Resettable Public-Key Encryption: How to Encrypt on a Virtual Machine

Scott Yilek

Dept. of Computer Science and Engineering
University of California, San Diego
`syilek@cs.ucsd.edu`

Abstract. Typical security models used for proving security of deployed cryptographic primitives do not allow adversaries to rewind or reset honest parties to an earlier state. Thus, it is common to see cryptographic protocols rely on the assumption that *fresh* random numbers can be continually generated. In this paper, we argue that because of the growing popularity of virtual machines and, specifically, their state snapshot and revert features, the security of cryptographic protocols proven under these assumptions is called into question. We focus on public-key encryption security in a setting where resetting is possible and random numbers might be reused. We show that existing schemes and security models are insufficient in this setting. We then provide new formal security models and show that making a simple and efficient modification to any existing PKE scheme gives us security under our new models.

1 Introduction

In the past few decades, cryptographers have modeled numerous cryptographic primitives and protocols in order to argue about their security. Because of this, we have strong tools to securely execute just about any desirable task. These tools include symmetric and public-key encryption, message authentication codes, digital signatures, secure key exchange protocols, and more. Moreover, the security guarantees are provable by reductions from problems conjectured to be difficult.

A typical security model has an adversary playing a game against an environment which may contain multiple honest parties. As the adversary interacts with the environment and time progresses, the states of the honest parties continually change to reflect events that take place. For example, if an adversary is executing an interactive protocol with an honest party (call her Alice) and the adversary sends a message to Alice, then Alice's next message will be a function of her current state, which will itself be a function of the past messages she has received from the adversary. This essentially means that an adversary cannot 'take back' messages and try others, effectively rewinding the protocol and resetting Alice's state.

Modeling security of protocols in this way is natural because it fits our understanding of how the world works. Time travel, as far as we know, is impossible, and the complex states of our computers are constantly changing every time we click a mouse or receive a packet from the network. Thus, it seems perfectly reasonable that protocols proven secure in this model will continue to be secure in the forseeable future. However, as we argue in this paper, this may not be the case because of the increasing popularity of *virtual machines*.

Virtual Machines. In the past few years, it has become common for systems to run on virtual machines. In short, a virtual machine (VM) is software that emulates a real machine. A VM consists of a virtual machine monitor (or hypervisor) which can emulate multiple virtual computers that can have varying instruction sets and run multiple operating systems. The VM monitor will then share the physical machine's resources among the virtual machines, translating machine instructions and acting as a simulator for the underlying operating systems.

It is especially common for servers to run on virtual machines. This will likely become even more typical in the future given the rising popularity of cloud computing services like Amazon's Elastic

Compute Cloud (EC2) [1]. In this service, a user buys some compute time and receives access to a virtual machine on one of Amazon's servers. Within that VM, the user can run a fully functional OS and, in particular, run a web server for his or her business.

Thus far, virtual machines have often been seen as being *beneficial* to security. Because VMs provide a type of sandbox, they have been used to test potentially malicious code [13] and isolate web browsers from the rest of the system to mitigate the effects of browser vulnerabilities [14]. VMs have also been used to more easily create large honeypots [21]. Despite this, the focus of this paper is on how VMs can be *detrimental* to security. The reason, which also happens to be one of the most useful features of VMs, is the ability to take state snapshots.

State Snapshots. Virtual machines allow a user to take a snapshot of the current system state. This snapshot contains the contents of all the virtual machine's disks and the contents in memory at the time of the snapshot. At a later point in time, the VM can be reverted back to this previous state and restarted. To see why this may be useful, consider the following scenario. Alice, a system administrator, is running an important web server on a virtual machine, and at some point in time there is a crash or some other major problem. Instead of spending time diagnosing the problem and getting the system working again, Alice can instead revert the VM back to a 'good state' for which she has a snapshot. In other words, Alice takes a snapshot of the system when things are running smoothly, and then reverts back to this state whenever things go wrong. In this scenario, the server has effectively traveled back in time; program variables and other state that may have been in memory are now active again. Thus, if an adversary is attacking Alice's server and can make it crash (using, for example, a DoS attack), he essentially has the ability to rewind the server.

This brings up some important questions. What happens to our supposedly secure cryptographic tools in a setting with resets? Are they still secure? Researchers have examined these questions before for zero-knowledge [12, 20, 3] and identification protocols [8], where the motivation was smart cards that cannot keep internal state. However, the growing popularity of virtual machines means we need to ask these questions for a wider range of cryptographic primitives.

To see why reset attacks can have negative effects on cryptographic protocols, consider a common assumption in cryptography: it is possible to contiually generate fresh and unbiased random numbers. This is an assumption made in nearly every cryptographic protocol. It is, however, considered reasonable since pseudorandom number generators (PRNGs) are well-studied both in theory and practice (c.f., [19, 16]). In deployed systems, PRNGs are often implemented in software and consist of numerous state variables and arrays that are occasionally seeded with entropy and used to generate pseudorandom numbers. For example, in OpenSSL [2], the software PRNG has a 1023-byte array (entropy pool) that is supposed to contain high entropy data from a variety of sources, as well as some variables with counters and other important state. At a high level, when random bytes are requested, data from the entropy pool and information in the state variables[1] is continually mixed together using a cryptographic hash function and the result is the output of the PRNG. However, these arrays and variables will be captured by a state snapshot since they reside in memory. If the machine is later reset, the PRNG could output a string of "random" bytes that it already outputted sometime in the past before the machine was reset. These un-fresh coins might then be used in a cryptographic operation with potentially disastrous consequences.

Our Results. In this paper, we focus on one particular primitive, public-key encryption, and make the following contributions. First, we provide formal security definitions to model public-key encryption security in the face of resetting attacks. Second, we show that existing PKE schemes and their common security notions IND-CPA [17] and IND-CCA [22] are insufficient when such resetting attacks are

[1] This might also include other information like the process ID of the process requesting the random bytes.

possible. Third, we show that, perhaps somewhat surprisingly, a small and efficient modification can be made to *any* existing PKE scheme secure under the typical notions (e.g., IND-CCA) in order to ensure security against resetting attacks. Our modification does not rely on random oracles [10], requires no extra assumptions, and is very efficient.

A Closer Look. The generally accepted "right" notion of security for public-key encryption is indistinguishability under chosen-ciphertext attack (IND-CCA). Though this is a strong notion of security, it fails to suffice in a setting where randomness may be reused. At a high level, the reason is that for many schemes, given a ciphertext and the corresponding plaintext it is often possible to learn some of the coins (or some useful function of the coins) used to encrypt the message. If another ciphertext is generated using those same coins, it may be possible for an adversary to learn parts of the underlying plaintext. More specifically, consider an encryption scheme that applies a trapdoor one-way function to a random value r and then concatenates $H(r) \oplus m$ and $G(r \parallel m)$. This scheme is known to be IND-CCA secure if H and G are modeled as random oracles [10]. Now, if another message m' is encrypted using the same coins r, the message will be xor'd *with the same pad* $H(r)$, and anyone who knows m will also know the pad and be able to learn m'.

Since IND-CCA is insufficient for our setting, we develop a new notion of security for PKE which we call IND-RA for indistinguishability under resetting attack. Our security notion is similar to IND-CCA except that we allow the adversary to continually see encryptions under the same coins, as if the adversary is continually resetting a server and observing new encryptions. An important aspect of our security definition is that we allow the adversary to see encryptions *under public keys chosen by the adversary* and using coins that are not fresh. In particular, the adversary could see a message encrypted under a public key for which it knows the secret key, allowing it to decrypt the ciphertext; because of this, it is important that in the process of decryption not too much information is leaked about the coins used to create the ciphertext, meaning that randomness-recovering encryption cannot meet our security definition. Allowing this power in the definition is important because it models the possibility that a machine sends an encrypted message to some user Bob, is reset by the adversary, and is then forced to encrypt a message to the adversary using the same coins. We want to ensure that even if this happens, the adversary does not learn any information about Bob's message. This is a strong security requirement, but nonetheless, we are able to meet it.

We note that though our security notion provides seemingly the best possible security guarantees for PKE under reset attacks, it may still be insufficient for some applications. This is due to an inherent limitation in a model that allows repeated randomness: if the same message is encrypted twice to the same public key using the same randomness, the resulting ciphertexts will be identical. Thus, plaintext equality may be leaked to an adversary, which could be problematic in some applications. Therefore, care should be taken when deploying applications using PKE on virtual machines.

Previous Work. Resettability has been considered in cryptography in the setting of zero-knowledge proof systems [12, 20, 3], the related area of identification protocols [8], and multiparty computation [18]. Zero-knowledge proofs allow a prover to prove an assertion to a verifier without revealing any information other than whether or not the assertion is true. Proving the soundness[2] and zero-knowledge properties in a setting where provers and verifiers can rewind each other is a difficult and interesting theoretical question. To see why, consider the notion of resettable-soundness in the standard model, considered by [3]. Nearly all known zero-knowledge proofs are designed specifically so that the ability to rewind the verifier allows one to easily convince it of any statement; this is useful for proving the zero-knowledge property. Yet, if we then give the prover that same ability to rewind the

[2] Informally, an interactive protocol is sound if it is difficult for a malicious prover to convince the verifier that a false statement is true.

verifier, it becomes problematic to prove soundness. This problem has also been studied extensively in other models (c.f., [20]). However, to the best of our knowledge, no one has previously looked at practical and deployed cryptographic primitives like public-key encryption in such a setting, nor has anyone realized the importance of the question given the rising popularity of virtual machines.

In the symmetric setting, Rogaway and Shrimpton [23] investigate secure key-wrap and discuss how their techniques can apply to handle IV misuse, where IVs, which should always be fresh, are reused (possibly because of a faulty implementation). Since IVs are typically counter variables or fresh random numbers, investigating their reuse is similar to investigating the effect of a state reset.

Our work is also loosely related to public-key encryption with randomness re-use [4] and stateful public-key encryption [9]. However, both are concerned with making PKE schemes more efficient by reusing *some, but not all* random coins and still require encrypting parties to have access to fresh and unbiased randomness.

Bellare et al. recently introduced *hedged public-key encryption* [6]. At a high level, they present encryption schemes that are IND-CPA secure when the randomness used to encrypt is good, while meeting a weaker notion they call IND-CDA when the randomness is bad but the message/randomness pairs still have high entropy. Interestingly, while their goal is similar to ours and reused randomness could be considered "bad" randomness, their definitions do not capture the resettability setting.

Paper Organization. In Section 2, we discuss important definitions and notation that will be needed in the rest of the paper. In Section 3, we define our new notion of security for public-key encryption that models resetting attacks. In Section 4, we give constructions for schemes that meet our new notion of security.

2 Preliminaries

Notation. For an integer $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, \ldots, n\}$. Throughout the entire paper, $k \in \mathbb{N}$ denotes the security parameter and 1^k its unary encoding. Unless stated otherwise, all algorithms in this paper are randomized. We use "PT" for polynomial-time.

Our security definitions use the code-based games from [11]. Security definitions are formulated by considering a game played with an adversary. Such a game consists of procedures **Initialize** and **Finalize** as well as procedures for handling oracle calls the adversary can make. At the start of the game, **Initialize** is run and its output is given to the adversary. The adversary then runs and may make oracle calls that are answered by the corresponding game procedures. When the adversary halts with output w, that becomes the input to the **Finalize** procedure and the resulting output of **Finalize** is called the output of the game. We denote by $G^A \Rightarrow w$ the event that game G, when run with adversary A, outputs w. Sometimes we let G^A denote the event $G^A \Rightarrow \mathsf{true}$.

Public-key encryption. A public-key encryption (PKE) scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is a triple of PT algorithms. The randomized key generation algorithm $\mathcal{K}$, on input the security parameter 1^k, outputs a pair of keys (pk, sk). The randomized encryption algorithm $\mathcal{E}$, on input public key pk and message $m \in \{0,1\}^{\eta(k)}$, outputs a ciphertext c. We let $\rho(k)$ denote the number of coins $\mathcal{E}$ uses on messages of length $\eta(k)$. Finally, the deterministic decryption algorithm $\mathcal{D}$, on input a secret key sk and ciphertext c, outputs either $\perp$ in the case of failure, or $m \in \{0,1\}^{\eta(k)}$. We require that for all $k \in \mathbb{N}$, all (pk, sk) outputted by $\mathcal{K}(1^k)$ and for all $\{0,1\}^{\eta(k)}$, it is true that $\mathcal{D}(sk, \mathcal{E}(pk, m)) = m$.

We say the IND-advantage of an adversary A is

$$\mathbf{Adv}^{\mathrm{ind}}_{\mathcal{AE},A}(k) = 2 \cdot \Pr\left[\, \mathrm{IND}^{A}_{\mathcal{AE}}(k) \Rightarrow \mathsf{true} \,\right] - 1 \;,$$

where the security game is found in Figure 1. To differentiate between chosen-plaintext and chosen-ciphertext attacks, we consider adversary classes. Let $\mathcal{A}^{\mathrm{CPA}}_{\mathrm{ind}}$ be the class of all PT ind-adversaries making 1 **LR** query and 0 **Dec** queries.[3] Let $\mathcal{A}_{\text{ind}}^{\text{CCA}}$ be the class of all PT ind-adversaries making 1 **LR** query and any number of **Dec** queries.

We let IND-XXX be the set of all PKE schemes $\mathcal{AE}$ such that $\mathbf{Adv}_{\mathcal{AE},A}^{\text{ind}}(k)$ is a negligible function in k for all $A \in \mathcal{A}_{\text{ind}}^{\text{XXX}}$, for XXX $\in \{\text{CPA}, \text{CCA}\}$.

```
Game IND_{AE}(k)

proc. Initialize(k):
    $b \leftarrow_\$ \{0,1\}$ ; $(pk, sk) \leftarrow_\$ \mathcal{K}(1^k)$
    $S \leftarrow \emptyset$
    Ret $pk$

proc. Dec(c):
    If $c \in S$ then return $\perp$
    Else return $\mathcal{D}(sk, c)$

proc. LR(m_0, m_1):
    $c \leftarrow_\$ \mathcal{E}(pk, m_b)$
    $S \leftarrow S \cup \{c\}$
    Return $c$

proc. Finalize(b'):
    Ret $(b = b')$
```

Fig. 1. Security game for IND security.

Pseudorandom Functions. Let $\mathsf{Fun} : \mathsf{Keys}_k \times \mathsf{Dom}_k \to \mathsf{Rng}_k$ be a family of functions indexed by a security parameter k. We say the PRF-advantage of a prf-adversary D is

$$\mathbf{Adv}_{\mathsf{Fun},D}^{\text{prf}}(k) = \Pr\left[\,\text{REAL}_{\mathsf{Fun}}^{D}(k) \Rightarrow 1\,\right] - \Pr\left[\,\text{RAND}_{\mathsf{Fun}}^{D}(k) \Rightarrow 1\,\right] \;,$$

where the security games can be found in Figure 2. While Keys_k, Dom_k, and Rng_k can be arbitrary finite sets, in this paper we will always consider families of functions with $\mathsf{Keys}_k = \{0,1\}^{\ell(k)}$, $\mathsf{Dom}_k = \{0,1\}^{n(k)}$, and $\mathsf{Rng}_k = \{0,1\}^{t(k)}$ for some polynomials $\ell(\cdot)$, $n(\cdot)$, and $t(\cdot)$.

```
Game REAL_{F}(k)

proc. Initialize(k):
    $K \leftarrow_\$ \mathsf{Keys}_k$
    Ret $1^k$

proc. Fun(x):
    Return $\mathsf{Fun}(K, x)$

proc. Finalize(a):
    Ret $a$
```

```
Game RAND_{F}(k)

proc. Initialize(k):
    $\texttt{FunTab} \leftarrow \emptyset$
    Ret $1^k$

proc. Fun(x):
    If $\texttt{FunTab}[x] = \perp$ then
        $\texttt{FunTab}[x] \leftarrow_\$ \mathsf{Rng}_k$
    Return $\texttt{FunTab}[x]$

proc. Finalize(a):
    Ret $a$
```

Fig. 2. Security games for pseudorandom function security.

3 Security Definition

Let $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a PKE scheme. We say the RA-advantage of an adversary A is

$$\mathbf{Adv}_{\mathcal{AE},A}^{\text{ra}}(k) = 2 \cdot \Pr\left[\,\text{RA}_{\mathcal{AE}}^{A}(k) \Rightarrow \mathsf{true}\,\right] - 1 \;.$$

The security game RA can be found in Figure 3. In the game, the adversary is given a target public key pk^* and can make queries to three oracles. It can query the **LR** oracle with index j and messages

[3] It is well known that allowing multiple **LR** queries is equivalent by a standard hybrid argument.

m_0 and m_1. In response, the adversary receives the encryption of m_b under the target public key pk^* using the coins indexed by j. The adversary is also given an **Enc** oracle which takes as input a public key pk, index j, and message m. The oracle returns the encryption of m under public key pk using the coins indexed by j. It is important that the adversary can choose the public key pk. In particular, the adversary can query **Enc** with a public key for which it knows the corresponding secret key. With both the **LR** and **Enc** oracles, an adversary can continually see messages encrypted under the same coins by repeatedly querying the same index. This is how we model resetting attacks. Of course, the adversary can also see messages encrypted under other coins by querying other indices. Finally, the adversary can also query a **Dec** oracle with a ciphertext and receive its decryption.

```
Game $\mathrm{RA}_{\mathcal{AE}}(k)$

proc. Initialize($k$):
    $b \leftarrow_{\$} \{0,1\}$ ; $(pk^*, sk^*) \leftarrow_{\$} \mathcal{K}(1^k)$
    $\texttt{CoinTab} \leftarrow \emptyset$ ; $S \leftarrow \emptyset$
    Ret $pk^*$

proc. Enc($pk, j, m$):
    If $\texttt{CoinTab}[j] = \perp$ then
        $\texttt{CoinTab}[j] \leftarrow_{\$} \{0,1\}^{\rho(k)}$
    $r_j \leftarrow \texttt{CoinTab}[j]$
    $c \leftarrow \mathcal{E}(pk, m; r_j)$
    Return $c$

proc. LR($j, m_0, m_1$):
    If $\texttt{CoinTab}[j] = \perp$ then
        $\texttt{CoinTab}[j] \leftarrow_{\$} \{0,1\}^{\rho(k)}$
    $r_j \leftarrow \texttt{CoinTab}[j]$
    $c \leftarrow \mathcal{E}(pk^*, m_b; r_j)$
    $S \leftarrow S \cup \{c\}$
    Return $c$

proc. Dec($c$):
    If $c \in S$ then return $\perp$
    Else return $\mathcal{D}(sk^*, c)$

proc. Finalize($b'$):
    Ret $(b = b')$
```

Fig. 3. Security game for defining security against resetting attacks.

Equality Patterns. As we mentioned above, if there are no restrictions on the **LR** queries that an ra-adversary A can make, then A can trivially win the game. To see this, consider an ra-adversary that first queries $\mathbf{Enc}(1, m)$ and then queries $\mathbf{LR}(1, m, m')$, where $m, m' \in \{0,1\}^\eta$ and $m \neq m'$. The **Enc** query will give the adversary the encryption of m under coins r_1, and the **LR** query will give the adversary either the encryption of the same message m under the same coins r_1, or it will give the adversary the encryption of m' under coins r_1. Clearly the adversary only needs to compare the two oracle answers and guess 0 if they are the same and guess 1 otherwise.

This attack is an inherent limitation of the resettable PKE setting, since for fixed coins encryption becomes a deterministic function. (It is also similar to limitions in the setting of deterministic PKE [5].) Nevertheless, as we said earlier, we are interested in achieving the best security possible in this situation. Therefore, we consider security against all adversaries "that don't trivially win". This informal notion is captured formally by the following definition:

Let A be any adversary that queries I different indices to its **LR** and **Enc** oracles and makes q_i queries to the **LR** oracle with index i. Let E_i be the set of all messages m such that A makes query $\mathbf{Enc}(pk^*, i, m)$. Let $(m_0^{i,1}, m_1^{i,1}), \ldots, (m_0^{i,q_i}, m_1^{i,q_i})$ be A's **LR** queries for index $i \in [I]$. Then, if for all $i \in [I]$ and for all $j \neq k \in [q_i]$,

$$m_0^{i,j} = m_0^{i,k} \text{ iff } m_1^{i,j} = m_1^{i,k} \ ,$$

and for all $i \in [I]$ and all $j \in [q_i]$

$$m_0^{i,j} \notin E_i \wedge m_1^{i,j} \notin E_i,$$

then we say that A is equality-pattern respecting.

Adversary Classes. To differentiate between various kinds of attacks, we use classes of adversaries. Let $\mathcal{A}_{\mathrm{ra}}^{(q,i)\text{-XXX}}$ be the class of all PT equality-pattern respecting adversaries that make q total **LR** queries, query **LR** and **Enc** with at most i different indices, and make 0 **Dec** queries if XXX = CPA and 0 or more **Dec** queries if XXX = CCA. Using hybrid arguments, we can prove the following two claims:

Claim. For every $i \in \mathbb{N}$ and every ra-adversary $A_{q,i} \in \mathcal{A}_{\mathrm{ra}}^{(q,i)\text{-XXX}}$ there exists an ra-adversary $A_{q,1} \in \mathcal{A}_{\mathrm{ra}}^{(q,1)\text{-XXX}}$ such that

$$\mathbf{Adv}^{\mathrm{ra}}_{\mathcal{AE},A_{q,i}}(k) \leq i \cdot \mathbf{Adv}^{\mathrm{ra}}_{\mathcal{AE},A_{q,1}}(k) ,$$

where the running time of $A_{q,1}$ is about the same as that of $A_{q,i}$.

Claim. For every $q \in \mathbb{N}$ and every ra-adversary $A_{q,1} \in \mathcal{A}_{\mathrm{ra}}^{(q,1)\text{-XXX}}$ there exists an ra-adversary $A_{1,1} \in \mathcal{A}_{\mathrm{ra}}^{(1,1)\text{-XXX}}$ such that

$$\mathbf{Adv}^{\mathrm{ra}}_{\mathcal{AE},A_{q,1}}(k) \leq q \cdot \mathbf{Adv}^{\mathrm{ra}}_{\mathcal{AE},A_{1,1}}(k) ,$$

where the running time of $A_{1,1}$ is about the same as that of $A_{q,1}$.

Proof of Claim 3 (Sketch). Let $A_{q,i}$ be any equality pattern respecting ra-adversary making q queries to **LR** and querying **LR** and **Enc** on at most i different randomness indices. We will build an adversary $A_{q,1}$ making at most q queries to the **LR** oracle and querying **LR** and **Enc** on at most 1 randomness index. The adversary $A_{q,1}$ runs $A_{q,i}$ and guesses an index $j \in \{1, \ldots, i\}$. It then uniformly chooses coins r_ℓ for $\ell \neq j$ and answers **Enc** and **LR** queries from $A_{q,i}$ as follows. On query $\mathbf{Enc}(pk, k, m)$, if $k = j$ then $A_{q,1}$ replies with its own **Enc** oracle and otherwise uses coins r_k to answer the query. On query $\mathbf{LR}(k, m_0, m_1)$, if $k < j$ (resp. $k > j$) then $A_{q,1}$ replies with the encryption of m_0 (resp. m_1) under coins r_k; if $k = j$ then $A_{q,1}$ replies using its own **LR** oracle. At the end of the simulation, $A_{q,1}$ outputs the same guess bit as $A_{q,i}$. ▮

Proof of Claim 3 (Sketch). Let $A_{q,1}$ be any equality pattern respecting ra-adversary making q queries to **LR** and querying **LR** and **Enc** on only a single randomness index. We can build an adversary $A_{1,1}$ making only a single **LR** query. Adversary $A_{1,1}$ guesses a query $t \in \{1, \ldots, q\}$ and answers the first $t-1$ **LR** queries using its **Enc** oracle applied to m_0, the tth query using its own **LR** oracle, and the rest of the queries again using **Enc**, this time applied to m_1. As above, $A_{1,1}$ outputs the same answer as $A_{q,1}$. ▮

Given the above claims, we let $\mathcal{A}_{\mathrm{ra}}^{\mathrm{XXX}} = \mathcal{A}_{\mathrm{ra}}^{(1,1)\text{-XXX}}$ for XXX $\in \{\mathrm{CPA}, \mathrm{CCA}\}$ and focus on adversaries in these classes for the rest of the paper. Notice that when we consider adversaries in this simpler class, the equality pattern restriction is much easier to state: an adversary A making **LR** and **Enc** queries with only a single index j and making one **LR** query (j, m_0, m_1) is equality-pattern respecting if for all **Enc** queries (pk^*, j, m) made by A, it is the case that $m \notin \{m_0, m_1\}$. In other words, an equality-pattern respecting adversary never requests the encryption of a message m under the target public key pk^* if that same message appears in its **LR** query.

Finally, let RA-XXX be the set of all PKE schemes $\mathcal{AE}$ such that $\mathbf{Adv}^{\mathrm{ra}}_{\mathcal{AE},A}(k)$ is a negligible function in k for all $A \in \mathcal{A}_{\mathrm{ra}}^{\mathrm{XXX}}$, for XXX $\in \{\mathrm{CPA}, \mathrm{CCA}\}$.

Relation to IND. Now that we have formally defined resettable security for public-key encryption, it is useful to compare it to indistinguishability under chosen plaintext and chosen-ciphertext attacks, the typical notions of security for PKE. The relationship is summarized in the following proposition.

Proposition 1. *For* XXX $\in$ {CPA, CCA},

$$\text{RA-XXX} \subsetneq \text{IND-XXX}\ .$$

Proof. We first show RA-XXX $\subseteq$ IND-XXX by contradiction. Suppose there is some encryption scheme $\mathcal{AE}$ that is RA-XXX but is not IND-XXX. Let A be an ind-adversary with noticeable advantage against $\mathcal{AE}$. We construct ra-adversary B as follows: B runs A and single oracle query $\mathbf{LR}(m_0, m_1)$ from A, adversary B queries its own $\mathbf{LR}$ oracle with (j, m_0, m_1) for a fresh index j. In the case that XXX is CCA, decryption queries from A are directly answered with the same decryption query from B. Now, B's queries are clearly equality pattern respecting since there is only one of them, and it easily follows that B has noticeable advantage in the RA game, giving us a contradiction.

Next, we show that IND-XXX $\not\subseteq$ RA-XXX. To do so we give a scheme $\mathcal{AE}$ that is IND-XXX but not RA-XXX. We will prove for XXX=CCA, but the proof easily extends to the CPA setting. Let $\overline{\mathcal{AE}} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ be an arbitrary IND-CCA scheme. We construct a new PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ such that $\mathcal{AE}$ is still in IND-CCA, but $\mathcal{AE}$ is not in RA-XXX. The scheme $\mathcal{AE}$ has encryption algorithm $\mathcal{E}(pk, m; r\|K\|K')$ that outputs $c_1\|c_2\|c_3$, where $c_1 = \overline{\mathcal{E}}(pk, K\|K'; r)$, $c_2 = K \oplus m$ and $c_3 = \mathsf{MAC}_{K'}(c_2)$. The IND-CCA security of $\mathcal{AE}$ follows from the well-known KEM/DEM composition theorem of [15]. We can construct an ra-adversary A with advantage 1 against $\mathcal{AE}$. Adversary A, upon receiving target public key pk^*, queries the $\mathbf{Enc}$ oracle with $(pk^*, 1, 0^{\eta(k)})$ and immediately learns K from the response, since K is xor'd with all 0s. Then, A queries $\mathbf{LR}(1, m_0, m_1)$ for unique messages m_0 and m_1 (which do not equal the string of all zeroes). A can then use K to decrypt the response and win the game. ▮

Discussion. There are a few important aspects of our security definition that require more discussion.

First, as shown in Proposition 1, our definition is stronger than previous notions of security. Since we are concerned about random coins being reused, one might ask why we even need a new definition and do not just use deterministic public-key encryption [5], eliminating the coins altogether. The reason is that we still want our schemes to meet the previous definitions (i.e., IND-CCA) to ensure they have as much security as possible, and it is well-known that no deterministic scheme can ever be IND-CCA (or IND-CPA) secure.

Second, we allow the adversary to give arbitrary public keys to the **Enc** oracle and see the resulting ciphertexts under those keys and the reused coins. As mentioned in the introduction, this is important to model the situation in which a machine is reset and then an encryption is sent to the adversary; we want to make sure other encryptions using the same coins still maintain their privacy. This aspect of our definition resembles a similar ability allowed in the definition of stateful PKE [9].

Third, one might wonder what our equality pattern restriction means in practice. It simply reflects the fact that if a message is encrypted twice using the same public key and the same coins, then the resulting ciphertexts will be the same. An adversary observing the two ciphertexts will know that the underlying plaintexts are the same. This attack is unavoidable in the resettability setting, and whether or not it is a problem likely depends on the application.

4 Achieving IND-RA Security

In this section we show that we can make a simple and efficient modification to any IND-XXX PKE scheme and immediately get an RA-XXX secure scheme. Our transformation relies only on the existence of pseudorandom functions and thus we do not require the random oracle model [10]. This means that if we take a PKE scheme that is IND-XXX secure in the standard model, our modified scheme will be RA-XXX secure in the standard model.

Let $\overline{\mathcal{AE}} = (\overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ be a PKE scheme and let $\mathsf{Fun} : \mathsf{Keys}_k \times \mathsf{Dom}_k \to \mathsf{Rng}_k$ be a family of functions with $\mathsf{Keys}_k = \{0,1\}^{\rho(k)}$, $\mathsf{Dom}_k = \{0,1\}^{n(k)}$, and $\mathsf{Rng}_k = \{0,1\}^{\rho(k)}$. The domain size $\{0,1\}^{n(k)}$ should be large enough to encode any public key generated from $\overline{\mathcal{K}}(1^k)$ and a message in $\{0,1\}^{\eta(k)}$. We build a PKE scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ from $\overline{\mathcal{AE}}$ and $\mathcal{F}$ as follows. Key generation and decryption are the same as in $\overline{\mathcal{AE}}$, and $\mathcal{E}(pk, m; r)$ computes $\bar{r} \leftarrow \mathsf{Fun}(r, pk, m)$ and returns $\overline{\mathcal{E}}(pk, m; \bar{r})$.

Theorem 1. *If $\overline{\mathcal{AE}}$ is* IND-XXX *and* Fun *is a secure* PRF*, then $\mathcal{AE}$ is* RA-XXX*.*

Proof. Let $\mathcal{AE}$ be constructed from Fun and $\overline{\mathcal{AE}}$ as above. Let $A \in \mathcal{A}_{\text{ra}}^{(1,1)\text{-XXX}}$ be an efficient ra-adversary attacking $\mathcal{AE}$. We assume that A never makes duplicate queries to the **Enc** oracle; this is without loss because all such queries will return the same response. Denote by G_0 the game RA defined in Section 3. Thus by definition,

$$\mathbf{Adv}^{\text{ra}}_{\mathcal{AE},A}(k) = 2 \cdot \Pr\left[\,\text{G}_0^A\,\right] - 1\ .$$

Now consider game G_1. The relevant procedures from games G_0 and G_1 are shown in Figure 4. In G_0, oracles **LR** and **Enc** use Fun to derive the randomness used to encrypt since this is what $\mathcal{AE}$ does. However, in G_1, those oracles choose fresh random coins and use those to encrypt the messages. We claim that these games appear close to adversary A by showing there exists an efficient prf-adversary B such that

$$\Pr\left[\,\text{G}_0^A\,\right] - \Pr\left[\,\text{G}_1^A\,\right] \leq \mathbf{Adv}^{\text{prf}}_{\mathsf{Fun},B}(k)\ .$$

The adversary B, attemping to decide if it is in the real or random world, flips a bit b and chooses a target public key pk^* by running the key generation algorithm. B then runs A just as in G_0 and G_1. On **Enc** and **LR** queries, B uses its **Fun** oracle to derive the randomness for encryption; in the case of the **LR** query, B encrypts the message corresponding to the bit b that it chose. In the CCA case, B answers **Dec** queries simply by using the secret key sk^* (which it knows because it chooses pk^* and sk^*). When A eventually outputs a guess bit b', B outputs 1 if $b = b'$ and 0 otherwise. We can see that when B is in the 'real' world (i.e., its **Fun** queries are answered using Fun), it perfectly simulates G_0 for A, while if B is in the 'random' world (i.e., its **Fun** queries are answered with random range points) then it perfectly simulates G_1 for A. The claim follows.

We then claim that there exists an efficient ind-adversary C such that

$$\mathbf{Adv}^{\text{ind}}_{\overline{\mathcal{AE}},C}(k) = 2 \cdot \Pr\left[\,\text{G}_1^A\,\right] - 1\ .$$

The ind-adversary C is given a target public key pk^* and access to an **LR** oracle to which it can make a single query. In the CCA setting it also has access to a **Dec** oracle. Adversary C runs A as in G_1, answering its oracle queries as follows. On A's single **LR** query, C simply answers with its own **LR** oracle. In the CCA setting C answers A's **Dec** queries using its own **Dec** oracle. On **Enc** queries from A, C encrypts the messages itself using fresh randomness and returns the resulting ciphertexts to A. At the end of execution, C outputs the same bit that A guesses. It is easy to see that C perfectly simulates the G_1 game for A and the claim follows.

Combining the above equations we can see that

$$\begin{aligned}\mathbf{Adv}^{\text{ra}}_{\mathcal{AE},A}(k) &= 2 \cdot \Pr\left[\,\text{G}_0^A\,\right] - 1\\ &\leq 2 \cdot (\mathbf{Adv}^{\text{prf}}_{\mathsf{Fun},B}(k) + \Pr\left[\,\text{G}_1^A\,\right]) - 1\\ &\leq 2 \cdot \mathbf{Adv}^{\text{prf}}_{\mathsf{Fun},B}(k) + \mathbf{Adv}^{\text{ind}}_{\overline{\mathcal{AE}},C}(k)\ .\end{aligned}$$

∎

The existence of secure PRFs is implied by the existence of one-way functions (which are necessary for PKE to exist), so we do not need any additional assumptions. In practice, one would want to instantiate the PRF using HMAC [7] or a block-cipher such as AES.

Notice that in the random oracle model we can replace $\mathsf{Fun}(r, pk, m)$ with $H(r, pk, m)$, where H is a random oracle, since a random oracle gives us a simple way to construct a PRF. This can lead to even more efficient schemes, but, of course, we lose the standard model guarantees.

Acknowledgements

We would like to thank Barath Raghavan for originally expressing to us his concern about the effects that resetting VMs might have on cryptographic primitives.

We would also like to thank Mihir Bellare and Thomas Ristenpart for providing useful feedback on an earlier version of this paper.

Scott Yilek is supported by NSF grant CNS–0831536.

References

1. `http://aws.amazon.com/ec2/`.
2. `http://www.openssl.org/`.
3. B. Barak, O. Goldreich, S. Goldwasser, and Y. Lindell. Resettably-sound zero-knowledge and its applications. In *42nd Annual Symposium on Foundations of Computer Science – FOCS 2001*, pages 116–125. IEEE, 2001.
4. M. Bellare, A. Boldyreva, K. Kurosawa, and J. Staddon. Multi-recipient encryption schemes: Efficient constructions and their security. *IEEE Transactions on Information Theory*, 53(11), 2007.
5. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology – CRYPTO 2007*, number 4622 in Lecture Notes in Computer Science, pages 535–552. Springer, 2007.
6. M. Bellare, Z. Brakerski, M. Naor, T. Ristenpart, G. Segev, H. Shacham, and S. Yilek. Hedged public-key encryption: How to protect against bad randomness. In *Advances in Cryptology – ASIACRYPT 2009*, LNCS. Springer, 2009. To Appear.
7. M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology – CRYPTO 1996*, number 1109 in Lecture Notes in Computer Science, pages 1–15. Springer, 1996.
8. M. Bellare, M. Fischlin, S. Goldwasser, and S. Micali. Identification protocols secure against reset attacks. In *Advances in Cryptology – EUROCRYPT 2001*, number 2045 in Lecture Notes in Computer Science, pages 495–511. Springer, 2001.
9. M. Bellare, T. Kohno, and V. Shoup. Stateful public-key cryptosystems: How to encrypt with one 160-bit exponentiation. In *Proceedings of the 13th ACM Conference on Computer and Communications Security – CCS 2006*, pages 380–389. ACM, 2006.
10. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of 1st ACM Conference on Computer and Communications Security – CCS 1993*, pages 62–73. ACM, 1993.
11. M. Bellare and P. Rogaway. Code-based game-playing proofs and the security of triple encryption. In *Advances in Cryptology – EUROCRYPT 2006*, number 4004 in Lecture Notes in Computer Science, pages 409–426. Springer, 2006.
12. R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable zero-knowledge. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing – STOC 2000*, pages 235–244. ACM, 2000.
13. P. M. Chen and B. D. Noble. When virtual is better than real. In *Proceedings of the 2001 Workshop on Hot Tpoics in Operating Systems*, pages 133–138, 2001.
14. R. S. Cox, S. D. Gribble, H. M. Levy, and J. G. Hansen. A safety-oriented platform for web applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 350–364. IEEE, 2006.
15. R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.
16. A. Desai, A. Hevia, and Y. L. Yin. A practice-oriented treatment of pseudorandom number generators. In *Advances in Cryptology – Eurocrypt 2002*, number 2332 in Lecture Notes in Computer Science, pages 368–383. Springer, 2002.
17. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.
18. V. Goyal and A. Sahai. Resettably secure computation. In *Advances in Cryptology – EUROCRYPT 2009*. Springer.
19. J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.
20. S. Micali and L. Reyzin. Soundness in the public-key model. In *Advances in Cryptology – CRYPTO 2001*, number 2139 in Lecture Notes in Computer Science. Springer, 2001.

21. N. Provos. A virtual honeypot framework. In *Proceedings of the 13th USENIX Security Symposium*, pages 1–14, 2004.
22. C. Rackoff and D. R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology – CRYPTO 1991*, number 576 in Lecture Notes in Computer Science, pages 433–444. Springer, 1992.
23. P. Rogaway and T. Shrimpton. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. In *Advances in Cryptology – EUROCRYPT 2006*, pages 373–390. Springer, 2006.

```
proc. Enc(pk, j, m):                          proc. LR(j, m_0, m_1):                    Game G_0
If CoinTab[j] = ⊥ then                        If CoinTab[j] = ⊥ then
    CoinTab[j] ←$ {0,1}^{ρ(k)}                    CoinTab[j] ←$ {0,1}^{ρ(k)}
r_j ← CoinTab[j]                              r_j ← CoinTab[j]
r̄ ← Fun(r_j, pk, m)                           r̄ ← Fun(r_j, pk*, m_b)
c ← Ē(pk, m; r̄)                               c ← Ē(pk*, m_b; r̄)
Return c                                      S ← S ∪ {c}
                                              Return c
```

```
proc. Enc(pk, j, m):                          proc. LR(j, m_0, m_1):                    Game G_1
If CoinTab[j] = ⊥ then                        If CoinTab[j] = ⊥ then
    CoinTab[j] ←$ {0,1}^{ρ(k)}                    CoinTab[j] ←$ {0,1}^{ρ(k)}
r_j ← CoinTab[j]                              r_j ← CoinTab[j]
r̄ ←$ {0,1}^{ρ(k)}                             r̄ ←$ {0,1}^{ρ(k)}
c ← Ē(pk, m; r̄)                               c ← Ē(pk*, m_b; r̄)
Return c                                      S ← S ∪ {c}
                                              Return c
```

Fig. 4. Games for the proof of Theorem 1. The procedures **Initialize**, **Finalize**, and **Dec** are omitted for brevity.