Deduction Soundness: Prove One, Get Five for Free

Florian Böhl
KIT
Karlsruhe, Germany
boehl@kit.edu

Véronique Cortier
LORIA - CNRS
Nancy, France
cortier@loria.fr

Bogdan Warinschi
University of Bristol
Bristol, United Kingdom
bogdan@cs.bris.ac.uk

ABSTRACT

Most computational soundness theorems deal with a limited number of primitives, thereby limiting their applicability. The notion of deduction soundness of Cortier and Warinschi (CCS'11) aims to facilitate soundness theorems for richer frameworks via composition results: deduction soundness extends, generically, with asymmetric encryption and public data structures. Unfortunately, that paper also hints at rather serious limitations regarding further composition results: composability with digital signatures seems to be precluded.

In this paper we provide techniques for bypassing the perceived limitations of deduction soundness and demonstrate that it enjoys vastly improved composition properties. More precisely, we show that a deduction sound implementation can be modularly extended with all of the basic cryptographic primitives (symmetric/asymmetric encryption, message authentication codes, digital signatures, and hash functions). We thus obtain the first soundness framework that allows for the joint use of multiple instances of all of the basic primitives.

In addition, we show how to overcome an important restriction of the bare deduction soundness framework which forbids sending encrypted secret keys. In turn, this prevents its use for the analysis of a large class of interesting protocols (e.g. key exchange protocols). We allow for more liberal uses of keys as long as they are hidden in a sense that we also define. All primitives typically used to send secret data (symmetric/asymmetric encryption) satisfy our requirement which we also show to be preserved under composition.

1. INTRODUCTION

Two main approaches have been developed for the analysis of security protocols. Symbolic models abstract away the cryptographic primitives, allowing to reason at a logical level, often in an automated way. Security proofs are therefore easier and often conducted by tools. In contrast, cryptographic models offer higher security guarantees, down to the bitstring level. Security proofs are usually done by hand and proceed by reduction, down to the security assumptions (such as the computational hardness of computing a discrete log).

Starting with the seminal work of Abadi and Rogaway [1], many results study under which assumptions it is possible to show that symbolic models are actually *sound* w.r.t. to cryptographic models. For example, the symbolic representation of symmetric encryption consists simply of two following deduction rules.

$$\frac{k \quad m}{\mathsf{enc}(k,m)} \qquad \frac{\mathsf{enc}(k,m) \quad k}{m}$$

An attacker can encrypt or decrypt only if he has the corresponding key. Given an encryption scheme, does it hold that all attacker's computations are reflected by these rules? Surprisingly, the answer is yes, provided that the encryption scheme satisfies some standard security requirements [2, 6] (here IND-CCA). Such soundness theorems have been established for active attackers for basically all standard cryptographic primitives: symmetric encryption [2, 6], asymmetric encryption [3, 9, 11], signatures [3, 14, 9], MACs [4], hashes [14, 7, 13] (consult [8] for a more comprehensive list).

However, these past results usually consider the primitives in isolation or, in the best case, treat at most two primitives at a time. Soundness proofs are complex, and including multiple primitives in the analysis easily leads to unmanageable proofs. A way to bring the complexity under control is to develop soundness results that are compositional. A first step in this direction is the work of Cortier and Warinschi [10]. They propose a notion of soundness which can be extended, in a generic way in several ways, most notably with asymmetric encryption: if a deduction system is sound for some primitive (in the sense that they define) then extending the deduction system with the usual deduction rules for asymmetric encryption is a sound abstraction for combined uses of the primitive and asymmetric encryption. Below, we refer to this notion as *deduction soundness*.

The central idea that allows for composability is that deduction soundness considers the use of the primitives in the presence of functions chosen *adversarially* from the class of *transparent* functions. These are publicly computable and efficiently invertible functions. Typical functions that are transparent are the constructors of public data structures like concatenation, lists, etc. It is then obvious that deduction soundness in this sense implies soundness for the use of primitive in the presence of other constructs that are naturally transparent (e.g. public data structures). Less obvious is that deduction soundness for a primitive also implies soundness when the primitive is used together with asymmetric encryption. In addition to this result (which is the main technical contribution of [10]) that paper also shows that deduction soundness implies that security of protocols in symbolic models yields security in the computational models, for a wide class of protocols.

Compositionality for the notion introduced in [10] is however limited, and the authors present rather compelling ev-

idence that the notion may not compose primitives other than encryption. The problem is that deduction soundness does not seem to preclude implementations that leak partial information about their inputs. In turn, this leak of information may impact the security of other primitives that one may want to include later.

More concretely, assume that one has established soundness of a deduction system that covers hash, but for an implementation of the hash function that reveals half of its input: $h(m_1\|m_2) = m_1\|g(m_2)$ where g is a standard hash function. If g is a "good" hash function then so is h. Now consider a signature scheme which duplicates signatures: $\mathsf{sign}(sk, m) = \mathsf{sign}'(sk, m)\|\mathsf{sign}'(sk, m)$ where sign' is some standard signature scheme. It is easy to see that if $\mathsf{sign}'(sk, m)$ is a secure signature scheme, then so is $\mathsf{sign}(sk, m)$. Yet, given $h(\mathsf{sign}(sk, m))$ an adversary can easily compute $\mathsf{sign}(sk, m)$ without breaking the signature scheme nor the hash: the hash function leaks sufficient information to be able to recover the underlying signature.

Our contributions. In this paper we provide new insights into the notion of deduction soundness. Despite the intuition outlined above, we prove that the compositionality properties of deduction soundness [10] reach further than previously understood. For example, we prove that to any deduction sound implementation of a set of primitives, one can add signatures, as long as the implementation for the signature satisfies a standard notion of security. This theorem refutes the counterexample above and provides evidence that deduction soundness is a more powerful (and demanding) security notion than previously understood. In particular, a corollary of the theorem is that there are no deduction sound abstractions for implementations that are "too leaky" (as the hash function from the counterexample).

The new level of understanding facilitates further compositionality proofs for deduction soundness: to any deduction sound system one can add any of the (remaining) standard cryptographic primitives: symmetric encryption, message authentication codes, and hash functions while preserving deduction soundness. The theorems hold under standard security assumptions for the implementation of encryption and MACs and require random oracles for adding hash functions. As a consequence, we obtain the first soundness result that encompasses all standard primitives: symmetric and asymmetric encryption, signatures, MACs, and hashes. In addition, our composition results allow for a settings where multiple schemes (that implement the same primitive) are used simultaneously, provided that each implementation fulfills our assumptions. Moreover, composition provides a stronger result: whenever deduction soundness is shown for some particular primitive, our result ensures that all standard primitives can be added for free, without any further proof.

The importance of composition cannot be overemphasized: obtaining such general results without being able to study each primitive separately would be unmanageable.

Our compositionality results hold under several restrictions most of which are quite common in soundness proofs, e.g. adversaries can corrupt keys only statically. Less standard is that we demand for secret keys to be used only for the cryptographic task for which they are intended. Quite reasonable most of the time, the restriction does not allow, for example, for the adversary to see encryptions of symmetric keys under public keys. The restriction is related to the signature-hash counterexample. If f is a primitive with a deduction sound system that leaks some information about its input and Enc is a secure encryption scheme it is not clear that $(f(k), Enc_k(m))$ hides m. Unfortunately, the technique that we used to bypass the signature-hash counterexample does not seem to apply here. At a high level, the difficulty is that in a potential reduction to the security of the encryption scheme, we are not be able to simulate $f(k)$ consistently.

One way to relax the restriction is to employ encryption schemes that are secure even when some (or even most) of the encryption key leaks [12, 15]. Current instantiations for such schemes are highly inefficient and we prefer the following alternative solution which, essentially, allows for other uses of symmetric keys, as long as these uses do not reveal information about the keys. In a bit more detail, we say that a function is *forgetful* for some argument if the function hides (computationally) all of the information about that input. The notion is a generalization for the security of encryption schemes: these can be regarded as forgetful with respect to their plaintext. We then show that a forgetful deduction sound implementation can be extended with symmetric encryption under more relaxed restrictions: soundness is preserved if encryption keys are used for encryption, or appear only in forgetful positions of other functions from the implementation we are extending. Finally, we show that, in addition to soundness, forgetfulness is preserved as well. Hence we can flexibly and add several layers of asymmetric/symmetric key encryption such that the keys of each layer may appear in any forgetful position of underlying layers. We feel that this allows us to capture almost every hierarchical encryption mechanism in practical protocols.

2. PRELIMINARIES

Throughout this paper, η denotes the *security parameter*. A function $f : \mathbb{N} \to \mathbb{R}$ is *negligible* if it vanishes faster than the inverse of any polynomial (i.e., if $\forall c \in \mathbb{R}\ \exists n_0 \in \mathbb{N}$ s.t. $\forall n \in \mathbb{N}\ |f(n)| < 1/n^c$). For a finite set R, we denote by $r \leftarrow R$ the process of sampling r uniformly from R.

3. THE SYMBOLIC MODEL

Our abstract models for the symbolic world—called *symbolic models*—consist of term algebras defined on a typed first-order signature.

Specifically we have a set of *data types* $\mathcal{T}$ with a subtype relation ($\leq$) which we require to be a preorder. We assume that $\mathcal{T}$ always contains a base type $\top$ such that every other type $\tau \in \mathcal{T}$ is a subtype of $\top$ ($\tau \leq \top$).

The *signature* Σ is a set of *function symbols* together with arities of the form $\mathrm{ar}(f) = \tau_1 \times \ldots \times \tau_n \to \tau$, $n \geq 0$ for $\tau_i, \tau \in \mathcal{T}$. We refer to τ as the type of f and require $\tau \neq \top$ for all f except for garbage of basetype $g_\top$. Function symbols with $n = 0$ arguments are called *constants*. We distinguish *deterministic* function symbols, e.g., for pairs, and *randomized* function symbols, e.g., for encryption.

For all symbolic models we fix an infinite set of typed *variables* $\{x, y \ldots\}$ and an infinite set of *labels* $\mathsf{labels} = \mathsf{labelsH} \cup \mathsf{labelsA}$ for infinite, disjoint sets of *honest labels* ($\mathsf{labelsH}$) and *adversarial labels* ($\mathsf{labelsA}$). Since labels are used to specify randomness, distinguishing honest and adversarial

labels (randomness) is important.

The set of *terms of type* τ is defined inductively by

t	::=		term of type τ
	|	x	variable x of type τ
	|	$f(t_1, \ldots, t_n)$	application of deterministic $f \in \Sigma$
	|	$f^l(t_1, \ldots, t_n)$	application of randomized $f \in \Sigma$

where for the last two cases, we further require that each t_i is a term of some type τ_i' with $\tau_i' \leq \tau_i$ for $\mathrm{ar}(f) = \tau_1 \times \ldots \times \tau_n \to \tau$ and for the last case that $l \in \mathsf{labels}$. The set of terms is denoted by $\mathsf{Terms}(\Sigma, \mathcal{T}, \leq)$ and is the union over all sets of terms of type τ for all $\tau \in \mathcal{T}$. For ease of notation we often write $\mathsf{Terms}(\Sigma)$ for the same set of terms, and refer to general terms as $t = f^l(t_1, \ldots, t_n)$ even if f could be a deterministic function symbol which doesn't carry a label.

Intuitively, for nonces, we use randomized constants. For example, assume that $n \in \Sigma$ is a constant. Then usual nonces can be represented by $n^{r_1}, n^{r_2}, \ldots$ where $r_1, r_2 \in \mathsf{labels}$ are labels. Labels in $\mathsf{labelsH}$ will be used when the function has been applied by an honest agent (thus the randomness has been honestly generated) whereas labels in $\mathsf{labelsA}$ will be used when the randomness has been generated by the adversary. Often when the label for a function symbol is clear from the context (e.g. when there is only one label that suits a particular function symbol) we may omit this label.

We require Σ to contain randomized constants g_τ of type τ for any $\tau \in \mathcal{T}$ that will be used for representing garbage of type τ. Garbage will typically be the terms associated to bit-strings produced by the adversary which cannot be parsed as a meaningful term (yet). If garbage can at some point be parsed as the application of a deterministic function symbol, the label is dropped.

Substitutions are written $\sigma = \{x_1 = t_1, \ldots, x_n = t_n\}$ with *domain* $\mathrm{dom}(\sigma) = \{x_1, \ldots, x_n\}$. We only consider *well-typed* substitutions, that is substitutions $\sigma = \{x_1 = t_1, \ldots, x_n = t_n\}$ for which t_i is of a subtype of x_i. The application of a substitution σ to a term t is written $\sigma(t) = t\sigma$.

Function symbols in Σ are intended to model cryptographic primitives, including generation of random data like e.g. nonces or keys. Identities will typically be represented by constants (deterministic function symbols without arguments). The symbolic model is equipped with a *deduction relation* $\vdash \subseteq 2^{\mathsf{Terms}} \times \mathsf{Terms}$ that models the information available to a symbolic adversary. $T \vdash t$ means that a formal adversary can build t out of T, where t is a term and T a set of terms. We say that t is *deducible* from T. Deduction relations are typically defined through deduction systems.

Definition 1. *A* deduction system $\mathcal{D}$ *is a set of rules* $\frac{t_1 \quad \cdots \quad t_n}{t}$ *such that* $t_1, \ldots, t_n, t \in \mathsf{Terms}(\Sigma, \mathcal{T}, \leq)$. *The deduction relation* $\vdash_{\mathcal{D}} \subseteq 2^{\mathsf{Terms}} \times \mathsf{Terms}$ *associated to* $\mathcal{D}$ *is the smallest relation satisfying:*

- $T \vdash_{\mathcal{D}} t$ *for any* $t \in T \subseteq \mathsf{Terms}(\Sigma, \mathcal{T}, \leq)$
- *If* $T \vdash_{\mathcal{D}} t_1\sigma, \ldots T \vdash_{\mathcal{D}} t_n\sigma$ *for some substitution* σ *and* $\frac{t_1 \quad \cdots \quad t_n}{t} \in \mathcal{D}$ *then* $T \vdash_{\mathcal{D}} t\sigma$.

We may omit the subscript $\mathcal{D}$ in $\vdash_{\mathcal{D}}$ when it is clear from the context. For all deduction systems $\mathcal{D}$ in this paper we require $\frac{}{g_\tau^l}$ for all garbage symbols $g_\tau \in \Sigma$ and $l \in \mathsf{labelsA}$.

Let σ be a substitution. We say that $\frac{t_1\sigma \quad \cdots \quad t_n\sigma}{t\sigma}$ is an *instantiation* of a rule $\frac{t_1 \quad \cdots \quad t_n}{t} \in \mathcal{D}$. Since we require the deduction relations in this paper to be efficiently decidable, we can, if we have $T \vdash t$, w.l.o.g. always find a sequence $\pi = T \overset{\alpha_1}{\to} T_1 \overset{\alpha_2}{\to} \cdots \overset{\alpha_n}{\to} T_n$ such that for all $i \in \{1, \ldots, n\}$: (i) $\alpha_i = \frac{t_1 \quad \cdots \quad t_n}{t'}$ is an instantiation of rules from $\mathcal{D}$, (ii) $t_1, \ldots, t_n \in T_{i-1}$, (iii) $t' \notin T_{i-1}$, (iv) $t' \in T_i$ and (v) $t \in T_n$. We call π a *deduction proof* for $T \vdash t$.

From now on we denote a symbolic model $\mathcal{M}$ as a tuple $(\mathcal{T}, \leq, \Sigma, \mathcal{D})$ where $\mathcal{T}$ is the set of data types, $\leq$ the subtype relation, Σ signature and $\mathcal{D}$ the deduction system. For all symbolic models defined in this paper we omit the garbage symbols and the corresponding reduction rules for the sake of brevity.

4. IMPLEMENTATION

An implementation $\mathcal{I}$ of a symbolic model is a family of tuples $(M_\eta, [\![\cdot]\!]_\eta, len_\eta, \mathsf{open}_\eta, \mathsf{valid}_\eta)_\eta$ for $\eta \in \mathbb{N}$. We usually omit the security parameter and just write $(M, [\![\cdot]\!], len, \mathsf{open}, \mathsf{valid})$ for an implementation.

M is a Turing Machine which provides concrete algorithms working on bit-strings for the function symbols in the signature. $[\![\cdot]\!] : \mathcal{T} \to 2^{\{0,1\}^*}$ is a function that maps each type to a set of bitstrings. $len : \mathsf{Terms} \to \mathbb{N}$ computes the length of a term if interpreted as a bitstring. With open the implementation provides an algorithm to interpret bitstrings as terms. valid is a predicate which states whether a concrete use of the implementation is valid. For example, a correct use of an implementation might exclude the creation of key cycles or dynamic corruption of keys from the valid use cases. More precisely we require the following from an implementation:

We assume a non-empty set of bitstrings $[\![\tau]\!] \subseteq \{0,1\}^\eta$ for each type $\tau \in \mathcal{T}$. For the base type $\top$, we assume $[\![\top]\!] = \{0,1\}^*$ and for any pair of types $\tau, \tau' \in \mathcal{T}$ with $\tau \leq \tau'$ we require $[\![\tau]\!] \subsetneq [\![\tau']\!]$ and $[\![\tau]\!] \cap [\![\tau']\!] = \emptyset$ otherwise (i.e., if $\tau \not\leq \tau'$). We write $[\![\mathcal{T}]\!]$ for $\cup_{\tau \in \mathcal{T} \setminus \{\top\}} [\![\tau]\!]$. Later, we often make use of a function $\langle c_1, \ldots, c_n, \tau \rangle$ that takes a list of bitstrings $c_1, \ldots, c_n$ and a type τ and encodes $c_1, \ldots, c_n$ as a bitstring $c' \in [\![\tau]\!]$. We assume that this encoding is bijective, i.e., we can uniquely parse c' as $\langle c_1, \ldots, c_n, \tau \rangle$ again.

We require the Turing Machine M itself to be deterministic. However, each time it is run, it is provided with a random tape $\mathcal{R}$. More specifically, we require for each $f \in \Sigma$ with $\mathrm{ar}(f) = \tau_1 \times \ldots \times \tau_n \to \tau$ that is not a garbage symbol that for input f M calculates a function $(M\ f)$ with domain $[\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!] \times \{0,1\}^*$ and range $[\![\tau]\!]$. The runtime of M and $(M\ f)$ has to be polynomial in the length of its input. Intuitively, to generate a bitstring for a term $t = f^l(t_1, \ldots, t_n)$ we apply $(M\ f)$ to the bitstrings generated for the arguments t_i and some randomness (which might not be used for deterministic function symbols). We call the resulting bitstring *concrete interpretation* of t. The randomness is provided by the $\mathsf{generate}$ function introduced in Section 4.2.

4.1 Interpretations

In cryptographic applications functions are often randomized and the same random coins may occur in different places within the same term. This is the case for instance when the same nonce occurs twice in the same term. We use a (partially defined) mapping $L : \{0,1\}^* \to \mathsf{HTerm}$ from bit-strings to *hybrid terms* to record this information. A hybrid term is either a garbage term or $f^l(c_1, \ldots, c_n)$ where $f \in \Sigma$ is a function symbol of arity n applied to bit-strings

$\mathsf{generate}_{M,\mathcal{R}}(t, L)$:
  if for some $c \in \mathrm{dom}(L)$ we have $L[[c]] = t$ then
    return c
  else
    for $i \in \{1, \ldots, n\}$ let $(c_i, L) := \mathsf{generate}_{M,\mathcal{R}}(t_i, L)$
    let $r := \mathcal{R}(t)$
    let $c := (M\ f)(c_1, \ldots, c_n; r)$
    let $L(c) := f^l(c_1, \ldots, c_n))$ $(l \in \mathsf{labelsH})$
    return (c, L)

Figure 1: The generate function (t is of the form $f^l(t_1, \ldots, t_n)$ (with possibly $n = 0$ and no label l for deterministic function symbols f)).

$c_i \in \{0,1\}^*$. By $\mathrm{dom}(L) \subseteq 2^{\{0,1\}^*}$ we denote the domain of L, i.e. the set of bit-strings for which L is defined. The mapping L induces an interpretation of bit-strings as terms. We define the *interpretation of bitstring* $c \in \mathrm{dom}(L)$ *with respect to a mapping* L as $L[[c]] := f^l(L[[c_1]], \ldots, L[[c_n]])$ if $L(c) = f^l(c_1, \ldots, c_n)$ and $L[[c]] := L(c)$ if $L(c)$ is a garbage term. We say that a mapping L is complete, if for all $(c, f^l(c_1, \ldots, c_n)) \in L$ $c_1, \ldots, c_n \in \mathrm{dom}(L)$. Note that $L[[c]]$ is only defined if L is complete.

4.2 Generating function

Given a mapping L we define a generating function that associates a concrete semantics for terms (given the terms already interpreted in L).

The generation function uses a random tape $\mathcal{R}$ as a source of randomness for M when generating the concrete interpretation of terms. We assume that there is an algorithm $\mathcal{R}(t)$ which maps a term t to a bitstring $r \in \{0,1\}^\eta$ that should be used as the randomness when t is generated. Even changing only one label in t leads to a changed term t' for which different randomness will be used. Figure 1 defines the generate function given a closed term $t = f^l(t_1, \ldots, t_n)$ and a mapping L.

Note that $\mathsf{generate}_{M,\mathcal{R}}(t, L)$ not only returns a bit-string c associated to t but also updates L (to remember, for example, the value associated to t). Note also that $\mathsf{generate}_{M,\mathcal{R}}$ depends on M and the random tape $\mathcal{R}$. When needed, we explicitly show this dependency, but in general we avoid it for readability. If a mapping L is complete, then for $(c, L') := \mathsf{generate}(t, L)$, L' is complete. Furthermore, the generate function requires that, for given inputs t, L, the following holds: For all $t' := f^l(t_1, \ldots, t_n) \in st(t)$ where $l \in \mathsf{labelsA}$ we find a $c \in \mathrm{dom}(L)$ s.t. $L[[c]] = t'$ and t doesn't contain garbage symbols carrying honest labels. This guarantees that all bitstrings introduced by the generate function correspond to the application of non-garbage function symbols carrying honest labels.

4.3 Parsing function

Conversely, we require the implementation to define a function parse to convert bit-string into terms. The function takes a bit-string c and a mapping L as input and returns a term t and an extended mapping L.

For parsing functions we require the concrete structure in Figure 2 (where $\mathsf{open} : \{0,1\}^* \times \mathsf{libs} \to \{0,1\}^* \times \mathsf{HTerm}$ a function that on call $\mathsf{open}(c, L)$ parses the bitstring c in presence of the library L and returns its hybrid interpretation).

$\mathsf{parse}(c, L)$:
  if $c \in \mathrm{dom}(L)$ then
    return $(L[[c]], L)$
  else
    let $L_h := \{(\overline{c}, f^l(\ldots)) \in L : l \in \mathsf{labelsH}\}$
    let $L := \left(\bigcup_{(\overline{c},\cdot)\in L} \mathsf{open}(\overline{c}, L_h)\right)$
    let $G := \left\{(\overline{c}, g_\top^{l(\overline{c})})\right\}$ $(l(\overline{c}) \in \mathsf{labelsA})$
    do
      let $L := (L \setminus G) \cup \left(\bigcup_{(\overline{c},\cdot)\in G} \mathsf{open}(\overline{c}, L_h)\right)$
      let $G := \left\{(\overline{c}, g_\top^{l(\overline{c})}) : (\overline{c}', f(\ldots, \overline{c}, \ldots)) \in L \text{ and } \overline{c} \notin \mathrm{dom}(L)\right\}$
    while $G \neq \emptyset$
    return $(L[[c]], L)$

Figure 2: The parsing function.

The exact definition of parse is left unspecified, as it depends on the particular behavior of open which is provided by a concrete implementation. We require this structure for the parsing function to provide a concrete context in which the open function of different implementations can be composed. Note that the open function is only allowed to use honestly generated bitstrings when dealing with a term. We will furthermore only use open functions later that ignore "foreign" bitstrings in the given library, i.e., bitstrings that are of a data type that is not part of the implementation open belongs to. Due to these properties the composition of open functions is commutative. This is important for our composition theorems later. Furthermore, we think that it meets the intuition that the composition of different implementations should be commutative.

4.4 Good implementation

Until now we have not restricted the behavior of implementations in any way. However, there are some properties we will need to hold for every implementation. We describe these properties in this section and say that a *good implementation* is one that satisfies all of them.

We stipulate that a good implementation is *length regular*, i.e., $len(f^l(t_1, \ldots, t_n)) := |(M\ f)(c_1, \ldots, c_n; r)|$ depends only on the length of the arguments c_i (which are the computational interpretations of the symbolic arguments t_i). Having such a length function is equivalent to having a set of length functions $len_f : \mathbb{N}^n \to \mathbb{N}$ for each function symbol $f \in \Sigma$ with n arguments. We need this to generically compose length functions of different implementations in Section 6.

We now explain what it means for an implementation to be collision free. A collision occurs if during a call of $\mathsf{generate}_{M,\mathcal{R}}(t, L)$ an execution of M yields a bitstring c that is already in the domain of L. Since the library L has to be well-defined, we can either overwrite the old value $L(c)$ with the new one or discard the new value. Both variants are problematic:

Overwriting changes the behavior of parse (i.e., bitstrings may now be parsed differently). This might have severe consequences. Imagine that the overwritten bitstring was an honestly signed message. Now this signature looks like the signature of a different message symbolically; possibly like a forgery. Note that this would not be a weakness of signatures

$\mathsf{generate}'_{M,\mathcal{R}}(t, L)$:
  if for some $c \in \mathrm{dom}(L)$ we have $L[[c]] = t$ then
    return c
  else
    for $i \in \{1, n\}$ let $(c_i, L) := \mathsf{generate}'_{M,\mathcal{R}}(t_i, L)$
    let $r := \mathcal{R}(t)$
    let $c := (M\ f)(c_1, \ldots, c_n; r)$
    if $c \in \mathrm{dom}(L)$ then
      exit game with return value 1 (collision)
    let $L(c) := f^l(c_1, \ldots, c_n))$ $(l \in \mathsf{labelsH})$
    return (c, L)

Figure 3: A collision-aware generate function.

but of the fact that collisions can be found for bitstrings corresponding to the signed terms. Discarding means that a bitstring c generated for a term t might not be parsed as t later which might wrongfully prevent the adversary from winning the soundness game.

Since we also need transparent implementations to be collision free we and still want the notion of collision freeness to be composable later, we need to fix a set of functions that reflect the capability of the adversary to pick arbitrary bitstrings for arguments of $\top$.

Definition 2 (Supplementary transparent functions). *For a set of bitstrings $\mathcal{B} \subseteq \{0,1\}^*$ we call define the transparent model $\mathcal{M}^{\mathsf{tran}}_{\mathsf{supp}}(\mathcal{B})$ as follows:*

- $\mathcal{T}^{\mathsf{tran}}_{\mathsf{supp}} := \{\top, \tau^{\mathsf{tran}}_{\mathsf{supp}}\}$. $\tau^{\mathsf{tran}}_{\mathsf{supp}}$ *is a subtype of* $\top$.
- $\Sigma^{\mathsf{tran}}_{\mathsf{supp}} := \{f_c : c \in \mathcal{B}\}$ *(all function symbols are deterministic)*
- $\mathcal{D}^{\mathsf{tran}}_{\mathsf{supp}} := \{\frac{}{f_c()} : c \in \mathcal{B}\}$

and an implementation $\mathcal{I}^{\mathsf{tran}}_{\mathsf{supp}}(\mathcal{B})$ as follows:

- $[\![\tau^{\mathsf{tran}}_{\mathsf{supp}}]\!] := \mathcal{B}$
- $(M^{\mathsf{tran}}_{\mathsf{supp}}\ f_c)()$ *returns* c
- $(M^{\mathsf{tran}}_{\mathsf{supp}}\ \mathsf{func})(c)$ *returns* f_c *if* $c \in \mathcal{B}$, $\bot$ *otherwise*

Now we can define what collision freeness means.

Definition 3 (Collision-free implementation). *Let $\mathsf{DS}'_{\mathcal{M},\mathcal{I},\mathcal{A}}(\eta)$ be the deduction soundness game from Figure 7 where we replace the* generate *function by the function* generate′ *from Figure 3. We say that an implementation $\mathcal{I}$ is* collision-free *if for all p.p.t. adversaries $\mathcal{A}$*

$$\mathbb{P}\left[\mathsf{DS}_{\mathcal{M}\cup\mathcal{M}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}]\!]),\mathcal{I}\cup\mathcal{I}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}]\!]),\mathcal{A}}(\eta) = 1\right]$$
$$-\mathbb{P}\left[\mathsf{DS}'_{\mathcal{M}\cup\mathcal{M}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}]\!]),\mathcal{I}\cup\mathcal{I}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}]\!]),\mathcal{A}}(\eta) = 1\right]$$

is negligible.

When we compose implementations later we will need that their open functions do not interfere. Intuitively, each open function should stick to opening the bitstrings it is responsible for (i.e., that are of types belonging to the same implementation the open function belongs to). This is reflected in the following definition.

Definition 4 (type-safe implementation). *We say that an implementation $\mathcal{I}$ of a symbolic model $\mathcal{M}$ is* type safe *if*

(i) $\mathsf{open}(c, L) = (c, g^l_\top)$ *for* $l \in \mathsf{labelsA}$ *if* $c \notin [\![\mathcal{T}]\!]$. *("*open *must not deal with foreign bitstrings.")*

(ii) $\mathsf{open}(c, L) = \mathsf{open}(c, L|[\![\mathcal{T}]\!])$ *where* $L|[\![\mathcal{T}]\!] := \{(c, h) \in L : \exists \tau \in \mathcal{T} \setminus \{\top\} : c \in [\![\tau]\!]\}$. *("The behavior of* open *must not be affected by foreign bitstrings in the library.")*

Since we need to simulate parsing later, we require $\mathsf{parse}(c, L)$ (based on open) to run in polynomial time in the size of the library.

Usually, to have computational soundness, we have to restrict the use of the implementation. For example we may only allow static corruption of keys. The purpose of the valid function is exactly this. It gets a trace of queries and outputs a boolean value which states whether the trace is valid or not. To be able to compose the valid functions of different implementations in a meaningful way we require valid to meet the following requirements

A trace $\mathbb{T}$ is a list of queries q. A query is either "init T, H" where T, H are lists of terms, "sgenerate t", or "generate t" where t is a term.

(i) If $\mathsf{valid}(\mathbb{T}+q) = \mathsf{true}$, then $\mathsf{valid}(\mathbb{T}+\hat{q}) = \mathsf{true}$ where $\hat{q}$ is a *variation* of q: If $q =$"generate t", then $\hat{q} =$"generate $\hat{t}$" (analogously for "sgenerate t"). Here, $\hat{t}$ is a variation of t according to the following rule: Any subterm $f^l(t_1, \ldots, t_n)$ of t where $f \notin \Sigma$ is a foreign function symbol may be replaced by $\hat{f}^{\hat{l}}(\hat{t}_1, \ldots, \hat{t}_m)$ where $\hat{f} \notin \Sigma$ is a foreign function symbol and $\hat{t}_i = t_j$ for some $j \in \{1, \ldots, n\}$ (where each t_j may only be used once) or $\hat{t}_i$ does not contain function symbols from Σ. As a special case we may also replace $f^l(t_1, \ldots, t_n)$ with a term $\hat{t}_1$ (i.e., $\hat{f}$ is "empty"). If $q =$"init T, H" then $\hat{q} =$"init $\hat{T}, \hat{H}$" where $T = (t_1, \ldots, t_n)$ and $\hat{T} = (\hat{t}_1, \ldots, \hat{t}_n)$ and $\hat{t}_i$ is a variation of t_i ($\hat{H}$ analogously).

(ii) If $\mathsf{valid}(\mathbb{T} + q) = \mathsf{true}$ and t is a term occurring in q, then $\mathsf{valid}(\mathbb{T}+$"sgenerate t'") $= \mathsf{true}$ for any subterm t' of t.

(iii) valid($\mathbb{T}$) can be evaluated in polynomial time (in the length of the trace $\mathbb{T}$).

Why are these restrictions necessary?

(i) This allows us to replace function symbols with transparent functions and even add or drop arguments during the simulation of a primitive using transparent functions. Intuitively, this requirement is justified since we don't know the semantics of foreign function symbols valid should not: (a) look at the concrete symbols (i.e., function symbols may be replaced), (b) look at the order of arguments (since it doesn't know what the foreign function does, valid shouldn't make decisions based on the order of arguments; also, if the reader accepts (a) a function symbol could be replaced by a semantically equivalent function symbol which just accepts arguments in a different order), (c) depend on the existence of own terms: since the foreign function

might just ignore an argument it wouldn't be meaningful to require its existence, (d) the existence of additional arguments for a foreign function (those could also be hardcoded).

(ii) If a term is valid in general, then any subterm should be valid at least if the adversary doesn't learn it. We need this when we add something to an implementation that features arguments that are hidden from the adversary (i.e., encryptions under honest keys). We cannot simulate those arguments with transparent functions and therefore need to generate them at some point.

(iii) This is needed to efficiently compute $\mathsf{valid}(\mathbb{T})$ during simulations.

5. TRANSPARENT FUNCTIONS

Typical primitives that are usually considered in soundness results include encryption, signatures, hash functions, etc.. Intuitively, such functions are efficiently invertible, and the type of their output can be efficiently determined. An example for such functions are data structures (i.e., pairs, lists, XML documents, etc.). We define and study soundness of such primitives when they are used together with a class of functions which we call *transparent* functions.

Towards this goal we define *transparent symbolic models* and the corresponding *transparent implementation* and show how to extend symbolic models and their implementations with transparent functions in a generic way.

A transparent symbolic model $\mathcal{M}_{\mathsf{tran}} = (\mathcal{T}_{\mathsf{tran}}, \leq_{\mathsf{tran}}, \Sigma_{\mathsf{tran}}, \mathcal{D}_{\mathsf{tran}})$ is a symbolic model where the deduction system is defined as follows (the label is omitted for deterministic function symbols):

$$\mathcal{D}_{\mathsf{tran}} = \left\{ \begin{array}{l} \frac{t_1 \ \cdots \ t_n}{f^l(t_n,\ldots,t_n)} \ l \in \mathsf{labelsA}, f \in \Sigma_{\mathsf{tran}} \\ \\ \frac{f^l(t_1,\ldots,t_n)}{t_i} \ 1 \leq i \leq n, l \in \mathsf{labels}, f \in \Sigma_{\mathsf{tran}} \end{array} \right\}$$

Formally, a transparent implementation of a transparent symbolic model $\mathcal{M} = (\mathcal{T}, \leq, \Sigma, \mathcal{D})$ is an implementation (and thus adhering to the requirements from Section 4.4) $\mathcal{I}_{\mathsf{tran}} = (M_{\mathsf{tran}}, [\![\cdot]\!], len, \mathsf{open}_{\mathsf{tran}}, \mathsf{valid}_{\mathsf{tran}})$ where $\mathsf{open}_{\mathsf{tran}}$ and $\mathsf{valid}_{\mathsf{tran}}$ are defined explicitly below. We require two additional modes of operation, func and proj, for the Turing Machine M_{tran} such that for all $f \in \Sigma$ with $\mathrm{ar}(f) = \tau_1 \times \ldots \times \tau_n \to \tau$

$$(M_{\mathsf{tran}}\ \mathsf{func}) : \{0,1\}^* \to \Sigma \cup \{\bot\}$$
$$(M_{\mathsf{tran}}\ \mathsf{proj}\ f\ i) : \{0,1\}^* \to \{0,1\}^* \cup \{\bot\}$$

and we have for any $c_i \in [\![\tau_i]\!]$, $1 \leq i \leq n$, $r \in \{0,1\}^\eta$

$$(M_{\mathsf{tran}}\ \mathsf{func})((M_{\mathsf{tran}}\ f)(c_1,\ldots,c_n;r)) = f$$
$$(M_{\mathsf{tran}}\ \mathsf{proj}\ f\ i)((M_{\mathsf{tran}}\ f)(c_1,\ldots,c_n;r)) = c_i$$

Furthermore, we require $(M_{\mathsf{tran}}\ \mathsf{func})(c) = \bot$ for all $c \notin [\![\mathcal{T}]\!]$. As expected, M_{tran} is required to run in polynomial time in η for this modes of operation as well.

For transparent implementations we explicitly define the open function $\mathsf{open}_{\mathsf{tran}}$ as in Figure 4. Note that a transparent implementation is automatically type safe according to Definition 4: Property (i) is required above and property (ii) holds since L is not used by $\mathsf{open}_{\mathsf{tran}}$.

We define $\mathsf{valid}_{\mathsf{tran}}(\mathbb{T}) = \mathsf{true}$ for all traces $\mathbb{T}$, i.e., the use of transparent functions is not restricted in any way.

$\mathsf{open}_{\mathsf{tran}}(c, L)$:
  if $c \in [\![\mathcal{T}]\!] \cap \mathrm{dom}(L)$ then
    return $(c, L(c))$
  else if $(M_{\mathsf{tran}}\ \mathsf{func})(c) = \bot$ then
    find unique $\tau \in \mathcal{T}$ s.t. $c \in [\![\tau]\!]$ and
      $c \notin [\![\tau']\!]$ for all $\tau' \in \mathcal{T}$ with $[\![\tau']\!] \subsetneq [\![\tau]\!]$
    return $(c, g_\tau^{l(c)})$ $(l(c) \in \mathsf{labelsA})$
  else
    let $f := (M_{\mathsf{tran}}\ \mathsf{func})(c)$ $(\mathrm{ar}(f) = \tau_1 \times \cdots \times \tau_n \to \tau)$
    if $(M_{\mathsf{tran}}\ \mathsf{proj}\ f\ i)(c) = \bot$ for some $i \in \{1,\ldots,n\}$ then
      return $(c, g_\tau^{l(c)})$ $(l(c) \in \mathsf{labelsA})$
    else
      for $i \in \{1,\ldots,n\}$ do
        let $\widetilde{c}_i := (M_{\mathsf{tran}}\ \mathsf{proj}\ f\ i)(\widetilde{c})$
      return $(c, f^{l(c)}(c_1,\ldots,c_n))$ $(l(c) \in \mathsf{labelsA})$

Figure 4: Parsing algorithm for a transparent implementation.

6. COMPOSITION

We next explain how to generically compose two symbolic models and their corresponding implementations.

Let $\mathcal{M}_1 = (\mathcal{T}_1, \leq_1, \Sigma_1, \mathcal{D}_1)$ and $\mathcal{M}_2 = (\mathcal{T}_2, \leq_2, \Sigma_2, \mathcal{D}_2)$ be symbolic models and $\mathcal{I}_1 = (M_1, [\![\cdot]\!]_1, len_1, \mathsf{open}_1, \mathsf{valid}_1)$ and $\mathcal{I}_2 = (M_2, [\![\cdot]\!]_2, len_2, \mathsf{open}_2, \mathsf{valid}_2)$ implementations of $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively.

We say that that $(\mathcal{M}_1, \mathcal{I}_1)$ and $(\mathcal{M}_2, \mathcal{I}_2)$ are *compatible* if $\mathcal{M}_1$ and $\mathcal{M}_2$ as well as $\mathcal{I}_1$ and $\mathcal{I}_2$ meet the requirements for compositions of symbolic models and implementations stated below respectively.

We define the composition of $\mathcal{M}' = \mathcal{M}_1 \cup \mathcal{M}_2$ of $\mathcal{M}_1$ and $\mathcal{M}_2$ if

(i) $\Sigma_1 \cap \Sigma_2 = \{g_\top\}$

(ii) $\mathcal{T}_1 \cap \mathcal{T}_2 = \{\top\}$

and then have $\mathcal{T}' := \mathcal{T}_1 \cup \mathcal{T}_2$, $\leq' := \leq_1 \cup \leq_2$, $\Sigma' := \Sigma_1 \cup \Sigma_2$ and $\mathcal{D}' := \mathcal{D}_1 \cup \mathcal{D}_2$.

The corresponding implementations $\mathcal{I}_1 = (M_1, [\![\cdot]\!]_1, len_1, \mathsf{open}_1, \mathsf{valid}_1)$ and $\mathcal{I}_2 = (M_2, [\![\cdot]\!]_2, len_2, \mathsf{open}_2, \mathsf{valid}_2)$ can be composed if the following requirements are met:

(i) For all types $\tau_1 \in \mathcal{T}_1 \setminus \{\top\}$, $\tau_2 \in \mathcal{T}_2 \setminus \{\top\}$ we have $[\![\tau_1]\!] \cap [\![\tau_2]\!] = \emptyset$.

(ii) The composition of $\mathcal{I}_1$ and $\mathcal{I}_2$ (as defined below) is a collision-free implementation of $\mathcal{M}'$ (Definition 3).

We then define the composition $\mathcal{I}' = \mathcal{I}_1 \cup \mathcal{I}_2$ as follows:

The Turing machine $(M'\ f)$ returns $(M_1\ f)$ for $f \in \Sigma_1$ and $(M_2\ f)$ if $f \in \Sigma_2$. This is non-ambiguous due to i.[1] Similarly, for all $\tau \in \mathcal{T}_1$ we set $[\![\tau]\!]' := [\![\tau]\!]_1$ and analogously for $\tau \in \mathcal{T}_2$. Note that $[\![\top]\!] = [\![\top]\!]_1 = [\![\top]\!]_2 = \{0,1\}^*$. Since implementations are required to be length regular we can also compose the length functions len_1 and len_2 in a straightforward way to get len'.

To compose the open functions we define

[1]Here we assume that the membership problem is efficiently decidable (since M' has to run in polynomial time). This can be achieve w.l.o.g. with a suitable encoding for the function symbols.

$(\mathsf{open}_1 \circ \mathsf{open}_2)(c, L)$:
  let $(c, t) := \mathsf{open}_1(c, L)$
  if $t = g^l_\top$ for some $l \in \mathsf{labelsA}$ then
    return $\mathsf{open}_2(c, L)$
  else
    return (c, t)

and consequently $\mathsf{open}' := \mathsf{open}_1 \circ \mathsf{open}_2$. Furthermore we set $\mathsf{valid}'(\mathbb{T}) := \mathsf{valid}_1(\mathbb{T}) \wedge \mathsf{valid}_2(\mathbb{T})$ where $\wedge$ is the conjunction.

Finally, we have to show that the composed implementation $\mathcal{I}'$ is a good implementation of the composed symbolic model $\mathcal{M}'$ by checking the requirements from Section 4: Requirements for types hold since they hold on $\mathcal{T}_1$ and $\mathcal{T}_2$ and by requirement ii for the composition of symbolic models. The latter furthermore implies $[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!] = \emptyset$. Due to this and since $\mathcal{I}_1$ and $\mathcal{I}_2$ are type safe, $\mathsf{open}' = \mathsf{open}_1 \circ \mathsf{open}_2 = \mathsf{open}_2 \circ \mathsf{open}_1$. Furthermore, $\mathcal{I}'$ is type safe since $\mathcal{I}_1$ and $\mathcal{I}_2$ are. The requirements for valid carry over obviously as well as the length regularity.

Unfortunately, it is not always straightforward to check requirement ii for the composition of implementations. However, we are going to show that ii is satisfied if, additionally to the other requirements some additional requirement for the valid functions of $\mathcal{I}_1$ and $\mathcal{I}_2$ hold. This is reflected in the following Lemma 1. We note that the valid predicates of the primitives we introduce later (public key encryption, signatures, secret key encryption, macs and hashes) all meet the additional requirements of Lemma 1. Hence we do not need to proof collision freeness separately when composing those.

LEMMA 1. *Let $\mathcal{M}_1$, $\mathcal{M}_2$ be symbolic models with implementations $\mathcal{I}_1$ and $\mathcal{I}_2$ respectively. If in addition to requirements i, ii and i the following requirements for $\mathsf{valid}'(\mathbb{T}) := \mathsf{valid}_1(\mathbb{T}) \wedge \mathsf{valid}_2(\mathbb{T})$ hold:*

1. *Let $\hat{\mathbb{T}}$ be $\mathbb{T}$ with all silent generate queries "sgenerate t" replaced with normal generate queries "generate t". Then $\mathsf{valid}'(\mathbb{T}) \Rightarrow \mathsf{valid}(\hat{\mathbb{T}})$.*
2. *Let $x \in \{1, 2\}$. If valid_x("init T, H"), then for each $t \in T \cup H$ all function symbols in t are from Σ_x or no function symbol in t is from Σ_x.*
3. *Let $x \in \{1, 2\}$. Let $\hat{\mathbb{T}}$ be an expansion of $\mathbb{T} = q_1 + \cdots + q_n$ in the following sense: A $q_i =$"generate t" for $i \in \{1, \ldots, n\}$ is be replaced with $q_i^1 + \cdots + q_i^m$ where $q_i^j =$"generate t_j", $t_j \in st(t)$ and t_j does not contain function symbols from Σ_x for $j \in \{1, \ldots, m\}$. Then $\mathsf{valid}_x(\mathbb{T}) \Rightarrow \mathsf{valid}_x(\hat{\mathbb{T}})$.*

then $(\mathcal{M}_1, \mathcal{I}_1)$ and $(\mathcal{M}_2, \mathcal{I}_2)$ are compatible

PROOF. Note that $\mathcal{I}_1$ and $\mathcal{I}_2$ are collision free since we are only dealing with good implementations. We prove the lemma with a sequence of games:

Game 0.

In Game 0 $\mathcal{A}$ plays $\mathsf{DS}'_{\mathcal{M}' \cup \mathcal{M}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}']\!]), \mathcal{I}' \cup \mathcal{I}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}']\!]), \mathcal{A}}(\eta)$ from Definition 3.

Game 1.

In Game 1 $\mathcal{A}$ plays Game 0 where the $\mathsf{generate}$ functions aborts only for collisions of bitstrings from $[\![\mathcal{T}_1]\!]$, i.e., we use a function $\mathsf{generate}'$ similar to that from Figure 3 with:

  if $c \in \mathrm{dom}(L) \cap [\![\mathcal{T}_1]\!]$ then
    exit game with return value 1 (collision)

Claim: Game 1 and Game 0 are indistinguishable.

Since the only difference between the games is the changed exit condition, it suffices to look at the probability of a 1-output: We show for any adversary $\mathcal{A}$ that wins Game 0 but not Game 1 with non-negligible probability that it can be used to break the collision-freeness of $\mathcal{I}_2$. Concretely, we provide a simulator $\mathcal{B}$ that plays

$$\mathsf{DS}'_{\mathcal{M}_2 \cup \mathcal{M}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}_2]\!]), \mathcal{I}_2 \cup \mathcal{I}^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}_2]\!]), \mathcal{A}}(\eta)$$

and simulates Game 0 for $\mathcal{A}$.

Setup. $\mathcal{B}$ maintains a couple of global states: $S := \emptyset$ to keep track of the terms generated for $\mathcal{A}$, $L := \emptyset$ to simulate the library for $\mathcal{I}_1$, $\Lambda := \emptyset$ is a partially defined mapping from transparent functions f_c ($c \in [\![\mathcal{T}_1]\!]$) to terms, $\mathcal{R} \leftarrow \{0,1\}^*$ is the random tape of and $\mathbb{T}$ the trace of queries received from $\mathcal{A}$.

Queries. Using the two helper functions $\mathsf{generate}_{\mathcal{B}}$ and $\mathsf{parse}_{\mathcal{B}}$, which will be defined below, $\mathcal{B}$ deals with the queries of $\mathcal{A}$ as follows (note that $\mathcal{A}$ doesn't send parameters in the collision game according to def Definition 3):

- "init T, H": $\mathcal{B}$ adds "init T, H" to $\mathbb{T}$ and returns 0 if $\mathsf{valid}'(\mathbb{T}) = \mathsf{false}$. Otherwise, it computes $C := \{\mathsf{generate}_{\mathcal{B}}(t) : t \in T\}$ and $\mathsf{generate}_{\mathcal{B}}(t)$ for all $t \in H$ and sends C to $\mathcal{A}$.
- "generate t": $\mathcal{B}$ adds "generate t" to $\mathbb{T}$ and returns 0 if $\mathsf{valid}'(\mathbb{T}) = \mathsf{false}$. Otherwise, it adds t to S and sends $\mathsf{generate}_{\mathcal{B}}(t)$ to $\mathcal{A}$.
- "sgenerate t": $\mathcal{B}$ returns 0 if $\mathsf{valid}'(\mathbb{T}+$"sgenerate t") $= \mathsf{false}$. Otherwise, it computes $\mathsf{generate}_{\mathcal{B}}(t)$ (but does not send the result to $\mathcal{A}$).
- "parse c": $\mathcal{B}$ computes $t := \mathsf{parse}_{\mathcal{B}}(c)$. If $S \vdash' t$, it sends t to $\mathcal{A}$. Otherwise it returns 1.

The $\mathsf{generate}_{\mathcal{B}}$ function. While $\mathcal{B}$ can compute $\mathcal{I}_1$ itself, it has only access to $\mathcal{I}_2$ via the game it is playing. Concretely, it can generate bitstrings for function symbols from Σ_1 directly (using the given machine M_1), while it has to query bitstrings for function symbols from the complement

$$\overline{\Sigma_1} := \Sigma' \setminus \Sigma_1 = \Sigma_2 \cup \{f_c : c \in [\![\mathcal{T}']\!]\}$$

This procedure is reflected in the function $\mathsf{generate}_{\mathcal{B}}$ from Figure 5. $\mathsf{generate}_{\mathcal{B}}$ updates the states L and Λ of $\mathcal{B}$ and makes use of the random tape $\mathcal{R}$. We write $t \in \Sigma$ if the term t contains only function symbols $f \in \Sigma$.

The $\mathsf{parse}_{\mathcal{B}}$ function. Analogously, $\mathcal{B}$ needs to distinguish bitstrings from the domain of $\mathcal{I}_1$ from bitstrings that have to be parsed by the game played by $\mathcal{B}$. It uses the function $\mathsf{parse}_{\mathcal{B}}$ from Figure 6 to handle parsing requests.

Indistinguishability. Finally, we argue that the simulation perfectly simulates Game 1 and that it can only be distinguished from Game 0 by an adversary that breaks the collision-freeness of $\mathcal{I}_2$. More concretely, we show

(i) $\mathcal{B}$ provides a perfect simulation for the output send to $\mathcal{A}$.

(ii) If the trace $\mathbb{T}_{\mathcal{A}}$ of $\mathcal{A}$'s queries is valid (i.e., $\mathsf{valid}'(\mathbb{T}_{\mathcal{A}}) = \mathsf{true}$), then the trace $\mathbb{T}_{\mathcal{B}}$ of $\mathcal{B}$'s queries is valid (i.e., $\mathsf{valid}_2(\mathbb{T}_{\mathcal{B}}) = \mathsf{true}$).

$\mathsf{generate}_{\mathcal{B}}(t)$:
  if $t \in \Sigma_1$ then
    let $(c, L) := \mathsf{generate}'_{M_1,\mathcal{R}}(t, L)$
    return c
  else if $t \in \overline{\Sigma_1}$ then
    return "generate t"
  else
    if $f \in \Sigma_1$ then
      for $i \in \{1, \ldots, n\}$ do
        let $c_i := \mathsf{generate}_{\mathcal{B}}(t_i)$
      if $L(c) = f^l(c_1, \ldots, c_n)$ for some c then
        return c
      else
        let $r := \mathcal{R}(f^l(t_1, \ldots, t_n))$
        let $c := (M_1\ f)(c_1, \ldots, c_n; r)$
        if $c \in \mathrm{dom}(L)$ then
          exit game with return value 1 (collision)
        else
          let $L(c) := f^l(c_1, \ldots, c_n)$
          return c
    else (i.e., $f \in \overline{\Sigma_1}$)
      for $i \in \{1, \ldots, n\}$ do
        if $t_i \in \overline{\Sigma_1}$ then
          $\tilde{t}_i := t_i$
        else
          let $c_i := \mathsf{generate}_{\mathcal{B}}(t_i)$
          let $\Lambda(f_{c_i}) := t_i$
          $\tilde{t}_i := f_{c_i}$
      return"generate $f^l(\tilde{t}_1, \ldots, \tilde{t}_n)$"

Figure 5: generate function used by the simulator $\mathcal{B}$. $t = f^l(t_1, \ldots, t_n)$ for a label $l \in \mathsf{labelsH}$. $\mathsf{generate}'$ is the collision-aware generate function from Figure 3.

$\mathsf{parse}_{\mathcal{B}}(c)$:
  if $c \in [\![\mathcal{T}_1]\!]$ then
    if $c \in \mathrm{dom}(L)$ then
      let $f^l(c_1, \ldots, c_n) := L(c)$
    else
      let $L_h := \{(c, f^l(\ldots)) \in L : l \in \mathsf{labelsH}$
      let $(c, f^l(c_1, \ldots, c_n)) := \mathsf{open}_1(c, L_h)$
      let $L(c) := f^l(c_1, \ldots, c_n)$
    for $i \in \{1, \ldots, n\}$ do
      let $t_i := \mathsf{parse}_{\mathcal{B}}(c_i)$
    return $f^l(t_1, \ldots, t_n)$
  else
    let $t :=$"parse c"
    let $T := \{f_{\hat{c}} : f_{\hat{c}} \in st(t)\}$
    let $\sigma := \emptyset$
    for each $f_{\hat{c}} \in T \cap \mathrm{dom}(\Lambda)$ do
      let $\sigma(f_{\hat{c}}) := \Lambda(f_{\hat{c}})$
    for each bitstring $\hat{c}$ s.t. $f_{\hat{c}} \in T \setminus \mathrm{dom}(\Lambda)$ do
      let $\hat{t} := \mathsf{parse}_{\mathcal{B}}(\hat{c})$
      let $\sigma(f_{\hat{c}}) := \hat{t}$
    return $t\sigma$ (replace each $f_{\hat{c}}$ with $\sigma(f_{\hat{c}})$)

Figure 6: parse function used by the simulator $\mathcal{B}$.

(iii) If a query "parse c" of $\mathcal{A}$ results in a non-DY term, $\mathcal{A}$ wins in the real game and in the simulation.

(iv) If a collision occurs, the simulation and Game 0 behave equivalently or the simulator $\mathcal{B}$ wins its game.

- Proof of (i): We observe that the calls to TMs M_1 and M_2 in Game 0 and in Game 1 coincide. Hence we find a bijection between the used random coins. For parsing we basically decompose the library from Game 0 into the part belonging to $\mathcal{I}_1$ and the part belonging to $\mathcal{I}_2$. Since the open functions are type safe applying them to the corresponding parts will yield the same behavior in both games.

- Proof of (ii): For the "init T, H" query, the simulator $\mathcal{B}$ cannot use $\mathsf{generate}_{\mathcal{B}}$ yet. However, due to requirement 2 for valid in this lemma, $\mathcal{B}$ can split the query into two disjoint parts for $\mathcal{I}_1$ and $\mathcal{I}_2$. Furthermore, we check that the additional requirements for valid capture the additional queries introduced by $\mathsf{generate}_{\mathcal{B}}$: Since $\mathsf{generate}_{\mathcal{B}}$ has to learn the bitstrings for terms from $\mathcal{M}_2$, it cannot use silent generate queries. The trace remains valid due to requirement 1. Additionally requirement 3 allows $\mathsf{generate}_{\mathcal{B}}$ to query the bitstrings for subterms that do not contain function symbols from Σ_1.

- Proof of (iii): This holds since (i) holds and the DY-ness check in the simulation (Game 1) is identical to the one in the real game (Game 0).

- Proof of (iv): First, note that all function symbols that $(M_1\ f)(c_1, \ldots, c_n; r) \neq (M_2\ f')(c'_1, \ldots, c'_m; r')$ for all function symbols $f \in \Sigma_1$, $g \in \Sigma_2$ and all bitstrings $c_1, \ldots, c_n, r, c'_1, \ldots, c'_m, r'$ of proper types. This holds since f and g cannot be of basetype (due to our requirements for symbolic models) and requirement i for composable implementation guarantees that the domains of non-basetype types are disjoint. Analogously, collisions with the supplementary functions cannot occur. Hence every collision is either a collision in the domain $[\![\mathcal{T}_1]\!]$ of $\mathcal{I}_1$ or in the domain $[\![\mathcal{T}_2]\!]$ of $\mathcal{I}_2$. In the first case, the simulation behaves like the real game. In the second case the simulation wins the collision freeness game for $\mathcal{I}_2$ (which may only happen with negligible probability since $\mathcal{I}_2$ is collision free).

This concludes the proof of our claim that Game 0 and Game 1 are indistinguishable.

Game 2.

Analogously to the previous step, we additionally abort only for collisions of bitstrings from $[\![\mathcal{T}_2]\!]$, i.e., we replace

> if $c \in \mathrm{dom}(L) \cap [\![\mathcal{T}_1]\!]$ then
> exit game with return value 1 (collision)

with

> if $c \in \mathrm{dom}(L) \cap [\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!]$ then
> exit game with return value 1 (collision)

in the generate function. Note that $[\![\mathcal{T}_1]\!] \cap [\![\mathcal{T}_2]\!] = \emptyset$ by requirement ii for the composition of symbolic models and requirement i for the composition of implementations. Hence

this game will never abort due to collisions and is equivalent to $\mathsf{DS}_{\mathcal{M}' \cup \mathcal{M}'^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}']\!]), \mathcal{I}' \cup \mathcal{I}'^{\mathsf{tran}}_{\mathsf{supp}}([\![\mathcal{T}']\!]), \mathcal{A}}(\eta)$.

The proof that Game 2 and Game 1 are indistinguishable works exactly like the proof of the indistinguishability of Game 1 and Game 0.

Since Game 2 and Game 0 are indistinguishable, $\mathcal{I}'$ is collision-free. □

7. DEDUCTION SOUNDNESS

In this section we recall the notion of deduction soundness of an implementation with respect to a symbolic model [10]. Informally, the definition considers an adversary that plays the following game against a challenger. The challenger maintains a mapping L between bitstrings and hybrid terms, as defined in Section 4. Recall that the such mappings are used to both generate bitstring interpretations for terms, and also to parse bitstrings as terms (Figures 1,2). Roughly, the adversary is allowed to request to see the interpretation of arbitrary terms, and also to see the result of the parsing function applied to arbitrary bitstrings. Throughout the execution the queries that the adversary makes need to satisfy a predicate valid (which is a parameter of the implementation). The goal of the adversary is to issue a parse request such that the result is a term, that is not deducible from the terms that he had queried in his generate requests: this illustrates the idea that the adversary, although operating with bitstrings, is restricting to only performing Dolev-Yao operations on the bit-strings that it receives.

The details of the game are in Figure 7. Our definition departs from the one of [10] in a few technical aspects. First, we introduce a query init which is used to "initialize" the execution by, for example, generating (and corrupting) keys. The introduction of this query allows for a clearer separation between the phases where keys are created and where they are used, and allows to simplify and clarify what are valid interactions between the adversary and the game.

Secondly, we also allow the adversary to issue sgenerate requests: these are generate requests except that the resulting bitstring is not returned to the adversary. These requests are a technical necessity that help in later simulations, and only strengthen the adversary.

Deduction soundness of an implementation $\mathcal{I}$ with respect to a symbolic model $\mathcal{M}$ for an implementation is defined by considering an adversary who plays the game sketched above against an implementation that mixes $\mathcal{I}$ with transparent functions provided by the adversary. To ensure uniform behavior on behalf of the adversary (e.g. ensure that the adversary does not provide a different set of transparent functions for each different security parameter), and also to satisfy other technical requirements like defining polynomial running time for mixed implementations, we introduce a notion of parametrized transparent functions/models.

Parametrization.

A *parametrized transparent symbolic model* $\mathcal{M}_{\mathsf{tran}}(\nu)$ maps a bitstring ν (the parameter) to a transparent symbolic model. Analogously, a *parametrized transparent implementation* $\mathcal{I}_{\mathsf{tran}}(\nu)$ of $\mathcal{M}_{\mathsf{tran}}$ maps a bitstring ν (the parameters) to a transparent implementation ν where the length of ν is polynomial in the security parameter. We say that a parameter ν is *good* if $\mathcal{I}(\nu)$ is a transparent implementation of $\mathcal{M}_{\mathsf{tran}}(\nu)$ and meets the requirements of a good implementation (i.e., type-safe, randomness-safe, ...) from Section 4.

$\mathsf{DS}_{\mathcal{M}(\nu),\mathcal{I}(\nu),\mathcal{A}}(\eta)$:
- let $S := \emptyset$ (set of requested terms)
- let $L := \emptyset$ (library)
- let $\mathbb{T} := \emptyset$ (trace of queries)
- $\mathcal{R} \leftarrow \{0,1\}^*$ (random tape)

Receive parameter ν from $\mathcal{A}$

on request "init T, H" do
- add "init T" to $\mathbb{T}$
- if valid($\mathbb{T}$) then
  - let $S := S \cup T$
  - let $C := \emptyset$ (list of replies)
  - for each $t \in T$ do
    - let $(c, L) := \mathsf{generate}_{M,\mathcal{R}}(t, L)$
    - let $C := C \cup \{c\}$
  - for each $t \in H$ do
    - let $(c, L) := \mathsf{generate}_{M,\mathcal{R}}(t, L)$
  - send C to $\mathcal{A}$
- else
  - return 0 ($\mathcal{A}$ is invalid)

on request "sgenerate t" do
- if valid($\mathbb{T}$+"sgenerate t") then
  - let $(c, L) := \mathsf{generate}_{M,\mathcal{R}}(t, L)$

on request "generate t" do
- add "generate t" to $\mathbb{T}$
- if valid($\mathbb{T}$) then
  - let $S := S \cup \{t\}$
  - let $(c, L) := \mathsf{generate}_{M,\mathcal{R}}(t, L)$
  - send c to $\mathcal{A}$
- else
  - return 0 ($\mathcal{A}$ is invalid)

on request "parse c" do
- let $(t, L) := \mathsf{parse}(c, L)$
- if $S \vdash_{\mathcal{D}} t$ then
  - send t to $\mathcal{A}$
- else
  - return 1 ($\mathcal{A}$ produced non-Dolev-Yao term)

Figure 7: Game defining deduction soundness. Whenever $\mathsf{generate}_{M,\mathcal{R}}(t, L)$ is called, the requirements for t are checked (i.e., all subterms of t with adversarial labels must already be in L and t does not contain garbage symbols with honest labels) and 0 is returned if the check fails (i.e., the $\mathcal{A}$ is considered to be invalid).

DEFINITION 5 (DEDUCTION SOUNDNESS). *Let $\mathcal{M}$ be a symbolic model and $\mathcal{I}$ be an implementation of $\mathcal{M}$. We say that $\mathcal{I}$ is a* deduction sound implementation of $\mathcal{M}$ *if for all parametrized transparent symbolic models $\mathcal{M}_{\mathsf{tran}}(\nu)$ and for all parametrized transparent implementations $\mathcal{I}_{\mathsf{tran}}(\nu)$ of $\mathcal{M}_{\mathsf{tran}}$ that are composable with $\mathcal{M}$ and $\mathcal{I}$ (see requirements from Section 6) we have that*

$$\mathbb{P}\left[\mathsf{DS}_{\mathcal{M} \cup \mathcal{M}_{\mathsf{tran}}(\nu), \mathcal{I} \cup \mathcal{I}_{\mathsf{tran}}(\nu), \mathcal{A}}(\eta) = 1\right]$$

is negligible for all probabilistic polynomial time (p.p.t.) ad-

versaries $\mathcal{A}$ sending only good parameters ν where DS *is the deduction soundness game defined in Figure 7. Note that $\mathcal{M} \cup \mathcal{M}_{\mathsf{tran}}(\nu)$ can be generically composed to a parametrized symbolic model $\mathcal{M}'(\nu)$ and parametrized implementation $\mathcal{I}(\nu)$ respectively.*

Collisions.

The deduction soundness game from Figure 7 doesn't prevent collisions. I.e., a query leading to calls of the generate function could produce bitstrings that are already in the library and therefore overwrite a value $L(c)$ with a new one. Note that "parse c" requests can never lead to collisions due to the structure of the parse function (see Figure 2). Fortunately, we can use a collision-free variant of the deduction soundness game.

LEMMA 2. *Let $\mathsf{DS}'_{\mathcal{M}(\nu),\mathcal{I}(\nu),\mathcal{A}}(\eta)$ be the deduction soundness game where we replace the* generate *function by the collision aware generate function from Figure 3. Then no p.p.t. adversary $\mathcal{A}$ can distinguish $\mathsf{DS}_{\mathcal{M}(\nu),\mathcal{I}(\nu),\mathcal{A}}(\eta)$ from $\mathsf{DS}'_{\mathcal{M}(\nu),\mathcal{I}(\nu),\mathcal{A}}(\eta)$ with non-negligible probability. (Note that the transparent functions are already included in $\mathcal{M}(\nu)$ and $\mathcal{I}(\nu)$ here.)*

PROOF. The only difference between DS and DS', and hence the only way to distinguish them, is to produce a collision. However, if collisions could be found with non-negligible probability, this would break the collision freeness of $\mathcal{I}(\nu)$ (we require that $\mathcal{I}(\nu)$ is a good implementation for all parameters ν in the definition of deduction soundness Definition 5). More specifically, we can use any adversary $\mathcal{A}$ that sends a parameter ν and can later distinguish $\mathsf{DS}'_{\mathcal{M}(\nu),\mathcal{I}(\nu),\mathcal{A}}(\eta)$ from $\mathsf{DS}_{\mathcal{M}(\nu),\mathcal{I}(\nu),\mathcal{A}}(\eta)$ to construct an adversary $\mathcal{B}$ that wins the collision freeness game for $\mathcal{I}(\nu)$ (according to Definition 3) whenever $\mathcal{A}$ can distinguish: All queries by $\mathcal{A}$ are forwarded by $\mathcal{B}$ to its own game. If $\mathcal{A}$ finds a collision, $\mathcal{B}$ wins. Otherwise $\mathcal{A}$ cannot distinguish. □

8. COMPOSITION THEOREMS

Our notion of deduction soundness enjoys the nice property of being easily extendable: if an implementation is deduction sound for a given symbolic model, it is possible to add other primitives, one by one, without having to prove deduction soundness, from scratch for the resulting set of primitives.

8.1 Public datastructures

An immediate observation with interesting implications is the following. Consider some symbolic model $\mathcal{M}$ with a deduction sound implementation $\mathcal{I}$. Now, extend $\mathcal{M}$ by a transparent symbolic model $\mathcal{M}_{\mathsf{tran}}$ and $\mathcal{I}$ by a transparent implementation $\mathcal{I}_{\mathsf{tran}}$ of $\mathcal{M}_{\mathsf{tran}}$. Then, the resulting implementation is a deduction sound of $\mathcal{M} \cup \mathcal{M}_{\mathsf{tran}}$.

The intuition behind this result is simple: if $\mathcal{I}$ is sound in presence of arbitrary transparent functions with an implementation selected by the adversary, adding transparent functions with some fixed transparent implementation preserves soundness. This idea is formalized by the following theorem.

THEOREM 1. *Let $\mathcal{M}$ be a symbolic model and $\mathcal{I}$ a deduction sound implementation of $\mathcal{M}$. Furthermore, let $\mathcal{M}_{\mathsf{tran}}$ be a transparent symbolic model and $\mathcal{I}_{\mathsf{tran}}$ a transparent implementation of $\mathcal{M}_{\mathsf{tran}}$. If $\mathcal{M}$ and $\mathcal{I}$ are composable with $\mathcal{M}_{\mathsf{tran}}$ and $\mathcal{I}_{\mathsf{tran}}$ (see Section 6), then $\mathcal{I} \cup \mathcal{I}_{\mathsf{tran}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{tran}}$.*

PROOF. Let $\mathcal{A}$ be a ppt adversary that breaks the deduction soundness of $\mathcal{I} \cup \mathcal{I}_{\mathsf{tran}}$, i.e., by Definition 5 there is a transparent symbolic model $\mathcal{M}^{\mathcal{A}}_{\mathsf{tran}}$ with a transparent implementation $\mathcal{I}^{\mathcal{A}}_{\mathsf{tran}}$ such that

$$\mathbb{P}\left[\mathsf{DS}_{(\mathcal{M}\cup\mathcal{M}_{\mathsf{tran}})\cup\mathcal{M}^{\mathcal{A}}_{\mathsf{tran}},(\mathcal{I}\cup\mathcal{I}_{\mathsf{tran}})\cup\mathcal{I}^{\mathcal{A}}_{\mathsf{tran}},\mathcal{A}}(\eta) = 1\right]$$

is non-negligible. Then clearly for the transparent symbolic model $\mathcal{M}_{\mathsf{tran}} \cup \mathcal{M}^{\mathcal{A}}_{\mathsf{tran}}$ and the transparent implementation $\mathcal{I}_{\mathsf{tran}} \cup \mathcal{I}^{\mathcal{A}}_{\mathsf{tran}}$ this adversary also breaks the deduction soundness of $\mathcal{I}$, i.e.,

$$\mathbb{P}\left[\mathsf{DS}_{\mathcal{M}\cup(\mathcal{M}_{\mathsf{tran}}\cup\mathcal{M}^{\mathcal{A}}_{\mathsf{tran}}),\mathcal{I}\cup(\mathcal{I}_{\mathsf{tran}}\cup\mathcal{I}^{\mathcal{A}}_{\mathsf{tran}}),\mathcal{A}}(\eta) = 1\right]$$

is non-negligible. Since $\mathcal{I}$ is a deduction sound implementation by requirement this concludes our proof. □

8.2 Public key encryption

In this section we define a symbolic model $\mathcal{M}_{\mathsf{PKE}}$ for public key encryption and a corresponding implementation $\mathcal{I}_{\mathsf{PKE}}$ based on a public key encryption scheme (PKE.KeyGen, PKE.Enc, PKE.Dec). We show that composition of $\mathcal{M}_{\mathsf{PKE}}$ and $\mathcal{I}_{\mathsf{PKE}}$ with any symbolic model $\mathcal{M}$ comprising a deduction sound implementation $\mathcal{I}$ preserves this property for the resulting implementation, i.e., $\mathcal{I} \cup \mathcal{I}_{\mathsf{PKE}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{PKE}}$. This result was already stated in [10]. However, since the definition of deduction soundness as well as other parts of the framework (e.g., parse function) were updated, we present an updated proof here.

8.2.1 Computational preliminaries

DEFINITION 6 (PUBLIC KEY ENCRYPTION SCHEME). *A* public key encryption scheme *(*PKE *scheme) is a triple of p.p.t. algorithms* (PKE.KeyGen, PKE.Enc, PKE.Dec).

The key generation algorithm PKE.KeyGen *takes an encoding of the security parameter and some randomness as inputs and generates a pair (ek, dk) containing the* encryption key *ek and the* decryption key *dk.*

The encryption algorithm PKE.Enc *takes three arguments: an encryption key ek, the message $m \in \{0,1\}^*$, and some randomness $r \in \{0,1\}^\eta$. If computes a* ciphertext $c := \mathsf{PKE.Enc}(ek, m, r)$.[2]

The decryption algorithm PKE.Dec *takes a decryption key and a ciphertext as inputs and returns a value from $\{0,1\}^* \cup \{\perp\}$. We require* perfect correctness*, i.e.,*

$$\mathsf{PKE.Dec}(dk, \mathsf{PKE.Enc}(ek, m, r')) = m$$

for all $r, r' \leftarrow \{0,1\}, m \in \{0,1\}^*$ and $(ek, dk) := \mathsf{PKE.KeyGen}(1^\eta, r)$.

DEFINITION 7 (IND-CCA SECURITY OF PKE SCHEMES). *A* PKE *scheme* (PKE.KeyGen, PKE.Enc, PKE.Dec) *is* IND-CCA secure *if for all p.p.t. adversaries $\mathcal{A}$ the probability*

$$\mathbb{P}\left[\mathit{IND\text{-}CCA}^{(\mathsf{PKE.KeyGen},\mathsf{PKE.Enc},\mathsf{PKE.Dec})}_{\mathcal{A}}(\eta) = 1\right]$$

is negligible for the IND-CCA game from Figure 8.

[2]Since the message m is of basetype in symbolic model given below, we require a scheme with message space $\{0,1\}^*$.

$\mathsf{IND\text{-}CCA}_{\mathcal{A}}^{(\mathsf{PKE.KeyGen},\mathsf{PKE.Enc},\mathsf{PKE.Dec})}(\eta)$:
$b \leftarrow \{0,1\}$
$\mathsf{oracles} := \emptyset$

on request "new oracle" do
let $r \leftarrow \{0,1\}^\eta$
let $(ek, dk) := \mathsf{PKE.KeyGen}(1^\eta, r)$
add ek to $\mathsf{oracles}$
let $\mathsf{ciphers}_{ek} := \emptyset$
send ek to $\mathcal{A}$

on request "$\mathcal{O}_{ek}^{enc}(m)$" do
if $ek \notin \mathsf{oracles}$ then
send $\perp$ to $\mathcal{A}$
else
let $r \leftarrow \{0,1\}^\eta$
if $b = 0$ then
let $c := \mathsf{PKE.Enc}(ek, 0^{|m|}, r)$
add (c, m) to $\mathsf{ciphers}_{ek}$
else
send $\mathsf{PKE.Enc}(ek, m, r)$ to $\mathcal{A}$

on request "$\mathcal{O}_{ek}^{dec}(c)$" do
if $ek \notin \mathsf{oracles}$ then
send $\perp$ to $\mathcal{A}$
else
if $b = 0$ then
if $(c, m) \in \mathsf{ciphers}_{ek}$ for some m then
send m to $\mathcal{A}$
else
send $\perp$ to $\mathcal{A}$
else
let dk be the decryption key for ek
send $\mathsf{PKE.Dec}(dk, c)$ to $\mathcal{A}$

on request "guess b'" do
if $b = b'$ then return 1 else return 0

Figure 8: The IND-CCA game for a public key encryption scheme (PKE.KeyGen, PKE.Enc, PKE.Dec)**.**

8.2.2 Symbolic model

We first define the symbolic model $(\mathcal{T}_{\mathsf{PKE}}, \leq_{\mathsf{PKE}}, \Sigma_{\mathsf{PKE}}, \mathcal{D}_{\mathsf{PKE}})$ for public key encryption. The signature Σ_{PKE} features the following function symbols

$$dk_x : \tau_{\mathsf{PKE}}^{\mathsf{dk}_x}$$
$$ek_x : \tau_{\mathsf{PKE}}^{\mathsf{dk}_x} \to \tau_{\mathsf{PKE}}^{\mathsf{ek}_x}$$
$$enc_x : \tau_{\mathsf{PKE}}^{\mathsf{ek}_x} \times \top \to \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}$$

for $x \in \{h, c\}$. The randomized function dk_h of arity $\tau_{\mathsf{PKE}}^{\mathsf{dk}_h}$ returns an honest decryption key. The deterministic function ek_h of arity $\tau_{\mathsf{PKE}}^{\mathsf{dk}_h} \to \tau_{\mathsf{PKE}}^{\mathsf{ek}_h}$ derives an honest encryption key from an honest decryption key. Analogously for corrupted decryption keys (dk_h) and corrupted encryption keys (ek_c). The randomized function enc_h for honest encryptions has arity $\tau_{\mathsf{PKE}}^{\mathsf{ek}_h} \times \top \to \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}$ (enc_c analogously). As above we sometimes write ek_x, dk_x, enc_x for $x \in \{h, c\}$ to reference the honest and the corrupted variants of the functions comprehensively. By abuse of notation, we will often write $ek_x^l()$ instead of $ek_x(dk_x^l())$ where $l \in \mathsf{labelsH}$. To complete the formal definition we set

$$\mathcal{T}_{\mathsf{PKE}} := \{\top, \tau_{\mathsf{PKE}}^{\mathsf{dk}_x}, \tau_{\mathsf{PKE}}^{\mathsf{ek}_x}, \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}\}$$

All introduced types are direct subtypes of the base type $\top$ (this defines $\leq_{\mathsf{PKE}}$). The deduction system captures the security of public key encryption

$$\mathcal{D}_{\mathsf{PKE}} := \left\{ \begin{array}{cc} \multicolumn{2}{c}{\frac{ek_x^l() \quad m}{enc_x^{l_a}(ek_x^l(), m)},} \\ \frac{enc_h^{l_a}(ek_h^l(), m)}{m}, & \frac{enc_c^{\hat{l}}(ek_c^l(), m)}{m} \end{array} \right\}$$

These rules are valid for arbitrary labels $l, \hat{l} \in \mathsf{labels}$ and adversarial labels $l_a \in \mathsf{labelsA}$. Read from top left to bottom right the following intuitions back up the rules:

- The adversary can use any honestly generated key to encrypt some term m.
- The adversary knows the message contained in any adversarial encryption.
- The adversary knows the message contained in any encryption under a corrupted key.

Since we only allow for static corruption we do not need a rule $\frac{enc_h^{\hat{l}}(ek_h^l(), m)}{ek_h^l()}$ although we are going to attach the encryption key to the ciphertext in the implementation (all encryption keys are going to be part of the response to the "init T, H" query by the adversary).

8.2.3 Implementation

We now give a concrete implementation $\mathcal{I}_{\mathsf{PKE}}$ for public key encryption. The implementation uses some IND-CCA secure public key encryption scheme (PKE.KeyGen, PKE.Enc, PKE.Dec). To prevent collisions of ciphertexts, we require that we have $\mathsf{PKE.Enc}(ek, m, r) = \mathsf{PKE.Enc}(ek, m', r')$ only with negligible probability for bitstrings m, m', r given by the adversary and r' uniformly chosen honest randomness . Many PKE schemes meet this requirement directly, e.g., all committing schemes . Furthermore, it is always possible to extend the output of PKE.Enc with a nonce to prevent these collisions. The computable interpretations of, dk_x, ek_x, enc_x (for $x \in \{h, c\}$) are as follows:

- $(M_{\mathsf{PKE}}\ dk_x)(r)$: Let $(ek, dk) := \mathsf{PKE.KeyGen}(1^\eta, r)$. Return $\langle ek, dk, \tau_{\mathsf{PKE}}^{\mathsf{dk}_x} \rangle$.
- $(M_{\mathsf{PKE}}\ ek_x)(\hat{dk})$: Parse $\hat{dk}$ as $\langle ek, dk, \tau_{\mathsf{PKE}}^{\mathsf{dk}_x} \rangle$. Return $\langle ek, \tau_{\mathsf{PKE}}^{\mathsf{ek}_x} \rangle$.
- $(M_{\mathsf{PKE}}\ enc_x)(\hat{ek}, m; r)$: Parse $\hat{ek}$ as $\langle ek, \tau_{\mathsf{PKE}}^{\mathsf{ek}_x} \rangle$. Let $c := \mathsf{PKE.Enc}(ek, m, r)$ and return $\langle c, ek, \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}} \rangle$.

The $\mathsf{valid}_{\mathsf{PKE}}$ *predicate.*

The predicate $\mathsf{valid}_{\mathsf{PKE}}$ guarantees, that all keys that may be used by the adversary later are generated during initialization (i.e., with the init query). We only allow static corruption of keys, i.e., the adversary has to decide which keys are honest and which are corrupted at this stage. Keys may only be used for encryption and decryption. This implicitly prevents key cycles. More formally, based on the current trace $\mathbb{T}$ of all parse and generate requests of the adversary, the predicate $\mathsf{valid}_{\mathsf{PKE}}$ returns true only if the following conditions hold:

$\mathsf{open}_{\mathsf{PKE}}(c, L)$
  if $c \in [\![\mathcal{T}_{\mathsf{PKE}}]\!] \cap \mathrm{dom}(L)$ then
    return $(c, L(c))$
  else if $c = \langle dk, \tau^{\mathsf{dk}_x}_{\mathsf{PKE}} \rangle$ then
    return $(c, g^{l(c)}_{\tau^{\mathsf{dk}_x}_{\mathsf{PKE}}})$
  else if $c = \langle ek, \tau^{\mathsf{ek}_x}_{\mathsf{PKE}} \rangle$ then
    if $\hat{dk} \in \mathrm{dom}(L)$ s.t. $\hat{dk} = \langle ek, dk, \tau^{\mathsf{dk}_x}_{\mathsf{PKE}} \rangle$ then
      return $(c, ek_x(\hat{dk}))$
    else
      return $(c, g^{l(c)}_{\tau^{\mathsf{ek}_x}_{\mathsf{PKE}}})$
  else if $c = \langle c', ek, \tau^{\mathsf{ciphertext}}_{\mathsf{PKE}} \rangle$ and $(\langle ek, \tau^{\mathsf{ek}_x}_{\mathsf{PKE}} \rangle, ek^l_x(\hat{dk}) \in L$ then
    parse $\hat{dk}$ as $\langle ek, dk, \tau^{\mathsf{dk}_x}_{\mathsf{PKE}} \rangle$
    let $m := \mathsf{PKE.Dec}(dk, c')$
    if $m = \bot$ then
      return $(c, g^{l(c)}_{\tau^{\mathsf{ciphertext}}_{\mathsf{PKE}}})$
    else
      return $(c, enc^{l(c)}_x(ek, m))$
  else
    return $(c, g^{l(c)}_{\top})$

Figure 9: Open function for public key encryption.

1. The trace starts with a query "init T, H" (T resp. H may be the empty list). There are no further init queries.

2. The adversary may only generate keys in the init query. Concretely, this is guaranteed by the following rules:

   (a) For the query "init T, H", the function symbols ek_x and dk_x may only occur in a term $t \in T$ (i.e., not as subterms of other terms) of one of the two following types (for $l \in \mathsf{labelsH}$):

   - $t = ek_h(dk^l_h())$ (to generate an honest encryption key)
   - $t = dk^l_c()$ (to generate a corrupted encryption key)

   Any label l for $dk^l_x()$ must be unique in T.

   (b) Any occurrence of $ek^l_x()$ or $dk^l_x()$ in a generate query must have occurred in the init query. $dk^l_x()$ may only occur as argument for ek_x. $ek^l_x()$ may only occur as first argument for enc_x.

3. The adversary must not use the function symbols for encryption (enc_h, enc_c) in the init query.

Checking the implementation.

We first observe that $\mathcal{I}_{\mathsf{PKE}}$ is collision-free (Definition 3): Basically, collisions for keys can only occur with negligible probability since they break the security of the scheme (which is IND-CCA secure). We prevent collisions of ciphertexts by the requirements stated above. Furthermore, it is easy to see that $\mathsf{open}_{\mathsf{PKE}}$ meets the requirements of Definition 4 and that $\mathsf{valid}_{\mathsf{PKE}}$ meets the requirements for valid functions.

8.2.4 PKE composability

THEOREM 2. *Let $\mathcal{M}$ be a symbolic model and $\mathcal{I}$ a deduction sound implementation of $\mathcal{M}$. If $(\mathcal{M}_{\mathsf{PKE}}, \mathcal{I}_{\mathsf{PKE}})$ and $(\mathcal{M},\mathcal{I})$ are compatible (see requirements in Section 6) and the PKE scheme* $(\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *is IND-CCA secure, then $\mathcal{I} \cup \mathcal{I}_{\mathsf{PKE}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{PKE}}$.*

PROOF. Let $\mathcal{A}$ be a p.p.t. adversary and $\mathcal{M}_{\mathsf{tran}}(\nu)$ a parametrized transparent symbolic model with a corresponding parametrized implementation $\mathcal{I}_{\mathsf{tran}}(\nu)$ such that $\mathcal{M}_{\mathsf{tran}}(\nu)$ and $\mathcal{I}_{\mathsf{tran}}(\nu)$ are composable with $\mathcal{M} \cup \mathcal{M}_{\mathsf{PKE}}$ and $\mathcal{I} \cup \mathcal{I}_{\mathsf{PKE}}$ (see requirements in Section 6) for ν send by $\mathcal{A}$. We have to show that $\mathcal{A}$ can win the deduction soundness game $\mathsf{DS}_{(\mathcal{M}\cup\mathcal{M}_{\mathsf{PKE}})\cup\mathcal{M}_{\mathsf{tran}}(\nu),(\mathcal{I}\cup\mathcal{I}_{\mathsf{PKE}})\cup\mathcal{I}_{\mathsf{tran}}(\nu)}(\eta)$ only with negligible probability.

We first explain the basic idea behind this proof. To win the deduction soundness game, $\mathcal{A}$ has to provide a bitstring corresponding to a term t that it does not "know" symbolically, i.e., $\mathcal{A}$ cannot deduce t from the terms it generated previously. Intuitively, by adding public key encryption, exactly one additional opportunity to learn something about such a term is created: $\mathcal{A}$ can retrieve honestly generated encryptions of t under honest encryption keys. In all other encryption scenarios, $\mathcal{A}$ knows the message by the rules of $\mathcal{D}_{\mathsf{PKE}}$. The strategy of this proof consists of two steps: First, we replace honestly generated encryptions of terms by encryptions of 0-bitstrings (using the IND-CCA security of the encryption scheme). Hence the adversary cannot learn anything about the corresponding clear texts (except their length) which eliminates the additionally opportunity for $\mathcal{A}$. Second, we show that encryption can be simulated by transparent functions. Thus, any other way for $\mathcal{A}$ to come up with non-DY terms leads to a non-DY request in the deduction soundness game for $\mathcal{M}$ and $\mathcal{I}$. Since $\mathcal{I}$ is deduction sound by assumption, this concludes our proof.

Concretely, we proof this with a sequence of games.

Game 0.

In Game 0 $\mathcal{A}$ plays the original deduction soundness game $\mathsf{DS}_{(\mathcal{M}\cup\mathcal{M}_{\mathsf{PKE}})\cup\mathcal{M}_{\mathsf{tran}}(\nu),(\mathcal{I}\cup\mathcal{I}_{\mathsf{PKE}})\cup\mathcal{I}_{\mathsf{tran}}(\nu)}(\eta)$.

Game 1.

In Game 1 we replace the generate function by the collision-aware generate function from Figure 3. Since $(\mathcal{I} \cup \mathcal{I}_{\mathsf{PKE}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu)$ is a collision-free implementation Game 0 and Game 1 are indistinguishable by Lemma 2.

Claim: Game 0 and Game 1 are indistinguishable.

Game 2.

In Game 2 we deprive the adversary of any option to learn something from ciphertexts or about honest decryption keys. We replace the honest decryption keys in the library by random bitstrings and add the rule $\frac{ek_x(dk^l_x())}{dk^l_x()}$ to the deduction system. Intuitively, $\mathcal{A}$ doesn't notice the difference since the PKE scheme is IND-CCA secure. More concretely, instead of calling $(M_{\mathsf{PKE}}\ ek_h)(r)$, we pick $r \leftarrow \{0,1\}^\eta$, compute $(ek, dk) := \mathsf{PKE.KeyGen}(1^\eta, r)$, pick $r' \leftarrow \{0,1\}^{|dk|}$, remember dk as the decryption key corresponding to ek and use $\hat{dk} := \langle ek, r', \tau^{\mathsf{dk}_x}_{\mathsf{PKE}} \rangle$ as bitstring for $dk^l_h()$. Additionally, we change the parsing function such that it now uses the remem-

bered corresponding keys for decryption instead of those in the library. Furthermore, to replace the ciphertexts created under honest keys by encryptions of 0, we replace the line

let $c := (M\ f)(c_1, \ldots, c_n; r)$

in the generate function with

if $t = \hat{enc_h^l}(ek_h^l(), m)$ then
  let $c := (M\ enc_h)(c_1, 0^{|c_2|}; r)$
else
  let $c := (M\ f)(c_1, \ldots, c_n; r)$

Note that c_1 and c_2 resemble the bitstrings corresponding to the encryption key and the message respectively. Thus, the generation of bitstrings works exactly as in Game 1 if t is not an encryption under an honest key. Otherwise we generate all subterms as usual but replace the message by a bitstring of zeros of appropriate length.

Claim: Game 1 and Game 2 are indistinguishable.

Let $\mathcal{A}$ be an adversary that can distinguish between playing Game 1 and Game 2. Then we can construct an adversary $\mathcal{B}$ that will win the oracle-based IND-CCA game for PKE scheme ($\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec}$) from Figure 8.

Generating bitstrings. For each query "generate $ek_h(dk_h^l())$", instead of calling the generate function, $\mathcal{B}$ does the following: It requests a new oracle from the IND-CCA game and receives an encryption key ek as well as access to a corresponding encryption oracle, denoted $\mathcal{O}_{ek}^{enc}$, (which either encrypts the given messages or 0-bitstrings of the same lengths) and to a corresponding decryption oracle, denoted $\mathcal{O}_{ek}^{dec}$. $\mathcal{B}$ now picks a random bitstring dk such that $\hat{dk} := \langle ek, dk, \tau_{\mathsf{PKE}}^{\mathsf{dk}_h}\rangle \in [\![\tau_{\mathsf{PKE}}^{\mathsf{dk}_h}]\!]$ and adds $(\hat{dk}, dk_h^l())$ and $(\langle ek, \tau_{\mathsf{PKE}}^{\mathsf{ek}_h}\rangle, ek_h(\hat{dk}))$ to L. Note that by requirement 2 $\mathcal{A}$ will never learn the value of dk. All other types of generate requests are handled by calling the generate function.

To use the provided oracles to generate honest encryptions, $\mathcal{B}$ furthermore uses a modified generate function. Concretely, it replaces the line

let $c := (M\ f)(c_1, \ldots, c_n; r)$

with

if $t = \hat{enc_h^l}(ek_h^l(), m)$ then
  let $c := \langle \mathcal{O}_{c_1}^{enc}(c_2), c_1, \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}\rangle$
else
  let $c := (M\ f)(c_1, \ldots, c_n; r)$

Again, c_1 and c_2 resemble the bitstrings corresponding to the encryption key and the message respectively. Note that this function produces encryptions like the generate function in Game 1 if the oracle encrypts the given message and like the generate function in Game 2 otherwise, i.e., if the oracle encrypts a 0-bitstring.

Parsing bitstrings. $\mathcal{B}$ also has to modify the parsing function to deal with adversarial ciphertexts created with honest keys. More specifically, it removes the lines

parse $\hat{dk}$ as $\langle ek, dk, \tau_{\mathsf{PKE}}^{\mathsf{dk}_x}\rangle$
let $m := \mathsf{PKE.Dec}(dk, c')$

in the $\mathsf{open}_{\mathsf{PKE}}$ function from Figure 9 and adds

if $(\langle ek, \tau_{\mathsf{PKE}}^{\mathsf{ek}_h}\rangle, ek_h^l()) \in L$ (honest key) then
  let $m := \mathcal{O}_{ek}^{dec}(c')$
else
parse $\hat{dk}$ as $\langle ek, dk, \tau_{\mathsf{PKE}}^{\mathsf{dk}_c}\rangle$
  let $m := \mathsf{PKE.Dec}(dk, c')$

For an IND-CCA secure public key encryption scheme this function decrypts like the original $\mathsf{open}_{\mathsf{PKE}}$ function. $\mathcal{B}$ just uses the decryption oracle instead of the decryption key during the simulation. Additionally, if $\mathsf{open}_{\mathsf{PKE}}$ is called for a bitstring $c \in [\![\tau_{\mathsf{PKE}}^{\mathsf{dk}_h}]\!]$, $\mathcal{B}$ parses c as $\langle ek, dk, \tau_{\mathsf{PKE}}^{\mathsf{dk}_h}\rangle$. $\mathcal{B}$ then picks a random message $m \leftarrow \{0,1\}^\eta$ and computes $\mathsf{PKE.Dec}(dk, \mathcal{O}_{ek}^{enc}(m)) = x$. If $x \neq \bot$ and $x = m$, $\mathcal{B}$ sends "guess 1" to the IND-CCA game and wins with overwhelming probability.

Analysis. If the oracle produces real encryptions, $\mathcal{B}$ simulates Game 1 for $\mathcal{A}$. The only difference are the random values for honest decryption keys in the library. Those values are only used when parsing bitstrings. The difference can be detected by $\mathcal{A}$ if it guesses one of the random bitstrings (which can only happen with negligible probability) or if it parses a bitstring belonging to a honest decryption key in Game 1. However, in the latter case, $\mathcal{B}$ wins the IND-CCA game as described above. Hence the simulation is indistinguishable for $\mathcal{A}$ if the oracles produce real encryptions.

If the oracles produces encryptions of zero, $\mathcal{B}$ perfectly simulates Game 2 for $\mathcal{A}$. Hence, every correct guess of $\mathcal{A}$ on which game he is playing leads to a correct guess of $\mathcal{B}$ in the IND-CCA game. Therefore, $\mathcal{A}$ cannot distinguish Game 1 and Game 2.

Game 3.

In Game 3 $\mathcal{A}$ interacts with an adversary $\mathcal{B}$ that plays the deduction soundness game for $\mathcal{M}$ and $\mathcal{I}$ and intuitively simulates Game 2 for $\mathcal{A}$. Basically, $\mathcal{B}$ uses transparent functions to add public key encryption to $\mathcal{M}$. We construct $\mathcal{B}$ as follows:

Transparent symbolic model for public key encryption. We first describe the parametrized transparent symbolic model $\mathcal{M}_{\mathsf{PKE}}^{\mathsf{tran}}(\nu)$ and the corresponding parametrized implementation $\mathcal{I}_{\mathsf{PKE}}^{\mathsf{tran}}(\nu)$ $\mathcal{B}$ will use to simulate $\mathcal{I}_{\mathsf{PKE}}$. We use the data types and subtype relation from $\mathcal{M}_{\mathsf{PKE}}$. ν is expected to be an encoding of a list of label-triple pairs $(l, (ek, dk, dk'))$ ($l \in \mathsf{labels}$) where the triple consist of a keypair ek, dk and an additional value dk' (used to represent honest decryption keys in the library). The signature $\Sigma_{\mathsf{PKE}}^{\mathsf{tran}}$ is the following:

- deterministic $f_{dk_x^l()}$ with $\mathrm{ar}(f_{dk_x^l()}) = \tau_{\mathsf{PKE}}^{\mathsf{dk}_x}$ for all labels $l \in \nu$
- deterministic $f_{ek_x^l()}$ with $\mathrm{ar}(f_{ek_x^l()}) = \tau_{\mathsf{PKE}}^{\mathsf{ek}_x}$ for all labels $l \in \nu$
- randomized $f_{enc_h(ek_h^l(),0^\ell)}$ with $\mathrm{ar}(f_{enc_h(ek_h^l(),0^\ell)}) = \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}$ for all $\ell \in \mathbb{N}$, $l \in \nu$
- randomized $f_{enc_h(ek_h^l(),\cdot)}$ with $\mathrm{ar}(f_{enc_h(ek_h^l(),\cdot)}) = \top \to \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}$ for all $l \in \nu$
- randomized $f_{enc_c((ek_c^l(),\cdot}$ with $\mathrm{ar}(f_{enc_c((ek_c^l(),\cdot)}) = \top \to \tau_{\mathsf{PKE}}^{\mathsf{ciphertext}}$ for all $l \in \nu$

We specify a parametrized implementation $\mathcal{I}^{\mathsf{tran}}_{\mathsf{PKE}}(\nu)$ for $\mathcal{M}^{\mathsf{tran}}_{\mathsf{PKE}}$ and for each $(l,(ek,dk,dk')) \in \nu$ as follows:

- $(M^{\mathsf{tran}}_{\mathsf{PKE}}\ f_{dk^l_x()})()$ returns $\langle ek, dk', \tau^{\mathsf{dk}_x}_{\mathsf{PKE}}\rangle$.
- $(M^{\mathsf{tran}}_{\mathsf{PKE}}\ f_{ek^l_x()})()$ returns $\langle ek, \tau^{\mathsf{ek}_x}_{\mathsf{PKE}}\rangle$.
- $(M^{\mathsf{tran}}_{\mathsf{PKE}}\ f_{enc_h(ek^l_h(),0^\ell)})(r)$ returns $(M_{\mathsf{PKE}}\ enc_h)(\langle ek, \tau^{\mathsf{ek}_x}_{\mathsf{PKE}}\rangle, 0^\ell; r)$.
- $(M^{\mathsf{tran}}_{\mathsf{PKE}}\ f_{enc_h(ek^l_h(),\cdot)})(m;r)$ returns $(M_{\mathsf{PKE}}\ enc_h)(\langle ek, \tau^{\mathsf{ek}_x}_{\mathsf{PKE}}\rangle, m; r)$.
- $(M^{\mathsf{tran}}_{\mathsf{PKE}}\ f_{enc_c((ek^l_c(),\cdot)})(m;r)$ returns $(M_{\mathsf{PKE}}\ enc_c)(\langle ek, \tau^{\mathsf{ek}_x}_{\mathsf{PKE}}\rangle, m; r)$.

$(M^{\mathsf{tran}}_{\mathsf{PKE}}\ \mathsf{func})(b)$:
  if $b = \langle ek, dk', \tau^{\mathsf{dk}_x}_{\mathsf{PKE}}\rangle$ for some $(l,(ek,dk,dk')) \in \nu$ then
    return $f_{dk^l_x()}$
  else if $b = \langle ek, \tau^{\mathsf{ek}_x}_{\mathsf{PKE}}\rangle$ for some $(l,(ek,dk,dk')) \in \nu$ then
    return $f_{ek^l_x()}$
  else if $b \in [\![\tau^{\mathsf{ciphertext}}_{\mathsf{PKE}}]\!]$ then
    parse b as $\langle c, ek, \tau^{\mathsf{ciphertext}}_{\mathsf{PKE}}\rangle$
    if there is $(l,(ek,dk,r')) \in \nu$ for some l, dk then
      let $m := \mathsf{PKE.Dec}(dk, c)$
        if $m \neq \bot$ then
          if l belongs to an honest key then
            return $f_{enc_h(ek^l_h(),\cdot)}$
          else
            return $f_{enc_c((ek^l_c(),\cdot}$
  return $\bot$

For b with $(M^{\mathsf{tran}}_{\mathsf{PKE}}\ \mathsf{func})(b) = f_{enc_h(ek^l_h(),\cdot)}$ we have $(l,(ek,dk,dk')) \in \nu$ with $\mathsf{PKE.Dec}(dk,c) =: m \neq \bot$ for $b = \langle c, ek, \tau^{\mathsf{ciphertext}}_{\mathsf{PKE}}\rangle$ and define $(M^{\mathsf{tran}}_{\mathsf{PKE}}\ f_{enc_h(ek^l_h(),\cdot)}1)(b) := m$. Analogously for $(M^{\mathsf{tran}}_{\mathsf{PKE}}\ \mathsf{func})(b) = f_{enc_c((ek^l_c(),\cdot}$.

Convert terms. Adversary $\mathcal{A}$ uses the function symbols of the original symbolic model $\mathcal{M}_{\mathsf{PKE}}$. Hence $\mathcal{B}$ needs to map these symbols to the corresponding transparent functions. Towards this goal we introduce the function convert as follows (the first matching rule is applied):

- $\mathsf{convert}(f^l(t_1,\ldots,t_n)) = f^l(\mathsf{convert}(t_1),\ldots,\mathsf{convert}(t_n))$ for all $f \notin \Sigma_{\mathsf{PKE}}$.
- $\mathsf{convert}(ek_x(dk^l_x())) = f_{ek^l_x()}$
- $\mathsf{convert}(dk^l_x()) = f_{dk^l_x()}$
- $\mathsf{convert}(\hat{enc^l_h}(ek^l_h(),m)) = f^{\hat{l}(m)}_{enc_h(ek^l_h(),0^\ell)}()$ if $\hat{l} \in \mathsf{labelsH}$
- $\mathsf{convert}(\hat{enc^l_h}(ek^l_h(),m)) = f^{\hat{l}}_{enc_h(ek^l_h(),\cdot)}(\mathsf{convert}(m))$ if $\hat{l} \in \mathsf{labelsA}$
- $\mathsf{convert}(\hat{enc^l_c}(ek^l_c(),m)) = f^{\hat{l}}_{enc_c((ek^l_c(),\cdot}(\mathsf{convert}(m))$

For a list of terms T we define $\mathsf{convert}(T) := \{\mathsf{convert}(t) : t \in T\}$.

$\mathcal{B}$ simulates Game 2 for $\mathcal{A}$ while playing

$$\mathsf{DS}_{\mathcal{M}\cup(\mathcal{M}^{\mathsf{tran}}_{\mathsf{PKE}}(\nu)\cup\mathcal{M}_{\mathsf{tran}}(\nu')),\mathcal{I}(\cup\mathcal{I}^{\mathsf{tran}}_{\mathsf{PKE}}(\nu)\cup\mathcal{I}_{\mathsf{tran}}(\nu'))}(\eta)$$

Note that we can generically compose $\mathcal{M}^{\mathsf{tran}}_{\mathsf{PKE}}(\nu) \cup \mathcal{M}_{\mathsf{tran}}(\nu')$ to one parametrized transparent model $\mathcal{M}'_{\mathsf{tran}}(\nu||\nu')$ since ν and ν' must be good (analogously for the implementation). However, for the sake of clarity, we keep them apart to distinguish the transparent functions (and parameter) provided by $\mathcal{A}$ from the additional transparent functions introduced by $\mathcal{B}$. Next we describe how $\mathcal{B}$ deals with the queries received from $\mathcal{A}$.

init query. $\mathcal{B}$ receives a lists of terms T, H from $\mathcal{A}$. Initially, $\mathcal{B}$ sets $\nu := \emptyset$. For each occurrence of $dk^l_x() \in T$ $\mathcal{B}$ then picks a nonce $r \leftarrow \{0,1\}^\eta$ and generates a keypair $(ek,dk) := \mathsf{PKE.KeyGen}(1^\eta, r)$. If $x = h$, $\mathcal{B}$ picks $dk' \leftarrow \{0,1\}^{|dk|}$. Otherwise $\mathcal{B}$ sets $dk' = dk$. (dk' represents the decryption key in the library and should be a fresh random value for honest decryption keys.) $\mathcal{B}$ then adds $(l,(ek,dk,dk'))$ to ν. Finally, $\mathcal{B}$ sends $\nu'||\nu$ to its game and subsequently queries "$\mathsf{init}\ \mathsf{convert}(T), \mathsf{convert}(H)$". Afterwards, $\mathcal{B}$ queries "$\mathsf{sgenerate}\ dk^l_h()$" for each $dk^l_h() \in T$.

generate queries. For each request "$\mathsf{generate}\ t$" $\mathcal{B}$ adds "$\mathsf{generate}\ t$" to $\mathbb{T}$. Then $\mathcal{B}$ sends "$\mathsf{generate}\ \mathsf{convert}(t)$" to its game and relays the response to $\mathcal{A}$. For each subterm $t' \in st(t)$ that is an honest encryption, i.e., $t' = enc^{\hat{l}}_h(ek^l_h(), m)$ $\mathcal{B}$ additionally sends "$\mathsf{sgenerate}\ \mathsf{convert}(m)$" to its game. Analogously for "$\mathsf{sgenerate}\ t$".

parse queries. For each request "$\mathsf{parse}\ c$" $\mathcal{B}$ sends "$\mathsf{parse}\ c$" to its game and receives a term t. $\mathcal{B}$ sends $\mathsf{convert}^{-1}(t)$ to $\mathcal{A}$.

Claim: Game 2 and Game 3 are indistinguishable.

We show that $\mathcal{B}$, while playing the game

$$\mathsf{DS}_{\mathcal{M}\cup(\mathcal{M}^{\mathsf{tran}}_{\mathsf{PKE}}(\nu)\cup\mathcal{M}_{\mathsf{tran}}(\nu')),\mathcal{I}(\cup\mathcal{I}^{\mathsf{tran}}_{\mathsf{PKE}}(\nu)\cup\mathcal{I}_{\mathsf{tran}}(\nu'))}(\eta),$$

perfectly simulates Game 2 for $\mathcal{A}$. First, we show that any valid trace produced by $\mathcal{A}$ in Game 2 leads to a valid trace by $\mathcal{B}$ (we say that these traces *correspond*). Then we show that every pair of valid corresponding traces leads to the same output for $\mathcal{A}$ by proving that a suitable invariant holds for the relation between the libraries in both settings.

Valid $\mathcal{A}$ leads to valid $\mathcal{B}$. First, we observe that any trace $\mathbb{T}_{\mathcal{A}}$ produced by $\mathcal{A}$ in Game 2 that is valid leads to a valid trace $\mathbb{T}_{\mathcal{B}}$ produced by $\mathcal{B}$: The application of $\mathsf{convert}$ to terms leads to variations the sense of requirement i for valid predicates. The additional $\mathsf{sgenerate}$ queries by $\mathcal{B}$ are valid by requirement ii. Furthermore, if a term t meets the requirement for the $\mathsf{generate}$ function in Game 2, $\mathsf{convert}(t)$ meets the requirements in the game $\mathcal{B}$ is playing.

Invariant. We still have to show that the output of the simulation matches the output of Game 2. First we observe that there is a bijection between the random coins used in Game 2 and the simulation. $\mathcal{B}$ uses the same amount of randomness to generate the keypairs as Game 2 does. All further uses of random coins coincide. Therefore, we can w.l.o.g. assume that the same random coins are used in Game 2 and the simulation. We show that the following invariants holds for all valid traces produced by $\mathcal{A}$ and the corresponding trace produced by $\mathcal{B}$:

1. $\mathrm{dom}(L_{ext}) = \mathrm{dom}(L_{small})$
2. $\forall c \in \mathrm{dom}(L_{small}) : \mathsf{convert}^{-1}(L_{small}[[c]]) = L_{ext}[[c]]$

where L_{ext} is the library in Game 2 and L_{small} the library in the game $\mathcal{B}$ is playing (which we call the *small game* from now on).

Initially, we have $L_{ext} = L_{small} = \emptyset$ and the invariant holds obviously.

init T, H. According to requirements 2, 3 for public key encryption we can distinguish the following three types of terms $t \in T \cup H$.

- $t = ek_h(dk_h^l())$, i.e., $\mathsf{convert}(t) = f_{ek_h^l()}$.
- $t = dk_c^l()$, i.e., $\mathsf{convert}(t) = f_{dk_c^l()}$.
- t doesn't contain function symbols from Σ_{PKE} and $\mathsf{convert}(t) = t$:

We observe that each initial term t that is a key generation in the extended game, $\mathcal{B}$ adds the corresponding converted term $\mathsf{convert}(t)$ to its init request. This corresponds to the key generation done in the extended game while the keys in the small game are hard coded in the transparent functions. After this step we have $\mathrm{dom}(L_{ext} \setminus \{(c, dk_h^l()) : c \in [\![\tau_{\mathsf{PKE}}^{\mathsf{dk}_h}]\!], l \in \mathsf{labelsH}\}) = \mathrm{dom}(L_{small})$ since the generated keys coincide but we do not add the honest decryption keys to the library in the simulation. This is gap is closed by the additional silent generate queries by $\mathcal{B}$. Then the invariants hold.

generate t. Assume that our invariants 1 and 2 hold for libraries L_{ext} and L_{small}. Then, they still hold after a valid query "$\mathsf{generate}\ t$" by $\mathcal{A}$ has been processed. In the extended game we have $(c_{ext}, L'_{ext}) := \mathsf{generate}_{M_{ext},\mathcal{R}}(t, L_{ext})$. In the simulation, $\mathcal{B}$ sends "$\mathsf{generate}\ \mathsf{convert}(t)$" to the small game and we have $(c_{small}, L'_{small}) := \mathsf{generate}_{M_{small},\mathcal{R}}(\mathsf{convert}(t), L_{small})$ (and maybe additional calls to $\mathsf{generate}_{M_{small},\mathcal{R}}$ if t contains honestly generated encryptions using honest keys). We observe the following (where M_{ext} and M_{small} are the Turing Machines in the extended and in the small game respectively):

- By requirement 2, M_{ext} is never called for ek_x, dk_x. Analogously for M_{small} and the transparent translations of keys.
- For an honest encryption subterm $t = enc_h^{\hat{l}}(ek_h^l(), m)$, we have a call $(c_{ext}, L'_{ext}) := \mathsf{generate}_{M_{ext},\mathcal{R}}(t, L_{ext})$ in the extended game and calls $(c_{small}, L'_{small}) := \mathsf{generate}_{M_{small},\mathcal{R}}(f^{\hat{l}}_{enc_h(ek_h^l(),0^\ell)}(m)(ek_h^l()), L_{small})$ and $(c'_{small}, L''_{small}) := \mathsf{generate}_{M_{small},\mathcal{R}}(m, L'_{small})$ in the small game (since $\mathcal{B}$ sends an additional query "$\mathsf{sgenerate}\ m$" to the small game). It is easy to see that the newly generated bitstrings coincide.
- For all other terms $\mathsf{convert}(t)$ only removes keys from t which are already part of the library. The rest of the term remains intact and hence the newly generated bitstrings coincide.

Our invariant hold for L'_{ext} and L'_{small}. Note that this implies $c_{ext} = c_{small}$ which is the response sent to $\mathcal{A}$ in both settings. Analogously for queries "$\mathsf{sgenerate}\ t$".

parse c. Assume that our invariants 1 and 2 hold for libraries L_{ext} and L_{small}. Then, they still hold after a valid query "$\mathsf{parse}\ c$" by $\mathcal{A}$ has been processed. If $c \in \mathrm{dom}(L_{ext})$, nothing changes and due to invariant 2 the response to $\mathcal{A}$ is the same in Game 2 and Game 3. Otherwise, since the implementations in both games are type-safe (Definition 4) we can focus our analysis on opening bitstrings from $[\![\mathcal{T}_{\mathsf{PKE}}]\!]$. For those it is easy to check that the open function for opens them structurally the same way (modulo conversion) the function $\mathsf{open}_{\mathsf{PKE}}$ does.

Claim: If $\mathcal{A}$ wins, then $\mathcal{B}$ wins Game 3 .

Due to the invariants 1 and 2 from above, we know that whenever a bitstring c sent by $\mathcal{A}$ is parsed as a term t in Game 3, it is parsed as $\mathsf{convert}(t)$ in the small game. By checking the deduction systems of both games we see that if t is non-DY in Game 3, then $\mathsf{convert}(t)$ is non-DY in the small game. Since $\mathcal{I}$ is a deduction sound implementation of $\mathcal{M}$ $\mathcal{A}$ can win Game 3 only with negligible probability which concludes our proof. □

8.3 Signatures

In this section we show that any deduction sound implementation can be extended by a signature scheme. More precisely, composition works if we require a strong EUF-CMA secure signature scheme and enforce static corruption. The result is again a deduction sound implementation.

8.3.1 Symbolic model

We first define the symbolic model $(\mathcal{T}_{\mathsf{SIG}}, \leq_{\mathsf{SIG}}, \Sigma_{\mathsf{SIG}}, \mathcal{D}_{\mathsf{SIG}})$ for signatures. The signature Σ_{SIG} features the following function symbols:

$$sk : \tau_{\mathsf{SIG}}^{\mathsf{sk}}$$
$$vk : \tau_{\mathsf{SIG}}^{\mathsf{sk}} \to \tau_{\mathsf{SIG}}^{\mathsf{vk}}$$
$$sig : \tau_{\mathsf{SIG}}^{\mathsf{sk}} \times \top \to \tau_{\mathsf{SIG}}^{\mathsf{sig}}$$

The randomized function sk of arity $\tau_{\mathsf{SIG}}^{\mathsf{sk}}$ returns a signing key. The deterministic function vk of arity $\tau_{\mathsf{SIG}}^{\mathsf{sk}} \to \tau_{\mathsf{SIG}}^{\mathsf{vk}}$ derives a verification key from a signing key. The randomized signing function sig has arity $\tau_{\mathsf{SIG}}^{\mathsf{sk}} \times \top \to \tau_{\mathsf{SIG}}^{\mathsf{sig}}$ and, given a signing key and a message of type $\top$, represents a signature of that message. To complete the formal definition we set the types

$$\mathcal{T}_{\mathsf{SIG}} := \{\top, \tau_{\mathsf{SIG}}^{\mathsf{sk}}, \tau_{\mathsf{SIG}}^{\mathsf{vk}}, \tau_{\mathsf{SIG}}^{\mathsf{sig}}\}$$

All introduced types are direct subtypes of the base type $\top$ (this defines $\leq_{\mathsf{SIG}}$). The deduction system captures the security of signatures

$$\mathcal{D}_{\mathsf{SIG}} := \left\{ \begin{array}{cc} & \frac{sk^l()}{vk(sk^l())}, \\ \frac{sig^{\hat{l}}(sk^l(),m)}{m}, & \frac{sk^l() \quad m}{sig^{l_a}(sk^l(),m)} \end{array} \right\}$$

These rules are valid for arbitrary labels $l, \hat{l} \in \mathsf{labels}$ and adversarial labels $l_a \in \mathsf{labelsA}$. Read from top left to bottom right the following intuitions back up the rules:

- The adversary can derive verification keys from signing keys.
- Signatures reveal the message that was signed.
- The adversary can use known signing keys to deduce signatures under those keys.

Although the verification key is going to be part of the computational implementation of a signatures, we don't need a rule $\frac{sig^{\hat{l}}(sk^l(),m)}{vk(sk^l)}$ since we enforce static corruption where adversary knows all verification keys.

8.3.2 Implementation

We now give a concrete implementation $\mathcal{I}_{\mathsf{SIG}}$ for signatures. The implementation uses some strong EUF-CMA secure signature scheme $(\mathsf{SIG.KeyGen}, \mathsf{SIG.Sig}, \mathsf{SIG.Vfy})$. As usual, here $\mathsf{SIG.KeyGen}$ is a generation algorithm for key pairs, $\mathsf{SIG.Sig}$ is an signing algorithm and $\mathsf{SIG.Vfy}$ is a verification algorithm. Note that $\mathsf{SIG.Sig}$ is an algorithm that

$\mathsf{open}_{\mathsf{SIG}}(c, L)$
  if $c \in [\![\mathcal{T}_{\mathsf{SIG}}]\!] \cap \mathrm{dom}(L)$ then
    return $(c, L(c))$
  else if $c = \langle sk, \tau^{\mathsf{sk}}_{\mathsf{SIG}} \rangle$ then
    return $(c, g^{l(c)}_{\tau^{\mathsf{sk}}_{\mathsf{SIG}}})$
  else if $c = \langle vk, \tau^{\mathsf{vk}}_{\mathsf{SIG}} \rangle$ then
    if $\hat{sk} \in \mathrm{dom}(L)$ s.t. $\hat{sk} = \langle vk, sk, \tau^{\mathsf{sk}}_{\mathsf{SIG}} \rangle$ then
      return $(c, vk(\hat{sk}))$
    else
      return $(c, g^{l(c)}_{\tau^{\mathsf{vk}}_{\mathsf{SIG}}})$
  else if $c = \langle \sigma, m, vk, \tau^{\mathsf{sig}}_{\mathsf{SIG}} \rangle$ then
    if $(\langle vk, \tau^{\mathsf{vk}}_{\mathsf{SIG}} \rangle, vk(\hat{sk})) \in L$
      and $\mathsf{SIG.Vfy}(vk, \sigma, m) = \mathsf{true}$ then
      return $(c, sig^{l(c)}(\hat{sk}, m))$
    else
      return $(c, g^{l(c)}_{\tau^{\mathsf{sig}}_{\mathsf{SIG}}})$
  else
    return $(c, g^{l(c)}_{\top})$

Figure 10: Open function for signatures.

takes three inputs: the signing key, the message to be signed and the randomness that is used for signing.[3] The computable interpretations of sk, vk, sig are as follows:

- $(M_{\mathsf{SIG}}\ sk)(r)$: Let $(vk, sk) := \mathsf{SIG.KeyGen}(1^\eta, r)$. Return $\langle vk, sk, \tau^{\mathsf{sk}}_{\mathsf{SIG}} \rangle$.
- $(M_{\mathsf{SIG}}\ vk)(\hat{sk})$: Parse $\hat{sk}$ as $\langle vk, sk, \tau^{\mathsf{sk}}_{\mathsf{SIG}} \rangle$. Return $\langle vk, \tau^{\mathsf{vk}}_{\mathsf{SIG}} \rangle$.
- $(M_{\mathsf{SIG}}\ sig)(\hat{sk}, m; r)$: Parse $\hat{sk}$ as $\langle vk, sk, \tau^{\mathsf{sk}}_{\mathsf{SIG}} \rangle$. Let $\sigma := \mathsf{SIG.Sig}(sk, m, r)$ and return $\langle \sigma, m, vk, \tau^{\mathsf{sig}}_{\mathsf{SIG}} \rangle$.

The $\mathsf{valid}_{\mathsf{SIG}}$ *predicate.*

Intuitively, we require static corruption of signing keys and that verification and signing keys are only used for signing and verification. Formally, based on the current trace $\mathbb{T}$ of all parse and generate requests of the adversary, the predicate $\mathsf{valid}_{\mathsf{SIG}}$ returns true only if the following conditions hold:

1. The trace starts with a query "$\mathsf{init}\ T, H$" (T, H may be the empty list respectively). There are no further init queries.
2. The adversary may only generate keys in the init query. Concretely, this is guaranteed by the following rules:
   (a) For the query "$\mathsf{init}\ T, H$", the function symbols sk and vk may only occur in a term $t \in T$ (i.e., not as subterms of other terms) of one of the two following types (for $l \in \mathsf{labelsH}$):
   - $t = vk(sk^l())$ (to generate an honest signing key)
   - $t = sk^l()$ (to generate a corrupted signing key)

   Any label l for $sk^l()$ must be unique in T.
   (b) Any occurrence of $vk(sk^l())$ or $sk^l()$ in a $\mathsf{generate}$ query must have occurred in the init query.
3. The adversary must not use the function symbol sig in the init query.
4. The term $sk^l()$ may only occur as the first argument of sig.

[3]Since the message m is of basetype, we require a scheme with message space $\{0,1\}^*$.

Checking the implementation.

We first observe that $\mathcal{I}_{\mathsf{SKE}}$ is collision-free (Definition 3): Basically, collisions for keys can only occur with negligible probability since they break the security of the scheme (which is strong EUF-CMA secure). Collisions of signatures can only occur with negligible probability as well due to the EUF-CMA security. Furthermore, it is easy to see that $\mathsf{open}_{\mathsf{SIG}}$ meets the requirements of Definition 4 and that $\mathsf{valid}_{\mathsf{SIG}}$ meets the requirements for valid functions.

THEOREM 3. *Let $\mathcal{M}$ be a symbolic model and $\mathcal{I}$ be deduction sound implementation of $\mathcal{M}$. If ($\mathcal{M}_{\mathsf{SIG}}$,$\mathcal{I}_{\mathsf{SIG}}$) and ($\mathcal{M}$,$\mathcal{I}$) are compatible (see requirements in Section 6), then $\mathcal{I} \cup \mathcal{I}_{\mathsf{SIG}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{SIG}}$ for any $\mathcal{I}_{\mathsf{SIG}}$ constructed from a strong EUF-CMA secure signature scheme.*

PROOF. First, we briefly describe the intuition behind the proof: Let $\mathcal{A}$ be an adversary playing the deduction soundness game. Assume that $\mathcal{A}$ queries "$\mathsf{parse}\ c$" and c is parsed as a non-DY term t that contains a signature $sig := sig^{\hat{l}}(sk^l(), m)$ and $S \not\vdash sig$ (where S is the list of terms generated for $\mathcal{A}$ in the deduction soundness game). We distinguish two possible ways the adversary could potentially have learned sig:

If sig was previously generated for $\mathcal{A}$ (i.e., $sig \in st(S)$ and $\hat{l} \in \mathsf{labelsH}$), we say that $\mathcal{A}$ *reconstructed* sig. Since signatures and transparent functions do not introduce function symbols that allow for signatures as input such that the signature is not derivable from the constructed term, $\mathcal{A}$ must have broken the deduction soundness of $\mathcal{I}$ in this case. Hence, using $\mathcal{A}$, we can construct an successful adversary $\mathcal{B}$ on the deduction soundness of $\mathcal{I}$. $\mathcal{B}$ simulates signatures using transparent functions.

If sig was not previously generated for $\mathcal{A}$ (i.e., $\hat{l} \in \mathsf{labelsA}$), we say that $\mathcal{A}$ *forged* sig. In this case $\mathcal{A}$ can be used to break the strong EUF-CMA security of the signature scheme.

Since reconstructions and forgeries can only occur with negligible probability the composed implementation $\mathcal{I} \cup \mathcal{I}_{\mathsf{SIG}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{SIG}}$.

Game 0.

In Game 0 $\mathcal{A}$ plays the original deduction soundness game $\mathsf{DS}_{(\mathcal{M} \cup \mathcal{M}_{\mathsf{SIG}}) \cup \mathcal{M}_{\mathsf{tran}}(\nu), (\mathcal{I} \cup \mathcal{I}_{\mathsf{SIG}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu)}(\eta)$.

Game 1.

In Game 1 we replace the $\mathsf{generate}$ function by the collision-aware generate function from Figure 3. Since $(\mathcal{I} \cup \mathcal{I}_{\mathsf{SIG}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu)$ is a collision-free implementation Game 0 and Game 1 are indistinguishable by Lemma 2.

Game 2.

Game 2 is Game 1 with a changed winning condition for the adversary. First we introduce the set of *reconstruction*

rules as follows: $\mathcal{D}_r := \{\frac{t}{sig^{\hat{h}}(sk^h(),m)} : t$ is a term from $(\mathcal{M} \cup \mathcal{M}_{\mathsf{SIG}}) \cup \mathcal{M}_{\mathsf{tran}}(\nu)$, $\hat{h}, h \in \mathsf{labelsH}$ and $sig^{\hat{h}}(sk^h(), m)$ is a subterm of $t\}$.

Let $\vdash_1$ denote the deduction relation of the previous game based on the rules from $\mathcal{D} \cup \mathcal{D}_{\mathsf{SIG}} \cup \mathcal{D}_{\mathsf{tran}}$. In this game we use the deduction relation $\vdash_2$ based on the rules from $\mathcal{D} \cup \mathcal{D}_{\mathsf{SIG}} \cup \mathcal{D}_{\mathsf{tran}} \cup \mathcal{D}_r$, i.e., the adversary may now use any honestly generated signature to deduce new terms. In other words, the adversary cannot win (produce a non-DY term) any longer by using a signature that has been generated for it.

Claim: Game 1 and Game 2 are indistinguishable.

The only difference between Game 1 and Game 2 is the handling of parse requests. Since $\vdash_2$ potentially allows to deduce more terms from a given set of terms S than $\vdash_1$, the adversary might produce a non-DY term in Game 1 that is DY in Game 2. We now show that the adversary breaks the deduction soundness of $\mathcal{I}$ in that case. This part of the proof is very similar to the proof of indistinguishability between Games 2 and 3 in Theorem 2.

The simulator. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ on the deduction soundness of $\mathcal{I}$. Towards this goal $\mathcal{B}$, in addition to the transparent symbolic model $\mathcal{M}_{\mathsf{tran}}(\nu')$ used by $\mathcal{A}$, uses a transparent symbolic model to simulate signatures in the deduction soundness game for $\mathcal{I}$.

Transparent symbolic model for signatures. We first describe the parametrized transparent symbolic model $\mathcal{M}^{\mathsf{tran}}_{\mathsf{SIG}}(\nu)$ and the corresponding parametrized implementation $\mathcal{I}^{\mathsf{tran}}_{\mathsf{SIG}}(\nu)$ $\mathcal{B}$ will use to simulate $\mathcal{I}_{\mathsf{SIG}}$. We use the data types and subtype relation from $\mathcal{M}_{\mathsf{SIG}}$. ν is expected to be an encoding of a list of label-triple pairs $(l, (ek, dk, sk'))$ ($l \in \mathsf{labels}$) where the triple consist of a keypair vk, sk and an additional value sk' (used to represent honest signing keys in the library). The signature $\Sigma^{\mathsf{tran}}_{\mathsf{SIG}}$ is the following:

- deterministic f^h_{sk} with $\mathrm{ar}(f^h_{sk}) = \tau^{\mathsf{sk}}_{\mathsf{SIG}}$ for all labels $l \in \nu$
- deterministic f^h_{vk} with $\mathrm{ar}(f^h_{vk}) = \tau^{\mathsf{vk}}_{\mathsf{SIG}}$ for all labels $l \in \nu$
- randomized $f_{sig(sk^h(),\cdot)}$ with $\mathrm{ar}(f_{sig(sk^h(),\cdot)}) = \top \to \tau^{\mathsf{sig}}_{\mathsf{SIG}}$ for all labels $l \in \nu$

We specify a parametrized implementation $\mathcal{I}^{\mathsf{tran}}_{\mathsf{SIG}}(\nu)$ for $\mathcal{M}^{\mathsf{tran}}_{\mathsf{SIG}}$ as follows for each $(l, (vk, sk, sk')) \in \nu$:

- $(M^{\mathsf{tran}}_{\mathsf{SIG}}\ f^h_{sk})()$ returns $\langle vk, sk', \tau^{\mathsf{sk}}_{\mathsf{SIG}}\rangle$
- $(M^{\mathsf{tran}}_{\mathsf{SIG}}\ f^h_{vk})()$ returns $\langle vk, \tau^{\mathsf{vk}}_{\mathsf{SIG}}\rangle$
- $(M^{\mathsf{tran}}_{\mathsf{SIG}}\ f_{sig(sk^h(),\cdot)})(m; r)$ returns $(M_{\mathsf{SIG}}\ sig)(\langle sk, \tau^{\mathsf{sk}}_{\mathsf{SIG}}\rangle, m; r)$

Furthermore, we have to define the transparent modes of operation for $M^{\mathsf{tran}}_{\mathsf{SIG}}$. $(M^{\mathsf{tran}}_{\mathsf{SIG}}\mathsf{proj}\ f_{sig(sk^h(),\cdot)}\ 1)(sig)$ parses sig as $\langle \sigma, m, vk, \tau^{\mathsf{sig}}_{\mathsf{SIG}}\rangle$ and returns m if parsing succeeds and $\perp$ otherwise. $(M^{\mathsf{tran}}_{\mathsf{SIG}}\ \mathsf{func})(b)$ returns f^h_{sk} if for some $(l, (vk, sk, sk')) \in \nu$ b can be parsed as $\langle vk, sk', \tau^{\mathsf{sk}}_{\mathsf{SIG}}\rangle$ and f^h_{vk} if b can be parsed as $\langle vk, \tau^{\mathsf{vk}}_{\mathsf{SIG}}\rangle$. If $b \in [\![\tau^{\mathsf{sig}}_{\mathsf{SIG}}]\!]$, $(M^{\mathsf{tran}}_{\mathsf{SIG}}\ \mathsf{func})$ tries to parse b as $\langle \sigma, m, vk, \tau^{\mathsf{sig}}_{\mathsf{SIG}}\rangle$. If parsing succeeds, $\mathsf{SIG.Vfy}(vk, \sigma, m) = \mathsf{true}$ and there is a $(l, (vk, sk, sk')) \in \nu$, then $(M^{\mathsf{tran}}_{\mathsf{SIG}}\ \mathsf{func})$ returns $f_{sig(sk^h(),\cdot)}$, $\perp$ otherwise.

Convert terms. Adversary $\mathcal{A}$ uses the function symbols of the original symbolic model $\mathcal{M}_{\mathsf{SIG}}$. Hence $\mathcal{B}$ needs to map these symbols to the corresponding transparent functions. Towards this goal we introduce the function convert as follows (the first matching rule is applied):

- $\mathsf{convert}(f^l(t_1, \ldots, t_n)) = f^l(\mathsf{convert}(t_1), \ldots, \mathsf{convert}(t_n))$ for all $f \notin \Sigma_{\mathsf{SIG}}$
- $\mathsf{convert}(sk^h()) = f^h_{sk}()$
- $\mathsf{convert}(vk(sk^h())) = f^h_{vk}()$
- $\mathsf{convert}(sig^{\hat{h}}(sk^h(), m)) = f^{\hat{h}}_{sig(sk^h(),\cdot)}(\mathsf{convert}(m))$

For a list of terms T we define $\mathsf{convert}(T) := \{\mathsf{convert}(t) : t \in T\}$.

$\mathcal{B}$ simulates Game 1 for $\mathcal{A}$ while playing

$$\mathsf{DS}_{\mathcal{M} \cup (\mathcal{M}^{\mathsf{tran}}_{\mathsf{SIG}}(\nu) \cup \mathcal{M}_{\mathsf{tran}}(\nu')), \mathcal{I}(\cup \mathcal{I}^{\mathsf{tran}}_{\mathsf{SIG}}(\nu) \cup \mathcal{I}_{\mathsf{tran}}(\nu'))}(\eta)$$

Note that we can generically compose $\mathcal{M}^{\mathsf{tran}}_{\mathsf{SIG}}(\nu) \cup \mathcal{M}_{\mathsf{tran}}(\nu')$ to one parametrized transparent model $\mathcal{M}'_{\mathsf{tran}}(\nu||\nu')$ since ν and ν' must be good (analogously for the implementation). However, for the sake of clarity, we keep them apart to distinguish the transparent functions (and parameter) provided by $\mathcal{A}$ from the additional transparent functions introduced by $\mathcal{B}$. Next we describe how $\mathcal{B}$ deals with the queries received from $\mathcal{A}$.

init query. $\mathcal{B}$ receives a lists of terms T, H from $\mathcal{A}$. Initially, $\mathcal{B}$ sets $\nu := \emptyset$. For each occurrence of $sk^l() \in T$ $\mathcal{B}$ then picks a nonce $r \leftarrow \{0,1\}^\eta$ and generates a keypair $(vk, sk) := \mathsf{SIG.KeyGen}(1^\eta, r)$. $\mathcal{B}$ sets $sk' = sk$. (dk' represents the signing key in the library and will be a fresh random value for honest signing keys in a later simulation.) $\mathcal{B}$ then adds $(l, (vk, sk, sk'))$ to ν. Finally, $\mathcal{B}$ sends $\nu'||\nu$ to its game and subsequently queries "init $\mathsf{convert}(T), \mathsf{convert}(H)$". Afterwards, $\mathcal{B}$ queries "sgenerate $sk^l()$" for each $sk^l() \in T$.

The other queries are handled exactly as in the simulation using transparent functions in Theorem 2. Likewise, the proof that the simulation - apart from the winning condition - perfectly simulates Game 1 (and thus Game 2) is analogous to the corresponding proof in Theorem 2.

The changed winning condition. Let us now assume that $\mathcal{A}$ sends a "parse c" such that c is parsed as a non-DY term t in Game 1 while t is DY in Game 2. Concretely, we have $S \not\vdash_1 t$ and $S \vdash_2 t$ where S is the set of the terms previously generated for $\mathcal{A}$. We have to show that $\mathsf{convert}(S) \not\vdash_{sim} \mathsf{convert}(t)$ where $\mathsf{convert}(S) := \{\mathsf{convert}(t) : t \in S\}$, i.e., that $\mathcal{A}$ breaks the small library in this case. Since the simulation is perfect, c is actually parsed as $\mathsf{convert}(t)$ in the simulation.

From $\vdash_2$ to $\vdash_1$. By Lemma 4 there is an S' with $S \vdash_2 S'$ such that $S' \not\vdash_1 t$ and $S' \cup \{sig^{\hat{h}}(sk^h, m)\} \vdash_1 t$ where $sig^{\hat{h}}(sk^h, m) \in st(S')$ (i.e., derivable by a rule from $\mathcal{D}_r$).

From $\vdash_{sim}$ to $\vdash_1$. In this paragraph we show that if $\mathsf{convert}(t)$ was deducible in the simulation, t would be deducible in Game 1. Therefor we assume $\mathsf{convert}(S') \vdash_{sim} \mathsf{convert}(t)$ towards contradiction. Concretely, we show that it implies $S' \vdash_1 t$ contradicting $S' \not\vdash_1 t$ which we have due to the previous paragraph.

Let π be a proof for $\mathsf{convert}(S') \vdash_{sim} \mathsf{convert}(t)$. Then there is a proof π' for $\mathsf{convert}(S') \vdash_{sim} \mathsf{convert}(t)$ such that for every $\alpha_i = \frac{m}{f^l_{sig(sk^h(),\cdot)}(m)}$ we have $f^l_{sig(sk^h(),\cdot)}(m) \in st(\mathsf{convert}(t))$.

Assume we had an $\alpha_i = \frac{m}{f^l_{sig(sk^h(),\cdot)}(m)}$ in π such that $f^l_{sig(sk^h(),\cdot)}(m) \notin st(\mathsf{convert}(t))$. Furthermore, and w.l.o.g., we assume that $f^l_{sig(sk^h(),\cdot)}(m) \notin st(\mathsf{convert}(S'))$ (a justification for this follows in the next paragraph). We define the substitution θ on terms as $\theta(f^l(t_1, \ldots, t_n)) = f^l(\theta(t_1), \ldots, \theta(t_n))$

for all function symbols $f! = f_{sig(sk^h(),\cdot)}$ and $\theta(f^l_{sig(sk^h(),\cdot)}(m)) = m$ (we could pick any term from S_{i-1} here). $\mathsf{convert}(S') =: S_0 \overset{\alpha_1}{\to} \cdots \overset{\alpha_{i-1}}{\to} S_{i-1} \overset{\theta\alpha_{i+1}}{\to} \theta(S_{i+1}) \cdots \overset{\theta\alpha_n}{\to} \theta(S_n)$ is a new proof $\widetilde{\pi}$ for $\mathsf{convert}(S') \vdash_{sim} \mathsf{convert}(t)$ since

- $S_n \ni \mathsf{convert}(t) \to \theta(S_n) \ni \theta(\mathsf{convert}(t)) = \mathsf{convert}(t)$ since $f^l_{sig(sk^h(),\cdot)}(m) \notin st(\mathsf{convert}(t))$
- $\theta(S_j) = S_j$ and $\theta(\alpha_j) = \alpha_j$ for $j \in \{1, \ldots, i-1\}$ since $f^l_{sig(sk^h(),\cdot)}(m) \notin st(\mathsf{convert}(S'))$
- $\theta(\alpha_j)$ is still an instantiation of the same rule as α_j for $j \in \{i+1, \ldots, n\}$ since the only available rule that uses the structure of $f^l_{sig(sk^h(),\cdot)}(m)$ is $\frac{f^l_{sig(sk^h(),\cdot)}(x)}{x}$. However, none of the α_j can be an instantiation of this rule since $m \in S_{j-1}$ for $j \in \{i+1, \ldots, n\}$ (and we are only considering proofs where already known terms must not be derived again).

Why can we assume $f^l_{sig(sk^h(),\cdot)}(m) \notin st(\mathsf{convert}(S'))$ w.l.o.g.? $f^l_{sig(sk^h(),\cdot)}(m) \in st(\mathsf{convert}(S'))$ implies $sig^l(sk^h(), m) \in st(S')$. Furthermore, the setting in Game 1 satisfies the requirements for Lemma 3 and $t' = sig^l(sk^h(), m)$: Instantiations of rules from $\mathcal{D} \cup \mathcal{D}_{\mathsf{tran}}$ use signature terms in a black-box way and hence satisfy property (i). Instantiations of rules from $\mathcal{D}_{\mathsf{SIG}}$ satisfy either (i) or (ii). Thus, by Lemma 3 we have $S' \vdash_1 sig^l(sk^h(), m)$. Converting this proof leads to a proof $\mathsf{convert}(S') \vdash_{sim} f^l_{sig(sk^h(),\cdot)}(m)$ that does not use a rule of the type $\frac{x}{f^l_{sig(sk^h(),\cdot)}(x)}$. Hence we can always find a proof where instantiations $\frac{m}{f^l_{sig(sk^h(),\cdot)}(m)}$ are only used for $f^l_{sig(sk^h(),\cdot)}(m) \notin st(\mathsf{convert}(S'))$.

In conclusion, if we find a proof π for $\mathsf{convert}(S') \vdash_{sim} \mathsf{convert}(t)$, then we find a proof π' that does not contain any α_i of type $\frac{m}{f^l_{sig(sk^h(),\cdot)}(m)}$. We then apply $\mathsf{convert}^{-1}$ to this proof and get a proof for $S' \vdash_1 t$ which is a contradiction to our requirements for S'. Hence we cannot have $\mathsf{convert}(S') \vdash_{sim} \mathsf{convert}(t)$ and thus "$\mathsf{parse}\ c$" lets the simulator win the deduction soundness game for $\mathcal{I}$ where signatures are replaced by transparent functions. This proves our claim that Game 1 and Game 2 are indistinguishable.

Game 3.

We define the set $\mathcal{D}_f := \{\frac{vk(x)}{x}\}$. Let $\vdash_2$ denote the deduction relation of the previous game based on the rules from $\mathcal{D} \cup \mathcal{D}_{\mathsf{SIG}} \cup \mathcal{D}_{\mathsf{tran}} \cup \mathcal{D}_r$. In this game we use the deduction relation $\vdash_3$ based on the rules from $\mathcal{D} \cup \mathcal{D}_{\mathsf{SIG}} \cup \mathcal{D}_{\mathsf{tran}} \cup \mathcal{D}_r \cup \mathcal{D}_f$, i.e., the adversary is allowed to derive the signing key for any verification key in use. This also means that the adversary can no longer win by producing a forgery. Furthermore we replace the honest signing keys in the library by random bistrings analogously to Game 2 in Theorem 2.

Claim: Game 2 and Game 3 are indistinguishable..

Let $\mathcal{A}$ be an adversary for Game 2. To prove our claim we show how an adversary $\mathcal{B}$ for the strong EUF-CMA security game can be constructed using $\mathcal{A}$. Whenever $\mathcal{A}$ wins Game 2 by having a bitstring c parsed as a term t that is non-DY in Game 2 but is DY in Game 3, $\mathcal{B}$ will win its game. Since our signature scheme is strongly EUF-CMA secure, such a "distinguishing" c can only be produced by an adversary with negligible probability. We first describe the adversary $\mathcal{B}$ in detail.

init request. By requirement (1) $\mathcal{A}$ starts with a request "$\mathsf{init}\ T, H$". Furthermore, by requirement (2), we can distinguish three types of terms t in T. They are handled by the simulator as follows:

- $t = vk(sk^l())$ ($\mathcal{A}$ requests an honest signing key): $\mathcal{B}$ request a verification key vk with a corresponding signing oracle from the strong EUF-CMA game. Then, it picks a random value sk' and sets $\hat{sk} := \langle vk, sk', \tau^{\mathsf{sk}}_{\mathsf{SIG}}\rangle$. We refer to the signing oracle corresponding to vk with $\mathcal{O}^{sig}_{\hat{sk}}(\cdot)$. $\mathcal{B}$ sets $L := L \cup \{(\hat{sk}, sk^l()), (\langle vk, \tau^{\mathsf{vk}}_{\mathsf{SIG}}\rangle, vk(\hat{sk}))\}$ and adds $\hat{vk}$ to the list of bistrings that will be returned to $\mathcal{A}$.
- otherwise we have $t = sk^h()$ ($\mathcal{A}$ requests a corrupted signing key) or t does not contain function symbols from Σ_{SIG} (note that t must not contain signatures by 3). In this case $\mathcal{B}$ uses the normal $\mathsf{generate}$ function and computes $(c, L) := \mathsf{generate}(t, L)$. It then adds c to the list of bitstrings that will be returned to $\mathcal{A}$.

After the init request, $\mathcal{B}$ changes the $\mathsf{generate}$ function to use the oracles for signatures under honest signing keys. Concretely, it replaces the line

$$\text{let } c := (M\ f)(c_1, \ldots, c_n; r)$$

with

$$\begin{array}{l}\text{if } t = \hat{sig^h}(sk^h(), m) \text{ then}\\ \quad \text{let } c := \langle \mathcal{O}^{sig}_{c_1}(c_2), c_2, vk, \tau^{\mathsf{sig}}_{\mathsf{SIG}}\rangle\\ \text{else}\\ \quad \text{let } c := (M\ f)(c_1, \ldots, c_n; r)\end{array}$$

Note that the bitstrings c_1 and c_2 correspond to the signing key and the message to be signed respectively. Using the updated $\mathsf{generate}$ function, $\mathcal{B}$ simulates the rest of Game 2 according to the normal deduction soundness game from Figure 7. The simulation is indistinguishable:

- There is a bijection between the randomness used in an execution of Game 2 and the randomness used in the simulation: The randomness used for key generation and for generating signatures under honest keys is used by the strong EUF-CMA game in the simulation (and this is the only difference between Game 2 and the simulation as far as the use of randomness is concerned).
- The fact that the library contains randomized honest signing keys cannot be detected by the adversary for the same reason the randomized honest encryption keys cannot be detected by the adversary in Game 2 in Theorem 2: According to $\mathsf{valid}_{\mathsf{SIG}}$ (requirement (4)) signing keys may only occur as the first argument to sig. If the adversary parses a bitstring that can be used as a signing key, $\mathcal{B}$ wins the EUF-CMA game.

Extracting a forgery. Let "$\mathsf{parse}\ c$" be a request sent by $\mathcal{A}$ such that $t := L[[c]]$ and $S \not\vdash_2 t$ but $S \vdash_3 t$.

We claim that t contains either an honest signing key $sk^h()$ or a forgery under an honest signing key $sig^l(sk^h(), m) \notin st(S)$. To prove our claim we assume towards contradiction that t contains neither and let π be a proof for $S \vdash_3 t$. Then,

analogously to above, we can first remove all instantiations of the rule $\frac{sk^h() \quad x}{sig^l(sk^h(),x)}$ for honest signing keys $sk^h()$ from π yielding a proof π'. Next we remove all instantiations of the rule $\frac{vk(sk^h())}{sk^h()}$ from π' following the same principle and get a proof π''. However, π'' is a proof for $S \vdash_2 t$ which contradicts our initial assumption. Hence t contains either an honest signing key or a forgery.

Since $\mathcal{B}$ could parse the bitstring c, the library L contains a bitstring corresponding to every subterm of t. If t contains the term for an honest signing key, $\mathcal{A}$ must have guessed the randomized bitstring sk' for this key in the library which was never used to compute any bitstring sent to $\mathcal{A}$. This can only happen with negligible probability. Hence, c contains a forgery with overwhelming probability and $\mathcal{B}$ can use this forgery to win the strong EUF-CMA game it is playing (note that we need strong EUF-CMA security here since the forgery could be a re-randomization of a signature that was generated for the adversary and we wouldn't break EUF-CMA security in this case).

$\mathcal{A}$ cannot win Game 3 with non-negligible probability.

To conclude the proof we observe that an adversary $\mathcal{A}$ that wins Game 3 with non-negligible probability also wins the deduction soundness game for $\mathcal{I}$ with non-negligible probability: Analogously to the proof for the indistinguishability of Game 0 and Game 2 we can construct an adversary $\mathcal{B}$ that attacks the deduction soundness of $\mathcal{I}$ and simulates signatures using transparent functions. The simulation is perfect and if $\mathcal{A}$ wins, $\mathcal{B}$ wins since the deduction rules in Game 3 are effectively a superset of the deduction rules in the simulation.

Hence $\mathcal{A}$ cannot win Game 3 with non-negligible probability and $\mathcal{I} \cup \mathcal{I}_{\mathsf{SIG}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{SIG}}$. □

LEMMA 3. *Let $\mathcal{M} = (\mathcal{T}, \leq, \Sigma, \mathcal{D})$ be a symbolic model and $\mathcal{I}$ an implementation of $\mathcal{M}$. For the set of terms generated for the adversary S during the deduction soundness game $\mathsf{DS}_{\mathcal{M},\mathcal{I},\mathcal{A}}(\eta)$ holds, that for any term $t' \in st(S)$ with adversarial label we have $S \vdash t'$ if for all instantiations $\alpha = \frac{t_1 \quad \dots \quad t_n}{t}$ of rules from $\mathcal{D}$ with $t' \in st(t)$ meet at least one of the following properties:*

(i) $t' \in st(t_i)$ for some $i \in \{1, \dots, n\}$

(ii) for any $\widetilde{S} \xrightarrow{\alpha} \widetilde{S}'$ we have $\widetilde{S}' \vdash t'$

PROOF. Since generate does not introduce adversarial labels, every subterm with adversarial label of any term in S must have been introduced by a previous parse request. Let "parse c" be the first parse request that returns a term t such that t' is a subterm of t. Since the adversary didn't win with that request, t must be a DY term with respect to $S' \subseteq S$ where S' denote the terms generated for the adversary until that parse request. Concretely, we have $S' \vdash t$ and thus a proof $S' =: S_0 \xrightarrow{\alpha_1} S_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} S_n$ such that $t \in S_n$. Let $i \in \{1, \dots, n\}$ be the smallest index such that $t' \in st(S_i)$ ($i \neq 0$ since $t' \in st(S')$). α_i cannot meet property (i) since $t' \notin st(S_{i-1})$ (i is minimal). Hence we have $S_i \vdash t'$ by (ii) and $S' \vdash t'$ since $S' \vdash S_i$ and $S \vdash t'$ since $S' \subseteq S$. □

LEMMA 4. *Let $\mathcal{M} = (\mathcal{T}, \leq, \Sigma, \mathcal{D})$ be a symbolic model and let $\mathcal{D}' \supseteq \mathcal{D}'$ be a set of deduction rules for $\mathcal{M}$. $\vdash$ and $\vdash'$ denote the deduction relations corresponding to $\mathcal{D}$ and $\mathcal{D}'$ respectively. Let S be a set of terms and t be a term such that $S \nvdash t$ and $S \vdash' t$. Then there is a set of terms S' and a term t' such that*

- $S \vdash' S'$
- $S' \nvdash t$
- $S' \cup \{t'\} \vdash t$
- $S' \vdash' t'$

PROOF. $S \vdash' t$ implies that there is a deduction proof $S =: S_0 \xrightarrow{\alpha_1} S_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} S_n$. For $\alpha_i = \frac{u_1 \quad \dots \quad u_m}{u}$ and $S_{i-1} \xrightarrow{\alpha_i} S_i$ we require $u_i \in S_{i-1}$ and w.l.o.g. $u \notin S_{i-1}$.

Let $j \in \{1, \dots, n\}$ be the biggest index such that α_j is an instantiation of a rule from $\mathcal{D}' \setminus \mathcal{D}$ and $S_j \nvdash t$. (There must be such a rule since we would have $S \vdash t$ otherwise.) Then we set $S' := S_{j-1}$ and t' to be the one element in $S_j \setminus S_{j-1}$. We have

- $S \vdash' S'$ since $S \xrightarrow{\alpha_1} S_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{j-1}} S_{j-1} = S'$
- $S' \nvdash t$ by requirements for j
- $S' \cup \{t'\} \vdash t$: If there is an instantiation $\alpha_{j'}$ of a rule from $\mathcal{D}' \setminus \mathcal{D}$ with $j' > j$, we have $S_{j'-1} \vdash t$ by requirements for j. For the smallest such index j' we observe $S_j \vdash S_{j'-1}$ and hence $S_j = S' \cup \{t'\} \vdash t$.
- $S' \vdash' t'$ obviously by application of α_j

This concludes our proof. □

8.4 Secret key encryption

In this section we define a symbolic model $\mathcal{M}_{\mathsf{SKE}}$ for secret key encryption and a corresponding implementation $\mathcal{I}_{\mathsf{SKE}}$ based on a secret key encryption scheme (SKE.KeyGen, SKE.Enc, SKE.Dec). We show that composition of $\mathcal{M}_{\mathsf{SKE}}$ and $\mathcal{I}_{\mathsf{SKE}}$ with any symbolic model $\mathcal{M}$ comprising a deduction sound implementation $\mathcal{I}$ preserves this property for the resulting implementation, i.e., $\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{SKE}}$ if we use IND-CCA secure authenticated secret key encryption scheme.

8.4.1 Symbolic model

We first define the symbolic model $(\mathcal{T}_{\mathsf{SKE}}, \leq_{\mathsf{SKE}}, \Sigma_{\mathsf{SKE}}, \mathcal{D}_{\mathsf{SKE}})$ for secret key encryption. The signature Σ_{SKE} features the following function symbols

$$k_x : \tau^{\mathsf{k}_x}_{\mathsf{SKE}}$$
$$E_x : \tau^{\mathsf{k}_x}_{\mathsf{SKE}} \times \top \to \tau^{\mathsf{ciphertext}}_{\mathsf{SKE}}$$

for $x \in \{h, c\}$. The randomized functions k_h and k_c return honest or corrupted keys respectively. The randomized function E_x has arity $\tau^{\mathsf{k}_x}_{\mathsf{SKE}} \times \top \to \tau^{\mathsf{ciphertext}}_{\mathsf{SKE}}$ and represents a ciphertext under the given key. To complete the formal definition we set

$$\mathcal{T}_{\mathsf{SKE}} := \{\top, \tau^{\mathsf{k}_x}_{\mathsf{SKE}}, \tau^{\mathsf{ciphertext}}_{\mathsf{SKE}}\}$$

All introduced types are direct subtypes of the base type $\top$ (this defines $\leq_{\mathsf{SKE}}$). The deduction system captures the security of secret key encryption

$$\mathcal{D}_{\mathsf{SKE}} := \left\{ \begin{array}{cc} \multicolumn{2}{c}{\frac{k^l_x() \quad m}{E^{la}_x(k^l_x(),m)},} \\ \frac{E^{la}_h(k^l_h(),m)}{m}, & \frac{E^{\hat{l}a}_c(k^l_c(),m)}{m} \end{array} \right\}$$

$\mathsf{open}_{\mathsf{SKE}}(c, L)$
  if $c \in [\![\mathcal{T}_{\mathsf{SKE}}]\!] \cap \mathrm{dom}(L)$ then
    return $(c, L(c))$
  else if $c = \langle k, \tau^{\mathsf{k}_x}_{\mathsf{SKE}} \rangle$ then
    return $(c, g^{l(c)}_{\tau^{\mathsf{k}_x}_{\mathsf{SKE}}})$
  else if $c = \langle c', \tau^{\mathsf{ciphertext}}_{\mathsf{SKE}} \rangle$ then
    for each $(\hat{k}, k^h_x()) \in L$ do
      parse $\hat{k}$ as $\langle k, \tau^{\mathsf{k}_x}_{\mathsf{SKE}} \rangle$
      let $m := \mathsf{SKE.Dec}(k, c')$
      if $m \neq \bot$ then
        return $(c, E^{l(c)}_x(\hat{k}, m))$
    return $(c, g^{l(c)}_{\tau^{\mathsf{ciphertext}}_{\mathsf{SKE}}})$
  else
    return $(c, g^{l(c)}_{\top})$

Figure 11: Open function for secret key encryption.

These rules are valid for arbitrary labels $l, \hat{l} \in \mathsf{labels}$ and adversarial labels $l_a \in \mathsf{labelsA}$. Read from top left to bottom right the following intuitions back up the rules:

- The adversary can use any honestly generated key to encrypt some term u.
- The adversary knows the message contained in any adversarial encryption.
- The adversary knows the message contained in any encryption under a corrupted key.

8.4.2 Implementation

We now give a concrete implementation $\mathcal{I}_{\mathsf{SKE}}$ for secret key encryption. The implementation uses some IND-CCA secure authenticated secret key encryption scheme ($\mathsf{SKE.KeyGen}$, $\mathsf{SKE.Enc}$, $\mathsf{SKE.Dec}$). As usual, here $\mathsf{SKE.KeyGen}$ is a generation algorithm for key pairs, $\mathsf{SKE.Enc}$ is an encryption algorithm and $\mathsf{SKE.Dec}$ is a decryption algorithm. Note that $\mathsf{SKE.Enc}$ is an algorithm that takes three inputs: the key, the message to be encrypted and the randomness that is used for encryption.[4]

The computable interpretations of, k_x and E_x (for $x \in \{h, c\}$) are as follows:

- $(M_{\mathsf{SKE}}\ k_x)(r)$: Let $k := \mathsf{SKE.KeyGen}(1^\eta, r)$. Return $\langle k, \tau^{\mathsf{k}_x}_{\mathsf{SKE}} \rangle$
- $(M_{\mathsf{SKE}}\ E_x)(\hat{k}, m)(r)$: Parse $\hat{k}$ as $\langle k, \tau^{\mathsf{k}_x}_{\mathsf{SKE}} \rangle$. Let $c := \mathsf{SKE.Enc}(k, m, r)$ and return $\langle c, \tau^{\mathsf{ciphertext}}_{\mathsf{SKE}} \rangle$

The $\mathsf{valid}_{\mathsf{SKE}}$ *predicate.*

The predicate $\mathsf{valid}_{\mathsf{SKE}}$ guarantees, that all keys that may be used by the adversary later are generated during initialization (i.e., with the init query). We only allow static corruption of keys, i.e., the adversary has to decide which keys are honest and which are corrupted at this stage. Keys may only be used for encryption and decryption. This implicitly prevents key cycles. More formally, based on the current trace $\mathbb{T}$ of all parse and generate requests of the adversary, the predicate $\mathsf{valid}_{\mathsf{SKE}}$ returns true only if the following conditions hold:

1. The trace starts with a query "$\mathsf{init}\ T, H$" (T resp. H may be the empty list). There are no further init queries.
2. The adversary may only generate keys in the init query. Concretely, this is guaranteed by the following rules:
   (a) For the query "$\mathsf{init}\ T, H$", the function symbol k_c may only occur in a term $k^l_c() \in T$. Analogously, k_h may only occur in H. Any label l for $k^l_x()$ must be unique in $T \cup H$.
   (b) Any occurrence of $k^l_x()$ in a $\mathsf{generate}$ query must have occurred in the init query. $k^l_x()$ may only occur as the first argument to E_x.
3. The adversary must not use the function symbols for encryption E_x in the init query.

Checking the implementation.

We first observe that $\mathcal{I}_{\mathsf{SKE}}$ is collision-free (Definition 3): Basically, collisions for keys can only occur with negligible probability since they break the security of the scheme (which is IND-CCA secure). Collision of ciphertexts can only occur with negligible probability since we are using authenticated encryption. Furthermore, it is easy to see that $\mathsf{open}_{\mathsf{SKE}}$ meets the requirements of Definition 4 and that $\mathsf{valid}_{\mathsf{SKE}}$ meets the requirements for valid functions.

8.4.3 SKE *composability*

THEOREM 4. *Let $\mathcal{M}$ be a symbolic model and $\mathcal{I}$ a deduction sound implementation of $\mathcal{M}$. If ($\mathcal{M}_{\mathsf{SKE}}$,$\mathcal{I}_{\mathsf{SKE}}$) and ($\mathcal{M}$,$\mathcal{I}$) are compatible (see requirements in Section 6) and the SKE scheme* ($\mathsf{SKE.KeyGen}, \mathsf{SKE.Enc}, \mathsf{SKE.Dec}$) *is IND-CCA secure, then $\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{SKE}}$.*

PROOF. This proof is very similar to that for public key encryption (Theorem 2). The main difference is that the adversary cannot create ciphertexts under honest keys (by $\mathcal{D}_{\mathsf{SKE}}$). Therefore we include an additional game hop to where we add rules of the type $\frac{m}{E_h(k^l_h(), m)}$ to the deduction system. If an adversary notices the difference (i.e., it was able to produce non-DY terms without these rules), we can use it to break the authentication of ciphertexts. Hence this can only happen with negligible probability.

Game 0.

In Game 0 $\mathcal{A}$ plays the original deduction soundness game $\mathsf{DS}_{(\mathcal{M} \cup \mathcal{M}_{\mathsf{SKE}}) \cup \mathcal{M}_{\mathsf{tran}}(\nu), (\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu)}(\eta)$.

Game 1.

In Game 1 we replace the $\mathsf{generate}$ function by the collision-aware generate function from Figure 3. Since $(\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu)$ is a collision-free implementation Game 0 and Game 1 are indistinguishable by Lemma 2.

Game 2.

As in Game 2 from Theorem 2 we replace the ciphertexts created under honest keys by encryptions of 0 and the honest keys in the library by random bitstrings. The simulation

[4]Since the message m is of basetype, we require a scheme with message space $\{0, 1\}^*$.

that Game 1 and Game 2 are indistinguishable works analogously to Theorem 2.

Game 3.

In Game 3 we add rules $\frac{m}{E_h^{l_a}(k_h^l(),m)}$ for all honest keys $k_h^l()$ and labels $l_a \in \mathsf{labelsA}$ to the deduction system. This establishes a deduction system similar to that of public key encryption. We show that an adversary that can distinguish Game 2 from Game 3 can be used to break the authentication of the encryption scheme. Towards this goal we use the same technique as for the proof of indistinguishability of Games 2 and 3 in Theorem 3[5]. From $\mathcal{A}$ we construct an adversary $\mathcal{B}$ on playing the and simulating Game 2 for $\mathcal{A}$. If a bitstring c sent by $\mathcal{A}$ is parsed as a term t such that $S \not\vdash_2 t$ but $S \vdash_3 t$, we can (using the same arguments as in Theorem 3) extract a forgery from c.

Game 4.

In Game 4 $\mathcal{A}$ interacts with an adversary $\mathcal{B}$ that plays the deduction soundness game for $\mathcal{M}$ and $\mathcal{I}$ and intuitively simulates Game 3 for $\mathcal{A}$. Basically, $\mathcal{B}$ uses transparent functions to add symmetric key encryption to $\mathcal{M}$.

Transparent symbolic model for symmetric key encryption. We first describe the parametrized transparent symbolic model $\mathcal{M}_{\mathsf{SKE}}^{\mathsf{tran}}(\nu)$ and the corresponding parametrized implementation $\mathcal{I}_{\mathsf{SKE}}^{\mathsf{tran}}(\nu)$ $\mathcal{B}$ will use to simulate $\mathcal{I}_{\mathsf{SKE}}$. Analogously to Theorem 2, we use the data types and subtype relation from $\mathcal{M}_{\mathsf{SKE}}$. ν is expected to be an encoding of a list of triples (l, k, k') ($l \in \mathsf{labels}, k \in \{0,1\}^*$). The signature $\Sigma_{\mathsf{SKE}}^{\mathsf{tran}}$ is the following:

- deterministic $f_{k_x^l()}$ with $\mathrm{ar}(f_{k_x^l()}) = \tau_{\mathsf{SKE}}^{\mathsf{k}_x}$ for all labels $l \in \nu$
- randomized $f_{E_h(k_h^l(),0^\ell)}$ with $\mathrm{ar}(f_{E_h(k_h^l(),0^\ell)}) = \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}}$ for all $\ell \in \mathbb{N}$, $l \in \nu$
- randomized $f_{E_h(k_h^l(),\cdot)}$ with $\mathrm{ar}(f_{E_h(k_h^l(),\cdot)}) = \top \to \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}}$ for all $l \in \nu$
- randomized $f_{E_c(k_c^l(),\cdot)}$ with $\mathrm{ar}(f_{E_c(k_c^l(),\cdot)}) = \top \to \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}}$ for all $l \in \nu$

We specify a parametrized implementation $\mathcal{I}_{\mathsf{SKE}}^{\mathsf{tran}}(\nu)$ for $\mathcal{M}_{\mathsf{SKE}}^{\mathsf{tran}}$ as follows for $(l, k, k') \in \nu$:

- $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ f_{k_x^l()})(r)$ returns $\langle k', \tau_{\mathsf{SKE}}^{\mathsf{k}_x}\rangle$
- $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ f_{E_h(k_h^l(),0^\ell)})(r)$ returns $(M_{\mathsf{SKE}}\ E_h)(\langle k, \tau_{\mathsf{SKE}}^{\mathsf{k}_x}\rangle, 0^\ell; r)$
- $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ f_{E_h(k_h^l(),\cdot)})(m; r)$ returns $(M_{\mathsf{SKE}}\ E_h)(\langle k, \tau_{\mathsf{SKE}}^{\mathsf{k}_x}\rangle, m; r)$
- $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ f_{E_c(k_c^l(),\cdot)})(m; r)$ returns $(M_{\mathsf{SKE}}\ E_c)(\langle k, \tau_{\mathsf{SKE}}^{\mathsf{k}_x}\rangle, m; r)$

$(M_{\mathsf{SKE}}^{\mathsf{tran}}\ \mathsf{func})(b)$:
  if $b = \langle k', \tau_{\mathsf{SKE}}^{\mathsf{k}_x}\rangle$ for some $(l, k, k') \in \nu$ then
    return $f_{k_x^l()}$
  if $b \in \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}}$ then
    parse b as $\langle c, \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}}\rangle$
    for each $(l, k, k') \in \nu$ do
      let $m := \mathsf{SKE.Dec}(k, c)$
      if $m \neq \perp$ then
        if l belongs to an honest key then
          return $f_{E_h(k_h^l(),\cdot)}$
        else
          return $f_{E_c(k_c^l(),\cdot)}$
  return $\perp$

For b with $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ \mathsf{func})(b) = f_{E_h(k_h^l(),\cdot)}$ we have $(l, k) \in \nu$ with $\mathsf{SKE.Dec}(k, c) =: m \neq \perp$ for $b = \langle c, \tau_{\mathsf{SKE}}^{\mathsf{ciphertext}}\rangle$ and define $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ \mathsf{proj}\ f_{E_h(k_h^l(),\cdot)}\ 1)(b) := m$. Analogously for $(M_{\mathsf{SKE}}^{\mathsf{tran}}\ \mathsf{func})(b) = f_{E_c(k_c^l(),\cdot)}$.

Convert terms. Adversary $\mathcal{A}$ uses the function symbols of the original symbolic model for encryption $\mathcal{M}_{\mathsf{SKE}}$. Hence $\mathcal{B}$ needs to map these symbols to the corresponding transparent functions introduced by $\mathcal{B}$. Towards this goal we introduce the function convert as follows:

- $\mathsf{convert}(f^l(t_1, \ldots, t_n)) = f^l(\mathsf{convert}(t_1), \ldots, \mathsf{convert}(t_n))$ for all $f \notin \Sigma_{\mathsf{SKE}}$.
- $\mathsf{convert}(k_x^l()) = f_{k_x^l()}$
- $\mathsf{convert}(E_h^{\hat{l}}(k_h^l(), m)) = f^{\hat{l}(m)}_{E_h(k_h^l(),0^\ell)}()$ if $\hat{l} \in \mathsf{labelsH}$
- $\mathsf{convert}(E_h^{\hat{l}}(k_h^l(), m)) = f^{\hat{l}}_{E_h(k_h^l(),\cdot)}(\mathsf{convert}(m))$ if $\hat{l} \in \mathsf{labelsA}$
- $\mathsf{convert}(E_c^{\hat{l}}(k_c^l(), m)) = f^{\hat{l}}_{E_c(k_c^l(),\cdot)}(\mathsf{convert}(m))$

$\mathcal{B}$ simulates the game $\mathsf{DS}_{(\mathcal{M}\cup\mathcal{M}_{\mathsf{SKE}})\cup\mathcal{M}_{\mathsf{tran}}(\nu'),(\mathcal{I}\cup\mathcal{I}_{\mathsf{SKE}})\cup\mathcal{I}_{\mathsf{tran}}(\nu')}(\eta)$ for $\mathcal{A}$ while playing $\mathsf{DS}_{\mathcal{M}\cup(\mathcal{M}_{\mathsf{SKE}}^{\mathsf{tran}}(\nu)\cup\mathcal{M}_{\mathsf{tran}}(\nu')),\mathcal{I}(\cup\mathcal{I}_{\mathsf{SKE}}^{\mathsf{tran}}(\nu)\cup\mathcal{I}_{\mathsf{tran}}(\nu'))}(\eta)$. Note that we can generically compose $\mathcal{M}_{\mathsf{SKE}}^{\mathsf{tran}}(\nu) \cup \mathcal{M}_{\mathsf{tran}}(\nu')$ to one parametrized transparent model $\mathcal{M}'_{\mathsf{tran}}(\nu||\nu')$ since ν and ν' must be good (analogously for the implementation). However, for the sake of clarity, we keep them apart to distinguish the transparent functions (and parameter) provided by $\mathcal{A}$ from the additional transparent functions introduced by $\mathcal{B}$.

The simulation. $\mathcal{B}$ receives a parameter ν' from $\mathcal{A}$. $\mathcal{B}$ initializes the $\mathbb{T} := \emptyset$ of $\mathcal{A}$'s queries it maintains. Analogously to Theorem 2 $\mathcal{B}$ extracts the keys for SKE from T, H and sets up the parameter ν for $\mathcal{I}_{\mathsf{SKE}}^{\mathsf{tran}}(\nu)$ accordingly. It deals with request exactly as the simulator in Theorem 2.

Claim: Game 3 and Game 4 are indistinguishable.

This part is also completely analogous to the corresponding part in Theorem 2.

Claim: If $\mathcal{A}$ wins, then $\mathcal{B}$ wins Game 4 .

As well analogous to Theorem 2. □

8.5 Macs

In this section we show that any deduction sound implementation can be extended by a mac scheme. More precisely, we require a strongly EUF-CMA secure mac scheme.

8.5.1 Symbolic model

We first define the symbolic model $(\mathcal{T}_{\mathsf{MAC}}, \leq_{\mathsf{MAC}}, \Sigma_{\mathsf{MAC}}, \mathcal{D}_{\mathsf{MAC}})$ for macs. The signature Σ_{MAC} features the following function symbols:

$$k : \tau_{\mathsf{MAC}}^{\mathsf{k}}$$
$$mac : \tau_{\mathsf{MAC}}^{\mathsf{k}} \times \top \to \tau_{\mathsf{MAC}}^{\mathsf{mac}}$$

[5]There we excluded forged signatures as a way to produce non-DY terms for the adversary.

$\mathsf{open}_{\mathsf{MAC}}(c, L)$
  if $c \in [\![\mathcal{T}_{\mathsf{MAC}}]\!] \cap \mathrm{dom}(L)$ then
    return $(c, L(c))$
  else if $c = \langle k, \tau^{\mathsf{k}}_{\mathsf{MAC}} \rangle$ then
    return $(c, g^{l(c)}_{\tau^{\mathsf{k}}_{\mathsf{MAC}}})$
  else if $c = \langle \sigma, m, \tau^{\mathsf{mac}}_{\mathsf{MAC}} \rangle$ then
    for each $(\hat{k}, k^l) \in L$ do
      parse $\hat{k}$ as $\langle k, \tau^{\mathsf{k}}_{\mathsf{MAC}} \rangle$
      if $\mathsf{MAC.Vfy}(k, \sigma, m) = \mathsf{true}$ then
        return $(c, mac^{l(c)}(\hat{k}, m))$
    return $(c, g^{l(c)}_{\tau^{\mathsf{sig}}_{\mathsf{SIG}}})$
  else
    return $(c, g^{l(c)}_{\top})$

Figure 12: Open function for macs.

for $x \in \{c, h\}$. The randomized function symbol k of arity $\tau^{\mathsf{k}}_{\mathsf{MAC}}$ represents keys. The randomized function symbol *mac* of arity $\tau^{\mathsf{k}}_{\mathsf{MAC}} \times \top \to \tau^{\mathsf{mac}}_{\mathsf{MAC}}$ represents the mac of a message. To complete the formal definition we set the types

$$\mathcal{T}_{\mathsf{MAC}} := \{\top, \tau^{\mathsf{k}}_{\mathsf{MAC}}, \tau^{\mathsf{mac}}_{\mathsf{MAC}}\}$$

All introduced types are direct subtypes of the base type $\top$ (this defines $\leq_{\mathsf{MAC}}$). The deduction system captures the security of macs

$$\mathcal{D}_{\mathsf{MAC}} := \left\{ \frac{mac^{\hat{l}}(k^l(), m)}{m}, \qquad \frac{k^l() \quad m}{mac^{l_a}(k^l(), m)} \right\}$$

These rules are valid for arbitrary labels $l, \hat{l} \in \mathsf{labels}$ and adversarial labels $l_a \in \mathsf{labelsA}$. The following intuitions back up the rules:

- Macs reveal the message that was signed.
- The adversary can use known keys to deduce macs under those keys.

8.5.2 Implementation

We now give a concrete implementation $\mathcal{I}_{\mathsf{MAC}}$ for macs. The implementation uses some strongly EUF-CMA secure mac scheme ($\mathsf{MAC.KeyGen}$, $\mathsf{MAC.Mac}$, $\mathsf{MAC.Vfy}$). As usual, here $\mathsf{MAC.KeyGen}$ is a generation algorithm for key pairs, $\mathsf{MAC.Mac}$ computes a mac and $\mathsf{MAC.Vfy}$ is a verification algorithm. Note that $\mathsf{MAC.Mac}$ is an algorithm that takes three inputs: the key, the message to be authenticated and the randomness that is used for computing the mac.

The computable interpretations of k and *mac* are as follows:

- $(M_{\mathsf{MAC}}\ k)(r)$: Let $k := \mathsf{MAC.Mac}(1^\eta, r)$. Return $\langle k, \tau^{\mathsf{k}}_{\mathsf{MAC}} \rangle$.
- $(M_{\mathsf{MAC}}\ sig)(\hat{k}, m; r)$: Parse $\hat{k}$ as $\langle k, \tau^{\mathsf{k}}_{\mathsf{MAC}} \rangle$. Let $\sigma := \mathsf{MAC.Mac}(k, m, r)$ and return $\langle \sigma, m, \tau^{\mathsf{mac}}_{\mathsf{MAC}} \rangle$.

The $\mathsf{valid}_{\mathsf{MAC}}$ *predicate.*

Based on the current trace $\mathbb{T}$ of all parse and generate requests of the adversary, the predicate $\mathsf{valid}_{\mathsf{MAC}}$ returns true only if the following conditions hold:

1. The trace starts with a query "$\mathsf{init}\ T, H$" (where T and H may be the empty list respectively). There are no further init queries.
2. The adversary may only generate keys in the init query. Concretely, this is guaranteed by the following rules:
   (a) For the query "$\mathsf{init}\ T, H$", the function symbol k may only occur in a term $k^h() \in T \cup H$ (i.e., not as subterm of other terms) for $l \in \mathsf{labelsH}$. Any label h for $k^h()$ must be unique in $T \cup H$.
   (b) Any occurrence of $k^h()$ in a $\mathsf{generate}$ query must have occurred in the init query.
3. The adversary must not use the function symbol *mac* in the init query.
4. $k^h()$ may only occur as the first argument for *mac*.

8.6 MAC composability

THEOREM 5. *Let $\mathcal{M}$ be a symbolic model and $\mathcal{I}$ be deduction sound implementation of $\mathcal{M}$. If ($\mathcal{M}_{\mathsf{MAC}}$,$\mathcal{I}_{\mathsf{MAC}}$) and($\mathcal{M}$,$\mathcal{I}$) are compatible (see the conditions in Section 6), then $\mathcal{I} \cup \mathcal{I}_{\mathsf{MAC}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{MAC}}$ for any $\mathcal{I}_{\mathsf{MAC}}$ constructed from a strong EUF-CMA secure mac scheme.*

PROOF. This proof is very similar to Theorem 3.

Game 0.

Game 0 is the original deduction soundness game for $\mathcal{I} \cup \mathcal{I}_{\mathsf{MAC}}$.

Game 1.

In Game 1 we abort in case of collisions. Game 0 and Game 1 are indistinguishable by Lemma 2 and using the fact that our implementation is collision free.

Game 2.

Analogously to Game 2 from Theorem 3 we change the deduction system to prevent the adversary from winning using reconstructed macs in Game 2. Concretely, we add rules that allow to deduce every honestly generated mac that is a subterm of a term t from t. Game 1 and Game 2 are indistinguishable since any $\mathcal{A}$ that notices a difference with non-negligible probability could be used to construct a successful adversary against the deduction soundness of $\mathcal{I}$.

Game 3.

In Game 3, analogously to Game 2 from Theorem 3, we change the deduction system to make arbitrary macs deducible. Furthermore we use random bitstrings to represent honestly generated mac-keys in the library. An adversary that can distinguish Game 2 and Game 3 can be used to break the strong EUF-CMA security of the mac scheme. It will either produce a forgery or one of the honest keys.

Game 4.

Finally, in Game 4, we simulate Game 3 using transparent functions for macs while playing the deduction soundness game for $\mathcal{I}$. Any adversary winning this game lets the simulator break the deduction soundness game of $\mathcal{I}$. By requirement this can only happen with negligible probability which concludes our proof. □

$\mathsf{open}_{\mathsf{HASH}}(c, L)$
  if $c \in [\![\mathcal{T}_{\mathsf{HASH}}]\!] \cap \mathrm{dom}(L)$ then
    return $(c, L(c))$
  else if $c = \langle h, \tau_{\mathsf{HASH}} \rangle$ then
    return $(c, g^{l(c)}_{\tau_{\mathsf{HASH}}})$
  else
    return $(c, g^{l(c)}_{\top})$

Figure 13: Open function for hash functions.

8.7 Hash functions

In this section we deal with the composition of deduction sound implementations of arbitrary primitives with hash functions. We consider hash functions implemented as random oracles [5]: in this setting calls to the hash function are implemented by calls to a random function which can only be accessed in a black-box way. We model this idea directly in our framework. In the symbolic model model we consider a symbolic function that is randomized and which is implemented by a randomized function. We recover the intuition that hash functions are deterministic by restricting the calls that an adversary can make: for each term t, the adversary can only call the hash function with the honest label $l(t)$.

8.7.1 Symbolic model

The symbolic model for hash functions is rather standard. It is given by the tuple $(\mathcal{T}_{\mathsf{HASH}}, \leq_{\mathsf{HASH}}, \Sigma_{\mathsf{HASH}}, \mathcal{D}_{\mathsf{HASH}})$ where

$$\mathcal{T}_{\mathsf{HASH}} := \{\top, \tau_{\mathsf{HASH}}\}$$

and $\tau_{\mathsf{HASH}} \leq_{\mathsf{HASH}} \top$. The signature Σ_{HASH} contains only a randomized function $H : \top \to \tau_{\mathsf{HASH}}$ characterized by the deduction rule:

$$\mathcal{D}_{\mathsf{HASH}} := \left\{ \ \frac{m}{H^l(m)} \ \right\}$$

where $l \in \mathsf{labelsH}$.

8.7.2 Implementation

The implementation $\mathcal{I}_{\mathsf{HASH}}$ for hash functions is via a randomized function: when called, the function simply returns a random value, and we will require that it does so consistently; Concretely $(M_{\mathsf{HASH}}\ H)(m; r)$ returns $\langle r, \tau_{\mathsf{HASH}} \rangle$.

The open function for hash functions is described in Figure 13. If the bitstring to be opened was not the result of a generate call, then it returns garbage of types either τ_{HASH} or $\top$, depending on what c encodes. Otherwise, it will return the entry in L that corresponds to c: by the requirements posed by $\mathsf{valid}_{\mathsf{HASH}}$ below this will be $H^{l(t)}(m)$ for some bitstring m with $L[[m]] = t$.

A useful observation is that by the description above, the library L will never contain an entry of the form $(c, H^l(m))$ for some adversarial label $l \in \mathsf{labelsA}$; moreover, if $(c, H^l(m))$ is in L, then $l = l(t)$ for some t, and $L[[m]] = t$.

The $\mathsf{valid}_{\mathsf{HASH}}$ *predicate.*

For simplicity we require that no hash is present in init requests (our results easily extend to the case where this restriction is not present). In addition we use the predicate $\mathsf{valid}_{\mathsf{HASH}}$ to enforce deterministic behavior of our hash implementation. We require that for any term t, all occurrences of $H(t)$ in generate and sgenerate requests use the same label. Concretely, we demand that for any term t, all generate requests for $H^{\hat{l}}(t)$ are labeled with the honest label $\hat{l} = l(t)$. The choice of label is not important: we could alternatively request that if $H^{l_1}(t)$ and $H^{l_2}(t)$ occur in a generate requests, then $l_1 = l_2$.

THEOREM 6. *Let $\mathcal{I}$ be a deduction sound implementation of $\mathcal{M}$. If $(\mathcal{M}_{\mathsf{HASH}}, \mathcal{I}_{\mathsf{HASH}})$ and $(\mathcal{M}, \mathcal{I})$ are compatible, then $\mathcal{I} \cup \mathcal{I}_{\mathsf{HASH}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{HASH}}$ in the random oracle model.*

The intuition behind this proof is simple: collisions due to tagging occur only with probability given by the birthday bound (so with negligible probability). Given an adversary that wins the deduction soundness game for the composed libraries, we construct an adversary that breaks deduction soundness of $(\mathcal{M}, \mathcal{I}, \mathsf{valid}_{\mathcal{I}})$. This latter adversary simulates the hash function via a randomized transparent function with no arguments: a generate $H^{l(t)}(t)$ call will be implemented by a generate call to $f^{l(t)}()$. Due to $\mathsf{valid}_{\mathsf{HASH}}$ the knowledge set S does not contain any occurrence of H with a dishonest label, hence the only "useful" deduction soundness rule which allows the adversary to learn/manipulate terms with dishonest labels are not applicable (we can cut them out of any deduction).

PROOF. Consider an adversary $\mathcal{A}$ that breaks deduction soundness of implementation $\mathcal{I} \cup \mathcal{I}_{\mathsf{HASH}}$ for $\mathcal{M} \cup \mathcal{M}_{\mathsf{HASH}}$, i.e.

$$\mathbb{P}\left[\mathsf{DS}_{(\mathcal{M} \cup \mathcal{M}_{\mathsf{HASH}}) \cup \mathcal{M}_{\mathsf{tran}}(\nu), (\mathcal{I} \cup \mathcal{I}_{\mathsf{HASH}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu), \mathcal{A}}(\eta) = 1\right]$$

is non-negligible for some some choice of $\mathcal{M}_{\mathsf{tran}}, \mathcal{I}_{\mathsf{tran}}$. We consider the transparent model/implementation $\mathcal{M}'_{\mathsf{tran}}, \mathcal{I}'_{\mathsf{tran}}$ obtained by adding to the functions in $\mathcal{M}_{\mathsf{tran}}$ a new (randomized) function f_H of arity 0; the implementation of the function is given by M_{f_H} defined by: $(M_{f_H}\ f_H)(r) = \langle r, \mathsf{HASH} \rangle$, i.e. the machine that simply outputs a proper encoding of its random coins.

We next show that adversary $\mathcal{A}$ yields an adversary $\mathcal{B}$ that contradicts the deduction soundness of $\mathcal{I}$ with respect to $\mathcal{M}$ when the transparent model/implementation is $(\mathcal{M}'_{\mathsf{tran}}, \mathcal{I}'_{\mathsf{tran}})$ defined above. Adversary $\mathcal{B}$ that we construct translates the queries of $\mathcal{A}$ into queries for $(\mathcal{M} \cup \mathcal{M}'_{\mathsf{tran}}, \mathcal{I} \cup \mathcal{I}'_{\mathsf{tran}})$ by using f_H to implement the hash function. This is accomplished using a conversion function convert from terms in $\mathcal{M} \cup \mathcal{M}_{\mathsf{HASH}} \cup \mathcal{M}_{\mathsf{tran}}$ to terms in $\mathcal{M} \cup \mathcal{M}'_{\mathsf{tran}}$.

- $\mathsf{convert}(f^l(t_1, \ldots, t_n)) = f^l(\mathsf{convert}(t_1), \ldots, \mathsf{convert}(t_n))$ for all $f \neq H$.
- $\mathsf{convert}(H^{l(t)}(t)) = f_H^{l(t)}$

The inverse of the convert function is defined in the obvious way. These conversion of terms will still preserve the validity of $\mathcal{B}$'s trace for every valid trace of $\mathcal{A}$ due to requirement (i) for valid predicates.

Adversary $\mathcal{B}$ processes the queries of $\mathcal{A}$ as follows.

init query. $\mathcal{B}$ forwards the init request to his game and forwards the answer to $\mathcal{A}$.

generate queries. For each request "generate t": for any t' such that $H^l(t') \in st(t)$ adversary $\mathcal{B}$ issues "sgenerate $\mathsf{convert}(t')$" (for convenience, we assume the order of these requests is in bottom up manner). These queries are valid by requirement (ii) for valid predicates. It then issues "generate $\mathsf{convert}(t)$" and returns the answer to this last query to $\mathcal{A}$. The additional sgenerate queries are necessary to preserve

an invariant on the libraries needed to show the indistinguishability of the real game and the simulation (see indistinguishability of Game 2 and Game 3 in Theorem 2).

$\mathcal{B}$ proceeds analogously for $\mathsf{sgenerate}$ requests (but no answer is returned to $\mathcal{A}$).

parse queries. For each request "$\mathsf{parse}\ c$" $\mathcal{B}$ sends "$\mathsf{parse}\ c$" to its game and receives a term t. $\mathcal{B}$ sends $\mathsf{convert}^{-1}(t)$ to $\mathcal{A}$.

We conclude by arguing that if $\mathcal{A}$ is successful, then so is $\mathcal{B}$. Let $\mathsf{Terms}_1 = \mathsf{Terms}(\Sigma \cup \Sigma_{\mathsf{HASH}} \cup \Sigma_{\mathsf{tran}})$ and $\mathsf{Terms}_2 = \mathsf{Terms}(\Sigma \cup \Sigma'_{\mathsf{tran}})$. Let $\vdash_1$ be the deduction system defined by $\mathcal{D} \cup \mathcal{D}_{\mathsf{HASH}} \cup \mathcal{D}_{\mathsf{tran}}$, and let $\vdash_2$ the one defined by $\mathcal{D} \cup \mathcal{D}_{\mathsf{tran}}$. Let $\mathcal{R} : \mathsf{Terms}_2 \to \{0,1\}^\eta$ be an arbitrary randomness assignment and $r_{\mathcal{A}}$ be arbitrary random coins for $\mathcal{A}$. Then adversary $\mathcal{B}$ simulates for $\mathcal{A}$ the game

$$\mathsf{DS}_{(\mathcal{M}\cup\mathcal{M}_{\mathsf{HASH}})\cup\mathcal{M}_{\mathsf{tran}}(\nu),(\mathcal{I}\cup\mathcal{I}_{\mathsf{HASH}})\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}$$

here the coins of adversary $\mathcal{A}$ are $r_{\mathcal{A}}$, and the randomness assignment $\mathcal{R}_1 : \mathsf{Terms}_1 \to \{0,1\}^\eta$ is defined by $\mathcal{R}_1(t) = \mathcal{R}_2(\mathsf{convert}(t))$.

In addition, if $L_2(\mathcal{R}_2, r_{\mathcal{A}})$ is the mapping maintained in $\mathsf{DS}_{\mathcal{M}\cup\mathcal{M}'_{\mathsf{tran}},\mathcal{I}\cup\mathcal{I}'_{\mathsf{tran}},\mathcal{B}}(\eta)$ then $(c,t) \in L_1$ if and only if $(c, \mathsf{convert}(t)) \in L_2$.

Next we show that if $\mathsf{parse}(\mathsf{convert}(t))$ is a Dolev-Yao request by $\mathcal{B}$, then $\mathsf{convert}(t)$ is a Dolev-Yao request by $\mathcal{A}$. This implies that if $\mathcal{A}$ is non Dolev-Yao, then so is $\mathcal{B}$.

Consider an arbitrary $\mathsf{parse}(c)$ request by $\mathcal{A}$, and let S be the set of terms present in all of the generate requests of $\mathcal{A}$. Per our construction, $\mathsf{convert}(S)$ is the set of terms in the generate requests of $\mathcal{B}$ (where $\mathsf{convert}$ is extended from terms to sets of terms in the obvious way). Assume there exists a proof $\mathsf{convert}(S) = S'_0 \overset{\alpha_1}{\to} S'_1 \overset{\alpha_2}{\to} S'_2 \ldots \overset{\alpha_n}{\to} S'_n$ with $\mathsf{convert}(t) \in S'_n$ for $\mathsf{convert}(S) \vdash_2 \mathsf{convert}(t)$. We show we can construct a proof for $S \vdash_1 t$.

By a previous remark, S and t do not contain any occurence of H^l with an adversarial label l. The only way to introduce instances of f_H labeled with an adversarial label is to use the rule instantiation $\frac{}{f_H^l}$. Assume that for some $i \in \{1,2,\ldots,n\}$ we have $S'_{i-1} \overset{\alpha_i}{\to} S'_i$ and α_i is the rule $\frac{}{f_H^l}$ for some adversarial label l. To eliminate the use of the rule let t be an arbitrary term in S, and consider the subsitution θ that replaces f_H^l with $\mathsf{convert}(t)$. Then $\mathsf{convert}(S) = S'_0 \overset{\alpha_1}{\to} \theta(S'_1) \overset{\alpha_2}{\to} \theta(S'_2) \ldots \overset{\alpha_{i-1}}{\to} \theta(S_{i-1}) \overset{\alpha_{i+1}}{\to} \theta(S_{i+1}) \ldots \overset{\alpha_n}{\to} \theta(S'_n)$ is a valid derivation for $\mathsf{convert}(t)$ which does not use the rule. Iteratively, we obtain a derivation $\mathsf{convert}(S) = S'_0 \overset{\alpha_1}{\to} S''_1 \ldots \alpha_m \to S''_m$ for $\mathsf{convert}(t)$ and if f_H^l occurs in any set, then $l = l(t)$ and is an honest label. We can therefore apply $\mathsf{convert}^{-1}$ to the above proof to obtain a proof for $S \vdash_1 t$. Hence $\mathcal{B}$ wins if $\mathcal{A}$ wins. $\square$

9. FORGETFULNESS

All the theorems from Section 8 have one important drawback: Key material cannot be sent around as the valid predicates forbid keys from being used in non-key positions. This takes the analysis of a large class of practical protocols (e.g, many key exchange protocols) outside the scope of our results. The problem is that deduction soundness does not guarantee that no information about non-DY terms is leaked by the computational implementation. E.g., we could think of a deterministic function symbol f that takes arguments of type nonce with only the rule $\frac{n^l()}{f(n^l())}$. An implementation of f could leak half of the bits of its input and still be sound. However, to send key material around, we need to rely on the fact that information theoretically nothing is leaked about the suitable positions for keys.

To solve this problem, we introduce *forgetful* symbolic models and implementations. A forgetful symbolic models features function symbols with positions that are marked as being forgetful. The corresponding implementation has to guarantee, that no information about the arguments at these positions will be leaked (except their length). We will formalize this intuition later in Definition 9. We start off by introducing some necessary extensions of our previous setting to allow for the concept of forgetfulness.

9.1 Preliminaries

We need to extend some definitions to capture the concept of forgetfulness.

Changed hybrid terms for function symbols with forgetful arguments.

To allow the handling of forgetful positions, extend the definition for hybrid terms with function symbols carrying an honest label in the library. Let f be a function symbol of arity $\mathrm{ar}(f) = \tau_1 \times \ldots \times \tau_n \to \tau$. Then a hybrid term of f may be $f^l(a_1,\ldots,a_n)$ where each a_i is either a bitstring from $[\![\tau_i]\!]$ or a term of type τ_i for forgetful positions i. For normal positions a_i must be a bitstring from $[\![\tau_i]\!]$ as usual. The definitions for the completeness of a library L and $L[[c]]$ are changed accordingly.

New valid *requirements.*

To allow forgetful arguments to be useful, we have to change the definition of valid requirements. Concretely, we allow the behavior of valid to additionally depend on a signature Σ_{valid} that features forgetful positions, i.e., positions of function symbols in $f \in \Sigma_{\mathsf{valid}}$ may be marked as forgetful. We then restate the requirements for valid as follows:

(i) If $\mathsf{valid}(\mathbb{T}+q) = \mathsf{true}$, then $\mathsf{valid}(\mathbb{T}+\hat{q}) = \mathsf{true}$ where $\hat{q}$ is a *variation* of q: If $q =$"$\mathsf{generate}\ t$", then $\hat{q} =$"$\mathsf{generate}\ \hat{t}$" (analogously for "$\mathsf{sgenerate}\ t$"). Here, $\hat{t}$ is a variation of t according to the following rule: Any subterm $f^l(t_1,\ldots,t_n)$ of t where $f \notin \Sigma \cup \Sigma_{\mathsf{valid}}$ is a foreign function symbol may be replaced by $\hat{f}^{\hat{l}}(\hat{t}_1,\ldots,\hat{t}_m)$ where $\hat{f} \notin \Sigma \cup \Sigma_{\mathsf{valid}}$ is a foreign function symbol and $\hat{t}_i = t_j$ for some $j \in \{1,\ldots,n\}$ (where each t_j may only be used once) or $\hat{t}_i$ does not contain function symbols from $\Sigma \cup \Sigma_{\mathsf{valid}}$. As a special case we may also replace $f^l(t_1,\ldots,t_n)$ with a term $\hat{t}_1$ (i.e., $\hat{f}$ is "empty"). If $q =$"$\mathsf{init}\ T, H$" then $\hat{q} =$"$\mathsf{init}\ \hat{T}, \hat{H}$" where $T = (t_1,\ldots,t_n)$ and $\hat{T} = (\hat{t}_1,\ldots,\hat{t}_n)$ and $\hat{t}_i$ is a variation of t_i ($\hat{H}$ analogously).

(ii) If $\mathsf{valid}(\mathbb{T}+q) = \mathsf{true}$ and t is a term occurring in q, then $\mathsf{valid}(\mathbb{T}+$"$\mathsf{sgenerate}\ t'$"$) = \mathsf{true}$ for any subterm t' of t that is not a subterm at a forgetful position.

(iii) $\mathsf{valid}(\mathbb{T})$ can be evaluated in polynomial time (in the length of the trace $\mathbb{T}$).

Basically, valid is now allowed to make statements about how the own fuction symbols (from Σ) are allowed to be used in the context of some foreign function symbols (Σ_{valid})

with forgetful positions. Consequently, we do require that a trace remains valid if those function symbols are replaced (see new requirement i). Furthermore, we do not require valid to allow for silent generation of subterms at forgetful positions because it might be essential that those subterms are never generated (see new requirement ii).

9.2 Forgetful symbolic models and implementations

We say that a symbolic model $\mathcal{M}$ is a *forgetful symbolic model* if arguments of a function symbol may be marked as *forgetful*. In order to formalize forgetful implementations, the computational counterpart of forgetful positions, we introduce the notion of an oblivious implementation. These are implementations for symbolic functions which can take as input natural numbers instead of actual bitstrings of the appropriate sort.

DEFINITION 8 (OBLIVIOUS IMPLEMENTATION). *Let $\mathcal{M}$ be a forgetful symbolic model. $\overline{\mathcal{I}} = (\overline{M}, [\![\cdot]\!], len, \mathsf{open}, \mathsf{valid})$ is an* oblivious implementation of $\mathcal{M}$ *if $\overline{\mathcal{I}}$ is an implementation of $\mathcal{M}$ with a slightly changed signature: For each function symbol $f \in \Sigma$ with arity $\mathrm{ar}(f) = \tau_1 \times \ldots \times \tau_n \to \tau$ the signature of $(\overline{M}\ f)$ is $\theta(\tau_1) \times \cdots \times \theta(\tau_n) \times \{0,1\}^\eta \to [\![\tau]\!]$ where $\theta(\tau_i) = \mathbb{N}$ if the ith argument of f is forgetful and $[\![\tau_i]\!]$ otherwise.*

Intuitively, oblivious implementations for all forgetful positions, take as input natural numbers; these will be the length of the actual inputs on the forgetful positions.

As indicated above, a forgetful implementation is one which is indistinguishable from an oblivious implementation. To formally define the notion we introduce a distinguishing game $\mathsf{FIN}^b_{\mathcal{M}(\nu),\mathcal{I}(\nu),\overline{\mathcal{I}}(\nu),\mathcal{A}}(\eta)$ where an adversary $\mathcal{A}$ tries to distinguish between the case when he interacts with the real implementation, or with an alternative implementation that is oblivious with respect to all of the forgetful arguments. We say that an implementation is forgetful, if there exists an oblivious implementation such that no adversary succeeds in this task.

DEFINITION 9 (FORGETFUL IMPLEMENTATION). *We say that an implementation $\mathcal{I} = (M, [\![\cdot]\!], len, \mathsf{open}, \mathsf{valid})$ is a* forgetful implementation of a forgetful symbolic model $\mathcal{M}$ *if there is an oblivious implementation $\overline{\mathcal{I}} = (\overline{M}, [\![\cdot]\!], len, \mathsf{open}, \mathsf{valid})$ such that for all for all parametrized transparent symbolic models $\mathcal{M}_{\mathsf{tran}}(\nu)$ and for all parametrized transparent implementations $\mathcal{I}_{\mathsf{tran}}(\nu)$ of $\mathcal{M}_{\mathsf{tran}}(\nu)$ compatible with $(\mathcal{M},\mathcal{I})$ we have that*

$$Prob[\mathsf{FIN}^0_{\mathcal{M}\cup\mathcal{M}_{\mathsf{tran}}(\nu),\mathcal{I}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\overline{\mathcal{I}}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta) = 1]$$
$$-Prob[\mathsf{FIN}^1_{\mathcal{M}\cup\mathcal{M}_{\mathsf{tran}}(\nu),\mathcal{I}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\overline{\mathcal{I}}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta) = 1]$$

is negligible for every p.p.t. adversary $\mathcal{A}$.

LEMMA 5. *Let $\mathcal{M}$ be an forgetful symbolic model, $\mathcal{I}$ be an forgetful implementation of $\mathcal{M}$ and $\overline{\mathcal{I}}$ a corresponding oblivious implementation. If $\mathcal{I}$ is deduction sound, then $\overline{\mathcal{I}}$ is deduction sound with respect to the deduction soundness game DS' that uses $\mathsf{generate}^{\mathsf{FIN}}$ (Figure 14) instead of $\mathsf{generate}$.*

PROOF. Let $\mathcal{A}$ be a p.p.t. adversary that wins the deduction soundness game for $\overline{\mathcal{I}}$ with non-negligible probability. We construct an adversary $\mathcal{B}$ that plays the game

$$\mathsf{FIN}^b_{\mathcal{M}\cup\mathcal{M}_{\mathsf{tran}}(\nu),\mathcal{I}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\overline{\mathcal{I}}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta)$$

$\mathsf{generate}^{\mathsf{FIN}}_{M,\mathcal{R}}(t, L)$:
  if for some $c \in \mathrm{dom}(L)$ we have $L[[c]] = t$ then
    return c
  else
    for $i \in \{1, n\}$ do
      if i is a forgetful argument then
        let $c_i := len(t_i)$
        let $a_i := t_i$
      else
        let $(c_i, L) := \mathsf{generate}_{M,\mathcal{R}}(t_i, L)$
        let $a_i := c_i$
    let $r := \mathcal{R}(t)$
    let $c := (M\ f)(c_1, \ldots, c_n; r)$
    let $L(c) := f^l(a_1, \ldots, a_n))$ $(l \in \mathsf{labelsH})$
    return (c, L)

Figure 14: The generate function for an oblivious implementation (t is of the form $f^l(t_1, \ldots, t_n)$ (with possibly $n = 0$ and no label l for deterministic function symbols f)). The requirements for the input t are those of the normal $\mathsf{generate}$ function.

and simulates the deduction soundness game for $\mathcal{A}$ (by just relaying the queries of $\mathcal{A}$). Depending on b, this is a perfect simulation of

$$\mathbb{P}\left[\mathsf{DS}_{\mathcal{M}\cup\mathcal{M}_{\mathsf{tran}}(\nu),\mathcal{I}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta) = 1\right]$$

or of the variant of the game for $\overline{\mathcal{I}}$

$$\mathbb{P}\left[\mathsf{DS}'_{\mathcal{M}\cup\mathcal{M}_{\mathsf{tran}}(\nu),\overline{\mathcal{I}}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta) = 1\right]$$

If $\mathcal{A}$ wins the deduction soundness game, $\mathcal{B}$ wins its game as well. Otherwise, i.e., if $\mathcal{A}$ is invalid $\mathcal{B}$ picks a random bit b and sends "$\mathsf{guess}\ b$" to its game. Since $\mathcal{I}$ is deduction sound, $\mathcal{A}$ will only win the first game with negligible probability. If $\mathcal{A}$ wins the game for $\overline{\mathcal{I}}$ with non-negligible probability, $\mathcal{B}$ has a non-negligible advantage. This contradicts the assumption that $\overline{\mathcal{I}}$ is an oblivious implementation corresponding to $\mathcal{I}$. □

Let $\overline{\mathcal{M}_{\mathsf{PKE}}}$ be the forgetful symbolic model derived from the symbolic symbolic model $\mathcal{M}_{\mathsf{PKE}}$ from Section 8.2 by marking the message m for honest encryptions $enc_h(ek, m)$ as forgetful. Then Lemma 6 capture the intuition that public key encryption schemes are forgetful with respect to their messages.

LEMMA 6. *$\mathcal{I}_{\mathsf{PKE}}$ from Section 8.2 is a forgetful implementation of $\overline{\mathcal{M}_{\mathsf{PKE}}}$.*

PROOF. We define an oblivious implementation $\overline{\mathcal{I}_{\mathsf{PKE}}}$ with the Turing Machine $\overline{M_{\mathsf{PKE}}}$ that differs only for the function symbol enc_h from M_{PKE}. We set $(\overline{M_{\mathsf{PKE}}}\ enc_h)(ek, \ell; r) := (M_{\mathsf{PKE}}\ enc_h)(ek, 0^\ell; r)$. $\overline{\mathcal{I}_{\mathsf{PKE}}}$ witnesses that $\mathcal{I}_{\mathsf{PKE}}$ is a forgetful implementation of $\overline{\mathcal{M}_{\mathsf{PKE}}}$.

Let $\mathcal{A}$ be a p.p.t. adversary such that the probability from Definition 9 is non-negligible. We can then use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that wins the IND-CCA game from Figure 8 with non-negligible probability. $\mathcal{B}$ simulates

$$\mathsf{FIN}^b_{\overline{\mathcal{M}_{\mathsf{PKE}}}\cup\mathcal{M}_{\mathsf{tran}}(\nu),\mathcal{I}_{\mathsf{PKE}}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\overline{\mathcal{I}_{\mathsf{PKE}}}\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta)$$

for $\mathcal{A}$ where the bit b corresponds the the bit picked by the IND-CCA game from Figure 8 ($b = 0$: produce encryptions of 0, $b = 1$ produce encryptions of the real messages).

$\mathsf{FIN}^b_{\mathcal{M}(\nu),\mathcal{I}(\nu),\overline{\mathcal{I}}(\nu),\mathcal{A}}(\eta)$:
  let $S := \emptyset$ (set of requested terms)
  let $L := \emptyset$ (library)
  let $\mathbb{T} := \emptyset$ (trace of queries)
  $\mathcal{R} \leftarrow \{0,1\}^*$ (random tape)

  if $b = 0$ then
    let $\mathsf{generate} := \mathsf{generate}^{\mathsf{FIN}}_{M,\mathcal{R}}$
  else
    let $\mathsf{generate} := \mathsf{generate}_{M,\mathcal{R}}$

  Receive parameter ν from $\mathcal{A}$

  on request "$\mathsf{init}\ T, H$" do
    add "$\mathsf{init}\ T$" to $\mathbb{T}$
    if $\mathsf{valid}(\mathbb{T})$ then
      let $S := S \cup T$
      let $C := \emptyset$ (list of replies)
      for each $t \in T$ do
        let $(c, L) := \mathsf{generate}(t, L)$
        let $C := C \cup \{c\}$
      for each $t \in H$ do
        let $(c, L) := \mathsf{generate}(t, L)$
      send C to $\mathcal{A}$
    else
      return 0 ($\mathcal{A}$ is invalid)

  on request "$\mathsf{sgenerate}\ t$" do
    if $\mathsf{valid}(\mathbb{T}$+"$\mathsf{sgenerate}\ t$") then
      let $(c, L) := \mathsf{generate}(t, L)$

  on request "$\mathsf{generate}\ t$" do
    add "$\mathsf{generate}\ t$" to $\mathbb{T}$
    if $\mathsf{valid}(\mathbb{T})$ then
      let $S := S \cup \{t\}$
      let $(c, L) := \mathsf{generate}(t, L)$
      send c to $\mathcal{A}$
    else
      return 0 ($\mathcal{A}$ is invalid)

  on request "$\mathsf{parse}\ c$" do
    let $(t, L) := \mathsf{parse}(c, L)$
    if $S \vdash_{\mathcal{D}} t$ then
      send t to $\mathcal{A}$
    else
      return 1 ($\mathcal{A}$ produced non-Dolev-Yao term)

  on request "$\mathsf{guess}\ b'$" do
    if $b = b'$ then
      return 1 ($\mathcal{A}$ wins)
    else
      return 0 ($\mathcal{A}$ looses)

Figure 15: Indistinguishability game for forgetful implementations.

The simulation works analogously to Game 2 in Theorem 2. Since $\mathcal{B}$ does not know the encryption keys while playing the IND-CCA game, we need to randomize them in the library. The arguments from the proof of indistinguishability of Game 1 and Game 2 in Theorem 2 can be easily translate to the setting at hand and show that the simulation, although not perfect, is indistinguishable from FIN^0 and FIN^1 respectively. Hence, $\mathcal{B}$ would break the IND-CCA security of the public key encryption scheme if such an adversary $\mathcal{A}$ would existed. □

9.3 Sending keys around

To be able to consider the case when symmetric keys are sent encrypted we introduce an extension of the model for symmetric key encryption of Section 8.4. The extension is that the $\mathsf{valid}_{\mathsf{SKE}}$ predicate can now depend on a signature Σ_{valid} that contains functions with forgetful positions. The new predicate allows for standard generation of keys for symmetric encryption (with the same restrictions as those in Section 8.4), but in addition it also allows for generate requests that contain occurrences of symmetric keys under functions from signature Σ_{valid}, as long as the occurrences are on forgetful positions.

Concretely, based on $\mathcal{I}_{\mathsf{SKE}}$ from Section 8.4 we introduce the implementation $\mathcal{I}_{\mathsf{SKE}}[\Sigma_{\mathsf{valid}}]$ for a signature Σ_{valid} featuring forgetful positions. We define the $\mathsf{valid}_{\mathsf{SKE}}$ predicate based on Σ_{valid} and, instead of requirement 2, now require:

1. For the query "$\mathsf{init}\ T, H$", the function symbol k_c may only occur in a term $k_c^l() \in T$. Analogously, k_h may only occur in H. Any label l for $k_x^l()$ must be unique in $T \cup H$.

2. Any occurrence of $k_x^l()$ in a $\mathsf{generate}$ query must have occurred in the init query. $k_x^l()$ may only occur as the first argument to E_x or as a subterm of a forgetful position for a function symbol $f \in \Sigma_{\mathsf{valid}}$.

We show in Theorem 7, that we can compose our extended implementation $\mathcal{I}_{\mathsf{SKE}}[\Sigma_{\mathsf{valid}}]$ (extended in the sense that its valid predicate allows for more scenarios) with any deduction sound forgetful implementation and preserve deduction soundness. Since the implementation for public key encryption $\mathcal{I}_{\mathsf{PKE}}$ from Section 8.2 is a forgetful implementation for the forgetful symbolic model $\overline{\mathcal{M}_{\mathsf{PKE}}}$ by Lemma 6, queries like "$\mathsf{generate}\ enc_h^{\tilde{l}}(ek_h^l(), k_h^{\tilde{l}}())$" are now possible. Intuitively, this corresponds to sending around symmetric keys encrypted under asymmetric keys in a protocol.

Furthermore, we show that, in the case of secret key encryption, forgetfulness is preserved as well (Theorem 8). This even holds for the obvious forgetful symbolic model of secret key encryption where the message position for honest encryptions under honest keys is a forgetful one. I.e., we could add several layers of secret key encryption to allow for the encryption of symmetric keys under other symmetric keys.

The last aspect shows why we need to fix the set of function symbols Σ_{valid} at the time of composition: We cannot allow to encrypt keys under forgetful positions in general since it would be impossible for $\mathsf{valid}_{\mathsf{SKE}}$ to detect key cycles. E.g., assume that Σ_{valid} contains a function symbol f with a forgetful second position. Do the terms $f^{\tilde{l}}(t', k_h^l())$ and $E_h^{\tilde{l}}(k_h^l(), t')$ contain a key cycle? We cannot tell without knowing the implementation of f and t'. Therefore we have

to require that the valid predicate of the implementation we are composing $\mathcal{I}_{\mathsf{SKE}}$ with does rely on the forgetfulness of function symbols from Σ_{SKE} in Theorem 7.

THEOREM 7. *Let $\mathcal{M}$ be a forgetful symbolic model and $\mathcal{I}$ be a forgetful deduction sound implementation of $\mathcal{M}$. $\mathcal{I}_{\mathsf{SKE}}$ denotes $\mathcal{I}_{\mathsf{SKE}}[\Sigma]$ where Σ is the signature from $\mathcal{M}$. If $(\mathcal{M}_{\mathsf{SKE}}, \mathcal{I}_{\mathsf{SKE}})$ and $(\mathcal{M}, \mathcal{I})$ are compatible (see requirements in Section 6) and the* valid *predicate of $\mathcal{I}$ does not depend on function symbols from Σ_{SKE}, then $\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}$ is a deduction sound implementation of $\mathcal{M} \cup \mathcal{M}_{\mathsf{SKE}}$.*

PROOF. This proof is very similar to that for Theorem 4. Basically, we just introduce an additional game hop where we replace $\mathcal{I}$ by an oblivious implementation $\overline{\mathcal{I}}$. This guarantees that, even if the adversary requests to generate a term t with honest keys at forgetful positions, the bitstring interpretation of those keys are not used to compute the bitstring corresponding to t. We can then follow the strategy from the proof for Theorem 4 and replace honest keys in the library with random bitstrings.

Game 0.

In Game 0 $\mathcal{A}$ plays the original deduction soundness game $\mathsf{DS}_{(\mathcal{M} \cup \mathcal{M}_{\mathsf{SKE}}) \cup \mathcal{M}_{\mathsf{tran}}(\nu), (\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu)}(\eta)$.

Game 1.

In Game 1 we replace the implementation $\mathcal{I}$ with a corresponding oblivious implementation $\overline{\mathcal{I}}$ (which exists since $\mathcal{I}$ is a forgetful implementation according to Definition 9). Note that $\overline{\mathcal{I}}$ must be composable with $\mathcal{I}_{\mathsf{SKE}}$ since $\mathcal{I}$ is composable with $\mathcal{I}_{\mathsf{SKE}}$. For this to work we also have to replace the generate function by $\mathsf{generate}^{\mathsf{FIN}}$ from Figure 14.

Claim: Game 0 and Game 1 are indistinguishable.

Basically, this indistinguishability holds due to the fact that $\mathcal{I}$ is a forgetful implementation. Let $\mathcal{A}$ be a distinguisher between Game 0 and Game 1. Then we construct an adversary $\mathcal{B}$ that plays the game

$$\mathsf{FIN}^{b}_{\mathcal{M} \cup \mathcal{M}'_{\mathsf{tran}}(\nu'), \mathcal{I} \cup \mathcal{I}'_{\mathsf{tran}}(\nu'), \overline{\mathcal{I}} \cup \mathcal{I}'_{\mathsf{tran}}(\nu'), \mathcal{B}}(\eta)$$

and simulates Game 0 or Game 1 for $\mathcal{A}$ (depending on the value of b). $\mathcal{B}$ simulates $\mathcal{I}_{\mathsf{SKE}}$ using transparent functions (as a part of $\mathcal{M}'_{\mathsf{tran}}(\nu')$ together with $\mathcal{M}_{\mathsf{tran}}(\nu')$. $\mathcal{B}$ checks the DY-ness of $\mathcal{A}$'s requests with respect to Game 1. Note that the simulation is perfect since $\mathcal{B}$ can know all generate all the keys and does not need to hide any arguments when simulating $\mathcal{I}_{\mathsf{SKE}}$ with transparent functions. If $\mathcal{A}$ can distinguish Game 0 from Game 1, $\mathcal{B}$ can break the indistinguishability of the oblivious implementation according to Definition 9. This can only happen with negligible probability.

Game 2.

In Game 2 we replace the $\mathsf{generate}^{\mathsf{FIN}}$ function with a collision-aware variant (similar to Figure 3. The indistinguishability is guaranteed analogously to Theorem 4.

Game 3.

Game 3 is analogous to Game 2 from Theorem 4: We replace honest encryptions under honest keys by encryptions of 0 and replace honest encryption keys in the library by random bitstrings. Note that we need that fact that we replaced $\mathcal{I}$ by $\overline{\mathcal{I}}$ here: The oblivious implementation guarantees that the bitstrings representing honest keys are not used for the generation of other terms (in particular this is interesting when honest keys appear at forgetful positions). Hence we can replace them with random bitstrings and still have an indistinguishable game. The rest of the indistinguishability argument is based on the IND-CCA security of the SKE scheme and analogous to Theorem 4.

Game 4.

In Game 4, analogously to Game 3 from Theorem 4, we show that the adversary cannot win by producing encryptions under honest keys. To show the indistinguishability of Game 4 and Game 3 we use the same arguments for "reconstructions" and "forgeries" as in Theorem 3. Note that we simulate $\mathcal{I}_{\mathsf{SKE}}$ using transparent functions within this process. Here, we need the requirement that valid predicate of $\mathcal{I}$ does not depend on function symbols from Σ_{SKE}. Without this, we couldn't replace the function symbols from Σ_{SKE} with their transparent counterparts and still expect to have a valid trace when we are playing the deduction soundness game for $\mathcal{I}$ in the simulation.

Game 5.

Finally, analogously to Game 4 from Theorem 4, the simulator $\mathcal{B}$ plays the variation of the deduction soundness game for $\overline{\mathcal{I}}$ which it cannot win with non-negligible probability by Lemma 5. □

Let $\overline{\mathcal{M}_{\mathsf{SKE}}}$ be the forgetful symbolic model based on $\mathcal{M}_{\mathsf{SKE}}$ when we mark the message m for honestly generated encryptions under honest keys $E^{\hat{l}}_h(k^l_h(), m$ as a forgetful position and pick $\mathcal{I}_{\mathsf{SKE}}[\Sigma]$ as an implementation of $\overline{\mathcal{M}_{\mathsf{SKE}}}$. Then the following holds:

THEOREM 8. *Let $\mathcal{M}$ be a forgetful symbolic model and $\mathcal{I}$ be a forgetful deduction sound implementation of $\mathcal{M}$. $\mathcal{I}_{\mathsf{SKE}}$ denotes $\mathcal{I}_{\mathsf{SKE}}[\Sigma]$ where Σ is the signature from $\mathcal{M}$. If $(\overline{\mathcal{M}_{\mathsf{SKE}}}, \mathcal{I}_{\mathsf{SKE}})$ and $(\mathcal{M}, \mathcal{I})$ are compatible (see requirements in Section 6), then $\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}$ is a forgetful implementation of $\mathcal{M} \cup \overline{\mathcal{M}_{\mathsf{SKE}}}$*

PROOF. We pick the obvious oblivious implementation $\overline{\mathcal{I}_{\mathsf{SKE}}}$ for $\mathcal{I}_{\mathsf{SKE}}$ and set $(\overline{M_{\mathsf{SKE}}}\ E_h)(k, ell; r) := (M_{\mathsf{SKE}}\ E_h)(k, 0^\ell; r)$ and proof the theorem with a sequence of games:

Game 0.

Game 0 is the game

$$\mathsf{FIN}^{1}_{(\mathcal{M} \cup \overline{\mathcal{M}_{\mathsf{SKE}}}) \cup \mathcal{M}_{\mathsf{tran}}(\nu), (\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu), (\overline{\mathcal{I}} \cup \overline{\mathcal{I}_{\mathsf{SKE}}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu), \mathcal{A}}(\eta)$$

Game 1.

In Game 1 we replace the implementation $\mathcal{I}$ with a corresponding oblivious implementation $\overline{\mathcal{I}}$ (which exists since $\mathcal{I}$ is a forgetful implementation according to Definition 9). We can do this analogously to Game 1 from Theorem 7 and the indistinguishability of Game 0 and Game 1 holds for the same reasons. Game 1 is

$$\mathsf{FIN}^{0}_{(\mathcal{M} \cup \overline{\mathcal{M}_{\mathsf{SKE}}}) \cup \mathcal{M}_{\mathsf{tran}}(\nu), (\mathcal{I} \cup \mathcal{I}_{\mathsf{SKE}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu), (\overline{\mathcal{I}} \cup \mathcal{I}_{\mathsf{SKE}}) \cup \mathcal{I}_{\mathsf{tran}}(\nu), \mathcal{A}}(\eta)$$

Game 2.

In Game 2 we replace $\mathcal{I}_{\mathsf{SKE}}$ by $\overline{\mathcal{I}_{\mathsf{SKE}}}$ and the honest keys in the library by random values. We have indistinguishability

of Game 1 and Game 2 by the IND-CCA security of the SKE scheme. Game 2 is indistinguishable[6] from

$$\mathsf{FIN}^0_{(\mathcal{M}\cup\overline{\mathcal{M}_{\mathsf{SKE}}})\cup\mathcal{M}_{\mathsf{tran}}(\nu),(\mathcal{I}\cup\mathcal{I}_{\mathsf{SKE}})\cup\mathcal{I}_{\mathsf{tran}}(\nu),(\overline{\mathcal{I}\cup\mathcal{I}_{\mathsf{SKE}}})\cup\mathcal{I}_{\mathsf{tran}}(\nu),\mathcal{A}}(\eta)$$

In conclusion, Game 0 and Game 2 are indistinguishable. Hence $\mathcal{I}\cup\mathcal{I}_{\mathsf{SKE}}$ is a forgetful implementation of $\mathcal{M}\cup\overline{\mathcal{M}_{\mathsf{SKE}}}$. □

10. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Research Council under the European Unionâs Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement number 258865 (ERC ProSecure project). Florian Böhl was supported by MWK grant “MoSeS”.

11. REFERENCES

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Proc. 1st IFIP International Conference on Theoretical Computer Science (IFIP–TCS'00)*, volume 1872 of *LNCS*, pages 3–22, 2000.

[2] M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Science Foundations Workshop (CSFW'04)*, pages 204–218, 2004.

[3] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. of 10th ACM Conference on Computer and Communications Security (CCS'05)*, pages 220 – 230, 2003.

[4] M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within simulatable cryptographic library. In *Proc. 8th European Symposium on Research in Computer Security (ESORICS'03)*, Lecture Notes in Computer Science, pages 271–290, 2003.

[5] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, editors, *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[6] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*, Alexandria, Virginia, USA, Oct. 2008. ACM Press.

[7] V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In *Proceedings of the 26th Conference on Fundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *LNCS*, pages 176–187, Kolkata, India, 2006. Springer.

[8] V. Cortier, S. Kremer, and B. Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *J. Autom. Reasoning*, 46(3-4):225–259, 2011.

[6]Note that we only have indistinguishability here due to the random values for honest keys in the library.

[9] V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *European Symposium on Programming (ESOP'05)*, volume 3444 of *LNCS*, pages 157–171, Edinburgh, UK, 2005. Springer.

[10] V. Cortier and B. Warinschi. A composable computational soundness notion. In *18th ACM Conference on Computer and Communications Security (CCS'11)*, pages 63–74, Chicago, USA, October 2011. ACM.

[11] A. Datta, A. Derek, J. C. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic Polynomial-time Semantics for a Protocol Security Logic. In *Proc. of 32nd International Colloquium on Automata, Languages and Programming, ICALP*, volume 3580 of *LNCS*, pages 16–29. Springer, 2005. Lisboa, Portugal.

[12] Y. Dodis, S. Goldwasser, Y. T. Kalai, C. Peikert, and V. Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In D. Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 361–381. Springer, 2010.

[13] F. D. Garcia and P. van Rossum. Sound and complete computational interpretation of symbolic hashes in the standard model. *Theoretical Computer Science*, 394:112–133, 2008.

[14] R. Janvier, Y. Lakhnech, and L. Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *European Symposium on Programming (ESOP'05)*, volume 3444 of *LNCS*, pages 172–185. Springer, 2005.

[15] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. *SIAM J. Comput.*, 41(4):772–814, 2012.