CoSP: A general framework for computational soundness proofs
– or –
The computational soundness of the applied pi-calculus

Michael Backes[1,3], Dennis Hofheinz[2], and Dominique Unruh[1]
[1]Saarland University [2]CWI [3]MPI-SWS

June 3, 2009

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 calculi only requires $x + y$ proofs (instead of $x \cdot y$ proofs without this framework), and the process of embedding calculi 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 π-calculus under active attacks. Concretely, we embed the applied π-calculus into CoSP and give a sound implementation of public-key encryption and digital signatures.

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 [37, 38, 51], see, e.g., [44, 56, 3, 50, 55, 18]. 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.[1] Moreover, many of these frameworks were freshly invented in the respective papers;

[1]To get a sense of the diversity of the statements: [11, 45, 32] all establish (among others results) the computational soundness of symbolic encryption (either symmetric or asymmetric): First, [11] expresses protocols as probabilistic input-output automata, exploits the communication model offered by the Reactive Simulatability (RSIM) framework [15], and shows computational soundness in the sense of reactive simulatability for IND-CCA2-secure, additionally randomized

they hence lack support from suitable verification tools and are more likely to suffer from idiosyncracies 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, etc.) stem from idiosyncracies of the underlying framework, or if they constitute conceptual limitations of computational soundness result 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.[2]

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 phrased 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 π-calculus. We show how to use CoSP to establish the first computational soundness result for a full-fledged applied π-calculus under active attacks. We consider the process calculus proposed in [24] additionally augmented with events; the calculus in [24] itself is a combination of the original applied π-calculus [2] with one of its dialects [22]. This combination offers the richness of the original applied π-calculus while additionally being accessible

encryption schemes. Second, [45] expresses protocols and their communication using a newly introduced concept called abstract algebras, and shows computational soundness in the sense of preservation of static equivalence in the presence of an adaptive, but passive adversary for IND-CPA-secure encryption schemes that are additionally length-concealing. Third, [32] expresses protocols and their communication within a small fragment of the applied π-calculus, and shows computational soundness in the sense of preservation of observational equivalence in the presence of an active adversary for IND-P1-C1-secure encryption schemes.

[2]We currently only consider the preservation of trace properties in CoSP. Extending CoSP to the preservation of more sophisticated properties such as observational equivalence [32] is left for future work.

to state-of-the-art verification tools such as ProVerif [21]; in particular, we allow arbitrary equational theories. Our result hence yields computational soundness guarantees for ProVerif.

We first syntactically embed the applied π-calculus into CoSP. This embedding is particularly instructive because the semantics of the applied π-calculus differs significantly from CoSP's semantics. We then show that computational soundness of the embedding entails computational soundness of the applied π-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. 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 π-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 [4, 46, 40, 1], but still apply only for passive adversaries. This excludes most 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 interactive protocols (within the Reactive Simulatability (RSIM) Framework [15]) was first given by Backes, Pfitzmann, and Waidner in [11] with extensions in [12, 9]. Tool support for this Dolev-Yao model was subsequently added in [57]. Laud [47] 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 [52] and Janvier, Lakhnech, and Mazaré [42] have presented cryptographic underpinnings for a Dolev-Yao model of public-key encryption. Their results are narrower than those in [11] since they are specific for public-key encryption and restricted classes of protocols, but they consider simpler real implementations. Baudet, Cortier, and Kremer [19] have established the soundness of specific classes of equational theories in a Dolev-Yao model under passive attacks. Canetti and Herzog [28] have shown that a Dolev-Yao-style symbolic analysis can be conducted using the framework of universal composability [27] for a restricted class of protocols.

Subsequent work concentrated on linking symbolic and cryptographic secrecy properties. Cortier and Warinschi [35] have shown that symbolically secret nonces are also computationally secret, i.e., indistinguishable from a fresh random value given the view of the adversary. Backes and Pfitzmann [10] and Canetti and Herzog [28] have established new symbolic criteria for showing that a key is cryptographically secret. Laud [48] has designed a type system for proving payload secrecy of security protocols based on the BPW model [11]. His work is extended to key secrecy in [8]. Kremer and Mazaré [45] 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 [32] 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 [30] 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 [34] and Backes, Pfitzmann, and Waidner [14] have shown computational soundness of hash functions in the random oracle model. Janvier, Lakhnech, and Mazaré [43] 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 [39] have showed computational soundness of hash functions under passive adversaries when implemented using perfect one-way hash functions [29]. Backes and Unruh [16] have shown computational soundness of non-interactive zero-knowledge proofs. Bresson, Lakhnech, Mazaré, and Warinschi [26] 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 Simultability were shown by Backes and Pfitzmann for hash functions [14] and the XOR operation [13].

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 [53, 49], security logics [41, 36] and cryptographic games [23, 25, 33, 54, 17, 7]. 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 2.5 introduces the notion of a simulator, and it identifies which properties a simulator need 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 A general 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 $t\varphi$ 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 $z \neq x$. We abbreviate $x_1, \dots, x_n$ with $\underline{x}$ if n is clear from the context. Given a set M and a function f, we define $f^{-1}(M) := \{x : f(x) \in M\}$. 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 $x \in M$ implies $x' \in M$ 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 and destructors) *A* constructor *C is a symbol with an (possibly zero) arity. A* nonce *N is a symbol with zero arity. We write $C/n \in \mathbf{C}$ to denote that $\mathbf{C}$ contains a constructor C with arity n. A* message type $\mathbf{T}$ over $\mathbf{C}$ and $\mathbf{N}$ *is a set of terms over constructors $\mathbf{C}$ and nonces $\mathbf{N}$.*

A destructor *D of arity n, written D/n, over a message type $\mathbf{T}$ is a partial map $\mathbf{T}^n \to \mathbf{T}$. If D is undefined on $\underline{t}$, we write $D(\underline{t}) = \bot$.*

In the following, we only consider sets of constructors $\mathbf{C}$ such that the same constructors cannot have different arities, i.e., $C/n, C/m \in \mathbf{C}$ implies $n = m$. (This restriction simplifies notion and is without loss of generality, as one can simulate multi-arity constructors by adding the arity to the name of the constructor.) We moreover assume that constructors have symbols that are bitstrings, and similarly for destructors and node identifiers in symbolic CoSP protocols as introduced below.

Predicates and how to model arbitrary (non-free) equational theories. A predicate P of arity n over a set of constructors $\mathbf{C}$ is a subset of $\mathbf{T}^n$. Predicates can be used to describe arbitrary tests that a protocol may perform. In particular, they can describe the equality test $=$ which is the diagonal on $\mathbf{T}^2$ for free equational theories, and the equivalence relation between terms in non-free equational theories. Since each predicate P can be realized using a destructor D by defining $D(t_1, \dots, t_n) := t_1$ if $P(t_1, \dots, t_n) = \mathit{true}$ and $D(t_1, \dots, t_n) := \bot$ otherwise, predicates however do not require an explicit treatment. This way, we can model any non-free equational theory as a free theory modulo a suitable equivalence relation formalized by predicates/destructors.

We now define which terms can be deduced from other terms; this is formalized using a deduction relation $\vdash$ over a set of terms $\mathbf{T}$. The intuition of $S \vdash m$ for $S \subseteq \mathbf{T}$ and $m \in \mathbf{T}$ is that the term m can be deduced from the terms in S.

Definition 2 (Deduction relation) *A* deduction relation $\vdash$ *over a message type* $\mathbf{T}$ *is a relation between* $2^{\mathbf{T}}$ *and* $\mathbf{T}$.

In most cases, the adversary can apply all constructors and destructors. This can be modelled by defining $S \vdash \underline{t} \Rightarrow S \vdash C(\underline{t})$ for every constructor C and $S \vdash \underline{t} \wedge D(\underline{t}) \neq \bot \Rightarrow S \vdash D(\underline{t})$ 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.

2.2 Symbolic protocols

We define a symbolic CoSP protocol as a tree with a distinguished root and with labels on both nodes and edges. Intuitively, the node labels correspond to different protocol actions: *Constructor and destructor nodes* produce terms (using a constructor or destructor); *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 determine the control flow of the protocol. Moreover, the node labels contain unique identifiers for each node. 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 3 (Symbolic CoSP protocol) *A* CoSP protocol Π_s *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 (labels):*

- Constructor nodes *are annotated with a constructor or nonce C/n together with the identifiers of n (not necessarily distinct) nodes. Constructor nodes have exactly two successors; the corresponding edges are labeled with yes and no, respectively.*
- Destructor nodes *are annotated with a destructor D/n together with the identifiers of n (not necessarily distinct) nodes. Destructor 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 (the out-metadata). A control node has at least one and up to countably many successors annotated with distinct bitstrings $l' \in \{0,1\}^*$. (We call l' the in-metadata.)*
- Nondeterministic nodes *have no further annotation. Nondetermininistic nodes have at least one and at most finitely many successors; the corresponding edges are labeled with bitstrings.*

If a node N contains an identifier N' in its label, then N' has to be on the path from the root to N (including the root, excluding N), and N' must be a constructor node, destructor node, or input node. In case N' is a destructor or constructor node, then 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 4 (Probabilistic CoSP protocol) *A* probabilistic CoSP protocol Π_p *is a CoSP protocol, where each nondeterministic node is additionally annotated with a probability distribution on the labels of the outgoing edges.*

In the following, we assume that such a probability distribution is encoded as a list of pairs, consisting of a label and a rational probability. Any probabilistic CoSP protocol Π_p can be transformed canonically into a CoSP protocol Π_s by erasing the probability distributions. We will call Π_s the symbolic protocol that *corresponds to* Π_p.

Probabilistic CoSP protocols will be crucial in the definition of computational soundness. Moreover, they often constitute an intermediate technical step within a larger proof. For instance, reasoning about implementations of CoSP protocols is difficult since they do not have a unique such implementation if nondeterministic nodes are present, in contrast to probabilistic CoSP protocols. With the notion of probabilistic CoSP protocols at hand, one can instead consider the set of all implementations of all probabilistic CoSP protocols whose corresponding CoSP protocol is Π_s.

Definition 5 (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.*

We finally provide the notions of a symbolic model and of a symbolic execution of a CoSP protocol.

Definition 6 (Symbolic model) *A* symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ *consists of a set of constructors* $\mathbf{C}$*, a set of nonces* $\mathbf{N}$*, a message type* $\mathbf{T}$ *over* $\mathbf{C}$ *and* $\mathbf{N}$ *with* $\mathbf{N} \subseteq \mathbf{T}$*, a set of destructors* $\mathbf{D}$ *over* $\mathbf{T}$*, and a deduction relation* $\vdash$ *over* $\mathbf{T}$.

The symbolic execution of a CoSP protocol for a given symbolic model consists of a sequence of triples (S, ν, f) where S represents the knowledge of the adversary, ν 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* $(\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ *and a CoSP protocol* Π_s *be given. A* full trace *is a (finite) list of tuples* (S_i, ν_i, f_i) *such that the following conditions hold:*

- Correct start*: $S_1 = \varnothing$, ν_1 is the root of* Π_s*, f_1 is a totally undefined partial function mapping node identifiers to terms.*
- Valid transition*: For every two consecutive tuples (S, ν, f) and (S', ν', f') in the list, let $\underline{\tilde{\nu}}$ be the node identifiers in the label of ν and $\tilde{t}_k := f(\tilde{\nu}_k)$. We have:*
  - *If ν is a constructor node with constructor C, then $S' = S$. If $m := C(\underline{\tilde{t}}) \in \mathbf{T}$, ν' is the yes-successor of ν in Π_s, and $f' = f(\nu := m)$. If $m \notin \mathbf{T}$, then ν' is the no-successor of ν and $f' = f$.*
  - *If ν is a destructor node with destructor D, then $S' = S$. Let $m := D(\underline{\tilde{t}})$. If $m \neq \bot$, then ν' is the yes-successor of ν in Π_s; if $m = \bot$, then ν' is the no-successor of ν in Π_s. We have $f' = f(\nu := m)$.*

– *If ν is an input node, then $S' = S$ and ν' is the successor of ν in Π_s and there exists an m with $S \vdash m$ and $f' = f(\nu := m)$.*
– *If ν is an output node, then $S' = S \cup \{\tilde{t}_1\}$, ν' is the successor of ν in Π_s and $f' = f$.*
– *If ν is a control node, then ν' is a successor of ν and $f' = f$ and $S' = S$.*
– *If ν is a nondeterministic node, then $S' = S$, ν' is some successor of ν in Π_s, and $f' = f$.*

A list of node identifiers (ν_i) is a node trace *if there is a full trace with these node identifiers.*

Example. To illustrate our notion of a CoSP protocol, and how exactly annotated constructors work, consider the setting of the Needham-Schroeder-Lowe protocol with two parties A and B. The first step of the protocol can be written as follows in standard protocol notation:

$$A \to B : \quad E_{ek_B}(N_A, id_A).$$

This means that A sends the encryption $E_{ek_B}(N_A, id_A)$ of a fresh nonce N_A and A's identity id_A under B's encryption key ek_B to B. We show how to model this protocol step as a CoSP protocol in the sense of Definition 3.

An appropriate full symbolic model $\mathbf{M}$ is described in Section 3. We will however only use the following constructors:

- A pairing $pair/2$ that constructs a compound term (x, y) from two terms x and y.
- Encryption key and decryption key constructors $ek/1$, resp. $dk/1$ that take a nonce N as argument. Intuitively, N is the randomness used for key generation. That is, $dk(N)$ is the decryption key associated to the encryption key $ek(N)$ for the same N.
- An encryption constructor $E/3$ that constructs a ciphertext $E(ek(N), m, N')$ from an encryption key $ek(N)$, a message term m, and a nonce N' that represents randomness used during encryption.

In our basic formalization, there is no notion of party identities. Such a notion can be formalized by constructors that, e.g., tie identities to public keys. However, for simplicity, we will treat the public encryption key of a party as its identity. Now first consider the following nodes, that represent the initial key setup phase and are not part of the actual protocol:

1. Choose randomness for A's keypair: A constructor for a protocol nonce $N_{A,key}$.
2. Construct A's encryption key $ek(N_{A,key})$: A constructor ek annotated with (the identifier of) node 1.
3. Publish A's encryption key $ek(N_{A,key})$: An output node annotated with (the identifier of) node 2.
4. Choose randomness for B's keypair: A constructor for a protocol nonce $N_{B,key}$.
5. Construct B's encryption key $ek(N_{B,key})$: A constructor ek annotated with (the identifier of) node 4.
6. Publish B's encryption key $ek(N_{B,key})$: An output node annotated with (the identifier of) node 5.

These nodes can be chained together by letting all successors (i.e., the unique successor in case of output nodes, or both the yes- and the no-successor in case of constructor nodes) of node i be node $i + 1$. We turn to the main protocol step:

7. Choose A's nonce: A constructor for a protocol nonce N_A.
8. Choose encryption randomness: A constructor for a protocol nonce $N_{A,enc}$.
9. Construct plaintext (N_A, id_A): A pairing constructor annotated with (the identifiers of) nodes 7 and 2. (Recall that we treat the encryption key as the party's identity.)
10. Construct ciphertext $E(ek(N_{B,key}), (N_A, id_A), N_{A,enc})$: A constructor E annotated with (the identifiers of) nodes 5, 9, and 8.
11. Send $E(ek(N_{B,key}), (N_A, id_A), N_{A,enc})$: An output node annotated with (the identifier of) node 10.

Again, these nodes can be chained by letting all successors of node i be node $i + 1$. Of course, the encryption key $ek_B = ek(N_{B,key})$ does not have to be hardwired into the protocol. Instead, it could be an external input, modeled as an input node. (In that case, the annotations of node 10 would be to that input node, and to the nodes 9 and 8.)

We sketch how to model that B processes an incoming ciphertext. We first add an input node to reflect that B receives a message. We then add a decryption destructor node that is annotated by a node representing B's decryption key $dk(N_{B,key})$, and by the previous input node. The no-successor of the decryption node is a node that represents failure of the authentication session (since decryption of the received ciphertext under B's decryption key failed). The yes-successor proceeds with the protocol (i.e., it first checks that the decryption corresponds to a pairing).

2.3 Computational model

We now define the computation implementation of a symbolic model as a family of functions that provide computation interpretations to constructors and destructors.

Definition 8 (Computational implementation) *Let a symbolic model* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ *be given. A* computational implementation of $\mathbf{M}$ *is a family of functions* $A = (A_x)_{x \in \mathbf{C} \cup \mathbf{D} \cup \mathbf{N}}$ *such that* A_F *for* $F/n \in \mathbf{C} \cup \mathbf{D}$ *is a total deterministic function* $\mathbb{N} \times (\{0,1\}^*)^n \to \{0,1\}^*$*, and* A_N *for* $N \in \mathbf{N}$ *is a probabilistic function with domain* $\mathbb{N}$ *and range* $\{0,1\}^*$ *(i.e., it specifies a probability distribution on bitstrings that depends on its argument).*

All functions A_C*,* A_D *have to be computable in deterministic polynomial-time, and* A_N *has to be computable in probabilistic polynomial-time.*[3]

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 occuring in the computational executions will not be needed in our later results.

Note on the computational interpretation of symbols. Here and in the following, we will assume a canonical bitstring representation of symbols. We do *not* require that the bitstring representation of a term, say, $E(m)$ hides (the bitstring representation of) m. (A suitable secrecy property will be ensured by an additional Dolev-Yao-like requirement on the computational machine that receives and sends bitstring representations of terms.) The purpose of the bitstring presentation is only to be *syntactically* able to consider a CoSP protocol in a computational setting.

Definition 9 (Computational execution) *Let a symbolic model* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$*, a computational implementation* A *of* $\mathbf{M}$*, and a probabilistic CoSP protocol* Π_p *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 the length of the first input of* E*), and let* p *be a polynomial. We define a probability distribution* $Nodes^p_{M,A,\Pi_p,E}(k)$ *on (finite) lists of node identifiers* (ν_i) *according to the following probabilistic algorithm (both the algorithm and* E *are run on input* 1^k*):*

- *Initial state:* $\nu_1 := \nu$ *is the root of* Π_p*. Let* f *be an initially empty partial function from node identifiers to bitstrings, and* n *an initially empty partial function from* $\mathbf{N}$ *to bitstrings.*
- *For* $i = 1, \ldots$ *do the following:*
  - *Let* $\underline{\tilde{\nu}}$ *be the node identifiers in the label of* ν*. Let* $\tilde{m}_k := f(\tilde{\nu}_k)$*.*
  - *Proceed depending on the type of node* ν*:*

[3]More precisely, there has to exist a single uniform probabilistic polynomial-time algorithm A that, given the name of C, D, or N, together with an integer k and the inputs $\underline{m}$, computes the output of A_C, A_D, and A_N or determines that the output is undefined. This algorithm must run in polynomial-time in $k + |m|$ and may not use random coins when computing A_C and A_D.

    * *If ν is a constructor node with nonce $N \in \mathbf{N}$: Let $m' := n(N)$ if $n(N) \neq \perp$ and sample m' according to $A_N(k)$ otherwise. Let ν' be the successor of ν, $f' := f(\nu := m')$, and $n' := n(N := m')$. Let $\nu := \nu'$, $f := f'$ and $n := n'$.*
    * *If ν is a constructor or destructor node with constructor or destructor F, then $m' := A_F(k, \underline{\tilde{m}})$. If $m' \neq \perp$, then ν' is the yes-successor of ν, if $m' = \perp$, then ν' is the no-successor of ν. Let $f' := f(\nu := m')$. Let $\nu := \nu'$ and $f := f'$.*
    * *If ν is an input node, ask for a bitstring m from E. Abort the loop if E halts. Let ν' be the successor of ν. Let $f := f(\nu := m)$ and $\nu := \nu'$.*
    * *If ν is an output node, send $\tilde{t}_1$ to E. Abort the loop if E halts. Let ν' be the successor of ν. Let $\nu := \nu'$.*
    * *If ν 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 ν' be the successor of ν along the edge labeled l' (or the lexicographically smallest edge if there is no edge with label l'). Let $\nu := \nu'$.*
    * *If ν is a nondeterministic node, let $\mathcal{D}$ be the probability distribution on the label of ν. Pick ν' according to the distribution $\mathcal{D}$, and let $\nu := \nu'$.*
  - *Let $\nu_i := \nu$.*
  - *Let len be the number of nodes from the root to ν plus the total length of all bitstrings in the range of f. If $len > p(k)$, 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 $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ be a symbolic model and Π_s a CoSP protocol. Then Π_s symbolically satisfies *a trace property $\mathcal{P}$ in $\mathbf{M}$ iff every node trace of Π_s is contained in $\mathcal{P}$. Let A be a computational implementation of $\mathbf{M}$ and let Π_p be a probabilistic CoSP protocol. Then (Π_p, A)* computationally satisfies *a trace property $\mathcal{P}$ in $\mathbf{M}$ iff for all probabilistic polynomial-time interactive machines E and all polynomials p, the probability is overwhelming that $Nodes^p_{\mathbf{M},A,\Pi_p,E}(k)$ is contained in $\mathcal{P}$.*

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

A computational soundness result with respect to a non-trivial message type $\mathbf{T}$ (a message type that does not contain all terms) may be useful when embedding a calculus in our model 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 π-calculus presented in Section 4. To handle such calculi, the following lemma can be used.

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

Proof. Fix a trace property $\mathcal{P}$ and an efficient probabilistic CoSP protocol Π_p with $\Pi_s \in P' \subseteq P$. Assume that (Π_p, A) does not computationally satisfy $\mathcal{P}$. Since A is a computationally sound implementation of $\mathbf{M}$, we have that Π_s does not symbolically satisfy $\mathcal{P}$ with respect to the model $\mathbf{M}$. Hence there is a node trace $\underline{\nu} \notin \mathcal{P}$ of Π_s with respect to $\mathbf{M}$. Since in the symbolic execution leading to $\underline{\nu}$, no **no**-successor of a constructor is ever reached, we have that $\underline{\nu}$ is also a symbolic trace with respect to the symbolic model $(\mathbf{C}, \mathbf{N}, \mathbf{T}', \mathbf{D}, \vdash)$. Since $\vdash'$ is finer than $\vdash$, $\underline{\nu}$ is also a symbolic trace with respect to $\mathbf{M}' = (\mathbf{C}, \mathbf{N}, \mathbf{T}', \mathbf{D}, \vdash')$. Thus Π_s does not symbolically satisfy $\mathcal{P}$ with respect to the symbolic model $\mathbf{M}'$. □

2.5 A Sufficient Criterion for Soundness

As a technical tool to show soundness, we introduce the notion of a simulator and identify properties a simulator might enjoy. We show that the existence of a simulator that enjoys these properties already suffices to establish computational soundness in the sense of Definition 11. A soundness proof can thus concentrate on the construction of a suitable simulator.

In the following, we fix a symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ and a computational implementation A of $\mathbf{M}$. In the following, 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 Sim that satisfies the following syntactic requirements:*

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

A simulator *Sim* is intuitively expected to translate a computational attack into a corresponding symbolic attack. Thus *Sim* essentially translates bitstrings to terms, and vice versa.

We proceed by defining the hybrid execution of a probabilistic CoSP protocol. We call this execution hybrid because it is a mixture of the symbolic and the computational execution. Concretely, we define a hybrid protocol machine Π^C that is associated to Π_p and interfaces *Sim* with a probabilistic CoSP protocol Π_p.

Definition 13 (Hybrid execution) *Let Π_p be a probabilistic CoSP protocol, and let Sim be a simulator. We define a probability distribution $H\text{-}Trace_{\mathbf{M},\Pi_p,Sim}(k)$ on (finite) lists of tuples (S_i, ν_i, f_i) called the* full hybrid trace *according to the following probabilistic algorithm Π^C, run on input 1^k, that interacts with Sim. (Π^C is called the hybrid protocol machine associated with Π_p and internally runs a symbolic simulation of Π_p as follows:)*

- Start*: $S_1 := S := \varnothing$, $\nu_1 := \nu$ is the root of Π_p, and $f_1 := f$ are totally undefined partial functions mapping node identifiers to $\mathbf{T}$. Run Π_p on ν.*
- Transition*: For $i = 1, \dots$ do the following:*
  - *Let $\underline{\tilde{\nu}}$ be the node identifiers in the label of ν. Let $\tilde{t}_k := f(\tilde{\nu}_k)$.*
  - *Proceed depending on the type of ν:*
    - *If ν is a constructor or destructor node with constructor, destructor, or nonce F, then let $m := F(\underline{\tilde{t}})$. If $m \neq \bot$, let ν' be the yes-successor of ν and let $f' := f(\nu := m)$. If $m = \bot$, let ν' be the no-successor of ν and let $f' := f$.*

  - ∗ *If ν is an output node, send $\underline{\tilde{t}}$ to Sim (but without handing over control to Sim). Let ν' be the unique successor of ν. Set $\nu := \nu'$.*
  - ∗ *If ν is an input node, hand control to Sim, and wait to receive $m \in \mathbf{T}$ from Sim. Let $f' := f(\nu := m)$, and let ν' be the unique successor of ν. Set $f := f'$ and $\nu := \nu'$.*
  - ∗ *If ν is a control node labeled with out-metadata l, send l to Sim, hand control to Sim, and wait to receive a bitstring l' from Sim. Let ν' be the successor of ν along the edge labeled l' (or the lexicographically smallest edge if there is no edge with label l'). Let $\nu := \nu'$.*
  - ∗ *If ν is a nondeterministic node, sample ν' according to the probability distribution specified in ν. Let $\nu := \nu'$.*
- *Send $(info, \nu, t)$ to Sim. When receiving an answer $(proceed)$ from Sim, continue.*
- *If Sim has terminated, stop. Otherwise let $(S_i, \nu_i, f_i) := (S, \nu, f)$.*

We write $Sim + \Pi^C$ to denote the execution of *Sim* and Π^C. We denote the probability on node traces of this execution by $H\text{-}Nodes_{\mathbf{M},\Pi_p,Sim}(k)$.

We proceed by defining properties of a simulator, such as adhering to a Dolev-Yao style deduction relation. Later we will show that simulators that satisfy these properties entail computational soundness results. Treating these properties separately instead of immediately conjoining them into a general soundness criterion allows us to more careful identify where these individual properties are exploited in computational soundness proofs.

The first property – Dolev-Yao-style – captures that *Sim* adheres to the deduction relation $\vdash$ in Definition 7 for input/output nodes. More precisely, the terms that *Sim* sends *to* the CoSP protocol have to be derivable from *Sim*'s symbolic view so far.

Definition 14 (Dolev-Yao style simulator) *A simulator Sim is* Dolev-Yao style *(short:* DY*) for $\mathbf{M}$ and Π_p, if with overwhelming probability the following holds:*

Whenever Sim does not fail, let m_ℓ be the ℓ-th term sent (during processing of one of Π^C's input nodes) from Sim to Π^C in that execution. Let T_ℓ the set of all terms that Sim has received from Π^C (during processing of output nodes) prior to sending m_ℓ. Then we have $T_\ell \vdash m_\ell$.

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 $\overset{c}{\approx}$ to denote computational indistinguishability.

Definition 15 (Indistinguishable simulator) *A simulator Sim is* indistinguishable *for $\mathbf{M}$, Π_p, an implementation A, an adversary E, and a polynomial p, if*

$$Nodes^p_{\mathbf{M},A,\Pi_p,E}(k) \overset{c}{\approx} H\text{-}Nodes_{\mathbf{M},\Pi_p,Sim}(k),$$

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

We define the following abbreviation.

Definition 16 (Good simulator) *A simulator is* good *for $\mathbf{M}$, Π_p, A, E, and p if it is Dolev-Yao style for $\mathbf{M}$, and Π_p, and indistinguishable for $\mathbf{M}$, Π_p, A, E, and p.*

We can now formally state and prove the main result of this section: the existence of a good simulator implies computational soundness.

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

Proof. We have to show that for every probabilistic CoSP protocol Π_p, we have that (Π_p, A) computationally satisfies $\mathcal{P}$ whenever Π_s symbolically satisfies a property $\mathcal{P}$ (where Π_s is the corresponding CoSP protocol of Π_p). Thus, for every E and p, $\mathit{Nodes}^p_{\mathbf{M},A,\Pi_p,E}(k)$ has to be contained in $\mathcal{P}$ with overwhelming probability. Fix Π_p, E, and p, and let Sim be a good simulator for $\mathbf{M}$, Π_p, A, E, and p. Let $A_{\mathcal{P}}$ denote a polynomial-time algorithm that decides property $\mathcal{P}$.

We first show a lemma on the hybrid node traces and then proceed with the overall proof.

Lemma 2 *Consider a hybrid execution of* $\mathit{Sim}+\Pi^C$ *in which* Sim *is DY, i.e., we have* $\{t_1, \dots, t_\ell\} \vdash m_\ell$ *for all* t_i *and* m_ℓ *as in Definition 14 and all* ℓ.

Let tr *be the full hybrid trace of that execution. Then* tr *is a full symbolic trace of* Π_s.

Proof. (of Lemma 2) We show that tr fulfills the conditions on full traces of Definition 7. This is clear for constructor, destructor, and control nodes, since the processing of these nodes in the hybrid setting of Definition 13 matches the one in the symbolic setting of Definition 7.

Input/output nodes in tr consist of a term $t \in \mathbf{T}$ sent from Π^C to Sim, or a term t' sent from Sim to Π^C. By the DY property of Sim, we know that $S \vdash t'$, where S denotes all terms (including t) sent from Π^C to Sim so far. Hence, the node satisfies the requirement for input/output nodes from Definition 7. This completes the proof of the lemma. □

Lemma 2 immediately entails that the probability is overwhelming that $\mathit{H\text{-}Nodes}_{\mathbf{M},\Pi_p,\mathit{Sim}}(k)$ is a symbolic node trace of Π_s, and hence that $\mathit{H\text{-}Nodes}_{\mathbf{M},\Pi_p,\mathit{Sim}} \in \mathcal{P}$. Since $A_{\mathcal{P}}$ decides $\mathcal{P}$, this means that

$$\mathsf{Pr}\left[A_{\mathcal{P}}(\mathit{H\text{-}Nodes}_{\mathbf{M},\Pi_p,\mathit{Sim}}(k)) = 1\right] \text{ is overwhelming.} \tag{1}$$

By Sim's indistinguishability property, we know that

$$\mathit{Nodes}^p_{\mathbf{M},A,\Pi_p,E}(k) \overset{c}{\approx} \mathit{H\text{-}Nodes}_{\mathbf{M},\Pi_p,\mathit{Sim}}(k).$$

Since $A_{\mathcal{P}}$ is polynomial-time in its input, and $\mathit{Nodes}^p_{\mathbf{M},A,\Pi_p,E}(k)$ is polynomially-sized in k by construction, this implies that

$$\mathsf{Pr}\left[A_{\mathcal{P}}(\mathit{Nodes}^p_{\mathbf{M},A,\Pi_p,E}(k)) = 1\right] \quad \text{is overwhelming,}$$

and hence that $\mathit{Nodes}^p_{\mathbf{M},A,\Pi_p,E}(k) \in \mathcal{P}$ with overwhelming probability. This concludes the proof of Theorem 1. □

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 $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$:

- Constructors and nonces: Let $\mathbf{C} := \{E/3, \mathit{ek}/1, \mathit{dk}/1, \mathit{sig}/3, \mathit{vk}/1, \mathit{sk}/1, \mathit{pair}/2, \mathit{string_0}/1, \mathit{string_1}/1, \mathit{empty}/0, \mathit{garbageSig}/2, \mathit{garbage}/1, \mathit{garbageE}/2\}$ and $\mathbf{N} := \mathbf{N}_P \cup \mathbf{N}_E$. Here $\mathbf{N}_P$ and $\mathbf{N}_E$ are countably infinite sets representing protocol and adversary nonces, respectively. Intuitively, encryption, decryption, verification, and signing keys are represented as $\mathit{ek}(r), \mathit{dk}(r), \mathit{vk}(r), \mathit{sk}(r)$ with a nonce r (the randomness used when generating the keys). $E(\mathit{ek}(r'), m, r)$ encrypts m using the encryption key $\mathit{ek}(r')$ and randomness r. $\mathit{sig}(\mathit{sk}(r'), m, r)$ is a signature of m using the signing key $\mathit{sk}(r')$ and randomness r. The constructors $\mathit{string_0}$, $\mathit{string_1}$, and empty are used to model arbitrary strings used as payload in a protocol (e.g., a bitstring 010 would be encoded as $\mathit{string_0}(\mathit{string_1}(\mathit{string_0}(\mathit{empty})))$). $\mathit{garbage}$, $\mathit{garbageE}$, and $\mathit{garbageSig}$ are constructors necessary to express certain invalid terms the adversary may send, these constructors are not used by the protocol.

- Message type: We define $\mathbf{T}$ as the set of all terms M matching the following grammar:

$$\begin{aligned}
M ::= {} & E(ek(N), M, N) \mid ek(N) \mid dk(N) \mid \\
& sig(sk(N), M, N) \mid vk(N) \mid sk(N) \mid \\
& pair(M, M) \mid S \mid \\
& garbage(N) \mid garbageE(M, N) \mid \\
& garbageSig(M, N) \\
S ::= {} & empty \mid string_0(S) \mid string_1(S)
\end{aligned}$$

  where the nonterminal N stands for nonces.
- Destructors: $\mathbf{D} := \{D/2, isenc/1, isek/1, ekof/1, verify/2, issig/1, isvk/1, vkof/2, fst/1, snd/1, unstring_0/1, unstring_1/1, equals/2\}$. The destructors *isek*, *isvk*, *isenc*, and *issig* realize predicates to test whether a term is an encryption key, verification key, ciphertext, or signature, respectively. *ekof* extracts the encryption key from a ciphertext, *vkof* extracts the verification key from a signature. $D(dk(r), c)$ decrypts the ciphertext c. $verify(vk(r), s)$ verifies the signature s with respect to the verification key $vk(r)$ and returns the signed message if successful. The destructors *fst* and *snd* are used to destruct pairs, and the destructors $unstring_0$ and $unstring_1$ allow to parse payload-strings.
  The behavior of the destructors is given by the following rules; an application matching none of these rules evaluates to $\bot$:

$$\begin{aligned}
D(dk(t_1), E(ek(t_1), m, t_2)) &= m \\
isenc(E(ek(t_1), t_2, t_3)) &= E(ek(t_1), t_2, t_3) \\
isenc(garbageE(t_1, t_2)) &= garbageE(t_1, t_2) \\
isek(ek(t)) &= ek(t) \\
ekof(E(ek(t_1), m, t_2)) &= ek(t_1) \\
ekof(garbageE(t_1, t_2)) &= t_1 \\[6pt]
verify(vk(t_1), sig(sk(t_1), t_2, t_3)) &= t_2 \\
issig(sig(sk(t_1), t_2, t_3)) &= sig(sk(t_1), t_2, t_3) \\
issig(garbageSig(t_1, t_2)) &= garbageSig(t_1, t_2) \\
isvk(vk(t_1)) &= vk(t_1) \\
vkof(sig(sk(t_1), t_2, t_3)) &= vk(t_1) \\
vkof(garbageSig(t_1, t_2)) &= t_1 \\[6pt]
fst(pair(x, y)) &= x \\
snd(pair(x, y)) &= y \\
unstring_0(string_0(s)) &= s \\
unstring_1(string_1(s)) &= s \\
equals(t_1, t_1) &= t_1
\end{aligned}$$

- Deduction relation: $\vdash$ is the smallest relation such that $m \in S \Rightarrow S \vdash m$, $N \in \mathbf{N}_E \Rightarrow S \vdash N$, and such that for any constructor or destructor $f \in \mathbf{C} \cup \mathbf{D}$ and for any $t_1, \dots, t_n \in \mathbf{T}$ with $S \vdash \underline{t}$ and $\bot \neq f(\underline{t}) \in \mathbf{T}$ we have $S \vdash f(\underline{t})$.

The computational implementation. Obtaining a computational soundness result for the symbolic model $\mathbf{M}$ requires its implementation to use an IND-CCA2 secure encryption scheme and a strongly existentially unforgeable signature scheme. More precisely, we require that (A_{ek}, A_{dk}), A_E, and A_D form the key generation, encryption and decryption algorithm of an IND-CCA2-secure scheme; and that (A_{vk}, A_{sk}), A_{sig}, and A_{verify} form the key generation, signing, and verification algorithm of a strongly existentially unforgeable signature scheme. Let $A_{isenc}(m) = m$ iff m is a ciphertext. (Only a syntactic check is performed; it is not necessary to check whether m was correctly generated.) A_{issig}, A_{isek}, and A_{isvk} are defined analogously. A_{ekof} extracts the encryption key from a ciphertext, i.e., we assume that ciphertexts are tagged with their public key. Similarly A_{vkof} extracts the verification key from a signature, and A_{verify} can be used to extract the signed message from a signature, i.e., we assume that signatures are tagged with their public key and the signed message. Nonces are implemented as (suitably tagged) random k-bit strings. A_{pair}, A_{fst}, and A_{snd} construct and destruct pairs. We require that the implementation of the constructors are length regular, i.e., the length of the result of applying a constructor depends only on the lengths of the arguments. No restrictions are put on $A_{garbage}$, $A_{garbageE}$, and $A_{garbageSig}$ as these are never actually used by the protocol.

The exact requirements are as follows:

Implementation conditions. We require that the implementation A of the symbolic model $\mathbf{M}$ has the following properties:

1. A is an implementation of $\mathbf{M}$ in the sense of Definition 8 (in particular, all functions A_f ($f \in \mathbf{C} \cup \mathbf{D}$) are polynomial-time computable).
2. There are disjoint and efficiently recognizable sets of bitstrings representing the types nonces, ciphertexts, encryption keys, decryption keys, signatures, verification keys, signing keys, pairs, and payload-strings. The set of all bitstrings of type nonce we denote Nonces_k.[4] (Here and in the following, k denotes the security parameter.)
3. The functions A_E, A_{ek}, A_{dk}, A_{sig}, A_{vk}, A_{sk}, A_{pair}, A_{string_0}, and A_{string_1} are length-regular. We call an n-ary function f length regular if $|m_i| = |m'_i|$ for $i = 1, \dots, n$ implies $|f(\underline{m})| = |f(\underline{m'})|$. All $m \in \mathrm{Nonces}_k$ have the same length.
4. A_N for $N \in \mathbf{N}$ returns a uniformly random $r \in \mathrm{Nonces}_k$.
5. Every image of A_E is of type ciphertext, every image of A_{ek} and A_{ekof} is of type encryption key, every image of A_{dk} is of type decryption key, every image of A_{sig} is of type signature, every image of A_{vk} and A_{vkof} is of type verification key, every image of A_{empty}, A_{string_0}, and A_{string_1} is of type payload-string.
6. For all $m_1, m_2 \in \{0,1\}^*$ we have $A_{fst}(A_{pair}(m_1, m_2)) = m_1$ and $A_{snd}(A_{pair}(m_1, m_2)) = m_2$. Every m of type pair is in the range of A_{pair}. If m is not of type pair, $A_{fst}(m) = A_{snd}(m) = \bot$.
7. For all m of type payload-string we have that $A_{unstring_i}(A_{string_i}(m)) = m$ and $A_{unstring_i}(A_{string_j}(m)) = \bot$ for $i, j \in \{0,1\}$, $i \neq j$. For $m = \mathit{empty}$ or m not of type payload-string, $A_{unstring_0}(m) = A_{unstring_1}(m) = \bot$. Every m of type payload-string is of the form $m = A_{unstring_0}(m')$ or $m = A_{unstring_1}(m')$ or $m = \mathit{empty}$ for some m' of type payload-string.
8. $A_{ekof}(A_E(p, x, y)) = p$ for all p of type encryption key, $x \in \{0,1\}^*$, $y \in \mathrm{Nonces}_k$. $A_{ekof}(e) \neq \bot$ for any e of type ciphertext and $A_{ekof}(e) = \bot$ for any e that is not of type ciphertext.
9. $A_{vkof}(A_{sig}(A_{sk}(x), y, z)) = A_{vk}(x)$ for all $y \in \{0,1\}^*$, $x, z \in \mathrm{Nonces}_k$. $A_{vkof}(e) \neq \bot$ for any e of type signature and $A_{vkof}(e) = \bot$ for any e that is not of type signature.
10. $A_E(p, m, y) = \bot$ if p is not of type encryption key.
11. $A_D(A_{dk}(r), m) = \bot$ if $r \in \mathrm{Nonces}_k$ and $A_{ekof}(m) \neq A_{ek}(r)$. (This implies that the encryption key is uniquely determined by the decryption key.)
12. $A_D(A_{dk}(r), A_E(A_{ek}(r), m, r')) = m$ for all $r, r' \in \mathrm{Nonces}_k$.
13. $A_{verify}(A_{vk}(r), A_{sig}(A_{sk}(r), m, r')) = m$ for all $r, r' \in \mathrm{Nonces}_k$.
14. For all $p, s \in \{0,1\}^*$ we have that $A_{verify}(p, s) \neq \bot$ implies $A_{vkof}(s) = p$.
15. $A_{isek}(x) = x$ for any x of type encryption key. $A_{isek}(x) = \bot$ for any x not of type encryption key.
16. $A_{isvk}(x) = x$ for any x of type verification key. $A_{isvk}(x) = \bot$ for any x not of type verification key.
17. $A_{isenc}(x) = x$ for any x of type ciphertext. $A_{isenc}(x) = \bot$ for any x not of type ciphertext.
18. $A_{issig}(x) = x$ for any x of type signature. $A_{issig}(x) = \bot$ for any x not of type signature.

[4]This would typically be the set of all k-bit strings with a tag denoting nonces.

19. We define an encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as follows: KeyGen picks a random $r \leftarrow \mathrm{Nonces}_k$ and returns $(A_{ek}(r), A_{dk}(r))$. $\mathsf{Enc}(p, m)$ picks a random $r \leftarrow \mathrm{Nonces}_k$ and returns $A_E(p, m, r)$. $\mathsf{Dec}(k, c)$ returns $A_D(k, c)$. We require that then $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-CCA secure.
20. We define a signature scheme $(\mathsf{SKeyGen}, \mathsf{Sig}, \mathsf{Verify})$ as follows: $\mathsf{SKeyGen}$ picks a random $r \leftarrow \mathrm{Nonces}_k$ and returns $(A_{vk}(r), A_{sk}(r))$. $\mathsf{Sig}(p, m)$ picks a random $r \leftarrow \mathrm{Nonces}_k$ and returns $A_{sig}(p, m, r)$. $\mathsf{Verify}(p, s, m)$ returns 1 iff $A_{verify}(p, s) = m$. We require that then $(\mathsf{SKeyGen}, \mathsf{Sig}, \mathsf{Verify})$ is strongly existentially unforgeable.

Note that any IND-CCA secure encryption scheme and strongly existentially unforgeable signature scheme can be transformed into an implementation satisfying the above conditions by suitably tagging and padding the ciphertexts, signatures, and keys.

Key-safe protocols. The computational soundness result we derive in this section requires that the CoSP protocol satisfies certain constraints. In a nutshell, these constraints require that encryption, signing, and key generation always use fresh randomness, that decryption only uses honestly generated decryption keys, that only honestly generated keys are used for signing, and that the protocol does not produce garbage terms. Decryption and signing keys may not be sent around. (In particular, this avoids the so-called key-cycle and key-commitment problems.) We call protocols satisfying these conditions *key-safe.*

The exact requirements are the following:

Protocol conditions. A CoSP protocol is *key-safe* if it satisfies the following conditions:

1. The argument of every *ek*-, *dk*-, *vk*-, and *sk*-constructor node and the third argument of every E- and *sig*-constructor node is an N-constructor node with $N \in \mathbf{N}_P$. (Here and in the following, we call the nodes referenced by a protocol node its arguments.)
2. Every constructor node that is the argument of an *ek*-constructor node or of a *dk*-constructor node on some path p occurs only as argument to *ek*- and *dk*-constructor nodes on that path p.
3. Every constructor node that is the argument of a *vk*-constructor node or of an *sk*-constructor node on some path p occurs only as argument to *vk*- and *sk*-constructor nodes on that path p.
4. Every constructor node that is the third argument of an E-constructor node or of a *sig*-constructor node on some path p occurs exactly once as an argument in that path p.
5. Every *dk*-constructor node occurs only as the first argument of a D-destructor node.
6. The first argument of a D-destructor node is a *dk*-constructor node.
7. Every *sk*-constructor node occurs only as the first argument of a *sig*-constructor node.
8. The first argument of a *sig*-constructor node is an *sk*-constructor node.
9. There are no constructor nodes with the constructors *garbage*, *garbageE*, *garbageSig*, or $N \in \mathbf{N}_E$.

Construction of the simulator. In the following, we define distinct nonces $N^m \in \mathbf{N}_E$ for each $m \in \{0,1\}^*$. In a hybrid execution, we call a term t *honestly generated* if it occurs as a subterm of a term sent by the protocol Π^C to the simulator before it has occurred as a subterm of a term sent by the simulator to the protocol Π^C.

For an adversary E and a polynomial p, we construct the simulator *Sim* as follows: In the first activation, it chooses $r_N \in \mathrm{Nonces}_k$ for every $N \in \mathbf{N}_P$. It maintains an integer *len*, initially 0. At any point in the execution, $\mathcal{N}$ denotes the set of all nonces $N \in \mathbf{N}_P$ that occurred in terms received from Π^C. *Sim* internally simulates the adversary E. When receiving a term $\tilde{t} \in \mathbf{T}$ from Π^C, it passes $\beta(\tilde{t})$ to E where the partial function $\beta : \mathbf{T} \to \{0,1\}^*$ is defined below. When E answers with $m \in \{0,1\}^*$, the simulator sends $\tau(m)$ to Π^C where the function $\tau : \{0,1\}^* \to \mathbf{T}$ is defined below. The bitstrings sent from the protocol at control nodes are passed through to E and vice versa. When the simulator receives (info, ν, t), the simulator increases *len* by $\ell(t) + 1$ where $\ell : \mathbf{T} \to \{0,1\}^*$ is defined below. If $\mathit{len} > p(k)$, the simulator terminates, otherwise it answers with $(\mathit{proceed})$.

Translation functions. The partial function $\beta : \mathbf{T} \to \{0,1\}^*$ is defined as follows (where the first matching rule is taken):

- $\beta(N) := r_N$ if $N \in \mathcal{N}$.
- $\beta(N^m) := m$.
- $\beta(E(ek(t_1), t_2, M)) := A_E(\beta(ek(t_1)), \beta(t_2), r_M)$ if $M \in \mathcal{N}$.
- $\beta(E(ek(M), t, N^m)) := m$ if $M \in \mathcal{N}$.
- $\beta(ek(N)) := A_{ek}(r_N)$ if $N \in \mathcal{N}$.
- $\beta(ek(N^m)) := m$.
- $\beta(dk(N)) := A_{dk}(r_N)$ if $N \in \mathcal{N}$.
- $\beta(sig(sk(N), t, M)) := A_{sig}(A_{sk}(r_N), \beta(t), r_M)$ if $N, M \in \mathcal{N}$.
- $\beta(sig(sk(M), t, N^s)) := s$.
- $\beta(vk(N)) := A_{vk}(r_N)$ if $N \in \mathcal{N}$.
- $\beta(vk(N^m)) := m$.
- $\beta(sk(N)) := A_{sk}(r_N)$ if $N \in \mathcal{N}$.
- $\beta(pair(t_1, t_2)) := A_{pair}(\beta(t_1), \beta(t_2))$.
- $\beta(string_0(t)) := A_{string_0}(\beta(t))$.
- $\beta(string_1(t)) := A_{string_1}(\beta(t))$.
- $\beta(empty) := A_{empty}()$.
- $\beta(garbage(N^c)) := c$.
- $\beta(garbageE(t, N^c)) := c$.
- $\beta(garbageSig(t_1, t_2, N^s)) := s$.
- $\beta(t) := \bot$ in all other cases.

The total function $\tau : \{0,1\}^* \to \mathbf{T}$ is defined as follows (where the first matching rule is taken):

- $\tau(r) := N$ if $r = r_N$ for some $N \in \mathcal{N}$.
- $\tau(r) := N^r$ if r is of type nonce.
- $\tau(c) := E(ek(M), t, N)$ if c has earlier been output by $\beta(E(ek(M), t, N))$ for some $N, M \in \mathcal{N}$.
- $\tau(c) := E(ek(N), \tau(m), N^c)$ if c is of type ciphertext and $\tau(A_{ekof}(c)) = ek(N)$ for some $N \in \mathcal{N}$ and $m := A_D(A_{dk}(r_N), c) \neq \bot$.
- $\tau(e) := ek(N)$ if e has earlier been output by $\beta(ek(N))$ for some $N \in \mathcal{N}$.
- $\tau(e) := ek(N^e)$ if e is of type encryption key.
- $\tau(s) := sig(sk(M), t, N)$ if s has earlier been output by $\beta(sig(sk(M), t, N))$ for some $M, N \in \mathcal{N}$.
- $\tau(s) := sig(sk(M), \tau(m), N^s)$ if s is of type signature and $\tau(A_{vkof}(s)) = vk(M)$ for some $M \in \mathbf{N}$ and $m := A_{verify}(A_{vkof}(s), s) \neq \bot$.
- $\tau(e) := vk(N)$ if e has earlier been output by $\beta(vk(N))$ for some $N \in \mathcal{N}$.
- $\tau(e) := vk(N^e)$ if e is of type verification key.
- $\tau(m) := pair(\tau(A_{fst}(m)), \tau(A_{snd}(m)))$ if m of type pair.
- $\tau(m) := string_0(m')$ if m is of type payload-string and $m' := A_{unstring_0}(m) \neq \bot$.
- $\tau(m) := string_1(m')$ if m is of type payload-string and $m' := A_{unstring_1}(m) \neq \bot$.
- $\tau(m) := empty$ if m is of type payload-string and $m = A_{empty}()$.
- $\tau(c) := garbageE(\tau(A_{ekof}(e)), N^c)$ if c is of type ciphertext.
- $\tau(s) := garbageSig(\tau(A_{vkof}(s)), N^s)$ if s is of type signature.
- $\tau(m) := garbage(N^m)$ otherwise.

The function $\ell : \mathbf{T} \to \{0,1\}^*$ is defined as $\ell(t) := |\beta(t)|$. Note that $\ell(t)$ does not depend on the actual values of r_N because of the length-regularity of A_E, A_{ek}, A_{dk}, A_{sig}, A_{vk}, A_{sk}, A_{pair}, A_{string_0}, and A_{string_1}. Hence $\ell(t)$ can be computed without accessing r_N.

The faking simulator. The simulator Sim' is defined exactly like Sim, except that it makes use of an encryption and a signing oracle (these oracles also supply keypairs (ek_N, dk_N), resp. (vk_N, sk_N)). When computing $\beta(ek(N))$ or $\beta(dk(N))$ with $N \in \mathcal{N}$, it instructs the encryption oracle to generate a new encryption/decryption key pair (ek_N, dk_N) (unless (ek_N, dk_N) are already defined) and retrieves ek_N or dk_N from the oracle, respectively. When computing $\beta(E(ek(N), t, M))$ with $N, M \in \mathcal{N}$, instead of computing $A_E(A_{ek}(r_N), \beta(t), r_M)$, Sim' requests the encryption $\mathsf{Enc}(ek_N, \beta(t))$ of $\beta(t)$ from the encryption oracle (that is, Sim' has to compute $\beta(t)$ but does not need to retrieve ek_N). However, the resulting ciphertext is stored and when later computing $\beta(E(ek(N), t, M))$ with the same arguments, the stored ciphertext is reused. When computing $\tau(c)$, instead of computing $A_D(A_{dk}(r_N), c)$, Sim' invokes the encryption oracle to decrypt c using the decryption key dk_N (again, Sim' does not need to retrieve dk_N).

Similarly, to compute $\beta(vk(N))$ or $\beta(sk(N))$, Sim' retrieves keys vk_N or sk_N from the signing oracle. To compute $\beta(sig(sk(N), t, M))$, Sim' invokes the signing oracle with message $\beta(t)$ to get a signature under the signing key sk_N. However, the resulting signature is stored and when later computing $\beta(sig(sk(N), t, M))$ with the same arguments, the stored ciphertext is reused. Sim' does not invoke the signing oracle for verifying signatures, instead Sim' executes A_{verify} directly (as does Sim).

The simulator Sim_f is defined like Sim', except that when computing $\beta(E(ek(N), t, M))$, instead of invoking the encryption oracle with plaintext $\beta(t)$, it invokes it with plaintext $0^{\ell(t)}$.

Properties of the simulator. We derive several properties of the simulators Sim and Sim_f that will finally allow to show that Sim is a good simulator for key-safe protocols. In the following, let Π' always denote an key-safe probabilistic CoSP protocol. By construction, Sim, Sim' and Sim_f run in polynomial time.

Lemma 3 *The full traces $H\text{-}Trace_{\mathbf{M},\Pi_p,Sim}$ and $H\text{-}Trace_{\mathbf{M},\Pi_p,Sim_f}$ are computationally indistinguishable.*

Proof. Note that the difference between Sim and Sim' is that the randomness for the key generation, the encryption, and the signing is chosen by the algorithms KeyGen, $\mathsf{SKeyGen}$, Enc, and Sig in Sim', while Sim uses nonces r_N instead. However, from protocol conditions 1, 2, 3, 4, it follows that Sim never uses a given randomness r_N twice. Hence the full traces $H\text{-}Trace_{\mathbf{M},\Pi_p,Sim}$ and $H\text{-}Trace_{\mathbf{M},\Pi_p,Sim'}$ are indistinguishable.

Note that Sim' invokes $\mathsf{Dec}(dk_N, c)$ only for values c that have not been output by $\beta(E(ek(M), t, N))$. Thus $\mathsf{Dec}(dk_N, c)$ is invoked only for values c that have not been output by $\mathsf{Enc}(ek_N, \cdot)$. Since $|\beta(t)| = |0^{\ell(t)}|$ by definition of ℓ, the IND-CCA property of $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ (implementation condition 19), the strong existential unforgeability of $(\mathsf{SKeyGen}, \mathsf{Sig}, \mathsf{Verify})$ (implementation condition 20), as well as protocol conditions 5 and 7 imply that the full traces $H\text{-}Trace_{\mathbf{M},\Pi_p,Sim'}$ and $H\text{-}Trace_{\mathbf{M},\Pi_p,Sim_f}$ are indistinguishable. Using the transitivity of computational indistinguishability, the lemma follows. □

Lemma 4 *Sim is indistinguishable for $\mathbf{M}$, Π, A, and for every polynomial p.*

Proof. We will first show that when fixing the randomness of the adversary and the protocol, the node trace $Nodes^p_{\mathbf{M},A,\Pi_p,E}$ in the computational execution and the node trace $H\text{-}Nodes_{\mathbf{M},\Pi_p,Sim}$ in the hybrid execution are equal. Hence, fix the variables r_N for all $N \in \mathbf{N}_P$, fix a random tape for the adversary, and for each node ν fix a choice e_ν of an outgoing edge.

Assume that r_N, $A_{ek}(r_N)$, $A_{dk}(r_N)$, $A_{vk}(N)$, and $A_{ek}(N)$, are pairwise distinct for all $N \in \mathcal{N}^*$. (This happens with overwhelming probability when the randomness is uniformly chosen.)

In the following, we designate the values f_i and ν_i in the computational execution by f'_i and ν'_i, and in the hybrid execution by f^C_i and ν^C_i. Let s'_i denote the state of the adversary E in the computational model, and s^C_i the state of the simulated adversary in the hybrid model.

Claim 1: In the hybrid execution, for any $b \in \{0,1\}^*$, $\beta(\tau(b)) = b$.

This claim follows by induction over the length of b and by distinguishing the cases in the definition of τ.

Claim 2: In the hybrid execution, for any term t stored at a node ν, $\beta(t) \neq \bot$.

By induction on the structure of t.

Claim 3: For all terms t that occur in the hybrid execution, $\tau(\beta(t)) = t$.

By induction on the structure of t and using the assumption that r_N, $A_{ek}(r_N)$, $A_{dk}(r_N)$, $A_{vk}(N)$, and $A_{ek}(N)$, are pairwise distinct for all $N \in \mathcal{N}^*$.

Claim 4: In the hybrid execution, at any constructor or destructor node $\nu = \nu_i$ with constructor, destructor, or nonce F and arguments $\bar{\nu}_1, \dots, \bar{\nu}_n$ the following holds: Let t_i be the term stored at node $\bar{\nu}_i$ (i.e., $t_j = f'_i(\bar{\nu}_j)$). Then $\beta(F(\underline{t})) = A_F(\beta(t_1), \dots, \beta(t_n))$. Here the left hand side is defined iff the right hand side is. (And we write $F(\underline{t}) := \bot$ if F is a constructor but $F(\underline{t}) \notin \mathbf{T}$.)

We show Claim 4. We distinguish the following cases:

Case 1: *"$F = N \in \mathbf{N}_P$".*
The claim holds because $\beta(N) = r_N = A_N$.

Case 2: *"$F = ek$".*
Note that by protocol condition 1, we have $t_1 \in \mathbf{N}_P$. Then $\beta(ek(t_1)) = A_{ek}(r_{t_1}) = A_{ek}(\beta(t_1))$.

Case 3: *"$F \in \{dk, vk, sk\}$".*
Analogous to the case $F = ek$.

Case 4: *"$F \in \{pair, fst, snd, string_0, string_0, unstring_0, unstring_1, empty\}$".*
Claim 4 follows directly from the definition of β.

Case 5: *"$F = isek$".*
If $t_1 = ek(t'_1)$, we have that $t'_1 = N \in \mathcal{N}$ or $t'_1 = N^m$ where m is of type ciphertext (as other subterms of the form $ek(\cdot)$ are neither produced by the protocol nor by τ). In both cases, $\beta(ek(t'_1))$ is of type encryption key. Hence $\beta(isek(t_1)) = \beta(ek(t'_1)) = A_{isek}(\beta(ek(t'_1))) = A_{isek}(\beta(t_1))$. If t_1 is not of the form $ek(\cdot)$, then $\beta(t_1)$ is not of type public key (this uses that τ only uses N^m with m of type public key inside a term $ek(N^m)$). Hence $\beta(isek(t_1)) = \bot = A_{isek}(\beta(t_1))$.

Case 6: *"$F \in \{isvk, isenc, issig\}$".*
Similar to the case $F = isek$.

Case 7: *"$F = ekof$".*
If $t_1 = E(ek(u_1), u_2, M)$ with $M \in \mathcal{N}$, we have that $\beta(t_1) = A_E(\beta(ek(u_1)), \beta(u_2), r_M)$. By implementation condition 8, $A_{ekof}(\beta(t_1)) = \beta(ek(u_1))$. Furthermore, $ekof(t_1) = ek(u_1)$, hence $A_{ekof}(\beta(t_1)) = \beta(ekof(t_1))$. If $t_1 = E(ek(u_1), u_2, N^m)$, by protocol condition 9, t_1 was not honestly generated. Hence, by definition of τ, m is of type ciphertext, and $ek(u_1) = \tau(A_{ekof}(m))$. Thus with Claim 1, $\beta(ek(u_1)) = A_{ekof}(m)$. Furthermore, we have $\beta(t_1) = m$ by definition of β and thus $A_{ekof}(\beta(t_1)) = \beta(ek(u_1)) = \beta(ekof(t_1))$. If $t_1 = garbageE(u_1, u_2)$, the proof is analogous. In all other cases for t_1, $\beta(t_1)$ is not of type ciphertext, hence $A_{ekof}(\beta(t_1)) = \bot$ by implementation condition 8. Furthermore $ekof(t_1) = \bot$. Thus $\beta(ekof(t_1)) = \bot = A_{ekof}(\beta(t_1))$.

Case 8: *"$F = vkof$".*
If $t_1 = sig(sk(N), u_1, M)$ with $N, M \in \mathcal{N}$, we have that $\beta(t_1) = A_{sig}(A_{sk}(r_N), \beta(u_2), r_M)$. By implementation condition 9, $A_{ekof}(\beta(t_1)) = A_{vk}(r_N)$. Furthermore, $vkof(t_1) = vk(N)$, hence $A_{vkof}(\beta(t_1)) = A_{vk}(r_N) = \beta(vk(N)) = \beta(vkof(t_1))$. All other cases for t_1 are handled like in the case of $F = ekof$.

Case 9: *"$F = E$".*
By protocol condition 1, $t_3 =: N \in \mathcal{N}$. If $t_1 = ek(u_1)$ we have $\beta(E(t_1, t_2, t_3)) = A_E(\beta(t_1), \beta(t_2), r_N)$ by definition of β. Since $\beta(N) = r_N$, we have $\beta(E(t_1, t_2, t_3)) = A_E(\beta(t_1), \beta(t_2), \beta(t_3))$. If t_1 is not of the form $ek(u_1)$, then $E(t_1, t_2, t_3) = \bot$ and by definition of β, $\beta(t_1)$ is not of type encryption key and hence by implementation condition 10, $\beta(E(t_1, t_2, t_3)) = A_{ek}(\beta(t_1), \dots) = \bot = \beta(E(t_1, t_2, t_3))$.

Case 10: *"$F = D$".*
By protocol condition 6, $t_1 = sk(N)$ with $N \in \mathcal{N}$. We distinguish the following cases for t_2:

Case 10.1: *"$t_2 = E(ek(N), t_2, M)$ with $M \in \mathcal{N}$".*
Then $A_D(\beta(t_1), \beta(t_2)) = A_D(A_{sk}(r_N), A_E(A_{ek}(N), \beta(t_2), r_M)) = \beta(t_2)$ by implementation condition 12. Furthermore $\beta(D(t_1, t_2)) = \beta(t_2)$ by definition of D.

Case 10.2: *"$t_2 = E(ek(N), t_2, N^c)$".*
Then t_2 was produced by τ and hence c is of type ciphertext and $\tau(A_D(A_{dk}(r_N), c)) = t_2$. Then by Claim 1, $A_D(A_{dk}(r_N), c) = \beta(t_2)$ and hence $A_D(\beta(t_1), \beta(t_2)) = A_D(A_{dk}(r_N), c) = \beta(t_2) = \beta(D(t_1, t_2))$.

Case 10.3: *"$t_2 = E(u_1, u_2, u_3)$ with $u_1 \neq ek(N)$".*
As shown above (case $F = ekof$), $A_{ekof}(\beta(E(u_1, u_2, u_3)) = \beta(ekof(E(u_1, u_2, u_3)) = \beta(u_1)$. Moreover, from Claim 3, $A_{ekof}(\beta(E(u_1, u_2, u_3)) = \beta(u_1) \neq \beta(ek(N)) = A_{ek}(r_N)$. Thus by implementation condition 11, $A_D(\beta(t_1), \beta(t_2)) = A_D(A_{dk}(r_N), \beta(E(u_1, u_2, u_3))) = \bot$. Furthermore, $D(t_1, t_2) = \bot$ and thus $\beta(D(t_1, t_2)) = \bot$.

Case 10.4: *"$t_2 = garbageE(u_1, N^c)$".*
Assume that $m := A_D(\beta(t_1), \beta(t_2)) = A_D(A_{sk}(r_N), c) \neq \bot$. By implementation condition 11 this implies $A_{ekof}(c) = A_{ek}(r_N)$ and thus $\tau(A_{ekof}(c)) = \tau(A_{ek}(r_N)) = ek(N)$. By protocol condition 9, t_2 has been produced by τ, i.e., $t_2 = \tau(c)$. Hence c is of type ciphertext. Then, however, we would have $\tau(c) = E(ek(N), \tau(m), N^c) \neq t_2$. This is a contradiction to $t_2 = \tau(c)$, so the assumption that $A_D(\beta(t_1), \beta(t_2)) \neq \bot$ was false. So $A_D(\beta(t_1), \beta(t_2)) = \bot = \beta(\bot) = \beta(D(t_1, garbageE(u_1, N^c)))$.

Case 10.5: *"All other cases".*
Then $\beta(t_2)$ is not of type ciphertext. By implementation condition 8, $A_{ekof}(\beta(t_2)) = \bot$. Hence $A_{ekof}(\beta(t_2)) \neq A_{ek}(r_N)$ and by implementation condition 11, $A_D(\beta(t_1), \beta(t_2)) = A_D(A_{dk}(r_N), \beta(t_2)) = \bot = \beta(D(t_1, t_2))$.

Case 11: *"$F = sig$".*
By protocol conditions 8 and 1 we have that $t_1 = sk(N)$ and $t_3 = M$ with $N, M \in \mathcal{N}$. Then $\beta(sig(\underline{t})) = A_{sig}(A_{sk}(r_N), \beta(t_3), r_M) = A_{sig}(\beta(sk(N)), \beta(t_2), \beta(M)) = A_{sig}(\beta(t_1), \beta(t_2), \beta(t_3))$.

Case 12: *"$F = verify$".*
We distinguish the following subcases:

Case 12.1: *"$t_1 = vk(N)$ and $t_2 = sig(sk(N), u_2, M)$ with $N, M \in \mathcal{N}$".*
Then $A_{verify}(\beta(t_1), \beta(t_2)) = A_{verify}(A_{vk}(r_N), A_{sig}(A_{sk}(r_N), \beta(u_2), r_M)) \overset{(*)}{=} \beta(u_2) = \beta(verify(\underline{t}))$ where $(*)$ uses implementation condition 13.

Case 12.2: *"$t_2 = sig(sk(N), u_2, M)$ and $t_1 \neq vk(N)$ with $N, M \in \mathcal{N}$".*
By Claim 3, $\beta(t_1) \neq \beta(vk(N))$ Furthermore $A_{verify}(\beta(vk(N)), \beta(t_2)) = A_{verify}(\beta(t_1), A_{sig}(A_{sk}(r_N), \beta(u_2), r_M)) \overset{(*)}{=} \beta(u_2) \neq \bot$. Hence with implementation condition 14, $A_{verify}(\beta(t_1), \beta(t_2)) = \bot = \beta(\bot) = verify(t_1, t_2)$.

Case 12.3: *"$t_1 = vk(N)$ and $t_2 = sig(sk(N), u_2, M^s)$".*
Then t_2 was produced by τ and hence s is of type signature with $\tau(A_{vkof}(s)) = vk(N)$ and $m := A_{verify}(A_{vkof}(s), s) \neq \bot$ and $u_2 = \tau(m)$. Hence with Claim 1 we have $m = \beta(\tau(m)) = \beta(u_2)$ and $\beta(t_1) = \beta(vk(N)) = \beta(\tau(A_{vkof}(s))) = A_{vkof}(s)$. Thus $A_{verify}(\beta(t_1), \beta(t_2)) = A_{verify}(A_{vkof}(s), s) = m = \beta(u_2)$. And $\beta(verify(t_1, t_2)) = \beta(verify(vk(N), sig(sk(N), u_2, M^s))) = \beta(u_2)$.

Case 12.4: *"$t_2 = sig(sk(N), u_2, M^s)$ and $t_1 \neq vk(N)$".*
As in the previous case, $A_{verify}(A_{vkof}(s), s) \neq \bot$ and $\beta(vk(N)) = A_{vkof}(s)$. Since $t_1 \neq vk(N)$, by Claim 3, $\beta(t_1) \neq \beta(vk(N)) = A_{vkof}(s)$. From implementation condition 14 and $A_{verify}(A_{vkof}(s), s) \neq \bot$, we have $A_{verify}(\beta(t_1), \beta(t_2)) = A_{verify}(\beta(t_1), s) = \bot = \beta(\bot) = \beta(verify(t_1, t_2))$.

Case 12.5: *"$t_2 = garbageSig(u_1, N^s)$".*
Then t_2 was produced by τ and hence s is of type signature and either $A_{verify}(A_{vkof}(s), s) = \bot$

or $\tau(A_{vkof}(s))$ is not of the form $vk(\dots)$. The latter case only occurs if $A_{vkof}(s) = \bot$ as otherwise $A_{vkof}(s)$ is of type verification key and hence $\tau(A_{vkof}(s)) = vk(\dots)$. Hence in both cases $A_{verify}(A_{vkof}(s), s) = \bot$. If $\beta(t_1) = A_{vkof}(s)$ then $A_{verify}(\beta(t_1), \beta(t_2)) = A_{verify}(A_{vkof}(s), s) = \bot = \beta(verify(t_1, t_2))$. If $\beta(t_1) \neq A_{vkof}(s)$ then by implementation condition 14, $A_{verify}(\beta(t_1), \beta(t_2)) = A_{verify}(\beta(t_1), s) = \bot$. Thus in both cases, with $verify(t_1, t_2) = \bot$ we have $A_{verify}(\beta(t_1), \beta(t_2)) = \bot = \beta(verify(t_1, t_2))$.

Case 12.6: *"All other cases".*
Then $\beta(t_2)$ is not of type signature, hence by implementation condition 9, $A_{vkof}(\beta(t_2)) = \bot$, hence $\beta(t_1) \neq A_{vkof}(\beta(t_2))$, and by implementation condition 14 we have $A_{verify}(\beta(t_1), \beta(t_2)) = \bot = \beta(verify(t_1, t_2))$.

Case 13: *"$F = equals$".*
If $t_1 = t_2$ we $\beta(equals(t_1, t_2)) = \beta(t_1) = A_{equals}(\beta(t_1), \beta(t_1)) = A_{equals}(\beta(t_1), \beta(t_2))$. If $t_1 \neq t_2$, by Claim 3, $\beta(t_1) \neq \beta(t_2)$ and hence $\beta(equals(t_1, t_2)) = \bot = A_{equals}(\beta(t_1), \beta(t_2))$.

Case 14: *"$F \in \{garbage, garbageE, garbageSig\} \cup \mathbf{N}_E$".*
By protocol condition 9, the constructors *garbage*, *garbageE*, *garbageSig*, and $N \in \mathbf{N}_E$ do not occur in the protocol.

Thus Claim 4 holds.

We will now show that for the random choices fixed above, $Nodes^p_{\mathbf{M},A,\Pi_p,E} = H\text{-}Nodes_{\mathbf{M},\Pi_p,Sim}$.

To prove this, we show the following invariant: $f'_i = \beta \circ f^C_i$ and $\nu'_i = \nu^C_i$ and $s_i = s'_i$ for all i.

We have $f'_0 = f^C_0 = \varnothing$ and $\nu'_0 = \nu^C_0$ is the root node, so the invariant is satisfied for $i = 0$. Assume that the invariant holds for some i. If ν'_i is a non-deterministic node, $\nu'_{i+1} = \nu^C_{i+1}$ is determined by $e_{\nu'_i} = e_{\nu^C_i}$. Since a non-deterministic node does not modify f and the adversary is not activated, $\tau \circ f'_{i+1} = f^C_{i+1}$ and $s_i = s'_i$. Hence the invariant holds for $i+1$ if ν'_i is a non-deterministic node.

If ν'_i is a constructor node with constructor C, we have that $f'_{i+1}(\nu'_i) = A_C(f'_i(\bar\nu_1), \dots, f'_i(\bar\nu_n)) = A_C(\beta(f^C_i(\bar\nu_1)), \dots, \beta(f^C_i(\bar\nu_n)))$ for some nodes $\bar\nu_s$ depending on the label of ν'_i. And $f^C_{i+1}(\nu'_i) = f^C_{i+1}(\nu^C_i) = C(f^C_i(\bar\nu_1), \dots, f^C_i(\bar\nu_n))$. From Claim 4 it follows that $\beta(f^C_{i+1}(\nu'_i)) = f'_{i+1}(\nu'_i)$ and hence $\beta \circ f^C_{i+1} = f'_{i+1}$. The successor node of a constructor node is unique, hence $\nu'_{i+1} = \nu^C_{i+1}$, and the adversary E is not invoked, hence $s'_{i+1} = s^C_{i+1}$. Hence the invariant holds for $i+1$ if ν'_i is a constructor node.

If ν'_i is a constructor node with destructor D, $\beta \circ f^C_{i+1} = f'_{i+1}$ and $s'_{i+1} = s^C_{i+1}$ are shown like in the case of constructor nodes. Furthermore, from Claim 4 it also follows that $A_C(f'_i(\bar\nu_1), \dots, f'_i(\bar\nu_n))$ is defined iff $C(f^C_i(\bar\nu_1), \dots, f^C_i(\bar\nu_n))$ is. Hence the same successor node is taken in both executions, so the invariant holds for $i+1$ if ν'_i is a destructor node.

In the case of a communication node, the adversary E in the computational execution gets a tuple $m' := (l, f'_i(\bar\nu_1), \dots, f'_i(\bar\nu_n))$ where l is the out-metadata of the node ν'_i and the nodes $\bar\nu_s$ depend on the label of ν'_i. In the hybrid execution, the simulator gets $(l, f^C_i(\bar\nu_1), \dots, f^C_i(\bar\nu_n))$ and sends $m^C := (l, \beta(f^C_i(\bar\nu_1)), \dots, \beta(f^C_i(\bar\nu_n)))$ to the simulated adversary E. By Claim 4 we then have $m' = m^C$, so the adversary gets the same input in both executions. Thus $s'_{i+1} = s^C_{i+1}$ and the adversary sends the same pair (l', b_1) in both executions. The successor nodes ν'_{i+1} and ν^C_{i+1} are determined by the in-metadata l', hence $\nu'_{i+1} = \nu^C_{i+1}$. Finally, we have $f'_{i+1}(\nu'_i) = b$ and $f^C_{i+1}(\nu'_i) = \tau(b)$ (because the simulator translates the bitstring b using τ before passing it to Π^C). Thus the invariant holds for $i+1$ in the case of a communication node.

From the invariant it follows, that the node trace is the same in both executions.

Since random choices with all r_N, $ek(r_N)$, and $dk(r_N)$ being pairwise distinct for $N \in \mathcal{N}^*$ occur with overwhelming probability, the node traces of the real and the hybrid execution are indistinguishable.
□

Lemma 5 *In a given step of the hybrid execution with Sim_f, let S be the set of messages sent from Π^c to Sim_f. Let $u' \in \mathbf{T}$ be the message sent from Sim_f to Π^c in that step. Let $\mathcal{C}$ be a context and*

$u \in \mathbf{T}$ such that $u' = \mathcal{C}[u]$ and $S \nvdash u$ and $\mathcal{C}$ does not contain a subterm of the form $sig(\square, \cdot, \cdot)$. ($\square$ denotes the hole of the context $\mathcal{C}$.)

Then there exists a term t_{bad} and a context $\mathcal{D}$ such that $\mathcal{D}$ obeys the following grammar

$$\begin{aligned}\mathcal{D} ::= \square \mid pair(t, \mathcal{D}) \mid pair(\mathcal{D}, t) \mid E(ek(N), \mathcal{D}, M)\\ \mid E(\mathcal{D}, t, M) \mid sig(sk(M), \mathcal{D}, M)\\ \mid garbageE(\mathcal{D}, M) \mid garbageSig(\mathcal{D}, M)\\ \text{with } N \in \mathbf{N}_P, M \in \mathbf{N}_E, t \in \mathbf{T}\end{aligned}$$

and such that $u = \mathcal{D}[t_{bad}]$ and such that $S \nvdash t_{bad}$ and such that one of the following holds: $t_{bad} \in \mathbf{N}_P$, or $t_{bad} = E(p, m, N)$ with $N \in \mathbf{N}_P$, or $t_{bad} = sig(k, m, N)$ with $N \in \mathbf{N}_P$, or $t_{bad} = sig(sk(N), m, M)$ with $N \in \mathbf{N}_P$, $M \in \mathbf{N}_E$ or $t_{bad} = ek(N)$ with $N \in \mathbf{N}_P$, or $t_{bad} = vk(N)$ with $N \in \mathbf{N}_P$.

Proof. We prove the lemma by structural induction on M. We distinguish the following cases:

Case 1: *"$u = garbage(u_1)$".*
By protocol condition 9 the protocol does not contain *garbage*-constructor nodes. Thus u is not an honestly generated term. Hence it was produced by an invocation $\tau(m)$ for some $m \in \{0,1\}^*$, and hence $u = garbage(N^m)$. Hence $S \vdash u$ in contradiction to the premise of the lemma.

Case 2: *"$u = garbageE(u_1, u_2)$".*
By protocol condition 9 the protocol does not contain *garbageE*-constructor nodes. Thus u is not an honestly generated term. Hence it was produced by an invocation $\tau(c)$ for some $c \in \{0,1\}^*$, and hence $u = garbageE(u_1, N^m)$. Since $S \vdash N^m$ and $S \nvdash u$, we have $S \nvdash u_1$. Hence by the induction hypothesis, there exists a subterm t_{bad} of u_1 and a context $\mathcal{D}$ satisfying the conclusion of the lemma for u_1. Then t_{bad} and $\mathcal{D}' := garbageE(\mathcal{D}, N^m)$ satisfy the conclusion of the lemma for u.

Case 3: *"$u = garbageSig(u_1, u_2)$".*
By protocol condition 9 the protocol does not contain *garbageSig*-constructor nodes. Thus u is not an honestly generated term. Hence it was produced by an invocation $\tau(c)$ for some $c \in \{0,1\}^*$, and hence $u = garbageSig(u_1, N^m)$. Since $S \vdash N^m$ and $S \nvdash u$, we have $S \nvdash u_1$. Hence by the induction hypothesis, there exists a subterm t_{bad} of u_1 and a context $\mathcal{D}$ satisfying the conclusion of the lemma for u_1. Then t_{bad} and $\mathcal{D}' := garbageSig(\mathcal{D}, N^m)$ satisfy the conclusion of the lemma for u.

Case 4: *"$u = dk(u_1)$".*
By protocol condition 5, any *dk*-constructor node occurs only as the first argument of a D-destructor node. The output of the destructor D only contains a subterm $dk(u_1)$ if its second argument already contained such a subterm. Hence a term $dk(u_1)$ cannot be honestly generated. But subterms of the form $dk(\cdot)$ are not in the range of τ. (Except if $dk(\cdot)$ was given as argument to a call to β. However, as β is only invoked with terms sent by Π^c, this can only occur if $dk(\cdot)$ was honestly generated or produced by τ.) Thus no term sent by Sim_f contains $dk(\cdot)$. Hence u cannot be a subterm of u'.

Case 5: *"$u = ek(u_1)$ with $u_1 \notin \mathbf{N}_P$".*
By protocol condition 1, the argument of a *ek*-constructor node is a N-constructor node with $N \in \mathbf{N}_P$. Hence u is not honestly generated. Hence it was produced by an invocation $\tau(e)$ for some $e \in \{0,1\}^*$, and hence $u = ek(N^e)$. Hence $S \vdash u$ in contradiction to the premise of the lemma.

Case 6: *"$u = ek(N)$ with $N \in \mathbf{N}_P$".*
The conclusion of the lemma is fulfilled with $\mathcal{D} := \square$ and $t_{bad} := u$.

Case 7: *"$u = vk(u_1)$ with $u_1 \notin \mathbf{N}_P$".*
By protocol condition 1, the argument of a *vk*-constructor node is an N-constructor node with $N \in \mathbf{N}_P$. Hence u is not honestly generated. Hence it was produced by an invocation $\tau(e)$ for some $e \in \{0,1\}^*$, and hence $u = vk(N^e)$. Hence $S \vdash u$ in contradiction to the premise of the lemma.

Case 8: *"$u = vk(N)$ with $N \in \mathbf{N}_P$".*
The conclusion of the lemma is fulfilled with $\mathcal{D} := \Box$ and $t_{bad} := u$.

Case 9: *"$u = sk(N)$".*
Say a subterm $sk(N)$ *occurs free* in some term t' if an occurrence of $sk(N)$ in t' is not the first argument of a *sig*-constructor in t'. Since $\mathcal{C}$ is not of the form $sig(\Box, \cdot, \cdot)$, we have that u occurs free in u'. However, by protocol condition 7, Π^c only sends a free $sk(N)$ if Sim_f first sends one. And by construction of τ, Sim_f sends a free $sk(N)$ only if $sk(N)$ was given as an argument to a call to β. And $sk(N)$ is given as an argument to β only if it is sent by Π^c. Hence Sim_f cannot have sent u' in contradiction to the premise of the lemma.

Case 10: *"$u = pair(u_1, u_2)$".*
Since $S \nvdash u$, we have $S \nvdash u_i$ for some $i \in \{1, 2\}$. Hence by induction hypothesis, there exists a subterm t_{bad} of u_i and a context $\mathcal{D}$ satisfying the conclusion of the lemma for u_i. Then t_{bad} and $\mathcal{D}' = pair(\mathcal{D}, u_2)$ or $\mathcal{D}' = pair(u_1, \mathcal{D})$ satisfy the conclusion of the lemma for u.

Case 11: *"$u = string_i(u_1)$ with $i \in \{0,1\}$ or $u = empty$".*
Then, since $u \in \mathbf{T}$, u contains only the constructors $string_0, string_1, empty$. Hence $S \vdash u$ in contradiction to the premise of the lemma.

Case 12: *"$u \in \mathbf{N}_P$".*
The conclusion of the lemma is fulfilled with $\mathcal{D} := \Box$ and $t_{bad} := u$.

Case 13: *"$u \in \mathbf{N}_E$".*
Then $S \nvdash u$ in contradiction to the premise of the lemma.

Case 14: *"$u = E(u_1, u_2, N)$ with $N \in \mathbf{N}_P$".*
The conclusion of the lemma is fulfilled with $\mathcal{D} := \Box$ and $t_{bad} := u$.

Case 15: *"$u = E(u_1, u_2, u_3)$ with $S \nvdash u_1$ and $u_3 \notin \mathbf{N}_P$".*
By protocol condition 1, the third argument of an E-constructor node is a N-constructor node with $N \in \mathbf{N}_P$. Hence u is not honestly generated. Hence it was produced by an invocation $\tau(c)$ for some $c \in \{0,1\}^*$, and hence $u = E(ek(N), u_2, N^c)$ for some $N \in \mathbf{N}_P$. Since $S \nvdash u_1$, by induction hypothesis, there exists a subterm t_{bad} of $u_1 = ek(N)$ and a context $\mathcal{D}$ satisfying the conclusion of the lemma for $ek(N)$. Then t_{bad} and $\mathcal{D}' = E(\mathcal{D}, u_2, N^c)$ satisfy the conclusion of the lemma for u.

Case 16: *"$u = E(u_1, u_2, u_3)$ with $S \vdash u_1$ and $u_3 \notin \mathbf{N}_P$".*
Analogous to the previous case, $u = E(ek(N), u_2, N^c)$ for some $N \in \mathbf{N}_P$. From $S \vdash u_1$, $S \vdash N^c$, and $S \nvdash u$ we have $S \nvdash u_2$. Hence by induction hyposthesis, there exists a subterm t_{bad} of u_2 and a context $\mathcal{D}$ satisfying the conclusion of the lemma for u_2. Then Then t_{bad} and $\mathcal{D}' = E(ek(N), \mathcal{D}, N^c)$ satisfy the conclusion of the lemma for u.

Case 17: *"$u = sig(u_1, u_2, N)$ with $N \in \mathbf{N}_P$".*
The conclusion of the lemma is fulfilled with $\mathcal{D} := \Box$ and $t_{bad} := u$.

Case 18: *"$u = sig(sk(N), u_2, u_3)$ with $u_3 \notin \mathbf{N}_P$ and $N \in \mathbf{N}_P$".*
Since $u \in \mathbf{T}$ we have $u_3 \in \mathbf{N}$, hence $u_3 \in \mathbf{N}_E$. The conclusion of the lemma is fulfilled with $\mathcal{D} := \Box$ and $t_{bad} := u$.

Case 19: *"$u = sig(u_1, u_2, u_3)$ with $S \vdash u_1$ and $u_3 \notin \mathbf{N}_P$ and u_1 is not of the form $sk(N)$ with $N \in \mathbf{N}_P$".*
By protocol condition 1, the third argument of an E-constructor node is an N-constructor node with $N \in \mathbf{N}_P$. Hence u is not honestly generated. Hence it was produced by an invocation $\tau(s)$ for some $s \in \{0,1\}^*$, and hence $u = sig(sk(N), u_2, N^s)$ for some $N \in \mathbf{N}$. Since u_1 is not of the form $sk(N)$ with $N \in \mathbf{N}_P$, we have $N \in \mathbf{N}_E$. From $S \vdash u_1$, $S \vdash N^c$, and $S \nvdash u$ we have $S \nvdash u_2$. Hence by induction hyposthesis, there exists a subterm t_{bad} of u_2 and a context $\mathcal{D}$ satisfying the conclusion of the lemma for u_2. Then Then t_{bad} and $\mathcal{D}' = sig(sk(N), \mathcal{D}, N^s)$ satisfy the conclusion of the lemma for u.

Case 20: *"$u = sig(u_1, u_2, N)$ with $S \nvdash u_1$ and $u_3 \notin \mathbf{N}_P$".*
As in the previous case, $u = sig(sk(N), u_2, N^s)$ for some $N \in \mathbf{N}$. Since $S \nvdash u_1$, $N \notin \mathbf{N}_E$. Hence $N \in \mathbf{N}_P$. Thus conclusion of the lemma is fulfilled with $\mathcal{D} := \Box$ and $t_{bad} := u$.

□

Lemma 6 *For any (direct or recursive) invocation of $\beta(t)$ performed by Sim_f, we have that $S \vdash t$ where S is the set of all terms sent by Π^c to Sim_f up to that point.*

Proof. Assume that a call to $\beta(t)$ is performed with $S \nvdash t$. By construction of β, $\beta(t)$ is only invoked either directly by Sim_f (and in this case $t \in S$), or recursively by $\beta(t')$ where t is a subterm of t'. Hence t is a subterm of some term $t' \in S$. Since $S \vdash t'$ and $S \nvdash t$, by definition of $\vdash$, there is a subterm t'' of t' and a context $\mathcal{C}$ such that $S \nvdash \mathcal{C}[t]$ and $\beta(t)$ is invoked recursively from $\beta(\mathcal{C}[t])$ and one of the following holds:

$t'' = E(t_1, \mathcal{C}[t], t_3)$ or $t'' = E(t_1, t_2, \mathcal{C}[t])$ or $t'' = ek(\mathcal{C}[t])$ or $t'' = dk(\mathcal{C}[t])$ or $t'' = vk(\mathcal{C}[t])$ or $t'' = sk(\mathcal{C}[t])$ or $t'' = sig(\mathcal{C}[t], t_2, t_3)$ or $t'' = sig(t_1, t_2, \mathcal{C}[t])$ or $t'' = garbage(\mathcal{C}[t])$ or $t'' = garbageE(t_1, \mathcal{C}[t])$ or $t'' = garbageSig(t_1, \mathcal{C}[t])$.

In all these cases except for the case $t'' = vk(\mathcal{C}[t])$, we have the following:

By definition of $\vdash$, and since $S \nvdash \mathcal{C}[t]$, we have that $S \vdash t''$ only if t'' is a subterm of some term in S. Hence any term sent by Sim_f contains t'' if Sim_f received a term containing t'' earlier. Thus t'' is honestly generated.

Similarly, in the case $t'' = vk(\mathcal{C}[t])$, we have that $S \vdash t''$ only if $t'' = vk(\mathcal{C}[t])$ or $sk(\mathcal{C}[t])$ is a subterm of some term in S_i. Hence $vk(\mathcal{C}[t])$ or $sk(\mathcal{C}[t])$ is honestly generated.

Because of the protocol conditions 1, 8, and 9, this leaves only the following possibilities for t'':

$t'' = E(t_1, \mathcal{C}[t], N)$ with $N \in \mathbf{N}_P$ or $t'' = E(t_1, t_2, \mathcal{C}[t])$ with $\mathcal{C}[t] \in \mathbf{N}_P$ or $t' = f(\mathcal{C}[t])$ with $f \in \{ek, dk, vk, sk\}$ and $\mathcal{C}[t] \in \mathbf{N}_P$ or $t'' = sig(\mathcal{C}[t], t_2, N)$ with $\mathcal{C}[t] = sk(N')$ and $N, N' \in \mathbf{N}_P$ and or $t'' = sig(sk(N), t_2, \mathcal{C}[t])$ with $N, \mathcal{C}[t] \in \mathbf{N}_P$.

Note that in all these cases, the computation $\beta(t')$ in Sim_f does not need to invoke $\beta(\mathcal{C}[t])$. (Remember that Sim_f queries the signing and fake encryption oracle instead.) However, this is a contradiction to the fact that $\beta(t)$ was invoked recursively by $\beta(\mathcal{C}[t])$. □

Lemma 7 *Sim_f is DY for $\mathbf{M}$ and Π.*

Proof. Let $a_1, \dots, a_n$ be terms sent by the protocol to Sim_f. Let $u_1, \dots, u_n$ be the terms sent by Sim_f to the protocol. Let $S_i := \{a_1, \dots, a_i\}$. If Sim_f is not DY, then with non-negligible probability there exists an i such that $S_i \nvdash u_i$. Fix the smallest such i_0 and set $S := S_{i_0}$ and $u := u_{i_0}$. By Lemma 5 (with $u' := u$ and $\mathcal{C} := \Box$), we have that there is a term t_{bad} and a context $\mathcal{D}$ obeying the grammar given in Lemma 5 and such that $u = \mathcal{D}[t_{bad}]$ and such that $S \nvdash t_{bad}$ and such that one of the following holds: (a) $t_{bad} \in \mathbf{N}_P$, or (b) $t_{bad} = E(p, m, N)$ with $N \in \mathbf{N}_P$, or (c) $t_{bad} = sig(k, m, N)$ with $N \in \mathbf{N}_P$, or (d) $t_{bad} = sig(sk(N), m, M)$ with $N \in \mathbf{N}_P$, $M \in \mathbf{N}_E$ or (e) $t_{bad} = ek(N)$ with $N \in \mathbf{N}_P$, or (f) $t_{bad} = vk(N)$ with $N \in \mathbf{N}_P$.

By construction of the simulator, if the simulator outputs u, we know that the simulated adversary E has produced a bitstring m such that $\tau(m) = u = \mathcal{D}[t_{bad}]$. By definition of τ, during the computation of $\tau(m)$, some recursive invocation of τ has returned t_{bad}. Hence the simulator has computed a bitstring m_{bad} with $\tau(m_{bad}) = t_{bad}$.

We are left to show that such a bitstring m_{bad} can be found only with negligible probability.

We distinguish the possible values for t_{bad} (as listed in Lemma 5):

If $t_{bad} = N \in \mathbf{N}_P$, then $m_{bad} = r_N$ and r_N is never used. So the probability that $m_{bad} = r_N$ occurs as output of τ is negligible.

If $t_{bad} = E(p, m, N)$ with $N \in \mathbf{N}_P$, then $\tau(m_{bad})$ returns t_{bad} only if m_{bad} was the output of an invocation of $\beta(E(p, m, N)) = \beta(t_{bad})$. But by Lemma 6, $\beta(t_{bad})$ is never invoked, so this case does not occur.

If $t_{bad} = \mathit{sig}(k, m, N)$ with $N \in \mathbf{N}_P$, then $\tau(m_{bad})$ returns t_{bad} only if m_{bad} was the output of an invocation of $\beta(\mathit{sig}(k, m, N)) = \beta(t_{bad})$. But by Lemma 6, $\beta(t_{bad})$ is never invoked, so this case does not occur.

If $t_{bad} = \mathit{sig}(\mathit{sk}(N), m, M)$ with $N \in \mathbf{N}_P$, $M \in \mathbf{N}_E$, then $\tau(m_{bad})$ returns t_{bad} only if m_{bad} was not the output of an invocation of β. In particular, m_{bad} was not produced by the signing oracle. Furthermore, $\tau(m_{bad})$ returns t_{bad} only if m_{bad} is a valid signature with respect to the verification key vk_N. Hence m_{bad} is a valid signature that was not produced by the signing oracle. Such a bitstring m_{bad} can only be produced with negligible probability by E because of the strong existential unforgeability of $(\mathsf{SKeyGen}, \mathsf{Sig}, \mathsf{Verify})$ (implementation condition 20).

If $t_{bad} = \mathit{ek}(N)$ with $N \in \mathbf{N}_P$, then by Lemma 6, $\beta(\mathit{ek}(N))$ is never computed and hence ek_N never requested from the encryption oracle. Furthermore, from protocol conditions 5 and 2, we have that no term sent by Π^c contains $\mathit{dk}(N)$, and all occurrences of N in terms sent by Π^c are of the form $\mathit{ek}(N)$. Thus $S \nvdash \mathit{sk}(N)$. Hence by Lemma 6, $\beta(\mathit{sk}(N))$ is never computed and sk_N is never requested from the encryption oracle. Furthermore, since $S \nvdash \mathit{ek}(N)$, for all terms of the form $t = E(\mathit{ek}(N), \dots, \dots)$, we have that $S \nvdash t$. Thus $\beta(t)$ is never computed and hence no encryption using ek_N is ever requested from the encryption oracle. However, decryption queries with respect to sk_N are still sent to the encryption oracle. Yet, by implementation conditions 11, these will always fail unless the ciphertext to be decrypted already satisfies $A_{\mathit{ekof}}(m) = \mathit{vk}_N$, i.e., if vk_N has already been guessed. Hence the probability that $\mathit{ek}_N = m_{bad}$ occurs as output of τ is negligible.

If $t_{bad} = \mathit{vk}(N)$ with $N \in \mathbf{N}_P$, then by Lemma 6, $\beta(\mathit{vk}(N))$ is never computed and hence vk_N is never requested from the signing oracle. Furthermore, since $S \nvdash \mathit{vk}(N)$, we also have $S \nvdash \mathit{sk}(N)$ and $S \nvdash t$ for $t = \mathit{sig}(\mathit{sk}(N), \dots, \dots)$. Thus $\beta(\mathit{sk}(N))$ and $\beta(t)$ never computed and hence neither sk_N nor a signature with respect to sk_N is requested from the signing oracle. Hence the probability that $\mathit{vk}_N = m_{bad}$ occurs as output of τ is negligible.

Summarizing, we have shown that if the simulator Sim_f is not DY, then with non-negligible probability Sim_f performs the computation $\tau(m_{bad})$, but m_{bad} can only occur with negligible probability as an argument of τ. Hence we have a contradiction to the assumption that Sim_f is not DY. $\square$

Theorem 2 *The implementation A (satisfying the implementation conditions given on page 14) is a computationally sound implementation of the symbolic model $\mathbf{M}$ (defined on page 12) for the class of key-safe protocols.*

Proof. By Lemma 7, Sim_f is DY for key-safe protocols. Whether a full trace satisfies the conditions from Definition 14 can be efficiently verified (since $\vdash$ is efficiently decidable). Hence Lemma 3 implies that Sim is DY for key-safe protocol, too. By Lemma 4, Sim is indistinguishable. Hence Sim is a good simulator for $\mathbf{M}$, key-safe Π, A, and polynomials p. By Theorem 1, the computational soundness of A for key-safe protocols follows. $\square$

4 Computational soundness of the applied π-calculus

In this section we show how to use CoSP to establish the first computational soundness result for the full-fledged applied π-calculus, including arbitrary equational theories, under active attacks. We consider the process calculus proposed in [24] additionally augmented with events; the calculus in [24] itself is a combination of the original applied π-calculus [2] with one of its dialects [22]. This combination offers the richness of the original applied π-calculus while additionally being accessible to state-of-the-art verification tools such as ProVerif [21]; in particular, we allow arbitrary equational theories. Our result hence yields computational soundness guarantees for ProVerif.

We first syntactically embed the applied π-calculus into CoSP. This embedding is particularly instructive because the applied π-calculus differs significantly from CoSP, e.g., the applied π-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 π-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 π-calculus in Section 4.2. In Section 4.3, we define a computational execution of the calculus (this is only necessary since the applied π-calculus does not come with an a-priori defined computational execution), called computational π-execution, as well as trace properties in the applied π-calculus, called π-trace properties. In Section 4.4, we establish the actual soundness result using CoSP: We first define a symbolic model of the applied π-calculus (in the sense of Definition 6) 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, then every π-calculus process that symbolically fulfills a π-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 1. (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 [24], except that we add processes of the form $event(e).P$ 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 π-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 π-constructors, π-traces, etc. The set of ground π-terms is denoted T_π. By $fn(P)$ we denote the set of free names of P, i.e., the names n not protected by a restriction. 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 π-constructors $\mathbf{C}_\pi$, a (possibly infinite) set of π-destructors $\mathbf{D}_\pi$, and an arbitrary equivalence relation $\approx$ over ground π-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 π-destructors and π-constructors in the following sense: For all π-constructors f and π-destructors d of arity n, for all ground π-terms $M_1, \dots, M_n, M'_1, \dots, M'_n$ with $M_i \approx M'_i$ for $i = 1, \dots, n$, we have that $f(\underline{M}) \approx f(\underline{M'})$, that $d(\underline{M}) = \bot$ iff $d(\underline{M'}) = \bot$, and that $d(\underline{M}) \approx d(\underline{M'})$. We also require $d(\underline{M}\tau) = d(\underline{M})\tau$ for any renaming τ 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 $equals$: Let $equals(x, y) = x$ for $x \approx y$ and define $if\ M = N\ then\ P\ else\ Q$ as $let\ x = equals(M, N)\ in\ P\ else\ Q$ for some $x \notin fv(P)$. In the following, we will assume $equals \in \mathbf{D}_\pi$. Furthermore, we write $let\ x = D\ in\ P$ for $let\ x = D\ in\ P\ else\ 0$ and analogously for if.

Given a ground destructor π-term D, we can evaluate it to a ground π-term $\mathrm{eval}^\pi(D)$ by evaluating all π-destructors. If one of the π-destructors returns $\bot$, we set $\mathrm{eval}^\pi(D) := \bot$.

The semantics of the calculus is standard and corresponds to the one defined in [24] except for the addition of events. The semantics hence consists of two possible transitions: $\to$ and $\xrightarrow{e}$. The latter

$M, N ::=$	terms	
	x, y, z	variables
	a, b, c	names
	$f(M_1, \ldots, M_n)$	constructor application
$D ::=$	destructor terms	
	M	terms
	$d(D_1, \ldots, D_n)$	destructor application
	$f(D_1, \ldots, D_n)$	constructor application
$P, Q ::=$	processes	
	$\bar{M}\langle N\rangle.P$	output
	$M(x).P$	input
	0	nil
	$P \mid Q$	parallel composition
	$!P$	replication
	$\nu a.P$	restriction
	$\mathit{let}\ x = D$ $\mathit{in}\ P\ \mathit{else}\ Q$	let
	$\mathit{event}(e).P$	event

Figure 1: Syntax of the process calculus.

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. Again, we prefix some notions with π to distinguish them from their corresponding notions in Section 2.2. The semantics is formally defined in Figure 2.

Definition 17 (π-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 $P \mid Q \to^* \xrightarrow{e_1} \to^* \xrightarrow{e_2} \to^* \cdots \to^* \xrightarrow{e_n}$. A* π-trace property *is an efficiently decidable and prefix-closed set of strings. A process P* symbolically satisfies *a* π-trace property *$\wp$ if we have $e \in \wp$ for all event traces e of P.*

4.3 Defining a computational execution

A computational π-implementation assigns a partial deterministic polynomial-time algorithm A^π_f to each π-constructor f, and a partial deterministic polynomial-time algorithm A^π_d to each π-destructor d. We also fix an efficiently sampleable set Nonces_k depending on a security parameter k. We require that $A^\pi_{equals}(1^k, x, x) = x$ and $A^\pi_{equals}(1^k, x, y) = \bot$ for $x \neq y$ (i.e., the computational interpretation of $\approx$ is the equality of bitstrings). Given an assignment μ from names to bitstrings and an assignment η from variables to bitstrings for names and variables occurring in a destructor term D, we can (computationally) evaluate D to a bitstring $\mathrm{ceval}_{\eta,\mu} D$. (Formally, the security parameter k is an additional input to ceval, but we omit k for readability). We set $\mathrm{ceval}_{\eta,\mu} D := \bot$ if the application of one of the algorithms A^π_f or A^π_d 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 non-determinism 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

$$\frac{}{P \mid 0 \equiv P} \qquad \frac{}{P \equiv P} \qquad \frac{}{P \mid Q \equiv Q \mid P} \qquad \frac{}{(P \mid Q) \mid R \equiv P \mid (Q \mid R)}$$

$$\frac{P \equiv Q \qquad Q \equiv R}{P \equiv R} \qquad \frac{}{\nu a.\nu b.P \equiv \nu b.\nu a.P} \qquad \frac{P \equiv Q}{P \mid R \equiv Q \mid R}$$

$$\frac{a \notin \mathit{fn}(P)}{\nu a.(P \mid Q) \equiv P \mid \nu a.Q} \qquad \frac{P \equiv Q}{\nu a.P \equiv \nu a.Q} \qquad \frac{N \approx N'}{\overline{N}\langle M\rangle.Q \mid N'(x).P \to Q \mid P\{M/x\}}$$

$$\frac{\mathrm{eval}^{\pi}\ D \neq \bot}{\mathit{let}\ x = D\ \mathit{in}\ P\ \mathit{else}\ Q \to P\{\mathrm{eval}^{\pi}\ D/x\}} \qquad \frac{\mathrm{eval}^{\pi}\ D = \bot}{\mathit{let}\ x = D\ \mathit{in}\ P\ \mathit{else}\ Q \to Q}$$

$$\frac{}{!P \to P \mid !P} \qquad \frac{P \to Q}{P \mid R \to Q \mid R \qquad \nu a.P \to \nu a.Q}$$

$$\frac{P' \equiv P \qquad P \to Q \qquad Q \equiv Q'}{P' \to Q'} \qquad \frac{}{\mathit{event}(e).P \xrightarrow{e} P}$$

$$\frac{P \xrightarrow{e} Q}{P \mid R \xrightarrow{e} Q \mid R \qquad \nu a.P \xrightarrow{e} \nu a.Q} \qquad \frac{P' \equiv P \qquad P \xrightarrow{e} Q \qquad Q \equiv Q'}{P' \xrightarrow{e} Q'}$$

Figure 2: Semantics of the applied π-calculus with events.

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.[5] 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 π-execution of a process is now defined as an interactive machine that executes the process and communicates with an adversary. The computational π-execution maintains a process P representing the current process, an environment η storing the bitstrings assigned to the free variables in P, and an interpretation μ of the free names in P as bitstrings.

Definition 18 (Computational π-execution) *Let P_0 be a closed process, and let C be an interactive machine called the adversary. We define the computational π-execution as an interactive machine $\mathrm{Exec}_{P_0}(1^k)$ that takes a security parameter k as 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 η be a totally undefined partial function mapping variables to bitstrings, let μ be a totally undefined partial function mapping names to bitstrings. Let $a_1, \dots, a_n$ denote the free names in P_0. For each i, pick $r_i \in \mathrm{Nonces}_k$ at random. Set $\mu := \mu(a_1 := r_1, \dots, a_n := r_n)$. Send $(r_1, \dots, r_n)$ to C.*[6]
- Main loop: *Send P to the adversary and expect an evaluation context E from the adversary. Distinguish the following cases:*

[5]Traditionally, one considers evaluation contexts where the hole may also be protected by a restriction. However, computationally the evaluation of a restriction has to be considered a proper reduction step (it corresponds to choosing a nonce).

[6]In the applied π-calculus, free names occurring in the initial process represent nonces that are honestly chosen but known to the attacker.

$$\frac{m \in S}{S \vdash m} \qquad \frac{N \in \mathbf{N}_E}{S \vdash N} \qquad \frac{S \vdash \underline{M} \qquad f \in \mathbf{C} \setminus \mathbf{N}}{S \vdash f(\underline{M})}$$

$$\frac{S \vdash \underline{M} \qquad d \in \mathbf{D} \qquad d(\underline{M}) \neq \bot}{S \vdash d(\underline{M})}$$

Figure 3: Deduction rules for the symbolic model of the applied π-calculus

- $P = E[M(x).P_1]$: *Request two bitstrings c, m from the adversary. If $c = \mathrm{ceval}_{\eta,\mu} M$, set $\eta := \eta(x := m)$ and $P := E[P_1]$.*
- $P = E[\nu a.P_1]$: *Pick $r \in \mathrm{Nonces}_k$ at random, set $P := E[P_1]$ and $\mu := \mu(a := r)$.*
- $P = E[\overline{M_1}\langle N\rangle.P_1][M_2(x).P_2]$: *If $\mathrm{ceval}_{\eta,\mu} M_1 = \mathrm{ceval}_{\eta,\mu} M_2$ then set $P := E[P_1][P_2]$ and $\eta := \eta(x := \mathrm{ceval}_{\eta,\mu} N)$.*
- $P = E[\mathit{let}\ x = D\ \mathit{in}\ P_1\ \mathit{else}\ P_2]$: *If $m := \mathrm{ceval}_{\eta,\mu} D \neq \bot$, set $\eta := \eta(x := m)$ and $P := E[P_1]$; Otherwise set $P := E[P_2]$.*
- $P = E[\mathit{event}(e).P_1]$: *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 η and μ, yielding a process $\tilde{P}_1$. Set $P := E[\tilde{P}_1 \mid !P_1]$.*
- $P = E[\overline{M}\langle N\rangle.P_1]$: *Request a bitstring c from the adversary. If $c = \mathrm{ceval}_{\eta,\mu} M$, set $P := E[P_1]$ and send $\mathrm{ceval}_{\eta,\mu} N$ to the adversary.*
- *In all other cases, do nothing.*

The execution of $\mathrm{Exec}_{P_0}(1^k)$ 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 η and μ. For a given polynomial-time interactive machine C, a closed process P_0, and a polynomial p, we let $\mathit{Events}_{C,P,p}(k)$ the list of events e raised within the first $p(k)$ computation steps (jointly counted for $C(1^k)$ and $\mathrm{Exec}_{P_0}(1^k)$).

We finally define the computational fulfillment of π-trace properties.

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

4.4 Computational soundness of the calculus

We will now derive the computational soundness of the applied π-calculus, i.e., we will show that if its computational implementation is computationally sound in the sense of Definition 11, then every symbolically satisfied π-trace property is also computationally satisfied. Applying Definition 11 first requires us to specify a symbolic model of the applied π-calculus (in the sense of Definition 6) and a computational implementation of this model (in the sense of Definition 8).

The symbolic model of the applied π-calculus contains all the π-constructors and π-destructors from the process calculus. We additionally add an infinite number of adversary nonces $\mathbf{N}_E$ and protocol nonces $\mathbf{N}_P$ 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 π-calculus) *For a π-destructor d, we define d' by $d'(\underline{t}) := d(\underline{t}\rho)\rho^{-1}$ where ρ is any injective map from the nonces occurring in the* CoSP*-terms $\underline{t}$ to*

$$\begin{tikzcd}
\boxed{P \text{ symb. satisfies } \wp} \arrow[r, "\text{Lemma 10}"] & \boxed{\mathrm{SExec}_{P_0} \text{ satisfies } \wp} \arrow[r, "\text{Lemma 8}"] & \boxed{\Pi_{P_0} \text{ symb. satisfies } \mathit{events}^{-1}(\wp)} \arrow[d, "\begin{array}{l}\text{Lemma 9}\\ A \text{ computationally sound}\end{array}"] \\
\boxed{P \text{ comp. satisfies } \wp} & & \boxed{(\Pi_{P_0}, A) \text{ comp. satisfies } \mathit{events}^{-1}(\wp)} \arrow[ll, "\text{Lemma 8}"']
\end{tikzcd}$$

Figure 4: Overview of the proof of Theorem 3.

names.[7] *Let* $\mathbf{N}_E$ *and* $\mathbf{N}_P$ *be countably infinite sets.*

The symbolic model of the applied π-calculus *is given by* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$*, where* $\mathbf{N} := \mathbf{N}_E \cup \mathbf{N}_P$*,* $\mathbf{C} := \mathbf{C}_\pi$*,* $\mathbf{D} := \{d' : d \in \mathbf{D}_\pi\}$*, and where* $\mathbf{T}$ *consists of all terms over* $\mathbf{C}$ *and* $\mathbf{N}$*,*[8] *and where* $\vdash$ *is defined by the rules in Figure 3.*

In the following, we fix $\mathbf{M}, \mathbf{C}, \mathbf{N}, \mathbf{D}, \vdash$ as in Definition 20. In particular, the destructor equals' thus induces an equivalence relation $\cong$ on the set of CoSP-terms with $x \cong y$ iff $\mathit{equals}'(x, y) \neq \bot$. The relation $\cong$ is the analogue to the equivalence relation $\approx$ describing the equational theory of the applied π-calculus.

The computational implementation of this symbolic model is now specified by the computational π-implementations A_f and A_d of the π-constructors and π-destructors, with nonces being chosen uniformly at random.

Definition 21 (Computational implementation of Def. 20) *The computational implementation A of the symbolic model* $\mathbf{M}$ *of the applied π-calculus is given by* $A_f := A^\pi_f$ *for all* $f \in \mathbf{C}$ *and* $A_d := A^\pi_d$ *for all* $d \in \mathbf{D}$*.* A_N *for* $N \in \mathbf{N}$ *picks* $r \in \mathrm{Nonces}_k$ *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 non-deterministic machine SExec_{P_0}, analogous to the machine Exec_{P_0} from Definition 18. Intuitively, the only difference between Exec_{P_0} and SExec_{P_0} is that the latter operates immediately on terms whenever the former operates on computational implementations of these terms.

In the following definition, note that for π-terms M, we have that $\mathrm{eval}^{\mathsf{CoSP}}\, M\eta\mu = M\eta\mu$ (this does not hold for destructor terms D). In these cases, we write the redundant $\mathrm{eval}^{\mathsf{CoSP}}$ anyway to emphasis the analogy to Definition 18.

Definition 22 (Symbolic execution of a π-process) *Let* P_0 *be a closed process, and let* C *be an interactive machine called the adversary. We define the symbolic π-execution as an interactive machine* SExec_{P_0} *that 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* η *be a totally undefined partial function mapping variables to* ***terms****, let* μ *be a totally undefined partial function mapping names to* ***terms****. Let* $a_1, \dots, a_n$ *denote the free names in* P_0*. For each* i*,* ***choose a different*** $\boldsymbol{r_i \in \mathbf{N}_P}$ *Set* $\mu := \mu(a_1 := r_1, \dots, a_n := r_n)$*. Send* $(r_1, \dots, r_n)$ *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* **CoSP*-terms*** c*,* m *from the adversary. If* $\boldsymbol{c \cong \mathrm{eval}^{\mathsf{CoSP}}\, M\eta\mu}$*, set* $\eta := \eta(x := m)$ *and* $P := E[P_1]$*.*
  - $P = E[\nu a.P_1]$*:* ***Choose*** $\boldsymbol{r \in \mathbf{N}_P \setminus \mathrm{range}\,\mu}$*, set* $P := E[P_1]$ *and* $\mu := \mu(a := r)$*.*

[7]This is well-defined and independent of ρ since for any renaming of names τ, we have $d(\underline{M}\tau) = d(\underline{M})\tau$; intuitively d' behaves as d except that it uses nonces instead of names.

[8]One can get a model with such a message type $\mathbf{T}$ using, e.g., Lemma 1.

- $P = E[\overline{M_1}\langle N\rangle.P_1][M_2(x).P_2]$: *If* $\mathbf{eval}^{\mathsf{CoSP}}\,\boldsymbol{M_1\eta\mu} \cong \mathbf{eval}^{\mathsf{CoSP}}\,\boldsymbol{M_2\eta\mu}$ *then set* $P := E[P_1][P_2]$ *and* $\eta := \eta(x := \mathbf{eval}^{\mathsf{CoSP}}\,\boldsymbol{N\eta\mu})$.
- $P = E[let\ x = D\ in\ P_1\ else\ P_2]$: *If* $m := \mathbf{eval}^{\mathsf{CoSP}}\,\boldsymbol{D\eta\mu} \neq \bot$, *set* $\eta := \eta(x := m)$ *and* $P := E[P_1]$*; Otherwise set* $P := E[P_2]$.
- $P = E[event(e).P_1]$: *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* η *and* μ*, yielding a process* $\tilde{P}_1$*. Set* $P := E[\tilde{P}_1\ |!P_1]$.
- $P = E[\overline{M}\langle N\rangle.P_1]$: *Request a* **CoSP-*term*** c *from the adversary. If* $\boldsymbol{c} \cong \mathbf{eval}^{\mathsf{CoSP}}\,\boldsymbol{M\eta\mu}$*, set* $P := E[P_1]$ *and send* $\mathbf{eval}^{\mathsf{CoSP}}\,\boldsymbol{N\eta\mu}$ *to the adversary.*
- *In all other cases, do nothing.*

The only differences between Definition 18 and Definition 22 are that the latter operates on CoSP-terms instead of bitstrings, it computes $\mathrm{eval}^{\mathsf{CoSP}}\,X\eta\mu$ instead of $\mathrm{ceval}_{\eta,\mu}\,X$, it compares CoSP-terms using $\cong$ instead of checking for equality of bitstrings, and it chooses a fresh CoSP-nonce $r \in \mathbf{N}_P$ instead of choosing a random bitstring r as value for a restricted name.

The interactive machine SExec_{P_0} performs only the following operations on CoSP-terms: Applying CoSP-constructors (this includes nonces) and CoSP-destructors, comparing using $\cong$ (which can be realized by an application of the destructor *equals*$'$), and sending and receiving terms. Hence this interactive machine can be realized as a CoSP protocol in the sense of Definition 3: The state of the machine SExec_{P_0} is used as a node identifier. However, CoSP-terms are not encoded directly into the node identifier; instead, the node in which they were created (or received) is referenced instead.[9] This is due to the fact that a CoSP protocol allows to treat CoSP-terms only as black boxes. Note that the process P and the π-terms occurring within P will be encoded in the node identifier (encoded as bitstrings). Operations on CoSP-terms can then be performed by using constructor and destructor nodes, and the input and output of CoSP-terms is handled using input/output nodes. Sending P to the adversary and receiving E is realized using control nodes (assuming a suitable encoding of terms as bitstrings). If an event e is raised, we model this by sending $(event, e)$ to the adversary using a control node with one successor. We call these nodes event nodes, and given a sequence of nodes $\underline{\nu}$, by $events(\underline{\nu})$ we denote the events e raised by the event nodes in $\underline{\nu}$. We call the resulting protocol Π_{P_0}. Since Π_{P_0} does not contain non-deterministic nodes, it is a CoSP protocol and a probabilistic protocol simultaneously.

Definition 23 *A non-deterministic 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* $S \vdash m$.

SExec_{P_0} *satisfies a* π*-trace property* $\wp$ *if in interaction with any Dolev-Yao adversary, the sequence of events raised by* SExec_{P_0} *is contained in* $\wp$.

Before we finally state and prove the soundness of the applied π-calculus, we provide three lemmas that are used to relate P_0, SExec_{P_0}, Π_{P_0}, and the computational implementation A, and to assert the efficiency of the protocol Π_{P_0}. Figure 4 illustrates the use of these lemmas in the overall proof.

Lemma 8 *Let* $\wp$ *be a trace property. Then* SExec_{P_0} *satisfies* $\wp$ *iff* Π_{P_0} *symbolically satisfies* $events^{-1}(\wp)$ *(in the sense of Definition 10). Moreover,* P_0 *computationally satisfies* $\wp$ *iff* (Π_{P_0}, A) *computationally satisfies* $events^{-1}(\wp)$ *(in the sense of Definition 10).*

Proof. The symbolic case is immediate from the construction of Π_{P_0}. For the computational case, note the fact that the computational implementation of P_0 is defined like the symbolic one, except that

[9]For technical reasons, we do not reference the nodes by their identifiers, but instead by their indexes in the path from the root to the referring node. Otherwise, the size of node identifiers would grow exponentially.

it uses the implementations of the CoSP-constructors and CoSP-destructors instead of the operating on abstractly on terms (and the implementation of *equals* uses the identity on bitstrings). $\square$

Lemma 9 *The protocol Π_{P_0} is efficient.*

Proof. By construction, there are efficient algorithms for computing the labels and successors of a node given its node identifier. It is left to show that the length of the node identifier of a node p is polynomial in the length of the path leading to that node. This is equivalent to showing that the state of SExec_{P_0} is of polynomial-length (when not counting the length of the representations of the CoSP-terms). For the variables η and μ, this is immediately satisfied because they grow by at most one entry in each activation of SExec_{P_0}. To show that the length of P is polynomially bounded, note the following facts: In each activation of SExec_{P_0}, P either gets smaller, or we have $P = E[!P_1]$ and P grows by the size of P_1. If $P = E[!P_1]$, then $!P_1$ is also a subterm of P_0 (up to renaming of names and variables). Hence in each activation, P grows at most by the size of P_0. Thus the size of P is linear in the number of activations of SExec_{P_0}. $\square$

Lemma 10 *If a closed process P_0 symbolically satisfies a π-trace property $\wp$, then* SExec_{P_0} *satisfies $\wp$.*

Proof. To show this lemma, it is sufficient to show that if SExec_{P_0} raises events $e_1, \dots, e_n$, then $\underline{e}$ is an event π-trace of P_0. Hence, for the following we fix an execution of SExec_{P_0} in interaction with a Dolev-Yao adversary E in which SExec_{P_0} raises the events $e_1, \dots, e_n$. We then prove the lemma by constructing a process $\tilde{Q}$ (that does not raise events) such that $\tilde{Q} \mid P_0 \to^* \xrightarrow{e_1} \to^* \xrightarrow{e_2} \to^* \cdots \to^* \xrightarrow{e_n} \to^*$.

For a given iteration of the main loop of SExec_{P_0}, let P, η, μ denote the corresponding variables from the state of SExec_{P_0} at the beginning of that iteration. Let E denote the evaluation context chosen in that iteration. Let $\underline{n}$ be the domain of μ without the names $r_1, \dots, r_n$ sent in the message $(r_1, \dots, r_n)$ in the very beginning of the execution of SExec_{P_0}. $P', \eta', \mu', \underline{n}'$ are the corresponding values *after* that iteration. Let *fromadv* be the list of terms received from the adversary in that iteration, and *toadv* the list of terms sent to the adversary. By $\bar{P}_0, \eta_0, \mu_0, \underline{n}_0$ we denote the corresponding values before the first iteration but after the sending of the message $(r_1, \dots, r_n)$,[10] and by $P_*, \eta_*, \mu_*, \underline{n}_*$ the values after the last iteration. We call a name or variable *used* if it occurs in the domain of μ_* or η_*, respectively. Note that $\mu_0 = (a_1 \mapsto r_1, \dots, a_n \mapsto r_n)$ where $\underline{a}$ are the free names in P_0, but $\underline{n}_0 = \varnothing$. Note that P will never contain unused free variables or names.

Let S denote the list of all CoSP-terms output by SExec_{P_0} up to the current iteration. We encode $S = (s_1, \dots, s_n)$ as a substitution φ mapping $x_i \mapsto s_i$ where x_i are arbitrary unused variables. We denote by S', φ' and S_0, φ_0 and S_*, φ_* the values of S, φ after the current iteration, before the first iteration (but after sending $(r_1, \dots, r_n)$), and after the last iteration, respectively. Note that $S_0 = (r_1, \dots, r_n)$.

Let γ be an injective partial function that maps every $N \in \mathbf{N}_E$ to an unused name, and every $N \in \operatorname{range} \mu_*$ to $\mu_*^{-1}(N)$. (This is possible because $\operatorname{range} \mu_* \subseteq \mathbf{N}_P$ and μ_* is injective.) We additionally require that all unused names are in $\operatorname{range} \gamma$. (This is possible since both $\mathbf{N}_E$ and the set of unused names are countably infinite.)

Note that for any π-destructor d and any π-terms $\underline{M}$ with $\mathit{fv}(\underline{M}) \subseteq \operatorname{dom} \eta$ and $\mathit{fn}(M) \subseteq \operatorname{dom} \mu$, we have that $\underline{M}\eta\mu$ are CoSP-terms and $d'(\underline{M}\eta\mu)\gamma = d(\underline{M}\eta\mu\gamma)$ (where d' is as in Definition 20). Hence for a destructor term D with $\mathit{fv}(D) \subseteq \operatorname{dom} \eta$ and $\mathit{fn}(D) \subseteq \operatorname{dom} \mu$, we have $\mathrm{eval}^{\mathsf{CoSP}}(D\eta\mu)\gamma = \mathrm{eval}^{\pi}(D\eta\mu\gamma)$. Since $a\mu\gamma = a$ for all names $a \in \operatorname{dom} \mu$, $D\eta\mu\gamma = D\eta\gamma$. Since $\mathrm{eval}^{\mathsf{CoSP}}(D\eta\mu)$ does not contain variables, $\mathrm{eval}^{\mathsf{CoSP}}(D\eta\mu) = \mathrm{eval}^{\mathsf{CoSP}}(D\eta\mu)\eta$. Thus for D with $\mathit{fv}(D) \subseteq \operatorname{dom} \eta$ and $\mathit{fn}(D) \subseteq \operatorname{dom} \mu$ we have

$$\mathrm{eval}^{\mathsf{CoSP}}(D\eta\mu)\eta\gamma = \mathrm{eval}^{\pi}(D\eta\gamma) \tag{2}$$

where the left hand side is defined iff the right hand side is.

[10] We use the variable name $\bar{P}_0$ because P_0 is already used for the input of SExec_{P_0}. Note however that $\bar{P}_0 \equiv P_0$.

Similarly to (2), if $fv(D) \subseteq \operatorname{dom}\varphi$ and $fn(D) \subseteq \operatorname{dom}\gamma^{-1}$, we have $\operatorname{eval}^{\mathsf{CoSP}}(D\varphi\gamma^{-1})\gamma = \operatorname{eval}^{\pi}(D\varphi\gamma)$. For a CoSP-term t with $S \vdash t$, from the definition of $\vdash$ it follows that $t = \operatorname{eval}^{\mathsf{CoSP}}(D_t\varphi\gamma^{-1})$ for some destructor π-term D_t containing only unused names and variables in $\operatorname{dom}\varphi$ (note that every $N \in \mathbf{N}_E$ can be expressed as $a\gamma^{-1}$ for some unused a). Since all unused names are in $\operatorname{dom}\gamma^{-1}$, we have

$$t\gamma = \operatorname{eval}^{\mathsf{CoSP}}(D_t\varphi\gamma^{-1})\gamma = \operatorname{eval}^{\pi}(D_t\varphi\gamma). \tag{3}$$

Given two CoSP-terms $t \cong u$ such that t and u only contain nonces $N \in \mathbf{N}_E \cup \operatorname{range}\mu_*$, we have that $equals'(t,u) \neq \perp$ by definition of $\cong$. By definition of $equals'$ (Definition 20) and using that γ is injective and defined on $\mathbf{N}_E \cup \operatorname{range}\mu_*$, we have $equals'(t,u) = equals(t\gamma, u\gamma)\gamma^{-1}$ and hence $equals(t\gamma,u\gamma) \neq \perp$. Hence, for t,u only containing nonces $N \in \mathbf{N}_E \cup \operatorname{range}\mu_*$, we have

$$t \cong u \qquad \Longrightarrow \qquad t\gamma \approx u\gamma \tag{4}$$

We call a process Q valid for φ if it does not contain events, all its free names are unused names, and all its free variables are in the domain of φ.

Claim: For all Q' valid for φ', there is a Q valid for φ such that $\nu\underline{n}.(Q\varphi\gamma|P\eta\gamma) \rightsquigarrow \nu\underline{n}'.(Q'\varphi'\gamma|P'\eta'\gamma)$. Here $\rightsquigarrow$ denotes $\xrightarrow{e}$ if an event e is raised in the current iteration, and $\to^*$ otherwise.

Assuming that we have shown this claim, it follows that for all Q_* valid for φ_*, there is a Q_0 valid for φ_0 such that $\nu\underline{n}_0.(Q_0\varphi_0\gamma|\bar{P}_0\eta_0\gamma) \to^*\xrightarrow{e_1}\to^*\xrightarrow{e_2}\to^* \cdots \to^*\xrightarrow{e_n}\to^* \nu\underline{n}.(Q_*\varphi_*\gamma|P_*\eta_*\gamma)$. Since $\eta_0 = \varnothing$ and since $\bar{P}_0$ does not contain $N \in \mathbf{N}$ (being a π-term) and since $\bar{P}_0$ is a renaming of P_0, we have $\bar{P}_0\eta_0\gamma = \bar{P}_0\gamma = \bar{P}_0 \equiv P_0$. Then, with $\tilde{Q} := Q_0\varphi_0\gamma$ and using $\underline{n}_0 = \varnothing$ we have $\tilde{Q} \mid P_0 \equiv \nu\underline{n}_0.(Q_0\varphi_0\gamma|\bar{P}_0\eta_0\gamma) \to^*\xrightarrow{e_1}\to^*\xrightarrow{e_2}\to^* \cdots \to^*\xrightarrow{e_n}\to^*$. Since $\tilde{Q}$ does not contain events, this implies that $\underline{e}$ is an event π-trace of P_0. This shows the lemma.

It is left to prove the claim. We distinguish the following cases:

- $P = E[M(x).P_1]$ and $fromadv = (c,m)$ and $\operatorname{eval}^{\mathsf{CoSP}} M\eta\mu \cong c$: Then $P' = E[P_1]$ and $\varphi' = \varphi$ and $\underline{n}' = \underline{n}$ and $\eta' = \eta(x := m)$. Furthermore, since SExec_{P_0} interacts with a Dolev-Yao adversary, $S \vdash c, m$. By Equation 3, there are destructor π-terms D_c, D_m containing only unused names and variables in $\operatorname{dom}\varphi$ such that $c\gamma = \operatorname{eval}^{\pi}(D_c\varphi\gamma)$ and $m\gamma = \operatorname{eval}^{\pi}(D_m\varphi\gamma)$. Since a Dolev-Yao adversary will never derive protocol nonces that have never been sent, we have that only nonces $N \in \mathbf{N}_E \cup \operatorname{range}\mu$ occur in c and in $M\eta\mu$. Hence with (4), from $M\eta\mu = \operatorname{eval}^{\mathsf{CoSP}} M\eta\mu \cong c$ it follows that $M\eta\gamma = M\eta\mu\gamma \approx c\gamma$.
  Pick some $y, z \notin fv(Q') \cup \operatorname{dom}\varphi'$. Let $Q := (let\ y = D_c\ in\ let\ z = D_m\ in\ \overline{y}\langle z\rangle.Q')$. Then $Q\varphi\gamma \to^* \overline{c\gamma}\langle m\gamma\rangle.Q'\varphi\gamma$. Then

$$\begin{aligned}&Q\varphi\gamma \mid P\eta\gamma\\ &= Q\varphi\gamma \mid (E\eta\gamma)[M\eta\gamma(x).P_1\eta\gamma]\\ &\to^* \overline{c\gamma}\langle m\gamma\rangle.Q'\varphi\gamma \mid (E\eta\gamma)[M\eta\gamma(x).P_1\eta\gamma]\\ &\to Q'\varphi\gamma \mid (E\eta\gamma)[P_1\eta\gamma\{m\gamma/x\}].\end{aligned}$$

  Since we maintain the invariant that all bound variables in P are distinct from all other variables in P or $\operatorname{dom}\eta$, we have $x \notin fv(E)$ and $x \notin \operatorname{dom}\eta$. Hence $E\eta\gamma = E\eta'\gamma$ and $P_1\eta\gamma = P_1\eta\{m/x\}\gamma = P_1\eta'\gamma$. Furthermore since $\varphi = \varphi'$ we have $Q'\varphi\gamma = Q'\varphi'\gamma$. Thus $Q'\varphi\gamma \mid (E\eta\gamma)[P_1\eta\gamma\{m\gamma/x\}] = Q'\varphi'\gamma \mid (E\eta'\gamma)[P_1\eta'\gamma] = Q'\varphi'\gamma \mid P'\eta'\gamma$. Thus $Q\varphi\gamma \mid P\eta\gamma \to^* Q'\varphi'\gamma \mid P'\eta'\gamma$ and with $\underline{n} = \underline{n}'$ we have $\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma) \to^* \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma)$.
  Since Q' is valid and $fv(D_c, D_m) \subseteq \operatorname{dom}\varphi$, we have that Q is valid.
- $P = E[\nu a.P_1]$: Then $P' = E[P_1]$, $\varphi' = \varphi$, $\eta = \eta'$, and $\underline{n}' = \underline{n}\|a$ for some $r \in \mathbf{N}_P \setminus \operatorname{range}\mu$, and $\mu' = \mu(a := r)$. Since $a \in \operatorname{dom}\mu' \subseteq \operatorname{dom}\mu_*$, a is used. Since Q' is valid for φ', this implies $a \notin fn(Q')$. Set $Q := Q'$. Then Q is valid for $\varphi = \varphi'$. Since we maintain the invariant that all bound names in P are pairwise distinct and distinct from all other names in P or $\operatorname{dom}\mu$, we have $a \notin fn(E)$ and $a \notin \underline{n}$. Furthermore, by the same invariant, we have $a \notin \operatorname{dom}\mu$. Note that the execution of SExec_{P_0} also maintains the following invariant: Any nonce $N \in \mathbf{N}_P$ occurring (as a subterm) in the range of η or φ is also in the range of μ. (This uses the fact that the Dolev-Yao

adversary cannot derive protocol nonces that have never been sent.) Hence $r \in \mathbf{N}_P \setminus \operatorname{range}\mu$ does not occur in the range of η or φ. Since $\mu^*(a) = r$, $\gamma(r) = a$ and hence using the injectivity of γ, for $N \neq r$ we have $\gamma(N) \neq a$. Thus for all variables x, $x\eta\gamma$ and $x\varphi\gamma$ do not contain a, and hence $a \notin fn(Q\varphi\gamma) \cup fn(E\eta\gamma)$. Together with $r \notin \underline{n}$ we get $\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma) = \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[\nu a.P_1\eta\gamma]) \equiv \nu\underline{n}.\nu a.(Q\varphi\gamma \mid (E\eta\gamma)[P_1\eta\gamma]) = \nu\underline{n}'.(Q'\varphi'\gamma \mid (E\eta'\gamma)[P_1\eta'\gamma]) = \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma)$.

- $P = E[\overline{M_1}\langle N\rangle.P_1][M_2(x).P_2]$ with $\text{eval}^{\mathsf{CoSP}}\, M_1\eta\mu \cong \text{eval}^{\mathsf{CoSP}}\, M_2\eta\mu$: Then $P' = E[P_1][P_2]$ and $\underline{n}' = \underline{n}$ and $\varphi' = \varphi$ and $\eta' = \eta(x := t)$ with $t := \text{eval}^{\mathsf{CoSP}}\, N\eta\mu$. Since we maintain the invariant that all bound variables in P are distinct from all other variables in P or $\operatorname{dom}\eta$, we have $x \notin fv(E) \cup fv(P_1)$ and $x \notin \operatorname{dom}\eta$. Hence $E\eta\gamma = E\eta'\gamma$ and $P_1\eta\gamma = P_1\eta'\gamma$. Furthermore, we have $P_2\eta\gamma\{N\eta\gamma/x\} = P_2\eta\gamma\{N\eta\mu\gamma/x\} = P_2\eta\{N\eta\mu/x\}\gamma = P_2\eta\{t/x\}\gamma = P_2\eta'\gamma$. Since a Dolev-Yao adversary will never derive protocol nonces that have never been sent, we have that only nonces $N \in \mathbf{N}_E \cup \operatorname{range}\mu_*$ occur in $M_1\eta\mu$ and $M_2\eta\mu$. With $M_1\eta\mu = \text{eval}^{\mathsf{CoSP}}\, M_1\eta\mu \cong \text{eval}^{\mathsf{CoSP}}\, M_2\eta\mu = M_2\eta\mu$ and (4) we get $M_1\eta\gamma = M_1\eta\mu\gamma \approx M_2\eta\mu\gamma = M_2\eta\gamma$.
  Hence with $Q := Q'$, we have
$$\begin{aligned}&\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma)\\ &= \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[\overline{M_1\eta\gamma}\langle N\eta\gamma\rangle.P_1\eta\gamma][M_2\eta\gamma(x).P_2\eta\gamma])\\ &\to \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[P_1\eta\gamma][P_2\eta\gamma\{N\eta\gamma/x\}])\\ &= \nu\underline{n}'.(Q\varphi'\gamma \mid (E\eta'\gamma)[P_1\eta'\gamma][P_2\eta'\gamma])\\ &= \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma).\end{aligned}$$
  Since Q' is valid for $\varphi' = \varphi$, $Q = Q'$ is valid for φ.
- $P = E[let\ x = D\ in\ P_1\ else\ P_2]$ and $\text{eval}^{\mathsf{CoSP}}(D\eta\mu) = \bot$: Then $P' = E[P_2]$ and $\varphi' = \varphi$, and $\underline{n}' = \underline{n}$ and $\eta' = \eta$. Set $Q := Q'$. Then Q is valid for $\varphi = \varphi'$. By (2), $\text{eval}^{\pi}(D\eta\gamma) = \bot$. Hence
$$\begin{aligned}&\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma)\\ &= \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[let\ x = D\eta\gamma\ in\ \dots\ else\ P_2\eta\gamma])\\ &\to \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[P_2\eta\gamma])\\ &= \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma).\end{aligned}$$
- $P = E[let\ x = D\ in\ P_1\ else\ P_2]$ and $\text{eval}^{\mathsf{CoSP}}(D\eta\mu) \neq \bot$: Then $P' = E[P_1]$ and $\varphi' = \varphi$ and $\underline{n}' = \underline{n}$ and $\eta' = \eta(x := \text{eval}^{\mathsf{CoSP}}\, D\eta\mu)$. Set $Q := Q'$. Then Q is valid for $\varphi = \varphi'$. By (2), $t := \text{eval}^{\pi}(D\eta\gamma) = \text{eval}^{\mathsf{CoSP}}(D\eta\mu)\eta\gamma \neq \bot$. Since we maintain the invariant that all bound variables in P are distinct from all other variables in P or $\operatorname{dom}\eta$, we have $x \notin fv(E)$ and $x \notin \operatorname{dom}\eta$. Hence $P\eta\gamma = (E\eta\gamma)[let\ x = D\eta\gamma\ in\ P_1\eta\gamma\ else\ \dots] \to (E\eta\gamma)[P_1\eta\gamma\{t/x\}])$. Furthermore, $P_1\eta\gamma\{t/x\} = P_1\eta\gamma\{\text{eval}^{\mathsf{CoSP}}(D\eta\mu)\eta\gamma/x\} = P_1\{\text{eval}^{\mathsf{CoSP}}(D\eta\mu)/x\}\eta\gamma = P_1\eta'\gamma$. Since $x \notin fv(E)$, $(E\eta\gamma)[P_1\eta'\gamma] = E[P_1]\eta'\gamma = P'\eta'\gamma$. Hence $P\eta\gamma \to P'\eta'\gamma$. Since $Q = Q'$, $\varphi = \varphi'$, and $\underline{n} = \underline{n}'$, it follows that $\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma) \to \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma)$.
- $P = E[event(e).P_1]$: Then $P' = E[P_1]$ and $\varphi' = \varphi$ and $\underline{n}' = \underline{n}$ and $\eta' = \eta$ and the event e is raised. Let $Q := Q'$. Then Q is valid for φ. We have $\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma) = \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[event(e).P_1\eta\gamma]) \xrightarrow{e} \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[P_1\eta\gamma]) = \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma)$.
- $P = E[!P_1]$: Then $P' = E[!P_1 \mid \tilde{P}_1]$ for some $\tilde{P}_1 \equiv P_1$ and $\varphi' = \varphi$ and $\underline{n}' = \underline{n}$ and $\eta' = \eta$. Set $Q := Q'$. Then Q is valid for $\varphi = \varphi'$. Hence
$$\begin{aligned}&\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma)\\ &= \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[!P_1\eta\gamma])\\ &\to \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[P_1\eta\gamma \mid !P_1\eta\gamma]\\ &\equiv \nu\underline{n}.(Q\varphi\gamma \mid (E\eta\gamma)[\tilde{P}_1\eta\gamma \mid !P_1\eta\gamma]\\ &= \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma).\end{aligned}$$

- $P = E[\overline{M}\langle N\rangle.P_1]$ with $t'_M := \mathit{fromadv} \cong t_M$ and $\mathit{toadv} = t_N$ where $t_M := \mathrm{eval}^{\mathsf{CoSP}}\, M\eta\mu$ and $t_N := \mathrm{eval}^{\mathsf{CoSP}}\, N\eta\mu$: Then $P' = E[P_1]$ and $S' = S\|t_N$ and $\varphi' = \varphi(x_{n+1} := t_N)$ where $x_{n+1} \notin \operatorname{dom}\varphi$ is unused and $\underline{n}' = \underline{n}$ and $\eta' = \eta$. Since t'_M was sent by the adversary, $S \vdash t'_M$. By (3), there is a destructor π-term D_M containing only unused names and variables in $\operatorname{dom}\varphi$ such that $t'_M\gamma = \mathrm{eval}^\pi(D_M\varphi\gamma)$.
  Since a Dolev-Yao adversary will never derive protocol nonces that have never been sent, we have that only nonces $N \in \mathbf{N}_E \cup \operatorname{range}\mu$ occur in t'_M and $M\eta\mu$. Hence with (4), from $t'_M \cong t_M$ it follows that $M\eta\gamma = M\eta\mu\gamma = t_M\gamma \approx t'_M\gamma$.
  Pick $y \notin \mathit{fv}(Q') \cup \operatorname{dom}\varphi'$. Let $Q := (\mathit{let}\ y = D_M\ \mathit{in}\ y(x_{n+1}).Q')$. Then $Q\varphi\gamma \to t'_M\gamma(x_{n+1}).Q'\varphi\gamma$. Then

$$\begin{aligned}
Q\varphi\gamma \mid P\eta\gamma &= Q\varphi\gamma \mid (E\eta\gamma)[\overline{M\eta\gamma}\langle N\eta\gamma\rangle.P_1\eta\gamma] \\
&\to t'_M\gamma(x_{n+1}).Q'\varphi\gamma \mid (E\eta\gamma)[\overline{M\eta\gamma}\langle N\eta\gamma\rangle.P_1\eta\gamma] \\
&\to Q'\varphi\gamma\{N\eta\gamma/x_{n+1}\} \mid (E\eta\gamma)[P_1\eta\gamma] \\
&= Q'\varphi\{N\eta\mu/x_{n+1}\}\gamma \mid P'\eta\gamma
\end{aligned}$$

  Since $x_{n+1} \notin \operatorname{dom}\varphi$, $Q'\varphi\{N\eta\mu/x_{n+1}\}\gamma = Q'\varphi'\gamma$. Hence $Q\varphi\gamma \mid P\eta\gamma \to^* Q'\varphi'\gamma \mid P'\eta\gamma = Q'\varphi'\gamma \mid P'\eta'\gamma$. Since $\underline{n} = \underline{n}'$ we have $\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma) \to^* \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma)$.
  Since Q' is valid, Q does not contain events, and its free names are unused names, and $\mathit{fv}(Q) \subseteq \operatorname{dom}\varphi' = \operatorname{dom}\varphi \cup \{x_{n+1}\}$. Since x_{n+1} is bound on top level in Q, $x_{n+1} \notin \mathit{fv}(Q)$, thus $\mathit{fv}(Q) \subseteq \operatorname{dom}\varphi$. Hence Q is valid.
- In all other cases we have $P = P'$, $\varphi = \varphi'$, $\eta = \eta'$, and $\underline{n} = \underline{n}'$. Hence with $Q := Q'$, we have $\nu\underline{n}.(Q\varphi\gamma \mid P\eta\gamma) = \nu\underline{n}'.(Q'\varphi'\gamma \mid P'\eta'\gamma)$.

□

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

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

If a closed process P_0 symbolically satisfies a π-trace property $\wp$, and $\Pi_{P_0} \in P$, then P_0 computationally satisfies $\wp$.

Proof. Assume that P_0 symbolically satisfies $\wp$. By Lemma 10, SExec_{P_0} satisfies $\wp$. By Lemma 8, Π_{P_0} symbolically satisfies $\mathit{events}^{-1}(\wp)$. Furthermore, since $\wp$ is an efficiently decidable, prefix closed set, so is $\mathit{events}^{-1}(\wp)$. So $\mathit{events}^{-1}(\wp)$ is a CoSP-trace property in the sense of Definition 10. From Lemma 9 we have that Π_{P_0} is an efficient protocol. By assumption, the computational implementation of the applied π-calculus is computationally sound; hence (Π_{P_0}, A) computationally satisfies $\mathit{events}^{-1}(\wp)$. Using Lemma 8, we obtain that P_0 computationally satisfies $\wp$. □

4.5 Encryption in the applied π-calculus

Consider an instantiation of our process calculus with constructors and destructors described in Section 3. Assume an implementation A of the constructors or destructors satisfying the implementation conditions given on page 14 in Section 3.

We call a process $\tilde{P}$ key-safe if it has the following grammar: Let x, x_d, k_s stand for different 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 $\tilde{M}, \tilde{N} ::= x \mid a \mid \mathit{pair}(\tilde{M}, \tilde{N}) \mid \tilde{S}$ with $\tilde{S} ::= \mathit{string}_0(\tilde{S}) \mid \mathit{string}_1(\tilde{S}) \mid \mathit{empty}$, the allowed destructor terms are $\tilde{D} ::= \tilde{M} \mid \mathit{isek}(\tilde{D}) \mid \mathit{isenc}(\tilde{D}) \mid D(x_d, \tilde{D}) \mid \mathit{fst}(\tilde{D}) \mid \mathit{snd}(\tilde{D}) \mid \mathit{ekof}(\tilde{D}) \mid \mathit{equals}(\tilde{D}, \tilde{D}) \mid \mathit{isvk}(\tilde{D}) \mid$

$\mathit{issig}(\tilde{D}) \mid \mathit{verify}(\tilde{D}, \tilde{D}) \mid \mathit{vkof}(\tilde{D}) \mid \mathit{unstring}_0(\tilde{D}) \mid \mathit{unstring}_1(\tilde{D})$. The allowed processes are

$$\begin{aligned}\tilde{P}, \tilde{Q} ::= {} & \overline{\tilde{M}}\langle \tilde{N} \rangle.\tilde{P} \mid \tilde{M}(x).\tilde{P} \mid 0 \mid (\tilde{P} \mid \tilde{Q}) \mid !\tilde{P} \mid \nu a.\tilde{P} \mid \\ & \mathit{let}\ x = \tilde{D}\ \mathit{in}\ \tilde{P}\ \mathit{else}\ \tilde{Q} \mid \mathit{event}(e).\tilde{P} \mid \\ & \nu r.\mathit{let}\ x = \mathit{ek}(r)\ \mathit{in}\ \mathit{let}\ x_d = \mathit{dk}(r)\ \mathit{in}\ \tilde{P} \mid \\ & \nu r.\mathit{let}\ x = E(\mathit{isek}(\tilde{D}_1), \tilde{D}_2, r)\ \mathit{in}\ \tilde{P}\ \mathit{else}\ \tilde{Q} \mid \\ & \nu r.\mathit{let}\ x = \mathit{vk}(r)\ \mathit{in}\ \mathit{let}\ x_s = \mathit{sk}(r)\ \mathit{in}\ \tilde{P} \mid \\ & \nu r.\mathit{let}\ x = \mathit{sig}(x_s, \tilde{D}_1, r)\ \mathit{in}\ \tilde{P}\ \mathit{else}\ \tilde{Q}\end{aligned}$$

(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_0 symbolically satisfies a π-trace property $\wp$, then P_0 computationally satisfies $\wp$.*

Proof. Let Π_{P_0} be the protocol corresponding to SExec_{P_0} (as in Section 4). From the definition of key-safe processes, it is easy to see that Π_{P_0} is a key-safe CoSP protocol.

By assumption, A satisfies the implementation conditions given on page 14 in Section 3. Hence by Theorem 2, A is a computationally sound implementation of the symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D}, \vdash)$ defined on page 12 for the class P of key-safe CoSP protocols and hence in particular for the CoSP protocol Π_{P_0}.

Let $\mathbf{M}' := (\mathbf{C}, \mathbf{N}, \mathbf{T}', \mathbf{D}, \vdash')$ where T' is the set of all terms over $\mathbf{C} \cup \mathbf{D}$ and $\vdash'$ is the reduction relation defined by the rules in Figure 3.

From the definition of key-safe processes, it is easy to see that Π_{P_0} is T-conform in the sense of Lemma 1. Hence, by Lemma 1, A is a computationally sound implementation of the symbolic model $\mathbf{M}'$ for the protocol Π_{P_0}. By Theorem 3 it follows that if P_0 symbolically satisfies $\wp$, then P_0 computationally satisfies $\wp$. $\square$

Analysis of Needham-Schroeder-Lowe. The Needham-Schroeder-Lowe protocol can be written as follows in our calculus (we use syntactic sugar (x, y) for $\mathit{pair}(x, y)$):

$$\begin{aligned}A := {} & !\mathit{net}(\mathit{ek}_X). \\ & \mathit{if}\ \mathit{ek}_X = \mathit{ek}_B\ \mathit{then}\ \mathit{event}(\mathit{beginAB}).A'\ \mathit{else}\ A' \\ A' := {} & \nu n_A.\overline{\mathit{net}}\langle n_A \rangle.\mathit{net}(c).\mathit{let}\ m = D(\mathit{dk}_A, c)\ \mathit{in} \\ & \mathit{if}\ n_A = \mathit{fst}(m)\ \mathit{then}\ \mathit{if}\ \mathit{ek}_X = \mathit{snd}(\mathit{snd}(m))\ \mathit{then} \\ & \nu r_2.\mathit{let}\ c' = E(\mathit{isek}(\mathit{ek}_X), \mathit{fst}(\mathit{snd}(m)), r_2)\ \mathit{in} \\ & \overline{\mathit{net}}\langle c' \rangle.0 \\ B := {} & !\mathit{net}(\mathit{ek}_X).\mathit{net}(n_A).\nu n_B. \\ & \nu r_1.\mathit{let}\ c = E(\mathit{isek}(\mathit{ek}_X), (n_A, (n_B, \mathit{ek}_B)), r_1)\ \mathit{in} \\ & \overline{\mathit{net}}\langle c \rangle.\mathit{net}(m).\mathit{if}\ n_B = D(\mathit{dk}_B, m)\ \mathit{then} \\ & \mathit{if}\ \mathit{ek}_X = \mathit{ek}_A\ \mathit{then}\ \mathit{event}(\mathit{endAB}) \\ P := {} & \nu r_A.\mathit{let}\ \mathit{ek}_A = \mathit{dk}(r_A)\ \mathit{in}\ \mathit{let}\ \mathit{dk}_A = \mathit{dk}(r_A)\ \mathit{in}\ \overline{\mathit{net}}\langle \mathit{ek}_A \rangle. \\ & \nu r_B.\mathit{let}\ \mathit{ek}_B = \mathit{dk}(r_B)\ \mathit{in}\ \mathit{let}\ \mathit{dk}_B = \mathit{dk}(r_B)\ \mathit{in}\ \overline{\mathit{net}}\langle \mathit{ek}_B \rangle. \\ & (A \mid B).\end{aligned}$$

We model the participants A and B as processes with an unbounded number of sessions that perform authentications with arbitrary participants (the adversary may control with which communication partner an authentication is performed by sending a public key ek_X over the public channel net). If A believes to perform an authentication with B ($\mathit{ek}_X = \mathit{ek}_B$), it raises the event $\mathit{beginAB}$. If B believes

```
fun E/3. fun ek/1. fun dk/1.
fun sig/3. fun vk/1. fun sk/1.
fun pair/2. fun garbage/1. fun garbageEnc/2. fun garbageSig/2.
fun string0/1. fun string1/1. fun empty/0.
reduc D(dk(t1),E(ek(t1),m,t2)) = m.
reduc isek(ek(t)) = ek(t).
reduc isenc(E(ek(t1),t2,t3)) = E(ek(t1),t2,t3);
      isenc(garbageEnc(ek(t1),t2)) = garbageEnc(ek(t1),t2).
reduc fst(pair(x,y)) = x.
reduc snd(pair(x,y)) = y.
reduc ekof(E(ek(t1),m,t2)) = ek(t1);
      ekof(garbageEnc(t1,t2)) = t1.
reduc equals(x,x) = x.
reduc verify(vk(t1),sig(sk(t1),t2,t3)) = t2.
reduc issig(sig(sk(t1),t2,t3)) = sig(sk(t1),t2,t3);
      issig(garbageSig(t1,t2)) = garbageSig(t1,t2).
reduc vkof(sig(sk(t1),t2,t3)) = vk(t1);
      vkof(garbageSig(t1,t2)) = garbageSig(t1,t2).
reduc isvk(vk(t1)) = vk(t1).
reduc unstring0(string0(s)) = s.
reduc unstring1(string0(s)) = s.

query evinj:endAB() ==> evinj:beginAB().

let A = !in(net,ekX); if ekX=ekB then event beginAB(); A' else A'.
let A' = new nA; out(net,nA); new r2; in(net,c); let m=D(dkA,c) in
         if nA=fst(m) then if ekX=snd(snd(m)) then
         let c'=E(isek(ekX),fst(snd(m)),r2) in out(net,c').
let B = !in(net,ekX); in(net,nA); new nB;
        new r1; let c=E(isek(ekX), pair(nA,pair(nB,ekB)), r1) in
        out(net,c); in(net,c'); if nB=D(dkB,c') then if ekX=ekA then event endAB().
process new rA; let ekA=ek(rA) in let dkA=dk(rA) in out(net,ekA);
        new rB; let ekB=ek(rB) in let dkB=dk(rB) in out(net,ekB); A|B
```

Figure 5: Needham-Schroeder-Lowe in ProVerif syntax

to have completed an authentication with A, it raises the event *endAB*. Entity authentication can be expressed by requiring that every *endAB* event is preceded by a *beginAB* event. The process P describing the whole protocol is an encryption-safe process (if r_1, r_2, r_A, r_B are declared as randomness names and dk_A, dk_B as secret key variables).

We can encode the processes and the equational theory in ProVerif as shown in Figure 5. Note that we moved νr_2 in A' up in front of the input $net(c)$. Obviously, this leads to an equivalent process (in terms of event traces), but it helps ProVerif to terminate.[11] ProVerif verifies the entity authentication property with no noticeable delay.

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 π-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 [45, 32] 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 [20] are hence particularly promising targets for this future work.

References

[1] M. Abadi, M. Baudet, and B. Warinschi. Guessing attacks and the computational soundness of static equivalence. In *Proc. 9th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 3921 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2006.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. 28th Symposium on Principles of Programming Languages (POPL)*, pages 104–115, 2001.

[3] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 36–47, 1997.

[4] Martín Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 82–94, 2001.

[5] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[6] Pedro Adão and Cédric Fournet. Cryptographically sound implementations for communicating processes. In *Proc. 32nd International Confererence on Automata, Languages and Programming (ICALP)*, pages 83–94, 2006.

[11] In general, when ProVerif does not terminate, it is helpful to move all restrictions upwards as far as possible.

[7] Michael Backes, Matthias Berg, and Dominique Unruh. A formal language for cryptographic pseudocode. In *Proc. 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 5330 of *Lecture Notes in Computer Science*, pages 353–376, 2008.

[8] Michael Backes and Peeter Laud. Computationally sound secrecy proofs by mechanized flow analysis. In *Proc. 13th ACM Conference on Computer and Communications Security*, 2006.

[9] Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004.

[10] Michael Backes and Birgit Pfitzmann. Relating symbolic and cryptographic secrecy. In *Proc. 26th IEEE Symposium on Security & Privacy*, pages 171–182, 2005. Extended version in IACR Cryptology ePrint Archive 2004/300.

[11] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, `http://eprint.iacr.org/`.

[12] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. 8th European Symposium on Research in Computer Security (ESORICS)*, volume 2808 of *Lecture Notes in Computer Science*, pages 271–290. Springer, 2003.

[13] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Reactively secure signature schemes. *International Journal of Information Security*, 4(4):242–252, 2005.

[14] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Limits of the Reactive Simulatability/UC of Dolev-Yao models with hashes. In *Proc. 11th European Symposium on Research in Computer Security (ESORICS)*, Lecture Notes in Computer Science. Springer, 2006. Preliminary version in IACR Cryptology ePrint Archive 2006/068, Feb. 2006, `http://eprint.iacr.org/`.

[15] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

[16] Michael Backes and Dominique Unruh. Computational soundness of symbolic zero-knowledge proofs against active attackers. In *21st IEEE Computer Security Foundations Symposium, CSF 2008*, pages 255–269, June 2008. Preprint on IACR ePrint 2008/152.

[17] Gilles Barthe, Benjamin Gregoire, Romain Janvier, and Santiago Zanella Beguelin. Formal certification of code-based cryptographic proofs. IACR ePrint Archive, August 2007. Online available at `http://eprint.iacr.org/2007/314`.

[18] David Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 2004.

[19] M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 652–663. Springer, 2005.

[20] Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. Refinement types for secure implementations. In *Proc. 21st IEEE Security Foundations Symposium (CSF)*, pages 17–32, 2008.

[21] Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW)*, pages 82–96, 2001.

[22] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 86–100, 2004.

[23] Bruno Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. 27th IEEE Symposium on Security & Privacy*, pages 140–154, 2006.

[24] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 75:3–51, 2008. Online available at `http://www.di.ens.fr/~blanchet/publications/BlanchetAbadiFournetJLAP07.pdf`.

[25] Bruno Blanchet and David Pointcheval. Automated security proofs with sequences of games. In *Advances in Cryptology: CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 2006.

[26] Emmanuel Bresson, Yassine Lakhnech, Laurent Mazaré, and Bogdan Warinschi. A generalization of DDH with applications to protocol analysis and computational soundness. In *Advances in Cryptology - CRYPTO 2007*, volume 4622 of *LNCS*, pages 482–499. Springer, 2007.

[27] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version in Cryptology ePrint Archive, Report 2000/67, `http://eprint.iacr.org/`.

[28] Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. 3rd Theory of Cryptography Conference (TCC)*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2006.

[29] Ran Canetti, Daniele Micciancio, and Omer Reingold. Perfectly one-way probabilistic hash functions. In *Proc. 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 131–140, 1998.

[30] Hubert Comon-Lundh. About models of security protocols. Unpublished manuscript. An abstract has been published in [31], 2008.

[31] Hubert Comon-Lundh. About models of security protocols (abstract). In Ramesh Hariharan, Madhavan Mukund, and V Vinay, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2008)*, Dagstuhl, Germany, 2008. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. Online available at `http://drops.dagstuhl.de/opus/volltexte/2008/1766/`.

[32] Hubert Comon-Lundh and Véronique Cortier. Computational soundness of observational equivalence. In *Proc. ACM Conference on Computer and Communications Security*, pages 109–118, 2008.

[33] Ricardo Corin and J. den Hartog. A probabilistic hoare-style logic for game-based cryptographic proofs. In *Proc. 33rd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4052 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2006.

[34] Véronique Cortier, Steve Kremer, Ralf Küsters, and Bogdan Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In *Proc. 26th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 176–187, 2006.

[35] Véronique Cortier and Bogdan Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP)*, pages 157–171, 2005.

[36] Anupam Datta, Ante Derek, John Mitchell, Vitalij Shmatikov, and Matthieu Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2005.

[37] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[38] Shimon Even and Oded Goldreich. On the security of multi-party ping-pong protocols. In *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 34–39, 1983.

[39] Flavio D. Garcia and Peter van Rossum. Sound and complete computational interpretation of symbolic hashes in the standard model. *Theor. Comput. Sci.*, 394(1-2):112–133, 2008.

[40] Jonathan Herzog, Moses Liskov, and Silvio Micali. Plaintext awareness via key registration. In *Advances in Cryptology: CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 548–564. Springer, 2003.

[41] Russell Impagliazzo and Bruce M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–381, 2003.

[42] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *Proc. 14th European Symposium on Programming (ESOP)*, pages 172–185, 2005.

[43] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Computational soundness of symbolic analysis for protocols using hash functions. *Electronic Notes in Theoretical Computer Science*, 186:121 – 139, 2007. Proceedings of the First Workshop in Information and Computer Security (ICS 2006).

[44] Richard Kemmerer, Catherine Meadows, and Jon Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[45] Steve Kremer and Laurent Mazaré. Adaptive soundness of static equivalence. In *Proc. 12th European Symposium On Research In Computer Security (ESORICS)*, pages 610–625, 2007.

[46] Peeter Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.

[47] Peeter Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.

[48] Peeter Laud. Secrecy types for a simulatable cryptographic library. In *Proc. 12th ACM Conference on Computer and Communications Security*, pages 26–35, 2005.

[49] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[50] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[51] Michael Merritt. *Cryptographic Protocols.* PhD thesis, Georgia Institute of Technology, 1983.

[52] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.

[53] J. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 725–733, 1998.

[54] D. Nowak. A framework for game-based security proofs. IACR Cryptology ePrint Archive 2007/199, 2007. `http://eprint.iacr.org/`.

[55] Lawrence Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.

[56] Steve Schneider. Security properties and CSP. In *Proc. 17th IEEE Symposium on Security & Privacy*, pages 174–187, 1996.

[57] Christoph Sprenger, Michael Backes, David Basin, Birgit Pfitzmann, and Michael Waidner. Cryptographically sound theorem proving. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW)*, pages 153–166, 2006.