Pseudo-randomness and partial information in symbolic security analysis

Daniele Micciancio*

May 29, 2009

Abstract

We prove computational soundness results for cryptographic expressions with pseudo-random keys, as used, for example, in the design and analysis of secure multicast key distribution protocols. In particular, we establish a symbolic notion of independence (for pseudo-random keys) that exactly matches the standard computational security definition (namely, indistinguishability from the uniform distribution) for pseudo-random generators. As a conceptual contribution, we initiate the study of partial information in the context of computationally sound symbolic security analysis. Specifically, we show that (within our admittedly simple, but hopefully evocative setting) partial information can be taken into account in the symbolic model, in a computationally sound way, by simply annotating each key with a label which specifies that the key is either known, unknown, or partially known, without further details about the amount and type of partial information.

Keywords: Computational soundness, formal methods for security, pseudo-random generators, partial information, greatest fix-points.

1 Introduction

Formal methods for security analysis (e.g., [10, 7, 15, 22, 24, 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. For example, any deterministic cryptographic primitive f (like a one-way function, pseudo-random generator, or any other primitive modeled as an easily computable function), the value $f(x)$ always gives partial information about x, because one can easily check if $x = 0$ by computing $f(0)$ and comparing the result with $f(x)$. Moreover, standard cryptographic definitions, either for simplicity or efficiency reasons, often leave open the possibility of partial information leakage, even when such leakage could in principle be avoided. For example, the standard definition of security for encryption [14, 6] allows ciphertexts to reveal partial information about the encryption key, even if stronger definitions of security are possible [5] that guarantee key privacy. Finally, 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 with different cryptographic primitives.

In the last few years, starting with the seminal work of Abadi and Rogaway [2], there has been considerable progress in combining the symbolic and computational approaches to security protocol design and

*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 grants CNS-0430595 and CNS-0831536. 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.

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.

As an example, we briefly describe the result of [2], which also forms the basis of the study reported in this paper. The setting considered by Abadi and Rogaway [2] involves a passive (wiretapping) adversary that can monitor, but not alter, the messages transmitted by the legitimate parties executing a protocol. These messages are commonly described as syntactic terms, like the expression $(\{\!|k_2|\!\}_{k_1}, \{\!|m|\!\}_{k_2})$ which may model the transmission of a short term key k_2 encrypted under long term key k_1, followed by a payload m encrypted under the short term key. In [2] it is proved that if encryption is the only cryptographic primitive used in the expressions, then one can describe the adversarial knowledge simply as a set of keys (namely, the keys that can be fully recovered by the so-called Dolev-Yao [10] rules[1] while all other keys are treated as if they were completely secret. These recoverable keys are then used to map the expressions to an observed pattern, e.g., an expression with "holes" like $(\{\!|\circ|\!\}_{k_1}, \{\!|\square|\!\}_{k_2})$ that models the fact that certain subexpressions are hidden to the adversary. In the computational setting, security is defined in terms of computational indistinguishability: two sequences of expressions are computationally equivalent if the probability distributions associated to them are indistinguishable to any computationally bounded adversary. The symbolic semantics studied in [2] is computationally sound in the sense that (subject to certain syntactic restrictions, but see also [17] for an unconditional result) if two sequences of messages lead to the same pattern, then the corresponding probability distributions (obtained when the protocol is executed) are computationally indistinguishable.

The computational soundness result of [2] relies on the fact that when encryption is the only cryptographic primitive used in a protocol, then 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 may as well treat k as if it were completely hidden. Similar results to those of [2] are proved in [18] and [3] for cryptographic expressions that combine encryption with other cryptographic primitives, like pseudo-random generation and secret sharing, but under the assumption that all transmitted messages satisfy certain ad-hoc syntactic restrictions, which are sufficient to guarantee that different cryptographic primitives do not interfere with each other, and allow to bypass issues related to partial information leakage.

Our results. In this paper we consider cryptographic expressions that combine encryption and pseudo-random generation, but without imposing ad-hoc syntactic restrictions. In particular, we consider cryptographic expressions as those studied in [2], but with pseudo-random keys of the form $k = \mathbf{G}_0(\mathbf{G}_0(\mathbf{G}_1(r)))$, obtained using a length doubling pseudo-random generator. The pseudo-random generator $k \mapsto \mathbf{G}_0(k); \mathbf{G}_1(k)$ can be used to map a single key k to a pair of seemingly random ones $\mathbf{G}_0(k)$ and $\mathbf{G}_1(k)$, which in turns can be used in any context where a key is allowed. In particular, pseudo-random keys $\mathbf{G}_0(k)$, $\mathbf{G}_1(k)$ can be used to both to encrypt messages (possibly containing pseudo-random keys as their components), or as input to the pseudo-random generator itself. Pseudo-random keys allow to design more efficient protocols, where, for example, a single seed k is used to compactly communicate a longer sequence of pseudo-random keys $\mathbf{G}_0(k), \mathbf{G}_0(\mathbf{G}_1(k)), \mathbf{G}_0(\mathbf{G}_1(\mathbf{G}_1(k)))$, etc., as done for example in the best known (in fact, optimal [20]) multicast key distribution protocols [8].

As already remarked, pseudo-random generators (like any deterministic cryptographic primitive) inevitably leak partial information about their input. 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 (e.g., by computing $\mathbf{G}_0(k')$ and comparing the result to $\mathbf{G}_0(k)$). 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 also need to to model the possibility that given different pieces of partial information about a key one may be able to recover that key completely. Our main result shows how to do all this within a relatively simple symbolic model of

[1] A typical Dolev-Yao rule is that given a key k and the encryption $\{\!|m|\!\}_k$ of some message m under k, one can compute the plaintext m.

computation, and still obtain computational soundness. Our treatment of partial information is extremely simple and in line with the ideas of formal methods and symbolic analysis: we use an all-or-nothing approach to partial information, where each key or secret can be completely known, partially known, or completely hidden, without further details about the amount and type of partial information. Still, we demonstrate 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.

Beside the introduction of a simple symbolic model to deal with partial information, a key technical contribution of our paper is a syntactic characterization of independent keys that exactly matches its computational counterpart. 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 captures the intuition behind the computational notion of pseudo-randomness: we prove 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.

Related work. Cryptographic expressions with pseudo-random keys, as those considered in this paper, were previously used in the study of multicast key distribution protocols [20, 18, 19], but using ad-hoc methods and subject to various syntactic restrictions. Even more general (so called "composed") encryption keys were considered in [16], but only under the random oracle heuristics. We remark that the use of such general composed keys seems unjustifiable in the standard model of computation, and the significance of the results of [16] outside the random oracle model is unclear.

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 develop our method to treat partial information in the symbolic setting, and apply it to define computationally sound symbolic semantics for cryptographic expressions with pseudo-random keys.

2 Preliminaries

In this section we review standard notation and notions from symbolic and computational cryptography used in the rest of the paper. The reader is referred to [2, 17] for more background on the symbolic model, and any modern cryptography text (e.g., [12, 13]) for details regarding the computational model.

2.1 Symbolic cryptography

In the symbolic setting, messages are described as 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 concatenation of e_1 and e_2, and $\{\!\!\{e\}\!\!\}_k$ denotes the encryption of e under k. Usually, $\mathbf{Keys} = \{k_1, \ldots, k_n\}$ 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 $\mathbf{G}_0$, $\mathbf{G}_1$ are 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 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 $\mathbf{G}_{b_1 b_2 \ldots b_n}(r)$. We write $\{0,1\}^*$ to denote the set of all binary strings, and ϵ for the empty string. 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, once or more times. Using this notation, the set of keys generated by the grammar (2) can be written as $\mathbf{Keys} = \mathbf{G}^*(\mathbf{Rand})$.

The symbolic semantics of cryptographic expressions is defined by mapping them to *patterns*, which are expressions[2]

$$\mathbf{Pat}(\mathbf{Keys}, \mathbf{Data}) \subset \mathbf{Exp}(\mathbf{Keys} \cup \{\circ\}, \mathbf{Data} \cup \{\square\}) \tag{3}$$

over extended sets of key and data terms that include two special symbols $\circ, \square \notin \mathbf{Keys}, \mathbf{Data}$, denoting unknown keys or data respectively. The keys and parts of a pattern are defined by structural induction according to the usual rules

$$\begin{aligned} \mathbf{Keys}(d) &= \emptyset & \mathbf{Parts}(d) &= \{d\} \\ \mathbf{Keys}(k) &= \{k\} \cap \mathbf{Keys} & \mathbf{Parts}(k) &= \{k\} \\ \mathbf{Keys}(e_1, e_2) &= \mathbf{Keys}(e_1) \cup \mathbf{Keys}(e_2) & \mathbf{Parts}(e_1, e_2) &= \mathbf{Parts}(e_1) \cup \mathbf{Parts}(e_2) \\ \mathbf{Keys}(\{\!\!\{e\}\!\!\}_k) &= (\{k\} \cap \mathbf{Keys}) \cup \mathbf{Keys}(e) & \mathbf{Parts}(\{\!\!\{e\}\!\!\}_k) &= \{\{\!\!\{e\}\!\!\}_k\} \cup \mathbf{Parts}(e) \end{aligned}$$

where $d \in \mathbf{Data} \cup \{\square\}$, $k \in \mathbf{Keys} \cup \{\circ\}$, and $e, e_1, e_2 \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$. 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, while the special symbol $\circ$ is never included in $\mathbf{Keys}(e)$. The keys that appear in e as a message are given by $\mathbf{Keys}(e) \cap \mathbf{Parts}(e)$, and those that appear only as an encryption key are $\mathbf{Keys}(e) \setminus \mathbf{Parts}(e)$. As a notational convention, we assume that concatenation is left associative, and omit unnecessary parenthesis. We also omit the symbol $\circ$ when it appears as an encryption subscript. E.g., we write $\{\!\!\{d_1, d_2, d_3\}\!\!\}$ instead of $\{\!\!\{((d_1, d_2), d_3)\}\!\!\}_\circ$.

2.2 Computational model

In this section we informally recall the notions from computational cryptography used in this paper. These are all standard and well established. For reference, the formal definitions are reported in Appendix A. We also describe how cryptographic expressions are mapped to probability distributions. This is also standard, but we pinpoint some details that are especially important in our setting.

Cryptography. In the computational setting, cryptographic expressions are evaluated to probability distributions over bit-strings, and two expressions are equivalent if the associated distributions are computationally indistinguishable, in the sense that no probabilistic polynomial time algorithm can tell the difference between samples coming from one or the other with probability substantially better than 1/2.

A (length doubling) pseudo-random generator is a polynomial time algorithm $\mathcal{G}$ that on input a key k chosen uniformly at random among all strings of length ℓ, outputs a string $\mathcal{G}(k)$ of length 2ℓ. The generator is computationally secure if the output distribution is computationally indistinguishable from the uniform distribution over strings of length 2ℓ.

[2]Not all expressions in $\mathbf{Exp}(\mathbf{Keys} \cup \{\circ\}, \mathbf{Data} \cup \{\square\})$ are valid patterns. Formally, the set of patterns is defined as the image $\mathbf{p}(\mathbf{Exp}(\mathbf{Keys}, \mathbf{Data}), \wp(\mathbf{Keys}))$ of the function $\mathbf{p}$ given in Figure 2. The reader can safely ignore this technical detail, which is important only when mapping patterns to probability distributions over bit-strings.

A (symmetric) encryption scheme is defined as a pair of (probabilistic) polynomial time encryption and decryption algorithms $\mathcal{E}, \mathcal{D}$ such that $\mathcal{D}(k, \mathcal{E}(k, m)) = m$ for any message m and key k. Here the message m is an arbitrary string, and the key k is a (uniformly) random string of some fixed length ℓ that depends on the desired security level. The encryption scheme is secure under *chosen plaintext attack* if the probability distributions $\mathcal{E}(k, m)$ and $\mathcal{E}(k, 0^{|m|})$ are computationally indistinguishable when k is chosen at random, and m is an arbitrary message.

Computational evaluation. In the computational setting, cryptographic expressions are mapped 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 $\sigma : \mathbf{Keys} \to \{0,1\}^\ell$. The value $\sigma[\![e]\!]$ is defined by induction on the structure of the expression e by the rules $\sigma[\![d]\!] = \gamma_d$ (where γ_d is a fixed bit-string associated to $d \in \mathbf{Data}$), $\sigma[\![k]\!] = \sigma(k)$, $\sigma[\![(e_1, e_2)]\!] = \sigma[\![e_1]\!] \cdot \sigma[\![e_2]\!]$ (where $\cdot$ is a pairing function used to join two strings into one), and $\sigma[\![\{\!|e|\!\}_k]\!] = \mathcal{E}(\sigma(k), \sigma[\![e]\!])$ where all applications of the encryption algorithm $\mathcal{E}$ are performed using independent randomness. The computational evaluation $[\![e]\!]$ of an expression e is defined as the probability distribution obtained by first choosing a random key assignment σ 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 equation

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

Length conventions and pattern evaluation. Since computational encryption schemes are not usually required to hide the length of the input, it is natural to require that all functions operating on messages are length-regular, i.e., the length of the output depends only on the length of the input. Throughout the paper we assume that the functions $d \mapsto \gamma_d$, $(x_1, x_2) \mapsto x_1 \cdot x_2$ and $\mathcal{E}$ are length regular, i.e., $|\gamma_d|$ is the same for all $d \in \mathbf{Data}$, $|\sigma(k)| = \ell$ for all keys k, $|x_1 \cdot x_2|$ depends only on $|x_1|$ and $|x_2|$, and $|\mathcal{E}(k, x)|$ depends only on $|\sigma(k)| = \ell$ and $|x|$. Under these assumptions, it is easy to see that any two expressions $e, e' \in \mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$ with the same structure $\mathtt{struct}(e) = \mathtt{struct}(e')$ (see Figure 2 for the definition) are always evaluated to strings of exactly the same length $|\sigma[\![e]\!]| = |\sigma[\![e']\!]|$. Using this fact, the computational evaluation function $\sigma[\![e]\!]$ is extended to patterns[3] by defining

$$\sigma[\![\mathtt{struct}(e)]\!] = 0^{|\sigma[\![e]\!]|}.$$

Notice that the definition is well given because $|\sigma[\![e]\!]|$ depends only on $\mathtt{struct}(e)$, and not on the specific expression e.

3 Symbolic model for pseudo-random keys

In this section we develop a symbolic framework for the treatment of pseudo-random keys, and prove that it is computationally sound. First, in Section 3.1, we introduce a symbolic notion of independence for pseudo-random keys. Informally, two (symbolic) keys are independent if neither of them can be derived from the other through the application of the pseudo-random generator. We give a computational justification for this notion by showing 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 pseudo-random precisely when the keys $k_1, \ldots, k_n$ are symbolically independent. Then, in Section 3.2, we use our 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. 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 pseudo-random keys, where equivalent expressions evaluate to *identical* probability distributions.

[3] Here we define $[\![e]\!]$ only for valid patterns, i.e., patterns that are in the image of the function $\mathbf{p}$.

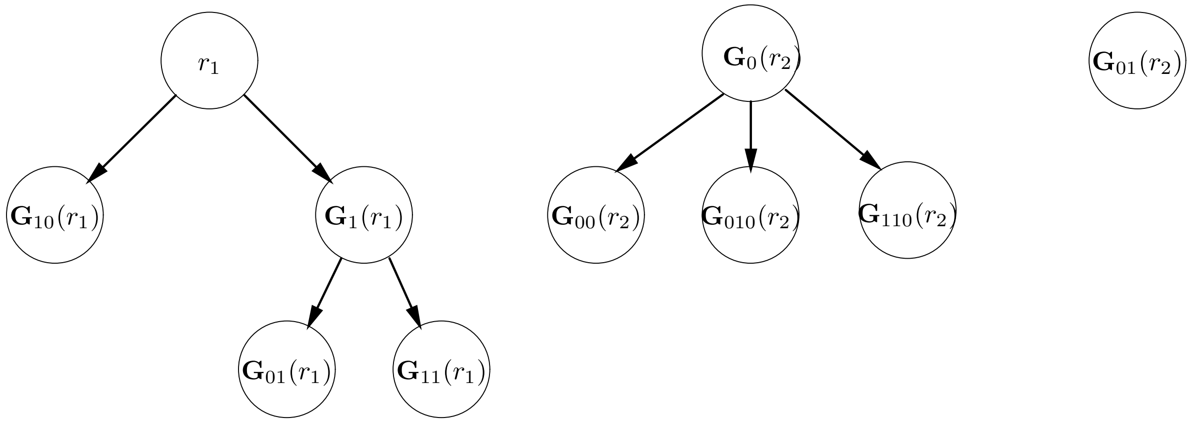

Figure 1: Hasse diagram associated to the set of keys $S = \{r_1, \mathbf{G}_{10}(r_1), \mathbf{G}_1(r_1), \mathbf{G}_{01}(r_1), \mathbf{G}_{11}(r_1), \mathbf{G}_0(r_2), \mathbf{G}_{00}(r_2), \mathbf{G}_{010}(r_2), \mathbf{G}_{110}(r_2), \mathbf{G}_{01}(r_2)\}$. For any two keys, $k_1 \preceq k_2$ if 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 $\mathbf{Roots}(S) = \{r_1, \mathbf{G}_0(r_2), \mathbf{G}_{01}(r_2)\}$.

3.1 Independence

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. As usual, we write $k_1 \prec k_2$ if $k_1 \preceq k_2$ and $k_1 \neq k_2$. Notice that $(\mathbf{Keys}, \preceq)$ is a partial order, i.e., the relation $\preceq$ is reflexive, antisymmetric and transitive. Two keys k_1, k_2 are *independent* 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 and k_j are independent for all $i \neq j$. Pictorially a set of keys $S \subseteq \mathbf{Keys}$ can be represented by the Hasse diagram[4] 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 pseudo-random keys $k_1, \ldots, k_n$ is pseudo-random, i.e., it is computationally indistinguishable from n truly random independently chosen keys. The following lemma shows that our symbolic notion of independence corresponds exactly to the standard cryptographic notion of computational pseudo-randomness. 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 [2], that only provide sufficient conditions for computational equivalence, and require additional work/assumptions to establish the completeness of the symbolic criterion [21, 11].

Theorem 1 *Let $k_1, \ldots, k_n \in \mathbf{Keys}$ be a sequence of symbolic keys. Then, for any secure (length doubling) pseudo-random 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.*

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,

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

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 pseudo-random 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 pseudo-random 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 pseudo-random 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 pseudo-random generator $\mathcal{G}$. The distinguisher $\mathcal{D}'$ for the pseudo-random generator $\mathcal{G}$ is given as input a pair of strings (σ', σ'') chosen either uniformly (and independently) at random or running the pseudo-random 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 to 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 $[\![K^{i+1}]\!]$. $\square$

3.2 Renaming pseudo-random keys

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 pseudo-random 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. A conservative approach would require a key renaming μ to act as a permutation over the set of atomic keys **Rand**. However, this is overly restrictive. We show that as long as the key renaming is compatible with the pseudo-random generator (in the sense specified in the following definition), more general key renamings can be allowed.

Definition 1 (Pseudo-renaming) *For any set of keys $S \subseteq \mathbf{Keys}$, a renaming $\mu: S \to \mathbf{Keys}$ is compatible with the pseudo-random 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 lemma follows almost immediately from the definition.

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 $\mu: S \to \mathbf{Keys}$ 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 $\mu(k_1) \preceq \mu(k_2) = \mathbf{G}_\epsilon(\mu(k_2))$ and $\mu(k_2) \preceq \mu(k_1) = \mathbf{G}_\epsilon(\mu(k_1))$. By definition of pseudo-renaming, this implies $k_1 \preceq \mathbf{G}_\epsilon(k_2) = k_2$ and $k_2 \preceq \mathbf{G}_\epsilon(k_1) = k_1$, which by the anti-symmetric property gives $k_1 = k_2$. 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. 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 bijections between two independent sets of keys, as shown in the following lemma.

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 $\mu: A \to B$ between two independent sets $A = \mathbf{Roots}(S)$ and $B = \mu(A)$.*

Proof. Let $\mu: S \to \mathbf{Keys}$ be a pseudo-renaming. For any $w \in \{0,1\}^*$ and $k \in S$, define $\bar{\mu}(\mathbf{G}_w(k)) = \mathbf{G}_w(\mu(k))$. 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 $\mathbf{G}_w(\mu(k)) = \mathbf{G}_{w'}(\mu(k'))$. 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 pseudorenamings 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 $B = \mu(A)$. □

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

Definition 2 *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 $\mu: \mathbf{Roots}(\mathbf{Keys}(e_1)) \to \mathbf{Roots}(\mathbf{Keys}(e_2))$ such that $\bar{\mu}(e_1) = \bar{\mu}(e_2)$.*

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 *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 $\mathcal{D}$. $\square$

3.3 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

$$e_4 = (\mathbf{G}_0(r_2), \{\!\!\{r_2\}\!\!\}_{\mathbf{G}_1(r_1), \mathbf{G}_{10}(r_1)}, \{\!\!\{d_1, d_2\}\!\!\}_{\mathbf{G}_{11}(r_1)}) \tag{4}$$

with $\mathbf{Keys}(e_4) = \{\mathbf{G}_0(r_2), \mathbf{G}_1(r_1), \mathbf{G}_{10}(r_1), r_2, \mathbf{G}_{11}(r_1)\}$. The pseudo-renaming μ can be applied to e_4 because the domain of μ equals the set of root keys

$$\mathbf{Roots}(\mathbf{Keys}(e_4)) = \{\mathbf{G}_1(r_1), \mathbf{G}_{10}(r_1), r_2\}$$

and $\mathbf{Keys}(e_4) \subset \mathbf{G}^*(\mathbf{Roots}(\mathbf{Keys}(e_4)))$. So, μ can be extended to a function

$k \in \mathbf{Keys}(e_4)$	$\mathbf{G}_1(r_1)$	$\mathbf{G}_{10}(r_1)$	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_4, yields

$$\bar{\mu}(e_4) = (\mathbf{G}_{011}(r_2), \{\!\!\{\mathbf{G}_{01}(r_2), r_1\}\!\!\}_{\mathbf{G}_{11}(r_2)}, \{\!\!\{d_1, d_2\}\!\!\}_{\mathbf{G}_{101}(r_2)}). \tag{5}$$

As another example, consider the expression

$$e_6 = (\{\!\!\{d_1, \mathbf{G}_0(r_1)\}\!\!\}_{\mathbf{G}_{01}(r_1)}, \mathbf{G}_{11}(r_1)). \tag{6}$$

We have $\mathbf{Roots}(\mathbf{Keys}(e_6)) = \{\mathbf{G}_0(r_1), \mathbf{G}_1(r_1)\}$. 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_6 is equivalent to $\bar{\mu}(e_6) = (\{\!\!\{d_1, r_1\}\!\!\}_{\mathbf{G}_0(r_2)}, \mathbf{G}_1(r_2))$. The probability distributions associated to the two expressions

$$[\![e_6]\!] = [\![\{\!\!\{d_1, \mathbf{G}_0(r_1)\}\!\!\}_{\mathbf{G}_{01}(r_1)}, \mathbf{G}_{11}(r_1)]\!] \qquad [\![\bar{\mu}(e_6)]\!] = [\![\{\!\!\{d_1, r_1\}\!\!\}_{\mathbf{G}_0(r_2)}, \mathbf{G}_1(r_2)]\!]$$

are statistically different (e.g., the support size of the second distribution is larger than that of the first one), but computationally indistinguishable.

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

Figure 2: Rules defining the pattern function $\mathbf{p}\colon \mathbf{Pat}(\mathbf{Keys},\mathbf{Data}) \times \wp(K) \to \mathbf{Pat}(\mathbf{Keys},\mathbf{Data})$ and auxiliary function $\mathtt{struct}\colon \mathbf{Pat}(\mathbf{Keys},\mathbf{Data}) \to \mathbf{Pat}(\emptyset,\emptyset)$, where $k \in K \cup \{\circ\}$, $d \in D \cup \{\square\}$, and $e, e_1, e_2 \in \mathbf{Pat}(\mathbf{Keys},\mathbf{Data})$. Intuitively, $\mathtt{struct}(e)$ represents structural information about e (e.g., its size) that may be leaked when encrypting e under standard computational encryption schemes, and $\mathbf{p}(e,T)$ is the pattern observable in e using the keys in T for decryption.

4 Cryptographic expressions with pseudo-random keys

In this section we prove our main result: the computational soundness of symbolic expressions with pseudo-random keys. As usual the symbolic semantics of cryptographic expressions is specified by means of a function $\mathbf{p}$ (defined, together with the auxiliary function $\mathtt{struct}$, in Figure 2) mapping each expression to a corresponding pattern. (The reader is referred to [2, 17] for motivation and discussion of these definitions.) 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 $\mathbf{p}(e,S)$ is the result of observing e using the keys in S for decryption. The set of keys known to a passive adversary observing e can be characterized as a fix-point of a key recovery operator $\mathcal{F}_e$, which informally maps any set of keys $S \subset \mathbf{Keys}$, to the set $\mathcal{F}_e(S)$ of keys which can be immediately recovered from e, given the ability to open any ciphertexts encrypted under keys in S. Traditionally, the adversarial knowledge in symbolic security analysis is defined as the least fix-point $\mathrm{fix}(\mathcal{F}_e)$ which corresponds to defining the set of known keys by induction. Here we follow a dual approach, put forward in [17], which takes the greatest fix-point $\mathrm{FIX}(\mathcal{F}_e)$ as the set of keys known to the adversary, and shows that this co-inductive approach results in a precise connection between symbolic and computational cryptography. The reader is referred to [17] for further discussion about fix-points and the use of induction versus co-induction.

Following [17], we introduce a key recovery function $\mathbf{r}$ (mapping patterns to sets of keys) such that the key recovery operator $\mathcal{F}$ can be expressed as

$$\mathcal{F}_e\colon S \mapsto \mathbf{r}(\mathbf{p}(e,S)). \tag{7}$$

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. In words, (7) says that $\mathcal{F}_e(S)$ is the set of keys that can be recovered from all the parts of the pattern $\mathbf{p}(e,S)$ obtained when observing e using the keys in S for decryption. We remark that the definition of $\mathbf{p}$ (given in Figure 2) treats pseudo-random keys $\mathbf{Keys} \subset \mathbf{G}^*(\mathbf{Rand})$ just as regular keys, disregarding their internal structure. In particular, no new definition is needed to extend $\mathbf{p}$ to expressions with pseudo-random keys, and here as in previous work the function $\mathbf{p}$ 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 informally state that $\mathbf{p}(\cdot,S)$ acts on patterns as a family of projection functions. The definition of the key recovery function $\mathbf{r}$ is specific to the class of cryptographic expressions under consideration, and is given in Section 4.2. All we need to know at this point is that the function $\mathbf{r}$ (defined in Section 4.2) satisfies the property

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

i.e., intuitively, projecting an expression (or pattern) e does not increase the amount of information recoverable from it. The projection properties (8-10) ensure that the key recovery operator $\mathcal{F}_e$ (7) is monotone, and therefore it admits both a least and a greatest fix-point, which can be computed as $\mathrm{fix}(\mathcal{F}_e) = \bigcup_n \mathcal{F}_e^n(\emptyset)$ and $\mathrm{FIX}(\mathcal{F}_e) = \bigcap_n \mathcal{F}_e^n(\mathbf{Keys})$. The (greatest fix-point) symbolic semantic of a cryptographic expression e is defined as the pattern

$$\mathbf{Pattern}(e) = \mathbf{p}(e, \mathrm{FIX}(\mathcal{F}_e)), \tag{11}$$

and two expressions e_1, e_2 are symbolically equivalent if they have the same pattern up to pseudo-renaming, i.e.,

$$\mathbf{p}(e_1, \mathrm{FIX}(\mathcal{F}_{e_1})) \cong \mathbf{p}(e_2, \mathrm{FIX}(\mathcal{F}_{e_2})).$$

The main result of [17] establishes a simple criterion for the computational soundness of the greatest fix-point semantics.

Theorem 2 ([17], Theorem 1) *Assume* $\mathbf{p}$ *and* $\mathbf{r}$ *satisfy (8–10). Then, the key recovery operator (7) is monotone, and the greatest fix-point semantic* $\mathbf{Pattern}(e) = \mathbf{p}(e, \mathit{FIX}(\mathcal{F}_e))$ *is well defined. Moreover, if the distributions* $[\![e]\!]$ *and* $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ *are computationally indistinguishable for any* $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$*, then the distributions* $[\![e]\!]$ *and* $[\![\mathbf{Pattern}(e)]\!]$ *are also computationally indistinguishable for any* $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$.

4.1 Modeling partial knowledge

In the standard setting, where keys are atomic symbols, and encryption is the only cryptographic primitive used, the set of keys $\mathbf{r}(e)$ that can be recovered from all parts of a pattern 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 multiple cryptographic primitives (e.g., as in this paper, both a pseudo-random generator and encryption function), one needs to describe partial information about keys, and how different pieces of partial information can be combined together. This can be done (symbolically) using an appropriate lattice[5] Λ, whose top element $\top$ denotes complete knowledge (of a key), and bottom element $\perp$ denotes complete lack of knowledge. Other elements of Λ can be used to symbolically quantify various amounts of partial knowledge. For example, $\Lambda = \{\perp, +, \top\}$ may consists of just three elements satisfying $\perp < + < \top$, where the value $+$ denotes some (unspecified) amount of partial information in-between $\perp$ and $\top$. We will show that even such a simple lattice is already enough to treat cryptographic expressions with encryption functions and pseudo-random keys. For concreteness, in the rest of the paper we fix $\Lambda = \{\top, \perp, +\}$ to this 3-element lattice, as this is all we need to establish our main result, but we remark that our definitions and proofs immediately extend to arbitrary lattices, which may be useful to deal with more complex settings, where, for example different encryption schemes are mixed together, etc.

We represent the adversarial knowledge as an *independent* set of keys S labeled with elements of Λ, which qualify the information known by the adversary about each key in S. For example, we write $\nu = \{k_1^\top, k_2^\top, k_3^+\}$ to represent complete knowledge about k_1 and k_2, together with partial information about k_3. Formally, the adversarial knowledge is represented as a function from S to Λ.

Definition 3 *For any set* $\mathbf{Keys} \subset \mathbf{G}^*(\mathbf{Rand})$ *and lattice* $(\Lambda, \leq)$*, let* $\mathcal{K}_\Lambda(\mathbf{Keys})$ *be the set of all functions* $\nu \colon S \to \Lambda \setminus \{\perp\}$ *such that the domain* $S \subseteq \mathbf{Keys}$ *is an* independent set.

We now define a partial order relation on $\mathcal{K}_\Lambda(\mathbf{G}^*(\mathbf{Keys}))$, where, intuitively, $\nu_1 \sqsubseteq \nu_2$ if ν_2 yields at least as much knowledge as ν_1 about any key.

[5] We remind the reader that a *lattice* is a partially ordered set $(\Lambda, \sqsubseteq)$ such that any nonempty finite subset $S \subseteq \Lambda$ admits a least upper bound $\bigsqcup S$ and greatest lower bound $\bigsqcap S$ with respect to the partial ordering relation $\sqsubseteq$. The operations $\bigsqcup$ and $\bigsqcap$ are called *join* and *meet*. Λ is a complete lattice if the same holds for arbitrary (possibly empty, or infinite) sets S. Any complete lattice has a top element $\top = \bigsqcup \Lambda = \bigsqcap \emptyset$ and bottom element $\perp = \bigsqcap \Lambda = \bigsqcup \emptyset$.

Definition 4 *For any $\nu \in \mathcal{K}_\Lambda(\mathbf{G}^*(\mathbf{Rand}))$, let f_ν be the function*

$$f_\nu(k) = \begin{cases} \nu(k) & \text{if } k \in S \\ \top & \text{if } k \in \mathbf{G}^+(\nu^{-1}(\top)) \\ \bot & \text{otherwise} \end{cases}$$

For any $\nu_1, \nu_2 \in \mathcal{K}_\Lambda(\mathbf{G}^*(\mathbf{Rand}))$, let $\nu_1 \sqsubseteq \nu_2$ if $f_{\nu_1}(k) \leq f_{\nu_2}(k)$ for all $k \in \mathbf{G}^*(\mathbf{Rand})$.

Below (Theorem 3) we show that $\sqsubseteq$ is a partial order relation, and, in fact, $\mathcal{K}_\Lambda(\mathbf{Keys})$ is a complete lattice. But first, we make the following remarks:

- We exclude the bottom element $\bot$ from the range of ν because lack of knowledge about a key k can be simply modeled by removing the key k from the domain S.
- We restrict the domain S of ν to be an independent set to ensure that all pieces of information are unrelated and the knowledge is represented concisely. This is taken into account in the definition of the function f_ν underlying the ordering relation $\sqsubseteq$. For example, if $\nu(k) = \top$, then $f_\nu^{-1}(\top)$ (the set of keys that can be completely recovered from ν) includes not only k, but also all keys $\mathbf{G}^*(k)$ obtained by repeatedly applying the pseudo-random generator to k.
- We do not assume that *partial* information about a key k allows to compute partial information about any other key in $\mathbf{G}^*(k)$. This captures the fact that there are computationally secure pseudo-random generators,[6] such that specific partial knowledge about k (e.g. as leaked when encrypting under k) does no yield any information about keys in $\mathbf{G}^+(k)$.
- Restricting the domain of ν to an independent set S introduces some constraints on the precise amount of information that can be described by elements of $\mathcal{K}_\Lambda(\mathbf{Keys})$. For example, there is no element in $\mathcal{K}_\Lambda(\mathbf{Keys})$ denoting precisely partial knowledge about two related keys, say, r^+ and $\mathbf{G}_0(r)^+$. This is intentional, and appears to be necessary to achieve computational soundness with respect to any computationally secure pseudo-random generator. We elaborate on this point below.

Theorem 3 *For any Λ, $(\mathcal{K}_\Lambda(\mathbf{G}^*(\mathbf{Rand})), \sqsubseteq)$ is a complete lattice. Moreover, for any $\mathbf{Keys} \subseteq \mathbf{G}^*(\mathbf{Rand})$, and nonempty set $\{\nu_i\}_{i \in I} \subseteq \mathcal{K}_\Lambda(\mathbf{Keys})$, we have $\bigsqcap_i \nu_i \in \mathcal{K}_\Lambda(\mathbf{Keys})$ and $\bigsqcup_i \nu_i \in \mathcal{K}_\Lambda(\mathbf{Keys})$.*

We will use the lattice $(\mathcal{K}(\mathbf{Keys}), \sqsubseteq)$ to study information about the keys $\mathbf{Keys}(e)$ that occur in an expression. Notice that the set $\mathcal{K}(\mathbf{Keys}(e))$ is finite, while $\mathcal{K}(\mathbf{G}^*(\mathbf{Rand}))$ is an infinite set. The second part of Theorem 3 shows that for any $\mathbf{Keys}$, the set $\mathcal{K}(\mathbf{Keys})$ is closed under the $\bigsqcup$ and $\bigsqcap$ operations of $\mathcal{K}(\mathbf{G}^*(\mathbf{Rand}))$. So, when studying an expression e, we can always restrict our attention to the finite sublattice $(\mathcal{K}(\mathbf{Keys}(e)), \sqsubseteq)$, rather than the entire $\mathcal{K}(\mathbf{G}^*(\mathbf{Rand}))$. The proof of Theorem 3 is given in Appendix B. Here we establish two simple corollaries that can be used to carry out computations in $\mathcal{K}_\Lambda(\mathbf{Keys}(e))$.

Corollary 2 *For any tuple of independent keys $k_1, \ldots, k_n \in \mathbf{Keys}$ and $\lambda_1, \ldots, \lambda_n \in \Lambda \setminus \{\bot\}$,*

$$\{k_1^{\lambda_1}\} \sqcup \cdots \sqcup \{k_n^{\lambda_n}\} = \{k_1^{\lambda_1}, \ldots, k_n^{\lambda_n}\}.$$

Proof. By Theorem 3, we have $\{k_1^{\lambda_1}\} \sqcup \cdots \sqcup \{k_n^{\lambda_n}\} \in \mathcal{K}_\Lambda(\{k_1, \ldots, k_n\})$. Since the keys $k_1, \ldots, k_n$ are independent, $\mathcal{K}_\Lambda(\{k_1, \ldots, k_n\})$ is precisely the set of all functions of the form $\nu : S \to \Lambda \setminus \{\bot\}$, where S is an arbitrary subset of $\{k_1, \ldots, k_n\}$. Any such a function satisfies $\{k_i^{\lambda_i}\} \sqsubseteq \nu$ if and only if $k_i \in S$, and $\nu(k_i) \geq \lambda_i$. Clearly the smallest function satisfying these properties for all i is the one mapping k_i to λ_i, i.e., $\nu = \{k_1^{\lambda_1}, \cdots, k_n^{\lambda_n}\}$. □

[6]Consider an encryption scheme $\mathcal{E}_k(m) = \mathcal{E}'_{k'}(m)$ that splits the key $k = k'; k''$ into two halves, uses the first half k' to encode the input message m using some other computationally secure encryption scheme $\mathcal{E}'$ (which accepts shorter keys than $\mathcal{E}$), and discards the second half of the key k''. Now consider a length-doubling pseudo-random generator $k \mapsto \mathcal{G}(k)$ that ignores the first half of the seed $k = k'; k''$, and expands the second half k'' by a factor 4. Clearly, the partial information about k leaked by $[\![\{\!|m|\!\}_k]\!]$ does not yield any information about any key in $[\![\mathbf{G}^+(k)]\!]$ because the two probability distributions depend on disjoint subsets of the key bits.

Corollary 3 *Let $\Lambda = \{\top, \bot, +\}$ and let $k_1 \prec k_2$ be two distinct, but related keys. Then, for any $\lambda_1, \lambda_2 \in \Lambda \setminus \{\bot\}$,*

$$\{k_1^{\lambda_1}\} \sqcup \{k_2^{\lambda_2}\} = \{k_1^{\top}\}.$$

Proof. By Theorem 3, we have $\{k_1^{\lambda_1}\} \sqcup \{k_2^{\lambda_2}\} \in \mathcal{K}_\Lambda(\{k_1^{\lambda_1}, k_2^{\lambda_2}\})$. Since $k_1 \prec k_2$, the lattice $\mathcal{K}_\Lambda(\{k_1, k_2\})$ contains only elements of the form $\{k_i^{\lambda}\}$, for $i = 1, 2$ and $\lambda \in \Lambda \setminus \{\bot\}$. It is easy to see that the only element in this set that is an upper bound to both $\{k_1^{\lambda_1}\}$ and $\{k_2^{\lambda_2}\}$ is $\{k_1^{\top}\}$, and therefore $\{k_1^{\lambda_1}\} \sqcup \{k_2^{\lambda_2}\} = \{k_1^{\top}\}$. □

The last corollary illustrates an important property of our lattice. Restricting the domain of ν to an independent set of keys implicitly forces the least upper bound operation to model the process of recovering a key from different pieces of partial information. For example, given partial information about $\mathbf{G}_1(r)$ (e.g., as leaked when using $\mathbf{G}_1(r)$ as an encryption key) and complete knowledge of $\mathbf{G}_{11}(r)$, one obtains the key $\mathbf{G}_1(r)$. In fact, it can be shown that there are encryption schemes and pseudo-random generators (satisfying the standard computational definitions of security) such that given the key $\mathbf{G}_{11}(r)$ and any message encrypted under $\mathbf{G}_1(r)$, one can efficiently break the encryption scheme, and recover the secret key $\mathbf{G}_1(r)$, even if neither $\mathbf{G}_{11}(r)$ nor the ciphertext allows individually to do so. Of course, this is not true for any computational encryption scheme and pseudo-random generator. Still, for the symbolic model to be computationally sound (with respect to any computational implementation satisfying the standard security definitions) we need to take into account the possibility that the key $\mathbf{G}_1(r)$ can be completely recovered.

We conclude with an example which illustrates how to compute the join of two elements of $\mathcal{K}_\Lambda(\mathbf{Keys})$. Let $\nu_1 = \{\mathbf{G}_{11}(r)^{\top}, \mathbf{G}_{00}(r)^{+}\}$ and $\nu_2 = \{\mathbf{G}_0(r)^{\top}, \mathbf{G}_1(r)^{+}\}$. We want to compute $\nu_1 \sqcup \nu_2$. Since the keys in each ν_i are independent, by Corollary 2 we have

$$\nu_1 \sqcup \nu_2 = \{\mathbf{G}_{11}(r)^{\top}\} \sqcup \{\mathbf{G}_{00}(r)^{+}\} \sqcup \{\mathbf{G}_0(r)^{\top}\} \sqcup \{\mathbf{G}_1(r)^{+}\}.$$

Next, using Corollary 3, we get

$$\begin{array}{rcl} \{\mathbf{G}_{11}(r)^{\top}\} \sqcup \{\mathbf{G}_1(r)^{+}\} & = & \{\mathbf{G}_1(r)^{\top}\} \\ \{\mathbf{G}_{00}(r)^{+}\} \sqcup \{\mathbf{G}_0(r)^{\top}\} & = & \{\mathbf{G}_0(r)^{\top}\} \end{array}$$

which combined together yields $\nu_1 \sqcup \nu_2 = \{\mathbf{G}_1(r)^{\top}\} \sqcup \{\mathbf{G}_0(r)^{\top}\} = \{\mathbf{G}_1(r)^{\top}, \mathbf{G}_0(r)^{\top}\}$.

4.2 Computational soundness

In order to instantiate the greatest fix-point framework for the symbolic equivalence of cryptographic expressions described at the beginning of this section, all we need to do is to specify the knowledge recovery function $\mathbf{r}\colon \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data}) \to \wp(\mathbf{Keys})$. Using the partial knowledge lattice $\mathcal{K}_\Lambda(\mathbf{Keys}(e))$, the definition of $\mathbf{r}$ is simple and natural. For each pattern $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$, define the information *immediately* revealed by e as

$$\Phi(e) = \begin{cases} \{k^{\top}\} & \text{if } e = k \in \mathbf{Keys} \\ \{k^{+}\} & \text{if } e = \{\!|m|\!\}_k \text{ for some } m \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data}) \\ \emptyset & \text{otherwise.} \end{cases}$$

In words, if a key is given in the clear, then the key can be completely recovered, while if it is used to encrypt, the resulting ciphertext yields partial information about it. We combine all pieces of information immediately recoverable from the parts of e and define

$$\mathbf{r}(e) = f_\nu^{-1}(\top) \qquad \text{where} \qquad \nu = \bigsqcup \Phi(\mathbf{Parts}(e))$$

as the set of all keys that can be completely recovered from ν. It is easy to see that $\mathbf{r}$ satisfies property (10), so it can be used to define a monotone key recovery operator $\mathcal{F}_e(S) = \mathbf{r}(\mathbf{p}(e, S))$, and instantiate the greatest fix-point semantics $\mathbf{Pattern}(e) = \mathbf{p}(e, \mathrm{FIX}(\mathcal{F}_e))$. The following theorem shows that this semantic is computationally sound.

Theorem 4 *Let $\mathcal{E}$ be a (length regular) encryption scheme semantically secure against chosen plaintext attack, and $\mathcal{G}$ a computationally secure length-doubling pseudo-random generator. For any two expressions $e_1, e_2 \in \mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$ such that $\mathbf{Pattern}(e_1) \cong \mathbf{Pattern}(e_2)$, the distributions $[\![e_1]\!]$ and $[\![e_2]\!]$ are computationally indistinguishable.*

Proof. In order to use Theorem 2 to establish the computational soundness for our semantics, we need the following technical lemma.

Lemma 3 *Under the assumptions of Theorem 4, for any $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$, the probability distributions $[\![e]\!]$ and $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ are computationally indistinguishable.*

Proof. First of all, we claim that proving the lemma for arbitrary patterns e reduces to proving it for the special case when $\mathbf{Roots}(e) \subset \mathbf{Rand}$. (In what follows, we refer to patterns such that $\mathbf{Roots}(e) \subset \mathbf{Rand}$ as "normal".) 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 $[\![\bar{\mu}(e)]\!]$. But, by Lemma 5 (see Appendix C), $\mathbf{Roots}(\mu(e)) = \mu(\mathbf{Roots}(e)) = \{r_1, \ldots, r_n\} \subseteq \mathbf{Rand}$, i.e., $\mu(e)$ is normal. So, if the lemma holds for normal patterns, then $[\![\mu(e)]\!]$ is indistinguishable from $[\![\mathbf{p}(\mu(e), \mathbf{r}(\mu(e)))]\!]$. It can be easily proved, by structural induction, that μ commutes with both $\mathbf{r}$ and $\mathbf{p}$ (see Lemma 6 and 7 in Appendix C) and therefore

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

So, the distribution $[\![\mathbf{p}(\bar{\mu}(e), \mathbf{r}(\bar{\mu}(e)))]\!]$ is identical to $[\![\bar{\mu}(\mathbf{p}(e, \mathbf{r}(e)))]\!]$. Finally, using Corollary 1 again, we see that $[\![\bar{\mu}(\mathbf{p}(e, \mathbf{r}(e)))]\!]$ 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 the lemma for the case of normal patterns e 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 $\{\!|\mathtt{struct}(e'')|\!\}_k$. Let $\nu = \bigsqcup \Phi(\mathbf{Parts}(e))$ so that $\mathbf{r}(e) = f_\nu^{-1}(\top)$. Consider any key $k \in \mathbf{Keys}(e) \setminus \mathbf{r}(e)$. Clearly, since $k \in \mathbf{Keys}(e)$, we have $\{k^+\} \sqsubseteq \nu$. Notice that $k \in \mathbf{Keys}(e) \setminus \mathbf{r}(e)$ must also satisfy the following properties:

- $k \in \mathbf{Rand}$. To see this, assume for contradiction $k \notin \mathbf{Rand}$. Then $k \notin \mathbf{Roots}(\mathbf{Keys}(e))$ because $\mathbf{Roots}(\mathbf{Keys}(e)) \subseteq \mathbf{Rand}$. So, $r \prec k$ for some $r \in \mathbf{Roots}(\mathbf{Keys}(e))$. Since $r \in \mathbf{Keys}(e)$, we have $\{r^+\} \sqsubseteq \nu$. It follows by Corollary 3 that $\{k^\top\} \sqsubset \{r^\top\} = \{r^+\} \sqcup \{k^+\} \sqsubseteq \nu$, and $k \in \mathbf{r}(e)$, a contradiction.

- $\mathbf{G}^*(k) \cap \mathbf{Parts}(e) = \emptyset$. To see this, assume for contradiction that there is some $\hat{k} \in \mathbf{G}^*(k) \cap \mathbf{Parts}(e)$. Then $\{\hat{k}^\top\} \sqsubseteq \nu$. Since $k \preceq \hat{k}$, by Corollary 3 we have $\{k^\top\} = \{k^+\} \sqcup \{\hat{k}^\top\} \sqsubseteq \nu$, and therefore $k \in \mathbf{r}(e)$, a contradiction.

- $\mathbf{G}^+(k) \cap \mathbf{Keys}(e) = \emptyset$. To see this, assume for contradiction that there is some $\hat{k} \in \mathbf{G}^+(k) \cap \mathbf{Keys}(e)$. Then $\{\hat{k}^+\} \sqsubseteq \nu$. Since $k \prec \hat{k}$, by Corollary 3 we have $\{k^\top\} = \{k^+\} \sqcup \{\hat{k}^\top\} \sqsubseteq \nu$, and therefore $k \in \mathbf{r}(e)$, a contradiction.

In summary, any $k \in \mathbf{Keys}(e) \setminus \mathbf{r}(e)$ is an atomic key that only appears as an encryption key in e, and no key in $\mathbf{G}^+(k)$ appears anywhere in e. So, the probability distribution $[\![e]\!]$ can be efficiently sampled without knowing the keys in $R' = \mathbf{Keys}(e) \setminus \mathbf{r}(e)$, provided we have access to $|R'|$ encryption oracles, one for every $r \in R'$. 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 semantic security of $\mathcal{E}$. □

Let us now go back to the proof of the main theorem. Let e_1, e_2 be two expressions such that $\mathbf{Pattern}(e_1) = \mu(\mathbf{Pattern}(e_2))$ for some pseudo-renaming μ. By Theorem 2 and Lemma 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, $[\![e_1]\!]$ and $[\![e_2]\!]$ are computationally indistinguishable. □

5 Conclusion

We presented a generalization of the computational soundness result of Abadi and Rogaway [2] (or, more precisely, its co-inductive variant put forward by the author in [17]) to expressions that mix encryption functions with a pseudo-random generator. Differently from previous work in the area of multicast key distribution protocols [18, 20, 19], 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. In our setting, we showed that partial information can be adequately represented symbolically in a very simple and intuitive way: the adversarial model is still modeled as a set of known keys, but each key is annotated with a label λ which equal either $\top$ (to represent that the key is completely known) or $+$ (to represent that only partial information is available about it.) The results can be easily generalized, using a larger set of labels, to deal with expressions that make use of even richer collections of cryptographic primitives, e.g., different types of (private and public key) encryption, secret sharing schemes (as used in [3]), and more.

As in [17], one of the major challenges at this point is to determine to what extent the ideas and results presented here can be transferred to more complex attack scenarios, like security against active attacks, as considered in [21], or even stronger models [4, 9, 23] supporting arbitrary composition of cryptographic primitives.

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

[3] 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.

[4] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Computer and Communications Security – Proceedings of CCS'03*, pages 220–230. ACM, Oct. 2003.

[5] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval. Key-privacy in public-key encryption. In C. Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, volume 2248 of *LNCS*, pages 566–582. Springer, Dec. 2001.

[6] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption: analysis of DES modes of operation. In *Proceedings of FOCS '97*, pages 394–403. IEEE, Oct. 1997.

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

[8] 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.

[9] R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In *Proceedings of TCC '06*, volume 3876 of *LNCS*, pages 380–403. Springer, Mar. 2006.

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

[11] 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.

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

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

[14] 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.

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

[16] 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.

[17] D. Micciancio. Computational soundness, co-induction and encryption cycles. Report 2009/269, IACR ePrint archive, 2009. URL http://eprint.iacr.org/2009/269.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for the analysis of cryptographic protocols. *Theoretical Computer Science*, 353(1-3):118–164, Mar. 2006. Preliminary version in MFPS'01.

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

A Cryptography

We briefly recall the formal definition of all computational cryptographic primitives used in this paper. For more information the reader is referred to [12, 13] or any other standard textbook on the subject.

Definition 5 *A (symmetric) encryption scheme is a pair of (probabilistic) polynomial time encryption and decryption algorithms $\mathcal{E}, \mathcal{D}$ such that $\mathcal{D}(k, \mathcal{E}(k, m)) = m$ for any message m and key k, where m is an arbitrary bit-string, and the key k is a (uniformly chosen random) string of some fixed length ℓ that depends on the desired security level. An encryption scheme $(\mathcal{E}, \mathcal{D})$ is indistinguishable under chosen plaintext attack if, for any probabilistic polynomial time adversary $\mathcal{A}$, the following holds. Choose a bit b and a key k of length ℓ uniformly at random and run $\mathcal{A}$ on input ℓ and with access to an encryption oracle $O_b(m)$ that outputs $\mathcal{E}(k, m)$ if $b = 1$, or $\mathcal{E}(k, 0^{|m|})$ if $b = 0$. The attacker $\mathcal{A}$ is required to run in time polynomial in the security* *parameter ℓ, and is supposed to guess the value of b. Then the quantity $|\Pr\{\mathcal{A}^{O_1}(\ell) = 1\} - \Pr\{\mathcal{A}^{O_0}(\ell) = 1\}|$ is negligible in the security parameter ℓ, i.e., it is smaller than $1/\ell^c$ for any constant c and sufficiently large ℓ.*

The definition of indistinguishability under chosen plaintext attack given above can be proved equivalent (via a standard hybrid argument) to a seemingly stronger definition where the attacker is given access to several encryption oracles, each encrypting under an independently chosen random key.

Definition 6 *An encryption scheme $(\mathcal{E}, \mathcal{D})$ is indistinguishable under chosen plaintext attack if, for any probabilistic polynomial time adversary $\mathcal{A}$ and polynomial p, the following holds. Choose a bit b and $n = p(\ell)$ keys $k_1, \ldots, k_n$ of length ℓ each, uniformly and independently at random and run $\mathcal{A}$ on input ℓ and with access to an encryption oracle $O_b(i, m)$ that outputs $\mathcal{E}(k_i, m)$ if $b = 1$, or $\mathcal{E}(k_i, 0^{|m|})$ if $b = 0$. The attacker $\mathcal{A}$ is required to run in time polynomial in the security parameter ℓ, and is supposed to guess the value of b. Then the quantity $|\Pr\{\mathcal{A}^{O_1}(\ell) = 1\} - \Pr\{\mathcal{A}^{O_0}(\ell) = 1\}|$ is negligible in the security parameter ℓ, i.e., it is smaller than $1/\ell^c$ for any constant c and sufficiently large ℓ.*

Computational equivalence between probability distributions over bit-strings is defined below.

Definition 7 *Let $\{A_i^0\}$ and $\{A_i^1\}$ be two probability ensembles, i.e., two sequences of probability distributions over bit-strings. $\{A_i^0\}$ and $\{A_i^1\}$ are computationally indistinguishable if for any probabilistic polynomial time adversary D,*

$$|\Pr\{D(A_i^0) = 1\} - \Pr\{D(A_i^1) = 1\}|$$

is negligible in i.

Definition 8 *A length doubling pseudo-random generator is a polynomial time algorithm $\mathcal{G}$ that on input a key k chosen uniformly at random among all strings of length ℓ, outputs a string $\mathcal{G}(k)$ of length 2ℓ. The pseudo-random generator $\mathcal{G}$ is secure if no adversary $\mathcal{A}$ can efficiently distinguish the output $\mathcal{G}(k)$ (when k is chosen as a random key of length ℓ) from a uniformly random string of length 2ℓ. Formally, for any probabilistic polynomial time algorithm $\mathcal{A}$, the quantity $|\Pr\{\mathcal{A}(\mathcal{G}(k)) = 1\} - \Pr\{\mathcal{A}(k_1; k_2) = 1\}|$ (where k, k_1 and k_2 are independently and uniformly chosen strings of length ℓ) is a negligible function of ℓ.*

B Proof of Theorem 3

In order to prove the theorem, we provide an alternative characterization of the elements $\nu \in \mathcal{K}_\Lambda(\mathbf{Keys})$ in terms of the functions f_ν used in the definition of the ordering relation $\sqsubseteq$ (see Definition 4).

Lemma 4 *A function $f\colon \mathbf{G}^*(\mathbf{Rand}) \to \Lambda$ equals $f = f_\nu$ for some $\nu \in \mathcal{K}_\Lambda(\mathbf{G}^*(\mathbf{Rand}))$ if and only if for any two related keys $k_1 \prec k_2$, one of the following (mutually exclusive) conditions holds:*

1. *$\bot \in \{f(k_1), f(k_2)\}$, or*
2. *$f(k_1) = f(k_2) = \top$.*

Moreover, if this is the case, then ν is the restriction of f to the set $\mathbf{Roots}(f^{-1}(\Lambda \setminus \{\bot\}))$.

It easily follows from Lemma 4 that the mapping $\nu \mapsto f_\nu$ is injective, and $\sqsubseteq$ is a partial order relation:

- *(Injectivity)* Assume $f_{\nu_1} = f_{\nu_2} = f$ for some $\nu_1, \nu_2 \in \mathbf{G}^*(\mathbf{Rand})$. Then, by Lemma 4, both ν_1 and ν_2 are the restriction of the same function f to the same set $\mathbf{Roots}(f^{-1}(\Lambda \setminus \{\bot\}))$. So, $\nu_1 = \nu_2$.
- *(Reflexivity)* This is a direct consequence of the definition of $\sqsubseteq$ and the reflexivity of $\leq$.
- *(Transitive property)* This is a direct consequence of the definition of $\sqsubseteq$ and the transitivity of $\leq$.

- *(Antisymmetry)* Assume $\nu_1 \sqsubseteq \nu_2 \sqsubseteq \nu_1$. From the definition of $\sqsubseteq$ and the antisymmetry of $\leq$, we get that $f_{\nu_1} = f_{\nu_2}$. Finally, using the injectivity of the mapping $\nu \mapsto f_\nu$, we conclude that $\nu_1 = \nu_2$.

We now prove that $\mathcal{K}_\Lambda(\mathbf{G}^*(\mathbf{Rand}))$ is a complete lattice. It is easy to see that $\mathcal{K}_\Lambda(\mathbf{G}^*(\mathbf{Rand}))$ has a top element $\top = \{r^\top : r \in \mathbf{Rand}\}$ and a bottom element $\bot = \emptyset$. Next we shows that any nonempty set $\{\nu_i\}_{i \in I} \subseteq \mathcal{K}_\Lambda(\mathbf{G}^*(\mathbf{Rand}))$ admits a greatest lower bound $\bigsqcap_{i \in I} \nu_i$. Define the function

$$f(k) = \inf_i f_{\nu_i}(k)$$

for all $k \in \mathbf{G}^*(\mathbf{Rand})$. We claim that f satisfies the properties in Lemma 4. To this end, let $k_1 \prec k_2$ be two distinct, but related keys. We distinguish two cases:

1. If $f_{\nu_i}(k_1) = f_{\nu_i}(k_2) = \top$ for all i, then $f(k_1) = f(k_2) = \top$.
2. Otherwise, it must be $\bot \in \{f_{\nu_i}(k_1), f_{\nu_i}(k_2)\}$ for some i. Say $f_{\nu_i}(k_j) = \bot$. It follows that $f(k_j) = \bot$ and $\bot \in \{f(k_1), f(k_2)\}$.

So, we can apply Lemma 4 and conclude that $f = f_\nu$ where ν is the restriction of f to the set

$$S = \mathbf{Roots}(f^{-1}(\Lambda \setminus \{\bot\})).$$

We claim that $\nu = \bigsqcap_i \nu_i$. Clearly, ν is a lower bound for all ν_i because for any k and i, $f_\nu(k) = f(k) = \inf_i f_{\nu_i}(k) \leq f_{\nu_i}(k)$. Now, let $\hat{\nu}$ be such that $\hat{\nu} \sqsubseteq \nu_i$ for all i. Then, $f_{\hat{\nu}}(k) \leq f_{\nu_i}(k)$ for all i, and therefore $f_{\hat{\nu}}(k) \leq \inf_i f_{\nu_i}(k) = f(k) = f_\nu(k)$. This proves that $\hat{\nu} \sqsubseteq \nu$, and ν is indeed a greatest lower bound. We also observe that if $\nu_i : S_i \to \Lambda \setminus \{\bot\}$, then $\nu = \bigsqcap_i \nu_i \in \mathcal{K}_\Lambda(\bigcup_i S_i)$. To this end, we consider any key $k \notin \bigcup_i S_i$, and prove that $k \notin S$. For any i, since $k \notin S_i$, it must be $f_{\nu_i}(k) \in \{\bot, \top\}$. We consider two cases:

1. If $f_{\nu_i}(k) = \bot$ for some i, then $f_\nu(k) = \inf_i f_{\nu_i}(k) = \bot$, and therefore $k \notin S$.
2. If $f_{\nu_i}(k) = \top$ for all i, then it must be $k \in \mathbf{G}^+(\nu_i^{-1}(\top))$. It follows that $k = \mathbf{G}_b(\hat{k})$ for some $b \in \{0, 1\}$, and $\hat{k}$ such that $f_{\nu_i}(\hat{k}) = \top$. Therefore, $f_\nu(\hat{k}) = \inf_i f_{\nu_i}(\hat{k}) = \top$, and $k \notin \mathbf{Roots}(f_\nu^{-1}(\Lambda \setminus \{\bot\})) = S$.

We now move to the existence of least upper bounds. For any nonempty set $\{\nu_i\}_{i \in I} \subseteq \mathcal{K}_\Lambda(\mathbf{G}^*(\mathbf{Rand}))$ define

$$\nu = \bigsqcap V \qquad \text{where} \qquad V = \{\hat{\nu} : \forall i \in I. \nu_i \sqsubseteq \hat{\nu}\}.$$

Since $\nu_i \sqsubseteq \hat{\nu}$ for all $\hat{\nu} \in V$, it follows that $\nu_i \sqsubseteq \bigsqcap V = \nu$, and ν is an upper bound to all ν_i. Finally, for any upper bound $\forall i. \nu_i \sqsubseteq \hat{\nu}$, we have $\nu = \bigsqcap V \sqsubseteq \hat{\nu}$ by definition, i.e., ν is the smallest of all upper bounds and $\nu = \bigsqcup \nu_i$. Also in this case we observe that if $\nu_i : S_i \to \Lambda \setminus \{\bot\}$ and $\nu : S \to \Lambda \setminus \{\bot\}$, then S must be a subset of $\bigcup_i S_i$. To see this, assume for contradiction that there exists a $k \in S \setminus \bigcup_i S_i$, and define another element $\hat{\nu}$ as follows:

- If $\nu(k) \neq \top$, then $\hat{\nu}$ is just the restriction of ν to $S \setminus \{k\}$.
- If $\nu(k) = \top$, then $\hat{\nu}$ is obtained by replacing $k^\top$ in ν with $\mathbf{G}_0(k)^\top, \mathbf{G}_1(k)^\top$.

It can be easily checked that $\hat{\nu} \sqsubset \nu$, and for all i, $\nu \sqsupseteq \nu_i$ implies $\hat{\nu} \sqsubseteq \nu_i$. So, $\hat{\nu}$ is an upper bound to all ν_i, and it is strictly smaller than ν, a contradiction. □

C Commutative properties

In this section we prove some simple commutative properties satisfied by pseudo-renamings.

Lemma 5 *For any pseudo-renaming $\mu : S \to$ **Keys** and set of keys $A \subseteq S$ we have $\mu(\mathbf{Roots}(A)) = \mathbf{Roots}(\mu(A))$.*

Proof. Since, by Lemma 1, μ is injective, 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, $\mu(\mathbf{Roots}(A)) = \mu(A) \setminus \mathbf{G}^+(\mu(A)) = \mathbf{Roots}(\mu(A))$. $\square$

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 $\mathbf{Keys}(e) \cup S \subseteq A$,*

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

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 $\mu(\mathbf{p}(\{\!|e'|\!\}_k, S)) = \{\!|\mathtt{struct}(e')|\!\}_{\mu(k)} = \mathbf{p}(\mu(\{\!|e'|\!\}_k), \mu(S))$, 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. $\square$

Lemma 7 *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. Recall that $\mathbf{r}(e) = f_\nu^{-1}(\top)$ where $\nu = \bigsqcup \Phi(\mathbf{Parts}(e))$. It is easy to see that $\mathbf{r}(e)$ can equivalently be defined as the set of all keys k such that $\{k^\top\} \sqsubseteq \bigsqcup \Phi(\mathbf{Parts}(e))$. Proceeding by induction on the structure of e, we get that $\mathbf{Parts}(\mu(e)) = \mu(\mathbf{Parts}(e))$, and therefore

$$\Phi(\mathbf{Parts}(\mu(e))) = \Phi(\mu(\mathbf{Parts}(e))) = \mu(\Phi(\mathbf{Parts}(e)))$$

where μ is extended to $\mathcal{K}_\Lambda(\mathbf{Keys})$ in the obvious way by the rule

$$\mu(\{k_1^{\lambda_1}, \ldots, k_n^{\lambda_n}\}) = \{\mu(k_1)^{\lambda_1}, \ldots, \mu(k_n)^{\lambda_n}\}.$$

Now notice that by the defining property of pseudo-renamings, the set of upper bounds to any $A \subseteq \mathcal{K}_\Lambda(\mathbf{Keys})$ is isomorphic to the set of upper bounds to $\mu(A)$ via the monotone mapping μ. It follows that $\bigsqcup \mu(A) = \mu(\bigsqcup A)$. Therefore

$$\begin{aligned}
\mu(\mathbf{r}(e)) &= \{\mu(k)\colon \{k^\top\} \sqsubseteq \textstyle\bigsqcup \Phi(\mathbf{Parts}(e))\} \\
&= \{\mu(k)\colon \mu(\{k^\top\}) \sqsubseteq \mu(\textstyle\bigsqcup \Phi(\mathbf{Parts}(e)))\} \\
&= \{\mu(k)\colon \{\mu(k)^\top\} \sqsubseteq \textstyle\bigsqcup \Phi(\mathbf{Parts}(\mu(e)))\} \\
&= \mathbf{r}(\mu(e)).
\end{aligned}$$

$\square$