# Computational soundness, co-induction, and encryption cycles

Daniele Micciancio[*]

June 1, 2009

**Abstract**

We analyze the relation between induction, co-induction and the presence of encryption cycles in the context of computationally sound symbolic equivalence of cryptographic expressions. Our main finding is that the use of co-induction in the symbolic definition of the adversarial knowledge allows to prove unconditional soundness results, that do not require syntactic restrictions, like the absence of encryption cycles. Encryption cycles are relevant only to the extent that the key recovery function associated to acyclic expressions can be shown to have a unique fix-point. So, when a cryptographic expression has no encryption cycles, the inductive (least fix-point) and co-inductive (greatest fix-point) security definitions produce the same results, and the computational soundness of the inductive definitions for acyclic expressions follows as a special case of the soundness of the co-inductive definition.

**Keywords:** Computational soundness, co-induction, greatest fix-points, formal methods for security, symbolic encryption, encryption cycles.

## 1 Introduction

The symbolic approach to security analysis (pioneered by Dolev and Yao in [12]) has been very useful in the construction and application of automated reasoning tools for the analysis of cryptographic protocols, like the Murphi model checker [27] and the Isabelle theorem prover [29], just to name two representative examples. However, the simplicity of the associated adversarial model (which enables the construction of automated analysis tools) is also the main weakness of symbolic security analysis: security is guaranteed only against attackers that abide to the rules of the Dolev-Yao model. In practice, one needs security against any (computationally feasible) attack as typically considered in modern computational cryptography. In the last few years, starting with the seminal work of Abadi and Rogaway [3], there has been considerable progress in understanding the relation between symbolic security analysis, and computational cryptography. Yet, it is fair to say that many problems related to the connection of symbolic and computational cryptography are still wide open.

The aim of this paper is to explore one specific aspect that sets the symbolic and computational models apart, and that has not received much attention so far: the use of induction versus co-induction in security proofs. We do so in the simplest possible setting considered in the literature: the indistinguishability of cryptographic expressions, i.e., expressions like $(\{\!|d_1|\!\}_{k_1}, \{\!|k_1|\!\}_{k_2})$, where $\{\!|m|\!\}_k$ represents the encryption of message $m$ under key $k$. These are the expressions typically used to model messages in cryptographic protocols. For example, the above expression may be used to represent the message in a protocol where a long term key $k_2$ is used to encrypt a session key $k_1$, which in turns is used to encrypt the actual message $d_1$. The standard notion of equivalence in cryptography is computational indistinguishability: two expressions are equivalent if no probabilistic polynomial time adversary can distinguish the probability distributions

[*] 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.

naturally associated to the two expressions in an actual execution of the protocol. In the symbolic setting, equivalence is usually defined by mapping each expression to a corresponding pattern. For example, the expression $(\{\!|d_1|\!\}_{k_1}, \{\!|d_1|\!\}_{k_2}, k_2)$ may be mapped to the pattern $(\{\!|\Box|\!\}_{k_1}, \{\!|d_1|\!\}_{k_2}, k_2)$ to model the fact that an adversary observing the messages $\{\!|d_1|\!\}_{k_1}$,$\{\!|d_1|\!\}_{k_2}$ and $k_2$, can recover the key $k_2$, decrypt the second ciphertext to $d_1$, and even detect that the first ciphertext uses a key different from $k_2$ (e.g., because decryption under $k_2$ fails), but cannot tell that the first ciphertext encrypts the same message $d_1$ as the second.

In the seminal paper [3], Abadi and Rogaway showed that the meaning associated to cryptographic expressions by standard symbolic methods is computationally sound, in the sense that (under appropriate restrictions) if two expressions are symbolically equivalent (i.e., they have the same pattern), then the associated probability distributions are computationally indistinguishable.

**Induction versus co-induction.** As most work in the area of formal analysis of security protocols, Abadi and Rogaway adopt an inductive approach to the symbolic modeling of adversarial knowledge: initially the attacker does not know any key and tries to learn as many keys as possible from a given cryptographic expression through the application of Dolev-Yao rules.[1] Technically, the knowledge of the adversary can be defined by associating to each cryptographic expression $e$ a corresponding key recovery operator $\mathcal{F}_e$ (mapping sets of keys to sets of keys) which roughly corresponds to a single application of the Dolev-Yao decryption rules. The adversarial knowledge (obtained from observing the expression $e$) can be characterized as the *least fix-point* of the key recovery operator $\mathcal{F}_e$, i.e., the smallest set of keys $S$ such that $\mathcal{F}_e(S) = S$. Operationally, this least fix-point can be obtained by starting from the empty set of keys $\emptyset$ (modeling the adversary's initial knowledge),[2] and applying the key recovery operator $\mathcal{F}_e$ to obtain more and more keys

$$\emptyset \subset \mathcal{F}_e(\emptyset) \subset \mathcal{F}_e^2(\emptyset) \subset \ldots \subset \mathcal{F}_e^m(\emptyset) = \mathcal{F}_e^{m+1}(\emptyset)$$

until the least fix-point $\mathcal{F}_e^m(\emptyset)$ is reached, and no additional keys can be recovered by further applications of $\mathcal{F}_e$.

In this paper we propose a dual, co-inductive approach. More specifically, we propose to define the set of recoverable keys as the *greatest fix-point* of $\mathcal{F}_e$, i.e., the *largest* set of keys $S$ such that $S = \mathcal{F}_e(S)$. As before, the greatest fix-point can be obtained by repeatedly applying the key recovery operator, but this time starting from the set of all keys **Keys**, and resulting in a sequence of smaller and smaller sets

$$\mathbf{Keys} \supset \mathcal{F}_e(\mathbf{Keys}) \supset \ldots \supset \mathcal{F}_e^m(\mathbf{Keys}) = \mathcal{F}_e^{m+1}(\mathbf{Keys})$$

until the greatest fix-point $\mathcal{F}_e^m(\mathbf{Keys})$ is reached. Intuitively, this corresponds to starting from the assumption that no key is guaranteed to be secure, and proving that more and more keys (namely, those in the complement of the sets $\mathcal{F}_e^i(\mathbf{Keys})$) are hidden to the adversary. As we are going to explain, this technical change in the definition of symbolic security has far reaching consequences when it comes to computational soundness.

**Encryption cycles.** In order to prove their soundness theorem, Abadi and Rogaway [3] need to impose a simple, but fundamental, technical restriction: the cryptographic expressions should not contain *encryption cycles*, e.g., sequences of messages of the form

$$\{\!|k_1|\!\}_{k_2}, \{\!|k_2|\!\}_{k_3}, \ldots, \{\!|k_{n-1}|\!\}_{k_n}, \{\!|k_n|\!\}_{k_1},$$

where each key $k_i$ is encrypted under the next key $k_{(i+1) \bmod n}$ in the sequence, circularly. While encrypting a key with itself is typically considered a dangerous cryptographic practice, encryption cycles do occur in a small number of applications (e.g., credential systems [11], encrypted data backups, etc.), and the

[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 plain-text $m$. Such rules are intended to capture the security features of the cryptographic operations used in the construction of messages, and the whole framework relies on the postulate that the adversary cannot perform any other operation. So, for example, given the cipher-text $\{\!|m|\!\}_k$, one cannot recover the message $m$, unless the encryption key $k$ is already known.

[2] The knowledge of the keys of corrupted parties can be modeled by including those keys as part of the expression $e$.

problem of designing encryption schemes supporting such a use has been the subject of many recent papers [18, 17, 10, 6, 16]. In the symbolic security setting it is customary to assume that encryption cycles are *secure*, in the sense that an adversary observing a sequence of circularly encrypted keys, cannot recover any of them. The only notable exception to this intuitively appealing (but computationally unjustified) assumption is perhaps a work of Laud [19], which proposes to resolve the tension between symbolic and computational security with respect to encryption cycles in an ingenious, but ad-hoc, method: strengthening the adversarial model by allowing the Dolev-Yao adversary to break the encryption function whenever an encryption cycle occurs.

**Our contribution.** The main contribution of this paper is to highlight the relation of encryption cycles to inductive and co-inductive definitions of security. Specifically, we prove that

- (Theorem 1) if the set of recoverable keys is defined by co-induction (i.e., as the greatest fix-point of the key recovery operator), then the computational soundness result of Abadi and Rogaway holds unconditionally: if two expressions (with or without encryption cycles) are symbolically equivalent, then their computational counterparts are indistinguishable.

- (Theorem 2) if an expression has no encryption cycles, then the associated key recovery function has a unique fix-point. In particular, the least and greatest fix-point coincide, and the conditional result of Abadi and Rogaway for acyclic expressions follows from the unconditional result in the co-inductive setting.

Our results show that what sets the symbolic and computational frameworks apart (e.g., with respect to their ability to deal with encryption cycles,) is not the inherent difference between the computational and symbolic protocol execution models. Rather, it is the modeling of adversarial knowledge, which is typically *inductive* in the case of symbolic analysis, while intrinsically *co-inductive* in the computational setting. (See Section 5 for a discussion on how recent results on circularly secure encryption schemes [10, 7] are not in contradiction with the overall co-inductive properties of the computational security framework.)

At the technical level, our main computational soundness result (Theorem 1) is fairly general, and applicable to classes of cryptographic expressions that occur in many application domains, like secure multicast key distribution [24, 23, 25], and cryptographically controlled access to XML documents [4]. A follow-up paper [22] demonstrates the generality of our techniques using Theorem 1 to establish a computational soundness theorem for expressions with pseudo-random keys, as those used in [24, 23, 25]. As in this work, the results of [22] hold unconditionally, i.e., without any syntactic restriction on the expressions.

We remark that the uniqueness of the fix-point for acyclic expressions is a purely symbolic result: neither the statement nor proof of Theorem 2 requires the use of the computational execution model. In fact, the proof is simple enough that one could model and verify it using an automated theorem prover. This fact, together with the simplicity of our computational soundness theorem (compared to analogous results from [3] and related papers), suggest that our greatest fix-point framework may be a useful tool even when one is interested in computational soundness with respect to the traditional inductive security definition. Specifically, in order to prove such computational soundness results one can

- first prove computational soundness for the corresponding co-inductive definition of security (possibly using Theorem 1), and

- then find and check (possibly with the help of automated symbolic reasoning tools) syntactic restrictions under which the inductive and co-inductive symbolic security definitions coincide.

So, even if induction may be the most intuitive and preferred way to analyze security protocols in practice, we believe that the co-inductive method would still be a valuable tool to establish the computational soundness of the inductive symbolic analysis.

**Related work** Computationally sound symbolic analysis has been the topic of many recent works. This paper is most closely related to the line of work initiated by Abadi and Rogaway in [3], where secrecy properties with respect to passive adversaries are considered. Subsequent developments along the same lines include [2, 23, 24, 4]. We mention that other approaches to symbolic analysis (e.g., [28]) inherit certain co-inductive ideas from the underlying process calculus, e.g., the use of bisimulation to define the equivalence between cryptographic processes. However, those frameworks are substantially more elaborate than the simple computational soundness setting considered in this paper, and their use of co-induction is quite different.

The problem of dealing with encryption cycles is a classic one in cryptography, already mentioned in the seminal paper [15] introducing the modern notion of computational security for encryption. Following [3], the problem has attracted renewed interest, both within the computational and symbolic setting. Two opposite approaches to resolving the discrepancy with respect to encryption cycles were proposed in [19, 5]. In [19] Laud strengthen the symbolic model by postulating that the Dolev-Yao adversary can break encryption cycles, i.e. whenever an encryption cycle occurs, the adversary can recover all involved keys. Under this assumption, [19] proves a soundness theorem for cryptographic expressions similar to the one of [3], but without the acyclicity restriction on the expressions. A different approach is used in [5], where Adao, Bana, Herzog and Scedrov prove a soundness theorem in the presence of key cycles using a strong security notion for encryption recently proposed in [9, 11]. This notion, called security under "Key Dependent Message" or "Key Dependent Input", allows encrypted messages to depend on the secret decryption key. At the time of [11, 9, 5], no scheme achieving this security notion was known in the standard model, and the only solutions (proposed in [11, 9]) relied on the random oracle heuristic. Since then, the problem of building KDM-secure cryptographic primitives has been investigated in various works [17, 18, 10, 16]. Similar results in the presence of active adversaries are given in [8]. In this paper, we do not consider the extended notions of computational security employed in these works, except for a brief discussion in Section 5. Rather, we focus on the question of the relation between symbolic and computational security when the standard computational security notion of indistinguishability under chosen plaintext attacks (still the golden standard in cryptography in the setting of passive attacks) is employed.

At the technical level the work of Laud [19] is closely related to some of the results presented in this paper, though the motivations and general framework are quite different. In [19], Laud addresses the problem of reconciling symbolic and computational analysis in the presence of key cycles by strengthening the symbolic adversary. Specifically, Laud augments the entailment relation used in inductive approaches with a special rule that explicitly allows the symbolic adversary to break encryption cycles. As a result, Laud proves a computational soundness theorem for encrypted expressions (essentially equivalent to our Corollary 1) that does not require syntactic restrictions. Interestingly, greatest fixpoint computations were suggested [21, equation 15] as an *algorithmic tool* to evaluate Laud's entailment relation. The main difference between [19, 21] and our work is that [19, 21] retain the inductive framework (and entailment relation, see Section 2) for modeling the adversarial knowledge, and resolve the encryption cycles issue using ad-hoc methods. Here we establish a close connection between greatest fixpoints and cryptographic expressions at the semantic (computational soundness) level, and present a general approach (based on the use of co-induction) that can be generalized to a larger class of cryptographic expressions, e.g., the expressions with pseudo-random keys [25, 23], secret sharing schemes [4], etc.

**Organization** The rest of the paper is organized as follows. In Section 2 we present some preliminary definitions on symbolic expressions. (For an overview of the computational cryptography notions used in this paper the reader is referred to the appendix.) In Section 3 we present our main technical results. In Section 4 we illustrate our results on a simple example expression. Section 5 concludes with a discussion of future research directions and open problems.

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

Figure 1: The keys and parts of a pattern.

## 2 Preliminaries

In this section we review the results and standard notation used in previous papers, mostly following the seminal work of Abadi and Rogaway [3]. For an overview of standard computational cryptography definitions and how symbolic expressions are evaluated to probability distributions over bitstrings, the reader is referred to Appendix A. Let $\mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$ be the set of cryptographic expressions built from two sets of key and data symbols $\mathbf{Keys}$,$\mathbf{Data}$, using pairing and encryption operations. Formally, $\mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$ is the set of 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 $k$. Define also the set of *patterns*

$$\mathbf{Pat}(\mathbf{Keys}, \mathbf{Data}) \subset \mathbf{Exp}(\mathbf{Keys} \cup \{\circ\}, \mathbf{Data} \cup \{\square\}), \tag{2}$$

where the symbols $\circ$ and $\square$ denote unknown keys or data respectively.[3] Notice that expressions are just a special case of patterns, while patterns can be regarded (at least syntactically) as expressions over an extended set of keys and data that include the special symbols $\circ$ and $\square$. This justifies the use (common throughout this paper) of the same symbols $e, e_1, e_2$ to denote both expressions and patterns. As a notational convention, we do not write the special key symbol $\circ$ when it occurs as an encryption key. We also assume the paring operation $(\cdot, \cdot)$ is right associative, and omit unnecessary parenthesis. So, for example, we write $(e_1, e_2, e_3)$ and $\{\!|e_1, e_2|\!\}$ instead of $(e_1, (e_2, e_3))$ and $\{\!|(e_1, e_2)|\!\}_\circ$.

The *keys* and *parts* of an expression or pattern (denoted $\mathbf{Keys}(e)$ and $\mathbf{Parts}(e)$) are defined in the standard way according to the rules given in Figure 1. Notice that the special symbol $\circ$ is never included among the keys of a pattern. With this notation, the set of keys $k \in \mathbf{Keys}$ that occur only as encryption subscripts in an expression (but never as messages) is precisely $\mathbf{Keys}(e) \setminus \mathbf{Parts}(e)$. Keys are usually viewed as bound names up to renaming. (E.g., as in the spi calculus [1].) Formally, two expressions or patterns $e_1, e_2$ are *equivalent up to renaming* (written $e_1 \cong e_2$), if there exists a bijection $\mu\colon \mathbf{Keys}(e_1) \to \mathbf{Keys}(e_2)$ such that $\mu(e_1) = e_2$, where $\mu$ acts on $e_1$ as a substitution. Notice that, by definition, $\mu$ only acts on $\mathbf{Keys}$, and the special symbol $\circ$ is always mapped to $\circ$.

The symbolic equivalence of cryptographic expressions is defined by means of a pattern function $\mathbf{p}$ (mapping expressions to corresponding patterns) and the auxiliary function $\texttt{struct}$, both defined in Figure 2. Intuitively, $\texttt{struct}(e)$ 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. Informally, $\texttt{struct}(e)$ is obtained by replacing all keys and data symbols in $e$ by $\circ$ and $\square$ respectively, and $\mathbf{p}(e, T)$ is obtained replacing all subexpressions $\{\!|e'|\!\}_k$ in $e$ such that $k \notin T$ by $\{\!|\texttt{struct}(e')|\!\}_k$. For example,

$$\mathbf{p}((\{\!|d_1|\!\}_{k_1}, \{\!|d_2|\!\}_{k_1}, \{\!|d_1, d_2|\!\}_{k_2}), \{k_1\}) = (\{\!|d_1|\!\}_{k_1}, \{\!|d_2|\!\}_{k_1}, \{\!|\square, \square|\!\}_{k_2}).$$

[3] To be precise, 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.

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

Figure 2: Rules defining the pattern function $\mathbf{p}\colon \mathbf{Pat}(K,D) \times \wp(K) \to \mathbf{Pat}(K,D)$ and auxiliary function $\texttt{struct}\colon \mathbf{Pat}(K,D) \to \mathbf{Pat}(\emptyset,\emptyset)$, where $k \in K \cup \{\circ\}$, $d \in D \cup \{\square\}$, and $e, e_1, e_2 \in \mathbf{Pat}(K,D)$.

The pattern in this example models the fact that, using the key $k_1$, an adversary observing the message $(\{\!\!\{d_1\}\!\!\}_{k_1}, \{\!\!\{d_2\}\!\!\}_{k_1}, \{\!\!\{d_1,d_2\}\!\!\}_{k_2})$ can detect that the first two ciphertexts are the encryption of $d_1$ and $d_2$ under $k_1$. The adversary can also determine that the third ciphertext uses a key different from $k_1$ (e.g., because decryption under $k_1$ fails), and encodes a message which is about the same size as the concatenation of $d_1$ and $d_2$ (e.g., by looking at the length of the ciphertext). However, the adversary cannot extract any other information about the third message. In particular, it cannot detect that the third message is indeed the concatenation of the first two.

Going back to the definition of symbolic equivalence, each expression is mapped to a pattern

$$\mathbf{pattern}(e) = \mathbf{p}(e, \mathbf{recoverable}(e)) \tag{3}$$

where $\mathbf{recoverable}(e) \subseteq \mathbf{Keys}$ is a set (to be defined) which informally consists of all keys that can be "recovered" by an adversary observing $e$. Two expressions $e_1, e_2$ are considered symbolically equivalent if $\mathbf{pattern}(e_1) \cong \mathbf{pattern}(e_2)$, i.e., if they have the same pattern up to key renaming.

In most previous work (starting from the original Dolev-Yao paper [12], and including the seminal contribution of Abadi and Rogaway [3]) the set of recoverable keys is defined as

$$\mathbf{recoverable}(e) = \{k \colon e \vdash k\}$$

where the entailment relation $\vdash$ is the *smallest* binary relation over $\mathbf{Exp}(\mathbf{Keys},\mathbf{Data})$ such that

1. $e \vdash e$ for all $e \in \mathbf{Exp}(\mathbf{Keys},\mathbf{Data})$,
2. if $e \vdash (e_1, e_2)$ then $e \vdash e_1$ and $e \vdash e_2$, and
3. if $e \vdash \{\!\!\{e_1\}\!\!\}_k$ and $e \vdash k$, then $e \vdash e_1$.

Informally, the entailment relation $\vdash$ represents the capabilities of a Dolev-Yao adversary, that given $e$, tries to learn as much as possible from $e$. For example the last rule stipulates that if the adversary can recover both the ciphertext $\{\!\!\{e_1\}\!\!\}_k$ and the key $k$, then she decrypt and recover the plaintext $e_1$ too.

We remark that other definitions of recoverable keys have been considered in the literature. Most notably, in an effort to remove the syntactic restriction to acyclic expressions, Laud [20] has proposed an alternative definition of the entailment relation that strengthens the Dolev-Yao adversary by explicitly allowing him to break the encryption cycles. Formally, Laud defines $\mathbf{recoverable}(e) = \{k \colon e \vdash_\emptyset k\}$ where the entailment relation $\vdash_S$ is defined as the *smallest* relation satisfying the following conditions

1. $e \vdash_S e$,
2. if $e \vdash_S (e_1, e_2)$ then $e \vdash_S e_1$ and $e \vdash_S e_2$,
3. if $e \vdash_S \{\!\!\{e'\}\!\!\}_k$ then $e \vdash_{S \cup \{k\}} e'$,
4. if $e \vdash_{S \cup \{k\}} e'$ and $e \vdash_S k$ then $e \vdash_S e'$,

5. if $e \vdash_S e'$ and $S \subseteq S'$ then $e \vdash_{S'} e'$,

6. if $e \vdash_{S \cup \{k\}} k$ then $e \vdash_S k$.

Intuitively, the relation $e \vdash_S e'$ models the fact that expression $e'$ can be recovered from expression $e$ using the keys in $S$ for decryption. So, for example, rule 5 simply states that increasing the number of available decryption keys does not decrease our ability to recover information from $e$. Rules 1 and 2 are the same as for the entailment relation $\vdash$ used by Abadi and Rogaway. Rules 3 and 4 together imply the standard decryption rule: if $e \vdash_S \{e'\}_k$ and $e \vdash_S k$, then $e \vdash_S e'$. The main novelty in Laud's definition is rule 6, which captures the idea that the adversary can break encryption cycles: if decrypting under $k$ allows to recover $k$, then $k$ is part of an encryption cycle and it can be recovered by the adversary.

## 3 Computationally sound greatest fix-point semantics

In order to compare our results to prior work, it is convenient to give a different, but equivalent definition of the set of recoverable keys. First of all, we extend the pattern computation function $\mathbf{p}$ of Abadi and Rogaway [3] to include patterns in its domain. This is done in the obvious way, namely, we let

$$\mathbf{p}\colon \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data}) \times \wp(\mathbf{Keys}) \to \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$$

be the function defined precisely by the same rules already given in Figure 2. Next, we introduce a key recovery function

$$\mathbf{r}\colon \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data}) \to \wp(\mathbf{Keys})$$

which is, in a sense, a counterpart to the pattern computation function $\mathbf{p}$ of [3]. Intuitively, the function $\mathbf{r}$ maps the expression or pattern $e$ to the set of keys recoverable from *all parts* of $e$. For the class of patterns used in this paper, the function $\mathbf{r}$ can be simply defined as

$$\mathbf{r}(e) = \{k \in \mathbf{Keys}(r)\colon k \in \mathbf{Parts}(e)\} = \mathbf{Keys}(e) \cap \mathbf{Parts}(e), \tag{4}$$

i.e., $\mathbf{r}(e)$ is the set of all keys that appear in $e$ as a message. In other words, $\mathbf{r}(e)$ includes all keys of $e$, except those that occur exclusively as encryption subscripts.

We observe that the functions $\mathbf{p}$ and $\mathbf{r}$ satisfy the following fundamental properties:

$$\begin{align} \mathbf{p}(e, \mathbf{Keys}) &= e \tag{5} \\ \mathbf{p}(\mathbf{p}(e,S),T) &= \mathbf{p}(e, S \cap T) \tag{6} \\ \mathbf{r}(\mathbf{p}(e,T)) &\subseteq \mathbf{r}(e) \tag{7} \end{align}$$

These are all very natural requirements. Properties (5) and (6) just say that $\mathbf{p}$ makes keys *act* on the patterns, or, more precisely, $(\wp(\mathbf{Keys}), \cap)$ acts[4] as a monoid on the set $\mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$. The third property (7) states that the action $\mathbf{p}(\cdot, T)$ does not increase the amount of information recoverable from (the parts of) a pattern. We will see later that these are the only properties needed to instantiate our general framework, but for now the reader may want to focus on the specific functions $\mathbf{p}$ and $\mathbf{r}$ defined in Figure 2 and (4).

The functions $\mathbf{p}, \mathbf{r}$ are used to associate to each expression $e \in \mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$ a corresponding key recovery operator

$$\mathcal{F}_e\colon T \mapsto \mathbf{r}(\mathbf{p}(e,T)) \tag{8}$$

that maps any $T \subseteq \mathbf{Keys}$ to the set of keys recoverable from all the parts of the observable pattern $\mathbf{p}(e,T)$. The function $\mathcal{F}_e$ models the process of using a set of keys $T$ to break an expression $e$ into parts, and then using all such parts to recover as many keys as possible. In Theorem 1 we will show that for any expression

[4] Recall that an action of a monoid $(G, \cdot)$ on a set $A$ is a binary operation $\times$ mapping $A \times G$ to $A$ such that $(a \times g_1) \times g_2 = a \times (g_1 \cdot g_2)$ and $a \times 1_G = a$.

$e$, the key recovery operator (8) is a monotone function. In particular, $\mathcal{F}_e$ admits both a least and a greatest fix-point

$$\mathrm{fix}(\mathcal{F}_e) = \bigcup_n \mathcal{F}^n(\emptyset) \qquad \mathrm{FIX}(\mathcal{F}_e) = \bigcap_n \mathcal{F}^n(\mathbf{Keys}).$$

It is a well known fact that the set of keys $\{k : e \vdash k\}$ recoverable by a Dolev-Yao adversary is precisely the least fix point of $\mathcal{F}_e$. So, the Abadi-Rogaway definition of the pattern of an expression can be reformulated as $\mathbf{pattern}(e) = \mathbf{p}(e, \mathrm{fix}(\mathcal{F}_e))$.

Our general framework is very similar to the one of Abadi and Rogaway, and we adopt most definitions given so far. The only difference is that, instead of defining recoverable keys as the least fix-point of $\mathcal{F}_e$, we take the greatest fix-point and let

$$\mathbf{Pattern}(e) = \mathbf{p}(e, \mathrm{FIX}(\mathcal{F}_e)). \tag{9}$$

As usual, two expressions are symbolically equivalent if they have the same pattern (9) up to key renaming. We refer the reader to Section 4 for an example of use of the greatest fix-point patterns.

In this section we prove that our new greatest fix-point symbolic semantic is computationally sound, i.e., for any two expressions $e_1, e_2$, if $\mathbf{Pattern}(e_1) \cong \mathbf{Pattern}(e_2)$, then the probability distributions $[\![e_1]\!]$ and $[\![e_2]\!]$ are computationally indistinguishable. We do so in a very general way, applicable to a wider class of cryptographic expressions than considered in this paper and in [3], as demonstrated in follow-up work [22]. Theorem 1 below states that, as long as properties (5-7) are satisfied, in order to establish the computational soundness of the greatest fix-point symbolic semantics (9) it is enough to test the following simpler condition: for any pattern $e$, the probability distributions $[\![e]\!]$, and $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ are computationally indistinguishable. Informally, this condition states that the keys $\mathbf{r}(e)$ recoverable from all parts of a pattern do not increase our knowledge about the pattern. This is a non-trivial assumption, as it depends on the security of the encryption scheme, but still it is a much easier-to-check condition than the conclusion of the soundness theorem. In particular, the indistinguishability of $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ and $[\![e]\!]$ can be usually proved in a fairly direct way, starting from the definition of secure encryption scheme, without the need to go through a complex hybrid argument.

**Theorem 1** *Let* $\mathbf{Keys}$ *and* $\mathbf{Data}$ *be two (disjoint) sets of key and constant symbols, and let* $\mathbf{p}$ *and* $\mathbf{r}$ *be functions satisfying the action properties (5–7). Then, for any expression* $e \in \mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$*, the key recovery operator (8) is a monotone function, and the greatest fix-point semantics (9) is well defined. 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, the distribution* $[\![e]\!]$ *is computationally indistinguishable from (9). In particular, for any two expressions* $e_1, e_2 \in \mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$*, if* $\mathbf{Pattern}(e_1) \cong \mathbf{Pattern}(e_2)$*, then the distributions* $[\![e_1]\!]$ *and* $[\![e_2]\!]$ *are computationally indistinguishable.*

*Proof.* First of all, we show that the key recovery operator is monotone. Let $S \subseteq T \subseteq \mathbf{Keys}$ be two sets of keys. From the definition of $\mathcal{F}_e$ and properties (6–7), we obtain

$$\begin{aligned} \mathcal{F}_e(S) &= \mathbf{r}(\mathbf{p}(e, S)) \\ &= \mathbf{r}(\mathbf{p}(e, T \cap S)) \\ &= \mathbf{r}(\mathbf{p}(\mathbf{p}(e, T), S)) \\ &\subseteq \mathbf{r}(\mathbf{p}(e, T)). \end{aligned}$$

So, $\mathcal{F}_e$ is a monotone operator and it admits a greatest fixpoint $\mathrm{FIX}(\mathcal{F}_e) = \bigcap_i \mathcal{F}^i(\mathbf{Keys})$.

Now consider an expression $e$ and the corresponding pattern $\mathbf{pattern}(e) = \mathbf{p}(e, \mathrm{FIX}(\mathcal{F}_e))$, and assume without loss of generality that $\mathbf{Keys} = \mathbf{Keys}(e)$, so that $n = |\mathbf{Keys}|$ is polynomially bounded in the size of $e$. Since $\mathcal{F}_e$ is a monotone function, we have $\mathrm{FIX}(\mathcal{F}_e) = \mathcal{F}_e^n(\mathbf{Keys})$, where $n = |\mathbf{Keys}|$ is the length of the longest chain in $\wp(\mathbf{Keys})$. We will show that for every $i$, $[\![\mathbf{p}(e, \mathcal{F}_e^{i+1}(\mathbf{Keys}))]\!]$ is computationally indistinguishable from $[\![\mathbf{p}(e, \mathcal{F}_e^{i}(\mathbf{Keys}))]\!]$. It follows, by transitivity, that $[\![\mathbf{p}(e, \mathrm{FIX}(\mathcal{F}_e))]\!] = [\![\mathbf{p}(e, \mathcal{F}_e^{n}(\mathbf{Keys}))]\!]$ is computationally indistinguishable from $[\![e]\!] = [\![\mathbf{p}(e, \mathbf{Keys})]\!] = [\![\mathbf{p}(e, \mathcal{F}_e^{0}(\mathbf{Keys}))]\!]$. More specifically, any probabilistic polynomial time algorithm distinguishing $[\![e]\!]$ from $[\![\mathbf{pattern}(e)]\!]$ with advantage $\delta$ can be turned into a probabilistic polynomial time algorithm that distinguishes $[\![\mathbf{p}(e, \mathcal{F}_e^{i+1}(\mathbf{Keys}))]\!]$ from $[\![\mathbf{p}(e, \mathcal{F}_e^{i}(\mathbf{Keys}))]\!]$ for some $i$ with advantage $\delta/n$.

Fix the value of the index $i$, and let $T = \mathcal{F}_e^i(\mathbf{Keys})$ and $e' = \mathbf{p}(e, T)$. Clearly, $\mathcal{F}_e(\mathbf{Keys}) \subseteq \mathbf{Keys}$ because $\mathbf{Keys}$ is the set of all keys in $e$, and from the monotonicity of $\mathcal{F}_e$ we get that $\mathcal{F}_e^{i+1}(\mathbf{Keys}) \subseteq \mathcal{F}_e^{i}(\mathbf{Keys})$ for all $i \geq 0$. In particular, $\mathcal{F}_e(T) \subseteq T$. We want to prove that $[\![\mathbf{p}(e, \mathcal{F}_e(T))]\!]$ is indistinguishable from $[\![\mathbf{p}(e, T)]\!]$. Notice that, using the definition of $\mathcal{F}_e(T) = \mathbf{r}(\mathbf{p}(e, T))$, we get

$$\begin{aligned}\mathbf{p}(e', \mathbf{r}(e')) &= \mathbf{p}(\mathbf{p}(e, T), \mathcal{F}_e(T)) \\ &= \mathbf{p}(e, T \cap \mathcal{F}_e(T)) \\ &= \mathbf{p}(e, \mathcal{F}_e(T)).\end{aligned}$$

Remember that by hypothesis, $[\![\mathbf{p}(e', \mathbf{r}(e'))]\!]$ is computationally indistinguishable from $[\![e']\!]$. Therefore, $[\![\mathbf{p}(e, \mathcal{F}_e(T))]\!] = [\![\mathbf{p}(e', \mathbf{r}(e'))]\!]$ is indistinguishable from $[\![e']\!] = [\![\mathbf{p}(e, T)]\!]$ as claimed. □

We remark that in Theorem 1 we have assumed that $e$ is an expression for simplicity only. The same result (and proof) holds true also when $e \in \mathbf{Pat}(\mathbf{Keys}, \mathbf{Data})$ is an arbitrary pattern. In the following corollary we apply Theorem 1 to the functions $\mathbf{p}$ and $\mathbf{r}$ defined in Figure 2 and (4).

**Corollary 1** *Let $\mathbf{p}$ and $\mathbf{r}$ as defined in Figure 2 and (4). If $\mathcal{E}$ is a (length regular) semantically secure encryption scheme, then 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.* We already observed that $\mathbf{p}$ and $\mathbf{r}$ satisfy properties (5–7). In order to apply Theorem 1 and conclude that $[\![e_1]\!]$ is indistinguishable from $[\![e_2]\!]$, we only need to prove that for any pattern $e$, the distributions $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ and $[\![e]\!]$ and computationally indistinguishable. To this end, assume for contradiction that there exists an efficient algorithm $\mathcal{D}$ that distinguishes distribution $[\![e]\!]$ from $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ with non-negligible probability. (See Definition 2 in Appendix A.) We use $\mathcal{D}$ to construct an efficient adversary that breaks the indistinguishability of the encryption scheme $\mathcal{E}$ used to evaluate the patterns. Let $T = \mathbf{Keys}(e) \setminus \mathbf{Parts}(e)$ be the set of all encryption keys in $e$ that do not also appear in $e$ as a message. We define an adversary $\mathcal{A}$ that is given access to $|T|$ encryption oracles $\mathcal{E}_b^t(\cdot, \cdot)$ (indexed by $t \in T$). The adversary $\mathcal{A}$ chooses keys $\sigma(k)$ independently at random for all $k \in \mathbf{Keys}(e) \setminus T = \mathbf{r}(e)$. It then evaluates the expression $e$ according to the usual evaluation rules, except for subexpressions of the form $\{\!|e'|\!\}_k$ where $k \in T$. These are evaluated using oracle $\mathcal{E}_b^k$. When $\mathcal{A}$ is done evaluating $e$, it submits the resulting string to the distinguisher $\mathcal{D}$. Notice that when $b = 1$, the adversary $\mathcal{A}$ produces a query which is distributed identically to $[\![e]\!]$, while when $b = 0$ the distribution is $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$. So, $\mathcal{A}$ will have the same advantage in breaking the encryption scheme as $\mathcal{D}$ has in distinguishing $[\![e]\!]$ from $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$. □

Corollary 1 is very similar in spirit to the soundness result proved by Abadi and Rogaway in [3]. However, our proof of Corollary 1 is much simpler than the original argument given by Abadi and Rogaway, which requires the expressions $e_1, e_2$ to be acyclic. The main difference is our use of greatest fix-points in the definition of adversarial knowledge, while [3] uses the traditional least fix-point definition. At first sight, the two results may seem incomparable, since they use different definitions of patterns. The following theorem bridges the gap between the two (inductive and co-inductive) definitions of pattern, showing that acyclic expressions have a unique fix-point. So, under the acyclicity hypothesis of [3] (common to most other work on computationally sound symbolic cryptography,) the traditional least fix-point semantics and the new greatest fix-point semantics are identical.

**Theorem 2** *If $e \in \mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$ is an acyclic expression, then $\mathit{fix}(\mathcal{F}_e) = \mathit{FIX}(\mathcal{F}_e)$.*

*Proof.* Assume $\mathrm{fix}(\mathcal{F}_e) \neq \mathrm{FIX}(\mathcal{F}_e)$. We prove that $e$ contains an encryption cycle. Since $\mathrm{fix}(\mathcal{F}_e) \subset \mathrm{FIX}(\mathcal{F}_e)$, the set $T = \mathrm{FIX}(\mathcal{F}_e) \setminus \mathrm{fix}(\mathcal{F}_e)$ is not empty. Notice that all $k \in T$ necessarily belong to $\mathbf{r}(e)$ because by monotonicity

$$T \subseteq \mathrm{FIX}(\mathcal{F}_e) = \mathcal{F}_e(\mathrm{FIX}(\mathcal{F}_e)) \subseteq \mathcal{F}_e(\mathbf{Keys}(e)) = \mathbf{r}(\mathbf{p}(e, \mathbf{Keys}(e)) = \mathbf{r}(e). \tag{10}$$

However, all occurrences of $k \in T$ in $e$ must be under the scope of an encryption operator $\{\!|\ldots k \ldots|\!\}_{k'}$ with $k' \notin \mathrm{fix}(\mathcal{F}_e)$, because $k \notin \mathrm{fix}(\mathcal{F}_e)$. Again, from (10), we get that at least some occurrence of $k$ in $e$ must not be encrypted under keys outside of $\mathrm{FIX}(\mathcal{F}_e)$. It follows, that $k$ must be encrypted under some key $k \in \mathrm{FIX}(\mathcal{F}_e) \setminus \mathrm{fix}(\mathcal{F}_e) = T$. Consider now the "encrypt" relation, restricted to the keys in $T$: for any $k_1, k_2 \in T$, $k_1$ encrypts $k_2$ (in $e$) if $e$ contains a subexpression $\{\!|e'|\!\}_{k_1}$ such that $k_2 \in \mathbf{Parts}(e')$. We just proved that all keys in $T$ are encrypted in $e$ under some key in $T$, i.e., all nodes $T$ in the graph of the "encrypt" relation, have in-degree at least one. Since $T$ is a non-empty finite set, it must necessarily contain a cycle. □

## 4 Example

In this section we illustrate our greatest fix-point symbolic framework on a simple example expression. Let

$$e = (\{\!|k_1, \{\!|\{\!|k_4|\!\}_{k_3}|\!\}_{k_4}|\!\}_{k_2}, \{\!|k_2|\!\}_{k_1}).$$

The set of recoverable keys associated to this expression is defined as the greatest fix-point of the key recovery operator $\mathcal{F}_e$. This fix-point is computed as follows. Start from the set $K_0 = \{k_1, k_2, k_3, k_4\}$ of all keys in the expression, and apply $\mathcal{F}_e$ to it to obtain the set

$$\begin{aligned} K_1 &= \mathcal{F}_e(K_0) \\ &= \mathbf{r}(\mathbf{p}(e, K_0)) \\ &= \mathbf{r}(\{\!|(k_1, \{\!|\{\!|k_4|\!\}_{k_3}|\!\}_{k_4})|\!\}_{k_2}, \{\!|k_2|\!\}_{k_1}) \\ &= \{k_1, k_2, k_4\}. \end{aligned}$$

As we apply $\mathcal{F}_e$ to $K_1$ we obtain

$$\begin{aligned} K_2 &= \mathcal{F}_e(K_1) \\ &= \mathbf{r}(\mathbf{p}(e, K_1)) \\ &= \mathbf{r}(\{\!|(k_1, \{\!|\{\!|\circ|\!\}_{k_3}|\!\}_{k_4})|\!\}_{k_2}, \{\!|k_2|\!\}_{k_1}) \\ &= \{k_1, k_2\}. \end{aligned}$$

If we apply $\mathcal{F}_e$ once more we obtain

$$\begin{aligned} K_3 &= \mathcal{F}_e(K_2) \\ &= \mathbf{r}(\mathbf{p}(e, K_2)) \\ &= \mathbf{r}(\{\!|(k_1, \{\!|\{\!|\circ|\!\}|\!\}_{k_4})|\!\}_{k_2}, \{\!|k_2|\!\}_{k_1}) \\ &= \{k_1, k_2\}. \end{aligned}$$

Notice that we obtained a decreasing sequence of sets

$$\mathcal{F}_e^0(\mathbf{Keys}) = \{k_1, k_2, k_3, k_4\} \supset \mathcal{F}_e^1(\mathbf{Keys}) = \{k_1, k_2, k_4\} \supset \mathcal{F}_e^2(\mathbf{Keys}) = \{k_1, k_2\} = \mathcal{F}_e^3(\mathbf{Keys})$$

and $\mathcal{F}_e^i(\mathbf{Keys}) = \{k_1, k_2\}$ for all $i \geq 2$. This is the greatest fixpoint of the operator $\mathcal{F}_e$, so the symbolic semantics of expression $e$ is

$$\mathbf{Pattern}(e) = \mathbf{p}(e, \{k_1, k_2\}) = \{\!|(k_1, \{\!|\{\!|\circ|\!\}|\!\}_{k_4})|\!\}_{k_2}, \{\!|k_2|\!\}_{k_1}).$$

This pattern tells us that the keys $k_1$ and $k_2$ are not guaranteed to be hidden from an adversary when a computational encryption scheme (satisfying the standard notion of indistinguishability against chosen plaintext attack) is used. On the other hand, the adversary cannot recover they keys $k_3$ and $k_4$, even if $k_4$ is part of an encryption cycle.

# 5 Discussion and open problems

We presented a general framework for the computationally sound symbolic analysis of cryptographic expressions, as those used to model messages in security protocols. The framework is essentially the same as the standard one proposed by Abadi and Rogaway [3], with the only difference that the adversarial knowledge is defined by co-induction (using greatest fixpoints), rather than induction (using least fixpoints). This simple change brings the computational and symbolic definitions much closer to each other.

We believe that our observations improve our understanding of the relation between symbolic and computational cryptography, and open up several new interesting research directions. In retrospect, the fact that co-inductive methods (in the symbolic setting) result in a closer connection to computational security should not come too much as a surprise, since the methods of computational cryptography (e.g., the notion of computational indistinguishability, a form of observational equivalence) have a very strong co-inductive flavor. Acyclicity and similar syntactic restrictions are not a peculiarity of [3]: most work on computationally sound symbolic security analysis (with just a few rare exceptions like [19]) seem to require restrictions of this sort. Our results suggest the use of co-induction in the symbolic modeling of adversarial knowledge as a general method to prove closer connections between symbolic and computational security in other settings. There is a need for more work in the area of co-inductive symbolic security analysis, and such work is likely to provide a better bridge between symbolic and computational cryptography than traditional methods based on induction.

Very recently it has been shown [10, 7] that it is possible to build computationally secure encryption schemes (under standard complexity assumptions) that remain secure in the presence of encryption cycles. It is natural to ask how these schemes fit into the overall co-inductive framework of computational security. We see no contradiction between the (perhaps unexpected) results of [10, 7], and the generic assertion that computational security is founded on co-inductive reasoning principles. We remark that [10, 7] achieve circular security by building encryption schemes satisfying very strong homomorphic properties that allow, for example, to build the encryption of $k$ under $k$ (i.e., an encryption cycle of length 1) given the encryption of 0 under $k$, and similarly for longer cycles. In order to properly model (symbolically) such encryption functions, one has to add the homomorphic properties to the Dolev-Yao rules, and modify the key recovery operator $\mathcal{F}_e$ accordingly. While a computationally sound symbolic treatment of homomorphic encryption is outside of the scope of this paper, we believe that for the encryption schemes of [10, 7] it would result in key recovery operators with unique fixpoint, so that circular security could still be explained in co-inductive terms. We leave a full investigation of computational soundness of encryption schemes with special properties to future work.

The generality of our approach (at least in the setting of secrecy properties in the presence of passive adversaries) has recently been demonstrated in [22], where Theorem 1 is used to establish the computational soundness of symbolic expressions with pseudorandom keys, as those employed in multicast key distribution protocols [24, 23, 25]. As in this paper, the result of [22] does not require the expressions to be acyclic or satisfy any syntactic restriction. We expect similar results can also be obtained for cryptographic expressions that make use of secret sharing schemes (as those employed in [4] in the analysis of cryptographically controlled access to xml documents), and most other cryptographic primitives achieving secrecy goals.

The main open problem at this point is to extend our co-inductive framework to prove computational soundness results in the presence of active adversaries, as those considered in [26]. We remark that moving from passive adversaries to active attacks requires substantial changes in the execution model. In a passive attack, an adversary only gets to see the sequence of messages transmitted during the execution of the protocol. So the entire adversary's view of the system can be modeled by a sequence of expressions (or even a single expression containing their concatenation.) In an active attack scenario, the adversary interacts with the honest parties, intercepting and injecting messages in the communication network. Security properties no longer pertain exclusively what information can be learned by the adversary, but also how the adversary can influence the messages. A general approach to computational soundness in the presence of active adversaries has been proposed in [26], where security properties are modeled as sets of traces, e.g., sequences of events that can occur during a run of the protocol. We leave the development of a co-inductive framework for the study of cryptographic trace properties in the presence of active attacks as an open problem.

# References

[1] M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. In *Proc. of CCS*, pages 36–47, 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 *LNCS*, pages 82–94. Springer, Oct. 2001.

[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] P. Adão, G. Bana, J. Herzog, and A. Scedrov. Soundness of formal encryption in the presence of key-cycles. In *Computer Security – Proceedings of ESORICS 2005*, volume 3679 of *Lecture Notes in Computer Science*, pages 374–396. Springer, 2005.

[6] P. Adão, G. Bana, and A. Scedrov. Computational and information theoretic soundness and completeness of formal encryption. In *CSFW'05*, pages 170–184. IEEE Computer Society, June 2005.

[7] B. Applebaum, D. Cash, C. Peikert, and A. Sahai. Fast circular-secure encryption based on hard learning problems. In *Proceedings of CRYPTO 2009*, LNCS. Springer, 2009. To appear.

[8] M. Backes, B. Pfitzmann, and A. Scedrov. Key-dependent message security under active attacks - brsim/uc-soundness of dolev-yao-style encryption with key cycles. *Journal of Computer Security*, 16(5):497–530, 2008. Preliminary version in CSF'07.

[9] J. Black, P. Rogaway, and T. Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In K. Nyberg and H. M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75, St. John's, Newfoundland, Canada, Aug. 2002. Springer.

[10] D. Boneh, S. Halevi, M. Hamburg, and R. Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *Advances in Cryptology – Proceedings of CRYPTO'08*, LNCS. IACR, Springer, 2008.

[11] J. Camenish and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity. In B. Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, volume 2045 of *LNCS*, pages 93–118. Springer, May 2001.

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

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

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

[15] 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.

[16] I. Haitner and T. Holenstein. On the (im)possibility of key dependent encryption. In *Theory of Cryptography Conference*, 2009. To appear. Preliminary version in IACR ePrint TR 2008-164.

[17] S. Halevi and H. Krawczyk. Security under key-dependent inputs. In *Computer and communications security – Proceedings of CCS'07*, pages 466–475, Alexandria, VA, USA, 2007. ACM.

[18] D. Hofheinz and D. Unruh. Towards key-dependent message security in the standard model. In *Advances in Cryptology – Proceedings of Eurocrypt'08*, volume 4965 of *LNCS*, pages 108–126. IACR, Springer, 2008.

[19] 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.

[20] 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.

[21] P. Laud and V. Vene. A type system for computationally secure information flow. In M. Liśkiewicz and R. Reischuk, editors, *Foundamentals of Computational Theory – Proc. of FCT'05*, volume 3623 of *Lecture Notes in Computer Science*, pages 365–377. Springer, 2005.

[22] D. Micciancio. Pseudo-randomness and partial information in symbolic security analysis. Report 2009/249, IACR ePrint archive, 2009. URL http://eprint.iacr.org/2009/249.

[23] 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.

[24] 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.

[25] 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.

[26] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Theory of Cryptography Conference – Proceedings of TCC'04*, volume 2951 of *LNCS*, pages 133–151. Springer, Feb. 2004.

[27] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murphi. In *IEEE Symposium on Security and Privacy*, pages 141–151. IEEE Computer Society, 1997.

[28] 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.

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

# A Cryptography

In the computational setting, given an encryption scheme $\mathcal{E}$, each expression $e \in \mathbf{Exp}(\mathbf{Keys}, \mathbf{Data})$ naturally maps to a probability distribution $[\![e]\!]$ over bitstrings. Two expressions $e_1, e_2$ are equivalent in the computational setting if the corresponding probability distributions $[\![e_1]\!] \equiv [\![e_2]\!]$ are computationally indistinguishable. In this appendix we briefly recall all the basic computational security definitions used in this paper. The reader is referred to any standard textbook (e.g., [13, 14]) for details.

**Encryption.** 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 $\ell$ that depends on the desired security level. The encryption scheme is considered secure if it satisfies the following property, called semantic security or indistinguishability under chosen plaintext attack.

**Definition 1** *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 $\ell$ uniformly at random and run $\mathcal{A}$ on input $\ell$ 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 $\ell$, 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 $\ell$, i.e., it is smaller than $1/\ell^c$ for any constant $c$ and sufficiently large $\ell$.*

The above definition 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 2** *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 $\ell$ each, uniformly and independently at random and run $\mathcal{A}$ on input $\ell$ 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 $\ell$, 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 $\ell$, i.e., it is smaller than $1/\ell^c$ for any constant $c$ and sufficiently large $\ell$.*

Computational equivalence between probability distributions over bitstrings is defined below.

**Definition 3** *Let $\{A_i^0\}$ and $\{A_i^1\}$ be two probability ensembles, i.e., two sequences of probability distributions over bitstrings. $\{A_i^0\}$ and $\{A_i^1\}$ are computationally indistinguishable if for any probabilistic polynomial time adversary $D$, the quantity $|\Pr\{D(A_i^0) = 1\} - \Pr\{D(A_i^1) = 1\}|$ is negligible in $i$.*

**Computational evaluation.** Cryptographic expressions can be evaluated using a computational encryption scheme $\mathcal{E}$. In order to map the expressions to strings we need also to fix a string value $\gamma_d$ for every piece of data $d \in \mathbf{Data}$ appearing in the expression, and a pairing function $\gamma\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$.

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 $\sigma[\![d]\!] = \gamma_d$, $\sigma[\![k]\!] = \sigma(k)$, $\sigma[\![(e_1, e_2)]\!] = \gamma(\sigma[\![e_1]\!], \sigma[\![e_2]\!])$, 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 $\sigma$ (by setting $\sigma(k) \in \{0,1\}^\ell$ to an independently and randomly chosen value for each key symbol $k \in \mathbf{Keys}$) and then computing $\sigma[\![e]\!]$.

**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 $d \mapsto \gamma_d$, $\gamma(\cdot, \cdot)$ 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 $k$, $|\gamma(x_1, x_2)|$ 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 $\texttt{struct}(e) = \texttt{struct}(e')$ 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 by defining $\sigma[\![\texttt{struct}(e)]\!] = 0^{|\sigma[\![e]\!]|}$. Notice that the definition is well given because $|\sigma[\![e]\!]|$ depends only on $\texttt{struct}(e)$, and not on the specific expression $e$.