Symbolic Encryption with Pseudorandom Keys*

Daniele Micciancio[1] [0000−0003−3323−9985]

University of California, San Diego, Mail Code 0404, La Jolla, CA 92093, USA. : daniele@cs.ucsd.edu
http://cseweb.ucsd.edu/~daniele/

Abstract. We give an efficient decision procedure that, on input two (acyclic) expressions making arbitrary use of common cryptographic primitives (namely, encryption and pseudorandom generators), determines (in polynomial time) if the two expressions produce computationally indistinguishable distributions for any cryptographic instantiation satisfying the standard security notions of pseudorandomness and indistinguishability under chosen plaintext attack. The procedure works by mapping each expression to a symbolic pattern that captures, in a fully abstract way, the information revealed by the expression to a computationally bounded observer. Our main result shows that if two expressions are mapped to different symbolic patterns, then there are secure pseudorandom generators and encryption schemes for which the two distributions can be distinguished with overwhelming advantage. At the same time if any two (acyclic) expressions are mapped to the same pattern, then the associated distributions are indistinguishable.

Keywords: Symbolic security · greatest fixed points · computational soundness · completeness · pseudorandom generators · information leakage.

1 Introduction

Formal methods for security analysis (e.g., [13,9,21,33,34,1]) typically adopt an all-or-nothing approach to modeling adversarial knowledge. For example, the adversary either knows a secret key or does not have any partial information about it. Similarly, either the message underlying a given ciphertext can be recovered, or it is completely hidden. In the computational setting, commonly used in modern cryptography for its strong security guarantees, the situation is much different: cryptographic primitives usually leak partial information about their inputs, and in many cases this cannot be avoided. Moreover, it is well known that computational cryptographic primitives, if not used properly, can easily lead to situations where individually harmless pieces of partial information can be combined to recover a secret in full. This is often the case when, for example, the same key or randomness is used within different cryptographic primitives.

* Research supported in part by NSF under grant CNS-1528068. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Starting with the seminal work of Abadi and Rogaway [3], there has been considerable progress in combining the symbolic and computational approaches to security protocol design and analysis, with the goal of developing methods that are both easy to apply (e.g., through the use of automatic verification tools) and provide strong security guarantees, as offered by the computational security definitions. Still, most work in this area applies to scenarios where the use of cryptography is sufficiently restricted that the partial information leakage of computational cryptographic primitives is inconsequential. For example, [3] studies expressions that use a single encryption scheme as their only cryptographic primitive. In this setting, the partial information about a key k revealed by a ciphertext $\{\!|m|\!\}_k$ is of no use to an adversary (except, possibly, for identifying when two different ciphertexts are encrypted under the same, unknown, key), so one can treat k as if it were completely hidden. Other works [28,4] combine encryption with other cryptographic primitives (like pseudorandom generation and secret sharing,) but bypass the problem of partial information leakage simply by assuming that all protocols satisfy sufficiently strong syntactic restrictions to guarantee that different cryptographic primitives do not interfere with each other.

1.1 Our results.

In this paper we consider cryptographic expressions that make arbitrary (nested) use of encryption and pseudorandom generation, without imposing any syntactic restrictions on the messages transmitted by the protocols. In particular, following [3], we consider cryptographic expressions like $(\{\!|m|\!\}_k, \{\!|\{\!|k|\!\}_{k'}|\!\}_{k''})$, representing a pair of ciphertexts: the encryption of a message m under a session key k, and a double (nested) encryption of the session key k under two other keys k', k''. But, while in [3] key symbols represent independent randomly chosen keys, here we allow for derived keys obtained using a length doubling pseudorandom generator $k \mapsto \mathbb{G}_0(k); \mathbb{G}_1(k)$ that on input a single key k outputs a pair of (statistically correlated, but computationally indistinguishable) keys $\mathbb{G}_0(k)$ and $\mathbb{G}_1(k)$. The output of the pseudorandom generator can be used anywhere a key is allowed. In particular, pseudorandom keys $\mathbb{G}_0(k)$, $\mathbb{G}_1(k)$ can be used to encrypt messages, or as messages themselves (possibly encrypted under other random or pseudorandom keys), or as input to the pseudorandom generator. So, for example, one can iterate the application of the pseudorandom generator to produce an arbitrary long sequence of keys $\mathbb{G}_1(r), \mathbb{G}_1(\mathbb{G}_0(r)), \mathbb{G}_1(\mathbb{G}_0(\mathbb{G}_0(r))), \ldots$.

We remark that key expansion using pseudorandom generators occurs quite often in real world cryptography. In fact, the usefulness of pseudorandom generators is not limited to reducing the amount of randomness needed by cryptographic algorithms, and pseudorandom generators are often used as an essential tool in secure protocol design. For example, they are used in the design of *forward-secure* cryptographic functions to refresh a user private key [7,25], they are used in the best known (in fact, optimal [30]) multicast key distribution protocols [10] to compactly communicate (using a seed) a long sequence of pseudorandom keys, and they play an important role in Yao's classic garbled circuit

construction for secure two-party computation to mask and selectively open part of a hidden circuit evaluation [35,24].

Pseudorandom generators (like any deterministic cryptographic primitive) inevitably leak partial information about their input key.[1] Similarly, a ciphertext $\{\!|e|\!\}_k$ may leak partial information about k if, for example, decryption succeeds (with high probability) only when the right key is used for decryption. As we consider the unrestricted use of encryption and pseudorandom generation, we need to model the possibility that given different pieces of partial information about a key, an adversary may be able to recover that key completely. Our main result shows how to do all this within a fairly simple symbolic model of computation, and still obtain strong computational soundness guarantees. Our treatment of partial information is extremely simple and in line with the spirit of formal methods and symbolic security analysis: we postulate that, given any two distinct pieces of partial information about a key, an adversary can recover the key in full. Perhaps not surprisingly, we demonstrate (Theorem 3) that the resulting symbolic semantics for cryptographic expressions is computationally sound, in the sense that if two (acyclic[2]) expressions are symbolically equivalent, then for any (length regular) semantically secure encryption scheme and (length doubling) pseudorandom generator the probability distributions naturally associated to the two expressions are computationally indistinguishable. More interestingly, we justify our symbolic model by proving a corresponding completeness theorem (Theorem 2), showing that if two cryptographic expressions are not symbolically equivalent (according to our definition), then there is an instantiation of the cryptographic primitives (satisfying the standard security notion of indistinguishability) such that the probability distributions corresponding to the two expressions can be efficiently distinguished with almost perfect advantage. In other words, if we want the symbolic semantics to be computationally sound with respect to any standard implementation of the cryptographic primitives, then our computationally sound symbolic semantics is essentially optimal. Moreover, our completeness theorem concretely shows what could go wrong when encrypting messages under related keys, even under a simple eavesdropping (passive) attack.

1.2 Techniques

A key technical contribution of our paper is a syntactic characterization of independent keys that exactly matches its computational counterpart, and a corresponding notion of computationally sound key renaming (Corollary 1). Our syntactic definition of independence is simple and intuitive: a set of keys $k_1, \ldots, k_n$

[1] For example, $\mathbb{G}_0(k)$ gives partial information about k because it allows to distinguish k from any other key k' chosen independently at random: all that the distinguisher has to do is to compute $\mathbb{G}_0(k')$ and compare the result to $\mathbb{G}_0(k)$.

[2] For cyclic expressions, i.e., expressions containing encryption cycles, our soundness theorem still holds, but with respect to a slightly stronger "co-inductive" adversarial model based on greatest fixed point computations [26].

is symbolically independent if no key k_i can be obtained from another k_j via the syntactic application of the pseudorandom generator. We show that this simple definition perfectly captures the intuition behind the computational notion of pseudorandomness: we prove (Theorem 1) that our definition is both computationally sound and complete, in the sense that the keys $k_1, \ldots, k_n$ are symbolically independent *if and only if* the associated probability distribution is indistinguishable from a sequence of truly independent uniformly random keys. For example, although the probability distributions associated to pseudorandom keys $\mathbb{G}_0(k)$ and $\mathbb{G}_1(k)$ are not independent in a strict information theoretic sense, the dependency between these distributions cannot be efficiently recognized when k is not known because the joint distribution associated to the pair $(\mathbb{G}_0(k), \mathbb{G}_1(k))$ is indistinguishable from a pair of independent random values.

A key component of our completeness theorem is a technical construction of a secure pseudorandom generator $\mathbb{G}$ and encryption scheme $\{\!|\cdot|\!\}_k$ satisfying some very special properties (Lemma 4) that may be of independent interest. The properties are best described in terms of pseudorandom functions. Let f_k be the pseudorandom function obtained from the length-doubling pseudorandom generator $\mathbb{G}$ using the classic construction of [17]. We give an algorithm that on input any string w and two ciphertexts $c_0 = \{\!|m_0|\!\}_{k_0}$ and $c_1 = \{\!|m_1|\!\}_{k_1}$ (for arbitrarily chosen, and unknown messages m_0, m_1) determines if $k_1 = f_{k_0}(w)$, and, if so, completely recovers the value of the keys k_0 and k_1 with overwhelming probability. Building on this lemma, we define the symbolic semantics by means of an abstract adversary that is granted the ability to recover the keys k_0, k_1 whenever it observes two ciphertext encrypted under them. Our completeness theorem offers a precise technical justification for such strong symbolic adversary.

1.3 Active attacks and other cryptographic primitives

Our work focuses on security definitions with respect to passive attacks for two reasons. First, indistinguishability is essentially[3] the only notion of security applicable to primitives as simple as pseudorandom generators. Second, using passive security definitions only makes our main result (Theorem 2) stronger: our completeness theorem shows that if two expressions map to different symbolic patterns, then security can be completely subverted even under a simple eavesdropping attack. Still, we remark that our definitions and techniques could be useful also for the analysis of security under more realistic attacks in the presence of active adversaries, e.g., if combined together with other soundness results [5,32,6,11,19]. Also, our results immediately extend to other cryptographic primitives (e.g., non-interactive commitment schemes) which can be modeled as a weakening of public key encryption. Possible extension to other cryptographic primitives, e.g., using the notion of *deduction soundness* [12,8] is also an inter-

[3] Active attacks against pseudorandom generators may be considered in the context of leakage resilient cryptography, fault injection analysis, and other side-channel attacks, which are certainly interesting, but also much more specialized models than those considered in this paper.

esting possibility. However, such extensions are outside the scope of this paper, and they are left to future work.

1.4 Related work.

Cryptographic expressions with pseudorandom keys, as those considered in this paper, are used in the symbolic analysis of various cryptographic protocols, including multicast key distribution [30,28,29], cryptographically controlled access to XML documents [4], and (very recently) the symbolic analysis of Yao's garbled circuit construction for secure two party computation [24]. However, these works (with the exception of [24], which builds on the results from a preliminary version of our paper [27]) use ad-hoc methods to deal with pseudorandom keys by imposing syntactic restrictions on the way the keys are used. Even more general (so called "composed") encryption keys are considered in [23], but only under the random oracle heuristics. We remark that the use of such general composed keys is unjustified in the standard model of computation, and the significance of the results of [23] outside the random oracle model is unclear. In fact, our completeness results clearly show that modeling key expansion as new random keys is not sound with respect to computationally secure pseudorandom generators in the standard model.

The problem of defining a computationally sound and complete symbolic semantics for cryptographic expressions has already been studied in several papers before, e.g., [3,31,14]. However, to the best of our knowledge, our is the first paper to prove soundness and completeness results with respect to the standard notion of computationally secure encryption [18]. In the pioneering work [3], Abadi and Rogaway proved the first soundness theorem for basic cryptographic expressions. Although in their work they mention various notions of security, they focus on a (somehow unrealistic) variant of the standard security definition that requires the encryption scheme to completely hide both the key and the message being encrypted, including its length. This is the notion of security used in many other works, including [22]. The issue of completeness was first raised by Micciancio and Warinschi [31] who proved that the logic of Abadi and Rogaway is both sound and complete if one assumes the encryption scheme satisfies a stronger security property called confusion freeness (independently defined also in [2], and subsequently weakened in [14]). We remark that most symbolic models are trivially complete for trace properties. However, the same is not true for indistinguishability security properties.

The notion of completeness used in [31,2,14] is different from the one studied in this paper. The works [31,2,14] consider restricted classes of encryption schemes (satisfying stronger security properties) such that the computational equivalence relation induced on expressions is the same for all encryption schemes in the class. In other words, if two expressions can be proved not equivalent within the logic framework, then the probability distributions associated to the two expressions by evaluating them according to *any* encryption scheme (from the given class) are computationally distinguishable. It can be shown that no

such notion of completeness can be achieved by the standard security definition of indistinguishability under chosen plaintext attack, as considered in this paper, i.e., different encryption schemes (all satisfying this standard notion of security) can define different equivalence relations. In this paper we use a different approach: instead of strengthening the computational security definitions to match the symbolic model of [3], we relax the symbolic model in order to match the standard computational security definition of [18]. Our relaxed symbolic model is still complete, in the sense that if two expressions evaluate to computationally equivalent distributions for any encryption scheme satisfying the standard security definition, then the equality between the two expressions can be proved within the logic. In other words, if two expressions are not equivalent in our symbolic model, then the associated probability distributions are not computationally equivalent for some (but not necessarily all) encryption scheme satisfying the standard computational security notion.

1.5 Organization.

The rest of the paper is organized as follows. In Section 2 we review basic notions from symbolic and computational cryptography as used in this paper. In Section 3 we present our basic results on the computational soundness of pseudorandom keys, and introduce an appropriate notion of key renaming. In Section 4 we present our symbolic semantics for cryptographic expressions with pseudorandom keys. In Section 5, we present our main result: a completeness theorem which justifies the definitional choices made in Section 4. A corresponding soundness theorem is given in Section 6. Section 7 concludes the paper with some closing remarks.

2 Preliminaries

In this section we review standard notions and notation from symbolic and computational cryptography used in the rest of the paper. The reader is referred to [3,26] for more background on the symbolic model, and [15,16,20] (or any other modern cryptography textbook) for more information about the computational model, cryptographic primitives and their security definitions.

We write $\{0,1\}^*$ to denote the set of all binary strings, $\{0,1\}^n$ for the set of all strings of length n, $|x|$ for the bitlength of a string x, ϵ for the empty string, and ";" (or simple juxtaposition) for the string concatenation operation mapping $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$ to $x;y \in \{0,1\}^{n+m}$. We also write $x \preceq y$ if x is a *suffix* of y, i.e., $y = zx$ for some $z \in \{0,1\}^*$. As usual, $x \prec y$ is $x \preceq y$ and $x \neq y$. The powerset of a set A is denoted $\wp(A)$.

As a general convention, we use bold uppercase names ($\mathbf{Exp}, \mathbf{Pat}$, etc.) for standard sets of symbolic expressions, bold lowercase names ($\mathbf{keys}, \mathbf{parts}$) for functions that return sets of symbolic expressions, and regular (non-bold) names ($\mathtt{shape}, \mathtt{norm}$) for functions returning a single symbolic expression. We also use uppercase letters (e.g., A, S) for set-valued variables, and lowercase letters (x, y)

for other variables. Calligraphic letters ($\mathcal{A}, \mathcal{G}, \mathcal{E}$, etc.) are reserved for probability distributions and algorithms in the computational setting.

2.1 Symbolic cryptography

In the symbolic setting, messages are described by abstract terms. For any given sets of key and data terms **Keys**, **Data**, define **Exp** as the set of cryptographic expressions generated by the grammar

$$\mathbf{Exp} ::= \mathbf{Data} \mid \mathbf{Keys} \mid (\mathbf{Exp}, \mathbf{Exp}) \mid \{\!|\mathbf{Exp}|\!\}_{\mathbf{Keys}}, \tag{1}$$

where (e_1, e_2) denotes the ordered pair of subexpressions e_1 and e_2, and $\{\!|e|\!\}_k$ denotes the encryption of e under k. We write $\mathbf{Exp}[\mathbf{Keys}, \mathbf{Data}]$ (and, similarly, for patterns $\mathbf{Pat}[\mathbf{Keys}, \mathbf{Data}]$ later on) to emphasize that the definition of **Exp** depends on the underlying sets **Keys** and **Data**. As a notational convention, we assume that the pairing operation is right associative, and omit unnecessary parentheses. E.g., we write $\{\!|d_1, d_2, d_3|\!\}_k$ instead of $\{\!|(d_1, (d_2, d_3))|\!\}_k$. All ciphertexts in our symbolic expressions represent independent encryptions (each using fresh randomness in the computational setting), even when carrying the same message. This is so that an adversary cannot distinguish between, say, $(\{\!|0|\!\}_k, \{\!|0|\!\}_k)$ and $(\{\!|0|\!\}_k, \{\!|1|\!\}_k)$. Sometimes (e.g., when adding an equality predicate "$\mathbf{Exp} = \mathbf{Exp}$" to the language of expressions) it is desirable for equality of symbolic terms to correspond to equality of their computational interpretations. This can be easily achieved by decorating symbolic ciphertexts with a "randomness" tag, so that identical expressions $\{\!|m|\!\}_k^r = \{\!|m|\!\}_k^r$ correspond to identical ciphertexts, while independent encryptions (of possibly identical messages) are represented by different symbolic expressions $\{\!|m|\!\}_k^r \neq \{\!|m|\!\}_k^{r'}$. An alternative (and syntactically cleaner) method to represent identical ciphertexts is to extend the symbolic syntax with a *variable assignment* operation, like

$$\texttt{let } c := \{\!|\mathbf{Exp}|\!\}_{\mathbf{Keys}} \texttt{ in } \mathbf{Exp}',$$

where the bound variable c may appear (multiple times) in the second expression $\mathbf{Exp}'$. Here, each "let" expression implicitly encrypts using independent randomness, and identical ciphertexts are represented using bound variables. All our definitions and results are easily adapted to these extended expressions with explicit randomness tags or bound variables.

In [3,26], $\mathbf{Keys} = \{k_1, \ldots, k_n\}$ and $\mathbf{Data} = \{d_1, \ldots, d_n\}$ are two flat sets of atomic keys and data blocks. In this paper, we consider pseudorandom keys, defined according to the grammar

$$\mathbf{Keys} ::= \mathbf{Rand} \mid \mathbb{G}_0(\mathbf{Keys}) \mid \mathbb{G}_1(\mathbf{Keys}), \tag{2}$$

where $\mathbf{Rand} = \{r_1, r_2, \ldots\}$ is a set of atomic key symbols (modeling truly random and independent keys), and $\mathbb{G}_0$, $\mathbb{G}_1$ represent the left and right half of a

length doubling pseudorandom generator $k \mapsto \mathbb{G}_0(k); \mathbb{G}_1(k)$. Notice that grammar (2) allows for the iterated application of the pseudorandom generator, so that from any key $r \in \mathbf{Rand}$, one can obtain keys of the form

$$\mathbb{G}_{b_1}(\mathbb{G}_{b_2}(\ldots(\mathbb{G}_{b_n}(r))\ldots))$$

for any $n \geq 0$, which we abbreviate as $\mathbb{G}_{b_1 b_2 \ldots b_n}(r)$. (As a special case, for $n = 0$, $\mathbb{G}_\epsilon(r) = r$.) For any set of keys $S \subseteq \mathbf{Keys}$, we write $\mathbb{G}^*(S)$ and $\mathbb{G}^+(S)$ to denote the sets

$$\mathbb{G}^*(S) = \{\mathbb{G}_w(k) \mid k \in S, w \in \{0,1\}^*\}$$
$$\mathbb{G}^+(S) = \{\mathbb{G}_w(k) \mid k \in S, w \in \{0,1\}^*, w \neq \epsilon\}$$

of keys which can be obtained from S through the repeated application of the pseudorandom generator functions $\mathbb{G}_0$ and $\mathbb{G}_1$, zero, one or more times. Using this notation, the set of keys generated by the grammar (2) can be written as $\mathbf{Keys} = \mathbb{G}^*(\mathbf{Rand})$. It is also convenient to define the set

$$\mathbb{G}^-(S) = \{k \mid \mathbb{G}^+(k) \cap S \neq \emptyset\} = \bigcup_{k' \in S} \{k \mid k' \in \mathbb{G}^+(k)\}.$$

Notice that, for any two keys k, k', we have $k \in \mathbb{G}^-(k')$ if and only if $k' \in \mathbb{G}^+(k)$, i.e., $\mathbb{G}^-$ corresponds to the inverse relation of $\mathbb{G}^+$.

The *shape* of an expression is obtained by replacing elements from **Data** and **Keys** with special symbols $\Box$ and $\circ$. Formally, shapes are defined as expressions over these dummy key/data symbols:

$$\mathbf{Shapes} = \mathbf{Exp}[\{\circ\}, \{\Box\}].$$

For notational simplicity, we omit the encryption keys $\circ$ in shapes and write $\{\!|s|\!\}$ instead of $\{\!|s|\!\}_\circ$. Shapes are used to model partial information (e.g., message size) that may be leaked by ciphertexts, even when the encrypting key is not known. (See Lemma 5 for a computational justification.)

The symbolic semantics of cryptographic expressions is defined by mapping them to *patterns*, which are expressions containing subterms of the form $\{\!|s|\!\}_k$, where $s \in \mathbf{Shapes}$ and $k \in \mathbf{Keys}$, representing undecryptable ciphertexts. Formally, the set of patterns $\mathbf{Pat}[\mathbf{Keys}, \mathbf{Data}]$ is defined as

$$\mathbf{Pat} ::= \mathbf{Data} \mid \mathbf{Keys} \mid (\mathbf{Pat}, \mathbf{Pat}) \mid \{\!|\mathbf{Pat}|\!\}_{\mathbf{Keys}} \mid \{\!|\mathbf{Shapes}|\!\}_{\mathbf{Keys}}. \quad (3)$$

Since expressions are also patterns, and patterns can be regarded as expressions over the extended sets $\mathbf{Keys} \cup \{\circ\}$, $\mathbf{Data} \cup \{\Box\}$, we use the letter e to denote expressions and patterns alike. We define a subterm relation $\sqsubseteq$ on $\mathbf{Pat}[\mathbf{Keys}, \mathbf{Data}]$ as the smallest reflexive transitive binary relation such that

$$e_1 \sqsubseteq (e_1, e_2), \qquad e_2 \sqsubseteq (e_1, e_2), \qquad \text{and} \qquad e \sqsubseteq \{\!|e|\!\}_k \quad (4)$$

for all $e, e_1, e_2 \in \mathbf{Pat}[\mathbf{Keys}, \mathbf{Data}]$ and $k \in \mathbf{Keys}$. The *parts* of a pattern $e \in \mathbf{Pat}$ are all of its subterms:

$$\mathbf{parts}(e) = \{e' \in \mathbf{Pat} \mid e' \sqsubseteq e\}. \quad (5)$$

The keys and shape of a pattern are defined by structural induction according to the obvious rules

$$\begin{array}{rlrl} \mathbf{keys}(d) &= \emptyset & \mathtt{shape}(d) &= \square \\ \mathbf{keys}(k) &= \{k\} & \mathtt{shape}(k) &= \circ \\ \mathbf{keys}(e_1, e_2) &= \mathbf{keys}(e_1) \cup \mathbf{keys}(e_2) & \mathtt{shape}(e_1, e_2) &= (\mathtt{shape}(e_1), \mathtt{shape}(e_2)) \\ \mathbf{keys}(\{\!|e|\!\}_k) &= \{k\} \cup \mathbf{keys}(e) & \mathtt{shape}(\{\!|e|\!\}_k) &= \{\!|\mathtt{shape}(e)|\!\} \end{array}$$

where $d \in \mathbf{Data}$, $k \in \mathbf{Keys}$, $e, e_1, e_2 \in \mathbf{Pat}[\mathbf{Keys}, \mathbf{Data}]$, and $\mathtt{shape}(s) = s$ for all shapes $s \in \mathbf{Shapes}$. Notice that, according to these definitions, $\mathbf{keys}(e)$ includes both the keys appearing in e as a message, and those appearing as an encryption key. On the other hand, $\mathbf{parts}(e)$ only includes the keys that are used as a message. As an abbreviation, we write

$$\mathbf{pkeys}(e) = \mathbf{parts}(e) \cap \mathbf{keys}(e)$$

for the set of keys that appear in e as a message. So, for example, if $e = (k, \{\!|0|\!\}_{k'}, \{\!|k''|\!\}_k)$ then $\mathbf{keys}(e) = \{k, k', k''\}$, but $\mathbf{pkeys}(e) = \{k, k''\}$. This is an important distinction to model the fact that an expression e only provides partial information about the keys in $\mathbf{keys}(e) \setminus \mathbf{parts}(e) = \{k'\}$.

2.2 Computational model

We assume that all algorithms and constructions take as an implicit input a (positive integer) security parameter ℓ, which we may think as fixed at the outset. We use calligraphic letters, $\mathcal{A}, \mathcal{B}$, etc., to denote randomized algorithms or the probability distributions defined by their output. We write $x \leftarrow \mathcal{A}$ for the operation of drawing x from a probability distribution $\mathcal{A}$, or running a probabilistic algorithm $\mathcal{A}$ with fresh randomness and output x. The uniform probability distribution over a finite set S is denoted by $\mathcal{U}(S)$, and we write $x \leftarrow S$ as an abbreviation for $x \leftarrow \mathcal{U}(S)$. Technically, since algorithms are implicitly parameterized by the security parameter ℓ, each $\mathcal{A}$ represents a *distribution ensemble*, i.e., a sequence of probability distributions $\{\mathcal{A}(\ell)\}_{\ell \geq 0}$ indexed by ℓ. For brevity, we will informally refer to probability ensembles $\mathcal{A}$ simply as probability distributions, thinking of the security parameter ℓ as fixed. We use standard asymptotic notation $O(f)$, $\omega(f)$, etc., and write $f \approx g$ if the function $\epsilon(\ell) = f(\ell) - g(\ell) = \ell^{-\omega(1)}$ is negligible. Two probability distributions $\mathcal{A}_0$ and $\mathcal{A}_1$ are *computationally indistinguishable* (written $\mathcal{A}_0 \approx \mathcal{A}_1$) if for any efficiently computable predicate $\mathcal{D}$, $\Pr\{\mathcal{D}(x) \colon x \leftarrow \mathcal{A}_0\} \approx \Pr\{\mathcal{D}(x) \colon x \leftarrow \mathcal{A}_1\}$.

Cryptographic Primitives In the computational setting, cryptographic expressions evaluate to probability distributions over binary strings, and two expressions are equivalent if the associated distributions are computationally indistinguishable. We consider cryptographic expressions that make use of two standard cryptographic primitives: pseudorandom generators, and (public or private key) encryption schemes.

A *pseudorandom generator* is an efficient algorithm $\mathcal{G}$ that on input a string $x \in \{0,1\}^\ell$ (the *seed*, of length equal to the security parameter ℓ) outputs a string $\mathcal{G}(x)$ of length bigger than ℓ, e.g., 2ℓ. We write $\mathcal{G}_0(x)$ and $\mathcal{G}_1(x)$ for the first and second half of the output of a (length doubling) pseudorandom generator, i.e., $\mathcal{G}(x) = \mathcal{G}_0(x); \mathcal{G}_1(x)$ with $|\mathcal{G}_0(x)| = |\mathcal{G}_1(x)| = |x| = \ell$. A pseudorandom generator $\mathcal{G}$ is computationally *secure* if the output distribution $\{\mathcal{G}(x) \colon x \leftarrow \{0,1\}^\ell\}$ is computationally indistinguishable from the uniform distribution $\mathcal{U}(\{0,1\}^{2\ell}) = \{y \colon y \leftarrow \{0,1\}^{2\ell}\}$.

A (private key) *encryption scheme* is a pair of efficient (randomized) algorithms $\mathcal{E}$ (for encryption) and $\mathcal{D}$ (for decryption) such that $\mathcal{D}(k, \mathcal{E}(k,m)) = m$ for any message m and key $k \in \{0,1\}^\ell$. The encryption scheme is *secure* if it satisfies the following definition of *indistinguishability under chosen plaintext attack*. More technically, for any probabilistic polynomial time adversary $\mathcal{A}$, the following must hold. Choose a bit $b \in \{0,1\}$ and a key $k \in \{0,1\}^\ell$ uniformly at random, and let $O_b(m)$ be an encryption oracle that on input a message m outputs $\mathcal{E}(k,m)$ if $b = 1$, or $\mathcal{E}(k, 0^{|m|})$ if $b = 0$, where $0^{|m|}$ is a sequence of 0s of the same length as m. The adversary $\mathcal{A}$ is given oracle access to $O_b(\cdot)$, and attempts to guess the bit b. The encryption scheme is secure if $\Pr\{\mathcal{A}^{O_b(\cdot)} = b\} \approx 1/2$. For notational convenience, the encryption $\mathcal{E}(k,m)$ of a message m under a key k is often written as $\mathcal{E}_k(m)$. Public key encryption is defined similarly. All our results hold for private and public key encryption algorithms, with hardly any difference in the proofs. So, for simplicity, we will focus the presentation on private key encryption, but we observe that adapting the results to public key encryption is straightforward.

In some of our proofs, it is convenient to use a seemingly stronger (but equivalent) security definition for encryption, where the adversary is given access to several encryption oracles, each encrypting under an independently chosen random key. More formally, the adversary $\mathcal{A}$ in the security definition is given access to a (stateful) oracle $O_b(i,m)$ that takes as input both a message m and a key index i. The first time $\mathcal{A}$ makes a query with a certain index i, the encryption oracle chooses a key $k_i \leftarrow \{0,1\}^\ell$ uniformly at random. The query $O_b(i,m)$ is answered using key k_i as in the previous definition: if $b = 1$ then $O_b(i,m) = \mathcal{E}(k_i, m)$, while if $b = 0$ then $O_b(i,m) = \mathcal{E}(k_i, 0^{|m|})$.

Computational evaluation. In order to map a cryptographic expression from **Exp** to a probability distribution, we need to pick a length doubling pseudorandom generator $\mathcal{G}$, a (private key) encryption scheme $\mathcal{E}$, a string representation γ_d for every data block $d \in \mathbf{Data}$, and a binary operation[4] π used to encode pairs of strings.

[4] We do not assume any specific property about π, other than invertibility and efficiency, i.e., $\pi(w_1, w_2)$ should be computable in polynomial (typically linear) time, and the substrings w_1 and w_2 can be uniquely recovered from $\pi(w_1, w_2)$, also in polynomial time. In particular, $\pi(w_1, w_2)$ is not just the string concatenation operation $w_1; w_2$ (which is not invertible), and the strings $\pi(w_1, w_2)$ and $\pi(w_2, w_1)$ may have different length. For example, $\pi(w_1, w_2)$ could be the string concatenation of a prefix-free encoding of w_1, followed by w_2.

Since encryption schemes do not hide the length of the message being encrypted, it is natural to require that all functions operating on messages are length-regular, i.e., the length of their output depends only on the length of their input. For example, $\mathcal{G}$ is length regular by definition, as it always maps strings of length ℓ to strings of length 2ℓ. Throughout the paper we assume that all keys have length ℓ equal to the security parameter, and the functions $d \mapsto \gamma_d$, π and $\mathcal{E}$ are length regular, i.e., $|\gamma_d|$ is the same for all $d \in \mathbf{Data}$, $|\pi(x_1, x_2)|$ depends only on $|x_1|$ and $|x_2|$, and $|\mathcal{E}(k, x)|$ depends only on ℓ and $|x|$.

Definition 1. *A computational interpretation is a tuple $(\mathcal{G}, \mathcal{E}, \gamma, \pi)$ consisting of a length-doubling pseudorandom generator $\mathcal{G}$, a length regular encryption scheme $\mathcal{E}$, and length regular functions γ_d and $\pi(x_1, x_2)$. If $\mathcal{G}$ is a secure pseudorandom generator, and $\mathcal{E}$ is a secure encryption scheme (satisfying indistinguishability under chosen plaintext attacks, as defined in the previous paragraphs), then we say that $(\mathcal{G}, \mathcal{E}, \gamma, \pi)$ is a* secure *computational interpretation.*

Computational interpretations are used to map symbolic expressions in **Exp** to probability distributions in the obvious way. We first define the evaluation $\sigma[\![e]\!]$ of an expression $e \in \mathbf{Exp}[\mathbf{Keys}, \mathbf{Data}]$ with respect to a fixed key assignment $\sigma\colon \mathbf{Keys} \to \{0,1\}^\ell$. The value $\sigma[\![e]\!]$ is defined by induction on the structure of the expression e by the rules $\sigma[\![d]\!] = \gamma_d$, $\sigma[\![k]\!] = \sigma(k)$, $\sigma[\![(e_1, e_2)]\!] = \pi(\sigma[\![e_1]\!], \sigma[\![e_2]\!])$, and $\sigma[\![\{\!|e|\!\}_k]\!] = \mathcal{E}(\sigma(k), \sigma[\![e]\!])$. All ciphertexts in a symbolic expressions are evaluated using fresh independent encryption randomness. The computational evaluation $[\![e]\!]$ of an expression e is defined as the probability distribution obtained by first choosing a random key assignment σ (as explained below) and then computing $\sigma[\![e]\!]$. When $\mathbf{Keys} = \mathbb{G}^*(\mathbf{Rand})$ is a set of pseudorandom keys, σ is selected by first choosing the values $\sigma(r) \in \{0,1\}^\ell$ (for $r \in \mathbf{Rand}$) independently and uniformly at random, and then extending σ to pseudorandom keys in $\mathbb{G}^+(\mathbf{Rand})$ using a length doubling pseudorandom generator $\mathcal{G}$ according to the rule

$$\mathcal{G}(\sigma(k)) = \sigma(\mathbb{G}_0(k)); \sigma(\mathbb{G}_1(k)).$$

It is easy to see that any two expressions $e, e' \in \mathbf{Exp}$ with the same shape $s = \mathtt{shape}(e) = \mathtt{shape}(e')$ always map to strings of exactly the same length, denoted $|[\![s]\!]| = |\sigma[\![e]\!]| = |\sigma'[\![e']\!]|$. The computational evaluation function $\sigma[\![e]\!]$ is extended to patterns by defining $\sigma[\![s]\!] = 0^{|[\![s]\!]|}$ for all shapes $s \in \mathbf{Shapes}$. Again, we have $|\sigma[\![e]\!]| = |[\![\mathtt{shape}(e)]\!]|$ for all patterns $e \in \mathbf{Pat}$, i.e., all patterns with the same shape evaluate to strings of the same length.

Notice that each expression e defines a probability ensemble $[\![e]\!]$, indexed by the security parameter ℓ defining the key length of $\mathcal{G}$ and $\mathcal{E}$. Two symbolic expressions (or patterns) e, e' are computationally equivalent (with respect to a given computational interpretation $(\mathcal{G}, \mathcal{E}, \gamma, \pi)$) if the corresponding probability ensembles $[\![e]\!]$ and $[\![e']\!]$ are computationally indistinguishable. An equivalence relation R on symbolic expressions is computationally *sound* if for any two equivalent expressions $(e, e') \in R$ and any secure computational interpretation, the distributions $[\![e]\!]$ and $[\![e']\!]$ are computationally indistinguishable. Conversely, we say that a relation R is *complete* if for any two unrelated expressions $(e, e') \notin R$,

there is a secure computational interpretation such that $[\![e]\!]$ and $[\![e']\!]$ can be efficiently distinguished.

3 Symbolic model for pseudorandom keys

In this section we develop a symbolic framework for the treatment of pseudorandom keys, and prove that it is computationally sound and complete. Specifically, we give a symbolic criterion for a set of keys which is satisfied *if and only if* the joint distribution associated to the set of keys is computationally indistinguishable from the uniform distribution. Before getting into the technical details we provide some intuition.

Symbolic keys are usually regarded as bound names, up to renaming. In the computational setting, this corresponds to the fact that changing the names of the keys does not alter the probability distribution associated to them. When pseudorandom keys are present, some care has to be exercised in defining an appropriate notion of key renaming. For example, swapping r and $\mathbb{G}_0(r)$ should not be considered a valid key renaming because the probability distributions associated to $(r, \mathbb{G}_0(r))$ and $(\mathbb{G}_0(r), r)$ can be easily distinguished.[5] A conservative approach would require a key renaming μ to act simply as a permutation over the set of atomic keys **Rand**. However, this is overly restrictive. For example, renaming $(\mathbb{G}_0(r), \mathbb{G}_1(r))$ to (r_0, r_1) should be allowed because $(\mathbb{G}_0(r), \mathbb{G}_1(r))$ represents a pseudorandom string, which is computationally indistinguishable from the truly random string given by (r_0, r_1). The goal of this section is to precisely characterize which key renamings can be allowed, and which cannot, to preserve computational indistinguishability.

The rest of the section is organized as follows. First, in Section 3.1, we introduce a symbolic notion of independence for pseudorandom keys. Informally, two (symbolic) keys are independent if neither of them can be derived from the other through the application of the pseudorandom generator. We give a computational justification for this notion by showing (see Theorem 1) that the standard (joint) probability distribution associated to a sequence of symbolic keys $k_1, \ldots, k_n \in \mathbf{Keys}$ in the computational model is pseudorandom precisely when the keys $k_1, \ldots, k_n$ are symbolically independent. Then, in Section 3.2, we use this definition of symbolic independence to define a computationally sound notion of key renaming. Intuitively, in order to be computationally sound and achieve other desirable properties, key renamings should map independent sets to independent sets. In Corollary 1 we prove that, under such restriction, applying a renaming to cryptographic expressions yields computationally *indistinguishable* distributions. This should be contrasted with the standard notion of key renaming used in the absence of pseudorandom keys, where equivalent expressions evaluate to *identical* probability distributions.

[5] All that the distinguisher has to do, on input a pair of keys (σ_0, σ_1), is to compute $\mathcal{G}_0(\sigma_1)$ and check if the result equals σ_0.

3.1 Independence

In this section we define a notion of independence for symbolic keys, and show that it is closely related to the computational notion of pseudorandomness.

Definition 2. *For any two keys $k_1, k_2 \in$ **Keys**, we say that k_1* yields *k_2 (written $k_1 \preceq k_2$) if $k_2 \in \mathbb{G}^*(k_1)$, i.e., k_2 can be obtained by repeated application of $\mathbb{G}_0$ and $\mathbb{G}_1$ to k_1. Two keys k_1, k_2 are* independent *(written $k_1 \perp k_2$) if neither $k_1 \preceq k_2$ nor $k_2 \preceq k_1$. We say that the keys $k_1, \ldots, k_n$ are independent if $k_i \perp k_j$ for all $i \neq j$.*

Notice that any two keys satisfy $\mathbb{G}_{w_0}(r_0) \preceq \mathbb{G}_{w_1}$ if and only if $r_0 = r_1$ and $w_0 \preceq w_1$. As an example, they keys $\mathbb{G}_0(r) \perp \mathbb{G}_{01}(r)$ are independent, but the keys $\mathbb{G}_0(r) \preceq \mathbb{G}_{10}(r)$ are not. As usual, we write $k_1 \prec k_2$ as an abbreviation for $(k_1 \preceq k_2) \wedge (k_1 \neq k_2)$. Notice that $(\mathbf{Keys}, \preceq)$ is a partial order, i.e., the relation $\preceq$ is reflexive, antisymmetric and transitive. Pictorially a set of keys $S \subseteq \mathbf{Keys}$ can be represented by the Hasse diagram[6] of the induced partial order $(S, \preceq)$. (See Figure 1 for an example.) Notice that this diagram is always a forest, i.e., the union of disjoint trees with roots $\mathbf{roots}(S) = S \setminus \mathbb{G}^+(S)$. S is an independent set if and only if $S = \mathbf{roots}(S)$, i.e., each tree in the forest associated to S consists of a single node, namely its root.

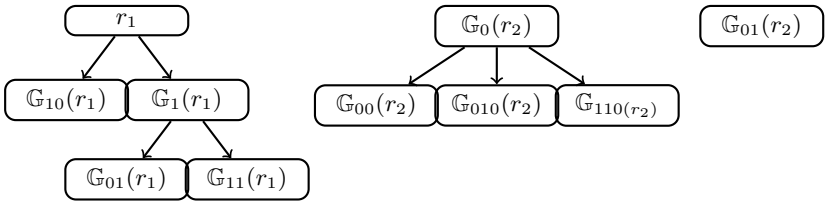

Fig. 1. Hasse diagram associated to the set of keys $S = \{r_1, \mathbb{G}_{10}(r_1), \mathbb{G}_1(r_1), \mathbb{G}_{01}(r_1), \mathbb{G}_{11}(r_1), \mathbb{G}_0(r_2), \mathbb{G}_{00}(r_2), \mathbb{G}_{010}(r_2), \mathbb{G}_{110}(r_2), \mathbb{G}_{01}(r_2)\}$. For any two keys, $k_1 \preceq k_2$ if there is a directed path from k_1 to k_2. The keys $\{\mathbb{G}_0(r_2), \mathbb{G}_{01}(r_2)\}$ form an independent set because neither $\mathbb{G}_0(r_2) \preceq \mathbb{G}_{01}(r_2)$, nor $\mathbb{G}_{01}(r_2) \preceq \mathbb{G}_0(r_2)$. The Hasse diagram of S is a forest consisting of 3 trees with roots $\mathbf{roots}(S) = \{r_1, \mathbb{G}_0(r_2), \mathbb{G}_{01}(r_2)\}$.

We consider the question of determining, symbolically, when (the computational evaluation of) a sequence of pseudorandom keys $k_1, \ldots, k_n$ is pseudorandom, i.e., it is computationally indistinguishable from n truly random independently chosen keys. The following lemma shows that our symbolic notion of independence corresponds exactly to the standard cryptographic notion of computational pseudorandomness. We remark that the correspondence proved

[6] The Hasse diagram of a partial order relation $\preceq$ is the graph associated to the transitive reduction of $\preceq$, i.e., the smallest relation R such that $\preceq$ is the symmetric transitive closure of R.

in the lemma is *exact*, in the sense that the symbolic condition is both *necessary* and *sufficient* for symbolic equivalence. This should be contrasted with typical computational soundness results [3], that only provide sufficient conditions for computational equivalence, and require additional work/assumptions to establish the completeness of the symbolic criterion [31,14].

Theorem 1. *Let $k_1, \ldots, k_n \in \mathbf{Keys} = \mathbb{G}^*(\mathbf{Rand})$ be a sequence of symbolic keys. Then, for any secure (length doubling) pseudorandom generator $\mathcal{G}$, the probability distributions $[\![k_1, \ldots, k_n]\!]$ and $[\![r_1, \ldots, r_n]\!]$ (where $r_1, \ldots, r_n \in \mathbf{Rand}$ are distinct atomic keys) are computationally indistinguishable if and only if the keys $k_1, \ldots, k_n$ are (symbolically) independent, i.e., $k_i \perp k_j$ for all $i \neq j$.*

Proof. We first prove the "only if" direction of the equivalence, i.e., independence is a necessary condition for the indistinguishability of $[\![r_1, \ldots, r_n]\!]$ and $[\![k_1, \ldots, k_n]\!]$. Assume the keys in $(k_1, \ldots, k_n)$ are not independent, i.e., $k_i \preceq k_j$ for some $i \neq j$. By definition, $k_j = \mathbb{G}_w(k_i)$ for some $w \in \{0,1\}^*$. This allows to deterministically compute $[\![k_j]\!] = \mathcal{G}_w([\![k_i]\!])$ from $[\![k_i]\!]$ using the pseudorandom generator. The distinguisher between $[\![r_1, \ldots, r_n]\!]$ and $[\![k_1, \ldots, k_n]\!]$ works in the obvious way: given a sample $(\sigma_1, \ldots, \sigma_n)$, compute $\mathcal{G}_w(\sigma_i)$ and compare the result to σ_j. If the sample comes from $[\![k_1, \ldots, k_n]\!]$, then the test is satisfied with probability 1. If the sample comes from $[\![r_1, \ldots, r_n]\!]$, then the test is satisfied with exponentially small probability because $\sigma_i = [\![r_i]\!]$ is chosen at random independently from $\sigma_j = [\![r_j]\!]$. This concludes the proof for the "only if" direction.

Let us now move to the "if" direction, i.e., prove that independence is a sufficient condition for the indistinguishability of $[\![r_1, \ldots, r_n]\!]$ and $[\![k_1, \ldots, k_n]\!]$. Assume the keys in $(k_1, \ldots, k_n)$ are independent, and let m be the number of applications of $\mathbb{G}_0$ and $\mathbb{G}_1$ required to obtain $(k_1, \ldots, k_n)$ from the basic keys in **Rand**. We define $m+1$ tuples $K^i = (k_1^i, \ldots, k_n^i)$ of independent keys such that

- $K^0 = (k_1, \ldots, k_n)$
- $K^m = (r_1, \ldots, r_n)$, and
- for all i, the distributions $[\![K^i]\!]$ and $[\![K^{i+1}]\!]$ are computationally indistinguishable.

It follows by transitivity that $[\![K^0]\!] = [\![k_1, \ldots, k_n]\!]$ is computationally indistinguishable from $[\![K^m]\!] = [\![r_1, \ldots, r_n]\!]$. More precisely, any adversary that distinguishes $[\![k_1, \ldots, k_n]\!]$ from $[\![r_1, \ldots, r_n]\!]$ with advantage δ, can be efficiently transformed into an adversary that breaks the pseudorandom generator $\mathcal{G}$ with advantage at least δ/m. Each tuple K^{i+1} is defined from the previous one K^i as follows. If all the keys in $K^i = \{k_1^i, \ldots, k_n^i\}$ are random (i.e., $k_j^i \in \mathbf{Rand}$ for all $j = 1, \ldots, n$), then we are done and we can set $K^{i+1} = K^i$. Otherwise, let $k_j^i = \mathbb{G}_w(r) \in \mathbf{Keys} \setminus \mathbf{Rand}$ be a pseudorandom key in K^i, with $r \in \mathbf{Rand}$ and $w \neq \epsilon$. Since the keys in K^i are independent, we have $r \notin K^i$. Let $r', r'' \in \mathbf{Rand}$ be two new fresh key symbols, and define $K^{i+1} = \{k_1^{i+1}, \ldots, k_n^{i+1}\}$ as follows:

$$k_h^{i+1} = \begin{cases} \mathbb{G}_s(r') & \text{if } k_h^i = \mathbb{G}_s(\mathbb{G}_0(r)) \text{ for some } s \in \{0,1\}^* \\ \mathbb{G}_s(r'') & \text{if } k_h^i = \mathbb{G}_s(\mathbb{G}_1(r)) \text{ for some } s \in \{0,1\}^* \\ k_h^i & \text{otherwise} \end{cases}$$

It remains to prove that any distinguisher $\mathcal{D}$ between $[\![K^i]\!]$ and $[\![K^{i+1}]\!]$ can be used to break (with the same success probability) the pseudorandom generator $\mathcal{G}$. The distinguisher $\mathcal{D}'$ for the pseudorandom generator $\mathcal{G}$ is given as input a pair of strings (σ', σ'') chosen either uniformly (and independently) at random or running the pseudorandom generator $(\sigma', \sigma'') = \mathcal{G}(\sigma)$ on a randomly chosen seed σ. $\mathcal{D}'(\sigma', \sigma'')$ computes n strings $(\sigma_1, \ldots, \sigma_n)$ by evaluating $(k_1^{i+1}, k_2^{i+1}, \ldots, k_n^{i+1})$ according to an assignment that maps r' to σ', r'' to σ'', and all other base keys $r \in \mathbf{Rand}$ to independent uniformly chosen values. The output of $\mathcal{D}'(\sigma', \sigma'')$ is $\mathcal{D}(\sigma_1, \ldots, \sigma_n)$. Notice that if σ' and σ'' are chosen uniformly and independently at random, then $(\sigma_1, \ldots, \sigma_n)$ is distributed according to $[\![K^{i+1}]\!]$, while if $(\sigma', \sigma'') = \mathcal{G}(\sigma)$, then $(\sigma_1, \ldots, \sigma_n)$ is distributed according to $[\![K^i]\!]$. Therefore the success probability of $\mathcal{D}'$ in breaking $\mathcal{G}$ is exactly the same as the success probability of $\mathcal{D}$ in distinguishing $[\![K^i]\!]$ from $[\![K^{i+1}]\!]$. □

3.2 Renaming pseudorandom keys

We will show that key renamings are compatible with computational indistinguishability as long as they preserve the action of the pseudorandom generator, in the sense specified by the following definition.

Definition 3 (pseudo-renaming). *For any set of keys $S \subseteq \mathbf{Keys}$, a renaming $\mu\colon S \to \mathbf{Keys}$ is compatible with the pseudorandom generator $\mathbb{G}$ if for all $k_1, k_2 \in S$ and $w \in \{0,1\}^*$,*

$$k_1 = \mathbb{G}_w(k_2) \qquad \text{if and only if} \qquad \mu(k_1) = \mathbb{G}_w(\mu(k_2)).$$

For brevity, we refer to renamings satisfying this property as pseudo-renamings.

Notice that the above definition does not require the domain of μ to be the set of all keys **Keys**, or even include all keys in **Rand**. So, for example, the function mapping $(\mathbb{G}_0(r_0), \mathbb{G}_1(r_0))$ to $(r_0, \mathbb{G}_{001}(r_1))$ is a valid pseudo-renaming, and it does not act as a permutation over **Rand**. The following lemmas show that Definition 3 is closely related to the notion of symbolic independence.

Lemma 1. *Let μ be a pseudo-renaming with domain $S \subseteq \mathbf{Keys}$. Then μ is a bijection from S to $\mu(S)$. Moreover, S is an independent set if and only if $\mu(S)$ is an independent set.*

Proof. Let $\mu\colon S \to \mathbf{Keys}$ be a pseudo-renaming. Then μ is necessarily injective, because for all $k_1, k_2 \in S$ such that $\mu(k_1) = \mu(k_2)$, we have $\mu(k_1) = \mu(k_2) = \mathbb{G}_\epsilon(\mu(k_2))$. By definition of pseudo-renaming, this implies $k_1 = \mathbb{G}_\epsilon(k_2) = k_2$. This proves that μ is a bijection from S to $\mu(S)$.

Now assume S is *not* an independent set, i.e., $k_1 = \mathbb{G}_w(k_2)$ for some $k_1, k_2 \in S$ and $w \neq \epsilon$. By definition of pseudo-renaming, we also have $\mu(k_1) = \mathbb{G}_w(\mu(k_2))$. So, $\mu(S)$ is not an independent set either. Similarly, if $\mu(S)$ is *not* an independent set, then there exists keys $\mu(k_1), \mu(k_2) \in \mu(S)$ (with $k_1, k_2 \in S$) such that $\mu(k_1) = \mathbb{G}_w(\mu(k_2))$ for some $w \neq \epsilon$. Again, by definition of pseudo-renaming, $k_1 = \mathbb{G}_w(k_2)$, and S is not an independent set. □

In fact, pseudo-renamings can be equivalently defined as the natural extension of bijections between two independent sets of keys.

Lemma 2. *Any pseudo-renaming μ with domain S can be uniquely extended to a pseudo-renaming $\bar{\mu}$ with domain $\mathbb{G}^*(S)$. In particular, any pseudo-renaming can be (uniquely) specified as the extension $\bar{\mu}$ of a bijection $\mu: A \to B$ between two independent sets $A = \mathbf{roots}(S)$ and $B = \mu(A)$.*

Proof. Let $\mu: S \to \mathbf{Keys}$ be a pseudo-renaming. For any $w \in \{0,1\}^*$ and $k \in S$, define $\bar{\mu}(\mathbb{G}_w(k)) = \mathbb{G}_w(\mu(k))$. This definition is well given because μ is a pseudo-renaming, and therefore for any two representations of the same key $\mathbb{G}_w(k) = \mathbb{G}_{w'}(k') \in \mathbb{G}^*(S)$ with $k, k' \in S$, we have $\mathbb{G}_w(\mu(k)) = \mu(\mathbb{G}_w(k)) = \mu(Gen_{w'}(k')) = \mathbb{G}_{w'}(\mu(k'))$. Moreover, it is easy to check that $\bar{\mu}$ is a pseudo-renaming, and any pseudo-renaming that extends μ must agree with $\bar{\mu}$. We now show that pseudo-renamings can be uniquely specified as bijections between two independent sets of keys. Specifically, for any pseudo-renaming μ with domain S, consider the restriction μ_0 of μ to $A = \mathbf{roots}(S)$. By Lemma 1, μ_0 is a bijection between independent sets A and $B = \mu_0(A)$. Consider the extensions of μ and μ_0 to $\mathbb{G}^*(S) = \mathbb{G}^*(\mathbf{roots}(S)) = \mathbb{G}^*(A)$. Since μ and μ_0 agree on $A = \mathbf{roots}(S)$, both $\bar{\mu}$ and $\bar{\mu}_0$ are extensions of μ_0. By uniqueness of this extension, we get $\bar{\mu}_0 = \bar{\mu}$. Restricting both functions to S, we get that the original pseudo-renaming μ can be expressed as the restriction of $\bar{\mu}_0$ to S. In other words, μ can be expressed as the extension to S of a bijection μ_0 between two independent sets of keys $A = \mathbf{roots}(S)$ and $B = \mu(A)$. □

We remark that a pseudo-renaming $\mu: S \to \mathbf{Keys}$ cannot, in general, be extended to one over the set $\mathbf{Keys} = \mathbb{G}^*(\mathbf{Rand})$ of all keys. For example, $\mu: \mathbb{G}_0(r_0) \mapsto r_1$ is a valid pseudo-renaming, but it cannot be extended to include r_0 in its domain.

The next lemma gives one more useful property of pseudo-renamings: they preserve the root keys.

Lemma 3. *For any pseudo-renaming $\mu: A \to \mathbf{Keys}$, we have $\mu(\mathbf{roots}(A)) = \mathbf{roots}(\mu(A))$.*

Proof. By Lemma 1, μ is injective. Therefore, $\mu(\mathbf{roots}(A))$ equals $\mu(A \setminus \mathbb{G}^+(A)) = \mu(A) \setminus \mu(\mathbb{G}^+(A))$. From the defining property of pseudo-renamings we also easily get that $\mu(\mathbb{G}^+(A)) = \mathbb{G}^+(\mu(A))$. Therefore, $\mu(\mathbf{roots}(A)) = \mu(A) \setminus \mathbb{G}^+(\mu(A)) = \mathbf{roots}(\mu(A))$. □

Using Lemma 2, throughout the paper we specify pseudo-renamings as bijections between two independent sets of keys. Of course, in order to apply $\mu: S \to \mu(S)$ to an expression e, the key set $\mathbf{keys}(e)$ must be contained in $\mathbb{G}^*(S)$. Whenever we apply a pseudo-renaming $\mu: S \to \mathbf{Keys}$ to an expression or pattern e, we implicitly assume that $\mathbf{keys}(e) \subset \mathbb{G}^*(S)$. (Typically, $S = \mathbf{roots}(\mathbf{keys}(e))$, so that $\mathbf{keys}(e) \subset \mathbb{G}^*(\mathbf{roots}(\mathbf{keys}(e))) = \mathbb{G}^*(S)$ is always satisfied.) Formally, the result of applying a pseudo-renaming μ to an expression

or pattern $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$ is defined as

$$\begin{aligned} \mu(d) &= d & \mu(\{\!|e|\!\}_k) &= \{\!|\mu(e)|\!\}_{\bar{\mu}(k)} \\ \mu(k) &= \bar{\mu}(k) & \mu(s) &= s \\ \mu(e_1, e_2) &= (\mu(e_1), \mu(e_2)) \end{aligned}$$

for all $d \in \mathbf{Data}$, $k \in \mathbf{Keys}$, $e, e_1, e_2 \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$ and $s \in \mathbf{Shapes}$. We can now define an appropriate notion of symbolic equivalence up to renaming.

Definition 4. *Two expressions or patterns $e_1, e_2 \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$ are* equivalent up to pseudo-renaming *(written $e_1 \cong e_2$), if there is a pseudo-renaming μ such that $\bar{\mu}(e_1) = e_2$. Equivalently, by Lemma 2, $e_1 \cong e_2$ if there is a bijection $\mu\colon \mathbf{roots}(\mathbf{keys}(e_1)) \to \mathbf{roots}(\mathbf{keys}(e_2))$ such that $\bar{\mu}(e_1) = e_2$.*

It easily follows from the definitions and Theorem 1 that $\cong$ is an equivalence relation, and expressions that are equivalent up to pseudo-renaming are computationally equivalent.

Corollary 1. *The equivalence relation $\cong$) is computationally sound, i.e., for any two patterns $e_1, e_2 \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$ such that $e_1 \cong e_2$, the distributions $[\![e_1]\!]$ and $[\![e_2]\!]$ are computationally indistinguishable.*

Proof. Assume $e_1 \cong e_2$, i.e., there exists a bijection $\mu : \mathbf{roots}(\mathbf{keys}(e_1)) \to \mathbf{roots}(\mathbf{keys}(e_2))$ such that $\bar{\mu}(e_1) = e_2$. Let n be the size of $A_1 = \mathbf{roots}(\mathbf{keys}(e_1))$ and $A_2 = \mathbf{roots}(\mathbf{keys}(e_2)) = \mu(A_1)$. We show that any distinguisher $\mathcal{D}$ between $[\![e_1]\!]$ and $[\![e_2]\!] = [\![\bar{\mu}(e_1)]\!]$ can be efficiently transformed into a distinguisher $\mathcal{A}$ between $[\![A_1]\!]$ and $[\![A_2]\!]$ with the same advantage as $\mathcal{D}$. Since A_1 and A_2 are independent sets of size n, by Theorem 1 the probability distributions $[\![A_1]\!]$ and $[\![A_2]\!]$ are indistinguishable from $[\![r_1, \ldots, r_n]\!]$. So, $[\![A_1]\!]$ and $[\![A_2]\!]$ must be indistinguishable from each other, and $\mathcal{A}$'s advantage must be negligible. We now show how to build $\mathcal{A}$ from $\mathcal{D}$. The distinguisher $\mathcal{A}$ takes as input a sample σ coming from either $[\![A_1]\!]$ or $[\![A_2]\!]$. $\mathcal{A}$ evaluates e_1 according to the key assignment $A_1 \mapsto \sigma$, and outputs $\mathcal{D}(\sigma[\![e_1]\!])$. By construction, $\sigma[\![e_1]\!]$ is distributed according to $[\![e_1]\!]$ when $\sigma = [\![A_1]\!]$, while it is distributed according to $[\![e_2]\!] = [\![\bar{\mu}(e_1)]\!]$ when $\sigma = [\![A_2]\!] = [\![\mu(A_1)]\!]$. It follows that $\mathcal{A}$ has exactly the same advantage as $\mathcal{D}$. □

Based on the previous corollary, it is convenient to define a notion of "normal pattern", where the keys have been renamed in some standard way.

Definition 5. *The normalization of $e \in \mathbf{Pat}$ is the pattern $\mathtt{norm}(e) = \mu(e)$ obtained by applying the pseudo-renaming $\mu(k_i) = r_i$, where $K = \{k_1, \ldots, k_n\} = \mathbf{roots}(\mathbf{keys}(e))$ and $r_1, \ldots, r_n \in \mathbf{Rand}$.*

It immediately follows from the definition that $\mathtt{norm}(e) \cong e$, and that any two patterns e_0, e_1 are equivalent up to renaming ($e_0 \cong e_1$) if and only if their normalizations $\mathtt{norm}(e_0) = \mathtt{norm}(e_1)$ are identical.

4 Symbolic Semantics

Following [3,26], the symbolic semantics of an expression $e \in \mathbf{Exp}$ is defined by specifying the set of keys $S \subseteq \mathbf{keys}(e)$ recoverable from e by an adversary, and a corresponding pattern $\mathtt{proj}(e, S)$, which, informally, represents the adversary's view of e when given the ability to decrypt only under the keys in S. Informally, $\mathtt{proj}(e, S)$ can be thought as the projection of e onto the subset of expressions that use only keys in S for encryption. More specifically, $\mathtt{proj}(e, S)$ is obtained from e by replacing all undecryptable subexpression $\{\!|e'|\!\}_k \sqsubseteq e$ (where $k \notin S$) with a pattern $\{\!|\mathtt{shape}(e')|\!\}_k$ that reveals only the shape of the encrypted message. The formal definition of $\mathtt{proj}$ is given in Figure 2.

We remark that the definition of $\mathtt{proj}$ is identical to previous work [3,26], as it treats pseudo-random keys $\mathbf{Keys} = \mathbb{G}^*(\mathbf{Rand})$ just as regular keys, disregarding their internal structure. (Relations between pseudorandom keys will be taken into account when defining the set of keys S known to the adversary.) In particular, as shown in [3,26], this function satisfies the following properties[7]

$$\mathtt{proj}(e, \mathbf{Keys}) = e \tag{6}$$

$$\mathtt{proj}(\mathtt{proj}(e, S), T) = \mathtt{proj}(e, S \cap T). \tag{7}$$

In order to define S, we need to specify the set of keys $\mathbf{rec}(e) \subseteq \mathbf{keys}(e)$ that an adversary may (potentially) extract from all the parts of an expression (or pattern) e. In the standard setting, where keys are atomic symbols, and encryption is the only cryptographic primitive, $\mathbf{rec}(e)$ can be simply defined as the set of keys appearing in e as a message. This is because the partial information about a key k revealed by a ciphertext $\{\!|m|\!\}_k$ is of no use to an adversary, except possibly for telling when two ciphertexts are encrypted under the same key. When dealing with expressions that make use of possibly related pseudorandom keys and multiple cryptographic primitives, one needs to take into account the possibility that an adversary may combine different pieces of partial information about the keys in mounting an attack. To this end, we define $\mathbf{rec}(e)$ to include all keys k such that either

1. e contains k as a message (directly revealing the value of k), or
2. e contains both a message encrypted under k (providing partial information about k) and some other related key k' (providing an additional piece of information about k).

In other words, our definition postulates that the symbolic adversary can fully recover a key k whenever it is given two distinct pieces of partial information about it. In addition, $\mathbf{rec}(e)$ contains all other keys that can be derived using the pseudorandom generator $\mathbb{G}$.

[7] Notice that by (7), the functions $\mathtt{proj}(\cdot, S)$ and $\mathtt{proj}(\cdot, T)$ commute, i.e., $\mathtt{proj}(\mathtt{proj}(e, S), T) = \mathtt{proj}(\mathtt{proj}(e, T), S)$ for any expression e. Indeed, for example, if $S = \{k_1\}$, $T = \{k_2\}$ and $e = \{\!|\{\!|m|\!\}_{k_1}|\!\}_{k_2}$, then $\mathtt{proj}(e, \{k_1\}) = \{\!|\{\!|\Box|\!\}|\!\}_{k_2}$, $\mathtt{proj}(e, \{k_2\}) = \{\!|\{\!|\Box|\!\}_{k_1}|\!\}_{k_2}$, and $\mathtt{proj}(\mathtt{proj}(e, \{k_1\}), \{k_2\}) = \mathtt{proj}(\mathtt{proj}(e, \{k_2\}), \{k_1\}) = \mathtt{proj}(e, \emptyset) = \{\!|\{\!|\Box|\!\}|\!\}_{k_2}$.

Definition 6. *For any pattern e, let* $\mathbf{rec}(e) = \mathbf{keys}(e) \cap \mathbb{G}^*(K)$ *where*

$$\begin{aligned} K &= \mathbf{keys}(e) \cap (\mathbf{parts}(e) \cup \mathbb{G}^-(\mathbf{keys}(e))) \\ &= \mathbf{pkeys}(e) \cup (\mathbf{keys}(e) \cap \mathbb{G}^-(\mathbf{keys}(e))). \end{aligned}$$

The expression $\mathbf{keys}(e) \cap \mathbb{G}^*(K)$ simply extends the set of known keys K using the pseudorandom generator. The interesting part of Definition 6 is the set K, which captures the key recovery capabilities of the adversary: $\mathbf{pkeys}(e)$ are all the keys that appear in e as a message, and $\mathbf{keys}(e) \cap \mathbb{G}^-(\mathbf{keys}(e))$ are the keys for which the adversary can obtain some additional partial information.[8] Our definition may seem overly conservative, as it postulates, for example, that a key k can be completely recovered simply given two ciphertexts $\{\!|\Box|\!\}_k$ and $\{\!|\Box|\!\}_{k'}$ where $k' = \mathbb{G}_{101}(k)$ is derived form k using the (one-way) functions $\mathbb{G}_0, \mathbb{G}_1$. In Section 5 we justify our definition by showing that there are encryption schemes and pseudorandom generators for which this is indeed possible, and proving a completeness theorem for the symbolic sematics associated to Definition 6. Specifically, if our definition enables a symbolic attacker to distinguish between two expressions e and e', then there is also an efficient computational adversary that distinguishes between the corresponding probability distributions for some valid computational interpretation of the cryptographic primitives.

The functions $\mathtt{proj}$ and $\mathbf{rec}$ are used to associate to each expression e a corresponding *key recovery map* $\mathbb{F}_e$, which, on input a set of keys S, outputs the set of keys $\mathbb{F}_e(S)$ potentially recoverable from e when using the keys in S for decryption.

$$\mathbb{F}_e \colon \mathbf{keys}(e) \to \mathbf{keys}(e) \qquad \text{where} \qquad \mathbb{F}_e(S) = \mathbf{rec}(\mathtt{proj}(e, S)). \tag{8}$$

A symbolic adversary that intercepts the expression e, and whose initial knowledge is the empty set of keys $S_0 = \emptyset$, can obtain more and more keys $S_1 = \mathbb{F}_e(S_0)$, $S_2 = \mathbb{F}_e(S_1), \ldots, S_{i+1} = \mathbb{F}_e(S_i) = \mathbb{F}_e^{i+1}(\emptyset)$, and ultimately recover all the keys in the set[9]

$$\mathrm{fix}(\mathbb{F}_e) = \bigcup_{n \geq 0} S_n = \bigcup_{n \geq 0} \mathbb{F}_e^n(\emptyset). \tag{9}$$

In summary, the symbolic semantics of an expression e can be defined as follows.

Definition 7. *The (least fixed point) symbolic semantics of a cryptographic expression e is the pattern*

$$\mathtt{pattern}(e) = \mathtt{norm}(\mathtt{proj}(e, \mathit{fix}(\mathbb{F}_e)))$$

where $\mathit{fix}(\mathbb{F}_e) = \bigcup_{n \geq 0} \mathbb{F}_e^n(\emptyset)$.

[8] By symmetry, and the final application of $\mathbb{G}$ in the definition of $\mathbf{rec}$, keys recoverable from partial information of type $\mathbf{keys}(e) \cap \mathbb{G}^+(\mathbf{keys}(e))$ are also captured implicitly by the this definition, simply by swapping the role of the two keys.

[9] As we will see, the key recovery map $\mathbb{F}_e$ is monotone, i.e., if $S \subseteq S'$, then $\mathbb{F}_e(S) \subseteq \mathbb{F}_e(S')$, for any two sets of keys S, S'. Therefore, $\mathbb{F}_e$ defines a monotonically increasing sequence of known sets of keys $S_0 \subset S_1 \subset S_2 \subset \ldots S_n = S_{n+1}$ and the set of keys recoverable by the adversary $S_n = \mathrm{fix}(\mathbb{F}_e)$ is precisely the least fixed point of $\mathbb{F}_e$, i.e., the smallest set S such that $\mathbb{F}_e(S) = S$.

$$\texttt{proj}(d, S) = d \qquad \texttt{proj}((e_1, e_2), S) = (\texttt{proj}(e_1, S), \texttt{proj}(e_2, S))$$
$$\texttt{proj}(k, S) = k \qquad \texttt{proj}(\{\!|e|\!\}_k, S) = \begin{cases} \{\!|\texttt{shape}(e)|\!\}_k & \text{if } k \notin S \\ \{\!|\texttt{proj}(e, S)|\!\}_k & \text{if } k \in S \end{cases}$$

Fig. 2. The the pattern function $\texttt{proj}\colon \mathbf{Pat}[\mathbf{Keys}, \mathbf{Data}] \times \wp(\mathbf{Keys}) \to \mathbf{Pat}[\mathbf{Keys}, \mathbf{Data}]$ where $k \in \mathbf{Keys}$, $d \in \mathbf{Data}$, and $(e_1, e_2), \{\!|e|\!\}_k \in \mathbf{Pat}[\mathbf{Keys}, \mathbf{Data}]$. Intuitively, $\texttt{proj}(e, S)$ is the observable pattern of e, when using the keys in S for decryption.

In the above definition,$S = \text{fix}(\mathbb{F}_e)$ is the set of all keys recoverable by an adversary that intercepts e, $\texttt{proj}(e, S)$ is (the symbolic representation of) what part of e can be decrypted by the adversary, and the final application of `norm` takes care of key renamings.

We conclude this section by observing that the function **rec** satisfies the fundamental property

$$\mathbf{rec}(\texttt{proj}(e, S)) \subseteq \mathbf{rec}(e) \tag{10}$$

which, informally, says that projecting an expression (or pattern) e does not increase the amount of information recoverable from it. In fact, for any pattern e, the set $\mathbf{rec}(e)$ depends only on the sets $\mathbf{keys}(e)$ and $\mathbf{pkeys}(e)$. Moreover, this dependence is monotone. Since we have $\mathbf{keys}(\texttt{proj}(e, S)) \subseteq \mathbf{keys}(e)$ and $\mathbf{pkeys}(\texttt{proj}(e, S)) \subseteq \mathbf{pkeys}(e)$, by monotonicity we get $\mathbf{rec}(\texttt{proj}(e, S)) \subseteq \mathbf{rec}(e)$.

As an application, [26, Theorem 1] shows that for any functions $\texttt{proj}, \mathbf{rec}$ satisfying properties (6), (7) and (10), the function $\mathbb{F}_e(S) = \mathbf{rec}(\texttt{proj}(e, S))$ is monotone, i.e., if $S \subseteq T$, then $\mathbb{F}_e(S) \subseteq \mathbb{F}_e(T)$.

5 Completeness

In this section we prove that the symbolic semantics defined in Section 4 is complete, i.e., if two cryptographic expressions map to different symbolic patterns (as specified in Definition 7), then the corresponding probability distributions can be efficiently distinguished. More specifically, we show that for any two such symbolic expressions e_0, e_1, there is a secure computational interpretation $[\![\cdot]\!]$ (satisfying the standard computational notions of security for pseudorandom generators and encryption schemes) and an efficiently computable predicate $\mathcal{D}$ such that $\Pr\{\mathcal{D}([\![e_0]\!])\} \approx 0$ and $\Pr\{\mathcal{D}([\![e_1]\!])\} \approx 1$.

The core of our completeness theorem is the following lemma, which shows that computationally secure encryption schemes and pseudorandom generators can leak enough partial information about their keys, so to make the keys completely recoverable whenever two keys satisfying a nontrivial relation are used to encrypt. The key recovery algorithm $\mathcal{A}$ described in Lemma 4 provides a tight computational justification for the symbolic key recovery function **rec** described in Definition 6.

Lemma 4. *If pseudorandom generators and encryption schemes exist at all, then there is a secure computational interpretation $(\mathcal{G}, \mathcal{E}, \gamma, \pi)$ and a deterministic polynomial time key recovery algorithm $\mathcal{A}$ such that the following holds. For any (symbolic) keys $k_0, k_1 \in \mathbf{Keys}$, messages m_0, m_1, and binary string $w \neq \epsilon$,*

- *if $k_1 = \mathbb{G}_w(k_0)$, then $\mathcal{A}(\mathcal{E}_{\sigma(k_0)}(m_0), \mathcal{E}_{\sigma(k_1)}(m_1), w) = \sigma(k_0)$ for any key assignment σ; and*
- *if $k_1 \neq \mathbb{G}_w(k_0)$, then $\mathcal{A}(\mathcal{E}_{\sigma(k_0)}(m_0), \mathcal{E}_{\sigma(k_1)}(m_1), w) = \perp$ outputs a special symbol $\perp$ denoting failure, except with negligible probability over the random choice of the key assignment σ.*

Proof. We show how to modify any (length doubling) pseudorandom generator $\mathcal{G}'$ and encryption scheme $\mathcal{E}'$ to satisfy the properties in the lemma. Before describing the actual construction, we provide some intuition. The idea is to use an encryption scheme that splits the key $k = (k[0]; k[1])$, uses half of the key (say, $k[1]$) and leaks the first half $k[0]$ as part of the ciphertext. Notice that this is already enough to tell if two ciphertexts are encrypted under the same key, as exposed by patterns like $(\{\!|\Box|\!\}_k, \{\!|\Box|\!\}_{k'})$. But, still, this does not leak any information about the messages, which are well protected by the undisclosed portion of the keys. In order to prove the lemma, we need an appropriate pseudorandom generator which, when combined with the encryption scheme, leads to a key recovery attack. Similarly to the encryption scheme, the pseudorandom generator uses only $k[0]$ (which is expanded by a factor 4, to obtain a string twice as long as the original k), and uses the result to "mask" the second part $k[1]$. Specifically, each half of the output $\mathcal{G}_b(k)$ equals $(\mathcal{G}'_{0b}(k[0]), \mathcal{G}'_{1b}(k[0]) \oplus k[1])$. Now, given an encryption under k (which leaks $k[0]$), and a one-way function $\mathcal{G}_b(k)$ (for any bit b) of the key, one can recover $k[1]$ as follows: expand $k[0]$ to $(\mathcal{G}'_{0b}(k[0]), \mathcal{G}'_{1b}(k[0]))$ and use the result to unmask $\mathcal{G}_b(k)$, to reveal $(0, k[1])$. The same argument is easily adapted to work for any one-way function $\mathcal{G}_w(k)$ corresponding to an arbitrary sequence of applications w of the pseudorandom generator. The problem with this intuitive construction is that it requires to see the full output of $\mathcal{G}_b(k)$. If, instead, we are given only two ciphertests (encrypted under k and $\mathcal{G}_b(k)$) one gets to learn only the first half $\mathcal{G}_b(k)$, which is not enough to recover $k[1]$. An easy fix to this specific problem is to let $\mathcal{G}_b(k)$ to mask $(k[1], k[1])$ instead of $(0, k[1])$. But this would not allow the attack to carry over to longer applications $\mathcal{G}_w(k)$ of the pseudorandom generator. So, the actual construction required to prove the lemma is a bit more complex, and splits the key into three parts.

The new $\mathcal{E}$ and $\mathcal{G}$ use keys that are three times as long as those of $\mathcal{E}'$ and $\mathcal{G}'$. Specifically, each new key $\sigma(k)$ consists of three equal length blocks which we denote as $\sigma(k)[0], \sigma(k)[1]$ and $\sigma(k)[2]$, where each block can be used as a seed or encryption key for the original $\mathcal{G}'$ and $\mathcal{E}'$. Alternatively, we may think of k as consisting of three atomic symbolic keys $k = (k[0], k[1], k[2])$, each corresponding to ℓ bits of $\sigma(k)$. For notational simplicity, in the rest of the proof, we fix a random key assignment σ, and, with slight abuse of notation, we identify the symbolic keys $k[i]$ with the corresponding ℓ-bit strings $\sigma(k)[i]$. So, for example, we will write k and $k[i]$ instead of $\sigma(k)$ and $\sigma(k)[i]$. Whether each $k[i]$ should be

interpreted as a symbolic expression or as a bitstring will always be clear from the context.

The new encryption scheme $\mathcal{E}(k,m) = k[0]; k[1]; \mathcal{E}'(k[2], m)$ simply leaks the first two blocks of the key, and uses the third block to perform the actual encryption. It is easy to see that if $\mathcal{E}'$ is secure against chosen plaintext attacks, then $\mathcal{E}'$ is also secure. Moreover, $\mathcal{E}$ can be made length regular simply by padding the output of $\mathcal{E}'$ to its maximum length.

For the pseudo-random generator, assume without loss of generality that $\mathcal{G}'$ is length doubling, mapping strings of length ℓ to strings of length 2ℓ. We need to define a new $\mathcal{G}$ mapping strings of length 3ℓ to strings of length 6ℓ. On input $k = k[0]; k[1]; k[2]$, the new $\mathcal{G}$ stretches $k[0]$ to a string of length 6ℓ corresponding to the symbolic expression

$$(\mathbb{G}_{00}(k[0]), \mathbb{G}_{010}(k[0]), \mathbb{G}_{110}(k[0]), \mathbb{G}_{01}(k[0]), \mathbb{G}_{011}(k[0]), \mathbb{G}_{111}(k[0])) \tag{11}$$

and outputs the exclusive-or of this string with $(0; k[2]; k[2]; 0; k[2]; k[2])$. The expression (11) is evaluated using $\mathcal{G}'$. Since $\mathcal{G}'$ is a secure length doubling pseudorandom generator, and the keys in (11) are symbolically independent, by Theorem 1 expression (11) is mapped to a pseudorandom string of length 6ℓ. Finally, since taking the exclusive-or with any fixed string $(0; k[2]; k[2]; 0; k[2]; k[2])$ maps the uniform distribution to itself, the output of $\mathcal{G}$ is also computationally indistinguishable from a uniformly random string of length 6ℓ. This proves that $\mathcal{G}$ is a secure length doubling pseudorandom generator as required. It will be convenient to refer to the first and second halves of this pseudorandom generator $\mathcal{G}(k) = \mathcal{G}_0(k); \mathcal{G}_1(k)$. Using the definition of $\mathcal{G}$, we see that for any bit $b \in \{0,1\}$, the corresponding half of the output consists of the following three blocks:

$$\mathcal{G}_b(k)[0] = [\![\mathbb{G}_{0b}(k[0])]\!] \tag{12}$$

$$\mathcal{G}_b(k)[1] = [\![\mathbb{G}_{01b}(k[0])]\!] \oplus k[2] \tag{13}$$

$$\mathcal{G}_b(k)[2] = [\![\mathbb{G}_{11b}(k[0])]\!] \oplus k[2]. \tag{14}$$

Next, we describe the key recovery algorithm $\mathcal{A}$. This algorithm takes as input two ciphertexts $\mathcal{E}_{k_0}(m_0)$, $\mathcal{E}_{k_1}(m_1)$ and a binary string w. The two ciphertexts are only used for the purpose of recovering the partial information about the keys $k_0[0], k_0[1], k_1[0], k_1[1]$ leaked by $\mathcal{E}$. So, we assume $\mathcal{A}$ is given $k_0[0], k_0[1]$ and $k_1[0], k_1[1]$ to start with. Let $w = w_n \dots w_1$ be any bitstring of length n, and define the sequence of keys $k^i = (k^i[0], k^i[1], k^i[2])$ by induction as

$$k^0 = k_0, \qquad k^{i+1} = \mathcal{G}_{w_{i+1}}(k^i)$$

for $i = 0, \dots, n-1$. Notice that, if k_0 and k_1 are symbolically related by $k_1 = \mathbb{G}_w(k_0)$, then the last key in this sequence equals $k^n = k_1$ as a string in $\{0,1\}^{3\ell}$.

Using (12), the first block of these keys can be expressed symbolically as

$$k^i[0] = [\![\mathbb{G}_{u_i}(k_0[0])]\!] \qquad \text{where} \qquad u_i = 0w_i0w_{i-1}\dots0w_1.$$

So, Algorithm $\mathcal{A}(k_0[0], k_0[1], k_1[0], k_1[1], w)$ begins by computing the value of all $k^i[0] = [\![\mathbb{G}_{u_i}(k_0[0])]\!]$ (for $i = 0, \dots, n$) starting from the input value $k_0[0]$ and

applying the pseudorandom generator $\mathcal{G}'$ as directed by u_i. At this point, $\mathcal{A}$ may compare $k^n[0]$ with its input $k_1[0]$, and expect these two values to be equal. If the values differ, $\mathcal{A}$ immediately terminates with output $\perp$. We will prove later on that if $k_1 \neq \mathbb{G}_w(k_0)$, then $k^n[0] \neq k_1[0]$ with high probability, and $\mathcal{A}$ correctly outputs $\perp$. But for now, let us assume that $k_1 = \mathbb{G}_w(k_0)$, so that $k_1 = [\![\mathbb{G}_w(k_0)]\!] = k^n$ and the condition $k^n[0] = k_1[0]$ is satisfied. In this case, $\mathcal{A}$ needs to recover and output the key k_0. Since algorithm $\mathcal{A}$ is already given $k_0[0]$ and $k_0[1]$ as part of its input, all we need to do is to recover the last block $k_0[2]$ of the key. To this end, $\mathcal{A}$ first uses (13) to compute $k^{n-1}[2]$ as

$$\begin{aligned} k_1[1] \oplus \mathcal{G}_0(\mathcal{G}_1(\mathcal{G}_{w_n}(k^{n-1}[0]))) &= k^n[1] \oplus [\![\mathbb{G}_{01w_n}(k^{n-1}[0])]\!] \\ &= k^n[1] \oplus (\mathcal{G}_{w_n}(k^{n-1})[1] \oplus k^{n-1}[2]) \\ &= k^n[1] \oplus (k^n[1] \oplus k^{n-1}[2]) = k^{n-1}[2]. \end{aligned}$$

Similarly, starting from $k^{n-1}[2]$, $\mathcal{A}$ uses (14) to compute $k^i[2]$ for $i = n-2, n-3, \ldots, 0$ as

$$\begin{aligned} k^{i+1}[2] \oplus \mathcal{G}_1(\mathcal{G}_1(\mathcal{G}_{w_{i+1}}(k^i[0]))) &= k^{i+1}[2] \oplus [\![\mathbb{G}_{11w_{i+1}}(k^i[0])]\!] \\ &= k^{i+1}[2] \oplus (\mathcal{G}_{w_{i+1}}(k^i)[2] \oplus k^i[2]) \\ &= k^{i+1}[2] \oplus (k^{i+1}[2] \oplus k^i[2]) = k^i[2]. \end{aligned}$$

At this point, $\mathcal{A}$ can output $(k_0[0], k_0[1], k^0[2]) = (k_0[0], k_0[1], k_0[2]) = k_2$. This completes the analysis for the case $k_1 = \mathbb{G}_w(k_0)$.

We need to show that if $k_1 \neq \mathbb{G}_w(k_0)$, then the probability that $k^n[0] = k_1[0]$ is negligible, so that $\mathcal{A}$ correctly outputs $\perp$. Since we are interested only in the first blocks $k^n[0], k_1[0]$ of the keys, we introduce some notation. For any bitstring $v = v_1 \ldots v_m$, let $0|v = 0v_1 0v_2 \ldots 0v_m$ be the result of shuffling v with a string of zeros of equal length. If we express $k^n[0] = \mathbb{G}_{0|w}(k_0[0])$ in terms of $k_0[0]$, the goal becomes to prove that $\mathbb{G}_{0|w}(k_0[0])$ and $k_1[0]$ evaluate to different strings with overwhelming probability. The proof proceeds by cases, depending on whether $k_0 \perp k_1$, $k_0 \prec k_1$, or $k_1 \preceq k_0$, and makes use of the symbolic characterization of computational independence from Section 3.

Case 1. If $k_0 \perp k_1$, then $k_0 = \mathbb{G}_{v_0}(r_0)$ and $k_1 = \mathbb{G}_{v_1}(r_1)$ for some r_0, r_1, v_0, v_1 such that either $r_0 \neq r_1$, or v_0, v_1 are not one a suffix of the other. It follows that $k_0[0] = \mathbb{G}_{0|v_0}(r_0)$ and $k_1[0] = \mathbb{G}_{0|v_1}(r_1)$ are also symbolically independent because either $r_0 \neq r_1$, or $(0|v_0), (0|v_1)$ are not one a suffix of the other. In this case, also $k^n[0] = \mathbb{G}_{0|w}(k_0[0])$ and $k_1[0]$ are symbolically independent. It follows, from Theorem 1, that the distribution $[\![\mathbb{G}_{0|w}(k_0[0]), k_1[0]]\!]$ is computationally indistinguishable from the evaluation $[\![r_0, r_1]\!]$ of two independent uniformly random keys. In particular, since r_0 and r_1 evaluate to the same bitstring with exponentially small probability $2^{-\ell}$, the probability that $k^n[0] = \mathbb{G}_{0|w}(k_0[0])$ and $k_1[0]$ evaluate to the same string is also negligible.

Case 2. If $k_1 \prec k_0$, then $k_0 = \mathbb{G}_v(k_1)$ for some string $v \neq \epsilon$, and $k_0[0] = \mathbb{G}_{0|v}(k_1[0])$. Then, the pair of keys $(k^n[0], k_1[0])$ where

$$k^n[0] = \mathbb{G}_{0|w}(k_0[0]) = \mathbb{G}_{0|w}(\mathbb{G}_{0|v}(k_1[0])) = \mathbb{G}_{0|wv}(k_1[0])$$

is symbolically equivalent to $(\mathbb{G}_u(r), r)$ for some $u = (0|wv) \neq \epsilon$. So, by Theorem 1, we can equivalently bound the probability δ (over the random choice of σ) that $[\![\mathbb{G}_u(r)]\!]_\sigma$ evaluates to $[\![r]\!]_\sigma$. The trivial (identity) algorithm $\mathcal{I}(y) = y$ inverts the function defined by $\mathbb{G}_u$ with probability at least δ. Since $u \neq \epsilon$, $\mathbb{G}_u$ defines a one-way function, and δ must be negligible.

Case 3. Finally, if $k_0 \preceq k_1$, then $k_1 = \mathbb{G}_v(k_0)$ for some string $v \neq w$, and $k_1[0] = \mathbb{G}_{0|v}(k_0[0])$. This time, we are given a pair of keys

$$(k^n[0], k_1[0]) = (\mathbb{G}_{0|w}(k_0[0]), \mathbb{G}_{0|v}(k_0[0]))$$

which are symbolically equivalent to $(\mathbb{G}_{0|w}(r), \mathbb{G}_{0|v}(r))$. As before, by Theorem 1, it is enough to evaluate the probability δ that $\mathbb{G}_{0|w}(r)$ and $\mathbb{G}_{0|v}(r)$ evaluate to the same bitstring. If v is a (strict) suffix of w or w is a (strict) suffix of v, then δ must be negligible by the same argument used in Case 2. Finally, if v and w are not one a suffix of the other, then $\mathbb{G}_{0|w}(r)$ and $\mathbb{G}_{0|v}(r)$ are symbolically independent, and δ must be negligible by the same argument used in Case 1.

We have shown that in all three cases, the probability δ that $\mathbb{G}_{u_n}(k_0[0])$ and $k_1[0]$ evaluate to the same bitstring is negligible. So, the test performed by $\mathcal{A}$ fails (expect with negligible probability) and $\mathcal{A}$ outputs $\perp$ as required by the lemma. □

We use Lemma 4 to distinguish between expressions that have the same shape. Expressions with different shapes can be distinguished more easily simply by looking at their bitsize. Recall that for any (length regular) instantiation of the cryptographic primitives, the length of all strings in the computational interpretation of a pattern $[\![e]\!]$ (denoted $|[\![e]\!]|$) depends only on $\mathtt{shape}(e)$. In other words, for any two patterns e_0, e_1, if $\mathtt{shape}(e_0) = \mathtt{shape}(e_1)$, then $|[\![e_0]\!]| = |[\![e_1]\!]|$. The next lemma provides a converse of this property, showing that whenever two patterns have different shape, they may evaluate to strings of different length. So, secure computational interpretations are not guaranteed to protect any piece of partial information about the shape of symbolic expressions.

Lemma 5. *If pseudorandom generators and encryption schemes exist at all, then for any two expressions e_0 and e_1 with $\mathtt{shape}(e_0) \neq \mathtt{shape}(e_1)$, there exists a secure computational interpretation $(\mathcal{G}, \mathcal{E}, \gamma, \pi)$ such that $|[\![e_0]\!]| \neq |[\![e_1]\!]|$.*

Proof. We show how to modify any secure computational interpretation simply by padding the output length, so that the lemma is satisfied. More specifically, we provide a computational interpretation such that the length of $[\![e_0]\!]$ is different from the length of any expression with different shape. Let $S = \{\mathtt{shape}(e) \mid e \in \mathbf{parts}(e_0)\}$ be the set of all shapes of subexpressions of e_0, and let $n = |S| + 1$. Associate to each shape $s \in S$ a unique number $\varphi(s) \in \{1, \ldots, n-1\}$, and define $\varphi(s) = 0$ for all shapes $s \notin S$. Data blocks and keys are padded to bit-strings of length congruent to $\varphi(\square)$ and $\varphi(\circ)$ modulo n, respectively. The encryption function first applies an arbitrary encryption scheme, and then pads the ciphertext $\mathcal{E}(m)$ so that its length modulo n equals $\varphi(\{\!|s|\!\})$, for some shape

s such that $|m| = \varphi(s)$. The pairing function π is defined similarly: if the two strings being combined in a pair have length $|m_0| = \varphi(s_0) \pmod n$ and $|m_1| = \varphi(s_1) \pmod n$, then the string encoding the pair (m_0, m_1) is padded so that its length equals $\varphi(s_0, s_1)$ modulo n. It is easy to check that all patterns e are evaluated to strings of length $|[\![e]\!]| = \varphi(\mathtt{shape}(e)) \pmod n$. Since $\mathtt{shape}(e_0) \in S$ and $\mathtt{shape}(e_1) \notin S$, we get $|[\![e_0]\!]| \neq 0 \pmod n$ and $|[\![e_1]\!]| = 0 \pmod n$. In particular, $|[\![e_0]\!]| \neq |[\![e_1]\!]|$. □

We are now ready to prove our completeness theorem, and establish the optimality of our symbolic semantics.

Theorem 2. *For any two expressions e_0 and e_1, if $\mathtt{pattern}(e_0) \neq \mathtt{pattern}(e_1)$, then there exists a secure computational interpretation $(\mathcal{G}, \mathcal{E}, \gamma, \pi)$ and a polynomial time computable predicate $\mathcal{D}$ such that $\Pr\{\mathcal{D}([\![e_0]\!])\} \approx 0$ and $\Pr\{\mathcal{D}([\![e_1]\!])\} \approx 1$, i.e., the distributions $[\![e_0]\!]$ and $[\![e_1]\!]$ can be distinguished with negligible probability of error.*

Proof. We consider two cases, depending on the shapes of the expressions. If $\mathtt{shape}(e_0) \neq \mathtt{shape}(e_1)$, then let $[\![\cdot]\!]$ be the computational interpretation defined in Lemma 5. Given a sample α from one of the two distributions, the distinguisher $\mathcal{D}$ simply checks if $|\alpha| = |[\![\mathtt{shape}(e_1)]\!]|$. If they are equal, it accepts. Otherwise it rejects. It immediately follows from Lemma 5 that this distinguisher is always correct, accepting all samples α from $[\![e_1]\!]$, and rejecting all samples α from $[\![e_0]\!]$.

The more interesting case is when $\mathtt{shape}(e_0) = \mathtt{shape}(e_1)$. This time the difference between the two expressions is not in their shape, but in the value of the keys and data. This time we use the computational interpretation $[\![\cdot]\!]$ defined in Lemma 4, and show how to distinguish between samples from $[\![e_0]\!]$ and samples from $[\![e_1]\!]$, provided $\mathtt{pattern}(e_0) \neq \mathtt{pattern}(e_1)$.

Let $S_b^i = \mathbb{F}_{e_b}^i(\emptyset)$ be the sequence of sets of keys defined by e_b. We know that $\emptyset = S_b^0 \subseteq S_b^1 \subseteq S_b^2 \subseteq \cdots \subseteq S_b^n = \mathrm{fix}(\mathbb{F}_{e_b})$ for some integer n. Let $e_b^i = \mathbf{Pat}(e_b, S_b^i)$ be the sequence of patterns defined by the sets S_b^i. Since $\mathtt{pattern}(e_0) \neq \mathtt{pattern}(e_1)$, we have $e_0^n \not\cong e_1^n$. Let i the smallest index such that $e_0^i \not\cong e_1^i$. We will give a procedure that iteratively recovers all the keys in the sets $S_b^0, S_b^1, \ldots, S_b^{i-1}$, and then distinguishes between samples coming from the two distributions associated to e_0 and e_1.

The simplest case is when $i = 0$, i.e., $e_0^0 \not\cong e_1^0$. In this case $S_0^0 = \emptyset = S_1^0$, and we do not need to recover any keys. Since e_0 and e_1 have the same shape, $\mathcal{D}$ can unambiguously parse α as a concatenation of data blocks d, keys k and ciphertexts of type $\{\!|s|\!\}_k$, without knowing if α comes from $[\![e_0]\!]$ or $[\![e_1]\!]$. If the two patterns e_0^0, e_1^0 differ in one of the data blocks, then $\mathcal{D}$ can immediately tell if α comes from e_0^0 or e_1^0 by looking at the value of that piece of data. So, assume all data blocks are identical, and e_0^0 and e_1^0 differ only in the values of the keys. Consider the set P of all key positions in e_0^0 (or, equivalently, in e_1^0), and for every position $p \in P$, let k_b^p be the key in e_b^0 at position p. (Positions incude both plain keys k_b^p and ciphertexts $\{\!|s_p|\!\}_{k_b^e}$.) For any two positions p, p', define

the relation $r_b(p,p')$ between the keys k_b^p and $k_b^{p'}$ to be

$$r_b(p,p') = \begin{cases} +w \text{ if } k_b^{p'} = \mathbb{G}_w(k_b^p) \text{ for some } w \in \{0,1\}^+ \\ -w \text{ if } k_b^p = \mathbb{G}_w(k_b^{p'}) \text{ for some } w \in \{0,1\}^+ \\ 0 \quad \text{if } k_b^p = k_b^{p'} \\ \bot \quad \text{otherwise.} \end{cases}$$

Notice that if $r_0(p,p') = r_1(p,p')$ for all positions $p,p' \in P$, then the map $\mu(k_0^p) = k_1^p$ is a valid pseudorenaming. Since $\mu(e_0^0) = e_1^0$, this would show that $e_0^0 \cong e_1^0$, a contradiction. So, there must be two positions p,p' such that $r_0(p,p') \neq r_1(p,p')$, i.e., the keys at positions p and p' in the two expressions e_0^0 and e_1^0 satisfy different relations. At this point we distinguish two cases:

- If two keys are identical ($r_b(p,p') = 0$) and the other two keys are unrelated ($r_{1-b}(p,p') = \bot$), then we can determine the value of b simply by checking if the corresponding keys recovered from the sample α are identical or not. Notice that even if the subexpression at position p (or p') is a ciphertext, the encryption scheme defined in Lemma 4 still allows to recover the first 2ℓ bits of the keys, and this is enough to tell if two keys are identical or independent with overwhelming probability.
- Otherwise, it must be the case that one of the two relations is $r_b(p,p') = \pm w$ for some string w. By possibly swapping p and p', and e_0 and e_1, we may assume that $r_0(p,p') = +w$ while $r_1(p,p') \neq +w$. In other words, $k_0^{p'} = \mathbb{G}_w(k_0^p)$, while $k_1^{p'} \neq \mathbb{G}_w(k_0^p)$. We may also assume that the subexpressions at position p and p' are ciphertexts. (If the subexpression at one of these positions is a key, we can simply use it to encrypt a fixed message m, and obtain a corresponding ciphertext.) Let α_0, α_0' be the ciphertexts extracted from α corresponding to positions p and p'. We invoke the algorithm $\mathcal{A}(\alpha_0, \alpha_0', w)$ from Lemma 4 and check if it outputs a key or the special failure symbol $\bot$. The distinguisher accepts if and only if $\mathcal{A}(\alpha_0, \alpha_0', w) = \bot$. By Lemma 4, if α was sampled from $[\![e_0]\!]$, then $\mathcal{A}(\alpha_0, \alpha_0', w)$ will recover the corresponding key with probability 1, and $\mathcal{D}$ rejects the sample α. On the other hand, if α was sampled from $[\![e_1]\!]$, then $\mathcal{A}(\alpha_0, \alpha_0', w) = \bot$ with overwhelming probability, and $\mathcal{D}$ accepts the sample α.

This completes the description of the decision procedure $\mathcal{D}$ when $i = 0$. When $i > 1$, we first use Lemma 4 to recover the keys in S_b^1. Then we use these keys to decrypt the corresponding subexpressions in α, and use Lemma 4 again to recover all the keys in S_b^2. We proceed in a similar fashion all the way up to S_b^{i-1}. Notice that since all the corresponding patterns $e_0^j \cong e_1^j$ (for $j \leq i$) are equivalent up to renaming, all the keys at similar positions p,p' satisfy the same relations $r_0(p,p') = r_1(p,p')$, and we can apply Lemma 4 identically, whether the sample α comes from $[\![e_0]\!]$ or $[\![e_1]\!]$. This allows to recover the keys in S_b^i, at which point we can parse (and decrypt) α to recover all the data blocks, keys and ciphertexts appearing in e_b^i. Finally, using the fact that $e_0^i \ncong e_1^i$, we proceed as in the case $i = 0$ to determine the value of b. □

6 Computational soundness

Computational soundness results for symbolic cryptography usually forbid encryption cycles, e.g., collections of ciphertexts where k_i is encrypted under k_{i+1} for $i = 1, \ldots, n-1$, and k_n is encrypted under k_1. Here we follow an alternative approach, put forward in [26], which defines the adversarial knowledge as the *greatest fixed point* of $\mathbb{F}_e$, i.e., the *largest* set S such that $\mathbb{F}_e(S) = S$. Interestingly, [26] shows that under this "co-inductive" definition of the set of known keys, soundness can be proved in the presence of encryption cycles, offering a tight connection between symbolic and computational semantics. At the same time, [26, Theorem 2] also shows that if e has no encryption cycles, then $\mathbb{F}_e$ has a unique fixed point, and therefore $\mathrm{fix}(\mathbb{F}_e) = \mathrm{FIX}(\mathbb{F}_e)$. So, computational soundness under the standard "least fixed point" semantics for acyclic expressions follows as a corollary. We remark that $\mathbb{F}_e$ may have a unique fixed point even if e contains encryption cycles. So, based on [26, Theorem 2], we generalize the definition of acyclic expressions to include all expressions e such that $\mathbb{F}_e$ has a unique fixed point $\mathrm{fix}(\mathbb{F}_e) = \mathrm{FIX}_e(\mathbb{F}_e)$.

In this section we extend the results of [26] to expressions with pseudorandom keys. But, before doing that, we explain the intuition behind the co-inductive (greatest fixed point) semantics. Informally, using the greatest fixed point corresponds to working by induction on the set of keys that are *hidden* from the adversary, starting from the empty set (i.e., assuming that no key is hidden a-priori), and showing that more and more keys are provably secure. Formulating this process in terms of the complementary set of *potentially known* keys, one starts from the set of all keys $K = \mathbf{keys}(e)$, and repeatedly applies $\mathbb{F}_e$ to it. By monotonicity of $\mathbb{F}_e$ the result is a sequence of smaller and smaller sets

$$K \supset \mathbb{F}_e(K) \supset \mathbb{F}_e^2(K) \supset \mathbb{F}_e^3(K) \supset \cdots$$

of potentially known keys, which converges to the greatest fixed point

$$\mathrm{FIX}(\mathbb{F}_e) = \bigcap_n \mathbb{F}_e^n(\mathbf{keys}(e)).$$

We emphasize $\mathrm{FIX}(\mathbb{F}_e)$ should be interpreted as the set of keys that are only *potentially* recoverable by an adversary. Depending on the details of the encryption scheme (e.g., if it provides some form of key dependent message security), an adversary may or may not be able to recover all the keys in $\mathrm{FIX}(\mathbb{F}_e)$. On the other hand, all keys in the complementary set $\mathbf{Keys}(e) \setminus \mathrm{FIX}(\mathbb{F}_e)$ are *provably secret*, for any encryption scheme providing the minimal security level of indistinguishability under chosen message attack. Using the greatest fixed point, one can define an alternative symbolic semantics for cryptographic expressions,

$$\mathtt{PATTERN}(e) = \mathtt{norm}(\mathtt{proj}(e, \mathrm{FIX}(\mathbb{F}_e))). \tag{15}$$

In general, $\mathrm{fix}(\mathbb{F}_e)$ can be a strict subset of $\mathrm{FIX}(\mathbb{F}_e)$, so (15) may be different from the patterns defined in Section 4. However, if e is acyclic, then $\mathrm{FIX}(\mathbb{F}_e) = \mathrm{fix}_e(\mathbb{F}_e)$, and therefore $\mathtt{PATTERN}(e) = \mathtt{pattern}(e)$.

Theorem 3. *For any secure computational interpretation* $(\mathcal{G}, \mathcal{E}, \gamma, \pi)$ *and any expression* e*, the distributions* $[\![e]\!]$ *and* $[\![\texttt{PATTERN}(e)]\!]$ *are computationally indistinguishable. In particular, if* $\texttt{PATTERN}(e_0) = \texttt{PATTERN}(e_1)$*, then* $[\![e_0]\!] \approx [\![e_1]\!]$*, i.e., the equivalence relation induced by* `PATTERN` *is computationally sound.*

Corollary 2. *If* e_0, e_1 *are acyclic expressions, and* $\texttt{pattern}(e_0) = \texttt{pattern}(e_1)$*, then* $[\![e_0]\!]$ *and* $[\![e_1]\!]$ *are computationally indistinguishable.*

The proof of the soundness theorem is pretty standard, and similar to previous work, and can be found in the full version of the paper [27].

7 Conclusion

We presented a generalization of the computational soundness result of Abadi and Rogaway [3] (or, more precisely, its co-inductive variant put forward in [26]) to expressions that mix encryption with a pseudo-random generator. Differently from previous work in the area of multicast key distribution protocols [28,30,29], we considered unrestricted use of both cryptographic primitives, which raises new issues related partial information leakage that had so far been dealt with using ad-hoc methods. We showed that partial information can be adequately taken into account in a simple symbolic adversarial model where the attacker can fully recover a key from any two pieces of partial information. While, at first, this attack model may seem unrealistically strong, we proved, as our main result, a completeness theorem showing that the model is essentially optimal.

A slight extension of our results (to include the random permutation of ciphertexts) has recently been used in [24], which provides a computationally sound symbolic analysis of Yao's garbled circtuit construction for secure two party computation. The work of [24] illustrates the usefulness of the methods developed in this paper to the analysis of moderately complex protocols, and also provides an implementation showing that our symbolic semantics can be evaluated extremely fast even on fairly large expressions, e.g., those describing garbled circuits with thousands of gates. Our results can be usefully generalized even further, to include richer collections of cryptographic primitives, e.g., different types of (private and public key) encryption, secret sharing schemes (as used in [4]), and more. Extensions to settings involving active attacks are also possible [32,19], but probably more challenging.

Acknowledgments The author thanks the anonymous Eurocrypt 2019 referees for their useful comments.

References

1. Abadi, M., Gordon, A.: A calculus for cryptogaphic protocols: the spi calculus. Information and Computation **148**(1), 1–70 (1999). https://doi.org/10.1006/inco.1998.2740

2. Abadi, M., Jürjens, J.: Formal eavesdropping and its computational interpretation. In: TACS'01. LNCS, vol. 2215, pp. 82–94. Springer (2001). https://doi.org/10.1007/3-540-45500-0_4
3. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (The computational soundness of formal encryption). J. Cryptology **15**(2), 103–127 (2002). https://doi.org/10.1007/s00145-007-0203-0
4. Abadi, M., Warinschi, B.: Security analysis of cryptographycally controlled access to XML documents. J. ACM **55**(2), 1–29 (2008). https://doi.org/10.1145/1346330.1346331, prelim. version in PODS'05
5. Backes, M., Pfitzmann, B.: Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In: CSFW'04. pp. 204–218. IEEE (2004). https://doi.org/10.1109/CSFW.2004.20
6. Backes, M., Pfitzmann, B., Waidner, M.: Symmetric authentication in a simulatable Dolev-Yao-style cryptographic library. Int. J. Inf. Sec. **4**(3), 135–154 (2005). https://doi.org/10.1007/s10207-004-0056-6
7. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: CRYPTO'99. LNCS, vol. 1666, pp. 431–448. Springer (1999). https://doi.org/10.1007/3-540-48405-1_28
8. Böhl, F., Cortier, V., Warinschi, B.: Deduction soundness: prove one, get five for free. In: CCS'13. pp. 1261–1272. ACM (2013). https://doi.org/10.1145/2508859.2516711
9. Burrows, M., Abadi, M., Needham, R.M.: A logic of authentication. ACM Trans. Comput. Syst. **8**(1), 18–36 (1990). https://doi.org/10.1145/77648.77649
10. Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M., Pinkas, B.: Multicast security: A taxonomy and some efficient constructions. In: INFOCOM'99. pp. 708–716 (1999). https://doi.org/10.1109/INFCOM.1999.751457
11. Comon-Lundh, H., Cortier, V.: Computational soundness of observational equivalence. In: CCS'08. pp. 109–118. ACM (2008). https://doi.org/10.1145/1455770.1455786
12. Cortier, V., Warinschi, B.: A composable computational soundness notion. In: CCS'11. pp. 63–74. ACM (2011). https://doi.org/10.1145/2046707.2046717
13. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2), 198–208 (1983). https://doi.org/10.1109/TIT.1983.1056650
14. Gligor, V., Horvitz, D.O.: Weak key authenticity and the computational completeness of formal encryption. In: CRYPTO'03. LNCS, vol. 2729, pp. 530–547 (2003). https://doi.org/10.1007/978-3-540-45146-4_31
15. Goldreich, O.: Foundations of Cryptography, vol. I - Basic Tools. Cambridge Unievrsity Press (2001)
16. Goldreich, O.: Foundation of Cryptography, vol. II - Basic Applications. Cambridge Unievrsity Press (2004)
17. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. J. ACM **33**(4), 792–807 (1986). https://doi.org/10.1145/6490.6503
18. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Computer and System Sience **28**(2), 270–299 (1984). https://doi.org/10.1016/0022-0000(84)90070-9
19. Hajiabadi, M., Kapron, B.M.: Computational soundness of coinductive symbolic security under active attacks. In: TCC'10. LNCS, vol. 7785, pp. 539–558. Springer (2013). https://doi.org/10.1007/978-3-642-36594-2_30
20. Katz, J., Lindell, Y.: Introduction to Modern Cryptography (Cryptography and Network Security Series). Chapman & Hall/CRC (2007)

21. Kemmerer, R.A., Meadows, C.A., Millen, J.K.: Three system for cryptographic protocol analysis. J. Cryptology **7**(2), 79–130 (1994). https://doi.org/10.1007/BF00197942
22. Laud, P.: Encryption cycles and two views of cryptography. In: NORDSEC'02. pp. 85–100. No. 2002:31 in Karlstad University Studies (2002)
23. Laud, P., Corin, R.: Sound computational interpretation of formal encryption with composed keys. In: ICISC'03. LNCS, vol. 2971, pp. 55–66 (2003). https://doi.org/10.1007/b96249
24. Li, B., Micciancio, D.: Symbolic security of garbled circuits. In: Computer Security Foundations Symposium - CSF'18. pp. 147–161. IEEE (2018). https://doi.org/10.1109/CSF.2018.00018
25. Malkin, T., Micciancio, D., Miner, S.K.: Efficient generic forward-secure signatures with an unbounded number of time periods. In: EUROCRYPT'02. LNCS, vol. 2332, pp. 400–417. Springer (2002). https://doi.org/10.1007/3-540-46035-7_27
26. Micciancio, D.: Computational soundness, co-induction, and encryption cycles. In: EUROCRYPT'10. LNCS, vol. 6110, pp. 362–380. Springer (2010). https://doi.org/10.1007/978-3-642-13190-5_19
27. Micciancio, D.: Symbolic encryption with pseudorandom keys. Report 2009/249, IACR ePrint archive (2018), https://eprint.iacr.org/2009/249, version 201800223:160820
28. Micciancio, D., Panjwani, S.: Adaptive security of symbolic encryption. In: TCC'05. LNCS, vol. 3378, pp. 169–187. Springer (2005). https://doi.org/10.1007/b106171
29. Micciancio, D., Panjwani, S.: Corrupting one vs. corrupting many: the case of broadcast and multicast encryption. In: ICALP'06. LNCS, vol. 4052, pp. 70–82. Springer (2006). https://doi.org/10.1007/11787006_7
30. Micciancio, D., Panjwani, S.: Optimal communication complexity of generic multicast key distribution. IEEE/ACM Trans. on Networking **16**(4), 803–813 (2008). https://doi.org/10.1109/TNET.2007.905593
31. Micciancio, D., Warinschi, B.: Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. J. Computer Security **12**(1), 99–129 (2004). https://doi.org/10.3233/JCS-2004-12105
32. Micciancio, D., Warinschi, B.: Soundness of formal encryption in the presence of active adversaries. In: TCC'04. LNCS, vol. 2951, pp. 133–151 (2004). https://doi.org/10.1007/978-3-540-24638-1_8
33. Millen, J.K., Clark, S.C., Freedman, S.B.: The Interrogator: protocol security analysis. IEEE Transactions on Software Engineering **SE-13**(2), 274–288 (1987). https://doi.org/https://doi.org/10.1109/TSE.1987.233151
34. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. J. Computer Security **6**(1–2), 85–128 (1998). https://doi.org/10.3233/JCS-1998-61-205
35. Yao, A.C.: Protocols for secure computations (extended abstract). In: FOCS'82. pp. 160–164 (1982). https://doi.org/https://doi.org/10.1109/SFCS.1982.38