, awkward for humans to generate. Hence, work towards the automation of such proofs started soon after the first protocols were developed. From the start, the actual cryptographic operations in such proofs were idealized into so-called Dolev-Yao models, following see, e.g., [45, 57, 3, 51, 56, 19]. This idealization simplifies proofs by freeing them from cryptographic details such as computational restrictions, probabilistic behavior, and error probabilities. While it was initially not clear whether Dolev-Yao models are a sound abstraction from real cryptography with its computational security definitions, a large number of results in the last ten years helped to establish a general understanding of which cryptographic primitives can or cannot be proven computationally sound in which adversarial settings under which assumptions (see Section 1.2).

A careful inspection of this series of results, however, reveals that the soundness theorems stated in these works, and even more so the frameworks that underly these theorems, differ from each other considerably in many respects. These differences range from various ways of syntactically expressing security protocols and corresponding restrictions on the set of permitted protocol classes, to different semantics for modelling protocol communication and communication with the adversary, to different (often incomparable) notions of computational soundness, to different assumptions on the adversary’s capabilities, etc. Moreover, many of these frameworks were freshly invented in the respective papers; they hence lack support from suitable verification tools and are more likely to suffer from idiosyncrasies than more established frameworks for reasoning about security protocols.

The lack of a common framework that underlies results about computational soundness complicates the thorough comparison of their strengths and limitations. Even worse, it often remains unclear whether assumptions in computational soundness results (additional randomization in the cryptographic implementation, the absence of key cycles,\(^\dagger\)) all establish (among others results) the computational soundness of symbolic encryption (either symmetric or asymmetric): First, [12] expresses protocols as probabilistic input-output automata, exploits the communication model offered by the Reactive Simulatability (RSIM) framework [10], and shows computational soundness in the sense of reactive simulatability for encryption schemes. Second, [46] expresses protocols and their communication using a newly introduced concept called abstract algebras, and shows computational soundness of encryption schemes in the sense of preservation of static equivalence in the presence of an adaptive, but passive adversary. Third, [43] expresses protocols and their communication within a small fragment of the applied \( \pi \)-calculus, and shows computational soundness for encryption schemes in the sense of preservation of observational equivalence in the presence of an active adversary.

*Supported by an NWO Veni grant
†Supported by the Cluster of Excellence “Multimodal Computing and Interaction”.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

We show how to use CoSP to establish the first computational soundness results for the prevalent cryptographic definitions. Moreover, framework-specific assumptions complicate the extension of existing results to other frameworks, or to more comprehensive settings, e.g., a more expressive set of cryptographic primitives or a stronger adversary. In fact, such results are often proven from scratch again (for an extended new framework). To put it bluntly: formally asserting computational soundness of $x$ different cryptographic primitives within $y$ different frameworks currently requires $x \cdot y$ separate proofs.

1.1 Our contribution

A general framework for computational soundness proofs. We describe CoSP, a general framework for conducting Computational Soundness Proofs in symbolic models that enables formulating soundness results in a unified and comparable manner and for embedding these proofs into formal calculi. CoSP comprises a general definition of symbolic protocols, their symbolic and computational execution, as well as a definition of computational soundness for trace properties.

CoSP does not put constraints on the symbolic model; in particular, it permits arbitrary sets of constructors, deduction rules, equational theories and computational implementations, and it is specifically designed for establishing soundness results in that it abstracts away from many details that are not crucial for proving computational soundness, such as message scheduling, corruption models, and even the internal structure of a protocol. Instead, we treat the whole protocol as a single entity that interacts with an attacker. This allows for a unified treatment of different symbolic models by embedding them into CoSP.

CoSP enables proving computational soundness results in a conceptually modular and generic way: every computational soundness proof phrase in CoSP automatically holds for all embedded calculi, and the process of embedding formal calculi is conceptually decoupled from computational soundness proofs. This is crucial since these two tasks are often tackled using different techniques and pursued by people with different backgrounds in computer science. Asserting computational soundness of $x$ cryptographic primitives within $y$ calculi hence requires only $x + y$ proofs in CoSP.

We stress that we do not develop soundness results for novel cryptographic primitives or less restricted protocol classes in this paper; nor do we unify all existing soundness results. However, CoSP provides a basis for doing so. To lay the foundation, we show computational soundness for public-key encryption and digital signatures in CoSP in this paper. This shows the computational soundness of these primitives in all calculi embedded in CoSP.

Computational soundness of the applied $\pi$-calculus. We show how to use CoSP to establish the first computational soundness result for a full-fledged applied $\pi$-calculus with encryption and signatures under active attacks. We consider the process calculus proposed in [25] additionally augmented with events; the calculus in [25] itself is a combination of the original applied $\pi$-calculus [2] with one of its dialects [23]. This combination offers the richness of the original applied $\pi$-calculus while additionally being accessible to state-of-the-art verification tools such as ProVerif [22]; in particular, our result can be extended to arbitrary equational theories. Our result hence yields computational soundness guarantees for ProVerif.

We first give a mapping of processes in the applied $\pi$-calculus to CoSP protocols. This embedding is particularly instructive because the semantics of the applied $\pi$-calculus differ significantly from CoSP’s semantics. We then show that a process in the applied $\pi$-calculus is computationally sound whenever the corresponding CoSP protocol is computationally sound.

Together with the computational soundness of encryptions and digital signatures in CoSP, this implies the computational soundness of the applied $\pi$-calculus with encryptions and signatures. In particular, the result shows that CoSP is capable of embedding a formal calculus that is well understood and accepted by the scientific community, that is expressive enough for expressing and reasoning about state-of-the-art protocols, and that is accessible to state-of-the-art verification tools.

As an example (essentially a litmus test for our framework), we use ProVerif to analyze the entity authentication property of the Needham-Schroeder-Lowe protocol. Using the aforementioned computational soundness of public-key encryption in CoSP, this yields an implementation of this protocol within the applied $\pi$-calculus such that the implementation is provably secure under active attacks.

1.2 Related work

Cryptographic underpinnings of a Dolev-Yao model were first addressed by Abadi and Rogaway in [5] for passive adversaries and symmetric encryption. The protocol language and security properties handled were extended in [1, 11, 41], but still apply only to passive adversaries. This excludes many of the typical ways of attacking protocols, e.g., man-in-the-middle attacks and attacks by reusing a message part in a concurrent protocol run.

A cryptographic justification of a Dolev-Yao model for arbitrary active attacks and within arbitrary surrounding interaction protocols within the Reactive Simulatability (RSIM) Framework [16] was first given by Backes, Pfitzmann, and Waidner in [12] with extensions in [13, 10]. Tool support for this Dolev-Yao model was subsequently added in [58]. Laud [48] has subsequently presented a cryptographic underpinning for a Dolev-Yao model of symmetric encryption under active attacks. His work enjoys a direct connection with a formal proof tool, but it is specific to certain confidentiality properties and restricts the surrounding protocols to straight-line programs. Micciancio and Warinschi [53] and Janvier, Lakhnech, and Mazaré [43] have presented cryptographic underpinnings for a Dolev-Yao model of public-key encryption. Their results are narrower than those in [12] since they are specific for public-key encryption and restricted classes of protocols, but they consider simpler real implementations. Baudet, Cortier, and Kremer [20] have established the soundness of specific classes of equational theories in a Dolev-Yao model under passive attacks. Canetti and Herzog [24] have shown that a Dolev-Yao-style symbolic analysis can be conducted using the framework of universal composability [28] for a restricted class of protocols.

Subsequent work concentrated on linking symbolic and cryptographic secrecy properties. Cortier and Warinschi [36] have shown that symbolically secret nonces are also computationally secure.
tationally secret, i.e., indistinguishable from a fresh random value given the view of the adversary. Backes and Pfitzmann [11] and Canetti and Herzog [29] have established new symbolic criteria for showing that a key is cryptographically secret. Laud [19] has designed a type system for proving payload secrecy of security protocols based on the BPW model [12]. His work is extended to key secrecy in [20]. Kremer and Mazaráre [40] have established computational soundness results for static equivalence. Adão and Fournet [6] have shown computational soundness in the sense of observational equivalence of cryptographic implementations of processes. Their work does not consider explicit symbolic abstractions of cryptographic primitives, e.g., encryption keys cannot be sent, and hence does not allow for describing most existing security protocols without ruling out attacks. Cortier and Comon-Lundh [31] have established computational soundness results as the preservation of observational equivalence within a fragment of the applied π-calculus. This fragment is restricted to protocols that do not branch, and it contains some non-standard extensions that are not supported by existing tools like ProVerif. Comon-Lundh [31] has presented general definitions for trace properties and observational equivalence that are parametric in the underlying equational theory and the computational implementation. His definition is more restrictive than ours in that it requires the existence of a so-called trace mapping. Moreover, in contrast to our result, it does not address how to embed existing calculi like the applied π-calculus into his model.

Further work aimed at establishing computational soundness results for additional cryptographic primitives. Cortier, Kremer, Küsters, and Warinschi [33] and Backes, Pfitzmann, and Waidner [15] have shown computational soundness of hash functions in the random oracle model. Janvier, Lakhnech, and Mazaráre [44] have shown computational soundness of hash functions under a non-standard cryptographic assumption in the standard model, i.e., without random oracles. Garcia and van Rossum [30] have shown computational soundness of hash functions under passive adversaries when implemented using perfect one-way hash functions [30]. Backes and Unruh [17] have shown computational soundness of non-interactive zero-knowledge proofs. Bresson, Lakhnech, Mazaráre, and Warinschi [27] have provided a computationally sound theory for reasoning about protocols based on the decisional Diffie-Hellman assumption (DDH) for passive adversaries. Limitations of computational soundness in the sense of Reactive Simulatability were shown by Backes and Pfitzmann for hash functions [15] and the XOR operation [14].

Recently, efforts have also been started to formulate syntactic calculi with a probabilistic, polynomial-time semantics to directly reason about cryptographic primitives/protocol, including approaches based on process algebra [23, 24, 50], security logics [24, 37], and cryptographic games [24, 50, 59, 15, 17]. In general, this line of work is orthogonal to the work of Justifying Dolev-Yao models, which offer a higher level of abstractions and thus much simpler proofs where applicable, so that proofs of larger systems can be automated.

1.3 Outline of the paper

Section 2 introduces our framework for computational soundness proofs. Section 2.3 introduces the notion of a simulator, and it identifies which properties a simulator needs to have to entail a computational soundness result. Section 3 contains a case study: how to establish the computational soundness of public-key encryption within the general framework by constructing a suitable simulator. Section 4 establishes the computational soundness of the applied π-calculus. Section 5 concludes and outlines future work.

2. CoSP: A FRAMEWORK FOR COMPUTATIONAL SOUNDNESS PROOFS

2.1 Preliminaries

We first introduce basic notations that are used in this paper, as well as central concepts such as constructors, destructors, and deduction relations.

Notation. Given a term t and a substitution φ, we denote by tφ the result of applying φ to t. Given a function f, f(x := y) is the function f‘ with f’(x) = y and f’(z) = f(z) for z ≠ x. We abbreviate x₁,...,xₙ with z if n is clear from the context. Given a set M and a function f, we define f⁻¹(M) := {x : f(x) ∈ M}. We call a set M efficiently decidable if there is a deterministic polynomial-time algorithm deciding membership in M. We call M prefix-closed if x ∈ M implies x‘ ∈ M for all prefixes x‘ of x. A non-negative function f is negligible if for every c and sufficiently large n, f(n) < n⁻ᶜ. f is overwhelming if 1 − f is negligible.

Definition 1 (Constructors, destructors, nonces, and messages types). A constructor C is a symbol with a (possibly zero) arity. A nonce N is a symbol with zero arity. We write C/n ∈ C to denote that C contains a constructor C with arity n. A message type T over C and N is a set of terms over constructors C and nonces N.

A destructor D of arity n, written D/n, over a message type T is a partial map Tⁿ → T. If D is undefined on T, we write D(T) = ⊥.

In the following, we only consider sets of constructors C such that the same constructors cannot have different arities, i.e., C/n, C/m ∈ C implies n = m. (This restriction simplifies notation and is without loss of generality, as one can simulate multi-arity constructors by adding the arity to the name of the constructor.)

To unify notation, for a constructor, destructor, or nonce F/n, we define the partial function evalF : Tⁿ → T as follows: If F is a constructor or nonce, evalF(t₁,...,tₙ) := F(t₁,...,tₙ) if F(t₁,...,tₙ) ∈ T and evalF(t₁,...,tₙ) := ⊥ otherwise. If F is a destructor, evalF(t₁,...,tₙ) := F(t₁,...,tₙ) if F(t₁,...,tₙ) ∈ T and evalF(t₁,...,tₙ) := ⊥ otherwise.

We now define which terms can be deduced from other terms; this is formalized using a deduction relation ⊨ over a set of terms T. The intuition of S ⊨ m for S ⊆ T and m ∈ T is that the term m can be deduced from the terms in S.

Definition 2 (Deduction relation). A deduction relation ⊨ over a message type T is a relation between S ⊆ T and T.

In most cases, the adversary can apply all constructors and destructors. This can be modelled by defining S ⊢ L := S ⊨ C(t) for every constructor C and S ⊢ [L] → D(t) for every destructor D, respectively. However, our model does not assume this in general, i.e., it supports private constructors as used by, e.g., ProVerif.

The constructors, destructors, and nonces, together with the message type and the deduction relation together form a symbolic model. Such a symbolic model describes a particular Dolev-Yao-style theory.
Definition 3 (Symbolic model). A symbolic model \( M = (C, N, T, D, \rightarrow) \) consists of a set of constructors \( C \), a set of nonces \( N \), a message type \( T \) over \( C \) and \( N \) with \( N \subseteq T \), a set of destructors \( D \) over \( T \), and a deduction relation \( \rightarrow \) over \( T \).

Predicates and how to model arbitrary (non-free) equational theories. A predicate \( P \) of arity \( n \) over a set of constructors \( C \) is a subset of \( T^n \). Predicates can be used to describe arbitrary tests that a protocol may perform. In particular, they can describe the equality test \( \text{equals}(x, y) \) which is the diagonal on \( T^n \) for free equational theories and the equivalence relation between terms in non-free equational theories (i.e., \( \text{equals}(E(D(m)), m) \)). For instance, a non-free theory in which \( T \) holds can be modeled by constructors \( E \) and \( D \), and by letting \( \text{equals}(E(D(m)), m) \). (In this case, of course, all destructor nodes should be compatible with \( \text{equals} \).) Furthermore, since each predicate \( P \) (including \( \text{equals} \)) can be realized using a destructor \( D_P \) by defining \( D_P(t_1, \ldots, t_n) := t_1 \) if \( P(t_1, \ldots, t_n) = \text{true} \) and \( D_P(t_1, \ldots, t_n) := \perp \) otherwise, predicates do not require an explicit treatment. For an example on how to model non-free equational theories, see Section 4

2.2 Symbolic protocols

We define a symbolic protocol in CoSP, called a CoSP protocol henceforth, as a tree with a distinguished root and with labels on both nodes and edges. Intuitively, the nodes correspond to different protocol actions: Computation nodes produce terms (using a constructor, destructor, or nonce); input and output nodes correspond to receive and send operations; nondeterministic nodes encode nondeterministic choices in the protocol. Control nodes allow an outside entity (i.e., an adversary) to influence the control flow of the protocol. The edge labels intuitively allow for distinguishing branches in the protocol execution, e.g., destructor nodes have two outgoing edges labelled with yes and no, corresponding to the two cases that the destructor is defined on the input term or not; hence we can, e.g., speak about the yes-successor of a destructor node.

Definition 4 (CoSP protocol). A CoSP protocol \( \Pi \) is a tree with a distinguished root and labels on both edges and nodes. Each node has a unique identifier \( N \) and one of the following types:

- Computation nodes are annotated with a constructor, nonce, or destructor \( F/n, \) together with the identifiers of \( n \) (not necessarily distinct) nodes. Computation nodes have exactly two successors; the corresponding edges are labeled with yes and no, respectively.
- Output nodes are annotated with the identifier of one node. An output node has exactly one successor.
- Input nodes have no further annotation. An input node has exactly one successor.
- Control nodes are annotated with a bitstring \( l \). A control node has at least one and up to countably many successors annotated with distinct bitstrings \( l' \in \{0,1\}^* \). (We call \( l \) to-out-metadata, and \( l' \) in-metadata.)

Our main application of CoSP in this paper—specifically, the embedding of the applied \( \pi \)-calculus—does not make use of nondeterministic nodes. However, we feel that nondeterministic nodes can be very useful for future applications.

- Non-deterministic nodes have no further annotation. Non-deterministic nodes have at least one and at most finitely many successors; the corresponding edges are labeled with distinct bitstrings.

We assume that the annotations are part of the node identifier \( N \). If a node \( N \) contains an identifier \( N' \) in its annotation, then \( N' \) has to be on the path from the root to \( N \) (including the root, excluding \( N \)), and \( N' \) must be a computation or input node. In case \( N' \) is a computation node, the path from \( N' \) to \( N \) has to additionally go through the outgoing edge of \( N' \) with label yes.

Assigning each nondeterministic node a probability distribution over its successors yields the notion of a probabilistic CoSP protocol.

Definition 5 (Probabilistic CoSP protocol). A probabilistic CoSP protocol \( \Pi_p \) is a CoSP protocol, where each nondeterministic node is additionally annotated with a probability distribution on the labels of the outgoing edges.

Any probabilistic CoSP protocol \( \Pi_p \) can be transformed canonically into a CoSP protocol \( \Pi_b \) by erasing the probability distributions. We will call \( \Pi_b \) the CoSP protocol that corresponds to \( \Pi_p \).

Probabilistic CoSP protocols will be crucial in the definition of computational soundness: we cannot implement nondeterministic protocols, but we can implement probabilistic protocols. Hence, for computational soundness, instead of directly talking about a CoSP protocol \( \Pi_b \), we will talk about all probabilistic CoSP protocols whose corresponding CoSP protocol is \( \Pi_b \).

Definition 6 (Efficient protocol). We call a probabilistic CoSP protocol efficient if:

- There is a polynomial \( p \) such that for any node \( N \), the length of the identifier of \( N \) is bounded by \( p(m) \) where \( m \) is the length (including the total length of the edge-labels) of the path from the root to \( N \).
- There is a deterministic polynomial-time algorithm that, given the identifiers of all nodes and the edge labels on the path to a node \( N \), computes the label of \( N \).

The symbolic execution of a CoSP protocol for a given symbolic model consists of a sequence of triples \((S, \nu, f)\) where \( S \) represents the knowledge of the adversary, \( \nu \) represents the current node identifier in the protocol, and \( f \) represents a partial function mapping already processed node identifiers to messages.

Definition 7 (Symbolic execution). Let a symbolic model \( (C, N, T, D, \rightarrow) \) and a CoSP protocol \( \Pi_b \) be given. A full trace is a (finite) list of tuples \((S_i, \nu_i, f_i)\) such that the following conditions hold:

- Correct start: \( S_1 = \emptyset, \nu_1 \) is the root of \( \Pi_b, f_1 \) is a totally undefined partial function mapping node identifiers to terms.
- Valid transition: For every two consecutive tuples \((S, \nu, f)\) and \((S', \nu', f')\) in the list, let \( \nu' \) be the node identifiers in the annotation of \( \nu \) and define \( l_{ij} \) through \( t_i := f(\nu_i) \). We have:
  - If \( \nu \) is a computation node with constructor, destructor, or nonce \( F/n \), then \( S' = S, f = f \).
  - If \( m := \text{eval}(l) \neq \perp, \nu' \) is the yes-successor of \( \nu \) in \( \Pi_b \), and \( f' = f(\nu := m) \).
  - If \( m = \perp, \nu' \) is the no-successor of \( \nu \) and \( f' = f \).
outgoing edges, the left edge is the yes-edge, and the right edge is the no-edge. Solid lines represent the edges in the protocol tree, and dashed lines refer to the nodes that are given to computation and outputs nodes as arguments. In a node with two outgoing edges, the left edge is the yes-edge, and the right one the no-edge. E.g., the E-computation node is executed if the NR-computation node succeeds and produces a term E(a, b, c) where a, b, c are the terms constructed by the input, the pair-computation, and the NR-computation node. We assume that, if a constructor application fails, the protocol is stuck, this is modeled by a subtree ∞ which consists of an infinite sequence of nondeterministic nodes. The overall effect of this tree is to compute E(ekB, pair(NA, ekA), NR) with ekA := ek(NR) when receiving ekB from the adversary.

A full-fledged modeling of the Needham-Schroeder protocol would also contain a description of the party B. In this case, the adversary might use control nodes to indicate which party to schedule at which point, each possible schedule would correspond to a path in the tree. One should keep in mind, however, that such trees are not intended to be produced by hand, instead, they are an intermediate representation needed to encode other calculi.

2.3 Computational execution

Here and in the following, we will assume a canonical bitstring representation of symbols, terms, and nodes. We do not require that the bitstring representation of a term, say, E(m) hides (the bitstring representation of) m.

A computational implementation of a symbolic model is a family of functions that provide computational interpretations to constructors, destructors, and nonces.

Definition 8 (Computational implementation). Let a symbolic model M = (C, N, T, D, ⊢) be given. A computational implementation of M is a family of functions A = (Aa)α∈C∪D∪N such that A_f for F/n ∈ C ∪ D is a partial deterministic function N × ((0, 1)^n → (0, 1)^n, and A_v for N ∈ N is a total probabilistic function with domain N and range {0, 1}^n (i.e., it specifies a probability distribution on bitstrings that depends on its argument). The first argument of A_F and A_v represents the security parameter.

All functions A_F and A_v have to be computable in (probabilistic) polynomial-time.

Requiring A_C and A_D to be deterministic is without loss of generality, since one can always add an explicit randomness argument that takes a nonce as input.

The computational execution of a probabilistic CoSP protocol defines an overall probability distribution on all possible node traces that the protocol proceeds through. In contrast to symbolic executions, we do not aim at defining the notion of a full trace: the adversary’s symbolic knowledge S has no formal counterpart in the computational setting, and the function f occurring in the computational executions will not be needed in our later results.

Figure 1 shows how to model this protocol step as a CoSP protocol. Solid lines represent the edges in the protocol tree, and dashed lines refer to the nodes that are given to computation and outputs nodes as arguments. In a node with two outgoing edges, the left edge is the yes-edge, and the right one the no-edge. E.g., the E-computation node is executed if the NR-computation node succeeds and produces a term E(a, b, c) where a, b, c are the terms constructed by the input, the pair-computation, and the NR-computation node. We assume that, if a constructor application fails, the protocol is stuck, this is modeled by a subtree ∞ which consists of an infinite sequence of nondeterministic nodes. The overall effect of this tree is to compute E(ekB, pair(NA, ekA), NR) with ekA := ek(NR) when receiving ekB from the adversary.

A full-fledged modeling of the Needham-Schroeder protocol would also contain a description of the party B. In this case, the adversary might use control nodes to indicate which party to schedule at which point, each possible schedule would correspond to a path in the tree. One should keep in mind, however, that such trees are not intended to be produced by hand, instead, they are an intermediate representation needed to encode other calculi.

Example. For the sake of exposition, we illustrate how to phrase a cryptographic protocol as a CoSP protocol, i.e., according to Definition 3. Consider the first step of the Needham-Schroeder public-key protocol:

\[ A \rightarrow B : E_{ekB}(NA, ekA). \]

In this step, A sends the encryption E_{ekB}(NA, ekA) of a fresh nonce NA and A’s encryption key ekA under B’s encryption key ekB to B. We assume that ekA has been generated by A himself, and ekB has been provided by the adversary. Figure 1 shows how to model this protocol step as a CoSP protocol. Solid lines represent the edges in the protocol tree, and dashed lines refer to the nodes that are given to computation and outputs nodes as arguments. In a node with two outgoing edges, the left edge is the yes-edge, and the right one the no-edge. E.g., the E-computation node is executed if the NR-computation node succeeds and produces a term E(a, b, c) where a, b, c are the terms constructed by the input, the pair-computation, and the NR-computation node. We assume that, if a constructor application fails, the protocol is stuck, this is modeled by a subtree ∞ which consists of an infinite sequence of nondeterministic nodes. The overall effect of this tree is to compute E(ekB, pair(NA, ekA), NR) with ekA := ek(NR) when receiving ekB from the adversary.

A full-fledged modeling of the Needham-Schroeder protocol would also contain a description of the party B. In this case, the adversary might use control nodes to indicate which party to schedule at which point, each possible schedule would correspond to a path in the tree. One should keep in mind, however, that such trees are not intended to be produced by hand, instead, they are an intermediate representation needed to encode other calculi.

2.3 Computational execution

Here and in the following, we will assume a canonical bitstring representation of symbols, terms, and nodes. We do not require that the bitstring representation of a term, say, E(m) hides (the bitstring representation of) m.

A computational implementation of a symbolic model is a family of functions that provide computational interpretations to constructors, destructors, and nonces.

Definition 8 (Computational implementation). Let a symbolic model M = (C, N, T, D, ⊢) be given. A computational implementation of M is a family of functions A = (A_α)α∈C∪D∪N such that A_F for F/n ∈ C ∪ D is a partial deterministic function N × ((0, 1)^n → (0, 1)^n, and A_v for N ∈ N is a total probabilistic function with domain N and range {0, 1}^n (i.e., it specifies a probability distribution on bitstrings that depends on its argument). The first argument of A_F and A_v represents the security parameter.

All functions A_F and A_v have to be computable in (probabilistic) polynomial-time.

Requiring A_C and A_D to be deterministic is without loss of generality, since one can always add an explicit randomness argument that takes a nonce as input.

The computational execution of a probabilistic CoSP protocol defines an overall probability distribution on all possible node traces that the protocol proceeds through. In contrast to symbolic executions, we do not aim at defining the notion of a full trace: the adversary’s symbolic knowledge S has no formal counterpart in the computational setting, and the function f occurring in the computational executions will not be needed in our later results.

Definition 8 (Computational execution). Let a symbolic model M = (C, N, T, D, ⊢) be given. A computational implementation A of M and a probabilistic CoSP protocol Π be given. Let a probabilistic polynomial-time interactive machine E (the adversary) be given (polynomial-time in the sense that the number of steps in all activations are bounded in its argument k), and let p be a polynomial. We define a probability distribution Nodes^{k}_{M,A,E(k)}(ν) on (finite) lists of node identifiers (ν) according to the following probabilistic algorithm (both the algorithm and E are run on input k):

- Initial state: ν := ν is the root of Π. Let f be an initially empty partial function from node identifiers to bitstrings, and let n be an initially empty partial function from N to bitstrings.
- For i = 2, ..., do the following:
- Let \( \tilde{v} \) be the node identifiers in the label of \( \nu \). Define \( \tilde{m} \) through \( \tilde{m}_i := f(\tilde{v}_i) \).
- Proceed depending on the type of node \( \nu \):
  * If \( \nu \) is a computation node with nonce \( N \in \mathbb{N} \): Let \( m' := n(N) \) if \( n(N) \neq \bot \) and sample \( m' \) according to \( L_N(k) \) otherwise. Let \( \nu' \) be the yes-successor of \( \nu \), \( f' := f(\nu := m') \), and \( n' := n(N := m') \). Let \( \nu := \nu', f := f' \) and \( n := n' \).
  * If \( \nu \) is a computation node with constructor or destructor \( F \), then \( m' := AF(k, \tilde{m}_i) \). If \( m' \neq \bot \), then \( \nu' \) is the yes-successor of \( \nu \), if \( m' = \bot \), then \( \nu' \) is the no-successor of \( \nu \). Let \( f' := f(\nu := m') \). Let \( \nu := \nu', f := f' \).
  * If \( \nu \) is an input node, ask for a bitstring \( m \) from \( E \). Abort the loop if \( E \) halts. Let \( \nu' \) be the successor of \( \nu \). Let \( f := f(\nu := m) \) and \( \nu := \nu' \).
  * If \( \nu \) is an output node, send \( \tilde{m}_i \) to \( E \). Abort the loop if \( E \) halts. Let \( \nu' \) be the successor of \( \nu \). Let \( \nu := \nu' \).
  * If \( \nu \) is a control node, labeled with out-metadata \( l \), send \( l \) to \( E \). Abort the loop if \( E \) halts. Upon receiving an answer \( l' \), let \( \nu' \) be the successor of \( \nu \) along the edge labeled \( l' \) (or the lexicographically smallest edge if there is no edge with label \( l' \)). Let \( \nu := \nu' \).
  * If \( \nu \) is a nondeterministic node, let \( D \) be the probability distribution on the label of \( \nu \). Pick \( \nu' \) according to the distribution \( D \), and let \( \nu := \nu' \).
- Let \( n_1 := \nu \).
- Let \( \text{len} \) be the number of nodes from the root to \( \nu \) plus the total length of all bitstrings in the range of \( f \). If \( \text{len} > p(k) \), stop.

### 2.4 Computational Soundness

We first define trace properties and their fulfillment by a (probabilistic) CoSP protocol. After that, we provide the definition of computational soundness for trace properties.

**Definition 10 (Trace property).** A trace property \( \mathcal{P} \) is an efficiently decidable and prefix-closed set of (finite) lists of node identifiers.

Let \( M = (C, N, T, D, +) \) be a symbolic model and \( \Pi_s \) a CoSP protocol. Then \( \Pi_s \) symbolically satisfies a trace property \( \mathcal{P} \) in \( M \) iff every node trace of \( \Pi_s \) is contained in \( \mathcal{P} \). Let \( A \) be a computational implementation of \( \Pi_s \) and let \( \Pi_c \) be a probabilistic CoSP protocol. Then \( (\Pi_c, A) \) computationally satisfies a trace property \( \mathcal{P} \) in \( M \) iff for all probabilistic polynomial-time interactive machines \( E \) and all polynomials \( p \), the probability is overwhelming that \( \text{Nodes}^p_{M, A, \Pi_c, k}(k) \in \mathcal{P} \).

**Definition 11 (Computational soundness).** A computational implementation \( A \) of a symbolic model \( M = (C, N, T, D, +) \) is computationally sound for a class \( \mathcal{P} \) of CoSP protocols iff for every trace property \( \mathcal{P} \) and for every efficient probabilistic CoSP protocol \( \Pi_s \), we have that \( (\Pi_s, A) \) computationally satisfies \( \mathcal{P} \) whenever the corresponding CoSP protocol \( \Pi_c \) of \( \Pi_s \) symbolically satisfies \( \mathcal{P} \) and \( \Pi_c \in \mathcal{P} \).

A computational soundness result with respect to a non-trivial message type \( T \) (a message type that does not contain all terms) may be useful when embedding a calculus in CoSP that supports typed messages (e.g., most modern programming languages). Many calculi, however, do not support typed messages. In this case, it may be impossible to directly represent the message type in the calculus. An example is the applied \( \pi \)-calculus presented in Section 4. To handle such calculi, the following lemma can be used. Intuitively, it states that if the protocol is guaranteed not to try to construct non-well-typed terms (terms not in \( T \)), one can remove the restriction to well-typed terms from the model, i.e., one can set \( T \) to be the set of all terms.

**Lemma 1 (Removing the message type).** Let \( M = (C, N, T, D, +) \) be a symbolic model, \( \mathcal{P} \) a class of symbolic protocols. We call a symbolic protocol \( \Pi_s \), \( T \)-conform if in any symbolic execution of \( \Pi_s \), no no-successor of a computation node annotated with a constructor is reached. Let \( T' \) be the set of all terms over \( C \cup N \) (the trivial message type), let \( \nu' \) be a relation on \( 2^T \times T \) with \( \nu' \supseteq \nu \), let \( M' = (C, N, T', D, +) \), and let \( \Pi' := (\Pi_{c, \Pi_s} \in \mathcal{P} : \Pi_s \text{ is } T\text{-conform}) \). Assume that \( A \) is a computationally sound implementation of \( M \) for protocols in \( \mathcal{P} \). Then \( A \) is a computationally sound implementation of \( M' \) for protocols in \( \mathcal{P}' \).

Due to space constraints, a proof of Lemma 1 appears in the full version [8].

### 2.5 A Sufficient Criterion for Soundness

To tame the complexity of computational soundness proofs, we introduce a technical tool to show soundness. We introduce the notion of a simulator and identify several properties such that the existence of a simulator with all of these properties already entails computational soundness in the sense of [Definition 11]. This notion might remind of the simulation-based proofs of computational soundness [12][10] [11][29], but it does not depend on framework-specific details such as scheduling, message delivery, etc. We stress that asserting computational soundness proofs need not be done in CoSP using this simulator-based characterization to enjoy CoSP’s benefits, but every other prevalently used technique for showing computational soundness can be used as well.

In what follows, we fix a symbolic model \( M = (C, N, T, D, +) \) and a computational implementation \( A \) of \( M \). We moreover assume that whenever a machine sends a term or a node, the term/node is suitably encoded as bitstring.

We proceed by introducing the notion of a simulator, essentially by imposing syntactic constraints on the set of all interactive machines.

**Definition 12 (Simulator).** A simulator is an interactive machine \( \text{Sim} \) that satisfies the following syntactic requirements:

- When activated without input, it replies with a term \( m \in T \). (This corresponds to the situation that the protocol expects a message from the adversary.)
- When activated with some \( t \in T \), it replies with an empty output. (This corresponds to the situation that the protocol sends a message to the adversary.)
- When activated with \( (\text{info}, \nu, t) \) where \( \nu \) is a node identifier and \( t \in T \), it replies with (proceed).
- At any point (in particular instead of sending a reply), it may terminate.

A simulator \( \text{Sim} \) is intuitively expected to translate a computational attack into a corresponding symbolic attack. Sim operates in a symbolic setting, but will usually simulate internally a computational adversary. Thus \( \text{Sim} \) essentially translates bitstrings to terms, and vice versa.
We denote the probability on node traces of this execution by \( H \). It is a mixture of the symbolic and the computational execution. Concretely, we define a hybrid protocol machine \( \Pi^C \) that is associated to \( \Pi_p \) and interfaces \( \text{Sim} \) with a probabilistic CoSP protocol \( \Pi_b \).

**Definition 13 (Hybrid execution).** Let \( \Pi_b \) be a probabilistic CoSP protocol, and let \( \text{Sim} \) be a simulator. We define a probability distribution \( H \cdot \text{Trace}_{\Pi_p, \Pi_b, \text{Sim}}(k) \) on (finite) lists of tuples \( (S, \nu, f_i) \) called the full hybrid trace according to the following probabilistic algorithm \( \Pi^C \), run on input \( k \), that interacts with \( \text{Sim} \). \( \Pi^C \) is called the hybrid protocol machine associated with \( \Pi_p \) and internally runs a symbolic simulation of \( \Pi_p \) as follows:

- **Start:** \( S_1 := S := \emptyset, \nu_1 := \nu \) is the root of \( \Pi_p \), and \( f_1 := f \) is a totally undefined partial function mapping node identifiers to \( T \). Run \( \Pi_p \) on \( \nu \).
- **Transition:** For \( i = 2, \ldots \) do the following:
  - Let \( \nu \) be the node identifiers in the label of \( \nu \). Let \( t_i := f(\nu_i) \).
  - **Proceed depending on the type of \( \nu \):**
    - If \( \nu \) is a constructor or destructor node with constructor, destructor, or nonce \( F \), then let \( m := F(\nu) \).
    - If \( m \neq \perp \), let \( \nu' \) be the yes-successor of \( \nu \) and let \( f' := f(\nu := m) \).
    - If \( m = \perp \), let \( \nu' \) be the no-successor of \( \nu \) and let \( f' := f \).
    - If \( \nu \) is an input node, send \( t_i \) to \( \text{Sim} \) (but without handing over control to \( \text{Sim} \)). Let \( \nu' \) be the unique successor of \( \nu \). Set \( \nu := \nu' \).
    - If \( \nu \) is an input node, hand control to \( \text{Sim} \), and wait to receive \( m \in T \) from \( \text{Sim} \). Let \( f' := f(\nu := m) \), and let \( \nu' \) be the unique successor of \( \nu \). Set \( f := f' \) and \( \nu := \nu' \).
    - If \( \nu \) is a control node labeled with out-metadata \( l \), send \( l \) to \( \text{Sim} \), hand control to \( \text{Sim} \), and wait to receive a bitstring \( l' \) from \( \text{Sim} \). Let \( \nu' \) be the successor of \( \nu \) along the edge labeled \( l' \) (or the lexicographically smallest edge if there is no edge with label \( l' \)). Let \( \nu := \nu' \).
    - If \( \nu \) is a nondeterministic node, sample \( \nu' \) according to the probability distribution specified in \( \nu \). Let \( \nu := \nu' \).
    - Send (info, \( \nu, t \)) to \( \text{Sim} \). When receiving an answer (proceed) from \( \text{Sim} \), continue.
    - If \( \text{Sim} \) has terminated, stop. Otherwise let \( (S, \nu, f) := (S, \nu, f) \).

We write \( \text{Sim} \cdot \Pi^C \) to denote the execution of \( \text{Sim} \) and \( \Pi^C \). We denote the probability on node traces of this execution by \( H \cdot \text{Nodes}_{\Pi_p, \Pi_b, \text{Sim}}(k) \).

We proceed by defining properties of a simulator that will imply soundness.

The first property – Dolev-Yao-style – captures that \( \text{Sim} \) adheres to the deduction relation \( \vdash \) in [Definition 7] for input/output nodes. More precisely, the terms that \( \text{Sim} \) sends to the CoSP protocol must be derivable from \( \text{Sim} \)'s symbolic view so far.

**Definition 14 (Dolev-Yao style simulator).** A simulator \( \text{Sim} \) is Dolev-Yao style (short: DY) for \( \Pi_p \) and \( \Pi_b \), if with overwhelming probability the following holds:

In an execution of \( \text{Sim} \cdot \Pi^C \), for each \( t \), let \( m_t \in T \) be the \( t \)-th term sent (during processing of one of \( \Pi^C \)'s input nodes) from \( \text{Sim} \) to \( \Pi^C \) in that execution. Let \( T_f \subseteq T \) the set of all terms that \( \text{Sim} \) has received from \( \Pi^C \) (during processing of output nodes) prior to sending \( m_t \). Then we have \( T_f \vdash m_t \).

The second property – indistinguishability – captures that the hybrid node traces are computationally indistinguishable from real node traces, i.e., the corresponding random variables cannot be distinguished by any probabilistic algorithm that runs in polynomial time in the security parameter. We write \( \approx \) to denote computational indistinguishability.

**Definition 15 (Indistinguishable simulator).** A simulator \( \text{Sim} \) is indistinguishable for \( \Pi_p, \Pi_b \), an implementation \( A \), an adversary \( E \), and a polynomial \( p \), if

\( \text{Nodes}^p_{\Pi_p, \Pi_b, \text{Sim}}(k) \approx H \cdot \text{Nodes}_{\Pi_p, \Pi_b, \text{Sim}}(k) \),

i.e., if the node trace and the hybrid node trace are computationally indistinguishable.

**Definition 16 (Good simulator).** A simulator is good for \( \Pi_p, \Pi_b, A, E \), and \( p \) if it is Dolev-Yao style for \( \Pi_p \) and \( \Pi_b \), and indistinguishable for \( \Pi_p, \Pi_b, A, E \), and \( p \).

We can now show that the existence of a good simulator implies computational soundness. The proof can be found in [3].

**Theorem 1 (Good simulator implies soundness).** Let \( \Pi = (C, N, T, D, \vdash) \) be a symbolic model, let \( P \) be a class of CoSP protocols, and let \( A \) be a computational implementation of \( \Pi \). Assume that for every efficient probabilistic CoSP protocol \( \Pi_b \) (whose corresponding CoSP protocol is in \( P \)), every probabilistic polynomial-time adversary \( E \), and every polynomial \( p \), there exists a good simulator for \( \Pi, \Pi_b, A, E \), and \( p \). Then \( A \) is computationally sound for protocols in \( P \).

3. CASE STUDY: COMPUTATIONAL SOUNDNESS OF PUBLIC-KEY ENCRYPTION AND SIGNATURES

In this section, we provide a symbolic model that allows for expressing encryption, signatures, and pairs, and we derive criteria under which a computational implementation of that model is computationally sound.

The symbolic model. We first specify the symbolic model \( \Pi = (C, N, T, D, \vdash) \):

- Constructors and nonces: Let \( C := \{E/3, c_k/1, d_k/1, \sigma_k/3, v_k/1, n_k/1, \text{pair}/2, \text{string}o/1, \text{string}1/1, \text{empty}/0, \text{garbage}Sig/2, \text{garbage}/1, \text{garbage}E/2 \} \) and \( N := N_P \cup N_E \). Here \( N_P \) and \( N_E \) are countably infinite sets representing protocol and adversary nonces, respectively. Intuitively, encryption, decryption, verification, and signing keys are represented as \( c_k(r), d_k(r), v_k(r), n_k(r) \) with a nonce \( r \) (the randomness used when generating the keys). \( E(c_k(r'), m, r) \) encrypts \( m \) using the encryption key \( c_k(r') \) and randomness \( r \). \( \sigma_k(sk(r'), m, r) \) is a signature of \( m \) using the signing key \( sk(r') \) and randomness \( r \). The constructors \( \text{string}o, \text{string}1, \text{empty} \) are used to model arbitrary strings used as payload in a protocol (e.g., a bitstring \( 010 \) would be encoded as \( \text{string}o(\text{string}1(\text{empty})) \)). \( \text{garbage}, \text{garbage}E, \) and \( \text{garbage}Sig \) are constructors necessary to express certain invalid terms the adversary may send, these constructors are not used by the protocol.
The computational implementation.

A computational soundness result for the symbolic model of these functions need not even fulfill equations like

\[ \pi \] .

We consider the process calculus proposed in [25] additionally augmented with events; the calculus in [25] itself is a combination of the original applied \( \pi \)-calculus with one of its dialects [23]. This combination offers the richness of the original applied \( \pi \)-calculus while additionally being accessible to state-of-the-art verification tools such as ProVerif [22]; in particular, we allow arbitrary equational theories. Our result hence yields computational soundness guarantees for ProVerif.

We first syntactically embed the applied \( \pi \)-calculus into CoSP. This embedding is particularly instructive because the applied \( \pi \)-calculus differs significantly from CoSP, e.g., the applied \( \pi \)-calculus models secrecy of nonces via restrictions, it does not rely on a labeled transition system, but it considers an equational theory. We then show that computational soundness of the embedding entails computational soundness of the applied \( \pi \)-calculus (in the sense of preservation of trace properties). Second, we provide a computational implementation of the embedding, and we prove it sound within CoSP.

4. COMPUTATIONAL SOUNDNESS OF THE APPLIED \( \pi \)-CALCULUS

In this section we show how to use CoSP to establish the first computational soundness result for the full-fledged applied \( \pi \)-calculus, including arbitrary equational theories, under active attacks. We consider the process calculus proposed in [25] additionally augmented with events; the calculus in [25] itself is a combination of the original applied \( \pi \)-calculus with one of its dialects [23]. This combination offers the richness of the original applied \( \pi \)-calculus while additionally being accessible to state-of-the-art verification tools such as ProVerif [22]; in particular, we allow arbitrary equational theories. Our result hence yields computational soundness guarantees for ProVerif.

We first syntactically embed the applied \( \pi \)-calculus into CoSP. This embedding is particularly instructive because the applied \( \pi \)-calculus differs significantly from CoSP, e.g., the applied \( \pi \)-calculus models secrecy of nonces via restrictions, it does not rely on a labeled transition system, but it considers an equational theory. We then show that computational soundness of the embedding entails computational soundness of the applied \( \pi \)-calculus (in the sense of preservation of trace properties). Second, we provide a computational implementation of the embedding, and we prove it sound within CoSP.

4.1 Overview of this section

We briefly outline the structure of this section, since it can be seen as a general guideline on how to embed other calculi into CoSP, and how to derive computational soundness guarantees for them.

We first review the syntax and the semantics of the applied \( \pi \)-calculus in Section 4.2. In Section 4.3 we define a computational execution of the calculus (this is only necessary since the applied \( \pi \)-calculus does not come with an \( \alpha \)-priori defined computational execution), called computational \( \pi \)-execution, as well as trace properties in the applied \( \pi \)-calculus, called \( \pi \)-trace properties. In Section 4.4 we establish the actual soundness result using CoSP. We first define a symbolic model of the applied \( \pi \)-calculus (in the sense of Definition 4) as well as a computational interpretation of this model (in the sense of Definition 8). The final theorem then asserts that if this computational implementation is computationally sound with respect to this symbolic model,
$M, N ::= \text{terms, build from variables } x, y, z, \text{ names } a, b, c, \text{ and constructor applications}$

$D ::= \text{ destructor terms, build from variables, names, constructor and destructor applications}$

$P, Q ::= \text{ processes}$

$M(N).P \quad \text{ output}$

$M(x).P \quad \text{ input}$

$0 \quad \text{ nil}$

$P | Q \quad \text{ parallel composition}$

$!P \quad \text{ replication}$

$\nu a.P \quad \text{ restriction}$

$let \ x = D \letin \ P \ else \ Q \quad \text{ let}$

$event(e).P \quad \text{ event}$

Figure 2: Syntax of the process calculus.

then every $\pi$-calculus process that symbolically fulfills a $\pi$-trace property also computationally fulfills this property.

4.2 Review of the calculus’ syntax and semantics

The syntax of the process calculus that we consider is provided in Figure 2 (We do not explicitly include an if-statement, but instead emulate it using destructor applications, see below.) Technically, it corresponds to the one considered in [25], except that we add processes of the form $\text{let } x = D \text{ let in } P \text{ else } Q$ for a string $e$. The intuitive meaning of such a process is that it raises an event $e$ and then proceeds to execute $P$.

In the following, we often call terms in the process calculus $\pi$-terms and terms in CoSP, i.e., in the sense of Section 2.2 CoSP-terms, in order to avoid ambiguities. We proceed similarly for other homonyms, such as $\pi$-constructors, $\pi$-traces, etc.

The set of ground $\pi$-terms is denoted $T_\pi$. By $fv(P)$ we denote the free variables of $P$, i.e., the variables that are not protected by a let or an input. We call a process closed if it has no free variables (but it may have free names).

The calculus is parametrized over a (possibly infinite) set of $\pi$-constructors $C_\pi$, a (possibly infinite) set of $\pi$-destructors $D_\pi$ (such as the constructors and destructors from Section 3), and an arbitrary equivalence relation $\approx$ over ground $\pi$-terms (describing, e.g., cancellations of certain terms). A destructor $d$ of arity $n$ is a partial function $T_\pi^n \to T_\pi$. We require that the equational theory is compatible with the $\pi$-constructors and $\pi$-constructors in the following sense: For all $\pi$-constructors $f$ and $\pi$-destructors $d$ of arity $n$, for all ground $\pi$-terms $M_1, \ldots, M_n, M_1', \ldots, M_n'$, with $M_i \approx M_i'$ for $i = 1, \ldots, n$, we have that $f(M_1, \ldots, M_n) \approx f(M_1', \ldots, M_n')$, that $d(M_1, \ldots, M_n) \approx d(M_1', \ldots, M_n')$, and that $d(M_1, \ldots, M_n, \tau) \approx d(M_1', \ldots, M_n', \tau)$ for any renaming $\tau$ of names.

We did not explicitly include an if-statement in the syntax of the calculus since such a statement can be expressed using an additional destructor $\text{equals}(x, y) = x$ for $x \approx y$ and define if $M = N$ then $P$ else $Q$ as let $x = \text{equals}(M, N)$ in $P$ else $Q$ for some $x \notin fv(P)$. In the following, we will assume $\text{equals} \in D_\pi$. We write let $x = D$ in $P$ for let $x = D$ in $P$ else 0 and analogously for if.

Given a ground destructor $\pi$-term $D$, we can evaluate it to a ground $\pi$-term $\text{eval}^\pi(D)$ by evaluating all $\pi$-destructors. If one of the $\pi$-destructors returns $\perp$, we set $\text{eval}^\pi(D) := \perp$. Analogously, we define $\text{eval}^{\text{CoSP}}(D)$ for terms $D$ involving CoSP-constructors, -constructors, and -nonces.

The semantics of the calculus is standard and corresponds to the one defined in [25] except for the addition of events. The semantics consists of two possible transitions: $\rightarrow$ and $\rightarrow^\ast$. The latter denotes that the event $e$ occurred, and we can define trace properties as properties over the sequence of events occurring in an execution of a process. For space constraints, we refer to the full version [8] for the formal definition of the semantics. Again, we prefix some notions with $\pi$ to distinguish them from their corresponding notions in Section 2.2.

Definition 17 ($\pi$-trace properties). A list of strings $e_1, \ldots, e_n$ is an event trace of $P$ if there is a process $Q$ that does not contain events such that $P \approx (Q | (e_1 \rightarrow \ast) | (e_2 \rightarrow \ast) | \ldots | (e_n \rightarrow \ast))$. A $\pi$-trace property is an efficiently decidable and prefix-closed set of strings. A process $P$ symbolically satisfies a $\pi$-trace property $\phi$ if we have $e \in \phi$ for all event traces $e$ of $P$.

4.3 Defining a computational execution

A computational $\pi$-implementation assigns a partial deterministic polynomial-time algorithm $A^n_\pi$ to each $\pi$-constructor $f$, and a partial deterministic polynomial-time algorithm $A^d_\pi$ to each $\pi$-destructor $d$. We also fix an efficiently sampleable set Nonces, depending on a security parameter $k$. We require that $A^{\text{eval}}(1^k, x, x) = x$ and $A^{\text{eval}}(1^k, x, y) = \perp$ for $x \neq y$, i.e., the computational interpretation of $\approx$ is the equality on bitstrings. Given an assignment $\mu$ from names to bitstrings and an assignment $\gamma$ from variables to bitstrings for names and variables occurring in a destructor term $D$, we can (computationally) evaluate $D$ to a bitstring $\text{eval}^{\mu,\gamma}(D)$. (Formally, the security parameter $k$ is an additional input to ceval, but we omit $k$ for readability.) We set $\text{eval}^{\mu,\gamma}(D) := \perp$ if the application of one of the algorithms $A^n_\pi$ or $A^d_\pi$ fails.

Given a computational implementation of the constructors and destructors, the computational execution of a process $P$ is already determined, except for the question how to model nondeterminism and which messages the adversary is allowed to observe. To resolve the nondeterminism in the calculus, we let the adversary have total control over the scheduling. This is a worst-case assumption and thus leads to the strongest result. An alternative would, e.g., be a scheduling that uniformly chooses between different execution paths. Furthermore, we have to reflect that the calculus allows the adversary to receive messages on any channel in his knowledge. For this, we allow the adversary to request a message on a channel if he can produce the bitstring corresponding to the channel’s name.

The computational implementation of a process is then defined using evaluation contexts: An evaluation context is a context with either one hole, or with two (distinguished) holes where each hole occurs only once and is located only below parallel compositions. In the case of two holes, we...
write $E[P][Q]$ to denote the replacement of the first hole by $P$ and of the second hole by $Q$.

The computational $\pi$-execution of a process is now defined as an interactive machine that executes the process and communicates with an adversary. The computational $\pi$-execution maintains a process $P$ representing the current process, an environment $\eta$ storing the bitstrings assigned to the free variables in $P$, and an interpretation $\mu$ of the free names in $P$ as bitstrings.

Due to space limitations, we have included the definition of the so-called symbolic $\pi$-execution into Definition 18 below (marked by […]). This notion can be ignored at this point.

**Definition 18 (Computational [[symbolic] $\pi$-execution]).** Let $P_0$ be a closed process, and let $C$ be an interactive machine called the adversary. We define the computational [symbolic] $\pi$-execution as an interactive machine $\text{Exec}_{\text{CoSP}}(1^k)$ that takes a security parameter $k$ as argument [interactive machine $\text{SEnc}_{\text{CoSP}}$ that takes no argument] and interacts with $C$:

- **Start:** Let $P := P_0$ (where we rename all bound variables and names such that they are pairwise distinct and distinct from all unbound ones). Let $\eta$ be a totally undefined function mapping variables to bitstrings $[[\text{CoSP-terms}]]$, let $\mu$ be a totally undefined function mapping names to bitstrings $[[\text{CoSP-terms}]]$. Let $a_1, \ldots, a_n$ denote the free names in $P_0$. For each $i$, pick $r_i \in \text{Nonces}_k$ at random [choose a different $r_i \in \mathbb{N}_P$]. Set $\mu := \mu(a_1 := r_1, \ldots, a_n := r_n)$. Send $(r_1, \ldots, r_n)$ to $C$.

- **Main loop:** Send $P$ to the adversary and expect an evaluation context $E$ from the adversary. Distinguish the following cases:
  - $P = E[M(x).P_1]$: Request two bitstrings $[[\text{CoSP-terms}]] c, m$ from the adversary. If $c = \text{ceval}_{\eta,\mu}(M)$ $[c \simeq \text{eval}_{\text{CoSP}}(M\eta\mu)]$, set $\eta := \eta(x := m)$ and $P := E[P_1]$.
  - $P = E[\nu.a.P_1]$: Pick $r \in \text{Nonces}_k$ at random [choose $r \in \mathbb{N}_P$] and $P := E[P_1]$ and $\mu := \mu(a := r)$.
  - $P = E[M_1(N).P_1][M_2(x).P_2]$: If $\text{ceval}_{\nu,a,\mu}(M_1)$ $[\text{ceval}\_{\nu,a,\mu}(M_2) \sim \text{eval}_{\text{CoSP}}(M_1\eta\mu) \simeq \text{eval}_{\text{CoSP}}(M_2\eta\mu)]$ then set $P := E[P_1][P_2]$ and $\eta := \eta(x := \text{ceval}_{\nu,a,\mu}(N))$ $[\eta := \eta(x := \text{eval}_{\text{CoSP}}(N\eta\mu))]$.
  - $P = E[\text{event}(e).P_1]$: Let $P := E[P_1]$ and raise the event $e$.
  - $P = E[!P_1]$: Rename all bound variables of $P_1$ such that they are pairwise distinct and distinct from all variables and names in $P$, and in the domains of $\eta$ and $\mu$, yielding a process $P_1$. Set $P := E[P_1][P_1]$.
  - $P = E[M(N).P_1]$: Request a bitstring $[[\text{CoSP-term}]] c$ from the adversary. If $c = \text{ceval}_{\nu,a,\mu}(M)$ $[c \simeq \text{eval}_{\text{CoSP}}(M\eta\mu)]$, set $P := E[P_1]$ and send $\text{ceval}_{\nu,a,\mu}(N) \simeq \text{eval}_{\text{CoSP}}(N\eta\mu)$ to the adversary.

- In all other cases, do nothing.

The execution of $\text{Exec}_{\text{CoSP}}(1^k)$ maintains the invariant that all bound variables and names in $P$ are pairwise distinct and that they are distinct from all variables and names in $P$ and in the domains of $\eta$ and $\mu$. For a given polynomial-time interactive machine $C$, a closed process $P_0$, and a polynomial $p$, we let $\text{Events}_{C,P_0,p}(k)$ the list of events $e$ raised within the first $\mu(k)$ computation steps (jointly counted for $C(1^k)$ and $\text{Exec}_{\text{CoSP}}(1^k)$).

In Definition 18, the adversary has full control over the scheduling, and he always knows at which point in the execution of the process we are. We stress that such a powerful adversary only makes our result stronger. However, since the adversary can easily distinguish between two syntactically different processes, say $A|B$ and $B|A$, we cannot expect preservation of observational equivalence. Thus, in an extension of our results to cover computational soundness of observational equivalence, the scheduling in the computational execution would have to be handled differently.

We finally define the computational fulfillment of $\pi$-trace properties.

**Definition 19 (Computational $\pi$-trace properties).** Let $P_0$ be a closed process, and $p$ a polynomial. We say that $P_0$ computationally satisfies a $\pi$-trace property $\phi$ if for all polynomial-time interactive machines $C$ and all polynomials $p$, we have that $\text{Pr}[\text{Events}_{C,P_0,p}(1^k) \in \phi]$ is overwhelming in $k$.

### 4.4 Computational soundness of the calculus

We will now derive the computational soundness of the applied $\pi$-calculus, i.e., we will show that its computational implementation is computationally sound in the sense of Definition 11 then every symbolically satisfied $\pi$-trace property is also computationally satisfied. Applying Definition 11 first requires us to specify a symbolic model of the applied $\pi$-calculus (in the sense of Definition 3) and a computational implementation of this model (in the sense of Definition 5).

The symbolic model of the applied $\pi$-calculus contains all the $\pi$-constructors and $\pi$-destructors from the process calculus. We additionally add an infinite number of adversary nonces $\mathbb{N}_E$ and protocol nonces $\mathbb{N}_P$ to represent names. The deduction relation allows the adversary to derive all adversary nonces and everything derivable by application of constructors and destructors.

**Definition 20 (Symbolic model of the applied $\pi$-calculus).** For a $\pi$-destructor $d$, we define $d' := d(\nu)(\eta)^{-1}$ where $\nu$ is any injective map from the nonces occurring in the CoSP-terms $\downarrow$ to names. Let $\mathbb{N}_E$ and $\mathbb{N}_P$ be countably infinite sets.

The symbolic model of the applied $\pi$-calculus is given by $M := (C, N, T, D, \vdash)$, where $N := \mathbb{N}_E \cup \mathbb{N}_P$, $C := \text{CoSP}$, $D := \{d : d \in \mathbb{D}_E\}$, and $T$ consists of all terms over $C$ and $\mathbb{N}$ and $\vdash$ is the smallest relation such that $m \in S \Rightarrow S \vdash m$, $N \in S \vdash N$, and such that for any $F \in C \cup D$ and any $t_1, \ldots, t_n \in T$ with $S \vdash \downarrow$ and $\text{eval}_{\nu}(\downarrow) \not\equiv \downarrow$ we have $S \vdash \text{eval}_{\nu}(\downarrow)$.

In the following, we fix $M, C, N, D, \vdash$ as in Definition 20

The destructor equals$^6$ induces an equivalence relation $\cong$ on$^6$

$^6$This is well-defined and independent of $\rho$ since for any renaming of names $\tau$, we have $d(M_\tau) = d(M)\tau$; intuitively $d'$ behaves as $d$ except that it uses nonces instead of names.

$^7$One can get a model with such a message type $T$ using, e.g., Lemma 1.
the set of CoSP-terms with \( x \equiv y \) iff \( \text{equal}(x, y) \neq \perp \). The relation \( \equiv \) is the analogue to the equivalence relation \( \approx \) describing the equational theory of the applied \( \pi \)-calculus.

The computational implementation of this symbolic model is now specified by the computational \( \pi \)-implementations \( A_f \) and \( A_d \) of the \( \pi \)-constructors and \( \pi \)-destructors, with noones being chosen uniformly at random.

**Definition 21. (Computational implementation of Def. 20)** The computational implementation \( A \) of the symbolic model \( M \) of the applied \( \pi \)-calculus is given by \( A_f := A^*_f \) for all \( f \) in \( C \) and \( A_d := A^*_d \) for all \( d \) in \( D \). \( A_N \) for \( N \) in \( N \) picks \( r \) in \( \text{Nonces}_r \) uniformly at random and returns \( r \).

In order to relate the symbolic and the computational semantics of a process, we define an additional symbolic execution for closed processes as a technical tool. This new semantics constitutes a safe approximation of the original semantics of the process calculus while at the same time being a direct analogue of the computational semantics presented in Definition 18. The semantics is defined by means of an interactive nondeterministic machine \( S \text{Exec}_\pi \), see the [...] part of Definition 18. The only difference between \( \text{Exec}_\pi \) and \( S \text{Exec}_\pi \) is that the latter operates immediately on terms whenever the former operates on computational implementations of these terms.

In the full version [8], we show that the machine \( S \text{Exec}_\pi \) can be realized as a CoSP protocol in the sense of Definition 4 by encoding protocol steps and adversarial interactions as CoSP nodes. We call this CoSP protocol \( \Pi_\pi \). Raising an event \( e \) is modeled in the protocol by a control node with one successor that sends (event, \( e \)) to the adversary. We call these nodes event nodes, and given a sequence of nodes \( \nu \), we denote by \( \text{events}(\nu) \) the events \( e \) raised by the event nodes in \( \nu \). By construction, \( \Pi_\pi \) does not contain nondeterministic nodes; hence it is a CoSP protocol and a probabilistic CoSP protocol at the same time.

**Definition 22.** A nondeterministic interactive machine \( C \) is a Dolev-Yao adversary if the following holds in an interaction with any interactive machine \( M \) in each step of the interaction: Let \( S \) be the set of all CoSP-terms sent by \( M \) up to the current step. Let \( m \) be the term sent by \( C \) in the current step. Then \( S \vdash m \).

\( S \text{Exec}_\pi \) satisfies a \( \pi \)-trace property \( \phi \) if in a finite interaction with any Dolev-Yao adversary, the sequence of events raised by \( S \text{Exec}_\pi \) is contained in \( \phi \).

Before we finally state and prove the soundness of the applied \( \pi \)-calculus, we provide three lemmas that are used to relate \( P_\pi \), \( S \text{Exec}_\pi \), \( \Pi_\pi \), and the computational implementation \( A \), and to assert the efficiency of the protocol \( \Pi_\pi \). Figure 8 illustrates the use of these lemmas in the overall proof.

**Lemma 2.** Let \( \phi \) be a trace property. Then \( S \text{Exec}_\pi \) satisfies \( \phi \) iff \( \Pi_\pi \) symbolically satisfies \( \text{events}^{-1}(\phi) \) (in the sense of Definition 18). Moreover, \( P_\pi \) computationally satisfies \( \phi \) iff \( (P_\pi, A) \) computationally satisfies \( \text{events}^{-1}(\phi) \) (in the sense of Definition 18).

**Lemma 3.** The probabilistic CoSP protocol \( \Pi_\pi \) is efficient.

**Lemma 4.** If a closed process \( P_\pi \) symbolically satisfies a \( \pi \)-trace property \( \phi \), then \( S \text{Exec}_\pi \) satisfies \( \phi \).

With these lemmas at hand, we are finally ready to state and prove the computational soundness of the applied \( \pi \)-calculus as the main result of this section.

**Theorem 3.** (Comp. soundness in the applied \( \pi \)-calculus.) Assume that the computational implementation of the applied \( \pi \)-calculus (Definition 21) is a computationally sound implementation (in the sense of Definition 11) of the symbolic model of the applied \( \pi \)-calculus (Definition 20) for a class \( P \) of protocols.

If a closed process \( P_\pi \) symbolically satisfies a \( \pi \)-trace property \( \phi \), and \( P_\pi \in P \), then \( P_\pi \) computationally satisfies \( \phi \).

**Proof.** Assume that \( P_\pi \) symbolically satisfies \( \phi \). By Lemma 3 \( S \text{Exec}_\pi \) satisfies \( \phi \). By Lemma 2 \( \Pi_\pi \) symbolically satisfies \( \text{events}^{-1}(\phi) \). Furthermore, since \( \phi \) is an efficiently decidable, prefix closed set, so is \( \text{events}^{-1}(\phi) \). So \( \text{events}^{-1}(\phi) \) is a CoSP-trace property in the sense of Definition 10. From Lemma 3 we have that \( \Pi_\pi \) is an efficient CoSP protocol. By assumption, the computational implementation \( A \) of the applied \( \pi \)-calculus is computationally sound; hence \((\Pi_\pi, A)\) computationally satisfies \( \text{events}^{-1}(\phi) \). Using Lemma 2 we obtain that \( P_\pi \) computationally satisfies \( \phi \).

4.5 Computationally sound encryption and signatures in the applied \( \pi \)-calculus

Consider an instantiation of our process calculus with the constructors and destructors described in Section 3. Assume an implementation of the constructors or destructors satisfying the implementation conditions given in Section 3.

We call a process \( P \) key-safe if it has the following grammar: Let \( x, x_d, x_s \) stand for distinct sets of variables (general purpose, decryption key, and signing key variables). Let \( a \) and \( r \) stand for two sets of names (general purpose and randomness names). Then the allowed terms are \( M, N := x \mid a \mid \text{pair}(M, N) \mid S \text{ with } S := \text{string}(\tilde{S}) \mid \text{empty}, \text{the allowed destructor terms are } D := \tilde{M} \mid \text{isek}(\tilde{D}) \mid \text{isenc}(\tilde{D}) \mid D(x_d, \tilde{D}) \mid \text{fst}(\tilde{D}) \mid \text{snd}(\tilde{D}) \mid \text{ekof}(\tilde{D}) \mid \text{equals}(\tilde{D}, \tilde{D}) \mid \text{isek}(\tilde{D}) \mid \text{issig}(\tilde{D}) \mid \text{verify}(\tilde{D}, \tilde{D}) \mid \text{ekof}(\tilde{D}) \mid \text{unstring}(\tilde{D}) \mid \text{unstring}(\tilde{D}) \).

The allowed processes are

\[
\tilde{P}, \tilde{Q} := M(N).\tilde{P} \mid M(x).\tilde{P} \mid 0 \mid (P \mid \tilde{Q}) \mid \tilde{P} \mid \text{var} \tilde{P} \mid \text{let } x = \tilde{D} \text{ in } \tilde{P} \text{ else } \tilde{Q} \mid \text{event}(\tilde{e}) \tilde{P} \mid \text{vr} \text{ let } x = \text{ek}(r) \text{ in } \tilde{D}_r \text{ in } \tilde{P} \mid \text{vr} \text{ let } x = D(\text{isek}(\tilde{D}_1), \tilde{D}_2, r) \text{ in } \tilde{P} \text{ else } \tilde{Q} \mid \text{vr} \text{ let } x = \text{ek}(r) \text{ in } \tilde{D}_r \text{ in } \tilde{P} \mid \text{vr} \text{ let } x = \text{sig}(x, \tilde{D}_1, r) \text{ in } \tilde{P} \text{ else } \tilde{Q}.
\]

(Note that in the last four production rules for key generation and for encryption, all occurrences of \( r \) denote the same name.)

**Theorem 4.** If a closed key-safe process \( P \) symbolically satisfies a \( \pi \)-trace property \( \phi \), then \( P \) computationally satisfies \( \phi \).

The constructors and destructors described in Section 3 constitute an equational theory suitable for analysis using Proverif. As an example application, we have verified the Needham-Schroeder-Lowe protocol in Proverif, and by Theorem 4 we know that the verification also entails the security of a computational implementation of Needham-Schroeder-Lowe.
5. CONCLUSION AND FUTURE WORK

We have described CoSP, a general framework for conducting computational soundness proofs of symbolic models and for embedding these proofs into formal calculi. CoSP considers arbitrary equational theories and computational implementations, and it abstracts away many details that are not crucial for proving computational soundness such as message scheduling, corruption models, and even the internal structure of a protocol. CoSP enables soundness results, in the sense of preservation of trace properties, to be proven in a conceptually modular and generic way: proving $x$ cryptographic primitives sound for $y$ calculi only requires $x + y$ proofs, and the process of embedding calculi is conceptually decoupled from computational soundness proofs of cryptographic primitives.

We have shown how to use CoSP to establish the first computational soundness result for the full-fledged applied $\pi$-calculus under active attacks, by embedding the calculus into CoSP and by particularly providing a sound implementation of public-key encryption and digital signatures.

CoSP currently only considers computational soundness in the sense of preservation of trace properties. We plan to leverage existing definitions of the preservation of more sophisticated properties such as static or observational equivalence into CoSP. Moreover, we plan to derive the computational soundness of additional calculi, especially those ones that strive for analyzing security protocols in more realistic settings. Calculi for reasoning about implementations of security protocols such as RCF are hence particularly promising targets for this future work.

Acknowledgements. We thank Hubert Comon-Lundh, Ankit Malik, Matteo Maffei, Esfandiar Mohammadi, and Bogdan Warinschi for valuable discussions. We also thank the anonymous reviewers for helpful and constructive comments.

6. REFERENCES


