Computational Soundness Results for ProVerif

Bridging the Gap from Trace Properties to Uniformity

Michael Backes[1,2], Esfandiar Mohammadi[1], and Tim Ruffing[3]

[1] CISPA, Saarland University, Germany
[2] Max Planck Institute for Software Systems (MPI-SWS), Germany
[3] MMCI, Saarland University, Germany

Abstract. Dolev-Yao models of cryptographic operations constitute the foundation of many successful verification tools for security protocols, such as the protocol verifier ProVerif. Research over the past decade has shown that many of these symbolic abstractions are computationally sound, i.e., the absence of attacks against the abstraction entails the security of suitable cryptographic realizations. Most of these computational soundness (CS) results, however, are restricted to trace properties such as authentication, and the few promising results that strive for CS for the more comprehensive class of equivalence properties, such as strong secrecy or anonymity, either only consider a limited class of protocols or are not amenable to fully automated verification.

In this work, we identify a general condition under which CS for trace properties implies CS for uniformity of bi-processes, i.e., the class of equivalence properties that ProVerif is able to verify for the applied π-calculus. As a case study, we show that this general condition holds for a Dolev-Yao model that contains signatures, public-key encryption, and corresponding length functions. We prove this result in the CoSP framework (a general framework for establishing CS results). To this end, we extend the CoSP framework to equivalence properties, and we show that an existing embedding of the applied π-calculus to CoSP can be re-used for uniform bi-processes. On the verification side, as analyses in ProVerif with symbolic length functions often do not terminate, we show how to combine the recent protocol verifier APTE with ProVerif. As a result, we establish a computationally sound automated verification chain for uniformity of bi-processes in the applied π-calculus that use public-key encryption, signatures, and length functions.

1 Introduction

Manual security analyses of protocols that rely on cryptographic operations are complex and error-prone. As a consequence, research has strived for the automation of such proofs soon after the first protocols were developed. To eliminate the inherent complexity of cryptographic operations that verification tools are struggling to deal with, cryptographic operations have been abstracted as symbolic terms that obey simple cancelation rules, so-called Dolev-Yao models [1,2]. A variety of automated verification tools have been developed based on this

M. Abadi and S. Kremer (Eds.): POST 2014, LNCS 8414, pp. 42–62, 2014.

abstraction, and they have been successfully used for reasoning about various security protocols [3,4,5,6,7,8,9,10]. In particular, a wide range of these tools is capable of reasoning about the more comprehensive class of *equivalence properties*, such as strong secrecy and anonymity, which arguably is the most important class of security properties for privacy-preserving protocols.

Research over the past decade has shown that many of these Dolev-Yao models are computationally sound, i.e., the absence of attacks against the symbolic abstraction entails the security of suitable cryptographic realizations. Most of these computational soundness (CS) results against active attacks, however, have been specific to the class of trace properties [11,12,13,14,15,16,17,18,19,20,21], which is only sufficient as long as strong notions of privacy are not considered, e.g., in particular for establishing various authentication properties. Only few CS results are known for the class of equivalence properties against active attackers, which are restricted in of the following three ways: either they are restricted to a small class of simple processes, e.g., processes that do not contain private channels and abort if a conditional fails [22,23,24], or they rely on non-standard abstractions for which it is not clear how to formalize any equivalence property beyond the secrecy of payloads [25,26,27], such as anonymity properties in protocols that encrypt different signatures, or existing automated tool support is not applicable [28,29]. We are thus facing a situation where CS results, despite tremendous progress in the last decade, still fall short in comprehensively addressing the class of equivalence properties and protocols that state-of-the-art verification tools are capable to deal with. Moreover, it is unknown to which extent existing results on CS for trace properties can be extended to achieve more comprehensive CS results for equivalence properties.

Our Contribution. In this work, we close this gap by providing the first result that allows to leverage existing CS results for trace properties to CS results for an expressive class of equivalence properties: the uniformity of bi-processes in the applied π-calculus. Bi-processes are pairs of processes that differ only in the messages they operate on but not in their structure; a bi-process is uniform if for all surrounding contexts, i.e., all interacting attackers, both processes take the same branches. Blanchet, Abadi, and Fournet [7] have shown that uniformity already implies observational equivalence. Moreover, uniformity of bi-processes corresponds precisely to the class of properties that the state-of-the-art verification tool ProVerif [30] is capable to analyze, based on a Dolev-Yao model in the applied π-calculus. In contrast to previous work dealing with equivalence properties, we consider bi-protocols that use the fully fledged applied π-calculus, in particular including private channels and non-determinate processes.

To establish this main result of our paper, we first identify the following general condition for Dolev-Yao models: "whenever a computational attacker can distinguish a bi-process, there is a test in the Dolev-Yao model that allows to successfully distinguish the bi-process." We say that Dolev-Yao models with this property *allow for self-monitoring*. We show that if a specific Dolev-Yao model fulfills this property, then there is for every bi-process a so-called *self-monitor*, i.e., a process that performs all relevant tests that the attacker could perform

on the two processes of the bi-process, and that raises an exception if of these tests in the symbolic model distinguishes the bi-process. We finally show that whenever a Dolev-Yao model allows for self-monitoring, CS for uniformity of bi-processes automatically holds whenever CS for trace properties has already been established. This result in particular allows for leveraging existing CS results for trace properties to more comprehensive CS results for uniformity of bi-processes, provided that the Dolev-Yao model can be proven to allow for self-monitoring.

We exemplarily show how to construct a self-monitor for a symbolic model that has been recently introduced and proven to be computationally sound for trace properties by Backes, Malik, and Unruh [31]. This symbolic model contains signatures and public-key encryption and allows to freely send and receive decryption keys. To establish that the model allows for self-monitoring, we first extend it using the common concept of a length function (without a length function, CS for uniformity of bi-processes and hence the existence of self-monitors trivially cannot hold, since encryptions of different lengths are distinguishable in general), and we show that this extension preserves the existing proof of CS for trace properties. Our main result in this paper then immediately implies that this extended model satisfies CS for uniformity of bi-processes.

We moreover investigate how computationally sound automated analyses can still be achieved in those frequent situations in which ProVerif does not manage to terminate whenever the Dolev-Yao model supports a length function. We proceed in two steps: first, we feed a stripped-down version of the protocol without length functions in ProVerif; ProVerif then yields a result concerning the uniformity of bi-processes, but only for this stripped-down protocol. Second, we analyze the original protocol using the APTE tool by Cheval, Cortier, and Plet [32], which is specifically tailored to length functions. This yields a result for the original protocol but only concerning trace equivalences. We show that both results can be combined to achieve uniformity of bi-processes for the original protocol, and thus a corresponding CS result for uniformity of bi-processes.

We present the first general framework for CS for equivalence properties, by extending the CoSP framework: a general framework for symbolic analysis and CS results for trace properties [15]. CoSP decouples the CS of Dolev-Yao models from the calculi, such as the applied π-calculus or RCF: proving x cryptographic Dolev-Yao models sound for y calculi only requires $x + y$ proofs (instead of $x \cdot y$). We consider this extension to be of independent interest. Moreover, we prove the existence of an embedding from the applied π-calculus to the extended CoSP framework that preserves the uniformity of bi-processes, using a slight variation of the already existing embedding for trace properties.

2 Equivalence Properties in the CoSP Framework

The results in this work are formulated within CoSP [15], a framework for conceptually modular CS proofs that decouples the treatment of the cryptographic primitives from the treatment of the calculi. Several calculi such as the

applied π-calculus [15] and RCF [33] (a core calculus of F#) can be embedded into CoSP and combined with CS results for cryptographic primitives.

The original CoSP framework is only capable of handling CS with respect to trace properties, i.e., properties that can be formulated in terms of a single trace. Typical examples include the non-reachability of a certain "bad" protocol state, in that the attacker is assumed to have succeeded (e.g., the protocol never reveals a secret), or correspondence properties such as authentication (e.g., a user can access a resource only after proving a credential). However, many interesting protocol properties cannot be expressed in terms of a single trace. For instance, strong secrecy or anonymity are properties that are, in the computational setting, usually formulated by means of a game in which the attacker has to distinguish between several scenarios.

To be able to handle the class of equivalence properties, we extend the CoSP framework to support equivalence properties. First, we recall the basic definitions of the original framework. Dolev-Yao models are formalized as follows in CoSP.

Definition 1 (Symbolic Model). *A* symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ *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* $\mathbf{T}$*. We require that* $\mathbf{N} = \mathbf{N_E} \uplus \mathbf{N_P}$ *for countable infinite sets* $\mathbf{N_P}$ *of protocol nonces and attacker nonces* $\mathbf{N_E}$*.*

We write $\underline{t}$ for a list $t_1, \ldots, t_n$ if n is clear from the context. A *constructor* C/n is a symbol with (possibly zero) arity. A *nonce* N is a symbol with zero arity. 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/n of arity 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 $D(\underline{t}) = \bot$.

To unify notation, we define for every constructor or destructor $F/n \in \mathbf{D} \cup \mathbf{C}$ and every nonce $F \in \mathbf{N}$ the partial function $eval_F : \mathbf{T}^n \to \mathbf{T}$, where $n = 0$ for a nonce, as follows: If F is a constructor, $eval_F(\underline{t}) := F(\underline{t})$ if $F(\underline{t}) \in \mathbf{T}$ and $eval_F(\underline{t}) := \bot$ otherwise. If F is a nonce, $eval_F() := F$. If F is a destructor, $eval_F(\underline{t}) := F(\underline{t})$ if $F(\underline{t}) \neq \bot$ and $eval_F(\underline{t}) := \bot$ otherwise.

Protocols. In CoSP, a protocol is represented as a tree. Each node in this tree corresponds to an action in the protocol: *computation nodes* are used for drawing fresh nonces, applying constructors, and applying destructors; *input* and *output nodes* are used to send and receive messages; *control nodes* are used for allowing the attacker to schedule the protocol.

Definition 2 (CoSP Protocol). *A* CoSP protocol I *is a tree of infinite depth with a distinguished root and labels on both edges and nodes. Each node has a unique identifier* ν *and one of the following types:*

- Computation nodes *are annotated with a constructor, nonce or destructor* F/n *together with the identifiers of* n *(not necessarily distinct) nodes; we call these annotations* references*, and we call the referenced nodes* arguments*. Computation nodes have exactly two successors; the corresponding edges are labeled with* `yes` *and* `no`*, respectively.*
- Input nodes *have no annotations. They have exactly one successor.*

- Output nodes *have a reference to exactly one node in their annotations. They have exactly one successor.*
- Control nodes *are annotated with a bitstring l. They have at least one and up to countably many successors; the corresponding edges are labeled with distinct bitstrings l'. (We call l the* out-metadata *and l' the* in-metadata.*)*

We assume that the annotations are part of the node identifier. A node ν can only reference other nodes ν' on the path from the root to ν; in this case ν' must be a computation node or input node. If ν' is a computation node, the path from ν' to ν has additionally to go through the outgoing edge of ν' with label `yes`.

Bi-protocols. To compare two variants of a protocol, we consider bi-protocols, which rely on the same idea as bi-processes in the applied π-calculus [7]. Bi-protocols are pairs of protocols that only differ in the messages they operate on.

Definition 3 (CoSP Bi-protocol). *A* CoSP bi-protocol Π *is defined like a protocol but uses bi-references instead of references. A* bi-reference *is a pair $(\nu_{\mathsf{left}}, \nu_{\mathsf{right}})$ of node identifiers of two (not necessarily distinct) nodes in the protocol tree. In the* left protocol $\mathsf{left}(\Pi)$ *the bi-references are replaced by their left components; the* right protocol $\mathsf{right}(\Pi)$ *is defined analogously.*

2.1 Symbolic Indistinguishability

In this section, we define a symbolic notion of indistinguishability. First, we model the capabilities of the symbolic attacker. Operations that the symbolic attacker can perform on terms are defined as follows, including the destruction of already known terms and the creation of new terms.[1]

Definition 4 (Symbolic Operation). *Let $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ be a symbolic model. A* symbolic operation O/n (of arity n) on $\mathbf{M}$ *is a finite tree whose nodes are labeled with constructors from $\mathbf{C}$, destructors from $\mathbf{D}$, nonces from $\mathbf{N}$, and formal parameters x_i with $i \in \{1, \ldots, n\}$. For constructors and destructors, the children of a node represent its arguments (if any). Formal parameters x_i and nonces do not have children.*

We extend the evaluation function to symbolic operations. Given a list of terms $\underline{t} \in \mathbf{T}^n$, the evaluation function $eval_O : \mathbf{T}^n \to \mathbf{T}$ recursively evaluates the tree O starting at the root as follows: The formal parameter x_i evaluates to t_i. A node with $F \in \mathbf{C} \cup \mathbf{N}_E \cup \mathbf{D}$ evaluates according to $eval_F$. If there is a node that evaluates to $\bot$, the whole tree evaluates to $\bot$.

A symbolic execution of a protocol is basically a valid path through the protocol tree. It induces a *view*, which contains the communication with the attacker.

[1] We deviate from the definition in the original CoSP framework [15], where a deduction relation describes which terms the attacker can deduce from the already seen terms. This modification is not essential; all results for trace properties that have been established in the original framework so far are compatible with our definition.

Definition 5 (Symbolic Execution). *Let a symbolic model* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ *and a CoSP protocol* I *be given. A* symbolic view *of the protocol* I *is a (finite) list of triples* (V_i, ν_i, f_i) *with the following conditions:*

For the first triple, we have $V_1 = \varepsilon$, ν_1 *is the root of* I*, and* f_1 *is an empty partial function, mapping node identifiers to terms. For every two consecutive tuples* (V, ν, f) *and* (V', ν', f') *in the list, let* $\underline{\tilde{\nu}}$ *be the nodes referenced by* ν *and define* $\underline{\tilde{t}}$ *through* $\tilde{t}_j := f(\tilde{\nu}_j)$*. We conduct a case distinction on* ν*.*

- ν ***is a computation node with constructor, destructor or nonce*** F*.* *Let* $V' = V$*. If* $m := eval_F(\underline{\tilde{t}}) \neq \bot$*,* ν' *is the* `yes`*-successor of* ν *in* I*, and* $f' = f(\nu := m)$*. If* $m = \bot$*, then* ν' *is the* `no`*-successor of* ν*, and* $f' = f$*.*
- ν ***is an input node.*** *If there exists a term* $t \in \mathbf{T}$ *and a symbolic operation* O *on* $\mathbf{M}$ *with* $eval_O(V_{Out}) = t$*, let* ν' *be the successor of* ν *in* I*,* $V' = V :: (\mathtt{in}, (t, O))$*, and* $f' = f(\nu := t)$*.*
- ν ***is an output node.*** *Let* $V' = V :: (\mathtt{out}, \tilde{t}_1)$*,* ν' *is the successor of* ν *in* I*, and* $f' = f$*.*
- ν ***is a control node with out-metadata*** l*.* *Let* ν' *be the successor of* ν *with the in-metadata* l' *(or the lexicographically smallest edge if there is no edge with label* l'*),* $f' = f$*, and* $V' = V :: (\mathtt{control}, (l, l'))$*.*

Here, V_{Out} *denotes the list of terms in* V *that have been sent at output nodes, i.e., the terms* t *contained in entries of the form* $(\mathtt{out}, t)$ *in* V*. Analogously,* $V_{Out\text{-}Meta}$ *denotes the list of out-metadata in* V *that has been sent at control nodes.*

The set of all symbolic views of I *is denoted by* $\mathsf{SViews}(I)$*. Furthermore,* V_{In} *denotes the partial list of* V *that contains only entries of the form* $(\mathtt{in}, (*, O))$ *or* $(\mathtt{control}, (*, l'))$ *for some symbolic operation* O *and some in-metadata* l'*, where the input term and the out-metadata have been masked with the symbol* $*$*. The list* V_{In} *is called* attacker strategy*. We write* $[V_{In}]_{\mathsf{SViews}(I)}$ *to denote the class of all views* $U \in \mathsf{SViews}(I)$ *with* $U_{In} = V_{In}$*.*

The knowledge of the attacker are the results of all the symbolic tests the attacker can perform on the messages output by the protocol. To define the attacker knowledge formally, we have to pay attention to two important details. First, we concentrate on whether a symbolic operation fails or not, i.e., if it evaluates to $\bot$ or not; we are not interested in the resulting term in case the operation succeeds. The following example illustrates why: suppose the left protocol of a bi-protocol does nothing more than sending a ciphertext c to the attacker, whereas the right protocol sends a different ciphertext c' (with the same plaintext length) to the attacker. Assume that the decryption key is kept secret. This bi-protocol should be symbolically indistinguishable. More precisely, the attacker knowledge in the left protocol should be statically indistinguishable from the attacker knowledge in the right protocol. Recall that $O = x_1$ is the symbolic operation that just returns the first message received by the attacker. If the result of O were part of the attacker knowledge, the knowledge in the left protocol (containing c) would differ from the knowledge in the right protocol (containing c'), which is not what we would like to express. On the other hand, our definition, which only cares about the failure or success of a operation, requires that the symbolic

model contains an operation *equals* to be reasonable. This operation *equals* allows the attacker to test equality between terms: consider the case where the right protocol sends a publicly known term t instead of c', but still of the same length as c. In that case the attacker can distinguish the bi-protocol with the help of the symbolic operation $equals(t, x_1)$.

The second observation is that the definition should cover the fact that the attacker knows which symbolic operation leads to which result. This is essential to reason about indistinguishability: consider a bi-protocol such that the left protocol sends the pair (n, t), but the right protocol sends the pair (t, n), where t is again a publicly known term and n is a fresh protocol nonce. The two protocols do not differ in the terms that the attacker can deduce after their execution; the deducible terms are all publicly known terms as well as n. Still, the protocols are trivially distinguishable by the symbolic operation $equals(O_t, snd(x_1))$ because $equals(O_t, snd((n, t))) \neq \bot$ but $equals(t, snd((t, n))) = \bot$, where *snd* returns the second component of a pair and O_t is a symbolic operation that constructs t.

Definition 6 (Symbolic Knowledge). *Let* $\mathbf{M}$ *be a symbolic model. Given a view* V *with* $|V_{Out}| = n$*, the* full symbolic knowledge function K_V *is a function from symbolic operations on* $\mathbf{M}$ *(see Definition 4) of arity* n *to* $\{\top, \bot\}$*, defined by* $K_V(O) := \bot$ *if* $eval_O(V_{Out}) = \bot$ *and* $K_V(O) := \top$ *otherwise.*

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

Definition 7 (Equivalent Views). *Let two views* V, V' *of the same length be given. We denote their* i*th entry by* V_i *and* V'_i*, respectively.* V *and* V' *are* equivalent *(*$V \sim V'$*), if the following three conditions hold:*

1. *(Same structure)* V_i *is of the form* $(s, \cdot)$ *if and only if* V'_i *is of the form* $(s, \cdot)$ *for some* $s \in \{\texttt{out}, \texttt{in}, \texttt{control}\}$.
2. *(Same out-metadata)* $V_{Out\text{-}Meta} = V'_{Out\text{-}Meta}$.
3. *(Same symbolic knowledge)* $K_V = K_{V'}$.

Finally, we define a bi-protocol to be symbolically indistinguishable if they lead to equivalent views when faced with the same attacker strategy.[2]

Definition 8 (Symbolic Indistinguishability). *Let* $\mathbf{M}$ *be a symbolic model and* P *be a class of bi-protocols on* $\mathbf{M}$*. Given an attacker strategy* V_{In} *(in the sense of Definition 5), a bi-protocol* $\Pi \in \mathrm{P}$ *is* symbolically indistinguishable under V_{In} *if for all views* $V_{\mathsf{left}} \in [V_{In}]_{\mathsf{SViews}(\mathsf{left}(\Pi))}$ *of the left protocol under* V_{In}*, there is a view* $V_{\mathsf{right}} \in [V_{In}]_{\mathsf{SViews}(\mathsf{right}(\Pi))}$ *of the right protocol under* V_{In} *such that* $V_{\mathsf{left}} \sim V_{\mathsf{right}}$*, and vice versa.*

[2] For the sake of convenience, we define CS for bi-protocols. However, our definition can be easily generalized to arbitrary pairs of protocols.

A bi-protocol $\Pi \in \mathrm{P}$ is symbolically indistinguishable*, if Π is indistinguishable under all attacker strategies. We write $\mathsf{left}(\Pi) \approx_s \mathsf{right}(\Pi)$ for the symbolic indistinguishability of Π.*

2.2 Computational Indistinguishability

On the computational side, the constructors and destructors in a symbolic model are realized with cryptographic algorithms, formalized as follows.

Definition 9 (Computational Implementation). *Let $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ be a symbolic model. A* computational implementation of $\mathbf{M}$ *is a family of functions $\mathbf{A} = (A_x)_{x \in \mathbf{C} \cup \mathbf{D} \cup \mathbf{N}}$ such that A_F for $F/n \in \mathbf{C} \cup \mathbf{D}$ is a partial deterministic function $\mathbb{N} \times (\{0,1\}^*)^n \to \{0,1\}^*$, and A_N for $N \in \mathbf{N}$ is a total probabilistic function with domain $\mathbb{N}$ and range $\{0,1\}^*$. The first argument of A_F and A_N represents the security parameter.*

All functions A_F have to be computable in deterministic polynomial time, and all A_N have to be computable in probabilistic polynomial time (ppt).

The computational execution of a protocol is a randomized interactive machine that runs against a ppt attacker $\mathcal{A}$. The transcript of the execution contains essentially the computational counterparts of a symbolic view.

Definition 10 (Computational Challenger). *Let $\mathbf{A}$ be a computational implementation of the symbolic model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ and I be a CoSP protocol. Let $\mathcal{A}$ be a ppt machine and p be a polynomial. For a security parameter k, the* computational challenger $\mathsf{Exec}_{\mathbf{M},\mathbf{A},I,p}(k)$ *is the following interactive machine:*

Initially, let ν be the root of I. Let f and n be empty partial functions from node identifiers to bitstrings and from $\mathbf{N}$ to bitstrings, respectively. Enter a loop and proceed depending on the type of ν:

- *ν **is a computation node with nonce** $N \in \mathbf{N}$. If $n(N) \neq \bot$, let $m' := n(N)$; otherwise sample m' according to $A_N(k)$. Let ν' be the yes-successor of ν. Let $f := f(\nu := m')$, $n := n(N := m')$, and $\nu := \nu'$.*
- *ν **is a computation node with constructor or destructor** F. Let $\underline{\tilde{\nu}}$ be the nodes referenced by ν and $\tilde{m}_j := f(\tilde{\nu}_j)$. Then, $m' := A_F(k, \underline{\tilde{m}})$. If $m' \neq \bot$, then ν' is the* `yes`*-successor of ν, if $m' = \bot$, then ν' is the* `no`*-successor of ν. Let $f := f(\nu := m')$ and $\nu := \nu'$.*
- *ν **is an input node.** Ask the adversary $\mathcal{A}$ for a bitstring m. Let ν' be the successor of ν. Let $f := f(\nu := m)$ and $\nu := \nu'$.*
- *ν **is an output node.** Send $\tilde{m}_1$ to $\mathcal{A}$. Let ν' be the successor of ν, and let $\nu := \nu'$.*
- *ν **is a control node with out-metadata** l. Send l to $\mathcal{A}$. Upon receiving in-metadata 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 $\nu := \nu'$.*

Let len be the number of nodes from the root to ν plus the total length of all bitstrings in the range of f. Stop if $len > p(k)$; otherwise continue the loop. We call V the computational view *of a run.*

Definition 11 (Computational Execution). *The interaction between the challenger* $\mathsf{Exec}_{\mathbf{M},\mathbf{A},\Pi,p}(k)$ *and the adversary* $\mathcal{A}(k)$ *is called the* computational execution*, denoted by* $\langle \mathsf{Exec}_{\mathbf{M},\mathbf{A},\Pi,p}(k) | \mathcal{A}(k) \rangle$*. It stops whenever one of the two machines stops, and the output of* $\langle \mathsf{Exec}_{\mathbf{M},\mathbf{A},\Pi,p}(k) | \mathcal{A}(k) \rangle$ *is the output of* $\mathcal{A}(k)$*.*

Given these definitions, computational indistinguishability for bi-protocols is naturally defined. A bi-protocol is indistinguishable if its challengers are computationally indistinguishable for every ppt attacker $\mathcal{A}$.

Definition 12 (Computational Indistinguishability). *Let* Π *be an efficient*[3] *CoSP bi-protocol and let* $\mathbf{A}$ *be a computational implementation of* $\mathbf{M}$*.* Π *is* computationally indistinguishable *if for all ppt attackers* $\mathcal{A}$ *and for all polynomials* p*, we have that* $\left\langle \mathsf{Exec}_{\mathbf{A},\mathbf{M},\mathsf{left}(\Pi),p}(k) \middle| \mathcal{A}(k) \right\rangle \approx_c \left\langle \mathsf{Exec}_{\mathbf{A},\mathbf{M},\mathsf{right}(\Pi),p}(k) \middle| \mathcal{A}(k) \right\rangle$*, where* $\approx_c$ *denotes computational indistinguishability of distribution ensembles.*

Computational Soundness. Having defined symbolic and computational indistinguishability, we are finally able to relate them. The previous definitions culminate in the definition of CS for equivalence properties. It states that the symbolic indistinguishability of a bi-protocol implies its computational indistinguishability. In other words, it suffices to check the security of the symbolic bi-protocol, e.g., using mechanized protocol verifiers such as ProVerif.

Definition 13 (Computational Soundness for Equivalence Properties). *Let a symbolic model* $\mathbf{M}$ *and a class* P *of efficient bi-protocols be given. An implementation* $\mathbf{A}$ *of* $\mathbf{M}$ *is* computationally sound *for* $\mathbf{M}$ *if for every* $\Pi \in \mathrm{P}$*, we have that* Π *is computationally indistinguishable whenever* Π *is symbolically indistinguishable.*

3 Self-monitoring

In this section, we identify a sufficient condition for symbolic models under which CS for trace properties implies CS for equivalence properties for a class of *uniformity-enforcing protocols*, which correspond to uniform bi-processes in the applied π-calculus. We say that a symbolic model that satisfies this condition *allows for self-monitoring*. The main idea behind self-monitoring is that a symbolic model is sufficiently expressive (and its implementation is sufficiently strong) such that the following holds: whenever a computational attacker can distinguish a bi-process, there is a test in the symbolic model that allows to successfully distinguish the bi-process.

CS for Trace Properties. We first review CS for trace properties. A trace property is a prefix-closed set of node identifiers. We refer to [15] for the precise definition of computational and symbolic satisfiability. CS for trace properties states that all attacks (against trace properties) that can be excluded for the

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

symbolic abstraction can be excluded for the computational implementation as well. Hence, if all the symbolic traces satisfy a certain trace property, then all computational traces satisfy this property as well.

Definition 14 (Computational Soundness for Trace Properties [15]). *A symbolic model* $(\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ *is* computationally sound for trace properties *with respect to an implementation* $\mathbf{A}$ *for a class* P *of efficient protocols if the following holds: for each protocol* $I \in \mathrm{P}$ *and each trace property* $\mathcal{P}$*, if* I *symbolically satisfies* $\mathcal{P}$ *then* I *computationally satisfies* $\mathcal{P}$*.*

Uniformity-enforcing. A bi-protocol is *uniform* if for each symbolic attacker strategy, both its variants reach the same nodes in the CoSP tree, i.e., they never branch differently.[4] Formally, we require that the bi-protocols are *uniformity-enforcing*, i.e., when the left and the right protocol of the bi-protocol Π take different branches, the attacker is informed. Since taking different branches is only visible after a control node is reached, we additionally require that computation nodes are immediately followed by control nodes.

Definition 15 (Uniformity-enforcing). *A class* P *of CoSP bi-protocols is* uniformity-enforcing *if for all bi-protocols* $\Pi \in \mathrm{P}$*:*

1. *Every control node in* Π *has unique out-metadata.*
2. *For every computation node* ν *in* Π *and for every path rooted at* ν*, a control node is reached before an output node.*

All embeddings of calculi the CoSP framework described so far, namely those of the applied π-calculus [15] and RCF [33], are formalized such that protocols written in these calculi fulfill both properties: these embeddings give the attacker a scheduling decision, using a control node, basically after every execution step.

3.1 Bridging the Gap from Trace Properties to Uniformity

The key observation for the connection to trace properties is that, given a bi-protocol Π, some computationally sound symbolic models allow to construct a self-monitor protocol $\mathsf{Mon}(\Pi)$ (not a bi-protocol!) that has essentially the same interface to the attacker as the bi-protocol Π and checks at run-time whether Π would behave uniformly. In other words, non-uniformity of bi-protocols can be formulated as a trace property $\mathtt{bad}$, which the protocol $\mathsf{Mon}(\Pi)$ detects.

The self-monitor $\mathsf{Mon}(\Pi)$ of a bi-protocol Π behaves like one of the two variants of the bi-protocol Π, while additionally simulating the opposite variant such that $\mathsf{Mon}(\Pi)$ itself is able to detect whether Π would be distinguishable. (For instance, one approach to detect whether Π is distinguishable could consist of reconstructing the symbolic view of the attacker in the variant of Π that is not executed by $\mathsf{Mon}(\Pi)$.) At the beginning of the execution of the self-monitor, the attacker chooses if $\mathsf{Mon}(\Pi)$ should basically behave like $\mathsf{left}(\Pi)$ or like $\mathsf{right}(\Pi)$.

[4] We show in Lemma 1 that uniformity of bi-protocols in CoSP corresponds to uniformity of bi-processes in the applied π-calculus.

We denote the chosen variant as $b \in \{\mathsf{left}, \mathsf{right}\}$ and the opposite variant as $\overline{b}$. After this decision, $\mathsf{Mon}(\Pi)$ executes the the b-variant $b(\Pi)$ of the bi-protocol Π, however, enriched with the computation nodes and the corresponding output nodes of the opposite variant $\overline{b}(\Pi)$.[5]

The goal of the self-monitor $\mathsf{Mon}(\Pi)$ is to detect whether the execution of $b(\Pi)$ would be distinguishable from $\overline{b}(\Pi)$ at the current state. If this is the case, $\mathsf{Mon}(\Pi)$ raises the event `bad`, which is the disjunction of two events `bad-branch` and `bad-knowledge`.

The event `bad-branch` corresponds to the case that the left and the right protocol of the bi-protocol Π take different branches. Since uniformity-enforcing protocols have a control node immediately after every computation node (see Definition 15), the attacker can always check whether $b(\Pi)$ and $\overline{b}(\Pi)$ take the same branch. We require (in Definition 17) the existence of a so-called *distinguishing subprotocol* $f_{\texttt{bad-branch},\Pi}$ that checks whether each destructor application in $b(\Pi)$ succeeds if and only if it succeeds in $\overline{b}(\Pi)$; if not, the distinguishing subprotocol $f_{\texttt{bad-branch},\Pi}$ raises `bad-branch`.

The event `bad-knowledge` captures that the messages sent by $b(\Pi)$ and $\overline{b}(\Pi)$ (via output nodes, i.e., not the out-metadata) are distinguishable. This distinguishability is only detectable by a protocol if the constructors and destructors, which are available to both the protocol and the symbolic attacker, capture all possible tests. We require (in Definition 17) the existence of a distinguishing subprotocol $f_{\texttt{bad-knowledge},\Pi}$ that raises `bad-knowledge` in $\mathsf{Mon}(\Pi)$ whenever a message, sent in Π, would allow the attacker to distinguish $b(\Pi)$ and $\overline{b}(\Pi)$.

Parameterized CoSP Protocols. For a bi-protocol Π, we formalize the distinguishing subprotocols $f_{\texttt{bad-knowledge},\Pi}$ and $f_{\texttt{bad-branch},\Pi}$ with the help of *parameterized CoSP protocols*, which have the following properties: Nodes in such protocols are not required to have successors and instead of other nodes, also formal parameters can be referenced. A parameterized CoSP protocol is intended to be plugged into another protocol; in that case the parameters references must be changed to references to nodes.

Definition 16 (Self-monitor). *Let Π be a CoSP bi-protocol. Let $f_{\texttt{bad-knowledge},\Pi}$ and $f_{\texttt{bad-branch},\Pi}$ be functions that map execution traces to parameterized CoSP protocols*[6] *whose leaves are either* `ok`*, in which case they have open edges, or nodes that raise the event* `bad-knowledge`*, or* `bad-branch` *respectively. Let Π a CoSP bi-protocol.*

[5] This leads to the fact that whenever there is an output node in Π, there are two corresponding output nodes in $\mathsf{Mon}(\Pi)$, which contradicts the goal that the interface of Π and $\mathsf{Mon}(\Pi)$ should be the same towards the attacker. However, this technicality can be dealt with easily when applying our method. For example, in the computational proof for our case study, we use the self-monitor in an interaction with a filter machine that hides the results of the output nodes of $\overline{b}(\Pi)$ to create a good simulation towards the computational attacker, whose goal is to distinguish Π. The filter machine is then used as a computational attacker against $\mathsf{Mon}(\Pi)$.

[6] These functions are candidates for distinguishing subprotocols for `bad-knowledge` and `bad-branch`, respectively, for the bi-protocol Π, as defined in Definition 17.

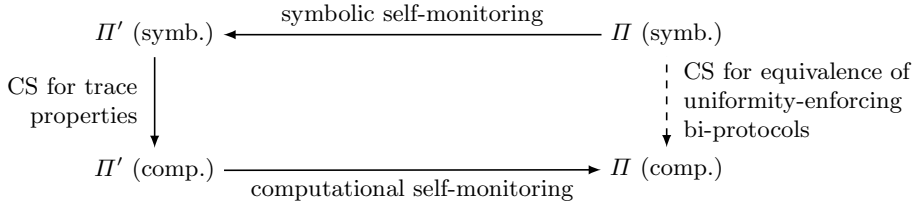

Fig. 1. Symbolic and computational self-monitoring

Recall that nodes ν of Π have bi-references (as defined in Definition 3) consisting of a left reference (to be used in the left protocol) and a right reference. We write $\mathsf{left}(\nu)$ *for the node with only the left reference and* $\mathsf{right}(\nu)$ *analogously. Let tr be the execution trace so far, i.e., the list of node identifiers on the path from ν to the root of Π. The* self-monitor $\mathsf{Mon}(\Pi)$ *protocol is defined as follows:*

Insert before the root node a control node with two copies of Π, called the left *branch (with $b := \mathsf{left}$) and the* right *branch (with $b := \mathsf{right}$). Apply the following modifications recursively for each node ν, starting at the root of Π:*

1. *If ν is a* computation node *of Π, replace ν with $f_{\texttt{bad-branch},\Pi}(b, tr)$. Append two copies* $\mathsf{left}(\nu)$ *and* $\mathsf{right}(\nu)$ *of the the computation node ν to each open edge of an* ok*-leaf. All left references that pointed to ν point in* $\mathsf{Mon}(\Pi)$ *to* $\mathsf{left}(\nu)$*, and all right references that pointed to ν point in* $\mathsf{Mon}(\Pi)$ *to* $\mathsf{right}(\nu)$*. The successor of* $\mathsf{right}(\nu)$ *is the subtree rooted at the successor of ν.*
2. *If ν is an* output node *of Π, replace ν with $f_{\texttt{bad-knowledge},\Pi}(b, tr)$. Append the sequence of the two output nodes* $\mathsf{left}(\nu)$ *(labeled with* `left`*) and* $\mathsf{right}(\nu)$ *(labeled with* `right`*) to each open edge of an* ok*-leaf. All left references that pointed to ν point in* $\mathsf{Mon}(\Pi)$ *to* $\mathsf{left}(\nu)$*, and all right references that pointed to ν point in* $\mathsf{Mon}(\Pi)$ *to* $\mathsf{right}(\nu)$*. The successor of* $\mathsf{right}(\nu)$ *is the subtree rooted at the successor of ν.*

Theorem 1 follows from two properties of the distinguishing subprotocols: *symbolic monitoring* and *computational monitoring* (see Figure 1). Symbolic monitoring states that whenever a bi-protocol Π is indistinguishable, the corresponding distinguishing subprotocol in $\mathsf{Mon}(\Pi)$ does not raise the event `bad`. Computational Monitoring, in turn, states that whenever the distinguishing subprotocol in $\mathsf{Mon}(\Pi)$ does not raises the event `bad`, then Π is indistinguishable.

Shortened Protocols Π_i. Since we prove Theorem 1 by induction over the nodes in a bi-protocol, we introduce a notion of *shortened protocols* in the definition of distinguishing subprotocols. For a (bi-)protocol Π, the shortened (bi-)protocol Π_i is for the first i nodes exactly like Π but that stops after the ith node that is either a control node or an output node.[7]

Definition 17 (Distinguishing Subprotocols). *Let* $\mathbf{M}$ *be a symbolic model and* $\mathbf{A}$ *a computational implementation of* $\mathbf{M}$*. Let Π be a bi-protocol and* $\mathsf{Mon}(\Pi)$

[7] Formally, the protocol only has an infinite chain of control nodes with single successors after this node.

its self-monitor. Let $e \in \{\texttt{bad-knowledge}, \texttt{bad-branch}\}$ *and* $n_{\texttt{bad-knowledge}}$ *denote the node type output node and* $n_{\texttt{bad-branch}}$ *denote the node type control node. Then the function* $f_{e,\Pi}(b, tr)$*, which takes as input* $b \in \{\mathsf{left}, \mathsf{right}\}$ *and the path to the root node, including all node and edge identifiers, is a* distinguishing subprotocol for e *for* Π *and* $\mathbf{M}$ *if it is computable in deterministic polynomial time, and if the following conditions hold for every* $i \in \mathbb{N}$*:*

1. symbolic self-monitoring: *If* Π_i *is symbolically indistinguishable,* `bad` *does symbolically not occur in* $\mathsf{Mon}(\Pi_{i-1})$*, and the* i*th node in* Π_i *is of type* n_e*, then the event* e *does not occur symbolically in* $\mathsf{Mon}(\Pi_i)$*.*
2. computational self-monitoring: *The event* e *in* $\mathsf{Mon}(\Pi_i)$ *occurs computationally with negligible probability,* Π_{i-1} *is computationally indistinguishable, and the* i*th node in* Π_i *is of type* n_e*, then* Π_i *is computationally indistinguishable.*

We say that a $\mathbf{M}$ *and a protocol class* allows for self-monitoring *if there are distinguishing subprotocols for* `bad-branch` *and* `bad-knowledge` *for every bi-protocol in the protocol class.*

Finally, we are ready to state our main theorem.

Theorem 1. *Let* $\mathbf{M}$ *be a symbolic model and* P *be a uniformity-enforcing class of bi-protocols. If* $\mathbf{M}$ *and* P *allow for self-monitoring (in the sense of Definition 17), then the following holds: If* $\mathbf{A}$ *is a computationally sound implementation of a symbolic model* $\mathbf{M}$ *with respect to trace properties then* $\mathbf{A}$ *is also a computationally sound implementation with respect to equivalence properties.*

4 The Applied π-calculus

In this section, we present the connection of uniform bi-processes in the applied π-calculus and our CS result in CoSP, namely that the applied π-calculus can be embedded into the extended CoSP framework. In contrast to previous work [22,23,24], we consider CS for bi-protocols from the full applied π-calculus. In particular, we also consider private channels and non-determinate processes.

We consider the variant of the applied π-calculus also used for the original CoSP embedding [15]. The operational semantics of the applied π-calculus is defined in terms of *structural equivalence* ($\equiv$) and *internal reduction* ($\rightarrow$); for a precise definition of the applied π-calculus, we refer to [7].

A uniform bi-process [30] in the applied π-calculus is the counterpart of a uniform bi-protocol in CoSP. A bi-process is a pair of processes that only differ in the terms they operate on. Formally, they contain expressions of the form $choice[a, b]$, where a is used in the left process and b is used in the right one. A bi-process Q can only reduce if both its processes can reduce in the same way.

Definition 18 (Uniform Bi-process). *A bi-process* Q *in the applied* π*-calculus is* uniform *if* $\mathsf{left}(Q) \rightarrow R_{\mathsf{left}}$ *implies that* $Q \rightarrow R$ *for some bi-process* R *with* $\mathsf{left}(R) \equiv R_{\mathsf{left}}$*, and symmetrically for* $\mathsf{right}(Q) \rightarrow R_{\mathsf{right}}$ *with* $\mathsf{right}(R) \equiv R_{\mathsf{right}}$*.*

The following lemma connects uniformity in the applied π-calculus to uniformity in CoSP. (See [34] for a proof.)

Lemma 1 (Uniformity in CoSP and the Applied π-calculus). *There is an embedding e from bi-processes in the applied π-calculus to CoSP uniformity-enforcing bi-protocols such that for every bi-process Q in the applied π-calculus, the following holds: If Q is uniform, then* $\mathsf{left}(e(Q)) \approx_s \mathsf{right}(e(Q))$.

5 Case Study: Encryption and Signatures with Lengths

We exemplify our method by proving a CS result for equivalence properties, which captures protocols that use public-key encryption and signatures. We use the CS result in [31] for trace properties, which we extend by a length function, realized as a destructor. Since encryptions of plaintexts of different length can typically be distinguished, we must reflect that fact in the symbolic model.

5.1 The Symbolic Model

Lengths in the Symbolic Model. In order to express lengths in the symbolic model, we introduce *length specifications*, which are the result of applying a special destructor *length*/1. We assume that the bitlength of every computational message m_c is of the form $|m_c| = rk$ for some natural number r, where k is the security parameter, i.e., the length of a nonce. This assumption will be made precise. With this simplification, length specifications only encode r; this can be done using Peano numbers, i.e., the constructors O (zero) and S (successor).

Even though this approach leads admittedly to rather inefficient realizations from a practical point of view,[8] the aforementioned assumption can be realized using a suitable padding. Essentially, this assumption is similar to the one introduced by Cortier and Comon-Lundh [22] for a symbolic model for symmetric encryption. The underlying problem is exactly the same: while the length of messages in the computational model, in particular the length of ciphertexts, may depend on the security parameter, there is no equivalent concept in the symbolic model. For instance, let n and m be nonces, and let ek be an encryption key. For certain security parameters in the computational model, the computational message $pair(n, m)$ may have the same length as the message $enc(ek, n)$; for other security parameters this may not be the case. Thus it is not clear if the corresponding symbolic messages should be of equal symbolic length. Comon-Lundh et al. [28] propose a different approach towards this problem, by labeling messages symbolically with an expected length and checking the correctness of these length computationally. However, it is not clear whether such a symbolic model can be handled by current automated verification tools.

Automated Verification: Combinding ProVerif and APTE. ProVerif is not able to handle recursive destructors such as *length*, e.g., $length(pair(t_1, t_2)) =$

[8] Consider, e.g., a payload string that should convey n bits. This message must be encoded using at least kn bits.

$length(t_1)+length(t_2)$. Recent work by Cheval and Cortier [32] extends the protocol verifier APTE, which is capable of proving trace equivalence of two processes in the applied π-calculus, to support such length functions. Since however trace equivalence is a weaker notion than uniformity, i.e., there are bi-processes that are trace equivalent but not uniform, our CS result does not carry over to APTE. Due to the lack of a tool that is able to check uniformity as well as to handle length functions properly, we elaborate and prove in [34] how APTE can be combined with ProVerif to make protocols on the symbolic model of our case study amenable to automated verification.

We consider the following symbolical model $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$.

Constructors and Nonces. We define $\mathbf{C} := \{enc/3, ek/1, dk/1, sig/3, vk/1, sk/1, string_0/1, string_1/1, emp/0, pair/2, O/0, S/1, garbageEnc/3, garbageSig/3, garbage/2, garbageInvalidLength/1\}$ and $\mathbf{N} := \mathbf{N_E} \uplus \mathbf{N_E}$ for countably infinite sets of protocol nonces $\mathbf{N_P}$ and attacker nonces $\mathbf{N_E}$. Encryption, decryption, verification, and signing keys are represented as $ek(r), dk(r), vk(r), sk(r)$ with a nonce r (the randomness used when generating the keys). The term $enc(ek(r'), m, r)$ encrypts m using the encryption key $ek(r')$ and randomness r. $sig(sk(r'), m, r)$ is a signature of m using the signing key $sk(r')$ and randomness r. The constructors $string_0$, $string_1$, and emp are used to model arbitrary strings used as payload in a protocol, e.g., a bitstring 010 would be encoded as $string_0(string_1(string_0(emp())))$. Length specifications can be constructed using O representing zero and S representing the successor of a number. *garbage*, *garbageInvalidLength*, *garbageEnc*, and *garbageSig* are not used by the protocol; they express invalid terms the attacker may send.

Message Type. We define $\mathbf{T}$ as the set of terms M according to this grammar:

$$\begin{aligned} M ::= {} & enc(ek(N), M, N) \mid ek(N) \mid dk(N) \mid \\ & sig(sk(N), M, N) \mid vk(N) \mid sk(N) \mid pair(M, M) \mid S \mid N \mid L \mid \\ & garbage(N, L) \mid garbageInvalidLength(N) \\ & garbageEnc(M, N, L) \mid garbageSig(M, N, L) \\ S ::= {} & emp() \mid string_0(S) \mid string_1(S) \qquad L ::= O() \mid S(L) \end{aligned}$$

The nonterminals N and L represent nonces and length specifications, respectively. Note that the garbage terms carry an explicit length specification to enable the attacker to send invalid terms of a certain length.

Destructors. We define $\mathbf{D} := \{dec/2, isenc/1, isek/1, isdk/1, ekof/1, ekofdk/1, verify/2, isvk/1, issk/1, issig/1, vkofsk/1, vkof/1, unstring_0/1, unstring_1/1, fst/1, snd/1, equals/2, length/1, unS/1\}$. The destructors *isek*, *isdk*, *isvk*, *issk*, *isenc*, and *issig* realize predicates to test whether a term is an encryption key, decryption key, verification key, signing key, ciphertext, or signature, respectively. *ekof* extracts the encryption key from a ciphertext, *vkof* extracts the verification key from a signature. $dec(dk(r), c)$ 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. *ekofdk* and *vkofsk* compute the encryption/verification key corresponding to a decryption/signing key. The destructors *fst* and *snd* are used

to destruct pairs, and the destructors $unstring_0$ and $unstring_1$ allow to parse payload-strings. The destructor *length* returns a the length of message, where the unit is the length of a nonce. The purpose of *unS* is destruct numbers that represent lengths. (The full description of all destructor rules is given in [34].)

Length Destructor. Our result is parametrized over the destructor *length* that must adhere to the following restrictions:

1. Each message except for *garbageInvalidLength* is assigned a length: $length(t) \neq \bot$ for all terms $t \in \mathbf{T} \setminus \{garbageInvalidLength(t') \mid t' \in \mathbf{T}\}$.
2. The length of garbage terms (constructed by the attacker) is consistent:

$$length(garbage(t,l)) = l, \qquad length(garbageEnc(t_1,t_2,l)) = l,$$
$$length(garbageSig(t_1,t_2,l)) = l, \quad length(garbageInvalidLength(t_1)) = \bot$$

3. Let $[\cdot]$ be the canonical interpretation of Peano numbers, given by $[O] = 0$ and $[S(l)] = [l] + 1$. We require the length destructor to be linear: For each constructor $C/n \in \mathbf{C} \setminus \{garbage, garbageInvalidLength, garbageEnc, garbageSig\}$ there are $a_i \in \mathbb{N}$ (where $i = 0, \ldots, n$) such that $length(t_i) = l_i$ for $i = 1, \ldots, n$ and $length(C(\underline{t})) = l$ together imply $[l] = \sum_{i=1}^{n} a_i \cdot [l_i] + a_0$.

5.2 Computational Soundness

Protocol Conditions and Implementation Conditions. For establishing CS, we require the protocols to fulfill several natural conditions regarding their use of randomness, e.g., that fresh randomness is used for key generation. Protocols that adhere to these protocol conditions are called *randomness-safe*. For the full protocol and implementation conditions, we refer to the extended version [34].

Additionally, the computational implementation needs to fulfill certain conditions, e.g., that the encryption scheme is PROG-KDM secure [35], and the signature scheme is SUF-CMA. Both protocol conditions and implementation conditions are similar to those in [31]. Requiring PROG-KDM [35] is only needed to handle protocols that send and receive decryption keys.[9]

For lengths in the computational model, we require that the computational implementation A_{length} of the destructor *length* computes the bitlength of the corresponding bitstring. To connect the symbolic result of the destructor *length* to bit-lengths in the computational world, we require length consistency.

Definition 19 (Length Consistency). *Let* $\mathbf{M} = (\mathbf{C}, \mathbf{N}, \mathbf{T}, \mathbf{D})$ *be a symbolic model such that there is a constructor* $length/1$ *in* $\mathbf{D}$*, and let* $[\cdot]$ *be an interpretation mapping length specifications to natural numbers.*

[9] In principle, our proofs do not rely on this particular security definition. For example, it would be possible to obtain a CS result for uniformity using weaker implementation conditions (IND-CCA secure public-key encryption) but a restricted protocol class, by applying our proof technique to the CS result for trace properties in [15].

Given a security parameter k, a computational variant of a message $m \in \mathbf{T}$ is obtained by implementing each constructor C and nonce N in m by the corresponding algorithm A_C or A_N, respectively. For example, for all random choices of $A_N(k)$, $A_{pair}(k, A_{string_0}(k, A_{emp}(k), A_{ek}(k, A_N(k)))$ is a computational variant of the message $pair(string_0(emp(), ek(N))$, where $N \in \mathbf{N}$.

We say that a computational implementation $\mathbf{A}$ of $\mathbf{M}$ is length-consistent with respect to the interpretation $[\cdot]$ *if for each message $m \in \mathbf{T}$ and all of its computational variants m_k under security parameter k, we have that $length(m) \neq \perp$ implies $|m_k| = [length(m)] \cdot k$.*

Length specifications are ordinary messages that the protocol can process, send and receive. We require length specifications to have a length itself. Moreover, we require that the decryption algorithm A_{dec} expects a length description of the plaintext and fails if the length of the plaintexts do not match.

CS for Trace Properties with Length Functions. We extend the CS result for trace properties by Backes, Unruh, and Malik [31], which holds for signatures and public-key encryption, to lengths functions.

Theorem 2. *Let $\mathbf{A}$ be a computational implementation fulfilling the implementation conditions from above, i.e., in particular $\mathbf{A}$ is length-consistent. Then, $\mathbf{A}$ is a computationally sound implementation of the symbolic model $\mathbf{M}$ for the class of randomness-safe protocols.*

Distinguishing Subprotocols for the Symbolic Model M. In this section, we discuss the distinguishing subprotocols for the symbolic model $\mathbf{M}$. The full descriptions and proofs can be found in the extended version [34].

We construct a distinguishing subprotocol $f_{\texttt{bad-branch},\Pi}(b, tr)$ for a computation node ν that investigates each message that has been received at an input node (in the execution trace tr of $\mathsf{Mon}(\Pi)$) by parsing the message using computation nodes. The distinguishing subprotocol then reconstructs an attacker strategy by reconstructing a possible symbolic operation for every input message. In more detail, in the symbolic execution, $f_{\texttt{bad-branch},\Pi}(b, tr)$ parses the input message with all symbolic operations in the model $\mathbf{M}$ that the attacker could have performed as well, i.e., with all tests from the *shared knowledge*. This enables $f_{\texttt{bad-branch},\Pi}(b, tr)$ to simulate the symbolic execution of $\overline{b}(\Pi)$ on the constructed attacker strategy. In the computational execution of the self-monitor, the distinguishing subprotocol constructs the symbolic operations (i.e., the symbolic inputs) by parsing the input messages with the implementations of all tests in the shared knowledge (i.e., lookups on output messages and implementations of the destructors). With this reconstructed symbolic inputs (i.e., symbolic operations, from messages that were intended for $b(\Pi)$, $f_{\texttt{bad-branch},\Pi}(b, tr)$ is able to simulate the symbolic execution of $\overline{b}(\Pi)$ even in the computational execution. The distinguishing subprotocol $f_{\texttt{bad-branch},\Pi}(b, tr)$ then checks whether this simulated symbolic execution of $\overline{b}(\Pi)$ takes in the same branch as $b(\Pi)$ would take, for the computation node ν in question. If this is not the case, the event `bad-branch` is raised.

Symbolic monitoring follows by construction because the distinguishing subprotocol reconstructs a correct attacker strategy and correctly simulates a symbolic execution. Hence, $f_{\texttt{bad-branch},\Pi}(b, tr)$ found a distinguishing attacker strategy for $b(\Pi)$ and $\overline{b}(\Pi)$. We show computational monitoring by applying the CS result for trace properties to conclude that the symbolic simulation of $\overline{b}(\Pi)$ suffices to check whether $b(\Pi)$ computationally branches differently from $\overline{b}(\Pi)$.

The distinguishing subprotocol $f_{\texttt{bad-knowledge},\Pi}(b, tr)$ for an output node ν starts like $f_{\texttt{bad-branch},\Pi}(b, tr)$ by reconstructing a (symbolic) attacker strategy and simulating a symbolic execution of $\overline{b}(\Pi)$. However, instead of testing the branching behavior of $\overline{b}(\Pi)$, the distinguishing subprotocol $f_{\texttt{bad-knowledge},\Pi}(b, tr)$ characterizes the message m that is output in $b(\Pi)$ at the output node ν in question, and then $f_{\texttt{bad-knowledge},\Pi}(b, tr)$ compares m to the message that would be output in $\overline{b}(\Pi)$. This characterization must honor that ciphertexts generated by the protocol are indistinguishable if the corresponding decryption key has not been revealed to the attacker so far. If a difference in the output of $b(\Pi)$ and $\overline{b}(\Pi)$ is detected, the event `bad-knowledge` is raised.

Symbolic monitoring for the distinguishing subprotocol $f_{\texttt{bad-knowledge},\Pi}(b, tr)$ follows by the same arguments as for $f_{\texttt{bad-branch},\Pi}(b, tr)$. We show computational monitoring by first applying the PROG-KDM property to prove that the computational execution of $b(\Pi)$ is indistinguishable from a *faking* setting: in the faking setting, all ciphertexts generated by the protocol do not carry any information about their plaintexts (as long as the corresponding decryption key has not been leaked). The same holds analogously for $\overline{b}(\Pi)$. We then consider all remaining real messages, i.e., all messages except ciphertexts generated by the protocol with unleaked decryption keys. We conclude the proof by showing that in the faking setting, $f_{\texttt{bad-knowledge},\Pi}(b, tr)$ is able to sufficiently characterize all real messages to raise the event `bad-knowledge` whenever the bi-protocol Π is distinguishable.

Lemma 2. *Let* P *be a uniformity-enforcing class of randomness-safe bi-protocols and* **A** *a computationally sound implementation of the symbolic model* **M**. *For each bi-protocol* Π, $f_{\texttt{bad-knowledge},\Pi}$ *and* $f_{\texttt{bad-branch},\Pi}$ *as described above are distinguishing subprotocols (see Definition 17) for* **M** *and* P.

CS for Uniform Bi-processes in the Applied π-calculus. Combining our results, we conclude CS for protocols in the applied π-calculus that use signatures, public-key encryption, and corresponding length functions.

Theorem 3 (CS for Enc. and Signatures in the Applied π-calculus). *Let* **M** *be as defined in Section 5. Let* Q *be a randomness-safe bi-process in the applied* π*-calculus, and let* **A** *of* **M** *be an implementation that satisfies the conditions from above. Let* e *be the embedding from bi-processes in the applied* π*-calculus to CoSP bi-protocols. If* Q *is uniform, then* $\mathsf{left}(e(Q)) \approx_c \mathsf{right}(e(Q))$.

Proof. By Lemma 2, there are for each bi-protocol Π distinguishing subprotocols $f_{\texttt{bad-knowledge},\Pi}$ and $f_{\texttt{bad-branch},\Pi}$ for **M**. The class of the embedding of the applied π-calculus is uniformity-enforcing by Lemma 1; thus, Theorem 1 entails the claim.

6 Conclusion

In this work, we provided the first result that allows to leverage existing CS results for trace properties to CS results for uniformity of bi-processes in the applied π-calculus. Our result, which is formulated in an extension of the CoSP framework to equivalence properties, holds for Dolev-Yao models that fulfill the property that all distinguishing computational tests are expressible as a process on the model. We exemplified the usefulness of our method by applying it to a Dolev-Yao model that captures signatures and public-key encryption.

We moreover discussed how computationally sound, automated analyses can still be achieved in those frequent situations in which ProVerif does not manage to terminate whenever the Dolev-Yao model supports a length function. We propose to combine ProVerif with the recently introduced tool APTE [32].

We leave as a future work to prove for more comprehensive Dolev-Yao models (e.g., for zero-knowledge proofs) the sufficient conditions for deducing from CS results for trace properties the CS of uniformity. Another interesting direction for future work is the extension of our result to observational equivalence properties that go beyond uniformity.

Acknowledgements. This work was partially supported by the German Universities Excellence Initiative, the ERC Grant End-2-End Security, and the Center for IT-Security, Privacy and Accountability (CISPA).

References

1. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory 29(2), 198–208 (1983)
2. Even, S., Goldreich, O.: On the security of multi-party ping-pong protocols. In: FOCS, pp. 34–39. IEEE (1983)
3. Kemmerer, R., Meadows, C., Millen, J.: Three systems for cryptographic protocol analysis. J. of Crypt. 7(2), 79–130 (1994)
4. Backes, M., Jacobi, C., Pfitzmann, B.: Deriving cryptographically sound implementations using composition and formally verified bisimulation. In: Eriksson, L.-H., Lindsay, P.A. (eds.) FME 2002. LNCS, vol. 2391, pp. 310–329. Springer, Heidelberg (2002)
5. Delaune, S., Kremer, S., Ryan, M.: Verifying privacy-type properties of electronic voting protocols. J. Comput. Secur. 17(4), 435–487 (2009)
6. Delaune, S., Kremer, S., Ryan, M.D., Steel, G.: Formal analysis of protocols based on tpm state registers. In: CSF, pp. 66–80. IEEE (2011)
7. Blanchet, B., Abadi, M., Fournet, C.: Automated Verification of Selected Equivalences for Security Protocols. In: LICS, pp. 331–340. IEEE (2005)
8. Backes, M., Maffei, M., Unruh, D.: Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In: S&P, pp. 202–215. IEEE (2008)
9. Cortier, V., Wiedling, C.: A formal analysis of the norwegian E-voting protocol. In: Degano, P., Guttman, J.D. (eds.) POST 2012. LNCS, vol. 7215, pp. 109–128. Springer, Heidelberg (2012)
10. Backes, M., Hritcu, C., Maffei, M.: Automated verification of remote electronic voting protocols in the applied pi-calculus. In: CSF, pp. 195–209. IEEE (2008)

11. Janvier, R., Lakhnech, Y., Mazaré, L.: Completing the picture: Soundness of formal encryption in the presence of active adversaries. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 172–185. Springer, Heidelberg (2005)
12. Micciancio, D., Warinschi, B.: Soundness of formal encryption in the presence of active adversaries. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 133–151. Springer, Heidelberg (2004)
13. Cortier, V., Warinschi, B.: Computationally sound, automated proofs for security protocols. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 157–171. Springer, Heidelberg (2005)
14. Cortier, V., Kremer, S., Küsters, R., Warinschi, B.: Computationally sound symbolic secrecy in the presence of hash functions. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 176–187. Springer, Heidelberg (2006)
15. Backes, M., Hofheinz, D., Unruh, D.: CoSP: a general framework for computational soundness proofs. In: CCS, pp. 66–78. ACM (2009)
16. Backes, M., Unruh, D.: Computational soundness of symbolic zero-knowledge proofs. J. of Comp. Sec. 18(6), 1077–1155 (2010)
17. Galindo, D., Garcia, F.D., van Rossum, P.: Computational soundness of non-malleable commitments. In: Chen, L., Mu, Y., Susilo, W. (eds.) ISPEC 2008. LNCS, vol. 4991, pp. 361–376. Springer, Heidelberg (2008)
18. Cortier, V., Warinschi, B.: A composable computational soundness notion. In: CCS, pp. 63–74. ACM (2011)
19. Böhl, F., Cortier, V., Warinschi, B.: Deduction soundness: Prove one, get five for free. In: CCS, pp. 1261–1272. ACM (2013)
20. Backes, M., Bendun, F., Unruh, D.: Computational soundness of symbolic zero-knowledge proofs: weaker assumptions and mechanized verification. In: Basin, D., Mitchell, J.C. (eds.) POST 2013. LNCS, vol. 7796, pp. 206–225. Springer, Heidelberg (2013)
21. Backes, M., Maffei, M., Mohammadi, E.: Computationally sound abstraction and verification of secure multi-party computations. In: FSTTCS, Schloss Dagstuhl, pp. 352–363 (2010)
22. Comon-Lundh, H., Cortier, V.: Computational soundness of observational equivalence. In: CCS, pp. 109–118. ACM (2008)
23. Comon-Lundh, H., Cortier, V., Scerri, G.: Security proof with dishonest keys. In: Degano, P., Guttman, J.D. (eds.) POST 2012. LNCS, vol. 7215, pp. 149–168. Springer, Heidelberg (2012)
24. Canetti, R., Herzog, J.: Universally composable symbolic security analysis. J. of Crypt. 24(1), 83–147 (2011)
25. Backes, M., Pfitzmann, B., Waidner, M.: A composable cryptographic library with nested operations (extended abstract). In: CCS, pp. 220–230 (2003)
26. Backes, M., Pfitzmann, B.: Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In: CSFW, pp. 204–218. IEEE (2004)
27. Backes, M., Laud, P.: Computationally sound secrecy proofs by mechanized flow analysis. In: CCS, pp. 370–379. ACM (2006)
28. Comon-Lundh, H., Hagiya, M., Kawamoto, Y., Sakurada, H.: Computational soundness of indistinguishability properties without computable parsing. In: Ryan, M.D., Smyth, B., Wang, G. (eds.) ISPEC 2012. LNCS, vol. 7232, pp. 63–79. Springer, Heidelberg (2012)
29. Sprenger, C., Backes, M., Basin, D., Pfitzmann, B., Waidner, M.: Cryptographically sound theorem proving. In: CSFW, pp. 153–166. IEEE (2006)
30. Blanchet, B., Fournet, C.: Automated verification of selected equivalences for security protocols. In: LICS 2005, pp. 331–340. IEEE (2005)

31. Backes, M., Malik, A., Unruh, D.: Computational soundness without protocol restrictions. In: CCS, pp. 699–711. ACM (2012)
32. Cheval, V., Cortier, V., Plet, A.: Lengths may break privacy – or how to check for equivalences with length. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 708–723. Springer, Heidelberg (2013)
33. Backes, M., Maffei, M., Unruh, D.: Computationally sound verification of source code. In: CCS, pp. 387–398. ACM (2010)
34. Backes, M., Mohammadi, E., Ruffing, T.: Bridging the gap from trace properties to uniformity (2014), `http://www.infsec.cs.uni-saarland.de/~mohammadi/bridge.html`
35. Unruh, D.: Programmable encryption and key-dependent messages. Technical report, IACR ePrint Report 2012/423 (2012)