Co-induction and Computational Semantics for Public-key Encryption with Key Cycles

Mohammad Hajiabadi, Bruce M. Kapron
Department of Computer Science
University of Victoria
Victoria, BC, CANADA
mhaji, bmkapron@csc.uvic.ca

***Abstract*—We consider the computational soundness and completeness of formal indistinguishability in public-key cryptography, in the presence of key-cycles. This problem in the absence of key-cycles is addressed in [1] by Herzog. It is known from [2] that the standard notions of computational security cannot give a computationally-sound interpretation to key-cyclic expressions. However, the work of [2] takes into account formal adversaries whose knowledge set is formulated using an inductive definition. In recent work by Micciancio [3], an alternate way of specifying the knowledge set of formal adversaries is proposed, giving a co-inductive definition of security. Using this new specification, a soundness result in the presence of key-cycles is proved in the private key setting. In this paper, we explore the co-inductive definition of security in public-key cryptography, and extend the soundness result of Herzog by proving it for key-cyclic expressions. We also show that the completeness result is achievable in the absence of key-cycles with respect to any length-revealing computational encryption system. In the presence of key-cycles, we show that the completeness result does not hold even with respect to CCA-2 secrecy.**

***Keywords*-Computational soundness and completeness, greatest fixed-points, least fixed points, weak public-key indistinguishability, strong public-key indistinguishability, CCA-2 security**

I. INTRODUCTION

Verifying the correctness of security protocols is a fundamental and challenging task in cryptography. The rigorous analysis of security protocols typically follows one of two approaches, based either on formal cryptography, or on computational cryptography. In formal cryptography:

- Messages are viewed as formal (symbolic) expressions, and cryptographic operations are considered as purely syntactical operations on formal expressions.
- A formal adversary is an attacker, possibly with an initial knowledge set and with the ability of enhancing its knowledge set in accordance with certain deductive rules which are assumed to be the only symbolic operations a formal adversary is able to perform.
- The notion of indistinguishability is interpreted as observational equivalence, that is symbolic identity with respect to the adversary's knowledge set.
- A strong assumption is placed on encryption by viewing it as a perfectly secure operation; namely, the only way for a formal adversary to obtain any information from a given ciphertext is to have the underlying key.

In computational cryptography, on the other hand, we have:

- Messages are bit-strings, and the cryptographic operations are functions over bit-strings.
- A computational adversary is a probabilistic polynomial time Turing machine.
- The notion of equivalence is that of computational indistinguishability: two computational expressions are computationally indistinguishable if any adversary has only a negligible chance of telling them apart.
- Proofs of security are performed via reduction; an encryption system is secure if its security is implied by some problem known to be computationally hard.

In the last few years, starting with the seminal work of Abadi and Rogaway on the computational soundness of formal encryption, there has been a lot of effort in relating these two views of cryptography. The problem of interest in this area is to examine the computational justification of formal operations. Computational soundness of formal expressions essentially means that equivalent formal expressions produce computationally indistinguishable probability distributions. It turns out that these two views cannot be completely reconciled, mainly because of the way in which they deal with encryption cycles [2], [4]. As an example, in formal cryptography, encrypting a key under itself, the simplest form of an encryption cycle, is considered to be secure in the sense that no formal adversary can distinguish it from any other encryption as long as it does not have the underlying key symbol. On the other hand, there are many computationally secure encryption systems under which a key encrypted under itself is totally visible [5]. Thus, it seems that in order to relate the two worlds of cryptography more closely, we have to take one step back and refine the definition of symbolic security for key-cyclic expressions. This is the subject of the work of [3] in which the adversarial knowledge set is defined with respect to the co-inductive method, as opposed

to the inductive method which was considered in previous work. In particular, the result of [3] indicates that formal equivalence based on the co-inductive definition of security implies computational equivalence even in the presence of key-cycles. The result of [3] is stated for private-key cryptography and a computational semantics is based on CPA-secure encryption systems.

Our results: In this paper, we focus on formal and computational models based on public-key cryptography, and we extend the results of [1] in several directions. In [1], a public-key version of computational indistinguishability – which we call strong public-key indistinguishability – is developed, and based on that, the computational soundness of formal encryption is proved in the absence of key-cycles. In this paper, we consider the co-inductive definition of symbolic security, as in [3], but in the setting of public-key cryptography. We also introduce a weaker notion of indistinguishability than the strong indistinguishability of [1], which we call weak public-key indistinguishability. Specifically, our contributions include:

- We establish a computational soundness result in the presence of key-cycles with respect to CCA-2 secure encryption schemes, when weak-public key indistinguishability is adopted as the notion of computational equivalence (Corollary 2).
- We prove that soundness cannot be established in the presence of key-cycles with respect to strong public-key indistinguishability (Theorem 3).
- We prove that in the absence of key-cycles the two indistinguishability notions, weak and strong, coincide (Claim 2).
- We show that in the presence of key-cycles, CCA-2 security does not give a complete computational semantics (Theorem 4), and we prove that in the absence of key-cycles, the completeness result holds not only with respect to CCA-2 secure encryption systems, but also with respect to any length-regular (not necessarily secure) encryption scheme (Claim 1).

Just as a quick comparison, the completeness result of [6] for private-key cryptography indicates that type-0 security, which was known to give soundness in the absence of key-cycles ([7]) does not give completeness even in the absence of key-cycles; to obtain completeness type-0 security must be extended to authenticated encryption. Our results give a completeness result based on any length-regular encryption scheme in the absence of key-cycles.

Related work. Computational soundness of equivalence for formal expressions has been addressed by many papers in recent years, starting with the work of Abadi and Rogaway [7] who prove the computational soundness of formal encryption in the absence of key-cycles with respect to type-0 security. In a paper by Micciancio and Warinschi [6], the completeness of Abadi-Rogaway logic is studied. In particular, the result of Micciancio and Warinschi shows that type-0 security does not give a complete computational semantics to formal expressions; completeness is obtained by strengthening the computational semantics. Computational soundness of cyclic encryptions has been addressed in a number of papers ([2], [8], [4], [3], [9]). In [8] computational soundness in the presence of key cycles is obtained by giving more deductive power to formal adversaries. In particular, Laud changes the set of derivability relations of Abadi-Rogaway logic by adding a specific rule which enables the formal adversary to break a key-cyclic expression. In [4], it is proved that attaining a sound computational semantics for key-cyclic expressions is beyond the power of standard notions of computational security, but may be obtained using a very strong notion of security, called KDM security [10]. In the same paper, it is also proved that the KDM security is not implied by any standard notion of security. In all these works, adversarial knowledge in the formal setting is defined inductively. In [3], Micciancio suggests a co-inductive method for formulating adversarial knowledge in order to obtain soundness in the presence of key-cyclic terms, and establishes the computational soundness of key-cyclic expressions in private key cryptography, where the computational semantics is based on CPA secure encryption schemes. Lei and Xue in [9] extend Micciancio's result to the case in which secret shares are used.

II. Formal public-key setting

A. Syntax

In this section, we review the generalization of Abadi-Rogaway logic to the case of asymmetric encryption, given in [1]. Let K_{pub} and K_{priv} be a set of public and private keys. We define $Keys = K_{pub} \cup K_{priv}$. Also assume that $Block$ is a fixed set (disjoint from the set of public and private keys) containing some basic messages. The set of *formal expressions* is given by the following grammar:

$$Exp ::= Block \mid K_{priv} \mid K_{pub} \mid \{Exp\}_{K_{pub}} \mid (Exp, Exp)$$

As in [1], we assume that there exists a bijection $inv : K_{pub} \rightarrow K_{priv}$ which maps the set of public keys to their private keys. We write K^{-1} to denote $inv(K)$ if K is a public key, and K^{-1} to denote $inv^{-1}(K)$ when K is a private key. Also, if T is a set of public or private keys, we define $T^{-1} = \{K^{-1} : K \in T\}$. Throughout the paper, when referring to K^{-1}, it can be realized from the context whether K^{-1} denotes some public key or private key symbol. In our work, we assume that the encryption might reveal certain information about the underlying plaintext. In particular, we assume the length and the *structure* of the plaintext are deducible from the encryption. To model this scenario in the formal setting, we need the following definition [1]:

Definition 1: The *structure* of a formal expression e, denoted by $struct(e)$, is defined as follows:

- if $e \in Block$, $struct(e) = \square$
- if e is a public key symbol, $struct(e) = \circ$
- if e is a private key symbol, $struct(e) = \circ_p$
- if $e = (e_1, e_2)$, $struct(e) = (struct(e_1), struct(e_2))$
- if $e = \{e_1\}_K$, $struct(e) = \{struct(e_1)\}_\circ$

Having defined the $struct$ function, we extend the class of expressions we consider to include *patterns*, defined as follows:

$$Pat ::= Exp \mid \{struct(Exp)\}_K \mid \{Pat\}_K \mid (Pat, Pat).$$

Henceforth we refer to the elements of Exp as expressions, and to the elements of Pat as patterns. The notion of symbolic equivalence is defined with respect to the pattern function P, which takes as input a set T of private keys and a pattern or structure e, and outputs the pattern or structure that is visible to an adversary having access to the keys in T: [1]

Definition 2: Let $Struct = \{struct(e) : e \in Exp\}$, and $Pat' = Pat \cup Struct$. We define $p : Pat' \times \mathcal{P}(K_{priv}) \rightarrow Pat'$ [1], the *public-key pattern function*, as follow:

- If $b \in Block$, $b \in Keys$, or $b \in Struct$, $p(b, T) = b$
- $p((e_1, e_2), T) = p(e_1, T), p(e_2, T))$
- $p(\{e\}_K, T) = \begin{cases} \{p(e, T)\}_K & K^{-1} \in T \\ \{struct(e)\}_K & \text{otherwise} \end{cases}$

Example 1: Suppose $e = (\{\{1\}_{K_1}\}_{K_2}, \{0\}_{K_3})$. We have:

$$e_1 = p(e, \{K_3^{-1}\}) = \{\{\{\square\}_\circ\}_{K_2}, \{0\}_{K_3})$$
$$e_2 = p(e_1, \{K_2^{-1}\}) = \{\{\{\square\}_\circ\}_{K_2}, \{\square\}_{K_3})$$

Next we define, $F_e(T)$, where T is a set of private keys and $e \in Pat$, which corresponds to the set of private keys that can be recovered from e by an adversary observing e and using set T for decryption [3]:

Definition 3: Suppose that T is a set of private keys and $e \in Pat \cup Struct$. We define $F_e : \mathcal{P}(K_{priv}) \rightarrow \mathcal{P}(K_{priv})$ *the key recovery function of* e recursively as follow:

- if $K \in K_{priv}$, $F_K(T) = K$
- if $b \in Block$, $b \in Struct$, or $b \in K_{pub}$, $F_b(T) = \emptyset$
- if $e = (e_1, e_2)$, $F_e(T) = F_{e_1}(T) \cup F_{e_2}(T)$
- if $e = \{e_1\}_K$,

$$F_e(T) = \begin{cases} F_{e_1}(T) & \text{if } K^{-1} \in T \\ \emptyset & \text{otherwise} \end{cases}$$

Example 2: Assume $e = (\{\{K_1^{-1}\}_{K_2}\}_{K_3}, K_2^{-1})$. Then

$$F_e(\{K_3^{-1}\}) = \{K_2^{-1}\}$$
$$F_e(\{K_3^{-1}, K_2^{-1}\}) = \{K_1^{-1}, K_2^{-1}\}.$$

[1]Note that we are including $Struct$ in the domain of function p (and the next functions introduced in this section) for technical reasons. In the rest of the paper, when we write $p(e, T)$, we are assuming that $e \in Pat$.

Note that $F_e(T)$ intuitively represents the set of keys which can be immediately inferred from e(only using keys of T for decryption). For instance, in the above example, although K_1^{-1} can be recovered from e by using the key set $\{K_3^{-1}\}$, but it is not immediately recoverable, and thus $K_1^{-1} \notin F_e(\{K_3^{-1}\})$.

For a formal expression e, we introduce two functions which extract the private and public keys of e.

Definition 4: We define the functions $privkeys : Pat \cup Struct \rightarrow \mathcal{P}(K_{priv})$ and $pubkeys : Pat \cup Struct \rightarrow \mathcal{P}(K_{pub})$ as follow:

- if $K \in K_{priv}$, $privkeys(K) = \{K\}$, $pubkeys(K) = \emptyset$
- if $K \in K_{pub}$, $privkeys(K) = \emptyset$, $pubkeys(K) = \{K\}$
- if $b \in Block$ or $b \in Struct$, $privkeys(b) = pubkeys(b) = \emptyset$
- $privkeys(e_1, e_2) = privkeys(e_1) \cup privkeys(e_2)$, $pubkeys(e_1, e_2) = pubkeys(e_1) \cup pubkeys(e_2)$
- $privkeys(\{e\}_K) = privkeys(e)$ $pubkeys(\{e\}_K) = pubkeys(e) \cup \{K\}$

Proofs of the following properties can be obtained by an easy structural induction on $e \in Pat$:

Proposition 1:
1) $privkeys(p(e, T)) = F_e(T)$
2) $p(p(e, T_1), T_2) = p(e, T_1 \cap T_2)$
3) $privkeys(p(e, T)) \subseteq privkeys(e)$
4) F_e is a monotone function: $T \subseteq S \Rightarrow F_e(T) \subseteq F_e(S)$

For a pattern e, as discussed earlier, we can associate a key recovery function F_e to it(definition 3). Now we say that a set T of private keys is a fixed point of F_e if $F_e(T) = T$. A set T_1 is the greatest fixed point of F_e if for every T that $F_e(T) = T$, it holds that $T \subseteq T_1$. We can define the least fixed point in a similar manner. In the following theorem, as in [3], we show the existence of the least fixed point and the greatest fixed point of F_e, which we denote by $fix(F_e)$ and $FIX(F_e)$ respectively. The idea of the proof is very much the same as its private key version [3], and we present it in the appendix section.

Lemma 1: Suppose that $e \in Pat$ and F_e is its associated key recovery function. Letting $n = |privkeys(e)|$, we have:

$$fix(F_e) = F_e^n(\emptyset) = \bigcup_i F_e^i(\emptyset)$$
$$FIX(F_e) = F_e^n(privkeys(e)) = \bigcap_i F_e^i(privkeys(e)).$$

Suppose that $e \in Pat$ and σ is a key bijection function(obviously the key function has the property that if $\sigma(k) = k_1$, then $\sigma(k^{-1}) = k_1^{-1}$). By $e\sigma$ we denote the formal term obtained from e by replacing each key k in e by $\sigma(k)$. Now we have the necessary tools to define the pattern of a formal expression. Following the same approach taken in [3], for $e \in Pat$ we define $pat(e) = p(e, FIX(F_e))$. We say two patterns e_1 and e_2 are symbolically equivalent, written as $e_1 \cong e_2$, if there exists a key bijection function σ under which $pat(e_1) = pat(e_2)\sigma$. Now suppose that T is a set of private keys. We can generalize the pattern definition when the adversary has priori information

T. For this purpose, denote by eT the pattern obtained from e by pairing it with all keys in T. We now define $pat(e,T) = p(e, FIX(F_{eT}))$. Note that we clearly have $pat(e) = pat(e,\emptyset)$. Also. the following fact can be easily verified:

$$pat(e_1,T) = pat(e_2,T) \Leftrightarrow pat(e_1T) = pat(e_2T)$$

Note that the above fact does not hold if we replace $=$ with $\cong$. For example, let $m = (\{0\}_{k_1}, \{1\}_{k_2})$ and $n = (\{0\}_{k_2}, \{1\}_{k_1})$, and $T = \{k_1^{-1}, k_2^{-1}\}$. We have $pat(m,T) \cong pat(n,T)$, $pat(mT) = (\{0\}_{k_1}, \{1\}_{k_2}, k_1^{-1}, k_2^{-1})$, and $pat(nT) = (\{0\}_{k_2}, \{1\}_{k_1}, k_1^{-1}, k_2^{-1})$. Now it is obvious that $pat(mT) \ncong pat(nT)$ (they are not equivalent even upto key-renaming).

Let $e \in Pat$. We say that public key k_1 encrypts private key k^{-1} in e, if there exists a pattern e_1 such that $k^{-1} \in privkeys(e_1)$ and $\{e_1\}_{k_1}$ appears in e. To every pattern e, we can associate an underlying key graph $G_e = (V_e, E_e)$ as follows:

$$V_e = \{(k, k^{-1}) \mid k \in pubkeys(e) \text{ or } k^{-1} \in privkeys(e))\}$$
$$E_e = \{((k, k^{-1}), (k_1, k_1^{-1}) \mid k \text{ encrypts } k_1^{-1} \text{ in } e\}.$$

A pattern e is *cyclic* (*acyclic*) if its underlying key graph is cyclic (acyclic). In the following lemma, we establish the relation between $FIX(F_e)$ and $fix(F_e)$. In particular, we show that as long as e is acyclic, we have $fix(F_e) = FIX(F_e)$. Note that the converse of this fact is not valid in general. For example if $e = (\{k^{-1}\}_k, k^{-1})$, then e is cyclic but $fix(F_e) = FIX(F_e)$.We remark that a symmetric version of this fact was proved in [3]. However, we can give an easier proof for the asymmetric case using some basics from graph theory. A proof is provided in the appendix.

Lemma 2: Suppose e is an acyclic pattern. We have: $fix(F_e) = FIX(F_e)$.

Given a pattern e and a set of private keys T, we want to specify the set of derivable sub-terms of e which are undecryptable with respect to T. We need this definition in the proof of our soundness results.

Definition 5: Let T be a set of private keys and $e \in Pat \cup Struct$. We define the set of *undecryptable expressions* of e with respect to T, denoted by $undec_e(T)$, as follows:

- if $e = (e_1, e_2)$, $undec_e(T) = undec_{e_1}(T) \cup undec_{e_2}(T)$.
- if $e = \{e_3\}_K$, then
$$undec_e(T) = \begin{cases} undec_{e_3}(T) & \text{if } K^{-1} \in T \\ \{e_3\}_K & \text{otherwise} \end{cases}$$
- if $e \neq (e_1, e_2)$ and $e \neq \{e_3\}_K$ then $undec_e(T) = \emptyset$

III. Computational setting

Definition 6: A *public key encryption scheme* is a tuple of probabilistic polynomial time algorithms $\Pi = (Gen, Enc, Dec)$ such that:

1) the key generation algorithm Gen takes as input the security parameter η and outputs a pair of public/private keys (pk, sk). We write this as : $(pk, sk) \leftarrow Gen(1^\eta)$
2) the encryption algorithm Enc takes as input a public key pk and a plaintext $m \in \{0,1\}^*$, and outputs a ciphertext c which is chosen according to the distribution $Enc(m, pk)$. We write this as: $c \leftarrow Enc_{pk}(m)$.
3) The decryption algorithm Dec takes as input a private key sk and a ciphertext c, and outputs the decryption of the ciphertext c under key sk. If (pk, sk) is a pair of keys output by Gen, we require that $Dec_{sk}(Enc_{pk}(m)) = m$ (so Dec must be deterministic.)

Convention Let $\vec{M} = (m_1, \ldots, m_n)$ be a vector of plaintexts and k be a key. We define: $Enc(\vec{M}, k) = (Enc(m_1, k), \ldots, Enc(m_n, k))$

Given a public-key encryption scheme Π, we can present an encoding which maps patterns into computational distributions. We first show how we can map expressions (i.e. member of Exp) into probability distributions, and then we extend the encoding function to include the computational interpretations of patterns.

Definition 7: Let $\Pi = (Gen, Enc, Dec)$ be a public-key encryption scheme and η be a security parameter. We define $[e]_\Pi^\eta$, *the computational encoding* of the expression e with respect to Π and η. We begin by setting $\langle \tau(K), \tau(K^{-1}) \rangle \leftarrow Gen(\eta)$ for each key K. We then have

- If K is a private key, $[K]_\Pi^\eta = \langle \tau(K), \text{“privkey”} \rangle$.
- If K^{-1} is a public key, $[K^{-1}]_\Pi^\eta = \langle \tau(K^{-1}), \text{“pubkey”} \rangle$.
- If $b \in Block$, $[b]_\Pi^\eta = \langle b', \text{“block”} \rangle$, where $|b'| = r$ for some fixed r.
- If $e = (e_1, e_2)$, $[(e_1, e_2)]_\Pi^\eta = \langle [e_1]_\Pi^\eta, [e_2]_\Pi^\eta, \text{“pair”} \rangle$.
- If $e = \{e_1\}_K$,
$$[e]_\Pi^\eta = \langle Enc([e_1]_\Pi^\eta, \tau(K)), \tau(K), \text{“ciphertext”} \rangle.$$

In the above definition, we call τ *the key-assignment function*, and $\langle\rangle$ *the pairing function*. Henceforth, when we say *bit-string* we mean some binary string in the range of the computational encoding function (i.e. the computational image of some formal expression). Also we use the general term *computational message* for denoting any binary string. Also if pk is a public key value, we denote any private key value of pk by pk^{-1}. Note that for a given pair of keys (pk, sk), it is possible to check in polynomial time whether sk is a private key of pk or not, using the fact this is the case iff for any x, $Dec_{sk}(Enc_{pk}(x)) = x$.

The inclusion of the underlying key in the computational image of encryptions is due to the fact that in public-key cryptography, an encryption does not hide its public key.[1]

In order to define the computational interpretation of patterns, we need to give a computational meaning to $struct(e)$

where $e \in Exp$. We do so by assigning $[struct(e)] = 0^{|[e]|}$ where $|x|$ represents the length of the bit-string x. In order to have this mapping well-defined, we will require that for all formal terms e_1 and e_2 , if $struct(e_1) = struct(e_2)$ then $|[e_1]| = |[e_2]|$. This condition can be guaranteed by requiring that the encryption function and the pairing function $\langle\rangle$ used in the above definition be length-regular (a function f is *length-regular*, if for every $x, y \in \{0,1\}^*$ we have: $|x| = |y| \Rightarrow |f(x)| = |f(y)|$). Also we assume that all elements of $Block$ are mapped to bit-strings of the same length [3].

We call an encryption scheme *length-regular* if its encryption function is length-regular. Note that if the encryption function is length-regular, it is obviously length-revealing; meaning that if $Enc_k(m) = c$, there exists an algorithm which can computer $|m|$ from $|c|$. We use the terms length-revealing and length-regular interchangeably in this paper.

We present a well-known notion of security in public-key cryptography, called CCA-2 security. This notion is typically defined with respect to a *single message CCA indistinguishability experiment* (see, e.g., [11] for more discussion). We will use a generalized version in which the experiment is run under a number of keys rather than just a single key and the adversary can submit multiple messages for encryption. We call this new experiment notion *multiple message CCA indistinguishability experiment*. It is not hard to see, using a standard hybrid argument, that the multiple-message-based definition is equivalent to the standard one.

Multiple message CCA indistinguishability experiment Given a public key encryption scheme $\Pi = (Gen, Enc, Dec)$ and security parameter η, we define the "Multiple message CCA indistinguishability experiment" as follows:

1) $G(1^\eta)$ is run r times to produce r pairs of public/private keys: $(pk_1, sk_1) \leftarrow G(1^\eta)$; ...; $(pk_r, sk_r) \leftarrow G(1^\eta)$
2) The adversary $\mathcal{A}$ is given $(pk_1, \ldots, pk_r)$ and access to the set of oracles $\{O_i^1(.)\}_{1 \le i \le r}$, where $O_i^1(x) = Dec(x, sk_i)$. For each key pk_i , $\mathcal{A}$ selects two vector of messages $\vec{M}_i = (m_i^1, \ldots, m_i^{p_i})$ and $\vec{N}_i = (n_i^1, \ldots, n_i^{p_i})$, with the restriction that $|m_i^j| = |n_i^j|$for all $1 \le j \le p_i$. Finally $\mathcal{A}$ outputs two vectors of vectors: $\vec{M} = (\vec{M}_1, \ldots, \vec{M}_r)$ and $\vec{N} = (\vec{N}_1, \ldots, \vec{N}_r)$.
3) A random bit $b \in \{0,1\}$ is chosen, and two vectors of ciphertexts are computed as follow : $\vec{B}_0 = (enc(\vec{M}_1, pk_1), \ldots, enc(\vec{M}_r, pk_r))$ and $\vec{B}_1 = (enc(\vec{N}_1, pk_1), \ldots, enc(\vec{N}_r, pk_r))$. Finally the vector $\vec{C} = \vec{B}_b$,the challenge ciphertext vector, is given to $\mathcal{A}$.
4) Let $\vec{C} = (\vec{C}_1, \ldots, \vec{C}_r)$ be the challenge ciphertext. $\mathcal{A}$ is now given access to the set of oracles $\{O_i^2(.)\}_{1 \le i \le r}$, where $O_i^2(x) = Dec(x, sk_i)$ if $x \notin \vec{C}_i$, and $O_i^2(x) = \perp$ if $x \in \vec{C}_i$. It finally outputs some bit g.
5) the probability of success is defined with respect to the probability that b = g .

Definition 8: A public-key encryption scheme $\Pi = (Gen, Enc, Dec)$ provides *indistinguishability of multiple encryptions under chosen-cipher text attack* (or *CCA-2 security*) if for all probabilistic polynomial time adversaries $\mathcal{A}$, the probability of success in the above experiment is negligible.

The notion of computational equivalence that we are going to see in our theorems is based on a stronger notion of computational indistinguishability, in which the distinguisher is granted access to oracles determined according to the input it receives from the two probability distributions. As in [1], we call the new computational indistinguishability notion *indistinguishability with respect to oracles*, and we formalize it in the following definition.

Definition 9: Suppose that $D = \{D_\eta\}$ and $D' = \{D'_\eta\}$ are two collections of probability distributions indexed by η. Let $\{O_x\}$ be a family of oracle sets. We say that the two ensembles D and D' are *computationally indistinguishable with respect to* $\{O_x\}$ if for all probabilistic polynomial time adversary $\mathcal{A}$ the function:

$$\epsilon(\eta) = pr[D(d \leftarrow \{D_\eta\} : \mathcal{A}^{O_d}(\eta, d) = 1] - pr[D(d \leftarrow \{D'_\eta\} : \mathcal{A}^{O_d}(\eta, d)]$$

is negligible.

Note that in the above definition, if the adversary receives d as input, its granted oracle is O_d. Thus, the underlying oracle is determined directly by the input to the adversary. We need to give one more definition before presenting the notions of equivalence in our computational model.

Definition 10: (Computational derivation) Let T be a set of public key values and s be a bit-string. We define the computational derivability relation $\vdash^T$ as the least relation with the following properties:

1) $s \vdash^T s$
2) if $s \vdash^T \langle s_1, s_2, \text{"pair"}\rangle$ then $s \vdash^T s_1$ and $s \vdash^T s_2$.
3) if $s \vdash^T \langle c, k, \text{"ciphertext"}\rangle$ and $s \vdash^T \langle k^{-1}, \text{"privkey"}\rangle$ then $s \vdash^T Dec_{k^{-1}}(c)$.
4) if $s \vdash^T \langle c, k, \text{"ciphertext"}\rangle$ and $k \in T$ then $s \vdash^T Dec_{k^{-1}}(c)$.

Let E be a bit-string and T be a set of computational public keys. We define the set of all ciphertexts visible in E relative to T, denoted by $Cipher_E(T)$, as follow: if $E \vdash^T \langle c, k, \text{"ciphertext"}\rangle$, add c to $Cipher_E(T)$

In the following, we describe the characterizations of two indistinguishability notions in public key cryptography, which constitute the notions of equivalence in computational models. The first notion of indistinguishability is the one introduced in [1] which we call strong-public key indistinguishability. We also present another notion

of indistinguishability, weak public key indistinguishability, which has a much simpler formulation than the strong version. We prove that in the absence of key-cycles, which was the framework of [1], these two indistinguishability notions collapse. In the presence of key-cycles, we establish the computational soundness result when weak public key indistinguishability is taken as the notion of computational equivalence. By a counter-example, we show that the computational soundness result does not hold in the presence of key-cycles for strong indistinguishability notion.

Definition 11: Let T be a set of public key symbols, and τ be a fixed key assignment function (definition 7). Given a public key encryption scheme Π, we say that $[e_1]_\Pi$ is *strongly computationally indistinguishable from* $[e_2]_\Pi$, written as $[e_1]_\Pi \approx_s^{O_x^{e_1,e_2,T}} [e_2]_\Pi$, if for all probabilistic polynomial time adversaries $\mathcal{A}$, and key assignment functions τ the function:

$$\epsilon(\eta) = pr[D(d \leftarrow [e_1]_\Pi^\eta : \mathcal{A}^{O_d^{e_1,e_2,T}}(\eta, d) = 1] - pr[D(d \leftarrow [e_2]_\Pi^\eta : \mathcal{A}^{O_d^{e_1,e_2,T}}(\eta, d) = 1]$$

is negligible; where $O_d^{e_1,e_2,T}(\sigma, pk)$ returns $\perp$ unless pk is a valid public key and:

1) either $pk = \tau(k)$ for some $k \in T$, or
2) $pk = \tau(k)$ for some $k \in pubkeys(e_1) \cup pubkeys(e_2)$ and $\sigma \notin Cipher_d(\tau(T))$

In these cases, the oracle returns $Dec_{pk^{-1}}(\sigma)$. Note that in the above formulation, both $[e_1]_\Pi^\eta$ and $[e_2]_\Pi^\eta$ are evaluated under τ.

As a notational convenience, if $pubkeys(e_1) \subseteq pubkeys(e_2)$, we can rewrite the above indistinguishability notion as $\approx_s^{O_x^{e_2,T}}$. The result of [1] indicates that in the absence of key-cycles, if $pubkeys(e_1) = pubkeys(e_2)$ and Π is a CCA-2 secure encryption scheme, then $pat(e_1, T) = pat(e_2, T)$ implies $[e_1]_\Pi \approx_s^{O_x^{e_1,T}} [e_2]_\Pi$. We now present a weaker version of the above definition.

Definition 12: Let T be a set of computational public key symbols, and τ be a fixed key assignment function. Given a public key encryption scheme Π, we say that $[e_1]_\Pi$ is *weakly computationally indistinguishable from* $[e_2]_\Pi$, written as $[e_1]_\Pi \approx_w^{O^T} [e_2]_\Pi$, if for all probabilistic polynomial time adversaries $\mathcal{A}$, and key assignment functions τ, the function:

$$\epsilon(\eta) = pr[D(d \leftarrow [e_1]_\Pi^\eta : \mathcal{A}^{O^T}(\eta, d) = 1] - pr[D(d \leftarrow [e_2]_\Pi^\eta : \mathcal{A}^{O^T}(\eta, d) = 1]$$

is negligible; where :

$$O^T(\sigma, pk) = \begin{cases} Dec_{pk^{-1}}(\sigma) & pk \in \tau(T) \\ \perp & \text{otherwise} \end{cases}$$

Note that in the above formulation both $[e_1]_\Pi^\eta$ and $[e_2]_\Pi^\eta$ are evaluated under τ.

Henceforth, we write $\approx_s^{e_1,e_2,T}$ and $\approx_w^T$ when referring to strong and weak public key indistinguishability. We may omit e_1 and e_2 when they are clear from the context. Obviously strong public-key indistinguishability implies weak public-key indistinguishability. Note that in the case of weak public key indistinguishability, if $T = \emptyset$ we arrive at the standard notion of computational indistinguishability [7], and we write it just as $\approx_w$.

IV. Computational soundness

In this section, we explore the computational soundness of formal expressions in the presence of key-cycles. We consider the two notions of computational equivalence separately. We first establish soundness when weak public-key indistinguishability is the notion of equivalence, and then we show that the same result does not hold for strong public-key indistinguishability.

A. Weak public-key indistinguishability

We establish computational soundness with respect to weak public-key indistinguishability as the notion of computational equivalence.

Theorem 1: Suppose that $\Pi = (Gen, Enc, Dec)$ is a CCA-2 secure public key encryption scheme, and e is a pattern(i.e. $e \in Pat$). Define $privkeys^{-1}(e) = \{k | k^{-1} \in privkeys(e)\}$. We then have

$$[e]_\Pi \approx_w^{privkeys^{-1}(e)} [p(e, privkeys(e))]_\Pi.$$

Proof: Suppose, for the sake of contradiction, that there exists an adversary $\mathcal{A}^{O^{privkeys^{-1}(e)}}$ which can distinguish between the two probability distributions $[e]_\Pi$ and $[p(e, privkeys(e))]_\Pi$. We use $\mathcal{A}$ to construct a new adversary $\mathcal{B}$ which breaks the CCA-2 security Π (definition 8) . The adversary $\mathcal{B}$ works as follow: for every private key $k^{-1} \in privkeys(e)$, it runs the key generation algorithm to obtain a pair of key values for (k, k^{-1}) . Letting $R = pubkeys(e) - privkeys^{-1}(e) = \{k_1, \ldots, k_{|R|}\}$ (note that R is the set of public key symbols whose private keys do not appear in e), we assume that the multiple message indistinguishability experiment that $\mathcal{B}$ is involved in is run under $|R|$ public keys $pk_1, \ldots, pk_{|R|}$, and thus we can think of pk_i as the public key value of k_i. Now for each pk_i, $\mathcal{B}$ creates two vectors of messages $\vec{M}_i$ and $\vec{N}_i$, which represent the set of messages that $\mathcal{B}$ submits to be encrypted under key pk_i. Let $undec_e(privkeys(e))$ be the set of undecryptable messages of e with respect to $privkeys(e)$ (see definition 5). For all $m \in undec_e(privkeys(e))$, if $m = \{r\}_{k_i}$, $\mathcal{B}$ generates two bit-strings: $m \leftarrow [r]$ and $n = 0^{|m|}$, and adds m to vector $\vec{M}_i$ and n to vector $\vec{N}_i$. Note that in order for $\mathcal{B}$ to evaluate $[r]$, it does not need to know the corresponding private keys of any of the keys in R, because their private keys do not appear as a message in e. After preforming this operation for all elements in $undec_e(privkeys(e))$, it finally submits the two vectors $\vec{M} = (\vec{M}_1, \ldots, \vec{M}_{|R|})$ and

$\vec{N} = (\vec{N_1}, \ldots, \vec{N_{|R|}})$. When provided with the challenge ciphertext vector $\vec{C}$, it uses $\vec{C}$ to evaluate $[e]$, and gives the resulting bit-string to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. Note that $\mathcal{B}$ is able to answer all queries of $\mathcal{A}$ because it has the key values of all private key symbols in $privkey(e)$. Now if the challenge ciphertext $\vec{C}$ is an encryption of $\vec{M}$, then $\mathcal{A}$ is given a sample from $[e]$, and if $\vec{C}$ is an encryption of $\vec{N}$ then $\mathcal{A}$ is given a sample from $[p(e, privkeys(e))]$. So if $\mathcal{A}$ is able to distinguish between $[e]$ and $[p(e, privkeys(e))]$ with non-negligible probability, then $\mathcal{B}$ breaks the "multiple indistinguishability of encryptions" property of Π, completing the proof. ■

Note that a weaker version of the above theorem clearly holds; namely, if $T \subseteq privkeys(e)$ then:

$$[e]_\Pi \approx_w^{T^{-1}} [p(e, privkeys(e))]_\Pi.$$

Theorem 2: Suppose that $\Pi = (Gen, Enc, Dec)$ is a CCA-2 secure public key encryption scheme. If e is a pattern then $[e]_\Pi \approx_w^{FIX^{-1}(F_e)} [pat(e)]_\Pi$

Proof: Let $T = pubkeys^{-1}(e) \cup privkeys(e)$. It is obvious that $e = p(e, T)$. Also suppose that n is the smallest number for which $F_e^n(T) = F_e^{n+1}(T)$, so we have $FIX(F_e) = F_e^n(T)$. For $0 \leq i \leq n$, let e_i denote $p(e, F_e^i(T))$. Note that we then have $e_0 = e$, $\ldots$, $e_n = pat(e)$. Letting $T_i = F_e^i(T)$, we prove that for every i: $[e_i] \approx_w^{T_{i+1}^{-1}} [e_{i+1}]$. Note that since $FIX^{-1}(F_e) \subseteq T_i^{-1}$, it also holds that $[e_i] \approx_w^{FIX^{-1}(F_e)} [e_{i+1}]$, and this will imply that $[e]_\Pi \approx_w^{FIX(F_e)^{-1}} [pat(e)]_\Pi$.

Now we prove: $[e_i] \approx_w^{T_{i+1}^{-1}} [e_{i+1}]$. According to theorem 1, we have $[e_i] \approx_w^{privkeys^{-1}(e_i)} [p(e_i, privkeys(e_i)]$. On the other hand, $privkeys(e_i) = privkeys(p(e, F_e^i(T))) =^{proposition1} F_e^{i+1}(T) = T_{i+1}$. Therefore we have: $[e_i] \approx_w^{T_{i+1}^{-1}} [p(e_i, F_e^{i+1}(T)]$. Substituting $e_i = p(e, F_e^i(T))$, we will get: $[p(e, F_e^i(T))] \approx_w^{T_{i+1}^{-1}} [p(p(e, F_e^i(T)), F_e^{i+1}(T))]$. Also according to proposition 1, we have $p(\ p(e, F_e^i(T))\ ,\ F_e^{i+1}(T)\ ) = p(e, F_e^i(T) \bigcap F_e^{i+1}(T)) = p(e, F_e^{i+1}(T))$. Thus $[p(e, F_e^i(T))] \approx_w^{T_{i+1}^{-1}} [p(e, F_e^{i+1}(T)]$, and this completes the proof.

■

Corollary 1: Suppose that m and n are two patterns such that $m \cong n$ without key renaming (i.e. $pat(m) = pat(n)$). Then it holds that $FIX(F_m) = FIX(F_n)$, and moreover if $\Pi = (Gen, Enc, Dec)$ is a CCA-2 secure public key encryption scheme, we have: $[m]_\Pi \approx_w^{FIX(F_m)^{-1}} [n]_\Pi$.

Proof: In order to prove $FIX(F_m) = FIX(F_n)$, we prove $FIX(F_{pat(m)}) = FIX(F_m)$, and since $pat(m) = pat(n)$, this yields $FIX(F_m) = FIX(F_n)$. In particular, we show that $F_{pat(m)}(T) = T$ iff $F_m(T) = T$ (i.e. T is a fixed point of $F_{pat(m)}$ if and only if T is a fixed point of F_m). So suppose that $F_{pat(m)}(T) = T$ (T is a fixed point of $F_{pat(m)}$). We obtain:

$$\begin{aligned} T &= F_{pat(m)}(T) \\ &= privkeys(p(pat(m), T)) && \text{(Prop. 1)} \\ &\subseteq privkeys(pat(m)) && \text{(Prop. 1)} \\ &= FIX(F_m) \end{aligned}$$

Now having $T \subseteq FIX(F_m)$ we can write:

$$\begin{aligned} T &= F_{pat(m)}(T) \\ &= privkeys(p(pat(m), T)) && \text{(Prop. 1)} \\ &= privkeys(p(p(m, FIX(F_m)), T)) \\ &= privkeys(p(m, FIX(F_m) \cap T)) && \text{(Prop. 1)} \\ &= privkeys(p(m, T)) \\ &= F_m(T) && \text{(Prop. 1)} \end{aligned}$$

therefore T is a fixed point of F_m too. For the other direction, suppose $F_m(T) = T$. Thus, it holds that $T \subseteq FIX(F_m)$. The rest is completely the same as above, and this completes the proof of the first part. For the second part, note that according to theorem 2 we have:

$$[m]_\Pi \approx_w^{FIX^{-1}(F_m)} [pat(m)]_\Pi = [pat(n)]_\Pi \approx_w^{FIX^{-1}(F_n)} [n]_\Pi$$

and since $FIX(F_m) = FIX(F_n)$, the transitivity holds and $[m]_\Pi \approx_w^{FIX^{-1}(F_m)} [n]_\Pi$. ■

Note that in the above corollary, if $m \cong n$ with key-renaming (i.e. $pat(m) = pat(n)\sigma$ for some non-identical key-bijection function σ), we cannot arrive at a general result as the above one. First of all, note that since $pat(m)$ and $pat(n)$ are just different in key names, we always have $[pat(m)]_\Pi \approx_w [pat(n)]_\Pi$ (i.e. $T = \emptyset$). On the other hand, since $pat(m)$ and $pat(n)$ have different key names, we cannot in general find a set T, rather than $T = \emptyset$, for which $[pat(m)]_\Pi \approx_w^T [pat(n)]_\Pi$. Thus, in this case, the most general result that we can achieve is $[m]_\Pi \approx_w [n]_\Pi$.

Corollary 2: (Computational soundness) Let m and n be two formal expressions, and Π be a CCA-2 secure encryption scheme. It holds that:

$$pat(m, T) = pat(n, T) \rightarrow [m]_\Pi \approx_w^{FIX^{-1}(F_{mT})} [n]_\Pi.$$

Proof: Note that as we mentioned before, we have $pat(m, T) = pat(n, T) \Leftrightarrow pat(mT) = pat(nT)$. Thus, we just need to prove that $pat(mT) = pat(nT)$ implies $[m]_\Pi \approx_w^{FIX^{-1}(F_{mT})} [n]_\Pi$. Therefore, suppose $pat(mT) = pat(nT)$. According to corollary 1, we have $FIX(F_{mT}) = FIX(F_{nT})$. Also according to theorem 2, we have $[mT]_\Pi \approx_w^{FIX^{-1}(F_{mT})} [pat(mT)]_\Pi$, and $[nT]_\Pi \approx_w^{FIX^{-1}(F_{nT})} [pat(nT)]_\Pi$. Note that since $pat(mT) = pat(nT)$, we have $[pat(mT)]_\Pi = [pat(nT)]_\Pi$.

Thus, we obtain $[mT]_\Pi \approx_w^{FIX^{-1}(F_{mT})} [nT]_\Pi$. This implies that $[m]_\Pi \approx_w^{FIX^{-1}(F_{mT})} [n]_\Pi$, and the proof is complete.

■

Note that since $T \subseteq FIX(F_{mT})$, the result of corollary 2 is much stronger than the following fact: $pat(m,T) = pat(n,T) \rightarrow [m]_\Pi \approx_w^{T^{-1}} [n]_\Pi$.

B. Strong public key indistinguishability

We show, by a counter example, that the soundness result presented for weak public key indistinguishability cannot be attained if we adopt strong indistinguishability as the notion of computational equivalence. The result of [1] indicates that in the absence of key-cycles, if $pat(m,T) = pat(n,T)$ and $pubkeys(m) = pubkeys(n)$, then it holds that $[m]_\Pi \approx_s^{m,T^{-1}} [n]_\Pi$. Our result shows that the same thing does not hold in the presence of key-cycles.

Theorem 3: Under CCA-2 secrecy-based computational semantics, strong public-key indistinguishability does not give a sound computational interpretation to formal equivalence in the presence of key-cycles.

Proof: We show this fact by a counter-example. Suppose $\Pi = (Gen, Enc, Dec)$ is a CCA-2 secure encryption scheme. Define $\Pi' = (Gen, Enc', Dec')$ as follows:

$$Enc'(x,pk) = \begin{cases} x0^{|Enc(x,pk)|-|x|} & x = \langle pk^{-1}, \text{“privkey”}\rangle \\ Enc(x,pk) & \text{otherwise} \end{cases}$$

It is not hard to see that Π' is also CCA-2 secure. In fact, Enc' is defined exactly like $Enc(x,pk)$ if $x \neq \langle pk^{-1}, \text{“privkey”}\rangle$. On the other hand, since Π is CCA-2 secure, during any single message experiment, the probability that an adversary outputs such a message for encryption is negligible. Thus Π' is also CCA-2 secure (we omit the details). Now let $e_1 = (\{\{0\}_{k_1}\}_k, \{k^{-1}\}_k)$ and $e_2 = (\{\{1\}_{k_1}\}_k, \{k^{-1}\}_k)$. Note that $pattern(e_1, \emptyset) = pattern(e_2, \emptyset)$. Now we claim that we have :

$$[e_1]^\eta_{\Pi'} \not\approx_s^\emptyset [e_2]^\eta_{\Pi'}.$$

Suppose that $e_i = (\{\{i\}_{k_1}\}_k, \{k^{-1}\}_k)$, and the adversary is given $E_i \leftarrow [e_i]^\eta_{\Pi'}$, where $i \in \{1,2\}$. If we let $E_i = (E_i^1, E_i^2)$, note that according to the definition of Enc', the adversary can recover the key value associated with k^{-1} from E_i^2. Call this key value K^{-1}. It can then use K^{-1} to recover $D_i = Dec_{K^{-1}}(E_i^1)$ (note that D_i would be the image of $\{i\}_{k_1}$.) Suppose $D_i = \langle C_i, K_1, \text{“ciphertext”}\rangle$. It is easy to see that we have $C_i \notin Cipher_{E_i}(\emptyset)$, and thus the adversary can ask its oracle the query $\sigma(C_i, K_1)$, and consequently it can recover i. ■

V. Computational completeness

We prove that in the absence of key-cycles, the computational semantics based on any length-regular encryption scheme is complete. However, by providing a counter-example, we prove that the completeness result does not hold even with respect to CCA-2 secure encryption schemes in the presence of key-cycles.

A. Presence of key-cycles

We show that in the presence of key-cycles, CCA-2 security does not provide completeness. We, however, need to make an assumption in our proof for incompleteness; we assume that there exists a CCA-2 secure encryption scheme Π which has the property that there exists only one private key for any given public key. In other words, if η is a security parameter, and $(pk, sk_1) \in Gen(\eta)$, $(pk, sk_2) \in Gen(\eta)$ then $sk_1 = sk_2$. We call such an encryption scheme *key-bijective*, and we show that under this assumption, completeness does not hold.

Theorem 4: The computational semantics of definition 7 is not complete in the presence of key-cycles if there exists a CCA-2 secure key-bijective encryption scheme.

Proof: We prove that there exists terms m and n such that $m \not\cong n$ but their computational encodings are indistinguishable. Let $m = \{k^{-1}\}_k$ and $n = \{k_1^{-1}\}_k$. We have $pat(m) = \{k^{-1}\}_k$ and $pat(n) = \{\circ\}_k$, and consequently $m \not\cong n$. Now suppose that $\Pi = (Gen, Enc, Dec)$ is a *key-bijective* CCA-2 secure encryption scheme. If $[m]_\Pi \approx [n]_\Pi$ then we are done. Otherwise let $\Pi' = (Gen, Enc', Dec')$ be another encryption scheme which has the same key generation algorithm as Π and its encryption and decryption functions are defined as follow:

$$Enc'(x,pk) = \begin{cases} Enc(x0,pk) & x \neq \langle pk^{-1}, \text{“privkey”}\rangle \\ Enc(1^{|x|+1},pk) & \text{otherwise} \end{cases}$$

$$Dec'(c,sk) = \begin{cases} x & (x0 = Dec(c,sk)) \\ \langle sk, \text{“privkey”}\rangle & (Dec(c,sk) = 1^\ell) \\ \perp & \text{otherwise} \end{cases}$$

where $\ell = |\langle sk, \text{“privkey”}\rangle| + 1$. As explained before, Π' is also CCA-2 secure. Note that key-bijectiveness here guarantees that the decryption function is well-defined. In other words, if we did not have this condition and sk_1 and sk_2 were both private keys of pk, then $Enc_{pk}(sk_1) = Enc_{pk}(sk_2)$, and thence the decryption function would not have been well-defined. Now note that we have:

$$\begin{aligned} [n]_{\Pi'} &= [\{k_1^{-1}\}_k]_{\Pi'} \\ &= \langle Enc'([k_1^{-1}], \tau(k)), \text{“ciphertext”}\rangle \\ &= \langle Enc([k_1^{-1}]0, \tau(k)), \text{“ciphertext”}\rangle \end{aligned}$$

$$\begin{aligned} [m]_{\Pi'} &= [\{k^{-1}\}_k]'_\Pi \\ &= \langle Enc'([k^{-1}], \tau(k)), \text{“ciphertext”}\rangle \\ &= \langle Enc(1^{|[k^{-1}]|+1}, \tau(k)), \text{“ciphertext”}\rangle \end{aligned}$$

$$= \langle Enc(1^{|[k_1^{-1}]|+1}, \tau(k)), \text{“ciphertext”} \rangle$$

Now since Π is CCA-2 secure, it holds that $Enc(\ 1^{|[k_1^{-1}]|+1}, \tau(k)\ ) \approx Enc(\ [k_1^{-1}]0, \tau(k)\ )$, which implies that $[m]_{\Pi'} \approx [n]_{\Pi'}$, and this completes the proof. ■

B. Absence of key-cycles

The main result of [1] implies that if e_1 and e_2 are two key-acyclic formal expressions with the same set of public-key symbols and Π is a CCA-2 secure encryption system, then it holds that (corollary 17 of [1]):

$$pat(e_1, T) = pat(e_2, T) \ \Rightarrow\ [e_1]_\Pi \approx_s^{T^{-1}} [e_2]_\Pi \qquad (*)$$

We extend the result of [1] in two directions:

Claim 1: (*Computational completeness*) We claim that the converse of (*) also holds with respect to any length-revealing (length regular) encryption scheme Π; namely, we prove that if Π is a length-regular encryption scheme, then

$$[e_1]_\Pi \approx_w^{T^{-1}} [e_2]_\Pi \ \Rightarrow\ pat(e_1, T) \cong pat(e_2, T)$$

which implies

$$[e_1]_\Pi \approx_s^{T^{-1}} [e_2]_\Pi \ \Rightarrow\ pat(e_1, T) \cong pat(e_2, T).$$

As a special case, letting $T = \emptyset$ we get :

$$[e_1]_\Pi \approx [e_2]_\Pi \ \Rightarrow\ e_1 \cong e_2$$

Claim 2: (*Equivalence of weak and strong indistinguishability*) We claim that in the absence of key-cycles and with respect to any CCA-2 secure encryption scheme, the two indistinguishability notions we have introduced (weak and strong) collapse up to renaming of keys. In other words, suppose Π is a CCA-2 secure encryption scheme. We then have:

$$[e_1]_\Pi \approx_w^T [e_2]_\Pi \ \Rightarrow\ \exists\sigma\ [e_1]_\Pi \approx_s^T [e_2\sigma]_\Pi$$

Note that we obviously have

$$[e_1]_\Pi \approx_s^T [e_2]_\Pi \ \Rightarrow\ [e_1]_\Pi \approx_w^T [e_2]_\Pi$$

Thus these two notions of indistinguishability are equivalent up to renaming of keys.

Note that the result of claim 2 directly follows from the result of of claim 1. In order to see this, suppose $[e_1]_\Pi \approx_w^T [e_2]_\Pi$. Now according to the result of claim 1 we have $pat(e_1, T) \cong pat(e_2, T)$, and consequently for some key-renaming function σ, we have $pat(e_1, T) = pat(e_2\sigma, T)$. Now according to [1], we have: (recall that the result of [1] is based on CCA-2 secure encryption schemes)

$$pat(e_1, T) = pat(e_2\sigma, T) \ \Rightarrow\ [e_1]_\Pi \approx_s^T [e_2\sigma]_\Pi$$

We establish the proof of claim 1 through a sequence of lemmas and theorems. We first show its validity for the special case $T = \emptyset$, and then we show how we can generalize the idea of the proof for any set T.

Theorem 5: (*Completeness theorem:* $T = \emptyset$) Suppose e_1 and e_2 are two key-acyclic formal expressions, and Π is a length regular encryption scheme. If $e_1 \ncong e_2$, then there exists a polynomial time adversary $\mathcal{A}$ which can distinguish between the two probability distributions $[e_1]_\Pi^\eta$ and $[e_2]_\Pi^\eta$ with non-negligible probability.

We now turn our attention to proving the above theorem. First of all, we consider the behavior of the formal and computational settings in asymmetric cryptography in more detail. In the formal setting, if two expressions e_1 and e_2 have different structures, their encryptions under the same key symbol are considered inequivalent. As a result, in order to have completeness, the computational images of e_1 and e_2 should be of different length. In other words:

$$struct(e_1) \neq struct(e_2) \Rightarrow |[e_1]| \neq |[e_2]| \qquad (1)$$

This is the very least condition that the computational semantics has to meet in order to give completeness. As a result, we have to see under what conditions we can ensure that two formal expressions of different structures yield different computational images of different lengths. As it can be seen, the length of the computational image of a formal expression e depends both on the underlying encryption function and the pairing function (see definition 7). On the other hand, the encryption function is not determined by us, it depends on the encryption scheme, and the length function could be different from one encryption scheme to another. As a result, we have to examine if we can come up with an appropriate pairing function relative to which condition 1 holds, no matter what the encryption function is. Note that for many pairing functions the condition 1 is very unlikely to hold. For example, suppose e_1 and e_2 have different structures. Then it is likely that under an inappropriately-chosen pairing function, the two terms (e_1, e_2) and (e_2, e_1) will be evaluated to equal-length bit-strings.

In the following, we show that there exists a computational pairing function relative to which condition 1 holds.

Proposition 2: There exists a polynomial time computable one-to-one function $\langle ., . \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ which has the following properties:

1) $|\langle x, y \rangle| = |\langle z, h \rangle| \Leftrightarrow |x| = |z|$ and $|y| = |h|$
2) The function $h(l_1, l_2) = |\langle x_1, x_2 \rangle|$, where $|x_1| = l_1$ and $|x_2| = l_2$ is well-defined, and is the length function of $\langle\rangle$. Moreover, both h and h^{-1} are polynomial-time computable.

Proof: Define : $f(x, y) = 1^{|x|}0xy$. Obviously f is a one-to-one function, and both f and f^{-1} are polynomial time computable. Moreover, we have: $|f(x, y)| \leq 2|x| + 2|y|$ and $poly(x + y) > f(x, y) > x + y$. Now we define $\langle ., . \rangle$ as follows:

$$\langle x, y \rangle = f(x, y)01^{f(2|x|, 2|y|) - |f(x,y)| - 1}.$$

Note that $|\langle x,y\rangle| = f(2|x|,2|y|)$. Since f is a one-to-one function, this implies that:

$$|\langle x,y\rangle| = |\langle z,h\rangle| \Leftrightarrow |x| = |z| \text{ and } |y| = |h|$$

For the second part we have: $h(l_1,l_2) = f(2l_1,2l_2)$, and thus both h and h^{-1} are polynomial-time computable. This completes the proof. ■

From now on, we assume that the pairing function is the function $\langle .,. \rangle$ introduced in the above proposition. Based on what we have proved so far, we can now give a precise definition of a computational semantics which satisfies condition 1.

(Computational encoding - Revisited) Recall that every bit-string is tagged with its type (i.e. "Block", "pubkey", ...). We assume that $|\text{"Block"}| = 1, |\text{"pubkey"}| = 2, |\text{"privkey"}| = 3, |\text{"pair"}| = 4$, and $|\text{"ciphertext"}| = 5$. Also we take the function $\langle .,. \rangle$ introduced in the above proposition as our pairing function. We write $\langle x_1,\ldots,x_n\rangle$ for $\langle\langle\langle x_1,x_2\rangle,\ldots\rangle,x_n\rangle$. Besides the length of tags, we make one more adjustment in the computational encoding function of definition 7 by defining $[k^{-1}]^\eta_\Pi = \langle \tau(k^{-1}),\tau(k),\text{"privkey"}\rangle$. In other words, we assume that the computational image of a private key symbol includes the computational image of its underlying public key. Although, this provision is well-justified in public-key cryptography, we will show at the end of this section that one can ignore such an assumption and still obtain the same completeness results, perhaps with more involved proofs.

Theorem 6: Let e_1 and e_2 be two formal expressions. It holds that:

$$struct(e_1) = struct(e_2) \Leftrightarrow |[e_1]^\eta_\Pi| = |[e_2]^\eta_\Pi|.$$

Proof: The proof of the only-if direction ($\Rightarrow$) is straightforward. For the if direction ($\Leftarrow$) note that if two formal expressions are of different types (i.e. have different tags), the length of their computational images is different (this is because of the fact that the function $\langle .,. \rangle$ satisfies the condition specified in proposition 2). Using this fact, we can easily prove the if direction by an structural induction on e_1. We omit the details. ■

We now go on to proving our main theorem of completeness. We take the same approach taken in [6]; we prove that if two formal expressions are not equivalent, then there exists an adversary which can distinguish between their corresponding probability distributions. We establish this by showing that there exists an adversary which can compute $pat(e)$ with non-negligible probability from $E \leftarrow [e]^\eta_\Pi$.

Remark 1: Suppose τ is the the key assignment function, which assigns a key value to each key symbol. Note that we have:

$$pr[\tau(k_1) = \tau(k_2)] = negl(\eta), \text{where } k_1 \neq k_2$$
$$pr[\tau(k_1^{-1}) = \tau(k_2^{-1})] = negl(\eta), \text{where } k_1^{-1} \neq k_2^{-1}$$

We implicitly use this simple fact in some of our proofs.

Definition 13: Suppose Π is an encryption scheme and τ is a key assignment which assigns a key value to each key symbol. Let T_c be the set of bit-strings corresponding to the images of all private-key symbols, and E be a bit-string. In other words,

$$T_c = \{[k^{-1}]^\eta_\Pi : k^{-1} \text{ is a private key symbol}\}^2.$$

We define $F'_E : \mathcal{P}(T_c) \rightarrow \mathcal{P}(T_c)$, *the computational key recovery function*, recursively as follow: ($T'_c \in \mathcal{P}(T_c)$)

- If $E = \langle pk, \text{"pubkey"}\rangle$, $F'_E(T'_c) = \emptyset$.
- If $E = \langle sk, pk, \text{"privkey"}\rangle$,

$$F'_E(T'_c) = \langle sk, pk, \text{"privkey"}\rangle$$

- If $E = \langle A, B, \text{"pair"}\rangle$, $F'_E(T'_c) = F'_A(T'_c) \cup F'_B(T'_c)$.
- If $E = \langle C, pk, \text{"ciphertext"}\rangle$, there are two cases:
  1) If there exists exactly one sk for which $\langle pk, sk, \text{"privkey"}\rangle \in T$, then
  $$F'_E(T'_c) = F'_{Dec(C,sk)}(T'_c)$$
  2) otherwise: $F'_E(T_c) = \emptyset$

Recall the key recovery function F introduced in the formal setting. Suppose that $e \in Exp$ and the key assignment function τ is fixed. The following lemma shows that F', the computational key recovery function, is in fact a computational counterpart of F. In other words, if T is a set of private key symbols, T_c is the set of their computational images, and $E \leftarrow [e]$, then $[F_e(T)] = F'_E(T_c)$ with overwhelming probability. The proof of the lemma can be easily obtained by an induction on the structure of e. We give a detailed proof in the appendix.

Lemma 3: Suppose Π is an encryption scheme, η is a security parameter, T is a set of private key symbols, and $e \in Exp$. Under a fixed key assignment function τ, let $E \leftarrow [e]^\eta_\Pi$ and $T_c = [T]^\eta_\Pi$. Also, assume that $T'_c = [F_e(T)]^\eta_\Pi$. We have :

$$pr[T'_c \neq F'_E(T_c)] \leq negl(\eta).$$

Note that since we are dealing with key-acyclic expressions, according to lemma 2, it holds that $FIX(F_e) = fix(F_e)$. Therefore, from now on, we focus our attention on the least fixed point. In particular, having defined the computational key recovery function, we can define its least fixed point in the usual way (i.e. the least fixed point of F'_E is the smallest set T_c for which $F'_E(T_c) = T_c$.) Suppose $e \in Exp$ and $E \leftarrow [e]^\eta_\Pi$ and $n = |privkeys(e)|$. Likewise lemma 1, it can be easily verified that the least fixed point of F'_E admits the following formula:

$$fix(F'_E) = \bigcup_i {F'}^i_E(\emptyset) = {F'}^n_E(\emptyset).$$

[2]Note that since the key assignment function τ is fixed, $[k^{-1}]^\eta_\Pi$ is just a bit-string.

Using the above formula, it can be seen that $fix(F'_E)$ is polynomial-time computable. The following lemma relates the computational least fixed point to the formal least fixed point:

Lemma 4: Suppose that $e \in Exp$, $n = |privkeys(e)|$, and Π is a length regular encryption scheme. Under a fixed key assignment τ, let $E \leftarrow [e]^\eta_\Pi$, and $T = [fix(F_e)]^\eta_\Pi$. It holds that:

$$pr[fix(F'_E) \neq T] \leq negl(\eta).$$

Proof: Note that $fix(F'_E) = {F'}^n_E(\emptyset)$ and $fix(F_e) = F^n_e(\emptyset)$. We have

$$\begin{aligned} pr[fix(F'_E) \neq T] &\leq pr[\exists i \leq n : [F^i_e(\emptyset)]^\eta_\Pi \neq {F'}^i_E(\emptyset)] \\ &\leq n \cdot negl(\eta) \\ &= negl(\eta) \end{aligned}$$

■

In theorem 6, we showed that under our proposed computational semantics, there exists a one-to-one correspondence between the length of a computational message and its structure. In the following lemma, we show that this correspondence is computationally constructible. That is, there exists a polynomial time adversary which can recover the structure of a computational message from its computational length.

Lemma 5: (*Structure recovery from computational length*) Let $e \in Exp$, $\Pi = (Gen, Enc, Dec)$ be a length regular encryption scheme, and $E \leftarrow [e]^\eta_\Pi$. There exists a polynomial time function which can compute the structure of e when given $|E|$ and η.

Proof: Since the adversary is given η, it can run the key generation algorithm to obtain a pair of public/private keys (k, k^{-1}). Let $f : \mathbb{N} \rightarrow \mathbb{N}$ be the length function of Enc_k which is by assumption one-to-one. Since the encryption function is publicly known, every adversary is able to computer f^{-1}. Also, let h be the length function of $\langle ., . \rangle$, namely if $|x_1| = l_1$ and $|x_2| = l_2$, then $h(l_1, l_2) = |\langle x_1, x_2 \rangle|$. Recall from proposition 2 that both h and $h^{-1}$are polynomial-time computable. Also, we write $h(l_1, l_2..., l_n)$ for $h(..., h(h(l_1, l_2), l_3), ..., l_n)$. We now present an algorithm which recovers the structure of a message from its length:

algorithm $STRUCT_REC(|E|)$
  if $|E| = h(r, |\text{“Block”}|)$
    return $\square$
  if $|E| = h(|k|, |\text{“pubkey”}|)$
    return $\circ$
  if $|E| = h(|k^{-1}|, |k|, |\text{“privkey”}|)$
    return $\circ_p$
  if $|E| = h(x_1, x_2, |\text{“pair”}|)$
    return $(STRUCT_REC(x_1), STRUCT_REC(x_2))$
  if $|E| = h(x, |k|, |\text{“ciphertext”}|)$
    return $\{STRUCT_REC(f^{-1}(x))\}_\circ$

Recall that the length of tags are different. Now using the fact that h is one-to-one, it can be seen by an straightforward induction on the structure of e that it holds that:

$$pr[STRUCT_REC(|E|) = struct(e)] = 1.$$

We omit the details of the proof. ■

We now have the necessary tools to present an algorithm which takes as input a bit-string and outputs the pattern of its underlying formal expression. We prove that the algorithm succeeds with probability $1 - negl(\eta)$.

Lemma 6: (*pattern recovery*) Let $e \in Exp$, T be a set of private-key symbols, $\Pi = (Gen, Enc, Dec)$ be a length regular encryption scheme, and $E \leftarrow [e]^\eta_\Pi$. There exists an adversary $\mathcal{A}$ which can computer $pat(e)$ when given E and η with probability $1 - negl(\eta)$.

Proof: In the proof of this theorem, we use K, $K_1, \ldots$, for referring to key values and k, $k_1, \ldots$, for denoting key symbols. As explained in remark 1, we can safely assume that every two public (private) key symbols receive different key values from the key assignment function τ. Also, let $f : \mathbb{N} \rightarrow \mathbb{N}$ be the length function of Enc_K, and φ be a function which is to assign a public (private) key symbol to every public (private) key value. Also, assume that Pub and $Priv$ are two lists of public/private key values which are initially empty. The adversary $\mathcal{A}$ first computes $fix(F'_E)$, and then calls $INITIALIZATION(fix_c(F'_E))$, defined as follows:

algorithm $INITIALIZATION(R)$
  for every $\langle K^{-1}, K, \text{“privkey”}\rangle \in R$ **do**
    add K to Pub and K^{-1} to $Priv$,
    create two new key symbols k and k^{-1},
    assign: $\varphi(K) = k$ and $\varphi(K^{-1}) = k^{-1}$
  $PAT_REC(E)$

algorithm $PAT_REC(E)$
  if $E = \langle E_1, \text{“Block”}\rangle$
    return E_1
  if $E = \langle K^{-1}, K, \text{‘privkey”}\rangle$
    return $\varphi(K^{-1})$
  if $E = \langle K, \text{“pubkey”}\rangle$
    if $K \notin Pub$
      add K to Pub, and create a new key symbol k,
      and assign $\varphi(K) = k$,
    return $\varphi(K)$
  if $E = \langle E_1, E_2, \text{“pair”}\rangle$
    return $(PAT_REC(E_1), PAT_REC(E_2))$
  if $E = \langle C, K, \ ''ciphertext''\rangle$
    if there exists a private key value K^{-1} for which
      $\langle K^{-1}, K, \text{“privkey”}\rangle \in fix_c(F'_E)$
      return $\{PAT_REC(Dec_{K^{-1}}(E))\}_{\varphi(K)}$

otherwise
if $K \notin Pub$
add K to Pub, and create a new key symbol k, and assign $\varphi(K) = k$,
return $\{STRUCT_REC(f^{-1}(|E|))\}_{\varphi(K)}$

Now if we define the key bijection function $\sigma(k) = \varphi(\tau(k))$, it can be easily seen by an structural induction on e that $pr[pat(e) \neq PAT_REC(E)\sigma] = negl(\eta)$. We omit the details of the proof. ■

Now we can prove our two main theorems we presented earlier.

(Proof of theorem 5) The proof of this theorem follows immediately from the pattern recovery lemma. Suppose $E_1 \leftarrow [e_1]^{\eta}_{\Pi}$ and $E_2 \leftarrow [e_2]^{\eta}_{\Pi}$. Note that since $e_1 \not\cong e_2$, it follows that $pat(e_1) \not\cong pat(e_2)$, and consequently, $pr[PAT_REC(E_1) \cong pat(e_2)] = negl(\eta)$. On the other hand, we have : $pr[PAT_REC(E_2) \cong pat(e_2)] = 1 - negl(\eta)$. This observation immediately leads to a distinguisher $\mathcal{A}$ which on input E_i computes $pat(e_i)$ by calling $PAT_REC(E_i)$, and compares it against $pat(e_2)$. If they match up to key renaming, it returns one, otherwise it returns zero. Obviously $\mathcal{A}$ works in polynomial time and is able to distinguish between the two probability distributions with non-negligible probability. ■

(Proof of claim 1) (Sketch) We just need to prove that

$$pat(e_1, T) \not\cong pat(e_2, T) \Rightarrow [e_1]_{\Pi} \not\approx_w^{T^{-1}} [e_2].$$

Suppose $pat(e_1, T) \not\cong pat(e_2, T)$. According to theorem 5, we have $[pat(e_1, T)]_{\Pi} \not\approx_w [pat(e_2, T)]_{\Pi}$. Thus it remains to prove that

$$[pat(e_1, T)]_{\Pi} \not\approx_w [pat(e_2, T)]_{\Pi} \Rightarrow [e_1]_{\Pi} \not\approx_w^{T^{-1}} [e_2].$$

This is easy to see based on what we have proved before. Suppose there exists an adversary $\mathcal{A}$ which can distinguish $[pat(e_1, T)]_{\Pi}$ from $[pat(e_2, T)]_{\Pi}$. From $\mathcal{A}$, we can now construct another adversary $\mathcal{B}^{O^{T^{-1}}}$ which can distinguish $[e_1]_{\Pi}$ from $[e_2]_{\Pi}$. The idea of construction is simple and we just outline the main idea and leave out the details. Suppose $\mathcal{B}^{O^{T^{-1}}}$ is given E_i which is a sample from $[e_i]_{\Pi}$. Using its oracle access, it can turn E_i into a sample D_i from $[pat(e_i, T)]_{\Pi}$, and give D_i to $\mathcal{A}$, and output whatever $\mathcal{A}$ outputs. In more detail, consider that $[pat(e_i, T)]_{\Pi} = [p(e_i, fix(F_{e_i T}))]_{\Pi}$. Now letting $T_1 = fix(F_{e_i T})$, note that $\mathcal{B}$, using its oracle access, can obtain the key values of all key symbols $T_1 - T$. For any $k \in T$, although it might now have its key value, it can ask its oracle to decrypt with respect to it. Now $\mathcal{B}$ can construct D_i as follow: for every $\langle c, pk, \text{“ciphertext”}\rangle$ which is derivable from E_i and is no more decryptable, replace c with $Enc_{pk}(0^d)$, where d is the length of the underlying plaintext of c.

Remark 2: As mentioned earlier, we have defined the computational image of private key symbols as $[k^{-1}]^{\eta}_{\Pi} = \langle \tau(k^{-1}), \tau(k), \text{“privkey”}\rangle$. This is not a restrictive assumption, and one can achieve the same results , possibly with longer proofs, if the computational image of private key symbols is defined as before(i.e. $[k^{-1}]^{\eta}_{\Pi} = \langle \tau(k^{-1}), \text{“privkey”}\rangle$). In this case, the only thing we need to change is the definition of computational key recovery function for the case that $E = \langle c, pk, \text{“cipher”}\rangle$. Namely we define:

- If there exists exactly one sk for which $\langle sk, \text{“privkey”}\rangle \in T$ and sk is a private key of pk, $F'_E(T) = F'_{Dec_{sk}(c)}(T)$.
- Otherwise, $F'_E(T) = \emptyset$

Now, it is easy to see that using this new modification of the computational key-recovery function, all the results presented above can be obtained. We leave out the details.

VI. Conclusion

We have investigated the computational soundness and completeness of formal expressions in public-key cryptography by taking into account the co-inductive definition of symbolic security. It is a known result from [1] that CCA-2 security gives soundness in the absence of key-cycles, when the notion of computational equivalence is strong public-key indistinguishability (definition11). We demonstrated that the same result does not hold in the presence of key-cycles, and we showed how we can weaken the characterization of strong public-key indistinguishability to achieve soundness in the presence of key-cycles. We also proved that in the absence of key-cycles, any length-regular encryption scheme provides completeness; namely, non-equivalent formal expressions are evaluated to computationally-distinguishable probability distributions with respect to to any length-regular encryption scheme. In the presence of key-cycles, we demonstrated via a counter-example that the completeness result does not hold even with respect to CCA-2 security. As a quick comparison with the past work on this area, [2] discusses computational soundness of key-cyclic expressions for the *inductive* definition of symbolic security. The main result of [2] demonstrates that standard notions of computational security cannot provide soundness in the presence of key-cycles. The counter-example presented in [2] to prove the validity of their claim is not applicable to the framework which takes the co-inductive definition of symbolic security. Also in [6] it is proved that, in private-key cryptography, completeness does not hold unless the underlying computational encryption provides authenticated encryption.

References

[1] J. Herzog, “A computational interpretation of dolev-yao adversaries,” *Theor. Comput. Sci.*, vol. 340, pp. 57–81, June 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1195980.1195983

[2] P. Ado, G. Bana, J. Herzog, and A. Scedrov, "Soundness of formal encryption in the presence of key-cycles," in *In Proc. 10th European Symposium on Research in Computer Security (ESORICS05), volume 3679 of LNCS*. Springer, 2005, pp. 374–396.

[3] D. Micciancio, "Computational soundness, co-induction, and encryption cycles," Cryptology ePrint Archive, Report 2009/227, 2009, http://eprint.iacr.org/.

[4] P. Adão, G. Bana, J. Herzog, and A. Scedrov, "Soundness and completeness of formal encryption: The cases of key cycles and partial information leakage," *J. Comput. Secur.*, vol. 17, pp. 737–797, October 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1662658.1662665

[5] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.

[6] D. Micciancio and B. Warinschi, "Completeness theorems for the abadi-rogaway language of encrypted expressions," *J. Comput. Secur.*, vol. 12, pp. 99–129, January 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=1297689.1297692

[7] M. Abadi and P. Rogaway, "Reconciling two views of cryptography (the computational soundness of formal encryption)," in *Proceedings of the International Conference IFIP on Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics*, ser. TCS '00. London, UK: Springer-Verlag, 2000, pp. 3–22. [Online]. Available: http://portal.acm.org/citation.cfm?id=647318.723498

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

[9] X. Lei, R. Xue, and T. Yu, "Computational soundness about formal encryption in the presence of secret shares and key cycles," Cryptology ePrint Archive, Report 2010/467, 2010, http://eprint.iacr.org/.

[10] J. Black, P. Rogaway, and T. Shrimpton, "Encryption-scheme security in the presence of key-dependent messages," in *Revised Papers from the 9th Annual International Workshop on Selected Areas in Cryptography*, ser. SAC '02. London, UK: Springer-Verlag, 2003, pp. 62–75. [Online]. Available: http://portal.acm.org/citation.cfm?id=646558.694914

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

VII. Appendix

A. Correspondence between formal length and structure

In the completeness section, we showed that it is possible to establish a one-to-one correspondence between the computational length and the structure of a message. Here we provide more comprehensive discussion on this subject and we show that it is possible to actually introduce a *formal length function* which has the property that it gives equal length to two formal expressions iff they have the same structure. Such a construction of formal length function suggests an equivalent way of characterizing formal indistinguishability, namely based on formal length.

Definition 14: (Formal length) Let $R : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ be a one-to-one function such that $R(x, y) > 1$ for all $x, y \in \mathbb{N}$. We define the formal length function $FL : Exp \rightarrow \mathbb{N}$ as follow:

- if k is a public key symbol, $FL(k) = R(1, 0)$
- if k^{-1} is a private key symbol, $FL(k^{-1}) = R(2, 0)$
- if $e \in Block$, $FL(e) = R(3, 0)$
- $FL(\{e\}_k) = R(FL(e), 1)$
- $FL((e_1, e_2)) = R(FL(e_1), FL(e_2))$

Proposition 3: Suppose e_1 and e_2 are two formal expressions. It holds that

$$struct(e_1) = struct(e_2) \Leftrightarrow FL(e_1) = FL(e_2).$$

Proof: The proof of only-if direction ($\Rightarrow$)can be easily established by an structural induction on e_1 (or equivalently e_2). For the other direction, we need to prove that:

$$FL(e_1) = FL(e_2) \Rightarrow struct(e_1) = struct(e_2).$$

First note that according to our formal length function, if e_1 and e_2 are not of the same type (for example, if one of them is a block symbol and the other one is a pair symbol), then $FL(e_1) \neq FL(e_2)$. As a result, $FL(e_1) = FL(e_2)$ implies that e_1 and e_2 are of the same type. We now prove that if $FL(e_1) = FL(e_2)$ then $struct(e_1) = struct(e_2)$, by an structural induction on e_1:

- Base case: $e_1 \in Block$ or e_1 is a key symbol. According to what we just mentioned, it implies that e_2 is also of the same type and consequently, $struct(e_1) = struct(e_2)$
- $e_1 = (d_1, d_2)$ is a pair term. Since e_1 and e_2 are of the same type, it implies that e_2 is also a pair term, say $e_2 = (d_3, d_4)$. Since $FL(e_1) = FL(e_2)$, we have: $FL(d_1) = FL(d_3)$ and $FL(d_2) = FL(d_4)$, and according to induction hypothesis, $struct(d_1) = struct(d_3)$ and $struct(d_2) = struct(d_4)$. Therefore we have: $struct(e_1) = struct(e_2)$
- $e_1 = \{d_1\}_{k_1}$ is an encryption term. Therefore, e_2 is also an encryption term, say $e_2 = \{d_2\}_{k_2}$. Note that $FL(e_1) = FL(e_2)$ implies that $FL(d_1) = FL(d_2)$ which in turn implies that $struct(d_1) = struct(d_2)$, according to the induction hypothesis. Therefore, $struct(e_1) = struct(e_2)$, and this completes the proof.

■

Corollary 3: Let FL be the formal length function presented in definition 14 and e_1 and e_2 be two formal expressions. Then the following statements are equivalent:

1) $FL(e_1) = FL(e_2)$

2) $struct(e_1) = struct(e_2)$
3) $|[e_1]^{\eta}_{\Pi}| = |[e_2]^{\eta}_{\Pi}|$

B. Proofs

Proof of lemma 1 We prove that $fix(F_e) = F_e^{n_1}(\emptyset)$ is the least fixed point of F_e. For this purpose, we need to first prove that $F_e^{n_1}(\emptyset)$ is actually a fixed point. This is easy to see. Consider the following sequence which contains n_1+1 sets of keys :

$$F_e^0(\emptyset) = \emptyset \subseteq F_e^1(\emptyset) \subseteq \cdots \subseteq F_e^{n_1}(\emptyset)$$

Note that for all i, $F_e^i(\emptyset) \subseteq privkeys(e)$. Now given the fact that $privkeys(e)$ has n_1 different subsets, there exists some i such that $0 \leq i \leq n_1$ and $F_e^i(\emptyset) = F_e^{i+1}(\emptyset)$. This implies that $F(F_e^{n_1}(\emptyset)) = F_e^{n_1}(\emptyset)$, which shows that $F_e^{n_1}(\emptyset)$ is a fixed point.

Now we prove that if T is a fixed point, then $F_e^{n_1}(\emptyset) \subseteq T$, which implies that $F_e^{n_1}(\emptyset)$ is the least fixed point. Note that since T is a fixed point, we have: $T = F_e^{n_1}(T)$. So we need to prove that $F_e^{n_1}(\emptyset) \subseteq F_e^{n_1}(T)$. This is easy to see because F_e is a monotone function (Proposition 1). In a similar manner, we can prove that $F_e^{n_1}(privkeys(e))$ is the greatest fixed point of F_e.

Therefore, $fix(F_e) = F_e^{n_1}(\emptyset)$ and $FIX(F_e) = F_e^{n_1}(privkeys(e))$. Since F_e is monotone and $\emptyset \subseteq F_e(\emptyset)$, we have $F_e^i(\emptyset) \subseteq F_e^{i+1}(\emptyset)$, and thus $fix(F_e) = F_e^{n_1}(\emptyset) = \bigcup_i F_e^i(\emptyset)$. Similarly, since $F_e(privkeys(e)) \subseteq privkeys(e)$, it holds that $F_e^{i+1}(privkeys(e)) \subseteq F_e^i(privkeys(e))$, and consequently, $FIX(F_e) = F_e^{n_1}(privkeys(e)) = \bigcap_i F_e^i(privkeys(e))$.

Proof of lemma 2 Assuming that $fix(F_e) \neq FIX(F_e)$, we prove that the graph G_e is cyclic. Let $T_1 = fix(F_e)$, $T_2 = FIX(F_e)$, and $T = T_2 - T_1$. Note that for every $k^{-1} \in T$, there exists another $k_1^{-1} \in T$ such that k_1 encrypts k^{-1}, because otherwise k^{-1} were in T_1 as well. Therefore, if we let G_1 be the induced subgraph of G_e on the vertices corresponding to T, we can see that every vertex in G_e has an in-degree of at least one. It is a well-known fact in graph theory that if every vertex of a directed graph G has an in-degree of at least one, then G is cyclic. Therefore G_e is cyclic and the proof is complete.

Proof of lemma 3 We prove this by structural induction on e. For the base case, if e is a block symbol or a key symbol(public or private), it obviously holds that: $pr[T'_c \neq F'_E(T_c)] = 0$. Now we have:

- if $e = (e_1, e_2)$ and $E = (E_1, E_2)$, we have: $pr[\ F'_E(T_c) \neq [F_e(T)]\ ] = pr[\ F'_{E_1}(T_c) \neq [F_{e_1}(T)]$ *or* $F'_{E_2}(T_c) \neq [F_{e_2}(T)]\ ] \leq pr[\ F'_{E_1}(T_c) \neq [F_{e_1}(T)\ ] + pr[\ F'_{E_2}(T_c) \neq [F_{e_2}(T)]\ ] \leq negl(\eta) + negl(\eta) = negl(\eta)$.
- if $e = \{e_1\}_k$ and $E = \langle c, pk, \text{“ciphertext”}\rangle$, then there are two cases(assume that E_1 is the computational image of e_1 in E):
  1) $k^{-1} \notin T$. Therefore, we have: $F_e(T) = \emptyset$, and consequently:

     $pr[\ F'_E(T_c) \neq [F_e(T)]\ ] = pr[\ F'_E(T_c) \neq \emptyset] = pr[\ \exists k_1^{-1} \in T \textit{ such that } \tau(k_1) = \tau(k)\ ] \leq negl(\eta)$.
  2) $k^{-1} \in T$. It holds that $F_e(T) = F_{e_1}(T)$, and consequently :

     $pr[\ F'_E(T_c) \neq [F_e(T)]\ ] = pr[\ F'_E(T_c) \neq [F_{e_1}(T)]\ ]$
     $\leq pr[\ \exists k_1^{-1} \in T \textit{ such that } k_1^{-1} \neq k^{-1}\ \&\ \tau(k_1) = \tau(k)\ ] +$
     $pr[\ F'_E(T_c) \neq [F_{e_1}(T)] \mid \nexists k_1^{-1} \in T \textit{ such that } k_1^{-1} \neq k^{-1}\ \&\ \tau(k_1) = \tau(k)\ ]$
     $\leq negl(\eta) + pr[\ F'_{E_1}(T_c) \neq [F_{e_1}(T)]\ ] \leq negl(\eta) + negl(\eta) = negl(\eta)$.

■