A Formal Treatment of Remotely Keyed Encryption*

Matt Blaze[1] Joan Feigenbaum[1] Moni Naor[2]

[1] AT&T Labs – Research
180 Park Avenue
Florham Park, NJ 07932 USA
{mab,jf}@research.att.com

[2] Dept. Applied Math. and Computer Science
Weizmann Institute of Science
Rehovot 76100, Israel
naor@wisdom.weizmann.ac.il

Abstract. Remotely keyed encryption schemes (RKESs), introduced by Blaze [6], support high-bandwidth cryptographic applications (such as encrypted video conferences) in which long-lived secrets (such as users' private keys) never leave lower-bandwidth environments such as secure smart-cards. We provide a formal framework in which to study the security of RKESs and give RKESs that satisfy our formal security requirements. Our RKESs are efficient in that the amount of communication and computation required of the smart-card is independent of the input size. In one proof of security, we use the pseudorandom permutation framework of Naor and Reingold [18] in an essential way.

Keywords: Block Ciphers, Pseudorandomness, Remotely Keyed Encryption, Session Keys, Smart-cards

1 Introduction

No cryptographic protocol is stronger than the mechanism protecting its secret keys. However, in many computing and communication systems, there is no "safe place" in which secret keys can be stored and cryptographic computations can be performed. This is especially true of modern networked computers; in some sense, every computer that communicates extensively with the world is bound at some point to be partly controlled by an unfriendly entity. Therefore, it is natural to consider adding an external, special-purpose device, such as a smart-card or a PCMCIA card, for storing cryptographic keys and computing cryptographic functions. Because they have only one purpose and communicate

* These results were presented in preliminary form at the Eurocrypt '98 conference (Helsinki, Finland), in June 1998. The third author was supported by a RAND2 grant from the EC.

only via a limited set of functions, such devices can be made much more secure than their general-purpose host machines. Furthermore, the difficulty and expense of designing and building a special-purpose device can be justified, because the device could be used in a wide variety of cryptographic protocols in *all* networked computers.

Unfortunately, it is not always practical to rely on such devices to perform all sensitive cryptographic operations. Inexpensive smart-cards, for example, have limited bandwidth, memory, and processor speed. If the host computer used such devices simple-mindedly, by just encrypting all external communication and all disk traffic, then the bandwidth of the link between the host and the cryptographic module would have to be at least as high as that between the host and the outside world. Even if the engineering problems of developing inexpensive high-bandwidth, high-performance cryptographic modules were completely solved, it would still be the case that, whenever the host's link to the outside world was upgraded, the modules and the secret keys they store would have to be changed, because cryptographic modules are typically designed never to reveal their keys.

This paper provides a formal treatment of the *remotely keyed encryption problem*: how to do bulk encryption and decryption for high-bandwidth applications in a way that takes advantage of both the superior power of the host and the superior security of the smart-card? If adversary $\mathcal{A}$ takes control of the host for a certain period, then clearly $\mathcal{A}$ will obtain whatever plaintext or ciphertext is resident in the host during that period. We would like to say formally that this is all it obtains: Once $\mathcal{A}$ loses control of the host, it cannot compute anything that it couldn't compute before it took control, except for the values it obtained explicitly while it was in control. (See Figure 1.)

Note that we are concerned with attacks on the host but *not* with direct attacks on the card; we assume that the card owner wants to safeguard the "remote keys" and that an attacker can only communicate with the card via its official communication channels. See, *e.g.*, Boneh *et al.* [7] and Biham and Shamir [5], for a discussion of direct attacks on cards. Note as well that the remotely keyed encryption problem is different from the one of having a smart-card take advantage of a host's superior processing power in order to do a *public-key* computation without leaking the *input* to the host. For a discussion of the (well-studied) problem of host-assisted public-key cryptography, see *e.g.*, Feigenbaum [10], Matsumoto *et al.* [17], and the references therein.

Furthermore, note that the goal of a remotely keyed encryption scheme (RKES) is not "session-key exchange" between two *different* hosts each connected to a card, where the two cards share a key. In an RKES application, such as the encryption of disk traffic, there is only one host; it encrypts at some point in time and then decrypts the stored ciphertext later. In settings in which the reason for exchanging keys is encryption, an RKES may replace a session-key exchange protocol and has the advantage of no interaction; however, there are other reasons for key exchange. See, for example, Shoup and Rubin [19] for a rigorous treatment of session-key exchange (following Bellare and Rogaway [3]),

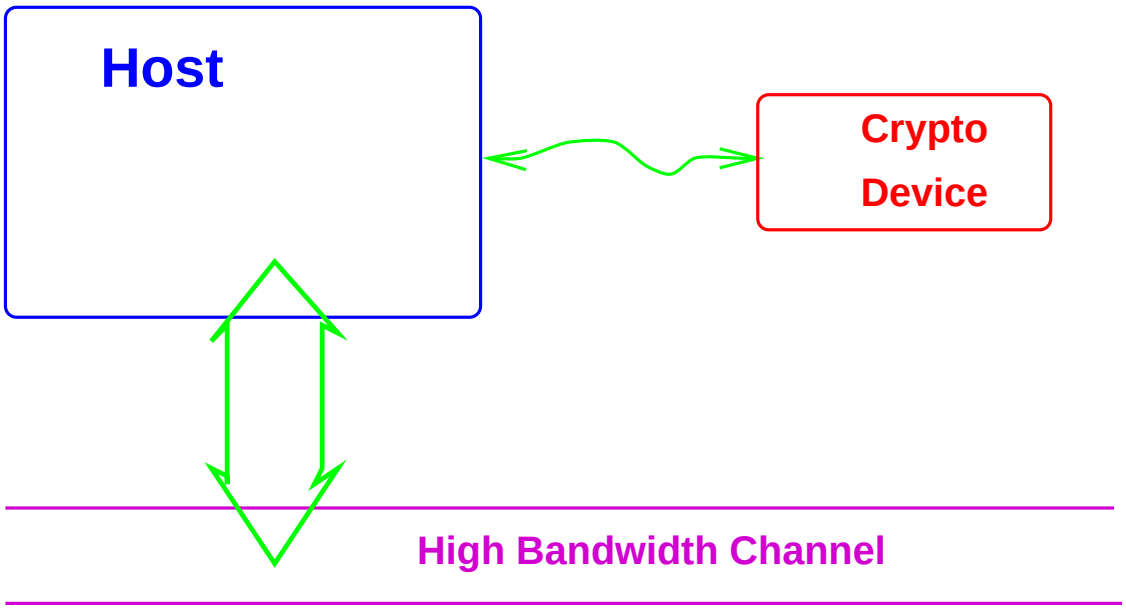

Fig. 1. High-level structure of a Remotely Keyed Encryption Scheme

in which the adversary is similar in power to the one we consider here. The Shoup-Rubin protocol requires several rounds of communication between the hosts.

We give formal definitions that capture the notion of "security" needed in our scenario and RKESs that satisfy the definitions. One of our schemes produces ciphertext of exactly the same length as the corresponding plaintext, and one produces ciphertext that is slightly longer. The length-preserving scheme has the advantage of allowing applications to adhere to strict formatting requirements, such as may be imposed on the encryption of disk traffic or data packets, while the length-increasing scheme has the advantage of satisfying a more stringent definition of security. Both of our RKESs have the desirable property that, for any input length, the amount of communication and computation that they require of the smart-card is independent of the input size.

History

Blaze [6] was the first to use the term "remotely keyed encryption" and to focus attention on the fact that many high-bandwidth applications need symmetric-key encryption schemes that store long-lived secret keys in low-bandwidth smart-cards.[1] He proposed a specific scheme but did not give a formal statement of

[1] Blaze [6] used the phrase "remotely keyed encryption *protocol*," and we use "remotely keyed encryption *scheme*." The terms are interchangeable.

the properties that an RKES should satisfy. Although the scheme in [6] does not satisfy the formal security requirements that we give in this paper, the basic idea of the scheme is sound, and we use it as a starting point in the design of an RKES that does satisfy our formal requirements. One weakness of the original scheme in [6] is that it may enable an adversary that has controlled the host during m interactions with the card subsequently to "forge" a plaintext/ciphertext pair that is not one of the m pairs he has obtained during the interaction.

Lucks [16] first noted that the RKES in [6] was not completely satisfactory; in particular, he noted the forgery weakness just described. Lucks [16] attempted to formalize the security properties that an RKES should have and to construct schemes that have them. Although the properties proposed in [16] are indeed desirable, we believe that the overall formalism and construction are flawed. Roughly speaking, [16] proposed that an RKES should have three properties: (i) Forgery security: If the adversary has controlled the host for m interactions, then it cannot produce $m+1$ plaintext/ciphertext pairs; (ii) Inversion security: Access to encryption should not imply the ability to decrypt and *vice versa*; (iii) Pseudorandomness: The encryption function should be a pseudorandom permutation. We suggest that an RKES might have these three properties but still be "insecure" in an intuitive sense.

In fact, the scheme in [16] is a good example of one that has properties (i), (ii), and (iii) but is intuitively insecure. That scheme uses the first two plaintext blocks in order to define an encryption key for the rest of the message; the encryption of these two blocks depends on the entire message, thus allowing property (i) to be satisfied. However, because the encryption key depends only on the first two plaintext blocks, an arbitrarily large set of messages all of which start with the same two blocks will always be encrypted with the same key. This is not a hypothetical situation: A set of files in a computer file system, for example, might always start with the same few bytes of structural information. An adversary that controls the host during the encryption or decryption of *one* file in such a set could subsequently decrypt the encryption of *any* file in the set. More fundamentally, the framework in [16] fails to recognize that it is nontrivial in this scenario to give a precise meaning to the statement that "the encryption function is a pseudorandom permutation." Once the adversary has witnessed one host/card interaction, it can subsequently distinguish between the encryption function and a random permutation by asking for the value on a single point.

We thus conclude that the formalism in [16] is inadequate. In this paper, we develop a formal framework that is both more precise and more stringent than those in the previous literature. In particular, we define pseudorandomness in a way that is meaningful for remotely keyed encryption.

Outline

We present our formalism in full detail in Section 3 below. A secure length-increasing RKES is given in Section 4, and a secure length-preserving RKES is given in Section 5.

2 Notation, Terminology, and Building Blocks

Definitions of standard cryptographic and complexity theoretic terms can be found in, for example, Goldreich [11], Luby [15], and Naor and Reingold [18]. The following is a description of the building blocks and terminology used throughout.

- The plaintext and the ciphertext are, respectively, X and Y. Usually, both are given in blocks and hence are denoted $X = (X_1, \ldots, X_n)$ and $Y = (Y_1, \ldots, Y_n)$, where each of X_i and Y_i is in $\{0,1\}^b$.
- The encryption and decryption functions of a block cipher are E and D. $E_S(X_j)$ denotes the encryption of plaintext block X_j with encryption key S, *i.e.*, $E_S, D_S : \{0,1\}^b \mapsto \{0,1\}^b$ and $D_S(E_S(X_j)) = X_j$.
  The security property required of $E_S(\cdot)$ is that it should be a strong pseudorandom permutation, *i.e.*, that any probabilistic, polynomial-time adversary given access to $E_S(\cdot)$ and $D_S(\cdot)$ cannot distinguish them from a truly random permutation. A thorough treatment of strong pseudorandom permutations is given by Luby [15], who calls them "super" pseudorandom permutations.
- A pseudorandom function $F_S : \{0,1\}^b \mapsto \{0,1\}^b$; it may or may not be identical to the encryption function E of the block cipher. (Note that every pseudorandom permutation is also a pseudorandom function, where the added advantage of a distinguisher is bounded by $m^2/2^b$.) We use F_S, rather than E_S, in situations that never require the function to be inverted.
- A length-preserving method G_S for encrypting an n-block plaintext $(X_1, \ldots, X_n)$ using encryption key S. $G_S^j(X_1, \ldots, X_n)$ denotes the jth block of the resulting ciphertext. The corresponding decryption function is denoted $\hat{G}_S$, and the jth block of the plaintext that results from decrypting $(Y_1, \ldots, Y_n)$ is denoted $\hat{G}_S^j(Y_1, \ldots, Y_n)$.
  The security requirement for G_S is that, for any $X_1, X_2, \ldots, X_n$, if S is chosen uniformly at random, then $G_S(X_1, \ldots, X_n)$ is pseudorandom (*i.e.*, indistinguishable from a random string of similar length). We impose a similar requirement on $\hat{G}_S$. Possible realizations of G_S are:
  - Apply a pseudorandom generator to S and Xor the resulting sequence with $X_1, \ldots, X_n$.
  - Use E_S with some sort of chaining, *e.g.*, CBC. The security of such an operation follows from [2].
- A collision-intractable hash function $H : \{0,1\}^* \mapsto \{0,1\}^b$. "Collision-intractability" means that it is computationally infeasible to find distinct X and X' such that $H(X) = H(X')$.
- The adversary is in general an oracle machine M where $M^{(f,f^{-1})}$ that has access to the *function pair* (f, f^{-1}). As in [18], M may submit two forms of queries to the function-pair oracle: A query of the form $(+, x)$ results in the answer $f(x)$, and one of the form $(-, y)$ results in the answer $f^{-1}(y)$.

As usual, various parameters are needed in order to express things in full detail. In particular, there is an underlying size (security) parameter u, and there are three polynomially bounded functions that measure the key length $\kappa(u)$, the

block length $b(u)$, and the number of blocks $n(u)$. The total length of the input to any of our protocols is a polynomial function of $\kappa(u)$, $b(u)$, and $n(u)$. For clarity of presentation, we suppress these parameters whenever possible, but they are an implicit part of everything that follows. For example, the statement "F is a pseudorandom function" means that $F : \{0,1\}^{\kappa(u)} \times \{0,1\}^{b(u)} \to \{0,1\}^{b(u)}$ is a pseudorandom function generator, in the sense of [12] or [15, Lecture 12]. Similarly, additional (but standard) detail is also required to say precisely what is meant by a "random" function, permutation, or function pair. These details can be found in, for example, [11,15,18].

In our schemes, the card stores the keys of several functions and permutations. Our "physical assumption" is that the adversary cannot read these keys directly. Note that we do *not* need the assumption that intermediary values, *i.e.*, results of applying these cryptographic primitives, remain secret.

3 Definitions of Secure Remotely Keyed Encryption

Intuitively, we would like an RKES to resist the following form of attack. Adversary $\mathcal{A}$ may gain control of the host temporarily. During this *host phase* of his attack, $\mathcal{A}$ may have a total of m interactions with the card, where m is polynomially bounded. He may send any message to the card during one of these interactions and may deviate from the protocol. However, since m is an upper bound on the total number of interactions, $\mathcal{A}$ obtains at most m plaintexts and m ciphertexts during the host phase. After these m interactions with the card, $\mathcal{A}$ loses control of the host. He should subsequently have no advantage in his attempts to find the encryption (resp., decryption) of plaintexts (resp., ciphertexts) other than those m that he found explicitly during the host phase.

One step in formalizing this intuition is to make precise what we mean by "no advantage." We will do this in terms of pseudorandomness. That is, the encryption and decryption protocols of the RKES compute some function pair (f, f^{-1}), and this function pair should appear truly random to $\mathcal{A}$. During the host phase, $\mathcal{A}$ learns the value of f and f^{-1} each at m points. This should give him "no advantage" in the sense that if, in a *distinguishing phase* that takes place after he loses control of the host, $\mathcal{A}$ is asked to distinguish between (f, f^{-1}) and a truly random function pair, he should be able to do so only with negligible probability.

Because the amount of communication between the host and the card in an efficient RKES should be much shorter than the input length, there is a complication that is missing in the standard definition of pseudorandomness. If, during the host phase, $\mathcal{A}$ learns the value of f on m points $X^1, X^2, \ldots, X^m$, then f almost certainly does *not* look random to $\mathcal{A}$ on these points, because the transcript of the host phase and the description of the protocol constitute a short description of $(X^1, f(X^1)), (X^2, f(X^2)), \ldots, (X^m, f(X^m))$. Our formalism addresses this by requiring that, between the host phase and the distinguishing phase, a nondeterministic choice occurs: Either (f, f^{-1}) is replaced with a truly

random function pair or it is kept the same. $\mathcal{A}$'s challenge is thus to decide whether or not a switch occurred or not.

Now a new complication arises: We cannot allow the adversary, during the distinguishing phase, to query the oracle about any of the m plaintexts and m ciphertexts that he obtained during the host phase. If the adversary $\mathcal{A}$ does so and receives the same answers as he did the first time around, then $\mathcal{A}$ will know, with high probability, that there was no switch and (f, f^{-1}) remained unchanged. If $\mathcal{A}$ receives a different answer on one of these queries during the distinguishing phase than he did during the host phase, then he can conclude with certainty that the oracle is not (f, f^{-1}).

We would like therefore to "filter" those values that appeared in the host phase. The problem with making this discussion rigorous is that the adversary's actions during the host phase do not necessarily correspond to specific inputs, and certainly there are many inputs that yield the same (host,card) transcript. To overcome this problem, we introduce an *arbiter* into our definition of secure remotely keyed encryption. The purpose of the arbiter, which we denote by $\mathcal{B}$, is to make sure that $\mathcal{A}$ does not ask during the distinguishing phase any of the queries that it asked during the host phase. $\mathcal{B}$ should be a simple function of the transcript of the communication that occurred during the host phase and should have limited filtering ability. Instead of saying that the inputs queried during the host phase are excluded (which is *not* well-defined), we say that if $\mathcal{A}$ has had m interactions with the card during the host phase, then $\mathcal{B}$ is allowed to filter no more than m queries during the distinguishing phase. Note that there is no need actually to implement this arbiter; rather, a (host,card) protocol is secure if there exists such an arbiter.

This discussion can be summarized as follows.

Definition 1. A **length-preserving RKES** is a pair of protocols, one for encryption and one for decryption, to be executed by a *host* and a *card*. The length of a ciphertext must be the same as that of the corresponding plaintext. The RKES is **secure** if there is a polynomial-time *arbiter* $\mathcal{B}$ that can enforce the following restriction on any probabilistic, polynomial-time *adversary* $\mathcal{A}$ and any polynomial bound m: During the *host phase*, $\mathcal{A}$ may play the role of the host in a total of m interactions with the card. During this phase, $\mathcal{A}$ may send any message to the card and does not necessarily follow the encryption or decryption protocol. Between the host phase and the *distinguishing phase*, a nondeterministic choice is made between continuing to use the RKES or switching to a random function pair. The arbiter $\mathcal{B}$ receives as input the transcript of the host-phase communication between the host and the card. During the distinguishing phase, $\mathcal{A}$ may run any probabilistic, polynomial-time test T that submits plaintexts or ciphertexts to $\mathcal{B}$; on at most m of the plaintexts and m of the ciphertexts, $\mathcal{B}$ may choose to run the RKES, even if a switch to a random function pair was made between phases. Otherwise the plaintext (resp. ciphertext) is given to the encryption (resp. decryption) protocol if no switch was made between phases and to the random function f (resp. f^{-1}) if a switch was made. The difference between the probability that T accepts on a continuation of the RKES and

the probability that T accepts on a switch to a random function pair must be negligible.

A natural way to relax the above requirement is to allow $\mathcal{B}$ to reject polynomially in m many input/output, instead of exactly m. However, the constructions given in Section 5 achieve the stricter notion.

We now turn our attention to length-increasing RKESs. These should be easier to construct than length-preserving RKESs. However, we can require additional security properties of length-increasing RKESs that are not achievable in the length-preserving case, and thus we need a second definition. In the length-increasing case, each plaintext may correspond to multiple ciphertexts, because the ciphertext space is bigger than the plaintext space. We can (and should) use a *probabilistic encryption* algorithm that induces, for each plaintext, a probability distribution on a corresponding set of ciphertexts [13]. Furthermore, if ciphertexts are of length $c(u)$, it need not be the case that every string in $\{0,1\}^{c(u)}$ is a legitimate encryption of some plaintext.

Because we have these two types of flexibility that are not present in the length-preserving case, we can impose two additional security properties. The first is *semantic security*, as defined by Goldwasser and Micali [13] – "whatever is efficiently computable given the ciphertext is efficiently computable without it." We prefer to work with the equivalent "real-or-random" definition: No probabilistic, polynomial-time adversary can distinguish between a random ciphertext and the encryption of a chosen plaintext, even if it had prior access to the encryption and decryption mechanisms (see Bellare *et al.* [1]). To make use of the property that not every string in $\{0,1\}^{c(u)}$ must correspond to a plaintext, we give the decryption algorithm the option of outputting a distinguished string "invalid." Intuitively, the decryption algorithm is supposed to return the correct plaintext when given as input a ciphertext that has been produced by the encryption algorithm but to return "invalid" when given anything else.

The second security requirement that we impose on length-increasing RKESs but not on length-preserving ones is *self-validation*: Even if it had prior access to the encryption and decryption mechanisms, a probabilistic, polynomial-time adversary should not be able to generate a new valid ciphertext, *i.e.*, one that it did not obtain explicitly from the encryption algorithm and on which the decryption algorithm does not output "invalid." Note that the combination of these two properties yields a private-key *non-malleable cryptosystem* as defined by Dolev *et al.* [8][2], in which it is infeasible not only to compute anything about a plaintext but also to generate the ciphertext of a related message. It is worth noting that self-validation together with semantic security is a *stricter* requirement than non-malleability: In a non-malleable cryptosystem, the adversary cannot produce ciphertexts of unrelated messages, but he may be able to produce ciphertexts of random messages.

For length-increasing RKESs, we would like to say that, after the host phase, the adversary cannot tell whether he is interacting with the real protocols or with

[2] The private-key case is discussed only in the expanded version.

a "random, self-validating black box," given that an arbiter is filtering based on a transcript of the host-phase communication. A "random, self-validating black box" contains an encryption box and a decryption box. On any input of the appropriate plaintext length, the encryption box outputs a random string of the appropriate ciphertext length. The decryption box outputs "invalid" on all inputs, except those that were previously output by the encryption box, and on those it outputs the input string on which the encryption box gave this output. Note that such a pair of boxes is *not* a encryption scheme in the usual sense: It is not "memoryless" but rather assumes that the encryption and decryption boxes can remember and "communicate about" the strings they have processed. The challenge in creating a self-validating encryption scheme is to enable the decryption algorithm to know when to output "invalid" even though it cannot communicate with the encryption algorithm, and neither algorithm can remember which strings it has previously processed.

Self-validating encryption makes sense only when the ciphertext length $c(u)$ exceeds the plaintext length by enough to make a random string in $\{0,1\}^{c(u)}$ "invalid" except with negligible probability. This is not an onerous requirement, as we will see in Section 4.

Definition 2. A **length-increasing RKES** is a pair of protocols, one for encryption and one for decryption, to be executed by a *host* and a *card*. The length of a ciphertext is greater than the length of the corresponding plaintext. If its input is a ciphertext that has previously been output by the encryption protocol, the decryption protocol outputs the corresponding plaintext; otherwise, it may output "invalid" (and in fact will do so on most inputs). The RKES is **secure** if there is a polynomial-time *arbiter* $\mathcal{B}$ that can enforce the following restriction on any probabilistic, polynomial-time *adversary* $\mathcal{A}$ and any polynomial bound m: During the *host phase*, $\mathcal{A}$ may play the role of the host in m interactions with the card. During this phase, $\mathcal{A}$ may send any message to the card and does not necessarily follow the encryption or decryption protocol. Between the host phase and the *distinguishing phase*, a choice is made between continuing to use the RKES or switching to a random, self-validating black box that has not yet received any queries. $\mathcal{B}$ gets as input the transcript of the host-phase communication between the host and the card. During the distinguishing phase, $\mathcal{A}$ may run any probabilistic, polynomial-time test T that submits plaintexts or ciphertexts to $\mathcal{B}$. On at most m of the ciphertexts (but *not* the plaintexts), $\mathcal{B}$ may choose to run the RKES, even if a switch was made between phases. Otherwise the plaintext (resp. ciphertext) is given to the encryption (resp. decryption) protocol if no switch was made between phases and to the random, self-validating black box if a switch was made. The difference between the probability that T accepts on a continuation of the RKES and the probability that T accepts on a switch to a random, self-validating black box must be negligible.

Three remarks are in order about this definition. First, it generalizes the corresponding definitions in standard (*i.e.*, not remotely keyed) encryption. If one fixes $m = 0$, *i.e.*, makes the host phase trivial, then Definition 1 reduces to

the definition of a strong pseudorandom permutation. Similarly, in Definition 2, fixing $m = 0$ yields the definition of a semantically secure and self-validating private-key cryptosystem. Second, note that an important difference between the length-preserving case and the length-increasing one is that, in the latter, the arbiter does *not* have the power to route plaintexts to the RKES. It may only do so with ciphertexts. The third remark worth making is that these definitions are concerned with security rather than efficiency. Note, for instance, that a strong pseudorandom permutation evaluated solely by the card satisfies Definition 1. Clearly, an RKES is most useful if the computational, memory, and bandwidth demands on the card are small. In particular, it is desirable for all to be slowly growing functions of the block length b and key length κ and to be independent of n, the number of blocks in the plaintext, as they are in the schemes given below. Finally, we assume that the length of the message is known. It may be implicitly known, *e.g.*, the size of a disk sector or a data packet, or it may be conveyed by some other protocol. Note that it is possible to set the protocols in Sections 4 and 5 so that they yield a different permutation for each message length.

4 A Secure, Length-Increasing RKES

We first describe a simple scheme (Scheme I1) that is *not* secure in the sense of Definition 1. The adversary may create arbitrarily many "valid" ciphertexts. At the start of the execution of the encryption protocol, the host obtains a random or pseudorandom number S by the best method at its disposal; alternatively, the card could provide S to the host. The RKES requires only that S can be encrypted using the block encryption algorithm E. The private key stored in the smart-card is denoted by k_1. The idea of the protocol is simple: The host generates a session-key and the card provides its encryption.

Scheme I1: Insecure, length-increasing RKES:

Encryption protocol: input $X_1, \ldots, X_n$**; output** $Y_0, Y_1, \ldots, Y_n$

I1-0 Generate S
I1-1 Host $\rightarrow$ Card: S
I1-2 Card: $Y_0 \leftarrow E_{k_1}(S)$
I1-3 Card $\rightarrow$ Host: Y_0
I1-4 Host: set Y_0 as received message; For $j \leftarrow 1$ to n, $Y_j \leftarrow G_S^j(X_1, \ldots, X_n)$.

Decryption protocol: input $Y_0, Y_1, \ldots, Y_n$**; output** $X_1, \ldots, X_n$

I1-5 Host $\rightarrow$ Card: Y_0
I1-6 Card: $S \leftarrow D_{k_1}(Y_0)$
I1-7 Card $\rightarrow$ Host: S
I1-8 Host: For $j \leftarrow 1$ to n, $X_j \leftarrow \hat{G}_S^j(Y_1, \ldots, Y_n)$.

Clearly, Scheme I1 makes no attempt to reject invalid ciphertexts. Any sequence $Y_0, Y_1, \ldots, Y_n$ will be decrypted. Furthermore, the adversary may "forge" as many plaintext/ciphertext pairs as he wishes, following a host phase in which he carries out just one execution of either the encryption protocol or the decryption protocol.

Scheme I2 is a secure, length-increasing RKES based on the same basic idea as Scheme I1, *i.e.*, using the luxury of an additional ciphertext block to store an encryption of a session key. To achieve self-validation, it uses another additional ciphertext block to store a value that an adversary cannot compute without running the encryption protocol, because of the properties of the cryptographic building blocks E, F, G, and H. The private key stored in the smart-card is partitioned into 4 parts, denoted k_1, k_2, k_3, and k_4, that play different roles in the protocols. As mentioned in Section 2, the pseudorandom function F is used when inversion is not needed, and the encryption functions E and G are used when it is.

We have designed Scheme I2 for maximum clarity and have not sought to optimize it in several ways that could save constant factors in space and thus might be relevant in applications with very tight constraints. For example, we have not attempted to minimize the number of private-key components $(k_1, \ldots, k_4)$ or the number of distinct cryptographic building blocks (E, F, G, and H), because such optimizations would not help to illustrate the overall structure that an RKES should have in order to satisfy our formal definition. It is possible, however, to use the same key component or the same building block in multiple roles.

Scheme I2: Secure, length-increasing RKES:

Encryption protocol: input $X_1, \ldots, X_n$**; output** $t, Y_0, Y_1, \ldots, Y_n$

I2-0 Generate S
I2-1 Host: For $j \leftarrow 1$ to n, $Y_j \leftarrow G_S^j(X_1, \ldots, X_n)$
I2-2 Host: $h \leftarrow H(Y_1, Y_2, \ldots, Y_n)$
I2-3 Host $\rightarrow$ Card: S, h
I2-4 Card: $Y_0 \leftarrow E_{k_1}(S)$
I2-5 Card: $t \leftarrow F_{k_4}(F_{k_3}(Y_0) \oplus F_{k_2}(h))$
I2-6 Card $\rightarrow$ Host: Y_0, t

Decryption protocol: input $t, Y_0, Y_1, \ldots, Y_n$**; output** $X_1, \ldots, X_n$ **or "invalid"**

I2-7 Host: $h \leftarrow H(Y_1, Y_2, \ldots, Y_n)$
I2-8 Host $\rightarrow$ Card: Y_0, h, t
I2-9 Card: If $t \neq F_{k_4}(F_{k_3}(Y_0) \oplus F_{k_2}(h))$ Then $S \leftarrow$ "invalid"
Else $S \leftarrow D_{k_1}(Y_0)$
I2-10 Card $\rightarrow$ Host: S
I2-12 Host: If $S \neq$ "invalid"
Then {For $j \leftarrow 1$ to n, $X_j \leftarrow \hat{G}_S^j(Y_1 \ldots, Y_n)$; Output $(X_1, \ldots, X_n)$}
Else Output "invalid"

As required by Definition 2, the arbiter $\mathcal{B}$ does not filter queries of the form $(+, (X_1, \ldots, X_n))$ during the distinguishing phase; it just sends them to the encryption protocol if no switch was made between phases and to a random, self-validating black box if a switch was made. On queries of the form $(-, (t, Y_0, Y_1, \ldots, Y_n))$, $\mathcal{B}$ computes $h = H(Y_1, \ldots, Y_n)$ and checks whether (h, Y_0, t) occurs in the transcript of the host phase. If it does, then $\mathcal{B}$ routes the query to the decryption protocol, regardless of whether a switch was made between phases; if it doesn't, then $\mathcal{B}$ routes it either to the decryption protocol or to the random, self-validating black box, depending on whether a switch was made.

Theorem 3. Scheme I2 is a secure, length-increasing RKES.

Proof. The definitions of the cryptographic building blocks E, F, and G imply straightforwardly that any sequence of *encryptions* is indistinguishable from a random one (and hence from the output of a random, self-validating box). Consider the case of decryption queries. Suppose that such a query $(-, (t, Y_0, Y_1, \ldots, Y_n))$ does *not* correspond to an encryption query $(+, (X_1, \ldots, X_n))$ that occurred earlier in the distinguishing phase and that $(H(Y_1, \ldots, Y_n), Y_0, t)$ did not appear in the host phase (*i.e.*, the query is not filtered by the arbiter). A random, self-validating black box will answer such a query by saying "invalid." The real protocol will also answer "invalid" if $t \neq F_{k_4}(F_{k_3}(Y_0) \oplus F_{k_2}(h))$, where $h = H(Y_1, \ldots, Y_n)$, but it will produce a decryption if $t = F_{k_4}(F_{k_3}(Y_0) \oplus F_{k_2}(h))$, or in other words if t "validates" (Y_0, h). Thus an adversary can tell whether a switch was made between phases only if it can find $(t, Y_0, Y_1, \ldots, Y_n)$ such that $t = F_{k_4}(F_{k_3}(Y_0) \oplus F_{k_2}(H(Y_1, \ldots, Y_n)))$ by some method other than submission of a query $(+, (X_1, \ldots, X_n))$.

The collision-intractability of H implies that, with all but non-negligible probability, the adversary cannot find $(Y_1, \ldots, Y_n) \neq (Y_1', \ldots, Y_n')$ such that $H(Y_1, \ldots, Y_n) = H(Y_1', \ldots, Y_n')$. Therefore the adversary is left with two possibilities: (i) Find "colliding pairs" $(Y_0, h) \neq (Y_0', h')$ such that

$$F_{k_3}(Y_0) \oplus F_{k_2}(h) = F_{k_3}(Y_0') \oplus F_{k_2}(h'),$$

or (ii) Guess the value of $F_{k_4}(F_{k_3}(Y_0) \oplus F_{k_2}(h))$ where the value $F_{k_3}(Y_0) \oplus F_{k_2}(h)$ did not appear in Step I2-5 of any previous query during the Host or Distinguishing Phases. However, the probability of (ii) is bounded by $1/2^b$ plus the probability of distinguishing F_{k_4} from a truly random function. Therefore we concentrate on the probability of finding colliding pairs. Our formal claim is:

Lemma 4. Let $\mathcal{A}$ be a probabilistic, polynomial-time adversary that has m_1 interactions with the card during the host phase and makes m_2 oracle queries during the distinguishing phase. Then the probability that $\mathcal{A}$ can produce colliding pairs is at most $\frac{(m_1+m_2)^2}{2^b} + \varepsilon$, where b is the block length and ε is an upper bound on the probability that $\mathcal{A}$ can distinguish at least one of the two pseudorandom functions F_{k_2} and F_{k_3} from random functions.

Proof. The argument that collisions are hard to find, even during the host phase, has a standard form: First show that the probability would be negligible if the F's were truly random functions and then use a "hybrid argument" to show that it remains negligible when the cryptographic F's are used. (For an explanation of hybrid arguments, see Goldreich [11]).

Suppose that F_{k_2} and F_{k_3} are truly random functions. Then all the values $F_{k_2}(Y_0)$ and $F_{k_3}(h)$ are random values. The probability that $(Y_0, h) \neq (Y_0', h')$ but $F_{k_3}(Y_0) \oplus F_{k_2}(h) = F_{k_3}(Y_0') \oplus F_{k_2}(h')$ is $1/2^b$. There are $(m_1 + m_2)^2$ possible pairs. (A similar argument was applied to cipher-block chaining by Bellare *et al.* [2].) □

Therefore, with all but negligible probability, the most that the adversary can obtain during the distinguishing phase is a collection of encryptions (and decryptions that it could have obtained anyway because they were submitted by the adversary as queries during the host phase). We conclude that it cannot distinguish between a random, self-validating black-box and the original encryption algorithm. The analysis further implies that the maximum number of different decryption queries that the arbiter will reroute is bounded by m_1. □

5 A Secure, Length-Preserving RKES

We now present Scheme P, a secure, length-preserving RKES. The card's secret key has four components k_1, k_2, k_3, and k_4. As in the previous section, we have designed Scheme P for maximum clarity and have not sought to optimize by, for example, minimizing the number of distinct key components or cryptographic building blocks.

The scheme is best understood as part of the Naor-Reingold [18] framework for constructing and proving the security of pseudorandom permutations. In this framework, the pseudorandom permutation Π is the composition of three permutations: $\Pi \equiv p_2^{-1} \circ J \circ p_1$. (See Figure 2.) In general, p_1 and p_2^{-1} are "lightweight," and J is where most of the work is done. In our setting, J will be the part performed mostly by the host, and p_1 and p_2^{-1} will be done mostly by the card.

The "heavyweight" building block J should behave as a random permutation on most inputs. An important step in applying the Naor-Reingold framework is the identification of a collection of input-output sequences that are called "J-good." For an input-output sequence $\langle (X^1, Y^1), \ldots, (X^m, Y^m) \rangle$ to be J-good, $Pr_J[Y^i = J(X^i), 1 \leq i \leq m]$ should be close to $2^{-\ell \cdot m}$, where $\ell = n \cdot b$, *i.e.*, the probability should be close to what it would be if J were a truly random function. The role of the permutations p_1 and p_2 is to ensure that, with overwhelming probability, the inputs and outputs to J form an J-good sequence, even if the inputs to Π are chosen by an adaptive adversary, under a chosen plaintext and ciphertext attack. Thus a sequence is J-good or not based on p_1 and p_2.

Scheme P: Secure length-preserving RKES:

Encryption protocol: input $X_1, \ldots, X_n$**; output** $Y_1, \ldots, Y_n$

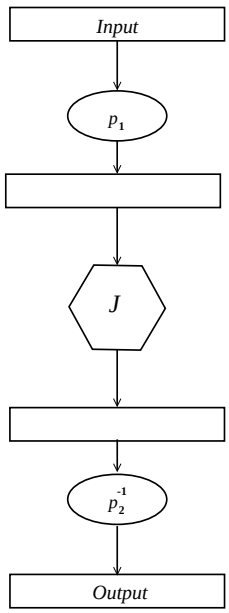

Fig. 2. High-level structure of pseudorandom permutation in Naor and Reingold's Framework

P1 Host: $h_x \leftarrow H(X_2, \ldots, X_n)$
P2 Host $\rightarrow$ Card: h_x, X_1
P3 Card: $W \leftarrow E_{F_{k_1}(h_x)}(X_1)$
P4 Card: $Z \leftarrow E_{k_2}(W)$
P5 Card: $S \leftarrow F_{k_3}(W)$
P6 Card $\rightarrow$ Host: S
P7 Host: For $j \leftarrow 2$ to n, $Y_j \leftarrow G_S^j(X_2, \ldots, X_n)$.
P8 Host: $h_y \leftarrow H(Y_2, \ldots, Y_n)$
P9 Host $\rightarrow$ Card: h_y
P10 Card: $Y_1 \leftarrow E_{F_{k_4}(h_y)}(Z)$
P11 Card $\rightarrow$ Host: Y_1

Decryption protocol: input $Y_1, \ldots, Y_n$**; output** $X_1, \ldots, X_n$

P12 Host: $h_y \leftarrow H(Y_2, \ldots, Y_n)$
P13 Host $\rightarrow$ Card: h_y, Y_1
P14 Card: $Z \leftarrow D_{F_{k_4}(h_y)}(Y_1)$
P15 Card: $W \leftarrow D_{k_2}(Z)$
P16 Card: $S \leftarrow F_{k_3}(W)$
P17 Card $\rightarrow$ Host: S
P18 Host: For $j \leftarrow 2$ to n, $X_j \leftarrow \hat{G}_S^j(Y_2, \ldots, Y_n)$.
P19 Host: $h_x \leftarrow H(X_2, \ldots, X_n)$
P20 Host $\rightarrow$ Card: h_x

P21 Card: $X_1 \leftarrow D_{F_{k_1}(h_x)}(W)$
P22 Card $\rightarrow$ Host: X_1

In our construction, p_1 and p_2 produce output that depends on all the input blocks, but they change only the first block. That is

$$p_1 : (X_1, X_2, \ldots, X_n) \mapsto (W, X_2, \ldots, X_n),$$

where W is a function of X_1 and $h_x = H(X_2, \ldots, X_n)$, and

$$p_2 : (Y_1, Y_2, \ldots, Y_n) \mapsto (Z, Y_2, \ldots, Y_n),$$

where Z is a function of Y_1 and $h_y = H(Y_2, \ldots, Y_n)$.

Good sequences will be those in which different $X_1, \ldots, X_n$ and $X'_1, \ldots, X'_n$ are mapped by p_1 to different W and W', and similarly different Y's are mapped by p_2 to different Z's. To obtain permutations p_1 and p_2 with the right properties, we define a new primitive called *non-colliding encryption*.

Definition 5. A *non-colliding encryption scheme* is a pair of keyed functions $C_k : \{0,1\}^b \times \{0,1\}^b \mapsto \{0,1\}^b$ and $\hat{C}_k : \{0,1\}^b \times \{0,1\}^b \mapsto \{0,1\}^b$ with the following two properties.

1. For all $V \in \{0,1\}^b$ and $h \in \{0,1\}^b$, the functions satisfy $\hat{C}_k(C_k(V,h),h) = V$ and $C_k(\hat{C}_k(V,h),h) = V$. (Note that this property allows us to use C to "store" V, provided h is retrievable.)
2. Let $\mathcal{A}$ be a probabilistic, polynomial-time adversary that is allowed to query C_k and $\hat{C}_k$ adaptively. We say that "(V,h) appears in a (polynomial-length) sequence of queries" if $\mathcal{A}$ asks for $C_k(V,h)$ directly or if V is the reply to some direct query $\hat{C}_k(U,h)$. If the key k is chosen at random, then $\mathcal{A}$ has only a negligible probability of finding two pairs $(V,h) \neq (V',h')$ such that (a) $C_k(V,h) = C_k(V',h')$, and (b) at least one of (V,h) and (V',h') did not appear in the sequence of queries.

In Scheme P, the permutations p_1 and p_2 are determined by $(C_k, \hat{C}_k)$. For example, $p_1 : (X_1, X_2, \ldots, X_n) \mapsto (W = C_{k_1}(X_1, H(X_2, \ldots, X_n)), X_2, \ldots, X_n)$, and p_2 is defined similarly. The "storage" capability of non-colliding encryption ensures that p_1 is indeed a permutation, because $p_1^{-1} : (W, X_2, \ldots, X_n) \mapsto (X_1 = \hat{C}_{k_1}(W, H(X_2, \ldots, X_n)), X_2, \ldots, X_n)$. Note that p_1, p_2, and their inverses can be computed efficiently, for any non-colliding encryption scheme $(C_k, \hat{C}_k)$, by the host and the card together. The permutation J depends on the two key components k_3 and k_4. $J : (W, X_2, \ldots, X_n) \mapsto (Z, Y_2, \ldots, Y_n)$, where $Z = E_{k_3}(W)$, $(Y_2, \ldots, Y_n) = G_{F_{k_4}(W)}(X_2, \ldots, X_n)$, and $G_S : \{0,1\}^{(n-1)b} \mapsto \{0,1\}^{(n-1)b}$. The overall permutation computed by the encryption protocol is $\Pi = p_2^{-1} \circ J \circ p_1$, and this too can be computed efficiently by the (host, card) pair.

Our main result is as follows.

Theorem 6. Scheme P is a secure, length-preserving RKES.

Proof. To prove this result, we must define an arbiter $\mathcal{B}$, construct a non-colliding encryption scheme, and apply the Naor-Reingold framework [18]. Applying the framework entails identifying J-good sequences and proving that the overall construction gives a strong pseudorandom permutation. The identification of J-good sequences has to take into account the "two-phase" aspect of the definition of security of RKESs; this is a complication that is not present in the original Naor-Reingold paper. We address each of these issues in turn.

Arbiter:

$\mathcal{B}$ records all the pairs (h_x, X_1) and (h_y, Y_1) that appear in the host phase. The list of pairs is easy to deduce from the transcript. During the distinguishing phase, $\mathcal{B}$ does the following for each encryption query $(+, (X_1, \ldots, X_n))$. First, it computes $h_x = H(X_2, \ldots, X_n)$. If the pair (h_x, X_1) appeared in the host phase, then $\mathcal{B}$ answers the query using the encryption protocol; otherwise, it uses either the encryption protocol or the random permutation, depending on whether the decision between phases was to continue or to switch. Similarly, when it receives a decryption query $(-, (Y_1, \ldots, Y_n))$ during the distinguishing phase, $\mathcal{B}$ first computes $h_y = H(Y_2, \ldots, Y_n)$; then, if the pair (h_y, Y_1) appeared in the host phase, $\mathcal{B}$ uses the decryption protocol to answer the query, and otherwise it uses either the decryption protocol or the random function inverse, depending on whether the decision between phases was to continue or to switch.

Non-colliding encryption:

Assume without loss of generality that the key k required by the non-colliding encryption scheme is the same length (b bits) as the key for the block cipher E. If the block-cipher keys are too short, they can be stretched using a pseudorandom generator, and if they are too long, they can be truncated. Recall that F is a pseudorandom function.

NCE Construction 1: Let $C_k(V, h) = E_{F_k(h)}(V)$ and $\hat{C}_k(V, h) = D_{F_k(h)}(V)$.

Lemma 7. NCE Construction 1 is a non-colliding encryption scheme.

Proof. This construction obviously satisfies Property 1 of Definition 5.

To prove that is also satisfies Property 2, consider an adaptive adversary $\mathcal{A}$ that makes a sequence of m queries to C_k and $\hat{C}_k$. Let ε_m^1 (resp. ε_m^2) be an upper bound on the probability that, with m queries, $\mathcal{A}$ can distinguish F from a truly random function (resp. an upper bound on the probability that, with m queries, $\mathcal{A}$ can distinguish a collection of $m+2$ pseudorandom permutations from a collection of $m+2$ truly random permutations). Then $\mathcal{A}$'s chance of finding two pairs $(V, h) \neq (V', h')$ such that (a) $C_k(V, h) = C_k(V', h')$, and (b) at least one of (V, h) and (V', h') did not appear in the sequence of queries is at most

$$\frac{m^2}{2^b} + \frac{1}{2^b - m} + \varepsilon_m^1 + \varepsilon_m^2. \tag{1}$$

In see this, compare $(C_k, \hat{C}_k)$ to the following process $(C', \hat{C}')$, which is defined in terms of *random* functions and permutations rather than pseudorandom functions and permutations. Let $E_1, E_2, \ldots, E_{m+2}$ be random permutations, and

let $D_1, D_2, \ldots, D_{m+2}$ be the corresponding inverse permutations. The process $(C', \hat{C}')$ acts as follows on $\mathcal{A}$'s i^{th} query (V_i, h_i), given that (V_1, h_1), (V_2, h_2), ..., (V_{i-1}, h_{i-1}) were $\mathcal{A}$'s $i-1$ previous queries. Suppose that h_i is the jth distinct element in the set $\{h_1, h_2, \ldots, h_i\}$. If (V_i, h_i) is a query is to C', then respond with $E_j(V_i)$, and if it is a query to $\hat{C}'$, then respond with $D_j(V_i)$.

We would like to bound the probability that, after m queries, $\mathcal{A}$ can find $(V, h) \neq (V', h')$ such that $C'(V, h) = C'(V', h')$ but at least one of (V, h) or (V', h') did not appear in the sequence (V_1, h_1), (V_2, h_2), ..., (V_m, h_m). Suppose that h is the j_1^{th} distinct element and h' is the j_2^{th} distinct element among $h_1, h_2, \ldots, h_m, h, h'$. If $j_1 = j_2$, we are done, because $E_{j_1}(V) \neq E_{j_1}(V')$. Otherwise, assume without loss of generality that it is the query (V, h) that did not appear in the sequence. Then the probability that $E_{j_1}(V) = E_{j_2}(V')$ is at most $1/(2^b - m)$, because the value of E_{j_1} has been specified on at most m points, and $E_{j_1}(V)$ is uniformly distributed among the remaining $2^b - m$ points in the range.

We now bound the probability that a polynomial-time adversary can distinguish between (truly random) $(C', \hat{C}')$ and (pseudorandom) $(C_k, \hat{C}_k)$. Essentially, we use a hybrid argument. If, instead of the pseudorandom F_k, a truly random function f were used, the probability that the adversary could find two different h and h' such that $f(h) = f(h')$ would be at most $m^2/2^b$. If this does not happen, then the randomness of f implies that the keys of the pseudorandom permutations are random; if the adversary could distinguish between such a process and $(C', \hat{C}')$, then it could distinguish between a collection of m pseudorandom permutations and a collection of m truly random permutations – this happens with probably at most ε_m^2. Distinguishing between the case in which a random f is used and the one in which a pseudorandom F_k is used adds probability at most ε_m^1, yielding (1). □

We provide another construction of non-colliding encryption . The proof is based on generating many "independent" permutations from a single one, as in Even and Mansour [9] and Kilian and Rogaway [14]. It may be the preferred construction if the smart-card constraints make it very difficult to change a key to a permutation.

NCE Construction 2: Let $k = (k_1, k_2, k_3)$, where k_2 and k_3 are used as keys to the pseudo-random function F and k_1 as a key to E. The idea is to apply F to h and obtain a "mask" for encrypting V. Formally, $C_k(V, h) = E_{k_1}(F_{k_2}(h) \oplus V) \oplus F_{k_3}(h)$, and $\hat{C}_k(V, h) = D_{k_1}(F_{k_3}(h) \oplus V) \oplus F_{k_2}(h)$.

Lemma 8. NCE Construction 2 is a non-colliding encryption scheme.

Proof. Even and Mansour showed that, if $E : \{0,1\}^b \mapsto \{0,1\}^b$ is a random permutation and m pairs $\langle (I_1, O_1), (I_2, O_2), \ldots, (I_n, O_n) \rangle$ are chosen independently and uniformly at random, then the permutations $E_i(X) = E(X \oplus I_i) \oplus O_i$ are indistinguishable from random. The pseudorandomness of F_{k_2} and F_{k_3} allows us to repeat the argument of Lemma 7. □

***J*-good sequences:**

Recall that we would like these to be the sequences in which different X^i's correspond to different W^i's and different Y^i's correspond to different Z^i's. Furthermore, the W's and Z's of the distinguishing phase should be different from those obtained during the host phase, except in those inputs filtered by the arbiter. Intuitively, these sequences are "good" for the pseudorandom permutation construction, because distinct W's produce distinct S's with overwhelming probability. The properties of the building blocks E, F, and G then ensure that there is a process $\tilde{J}$, indistinguishable from J, such that, for all J-good input-output sequences $\langle (X^1, Y^1), \ldots, (X^m, Y^m) \rangle$

$$\Pr_{\tilde{J}}[p_2(Y^i) = \tilde{J}(p_1(X^i)), 1 \leq i \leq m] \approx 2^{-\ell \cdot m}.$$

More precisely, let $\mathcal{A}$ be a probabilistic, polynomial-time adversary that has m_1 interactions with the card during the host phase and makes m_2 oracle queries during the distinguishing phase. The sequences we are interested in consist of

$$(X_1^1, h_x^1, Y_1^1, h_y^1), (X_1^2, h_x^2, Y_1^2, h_y^2), \ldots, (X_1^{m_1}, h_x^{m_1}, Y_1^{m_1}, h_y^{m_1})$$

from the host phase and

$$(X^{m_1+1}, Y^{m_1+1}), (X^{m_1+2}, Y^{m_1+2}), \ldots, (X^{m_1+m_2}, Y^{m_1+m_2})$$

from the distinguishing phase. For any such sequence, the permutations p_1 and p_2 determine W^1, ..., W^{m_1}, W^{m_1+1}, ..., $W^{m_1+m_2}$ and Z^1, ..., Z^{m_1}, Z^{m_1+1}, ..., $Z^{m_1+m_2}$. An encryption query $(+, X^{m_1+i})$ or decryption query $(-, Y^{m_1+i})$ is filtered during the distinguishing phase by $\mathcal{B}$ if there is a $1 \leq j \leq m_1$ for which

$$(X_1^{m_1+i}, H(X_2^{m_1+i}, \ldots, X_n^{m_1+i})) = (X_1^j, h_x^j)$$

(or analogously $(Y_1^{m_1+i}, H(Y_2^{m_1+i}, \ldots, Y_n^{m_1+i})) = (Y_1^j, h_y^j)$). Note that the adversary should not be able to find more than m_1 inputs and m_1 outputs that are filtered – otherwise, the pigeonhole principle implies that the adversary would have found in the distinguishing phase two encryption queries $(+, X^{m_1+j_1}) \neq (+, X^{m_1+j_2})$ such that $(X_1^{m_1+j_1}, h_x^{m_1+j_1}) = (X_1^{m_1+j_2}, h_x^{m_1+j_2}) = (X_i, h_x^i)$ for some $1 \leq i \leq m_1$ and $1 \leq j_1, j_2 \leq m_2$ (or two analogous decryption queries). However, that would mean that it had broken the collision-intractable hash function H.

We say that a sequence is J-**good for** p_1 **and** p_2 if

1. For all $1 \leq i < j \leq m_2$, if $X^{m_1+i} \neq X^{m_1+j}$ and X^{m_1+i} and X^{m_1+j} are not filtered by $\mathcal{B}$, then $W^{m_1+i} \neq W^{m_1+j}$, and, if $Y^{m_1+i} \neq Y^{m_1+j}$ and Y^{m_1+i} and Y^{m_1+j} are not filtered by $\mathcal{B}$, then $Z^{m_1+i} \neq Z^{m_1+j}$.
2. For all $1 \leq i \leq m_2$, if X^{m_1+i} is not filtered by $\mathcal{B}$, then $W^{m_1+i} \neq W^j$ for all $1 \leq j \leq m_1$, and, if Y^{m_1+i} is not filtered by $\mathcal{B}$, then $Z^{m_1+i} \neq Z^j$ for all $1 \leq j \leq m_1$.

In other words, the W's and Z's of the distinguishing phase are different from one another and from those of the host phase. We must show that the adversary is not able to find bad sequences, except with negligible probability.

Lemma 9. For any permutation J, for any probabilistic, polynomial-time adversary $\mathcal{A}$ that has m_1 interactions with the card during the host phase and makes m_2 oracle queries during the distinguishing phase, the probability that $\mathcal{A}$ finds a sequence that is not J-good for p_1 and p_2 is negligible. The probability is computed over the choice of p_1 and p_2 and the random coin-flips of $\mathcal{A}$. Note that J is not necessarily secret.

Proof. Let ε_1 be an upper bound on the probability that an adversary with $\mathcal{A}$'s resources breaks the collision-intractable hash function H, and let ε_2 be an upper bound on the probability that an adversary with $\mathcal{A}$'s resources breaks the non-colliding encryption scheme. Then $\mathcal{A}$'s probability of finding a sequence that is not J-good for p_1 and p_2 is upper-bounded by $\varepsilon_1 + \varepsilon_2$.

Suppose that the first query that witnesses the fact that this sequence is not J-good occurs at the jth step of the distinguishing phase, and assume without loss of generality that it is an encryption query. We divide this event into two cases. In case 1, Property 1 is violated, *i.e.*, there are i and j, $1 \leq i < j \leq m_2$, such that $X^{m_1+i} \neq X^{m_1+j}$ but $X_1^{m_1+i} = X_1^{m_1+j}$ and $H(X_2^{m_1+i}, \ldots, X_n^{m_1+i}) = H(X_2^{m_1+j}, \ldots, X_n^{m_1+j})$. This means that $\mathcal{A}$ has broken the collision-intractable function H, which happens with probability at most ε_1. The other possibility is that $Z^{m_1+j} = Z^i$ but $(X_1^{m_1+j}, h_x^{m_1+j}) \neq (X_1^i, h_x^i)$ where i $(1 \leq i \leq m_1 + j - 1)$ is either from the host phase or from the distinguishing phase. However, note that $(X_1^{m_1+j}, h_x^{m_1+j})$ did not appear explicitly before in the sequence (if it had, it would have been filtered, or j would not be the first "bad location" in the sequence), but $C_{k_1}(X_1^{m_1+j}, h_x^{m_1+j}) = C_{k_1}(X_1^i, h_x^i)$. Thus $\mathcal{A}$ could break the non-colliding encryption scheme $(C_k, \hat{C}_k)$, which happens with probability at most ε_2. □

Indistinguishability:

As in the original Naor-Reingold paper, we consider what happens when J is replaced with a "more random" process. Let $\tilde{J}_1$ be obtained from J by replacing E_{k_3} with a random permutation and replacing G_S with a process $\tilde{G}$ that, on input S, produces a random string of length $(n-1) \cdot b$ (*i.e.*, a random function $\{0,1\}^b \mapsto \{0,1\}^{(n-1)\cdot b}$) and Xors the string with $X_2, \ldots, X_n$. Let $\tilde{J}_2$ be a random permutation. Note that, if J is replaced with $\tilde{J}_2$, the composition $p_2^{-1} \circ \tilde{J}_2 \circ p_1$ is a random permutation for any p_1 and p_2.

We complete the proof of Theorem 6 by showing (i) when J is replaced with $\tilde{J}_1$, the result is indistinguishable by probabilistic, polynomial-time adversaries, and (ii) when $\tilde{J}_1$ is replaced by $\tilde{J}_2$ the result is indistinguishable to adversaries restricted to good sequences.

Lemma 10. Suppose that, following the host phase of an attack on Scheme P, a nondeterministic choice is made between replacing the function J by $\tilde{J}_1$ or continuing to use J. Then any probabilistic, polynomial-time adversary $\mathcal{A}$ has only a negligible probability of determining whether a switch was made, where the probability is over the choice of $J, p_1, p_2, \tilde{J}_1$ and $\mathcal{A}$'s coin-flips.

Proof. Observe first that J can be partly transformed without detection: Suppose that E_{k_3} is replaced with a random permutation and F_{k_4} is replaced with a random function prior to the beginning of the host phase. This should be indistinguishable to $\mathcal{A}$, because the only information $\mathcal{A}$ has about E_{k_3}, D_{k_3}, and F_{k_4} is their values (or some function of them) at some specific points. Therefore the important part of a potential switch is the replacement of G by $\tilde{G}$.

By Lemma 9, except with negligible probability, all the W's of the distinguishing phase are different from those of the host phase and different from each other, except those that were filtered. These W^i's are assigned a random value S^i. Recall that G has the property that, if S is chosen at random, then $G_S(X_2, \ldots, X_n)$ is indistinguishable from a truly random string of the same length for any $X_2, \ldots, X_n$, and similarly for $\hat{G}$. In case a switch is not made between phases, then G (or $\hat{G}$) produces an input that is indistinguishable from a truly random one. If a switch is made, then $\tilde{G}$ is used, and the result is a random and independent string (except when S_i is a collision, which happens with probability at most $m^2/2^b$). Therefore, the overall probability with which a switch is detected is negligible. □

Lemma 11. Suppose that, following the host phase of an attack on Scheme P, a nondeterministic choice is made between replacing J by $\tilde{J}_1$ or replacing J by $\tilde{J}_2$. Then any probabilistic, polynomial-time adversary $\mathcal{A}$ has a negligible probability of distinguishing between the two cases, where the probability is computed over the choice of $J, p_1, p_2, \tilde{J}_1, \tilde{J}_2$, and $\mathcal{A}$'s coin-flips.

Proof. Fix the adversary $\mathcal{A}$ to be the best deterministic machine, and fix p_1, p_2, and J. This determines the queries made during the host phase. Now consider any sequence

$$SEQ = \langle (X^{m_1+1}, Y^{m_1+1}), \ldots, (X^{m_1+m_2}, Y^{m_1+m_2}) \rangle$$

such that SEQ (together with the host phase queries) is J-good for p_1 and p_2. If SEQ is a *possible* outcome for $\mathcal{A}$, given the fixed J, p_1, and p_2, then

$$\Pr_{\tilde{J}_1}[SEQ \text{ is produced}] = \frac{1}{2^{bn}} \cdot \frac{1}{(2^b - 1) \cdot 2^{b(n-1)}} \cdots \frac{1}{(2^b - m_2 + 1) \cdot 2^{b(n-1)}}$$

and

$$\Pr_{\tilde{J}_2}[SEQ \text{ is produced}] = \frac{1}{2^{bn}} \cdot \frac{1}{2^{bn} - 1} \cdots \frac{1}{2^{bn} - m_2 + 1}.$$

Therefore, $\Pr_{\tilde{J}_1}[SEQ \text{ is produced}]$ is at least

$$\Pr_{\tilde{J}_2}[SEQ \text{ is produced}] \geq (1 - \frac{m_2}{2^b})^{m_2} \cdot \Pr_{\tilde{J}_1}[SEQ \text{ is produced}].$$

Now consider any collection $\mathcal{C}$ of sequences that are J-good for p_1 and p_2. The probabilities that a member of this collection is produced by $\tilde{J}_1$ and $\tilde{J}_2$ are close, because the ratio of $\Pr_{\tilde{J}_1}[\mathcal{C}]$ and $\Pr_{\tilde{J}_2}[\mathcal{C}]$ is between 1 and $1 - (m_2^2/2^b)$.

Suppose without loss of generality that, just before $\mathcal{A}$ has to guess whether it is querying process $\tilde{J}_1$ or process $\tilde{J}_2$, the permutations p_1 and p_2 are revealed. This can only help $\mathcal{A}$. If we consider all possible executions of $\mathcal{A}$, then with all but negligible probability (over p_1, p_2), the sequence generated is J-good for p_1 and p_2. Partition the executions that end with an J-good sequence into $\mathcal{C}_1$ (those for which $\mathcal{A}$ announces 1), and $\mathcal{C}_2$ (those where it announces 2). Summing over p_1, p_2, and J, the probabilities $\Pr_{\tilde{J}_1}[\mathcal{C}_1]$ and $\Pr_{\tilde{J}_2}[\mathcal{C}_1]$ are close (and similarly for $\Pr_{\tilde{J}_1}[\mathcal{C}_2]$ and $\Pr_{\tilde{J}_2}[\mathcal{C}_2]$). Therefore, we can conclude that $\tilde{J}_1$ and $\tilde{J}_2$ are indistinguishable for $\mathcal{A}$. □

To complete the proof of Theorem 6, note that $p_2^{-1} \circ \tilde{J}_2 \circ p_1$ is a random permutation. Thus, $\mathcal{A}$ has at most a negligible probability of determining whether J was switched with a random permutation between phases. □

6 Open Questions

Remaining questions include:

Question 1. The protocol in Section 5 requires two rounds of interaction between the host and the card. Is there a secure, length-preserving RKES that requires only one round of interaction?

Question 2. Is the existence of a one-way function sufficient for the construction of a provably secure RKES? Note that a collision-intractable hash function is used in our constructions and that it is not known how to build such a hash function based only on the assumption that a one-way function exists. (See [4] for a discussion of the desirability of using UOWHFs instead of collision-intractable hash functions.)

Acknowledgments

We thank Omer Reingold for useful discussions and the Eurocrypt '98 Program Committee members for their comments.

References

1. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A Concrete Security Treatment of Symmetric Encryption," in *Proceedings of the 38th Symposium on Foundation of Computer Science*, IEEE Computer Society Press, Los Alamitos, pp. 394–403, 1997.
2. M. Bellare, J. Kilian, and P. Rogaway, "The Security of Cipher Block Chaining," in *Advances in Cryptology – Crypto '94*, Lecture Notes in Computer Science, vol. 839, Springer, Berlin, pp. 341–358, 1994.
3. M. Bellare and P. Rogaway, "Provably Secure Session Key Distribution – The Three Party Case," in *Proceedings of the 27th Symposium on Theory of Computing*, ACM, New York, pp. 57–66, 1995.

4. M. Bellare and P. Rogaway, "Collision Resistant Hashing, Towards Making UOWHFs practical," in *Advances in Cryptology – Crypto '97*, Lecture Notes in Computer Science, vol. 1294, Springer, Berlin, pp. 470–484, 1997.
5. E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," in *Advances in Cryptology – Crypto '97*, Lecture Notes in Computer Science, vol. 1294, Springer, Berlin, pp. 513–525, 1997.
6. M. Blaze, "High-Bandwidth Encryption with Low-Bandwidth Smartcards," in *Proceedings of the Fast Software Encryption Workshop*, Lecture Notes in Computer Science, vol. 1039, Springer, Berlin, pp. 33–40, 1996.
7. D. Boneh, R. A. Demillo, and R. J. Lipton, "On the Importance of Checking Protocols for Faults," in *Advances in Cryptology – Eurocrypt '97*, Lecture Notes in Computer Science vol. 1233, Springer, Berlin, pp. 37–51, 1997.
8. D. Dolev, C. Dwork, and M. Naor, "Non-Malleable Cryptography," in *Proceedings of the 23rd Symposium on Theory of Computing*, ACM, New York, pp. 542–552, 1991. Expanded version available as Weizmann Institute Technical Report CS95-27, `http://www.wisdom.weizmann.ac.il/Papers/trs/CS95-27/`.
9. S. Even and Y. Mansour, "A construction of a cipher from a single pseudorandom permutation," *J. Cryptology*, 10 (1997), pp. 151–161.
10. J. Feigenbaum, "Locally Random Reductions in Interactive Complexity Theory," in *Advances in Computational Complexity Theory*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 13, American Mathematical Society, Providence, pp. 73–98, 1993.
11. O. Goldreich, **Foundations of Cryptography (Fragments of a Book)**, 1995. `http://www.eccc.uni-trier.de/eccc/info/ECCC-Books/eccc-books.html`
12. O. Goldreich S. Goldwasser, and S. Micali, "How to Construct Random Functions," *J. of the ACM*, 33 (1986), pp. 792-807.
13. S. Goldwasser and S. Micali, "Probabilistic Encryption," *J. Computer and System Sciences*, 28 (1984), pp. 270–299.
14. J. Kilian and P. Rogaway, "How to protect DES against exhaustive key search," in *Advances in Cryptology - CRYPTO '96*, Lecture Notes in Computer Science, vol. 1109, Springer, Berlin, pp. 252-267, 1996.
15. M. Luby, **Pseudorandomness and Cryptographic Applications**, Princeton University Press, Princeton, 1996.
16. S. Lucks, "On the Security of Remotely Keyed Encryption," in *Proceedings of the Fast Software Encryption Workshop*, Lecture Notes in Computer Science, vol. 1267, Springer, Berlin, pp. 219–229, 1997.
17. T. Matsumoto, K. Kato, and H. Imai, "Speeding Up Secret Computations with Insecure Auxiliary Devices," in *Advances in Cryptology – Crypto '88*, Lecture Notes in Computer Science, vol. 403, Springer, Berlin, pp. 497–506, 1990.
18. M. Naor and O. Reingold, "On the Construction of Pseudo-Random Permutations: Luby-Rackoff Revisited," to appear in *J. Cryptology*. Extended abstract appears in *Proceedings of the 29th Symposium on Theory of Computing*, ACM, New York, pp. 189–199, 1997.
19. V. Shoup and A. Rubin, "Session Key Distribution Using Smart Cards," in *Advances in Cryptology – Eurocrypt '96*, Lecture Notes in Computer Science vol. 1070, Springer, Berlin, pp. 321–331, 1996.