Symbolic Encryption with Pseudorandom Keys

Daniele Micciancio*

February 23, 2018

Abstract

We give an efficient decision procedure that, on input two (acyclic) cryptographic expressions making arbitrary use of an encryption scheme and a (length doubling) pseudorandom generator, determines (in polynomial time) if the two expressions produce computationally indistinguishable distributions for any pseudorandom generator and encryption scheme 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. We then prove that if any two (possibly cyclic) expressions are mapped to the same pattern, then the associated distributions are indistinguishable. At the same time, if the expressions are mapped to different symbolic patterns and do not contain encryption cycles, there are secure pseudorandom generators and encryption schemes for which the two distributions can be distinguished with overwhelming advantage.

Keywords: Symbolic security, computational soundness, completeness, pseudo-random generators, information leakage, greatest fixed points.

1 Introduction

Formal methods for security analysis (e.g., [14, 11, 25, 39, 40, 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.

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 [34, 4] combine encryption with other cryptographic

*University of California at San Diego, 9500 Gilman Dr., Mail Code 0404, La Jolla, CA 92093, USA. e-mail: daniele@cs.ucsd.edu. 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.

primitives (like pseudo-random 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.

Our results. In this paper we consider cryptographic expressions that make arbitrary (nested) use of encryption and pseudo-random generation, without imposing any syntactic restrictions on the messages transmitted by the protocols. In particular, we consider cryptographic expressions as those studied in [3], 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 pseudorandom keys obtained using a length doubling pseudorandom generator $k \mapsto \mathbf{G}_0(k); \mathbf{G}_1(k)$ that on input a single key k outputs a pair of (statistically correlated, but computationally independent) keys $\mathbf{G}_0(k)$ and $\mathbf{G}_1(k)$. The output of the pseudorandom generator can be used anywhere a key is allowed. In particular, pseudo-random keys $\mathbf{G}_0(k)$, $\mathbf{G}_1(k)$ can be used to encrypt messages, or as messages themselves (possibly encrypted under other random or pseudorandom keys), or as input to the pseudo-random generator. So, for example, one can iterate the application of the pseudorandom generator to produce an arbitrary long sequence of keys $\mathbf{G}_1(r), \mathbf{G}_1(\mathbf{G}_0(r)), \mathbf{G}_1(\mathbf{G}_0(\mathbf{G}_0(r))), \ldots$.

We remark that the usefulness of pseudorandom generators in cryptographic protocols is not limited to reducing the amount of randomness needed by cryptographic algorithms. 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 [8, 31], they are used in the best known (in fact, optimal [36]) multicast key distribution protocols [12] to compactly communicate (using a seed) a long sequence of pseudo-random 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 [42, 30].

Pseudo-random 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 pseudo-random 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 and Corollary 2) that the resulting symbolic semantics for cryptographic expressions is computationally sound, in the sense that if two expressions are symbolically equivalent, then for any (length regular) semantically secure encryption scheme and (length doubling) pseudo-random 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 4), showing that if two (acyclic[2]) 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.

[1] For example, $\mathbf{G}_0(k)$ 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 $\mathbf{G}_0(k')$ and compare the result to $\mathbf{G}_0(k)$.

[2] For cyclic expressions, i.e., expressions containing encryption cycles, our completeness theorem still holds, but with respect to a slightly weaker adversarial model based on least fixed point computations.

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 $k_1, \ldots, k_n$ is symbolically independent if no key k_i can be obtained from another k_j via the application of the pseudo-random generator. We show that this simple definition perfectly captures the intuition behind the computational notion of pseudo-randomness: 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 pseudo-random keys $\mathbf{G}_0(k)$ and $\mathbf{G}_1(k)$ 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 $(\mathbf{G}_0(k), \mathbf{G}_1(k))$ 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 $\mathbf{G}$ and encryption scheme $\{\!|\cdot|\!\}_k$ satisfying some very special properties (Lemma 7) 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 $\mathbf{G}$ using the classic construction of [18]. 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.

Related work. Cryptographic expressions with pseudo-random keys, as those considered in this paper, are used in the symbolic analysis of various cryptographic protocols, including multicast key distribution [36, 34, 35], cryptographically controlled access to XML documents [4], and (very recently) the symbolic analysis of Yao's garbled circuit construction for secure two party computation [30]. However, these works (with the exception of [30], which builds on the results from a preliminary version of our paper [32]) 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 [29], 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 [29] outside the random oracle model is unclear.

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, 37, 15]. 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 [19]. 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 [28]. The issue of completeness was first raised by Micciancio and Warinschi [37] 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 [15]). The notion of completeness used in [37, 2, 15] is different from the one studied in this paper. The works [37, 2, 15] 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 [19]. 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.

Circular security is a recurring theme in computationally sound symbolic security analysis, and in the last several years the problem has received renewed interest due to its application to the design of fully homomorphic encryption (FHE) schemes. Applications to computational soundness require a strong form of circular security ("key dependent message"- or KDM-security, [10]), where keys can be used to encrypt arbitrary subexpressions that may depend on the value of one or more keys. A computational soundness results using KDM secure encryption is proved in [6]. The standard definition of security (CPA security, or indistinguishability under chosen plaintext attack [19]) does not guarantee the security of a key k in this setting, and it is easy to build encryption schemes for which k is immediately recoverable from $\mathcal{E}_k(k)$. Similar separations between standard and circular security are given in [5, 13, 9] for cycles of length 2, in [41] for bit encryption schemes, and in [7, 20, 21, 23, 26, 27] for cycles of arbitrary length. These results show that one cannot assume, in general, that encryption schemes satisfying the traditional notion of CPA security are secure in the presence of key cycles or key dependent messages. In this paper, we focus on the standard definition of CPA security, which does not protect the encryption key in the presence of encryption cycles. In [3] the circular security problem is addressed by assuming that cryptographic expressions do not contain encryption cycles. In this paper we follow an alternative "co-inductive" approach [33] which proves security unconditionally (i.e., possibly in the presence of key cycles) by defining the adversarial knowledge by means of a greatest fixed point computation.

This paper is a major extension and revision of an earlier unpublished manuscript [32]. Our soundness results (Section 4) are similar to those already presented in [32], but we have substantially simplified the presentation by providing a simple direct definition of the key recovery function. (In [32], soundness results were formulated and proved using a rather complex algebraic framework to represent the adversarial knowledge.) Our completeness results (Section 5) are completely new, and constitute one of the main technical contributions of this work.

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 pseudo-random keys, and introduce an appropriate notion of key renaming. In Section 4 we present our symbolic semantics for cryptographic expressions with pseudorandom keys and prove that it is computationally sound. Finally, in Section 5, we justify the definitional choices made in Section 4 by proving a corresponding completeness result. Section 6 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, 33] for more background on the symbolic model, and [16, 17, 24] (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 $x;y \in \{0,1\}^{n+m}$.

2.1 Symbolic cryptography

In the symbolic setting, messages are described by abstract terms. For any sets of key and data terms **Keys**, **Data**, define **Exp**(**Keys**, **Data**) 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. As a notational convention, we assume that the pairing operation is right associative, and omit unnecessary parenthesis. E.g., we write $\{\!|d_1, d_2, d_3|\!\}_k$ instead of $\{\!|(d_1, (d_2, d_3))|\!\}_k$.

In [3, 33], $\mathbf{Keys} = \{k_1, \ldots, k_n\}$ and $\mathbf{Data} = \{d_1, \ldots, d_n\}$ are two flat sets of atomic keys and data blocks. In this paper, we consider pseudo-random keys, defined according to the grammar

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

where $\mathbf{Rand} = \{r_1, r_2, \ldots\}$ is a set of atomic key symbols (modeling truly random and independent keys), and $\mathbf{G}_0$, $\mathbf{G}_1$ represent the left and right half of a length doubling pseudo-random generator $k \mapsto \mathbf{G}_0(k); \mathbf{G}_1(k)$. Notice that grammar (2) allows for the iterated application of the pseudo-random generator, so that from any key $r \in \mathbf{Rand}$, one can obtain keys of the form $\mathbf{G}_{b_1}(\mathbf{G}_{b_2}(\ldots(\mathbf{G}_{b_n}(r))\ldots))$ for any $n \geq 0$, which we abbreviate as $\mathbf{G}_{b_1 b_2 \ldots b_n}(r)$. (As a special case, for $n = 0$, $\mathbf{G}_\epsilon(r) = r$.) For any set of keys $S \subseteq \mathbf{Keys}$, we write $\mathbf{G}^*(S)$ and $\mathbf{G}^+(S)$ to denote the sets

$$\begin{aligned} \mathbf{G}^*(S) &= \{\mathbf{G}_w(k) \mid k \in S, w \in \{0,1\}^*\} \\ \mathbf{G}^+(S) &= \{\mathbf{G}_w(k) \mid k \in S, w \in \{0,1\}^*, w \neq \epsilon\} \end{aligned}$$

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

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

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

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.

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 patters **Pat**(**Keys**, **Data**) is defined as

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

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 **Pat**(**Keys**, **Data**) as the smallest reflexive transitive binary relation such that

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

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

$$\mathbf{Parts}(e) = \{e' \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data}) \mid e' \sqsubseteq e\}. \tag{5}$$

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

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

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

In this section we describe the computational model used by modern cryptography, and recall the definition of the cryptographic primitives used in this paper. We also describe how cryptographic expressions are mapped to probability distributions in the computational model. This is all pretty standard, but we pinpoint some details that are especially important in our setting.

We assume that all algorithms and constructions take as an implicit input a (positive integer) security parameter ℓ. For simplicity, we let all algorithms use the same 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. But, when analyzing algorithms, running times and success probabilities should always be understood as functions of ℓ. We use standard asymptotic notation $f = O(g)$ or $g = \Omega(f)$ if $\lim_{\ell \to \infty} |f(\ell)/g(\ell)| < \infty$, and $f = \omega(g)$ or $g = o(f)$ if $\lim_{\ell \to \infty} f(\ell)/g(\ell) = 0$. A probability $\varepsilon(\ell)$ is *negligible* if $\varepsilon(\ell) = \ell^{-\omega(1)}$. We say that a probability $\delta(\ell)$ is *overwhelming* if $1 - \delta(\ell)$ is negligible. For any two functions f, g of the security parameter, we write $f \approx g$ if their difference $\epsilon(\ell) = f(\ell) - g(\ell)$ is negligible.

We say that an algorithm is *efficient* if it runs in time $\ell^{O(1)}$, polynomial in the security parameter. Unless otherwise stated, we assume that all probability distributions used in this paper are generated by efficient algorithms. An efficiently computable *predicate* is an algorithm $\mathcal{D}$ that outputs either 0 (reject/false) or 1 (accept/true). Two probability distributions $\mathcal{A}_0$ and $\mathcal{A}_1$ are *computationally indistinguishable* if for any efficient algorithm $\mathcal{D}$, $\Pr\{\mathcal{D}(x) \colon x \leftarrow \mathcal{A}_0\} \approx \Pr\{\mathcal{D}(x) \colon x \leftarrow \mathcal{A}_1\}$. Equivalently, $\mathcal{A}_0$ and $\mathcal{A}_1$ are computationally indistinguishable if $\Pr\{\mathcal{D}(x) = b \colon b \leftarrow \{0,1\}, x \leftarrow \mathcal{A}_b\} \approx 1/2$.

Cryptographic Primitives In the computational setting, cryptographic expressions evaluate to probability distributions over bit strings, and two expressions are considered 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 bit 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. We assume without loss of generality[3] that $\mathcal{G}$ is a *length doubling* pseudorandom generator, i.e., it always outputs strings of length 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* it if satisfies the following definition of *indistinguishability under chosen plaintext attack*. Informally, the definition requires that the distributions $\mathcal{E}(k,m)$ and $\mathcal{E}(k, 0^{|m|})$ are computationally indistinguishable, when k is chosen uniformly at random, and m is an arbitrary (adversarially chosen) message. 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 $\Pr\{\mathcal{A}^{O_b(\cdot)} = b\} \approx 1/2$, i.e., the adversary cannot determine the bit b with probability substantially better than $1/2$. 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, with the following modifications: (1) the encryption $\mathcal{E}(\mathrm{pk}, m)$ and decryption $\mathcal{D}(\mathrm{sk}, c)$ algorithms are given different keys, produced by a key generation algorithm $(\mathrm{sk}, \mathrm{pk}) \leftarrow \mathcal{K}$, and (2) the public key is given as input to the adversary $\mathcal{A}(\mathrm{pk})$ in the security definition of indistinguishability under chosen plaintext attack. In the public key setting, it may be assumed, without loss of generality, that the adversary makes a single query to the encryption oracle O_b. 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 $O_b(i,m) = \mathcal{E}(k_i, 0^{|m|})$.

We remark that secure encryption, as defined above, allows ciphertexts to leak partial information about the secret key. For example, for any secure encryption scheme $\mathcal{E}$ (using keys of length ℓ), one can define a modified scheme $\mathcal{E}'((k_0; k_1), m) = (k_0; \mathcal{E}(k_1, m))$ that uses keys of length 2ℓ, and leaks the first half of the key k_0, using only the second half k_1 for the actual encryption. Since no additional information is leaked about k_1, this still satisfies the standard definition of indistinguishability under chosen plaintext attack. Partial information leakage about the key allows, for example, to tell (with negligible probability of error) if two ciphertexts $\mathcal{E}(k_0, m_0)$ and $\mathcal{E}(k_1, m_1)$ use the same key $k_0 = k_1$, or independently chosen keys $k_0 \neq k_1$. Stronger definitions of security providing a form of key anonymity are also possible. But the basic definition of indistinguishability under chosen plaintext attack is the most widely used one, both for private and public key encryption. So, for simplicity, we focus on the standard basic definition, leaving the adaptation of our results to stronger definitions to future work.

Computational encryption also leaks the length of the message being encrypted. When encrypting messages of bounded size, message size can be hidden by padding the messages to their maximum length. But leaking information about the message is somehow unavoidable when encrypting arbitrarily long messages, as those generated by (1).

[3] A pseudorandom generator $\mathcal{G}^{(n)}$ with output length $\ell + n$ can be defined by setting $\mathcal{G}^{(1)}(x)$ to the first $\ell + 1$ bits of $\mathcal{G}(x)$, and $\mathcal{G}^{(n)}(x) = y_0; \mathcal{G}^{(1)}(y_1)$ for $n > 1$, where $(y_0; y_1) = \mathcal{G}^{(n-1)}(x)$, $|y_0| = n - 1$ and $|y_1| = \ell$.

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 pairing function[4] π used to encode pairs of strings.

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 $\mathbf{Keys} = \mathbf{G}^*(\mathbf{Rand})$ is a set of pseudo-random 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 pseudo-random keys in $\mathbf{G}^+(\mathbf{Rand})$ using a length doubling pseudo-random generator $\mathcal{G}$ according to the rule

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

It is easy to see that any two expressions $e, e' \in \mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$ 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 $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$, 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.

[4] We do not assume any specific property for the pairing function, 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 $w_1 \cdot w_2$, 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, $w_1 \cdot w_2$ could be the string concatenation of a prefix-free encoding of w_1, followed by w_2.

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. 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 $\mathbf{G}_0(r)$ should not be considered a valid key renaming because the probability distributions associated to $(r, \mathbf{G}_0(r))$ and $(\mathbf{G}_0(r), r)$ 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 $(\mathbf{G}_0(r), \mathbf{G}_1(r))$ to (r_0, r_1) should be allowed because $(\mathbf{G}_0(r), \mathbf{G}_1(r))$ 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.

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 $k_1, k_2 \in \mathbf{Keys}$, we say that k_1* yields *k_2 (written $k_1 \preceq k_2$) if $k_2 \in \mathbf{G}^*(k_1)$, i.e., k_2 can be obtained by repeated application of $\mathbf{G}_0$ and $\mathbf{G}_1$ 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 $i \neq j$.*

As an example, they keys $\mathbf{G}_0(r) \perp \mathbf{G}_{01}(r)$ are independent, but the keys $\mathbf{G}_0(r) \preceq \mathbf{G}_{10}(r)$ 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

$$\mathbf{Roots}(S) = S \setminus \mathbf{G}^+(S).$$

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.

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

[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.

[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.

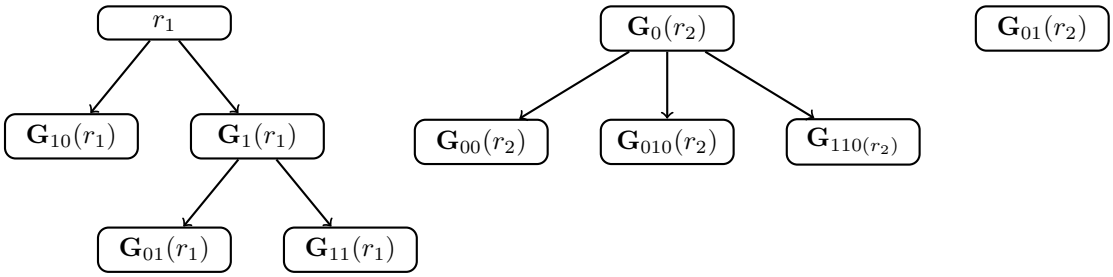

Figure 1: Hasse diagram associated to the set of keys $S = \{r_1, \mathbf{G}_{10}(r_1), \mathbf{G}_1(r_1), \mathbf{G}_{01}(r_1), \mathbf{G}_{11}(r_1), \mathbf{G}_0(r_2), \mathbf{G}_{00}(r_2), \mathbf{G}_{010}(r_2), \mathbf{G}_{110}(r_2), \mathbf{G}_{01}(r_2)\}$. For any two keys, $k_1 \preceq k_2$ if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

corresponds exactly to the standard cryptographic notion of computational pseudorandomness. We remark that the correspondence proved 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 [37, 15].

Theorem 1 *Let $k_1, \ldots, k_n \in \mathbf{Keys} = \mathbf{G}^*(\mathbf{Rand})$ be a sequence of symbolic keys. Then, for any secure (length doubling) pseudorandom generator $\mathcal{G}$, the probability distribution $[\![k_1, \ldots, k_n]\!]$ is computationally indistinguishable from $[\![r_1, \ldots, r_n]\!]$ (where $r_1, \ldots, r_n \in \mathbf{Rand}$ are distinct atomic keys), 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, $k_j = \mathbf{G}_w(k_i)$ 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 $\mathbf{G}_0$ and $\mathbf{G}_1$ 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 $k_j^i = \mathbf{G}_w(r) \in \mathbf{Keys} \setminus \mathbf{Rand}$ 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:

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

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 $\mathbf{G}$ if for all $k_1, k_2 \in S$ and $w \in \{0,1\}^*$,*

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

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 $(\mathbf{G}_0(r_0), \mathbf{G}_1(r_0))$ to $(r_0, \mathbf{G}_{001}(r_1))$ 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 $\mu(k_1) = \mu(k_2) = \mathbf{G}_\epsilon(\mu(k_2))$. By definition of pseudo-renaming, this implies $k_1 = \mathbf{G}_\epsilon(k_2) = k_2$. This proves that μ is a bijection from S to $\mu(S)$.

Now assume S is *not* an independent set, i.e., $k_1 = \mathbf{G}_w(k_2)$ for some $k_1, k_2 \in S$ and $w \neq \epsilon$. By definition of pseudo-renaming, we also have $\mu(k_1) = \mathbf{G}_w(\mu(k_2))$. 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 $\mu(k_1) = \mathbf{G}_w(\mu(k_2))$ for some $w \neq \epsilon$. Again, by definition of pseudo-renaming, $k_1 = \mathbf{G}_w(k_2)$, 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 $\mathbf{G}^*(S)$. In particular, any pseudo-renaming can be (uniquely) specified as the extension $\bar{\mu}$ of a bijection $\mu\colon A \to B$ between two independent sets $A = \mathbf{Roots}(S)$ and $B = \mu(A)$.*

Proof. Let $\mu\colon S \to \mathbf{Keys}$ be a pseudo-renaming. For any $w \in \{0,1\}^*$ and $k \in S$, define $\bar{\mu}(\mathbf{G}_w(k)) = \mathbf{G}_w(\mu(k))$. This definition is well given because μ is a pseudo-renaming, and therefore for any two representations of the same key $\mathbf{G}_w(k) = \mathbf{G}_{w'}(k') \in \mathbf{G}^*(S)$ with $k, k' \in S$, we have $\mathbf{G}_w(\mu(k)) = \mu(\mathbf{G}_w(k)) = \mu(Gen_{w'}(k')) = \mathbf{G}_{w'}(\mu(k'))$. 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 $\mathbf{G}^*(S) = \mathbf{G}^*(\mathbf{Roots}(S)) = \mathbf{G}^*(A)$. 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 $\mu\colon S \to \mathbf{Keys}$ cannot, in general, be extended to one over the set $\mathbf{Keys} = \mathbf{G}^*(\mathbf{Rand})$ of all keys. For example, $\mu\colon \mathbf{G}_0(r_0) \mapsto r_1$ 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 $\mu\colon A \to \mathbf{Keys}$ we have $\mu(\mathbf{Roots}(A)) = \mathbf{Roots}(\mu(A))$.*

Proof. By Lemma 1, μ is injective. Therefore, we have

$$\mu(\mathbf{Roots}(A)) = \mu(A \setminus \mathbf{G}^+(A)) = \mu(A) \setminus \mu(\mathbf{G}^+(A)).$$

From the defining property of pseudo-renamings we also easily get that $\mu(\mathbf{G}^+(A)) = \mathbf{G}^+(\mu(A))$. Therefore, $\mu(\mathbf{Roots}(A)) = \mu(A) \setminus \mathbf{G}^+(\mu(A)) = \mathbf{Roots}(\mu(A))$. □

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

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

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 *Equivalence up to pseudo-renaming ($\cong$) is a computationally sound relation, 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 *Let $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$ be any pattern, and $K = \{k_1, \ldots, k_n\} = \mathbf{Roots}(\mathbf{Keys}(e))$ be the set of root keys that occur in e, numbered according to the order in which they appear in e. The normalized pattern $\mathbf{Norm}(e) = \mu(e)$ is obtained by applying the pseudorandom key renaming $\mu(k_i) = r_i$ to e, where $r_1, \ldots, r_n$ is any fixed ordering of the keys of* **Rand**.

It immediately follows from the definition that $\mathbf{Norm}(e) \cong e$, 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 $\mathbf{Norm}(e_0) = \mathbf{Norm}(e_1)$ are identical.

3.3 Examples

We conclude this section with some illustrative examples. Let μ be the function mapping $\mathbf{G}_1(r_1) \mapsto \mathbf{G}_{01}(r_2)$, $\mathbf{G}_{10}(r_1) \mapsto r_1$ and $r_2 \mapsto \mathbf{G}_{11}(r_2)$. This is a pseudo-renaming because it is a bijection between two independent sets $\{\mathbf{G}_1(r_1), \mathbf{G}_{10}(r_1), r_2\}$ and $\{\mathbf{G}_{01}(r_2), r_1, \mathbf{G}_{11}(r_2)\}$. Consider the expression

$$e_6 = (\mathbf{G}_0(r_2), \{\!|r_2, \mathbf{G}_1(r_1)|\!\}_{\mathbf{G}_{10}(r_1)}, \{\!|d_1, d_2|\!\}_{\mathbf{G}_{11}(r_1)}) \tag{6}$$

with $\mathbf{Keys}(e_6) = \{\mathbf{G}_0(r_2), \mathbf{G}_1(r_1), \mathbf{G}_{10}(r_1), r_2, \mathbf{G}_{11}(r_1)\}$. The pseudo-renaming μ can be applied to e_6 because the domain of μ equals the set of root keys

$$\mathbf{Roots}(\mathbf{Keys}(e_6)) = \{\mathbf{G}_1(r_1), \mathbf{G}_{10}(r_1), r_2\}$$

and $\mathbf{Keys}(e_6) \subset \mathbf{G}^*(\mathbf{Roots}(\mathbf{Keys}(e_6)))$. So, μ can be extended to a function

$k \in \mathbf{Keys}(e_6)$	$\mathbf{G}_1(r_1)$	$\mathbf{G}_{10}(r_1)$	r_2	$\mathbf{G}_0(r_2)$	$\mathbf{G}_{11}(r_1)$
$\bar{\mu}(k)$	$\mathbf{G}_{01}(r_2)$	r_1	$\mathbf{G}_{11}(r_2)$	$\mathbf{G}_{011}(r_2)$	$\mathbf{G}_{101}(r_2)$

which, applied to expression e_6, yields

$$\mu(e_6) = (\mathbf{G}_{011}(r_2), \{\!|\mathbf{G}_{01}(r_2), r_1|\!\}_{\mathbf{G}_{11}(r_2)}, \{\!|d_1, d_2|\!\}_{\mathbf{G}_{101}(r_2)}).$$

Both expressions normalize to

$$\mathbf{Norm}(e_6) = \mathbf{Norm}(\mu(e_6)) = (\mathbf{G}_0(r_1), \{\!|r_1, r_2|\!\}_{r_3}, \{\!|d_1, d_2|\!\}_{\mathbf{G}_1(r_2)})$$

where $\mathbf{Roots}(\mathbf{Norm}(e_6)) = \{r_1, r_2, r_3\}$.

As another example, consider the expression

$$e_7 = (\{\!|d_1, \mathbf{G}_0(r_1)|\!\}_{\mathbf{G}_{01}(r_1)}, \mathbf{G}_{11}(r_1)). \tag{7}$$

We have $\mathbf{Roots}(\mathbf{Keys}(e_7)) = \{\mathbf{G}_0(r_1), \mathbf{G}_1(r_1)\}$. Let μ be the pseudo-renaming mapping $\mathbf{G}_0(r_1) \mapsto r_1$ and $\mathbf{G}_1(r_1) \mapsto r_2$. Then, expression e_7 is equivalent to $\mu(e_7) = (\{\!|d_1, r_1|\!\}_{\mathbf{G}_0(r_2)}, \mathbf{G}_1(r_2))$. The probability distributions associated to the two expressions

$$[\![e_7]\!] = [\![\{\!|d_1, \mathbf{G}_0(r_1)|\!\}_{\mathbf{G}_{01}(r_1)}, \mathbf{G}_{11}(r_1)]\!] \qquad [\![\mu(e_7)]\!] = [\![\{\!|d_1, r_1|\!\}_{\mathbf{G}_0(r_2)}, \mathbf{G}_1(r_2)]\!]$$

are statistically different (e.g., the support size of the second distribution is larger than that of the first one), but computationally indistinguishable. The expression $\mu(e_7)$ is the normalization of e_7), i.e., $\mathbf{Norm}(e_7) = \mathbf{Norm}(\mu(e_7)) = e_7$.

4 Cryptographic expressions with pseudo-random keys

In this section we present our main result: we give symbolic semantics for cryptographic expressions with pseudorandom keys, and prove that it is computationally sound. We begin by recalling the framework of [3, 33] to define computationally sound symbolic semantics of cryptographic expressions. Then, in Section 4.1 we show how to instantiate it to model cryptographic expressions with pseudorandom keys, and in Section 4.2 we prove that the resulting semantics is computationally sound. A justification for the definitional choices made in this section is provided in Section 5, where we prove a corresponding completeness theorem showing that our semantics precisely characterizes the computational indistinguishabilitly of (acyclic) cryptographic expressions with pseudorandom keys.

Following [3, 33], the symbolic semantics of cryptographic expressions $e \in \mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$ is defined by a set of known keys $S \subseteq \mathbf{Keys}(e)$ (to be specified) and a corresponding pattern $\mathbf{p}(e, S)$ obtained from e by replacing all undecryptable subexpression $\{\!|e'|\!\}_k \sqsubseteq e$ (where $k \notin S$) with a subpattern $\{\!|\mathtt{shape}(e')|\!\}_k$ that reveals only the shape of the encrypted message. The definition of $\mathbf{p}$ is standard, and it is formally specified in Figure 2. Notice that $\mathbf{p}$ is defined not just for expressions, but for arbitrary patterns $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$. Informally, for any expression or pattern $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$ and set of "known keys" $S \subseteq \mathbf{Keys}$, the pattern $\mathbf{p}(e, S)$ represents the result of observing e when using (only) the keys in S for decryption. We remark that the definition of $\mathbf{p}$ is identical to previous work [3, 33], as it treats pseudo-random keys $\mathbf{Keys} \subset \mathbf{G}^*(\mathbf{Rand})$ 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, 33], this function satisfies the properties

$$\mathbf{p}(e, \mathbf{Keys}) = e \tag{8}$$

$$\mathbf{p}(\mathbf{p}(e, S), T) = \mathbf{p}(e, S \cap T) \tag{9}$$

which can be informally stated by saying that $\mathbf{p}(\cdot, S)$ acts on patterns as a family of *projections*.

What is left to do is to define the set S of keys known to the adversary. Following [33], this involves the definition of a key recovery function $\mathbf{r}$, mapping patterns to sets of keys. Informally, $\mathbf{r}(e) \subseteq \mathbf{Keys}$ is the set of keys that can be (potentially) recovered combining the information obtained from all the parts of e. The definition of $\mathbf{r}$ is specific to our work, and it is given in Section 4.1. But before defining $\mathbf{r}$, we recall how it is used in [33] to define the set of adversarially known keys S, and the corresponding symbolic semantics $\mathbf{p}(e, S)$ for cryptographic expressions.

The key recovery function $\mathbf{r}$ is required to satisfy the monotonicity property

$$\mathbf{r}(\mathbf{p}(e, S)) \subseteq \mathbf{r}(e) \tag{10}$$

for any e and S, which, informally, says that projecting an expression (or pattern) e does not increase the amount of information recoverable from it. The function $\mathbf{p}$ and $\mathbf{r}$ are used to associate to each expression e a corresponding *key recovery operator*

$$\mathcal{F}_e(S) = \mathbf{r}(\mathbf{p}(e, S)) \tag{11}$$

mapping any $S \subseteq \mathbf{Keys}(e)$ to the set of keys $\mathcal{F}_e(S)$ that can be potentially recovered from the pattern $\mathbf{p}(e, S)$ observed by an adversary when using the keys in S for decryption. It is a easy consequence of (10)

that the key recovery operation $\mathcal{F}_e$ is monotone, i.e., if $S \subseteq S'$, then $\mathcal{F}_e(S) \subseteq \mathcal{F}_e(S')$, for any two sets of keys S, S'. It follows that $\mathcal{F}_e$ admits a least fixed point[7]

$$\text{fix}(\mathcal{F}_e) = \bigcup_n \mathcal{F}_e^n(\emptyset)$$

which can be algorithmically computed starting from the empty set of keys, and repeatedly applying $\mathcal{F}_e$ to it, recovering more and more keys

$$\emptyset \subset \mathcal{F}_e(\emptyset) \subset \mathcal{F}_e^2(\emptyset) \subset \mathcal{F}_e^3(\emptyset) \subset \cdots$$

until the fixed point $\mathcal{F}_e^n(\emptyset) = \mathcal{F}_e^{n+1}(\emptyset) = \text{fix}(\mathcal{F}_e)$ is reached and no more keys can be recovered. This inductive definition of known keys $S = \text{fix}(\mathcal{F}_e)$ is the traditional method to define the adversarial knowledge in symbolic security analysis, and the method used in [3] when proving a computational soundness result for acyclic expressions, i.e., expressions with no encryption cycles.

Here we follow a dual approach, put forward in [33], which defines the adversarial knowledge as the greatest fixed point $\text{FIX}(\mathcal{F}_e)$ (i.e., the largest set S such that $\mathcal{F}_e(S) = S$) and shows that this co-inductive approach results in a more precise connection between symbolic and computational cryptography. Informally, using the greatest fixed point corresponds to working by induction on the set of keys that are provably hidden from the adversary, starting from the empty set (i.e., assuming no a-priori guarantee that any key is secure), and showing that more and more keys are protected from the peering eyes of the adversary. Formulating this process in terms of the complementary set of "potentially known keys", one starts from the set of all keys $\mathbf{Keys}(e)$, and repeatedly applies the key recovery operator $\mathcal{F}_e$ to it. The result is a sequence of smaller and smaller sets

$$\mathbf{Keys}(e) \supset \mathcal{F}_e(\mathbf{Keys}(e)) \supset \mathcal{F}_e^2(\mathbf{Keys}(e)) \supset \mathcal{F}_e^3(\mathbf{Keys}(e)) \supset \cdots$$

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

$$\text{FIX}(\mathcal{F}_e) = \bigcap_n \mathcal{F}_e^n(\mathbf{Keys}(e)),$$

i.e., the largest set of keys that are guaranteed to be unknown to an adversary that observes the expression e. In summary, the symbolic semantics of an expression e is defined as

$$\mathbf{Pattern}(e) = \mathbf{p}(e, \text{FIX}(\mathcal{F}_e)) \tag{12}$$

where $\text{FIX}(\mathcal{F}_e)$ is the greatest fixed point of $\mathcal{F}_e$, representing the keys (potentially) known to the adversary. It is easy to see that any fixed point $S = \mathcal{F}_e(S)$ necessarily satisfies $\text{fix}(\mathcal{F}_e) \subseteq S \subseteq \text{FIX}(\mathcal{F}_e)$. In general, $\text{fix}(\mathcal{F}_e)$ may be a strict subset of $\text{FIX}(\mathcal{F}_e)$, but [33, Theorem 2] shows that if e is an acyclic expressions, then $\text{fix}(\mathcal{F}_e) = \text{FIX}_e(\mathcal{F}_e)$ and $\mathcal{F}_e$ has a unique fixed point. In summary, the inductive and co-inductive approaches produce the same patterns for acyclic expressions. Based on [33, Theorem 2], we generalize the definition of acyclic expressions to include all expressions e such that $\mathcal{F}_e$ has a unique fixed point $\text{fix}(\mathcal{F}_e) = \text{FIX}_e(\mathcal{F}_e)$.

The main result of [33], which we will use in our work, is given by the following soundness theorem.

Theorem 2 ([33, Theorem 1]) *Let* $\mathbf{p}\colon \mathbf{Pat} \times (\mathbf{Keys}) \to \mathbf{Pat}$ *and* $\mathbf{r}\colon \mathbf{Pat} \to \wp(\mathbf{Keys})$ *be any two functions satisfying (8), (9) and (10). Then, for any expression* $e \in \mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$*, the key recovery operator* $\mathcal{F}_e$ *defined in (11) is monotone, and admits a greatest fixed point* $FIX(\mathcal{F}_e) = \bigcap_{n \geq 0} \mathcal{F}_e^n(\mathbf{Keys})$*. Moreover, if, for any* $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$*, the distributions* $[\![e]\!]$ *and* $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ *are computationally indistinguishable, then the symbolic semantics* $\mathbf{Pattern}(e) = \mathbf{p}(e, FIX(\mathcal{F}_e))$ *is computationally sound, in the sense that the distributions* $[\![e]\!]$ *and* $[\![\mathbf{Pattern}(e)]\!]$ *are computationally indistinguishable.*

The reader is referred to [33] for a proof of the theorem, and further discussion about fixed points and the use of induction versus co-induction.

[7] We recall that a fixed point for $\mathcal{F}_e$ is a set of keys $S \subseteq \mathbf{Keys}(e)$ such that $S = \mathcal{F}_e(S)$.

$$
\begin{array}{rcl}
\mathbf{p}(d, S) & = & d \\
\mathbf{p}(k, S) & = & k \\
\mathbf{p}((e_1, e_2), S) & = & (\mathbf{p}(e_1, S), \mathbf{p}(e_2, S)) \\
\mathbf{p}(\{\!|e|\!\}_k, S) & = & \begin{cases} \{\!|\mathtt{shape}(e)|\!\}_k & \text{if } k \notin S \\ \{\!|\mathbf{p}(e, S)|\!\}_k & \text{if } k \in S \end{cases}
\end{array}
$$

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

4.1 Key recovery with pseudorandom keys

In order to instantiate the framework of [33] and use Theorem 2, we need to specify an appropriate knowledge recovery function $\mathbf{r}\colon \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data}) \to \wp(\mathbf{Keys})$ describing the set of keys $\mathbf{r}(e)$ that an adversary may be able to extract from all the parts of e. In the standard setting, where keys are atomic symbols, and encryption is the only cryptographic primitive, $\mathbf{r}(e)$ can be simply defined as the set of keys appearing in e as a message, i.e., $\mathbf{r}(e) = \mathbf{Keys}(e) \cap \mathbf{Parts}(e)$. 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 $\mathbf{r}(e)$ to include all keys k such that either

1. e contains k as a subexpression (directly revealing the value of k), or
2. e contains both k as an encryption key (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, $\mathbf{r}(e)$ contains all other keys that can be derived using the pseudorandom generator $\mathbf{G}$.

Definition 6 (Key Recovery Function) *For any expression or pattern e, let $\mathbf{r}(e) = \mathbf{Keys}(e) \cap \mathbf{G}^*(K)$ where*

$$K = \mathbf{Keys}(e) \cap (\mathbf{Parts}(e) \cup \mathbf{G}^-(\mathbf{Keys}(e))) = \mathbf{PKeys}(e) \cup (\mathbf{Keys}(e) \cap \mathbf{G}^-(\mathbf{Keys}(e))).$$

The expression $\mathbf{Keys}(e) \cap \mathbf{G}^*(K)$ simply extends the set of known keys K using the pseudorandom generator. The interesting part of Definition 6 is the set K. This 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 $k' = \mathbf{G}_{101}(k)$ is derived form k using the (one-way) functions $\mathbf{G}_0, \mathbf{G}_1$. In Section 5 we justify our definition showing that it leads to a symbolic semantics which is optimal, in the sense that for any two (acyclic) expressions with different patterns, there are computationally secure encryption schemes and pseudorandom generators for which the corresponding probability distributions can be efficiently distinguished. So, the power granted by Definition 6 to the symbolic adversary is necessary (and sufficient) to achieve computational soundness with respect to any valid instantiation of the cryptographic primitives.

We illustrate the definition of $\mathbf{r}$ by providing a few examples:

$$
\begin{array}{rcl}
\mathbf{r}(\{\!|r_1|\!\}_{\mathbf{G}_0(r_1)}, \{\!|\mathbf{G}_0(r_2)|\!\}_{\mathbf{G}_1(r_2)}) & = & \{r_1, \mathbf{G}_0(r_1), \mathbf{G}_0(r_2)\} \\
\mathbf{r}(\mathbf{G}_1(r_1), \{\!|d|\!\}_{\mathbf{G}_0(r_1)}, \mathbf{G}_0(r_2), \{\!|d|\!\}_{r_2}) & = & \{\mathbf{G}_1(r_1), \mathbf{G}_0(r_2), r_2\} \\
\mathbf{r}(\{\!|\mathbf{G}_0(r_1)|\!\}_{\mathbf{G}_1(r_2)}, \{\!|r_1|\!\}_{r_2}) & = & \{\mathbf{G}_1(r_2), \mathbf{G}_0(r_1), r_1, r_2\}
\end{array}
$$

In the first example, $r_1, \mathbf{G}_0(r_2)$ are (potentially) recoverable because they appear as a message. Therefore, also $\mathbf{G}_0(r_1)$ is (potentially) recoverable applying $\mathbf{G}_0$ to r_2. On the other hand, $\mathbf{G}_1(r_2)$ is not recoverable because it only appears as an encryption subscript, and it is symbolically independent from the other keys. In the second example, r_2 can be (potentially) recovered by combining partial information about it from $\mathbf{G}_0(r_2)$ and $\{\!|d|\!\}_{r_2}$. However, $\mathbf{G}_0(r_1)$ is unrecoverable because it is symbolically independent from $\mathbf{G}_1(r_1)$. In the last example, all keys can be potentially recovered. Notice that the definition of $\mathbf{r}$ does not take encryption into account, and assumes the adversary has potential access to all parts of a given expression, even undecryptable ones. Hiding undecryptable expressions from the adversary is taken care of by $\mathbf{p}$ in the definition of the key recovery operator (11), before $\mathbf{r}$ is applied.

It is easy to see that $\mathbf{r}$ satisfies property (10).

Lemma 4 *Let* $\mathbf{r}$ *as in Definition 6. Then,* $\mathbf{r}(\mathbf{p}(e, S)) \subseteq \mathbf{r}(e)$ *for any pattern* e.

Proof. For any patter e, the set $\mathbf{r}(e)$ depends only on the sets $\mathbf{Keys}(e)$ and $\mathbf{PKeys}(e)$. Moreover, this dependence is monotone. Since $\mathbf{Keys}(\mathbf{p}(e, S)) \subseteq \mathbf{Keys}(e)$ and $\mathbf{PKeys}(\mathbf{p}(e, S)) \subseteq \mathbf{PKeys}(e)$, by monotonicity we get $\mathbf{r}(\mathbf{p}(e, S)) \subseteq \mathbf{p}(e)$. □

We will use the fact that the functions $\mathbf{r}$ and $\mathbf{p}$ commute with pseudo-renamings.

Lemma 5 *For any expression or pattern* $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$ *and pseudo-renaming* $\mu\colon S \to \mathbf{G}^*(\mathbf{Rand})$ *such that* $\mathbf{Keys}(e) \subseteq S$*, we have*

$$\mathbf{r}(\mu(e)) = \mu(\mathbf{r}(e)).$$

Proof. This is a simple consequence of $\mathbf{Keys}(\mu(e)) = \mu(\mathbf{Keys}(e))$, $\mathbf{Parts}(\mu(e)) = \mu(\mathbf{Parts}(e))$, and the fact that μ preserves the $\prec$ relation. □

Lemma 6 *For any pattern* $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$*, set of keys* S*, and pseudo-renaming* $\mu\colon A \to \mathbf{G}^*(\mathbf{Rand})$ *such that* $\mathbf{Keys}(e) \cup S \subseteq A$*, we have*

$$\mu(\mathbf{p}(e, S)) = \mathbf{p}(\mu(e), \mu(S)).$$

Proof. The proof is by induction on the structure of e. The first three cases are simple:

- If $e = d \in D$, then $\mu(\mathbf{p}(d, S)) = d = \mathbf{p}(\mu(d), \mu(S))$.
- If $e = k \in K \cup \{\circ\}$, then $\mu(\mathbf{p}(k, S)) = \mu(k) = \mathbf{p}(\mu(k), \mu(S))$.
- If $e = (e_1, e_2)$, then using the induction hypothesis we get $\mu(\mathbf{p}((e_1, e_2), S)) = \mathbf{p}(\mu(e_1, e_2), \mu(S))$.

We are left with the case when $e = \{\!|e'|\!\}_k$ is a ciphertext. There are two possibilities, depending on whether $k \in S$ or not. Notice that μ is injective on $\mathbf{G}^*(A)$. So, for any $k \in \mathbf{Keys}(e) \subset \mathbf{G}^*(A)$ and $S \subset \mathbf{G}^*(A)$, we have $k \in S$ if and only if $\mu(k) \in \mu(S)$. Therefore,

- if $k \notin S$, then $\mu(\mathbf{p}(\{\!|e'|\!\}_k, S)) = \{\!|\mathtt{shape}(e')|\!\}_{\mu(k)} = \mathbf{p}(\mu(\{\!|e'|\!\}_k), \mu(S))$, and
- if $k \in S$, then $\mu(\mathbf{p}(\{\!|e'|\!\}_k, S)) = \{\!|\mu(\mathbf{p}(e', S))|\!\}_{\mu(k)} = \mathbf{p}(\mu(\{\!|e'|\!\}_k), \mu(S))$,

where in the second case we have also used the induction hypothesis. □

4.2 Computational Soundness

We are now ready to prove that the greatest fixed point semantics $\mathbf{Pattern}(e) = \mathbf{p}(e, \mathrm{FIX}(\mathcal{F}_e))$ is computationally sound with respect to any secure computational interpretation $(\mathcal{G}, \mathcal{E}, \gamma, \pi)$, as defined in Section 2.

Theorem 3 *For any secure computational interpretation and any expression $e \in \mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$, the distributions $[\![e]\!]$ and $[\![\mathbf{Pattern}(e)]\!]$ are computationally indistinguishable.*

Proof. In order to prove the theorem, it is enough to check that the conditions in Theorem 2 are satisfied. We already observed that the function $\mathbf{p}$ defined in Figure 2 satisfies (8) and (9). Moreover, by Lemma 4, the function $\mathbf{r}$ satisfies (10). It remains to check that for any $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$, the probability distributions $[\![e]\!]$ and $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ are computationally indistinguishable.

First of all, we claim that proving the lemma for arbitrary patterns e reduces to proving it for the special case of normal patterns, i.e., patterns such that $\mathbf{Roots}(e) \subset \mathbf{Rand}$. To establish the claim, consider an arbitrary pattern e with $\mathbf{Roots}(\mathbf{Keys}(e)) = \{k_1, \ldots, k_n\}$ and let μ be the pseudo-renaming mapping k_i to r_i, for n distinct keys $r_1, \ldots, r_n \in \mathbf{Rand}$. We know, from Corollary 1, that $[\![e]\!]$ is indistinguishable from $[\![\mu(e)]\!]$. But, by Lemma 3, $\mathbf{Roots}(\mu(e)) = \mu(\mathbf{Roots}(e)) = \{r_1, \ldots, r_n\} \subseteq \mathbf{Rand}$, i.e., $\mu(e)$ is normal. So, if we can prove the property for normal patterns, then $[\![\mu(e)]\!]$ is indistinguishable from $[\![\mathbf{p}(\mu(e), \mathbf{r}(\mu(e)))]\!]$. Using the commutativity properties from Lemma 6 and 5 we get

$$\mathbf{p}(\mu(e), \mathbf{r}(\mu(e))) = \mathbf{p}(\mu(e), \mu(\mathbf{r}(e))) = \mu(\mathbf{p}(e, \mathbf{r}(e))).$$

So, the distribution $[\![\mathbf{p}(\mu(e), \mathbf{r}(\mu(e)))]\!]$ is identical to $[\![\mu(\mathbf{p}(e, \mathbf{r}(e)))]\!]$. Finally, using Corollary 1 again, we see that $[\![\mu(\mathbf{p}(e, \mathbf{r}(e)))]\!]$ is indistinguishable from $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$. The indistinguishability of $[\![e]\!]$ from $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ follows by transitivity.

Let us now prove that $[\![e]\!]$ and $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ are computationally indistinguishable for any normal patterns e. So, let e be any pattern satisfying $\mathbf{Roots}(\mathbf{Keys}(e)) \subseteq \mathbf{Rand}$. Consider the pattern $e' = \mathbf{p}(e, \mathbf{r}(e))$. We want to prove that $[\![e']\!]$ is indistinguishable from $[\![e]\!]$. The pattern e' is obtained from e by replacing all subexpressions of e of the form $\{\!|e''|\!\}_k$ with $k \notin \mathbf{r}(e)$ by $\{\!|\mathtt{shape}(e'')|\!\}_k$.

Let $K = \mathbf{Keys}(e) \setminus \mathbf{r}(e)$ be the set of all keys not in $\mathbf{r}(e)$, and let $k \in K$ be an arbitrary element of K. Notice that k must necessarily satisfy the following properties:

1. $k \in \mathbf{Rand}$, i.e., k is a truly random key. This is because, otherwise, using the fact that e is normal, we would have $r \prec k$ for some $r \in \mathbf{Rand} \cap \mathbf{Keys}(e)$. So, by definition of $\mathbf{r}$, we would also have $r \in \mathbf{r}(e)$, $k \in \mathbf{r}(e)$, and $k \notin K$.

2. $k \in \mathbf{Keys}(e) \setminus \mathbf{Parts}(e)$, i.e., k appears in e only as an encryption key, and never as a message. Again, this is because otherwise we would have $k \in \mathbf{r}(e)$ and $k \notin K$.

3. $\mathbf{G}^+(k) \cap \mathbf{Keys}(e) = \emptyset$, i.e., no key in $\mathbf{G}^+(k)$ appears anywhere in e. As before, this is because, otherwise, we would have $k \in \mathbf{r}(e)$ and $k \notin K$.

Using these properties, we see that the probability distribution $[\![e]\!]$ can be efficiently sampled without knowing any of the keys in $K \subseteq \mathbf{Rand}$, provided we have access to $|K|$ encryption oracles, one for every $r \in K$. Moreover, if instead of properly encrypting the query messages m, the oracles encrypt $0^{|m|}$, then the same evaluation algorithm produces a sample from $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$. So, any algorithm to distinguish $[\![e]\!]$ from $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ can be turned into an algorithm to break the indistinguishability of $\mathcal{E}$ under chosen plaintext attacks. □

An easy consequence of Theorem 3 is that the equivalence relation defined by normalized patterns is computationally sound.

Corollary 2 *For any two expressions e_1, e_2, if $\mathbf{Norm}(\mathbf{Pattern}(e_1)) = \mathbf{Norm}(\mathbf{Pattern}(e_2))$ then $[\![e_1]\!]$ and $[\![e_2]\!]$ are computationally indistinguishable.*

Proof. If $\mathbf{Norm}(\mathbf{Pattern}(e_1)) = \mathbf{Norm}(\mathbf{Pattern}(e_2))$, then $\mathbf{Pattern}(e_1) \cong \mathbf{Pattern}(e_2)$, and $\mathbf{Pattern}(e_1) = \mu(\mathbf{Pattern}(e_2))$ for some pseudo-renaming μ. By Theorem 3, the probability distribution $[\![e_i]\!]$ is computationally indistinguishable from $[\![\mathbf{Pattern}(e_i)]\!]$, for $i = 1, 2$. Moreover, by Corollary 1, $[\![\mathbf{Pattern}(e_2)]\!]$ is indistinguishable from $[\![\mu(\mathbf{Pattern}(e_2))]\!] = [\![\mathbf{Pattern}(e_1)]\!]$. Therefore, by transitivity, we get that the distributions

$$[\![e_1]\!] \approx [\![\mathbf{Pattern}(e_1)]\!] = [\![\mu(\mathbf{Pattern}(e_2))]\!] \approx [\![\mathbf{Pattern}(e_2)]\!] \approx [\![e_2]\!]$$

are computationally indistinguishable. □

5 Completeness

In this section we prove a converse to our main soundness result (Theorem 3 and Corollary 2), showing that if two (acylic) expressions have different patterns, then the corresponding 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 $\Pr\{\mathcal{D}([\![e_1]\!])\} \approx 1$. So, the symbolic equivalence notion put forward in Theorem 3 and Corollary 2 is the best possible one for which computational soundness is guaranteed with respect to any computationally secure interpretation. In other words, if we want a cryptographic application to be computationally secure with respect to any instantiation of the cryptographic primitives satisfying standard security requirements, then the symbolic equivalence condition specified in Corollary 2 is both necessary and sufficient for computational indistinguishability.

As in previous work on the completeness of symbolic semantics [37, 15], we do this by modeling the adversarial knowledge as the *least fixed point* of the key recovery operator, yielding the pattern

$$\begin{aligned} \mathbf{pattern}(e) &= \mathbf{p}(e, \mathrm{fix}(\mathcal{F}_e)), \\ \mathrm{fix}(\mathcal{F}_e) &= \bigcup_{n \geq 0} \mathcal{F}_e^n(\emptyset). \end{aligned}$$

We recall that this produces exactly the same patterns as the greatest fixed point semantics $\mathbf{Pattern}(e) = \mathbf{p}(e, \mathrm{FIX}(\mathcal{F}_e))$ used in Section 4 when e is an expression without encryption cycles. In fact, for these expressions, the key recovery operator has a unique fixed point $\mathrm{fix}(\mathcal{F}_e) = \mathrm{FIX}(\mathcal{F}_e)$. But, just as greatest fixed points are more natural to use when proving soundness theorems [33], least fixed points are the best fit for completeness proofs [37, 15].

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 7 provides a tight computational justification for the symbolic key recovery function $\mathbf{r}$ described in Definition 6.

Lemma 7 *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 $k_1 = \mathbf{G}_w(k_0)$, 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 $k_1 \neq \mathbf{G}_w(k_0)$, 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. 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

$$(\mathbf{G}_{00}(k[0]), \mathbf{G}_{010}(k[0]), \mathbf{G}_{110}(k[0]), \mathbf{G}_{01}(k[0]), \mathbf{G}_{011}(k[0]), \mathbf{G}_{111}(k[0])) \tag{13}$$

and outputs the exclusive-or of this string with $(0; k[2]; k[2]; 0; k[2]; k[2])$. The expression (13) is evaluated using $\mathcal{G}'$. Since $\mathcal{G}'$ is a secure length doubling pseudorandom generator, and the keys in (13) are symbolically independent, by Theorem 1 expression (13) is mapped to a pseudorandom string of length 6ℓ. Finally, since taking the exclusive-or with any fixed string (e.g., $(0; k[2]; k[2]; 0; k[2]; k[2])$) 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:

$$\mathcal{G}_b(k)[0] = [\![\mathbf{G}_{0b}(k[0])]\!] \tag{14}$$

$$\mathcal{G}_b(k)[1] = [\![\mathbf{G}_{01b}(k[0])]\!] \oplus k[2] \tag{15}$$

$$\mathcal{G}_b(k)[2] = [\![\mathbf{G}_{11b}(k[0])]\!] \oplus k[2]. \tag{16}$$

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 to recover the partial information about the keys $k_0[0], k_0[1], k_1[0], k_1[1]$ leaked by $\mathcal{E}$. So, we will 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 $k_1 = \mathbf{G}_w(k_0)$, then the last key in this sequence equals $k^n = k_1$ as a string in $\{0, 1\}^\ell$.

Using (14), the first block $k^i[0]$ of these keys can be expressed symbolically as

$$k^i[0] = [\![\mathbf{G}_{u_i}(k_0[0])]\!] \qquad \text{where} \qquad u_i = 0w_i0w_{i-1} \dots 0w_1.$$

So, Algorithm $\mathcal{A}(k_0[0], k_0[1], k_1[0], k_1[1], w)$ begins by computing the value of all $k^i[0] = [\![\mathbf{G}_{u_i}(k_0[0])]\!]$ (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 $k_1 \neq \mathbf{G}_w(k_0)$, then $k^n[0] \neq k_1[0]$ with high probability, and $\mathcal{A}$ correctly outputs $\perp$. But for now, let us assume that $k_1 = \mathbf{G}_w(k_0)$, so that $k_1 = [\![\mathbf{G}_w(k_0)]\!] = k^n$ 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 (15) to compute $k^{n-1}[2]$ as

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

Similarly, starting from $k^{n-1}[2]$, $\mathcal{A}$ uses (16) to compute $k^i[2]$ for $i = n-2, n-3, \ldots, 0$ as

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

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 $k_1 = \mathbf{G}_w(k_0)$.

We need to show that if $k_1 \neq \mathbf{G}_w(k_0)$, then the probability that $k^n[0] = k_1[0]$ is negligible, so that $\mathcal{A}$ correctly outputs $\perp$. This is proved expressing $k^n[0]$ symbolically in terms of $k_0[0]$, and then using the symbolic characterization of computational independence from Section 3. The proof proceeds by cases, depending on whether $k_0 \in \mathbf{G}^*(k_1)$, $k_1 \in \mathbf{G}^*(k_0)$, or $\mathbf{G}^*(k_0) \cap \mathbf{G}^*(k_1) = \emptyset$, i.e., k_0 and k_1 are symbolically independent. Since we are interested only in the first blocks $k_0[0], k_1[1]$ of these two 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. Then, by construction, we have the following:

1. If $k_1 \in \mathbf{G}^*(k_0)$, then $k_1 = \mathbf{G}_v(k_0)$ for some string v, and $k_1[0] = \mathbf{G}_{0|v}(k_0[0])$.
2. Similarly, if $k_0 \in \mathbf{G}^*(k_1)$, then $k_0 = \mathbf{G}_v(k_1)$ for some string v, and $k_0[0] = \mathbf{G}_{0|v}(k_1[0])$.
3. Finally, $k_0 = \mathbf{G}_{v_0}(r_0)$ and $k_1 = \mathbf{G}_{v_1}(r_1)$ are symbolically independent (i.e., either $r_0 \neq r_1$, or v_0, v_1 are not one a prefix of the other,) then $k_0[0] = \mathbf{G}_{0|v_0}(r_0)$ and $k_1[0] = \mathbf{G}_{0|v_1}(r_1)$ are also symbolically independent (i.e., either $r_0 \neq r_1$, or $(0|v_0), (0|v_1)$ are not one a prefix of the other).

We need to show that in all three cases the probability that $k^n[0] = \mathbf{G}_{0|w}(k_0[0])$ and $k_1[0]$ evaluates to the same bitstring is negligible. We consider each case separately.

Case 3. Let $k_0[0]$ and $k_1[0]$ be symbolically independent. In this case, also $k^n[0] = \mathbf{G}_{0|w}(k_0[0])$ and $k_1[0]$ are symbolically independent. It follows, from Theorem 1, that the distribution $[\![\mathbf{G}_{0|w}(k_0[0]), k_1[0]]\!]$ 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 $k^n[0] = \mathbf{G}_{0|w}(k_0[0])$ and $k_1[0]$ evaluate to the same string is also negligible.

Case 2. Let $k_0[0] = \mathbf{G}_{0|v}(k_1[0])$ for some string v. Then, the pair of keys

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

is symbolically equivalent to $(\mathbf{G}_u(r), r)$ 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 $[\![\mathbf{G}_u(r)]\!]_\sigma$ evaluates to $[\![r]\!]_\sigma$. The trivial (identity) algorithm $\mathcal{I}(y) = y$ inverts the function defined by $\mathbf{G}_u$ with probability at least δ. Since $u \neq \epsilon$, $\mathbf{G}_u$ defines a one-way function, and δ must be negligible.

Case 1. Let $k_1[0] = \mathbf{G}_{0|v}(k_0[0])$ for some $v \neq w$. This time, we are given a pair of keys

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

which are symbolically equivalent to $(\mathbf{G}_{0|w}(r), \mathbf{G}_{0|v}(r))$. As before, by Theorem 1, it is enough to evaluate the probability δ that $\mathbf{G}_{0|w}(r)$ and $\mathbf{G}_{0|v}(r)$ 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 $\mathbf{G}_{0|w}(r)$ and $\mathbf{G}_{0|v}(r)$ 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 $\mathbf{G}_{u_n}(k_0[0])$ 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 will use Lemma 7 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 8 *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 the computationally sound symbolic semantics from Section 4.

Theorem 4 *For any two expressions e_0 and e_1, if $\mathbf{pattern}(e_0) \ncong \mathbf{pattern}(e_1)$, 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 8. 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 8 that this distinguisher is always correct, accepting all sample α from $[\![e_1]\!]$, and rejecting all samples α from $[\![e_0]\!]$.

The more interesting case is when $\texttt{shape}(e_0) = \texttt{shape}(e_1)$. 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 7, and show how to distinguish between samples from $[\![e_0]\!]$ and samples from $[\![e_1]\!]$, provided $\mathbf{pattern}(e_0) \ncong \mathbf{pattern}(e_1)$.

Let $S_b^i = \mathcal{F}_{e_b}(\emptyset)$ be the sequence of sets of keys defined by the key recovery operator associated to e_b. We know that $\emptyset = S_b^0 \subseteq S_b^1 \subseteq S_b^2 \subseteq \cdots \subseteq S_b^n = \text{fix}(\mathcal{F}_{e_b})$ 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, and $e_b^n = \mathbf{Pat}(e_b, \text{fix}(\mathcal{F}_{e_b})) = \mathbf{pattern}(e_b)$. Since $\mathbf{pattern}(e_0) \ncong \mathbf{pattern}(e_1)$, there is an index i such that $e_0^n \ncong e_1^n$. Let i the smallest such index. We will give a procedure that iteratively recovers all the keys in the sets $S_b^0, S_b^1, \ldots, S_b^i$, and then distinguishes between samples coming from the two distributions.

The simplest case is when $i = 0$, i.e., $e_0^0 \ncong e_1^0$. 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 $\{\!|m|\!\}_k$, without knowing if α comes from $[\![e_0]\!]$ or $[\![e_1]\!]$. These data blocks, keys and ciphertexts corresponds precisely to the atomic components of e_0^0 or e_1^1. If these two patterns 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. 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

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

Notice that if $r_0(p, p') = r_1(p, p')$ for all positions $p, p' \in P$, then $e_0^0 \cong e_1^0$. 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 ($r_{1-b}(p, p') = \perp$), 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 7 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, $k_0^{p'} = \mathbf{G}_w(k_0^p)$, while $k_1^{p'} \neq \mathbf{G}_w(k_0^p)$. 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 7 and check if it outputs a key or the special failure symbol $\perp$. The distinguisher accepts if and only if $\mathcal{A}(\alpha_0, \alpha_0', w) = \perp$. By Lemma 7, 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 $\mathcal{A}(\alpha_0, \alpha_0', w) = \perp$ 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 7 to recover the keys in S_b^1. Then we use these keys to decrypt the corresponding subexpressions in α, and use Lemma 7 again to recover all the keys in S_b^2. We proceed in a similar fashion all the way up to S_b^i. 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' will satisfy the same relations $r_0(p, p') = r_1(p, p')$, and we can apply Lemma 7 identically, whether the sample α comes from $[\![e_0]\!]$ or $[\![e_1]\!]$. This will allow 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 $e_0^i \not\cong e_1^i$, we proceed as in the case $i = 0$ to determine the value of b. $\square$

6 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 [33]) to expressions that mix encryption with a pseudo-random generator. Differently from previous work in the area of multicast key distribution protocols [34, 36, 35], 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 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 [30], which provides a computationally sound symbolic analysis of Yao's garbled circtuit construction for secure two party computation. The work of [30] illustrates the usefulness of the methods developed in this paper to the analysis of moderately complex protocols. 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 [38, 22], but probably more challenging.

References

[1] M. Abadi and A. Gordon. A calculus for cryptogaphic protocols: the spi calculus. *Information and Computation*, 148(1):1–70, Jan. 1999. Preliminary version in CCS 1997.

[2] M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In N. Kobayashi and B. Pierce, editors, *Proceedings of the 4th International Symposium on Theoretical Aspects of Computer Software - TACS 2001*, volume 2215 of *Lecture Notes in Computer Science*, pages 82–94, Sendai, Japan, Oct. 2001. Springer.

[3] M. Abadi and P. Rogaway. Reconciling two views of cryptography (The computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[4] M. Abadi and B. Warinschi. Security analysis of cryptographycally controlled access to XML documents. *Journal of the ACM*, 55(2):1–29, 2008. Prelim. version in PODS'05.

[5] T. Acar, M. Belenkiy, M. Bellare, and D. Cash. Cryptographic agility and its relation to circular encryption. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 403–422. Springer, 2010.

[6] P. Adão, G. Bana, J. Herzog, and A. Scedrov. Soundness and completeness of formal encryption: The cases of key cycles and partial information leakage. *Journal of Computer Security*, 17(5):737–797, 2009.

[7] N. Alamati and C. Peikert. Three's compromised too: Circular insecurity for any cycle length from (ring-)lwe. In M. Robshaw and J. Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 659–680. Springer, 2016.

[8] M. Bellare and S. K. Miner. A forward-secure digital signature scheme. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA,*

August 15-19, 1999, Proceedings, volume 1666 of *Lecture Notes in Computer Science*, pages 431–448. Springer, 1999.

[9] A. Bishop, S. Hohenberger, and B. Waters. New circular security counterexamples from decision linear and learning with errors. In T. Iwata and J. H. Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 776–800. Springer, 2015.

[10] J. Black, P. Rogaway, and T. Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2002.

[11] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London, Series A*, 426:233–271, 1989.

[12] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *INFOCOM 1999. Proceedings of the Eighteenth Annual Joint conference of the IEEE computer and communications societies*, volume 2, pages 708–716. IEEE, Mar. 1999.

[13] D. Cash, M. Green, and S. Hohenberger. New definitions and separations for circular security. In M. Fischlin, J. A. Buchmann, and M. Manulis, editors, *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, volume 7293 of *Lecture Notes in Computer Science*, pages 540–557. Springer, 2012.

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

[15] V. Gligor and D. O. Horvitz. Weak key authenticity and the computational completeness of formal encryption. In *Proceedings of CRYPTO '03*, volume 2729 of *LNCS*, pages 530–547. Springer, Aug. 2003.

[16] O. Goldreich. *Foundations of Cryptography*, volume I - Basic Tools. Cambridge Unievrsity Press, 2001.

[17] O. Goldreich. *Foundation of Cryptography*, volume II - Basic Applications. Cambridge Unievrsity Press, 2004.

[18] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[19] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sience*, 28(2):270–299, 1984. Preliminary version in Proc. of STOC 1982.

[20] R. Goyal, V. Koppula, and B. Waters. Separating IND-CPA and circular security for unbounded length key cycles. In S. Fehr, editor, *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*, volume 10174 of *Lecture Notes in Computer Science*, pages 232–246. Springer, 2017.

[21] R. Goyal, V. Koppula, and B. Waters. Separating semantic and circular security for symmetric-key bit encryption from the learning with errors assumption. In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 528–557, 2017.

[22] M. Hajiabadi and B. M. Kapron. *Computational Soundness of Coinductive Symbolic Security under Active Attacks*, pages 539–558. Springer, 2013.

[23] M. Hajiabadi and B. M. Kapron. Toward fine-grained blackbox separations between semantic and circular-security notions. In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 561–591, 2017.

[24] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[25] R. A. Kennerer, C. Meadows, and J. K. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[26] V. Koppula, K. Ramchen, and B. Waters. Separations in circular security for arbitrary length key cycles. In Y. Dodis and J. B. Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 378–400. Springer, 2015.

[27] V. Koppula and B. Waters. Circular security separations for arbitrary length cycles from LWE. In M. Robshaw and J. Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2016.

[28] P. Laud. Encryption cycles and two views of cryptography. In *NORDSEC 2002 - Proceedings of the 7th Nordic Workshop on Secure IT Systems*, number 2002:31 in Karlstad University Studies, pages 85–100, Karlstad, Sweden, Nov. 2002. Karlstad University Studies.

[29] P. Laud and R. Corin. Sound computational interpretation of formal encryption with composed keys. In *Information Security and Cryptology, 6th Int. Conf. – Proc. of ICISC'03*, volume 2971 of *LNCS*, pages 55–66, Seoul, Korea, Nov. 2003. Springer.

[30] B. Li and D. Micciancio. Symbolic security of garbled circuits. Cryptology ePrint Report 141, IACR, 2018.

[31] T. Malkin, D. Micciancio, and S. K. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 400–417. Springer, 2002.

[32] D. Micciancio. Pseudo-randomness and partial information in symbolic security analysis. Cryptology ePrint Report 249, IACR, 2009.

[33] D. Micciancio. Computational soundness, co-induction, and encryption cycles. In *Advances in Cryptology - Proceedings of EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 362–380. Springer, 2010.

[34] D. Micciancio and S. Panjwani. Adaptive security of symbolic encryption. In *Theory of Cryptography Conference – Proceedings of TCC'05*, volume 3378 of *LNCS*, pages 169–187. Springer, Feb. 2005.

[35] D. Micciancio and S. Panjwani. Corrupting one vs. corrupting many: the case of broadcast and multicast encryption. In *Proceedings of ICALP '06*, volume 4052 of *LNCS*, pages 70–82. Springer, July 2006.

[36] D. Micciancio and S. Panjwani. Optimal communication complexity of generic multicast key distribution. *IEEE/ACM Transactions on Networking*, 16(4):803–813, Aug. 2008. Preliminary version in Eurocrypt 2004.

[37] D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway logic of encrypted expressions. *Journal of Computer Security*, 12(1):99–129, 2004. Preliminary version in WITS'02.

[38] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Theory of Cryptography Conference, Proceedings of TCC 2004*, volume 2951 of *LNCS*, pages 133–151. Springer, 2004.

[39] J. K. Millen, S. C. Clark, and S. B. Freedman. The Interrogator: protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2):274–288, Feb. 1987.

[40] L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1–2):85–128, 1998.

[41] R. Rothblum. On the circular security of bit-encryption. In A. Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 579–598. Springer, 2013.

[42] A. C. Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164, 1982.