A preliminary version of this paper appeared in *Advances in Cryptology – Eurocrypt 94 Proceedings*, Lecture Notes in Computer Science Vol. 950, A. De Santis ed., Springer-Verlag, 1994.

Optimal Asymmetric Encryption — How to Encrypt with RSA

Mihir Bellare* Phillip Rogaway†

November 19, 1995

Abstract

Given an arbitrary k-bit to k-bit trapdoor permutation f and a hash function, we exhibit an encryption scheme for which (i) any string x of length slightly less than k bits can be encrypted as $f(r_x)$, where r_x is a simple probabilistic encoding of x depending on the hash function; and (ii) the scheme can be proven semantically secure assuming the hash function is "ideal." Moreover, a slightly enhanced scheme is shown to have the property that the adversary can create ciphertexts only of strings for which she "knows" the corresponding plaintexts—such a scheme is not only semantically secure but also non-malleable and secure against chosen-ciphertext attack.

* Department of Computer Science & Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093. E-mail: `mihir@cs.ucsd.edu`

† Department of Computer Science, University of California at Davis, Davis, CA 95616, USA. E-mail: `rogaway@cs.ucdavis.edu`

1 Introduction

Asymmetric (i.e. public key) encryption is a goal for which there is a large and widely-recognized gap between practical schemes and provably-secure ones: the practical methods are efficient but not well-founded, while the provably-secure schemes have more satisfying security properties but are not nearly as efficient.[1] The goal of this paper is to (nearly) have it all: to do asymmetric encryption in a way as efficient as any mechanism yet suggested, yet to achieve an assurance benefit almost as good as that obtained by provable security.

In the setup we consider a sender who holds a k-bit to k-bit trapdoor permutation f and wants to transmit a message x to a receiver who holds the inverse permutation f^{-1}. We concentrate on the case which arises most often in cryptographic practice, where $n = |x|$ is at least a little smaller than k.

What practioners want is the following: encryption should require just one computation of f; decryption should require just one computation of f^{-1}; the length of the enciphered text should be precisely k; and the length n of the text x that can be encrypted is close to k. Since heuristic schemes achieving these conditions exist [22, 15], if provable security is provided at the cost of violating any of these conditions (e.g., two applications of f to encrypt, message length $n + k$ rather than k) practioners will prefer the heuristic constructions. Thus to successfully impact practice one must provide provably-secure schemes which meet the above constraints.

The heuristic schemes invariably take the following form: one (probabilistically, invertibly) embeds x into a string r_x and then takes the encryption of x to be $f(r_x)$.[2] Let's call such a process a *simple-embedding scheme*. We will take as our goal to construct provably-good simple-embedding schemes which allow n to be close to k.

Assuming an ideal hash function and an arbitrary trapdoor permutation, we describe and prove secure two simple-embedding schemes that are bit-optimal (i.e., the length of the string x that can be encrypted by $f(r_x)$ is almost k). Our first scheme achieves semantic security [11], while our second scheme achieves a notion of plaintext-aware encryption, which we introduce here. This new notion is very strong, and in particular implies "ambitious" goals like chosen-ciphertext security and non-malleability [7] in the ideal-hash model which we assume.

The methods of this paper are simple and completely practical. They provide a good starting point for an asymmetric encryption/key distribution standard.

Next we describe our schemes and their properties. We refer the reader to Section 1.7 for discussion of previous work on encryption and comparisons with ours.

1.1 The basic scheme

Recall k is the security parameter, f mapping k-bits to k-bits is the trapdoor permutation. Let k_0 be chosen such that the adversary's running time is significantly smaller than 2^{k_0} steps. We fix the length of the message to encrypt as $n = k - k_0$ bits (shorter messages can be suitably padded to this length). The scheme makes use of a "generator" G: $\{0,1\}^{k_0} \to \{0,1\}^n$ and a "hash function"

[1] By a provably-secure scheme we mean here one shown, under some standard complexity-theoretic assumption, to achieve a notion of security at least as strong as semantic security [11].

[2] It is well-known that a naive embedding like $r_x = x$ is no good: besides the usual deficiencies of any deterministic encryption, f being a trapdoor permutation does not mean that $f(x)$ conceals all the interesting properties of x. Indeed it was exactly such considerations that helped inspire ideas like semantic security [11] and hardcore bits [5, 26].

$H\colon \{0,1\}^n \to \{0,1\}^{k_0}$. To encrypt $x \in \{0,1\}^n$ choose a random k_0-bit r and set

$$\mathcal{E}^{G,H}(x) \quad = \quad f(x \oplus G(r) \parallel r \oplus H(x \oplus G(r))).$$

Here "$\|$" denotes concatenation. The decryption function $\mathcal{D}^{G,H}$ is defined in the obvious way, and the pair $(\mathcal{E}, \mathcal{D})$ constitutes what we call the "basic" scheme.

We prove security under the assumption that G, H are "ideal." This means G is a random function of $\{0,1\}^{k_0}$ to $\{0,1\}^n$ and H is a random function of $\{0,1\}^n \to \{0,1\}^{k_0}$. The formal statement of our result is in Theorem 4.1. It says that if f is a trapdoor permutation and G, H are ideal then the basic scheme achieves the notion of semantic security [11] appropriately adjusted to take account of the presence of G, H.

In practice, G and H are best derived from some standard cryptographic hash function. (For example, they can be derived from the compression function of the Secure Hash Algorithm [18] following the methods described in [2]).

1.2 The plaintext-aware scheme

A variety of goals for encryption have come to be known which are actually stronger than the notion of [11]. These include non-malleability [7] and chosen ciphertext security. We introduce a new notion of an encryption scheme being *plaintext-aware*—roughly said, it should be impossible for a party to produce a valid ciphertext without "knowing" the corresponding plaintext (see Section 3 for a precise definition). In the ideal-hash model that we assume, this notion can be shown to imply non-malleability and chosen-ciphertext security.

We construct a plaintext-aware encryption scheme by slightly modifying the basic scheme. Let k and k_0 be as before and let k_1 be another parameter. This time let $n = k - k_0 - k_1$. Let the generator be $G\colon \{0,1\}^{k_0} \to \{0,1\}^{n+k_1}$ and the hash function $H\colon \{0,1\}^{n+k_1} \to \{0,1\}^{k_0}$. To encrypt, choose a random k_0-bit r and set

$$\mathcal{E}^{G,H}(x) \quad = \quad f(x0^{k_1} \oplus G(r) \parallel r \oplus H(x0^{k_1} \oplus G(r))).$$

The decryption $\mathcal{D}^{G,H}$ is defined in the obvious way and the pair constitutes the scheme we call "plaintext-aware."

The formal statement of our results are in Theorems 6.1 and 6.2. They say that if f is a trapdoor permutation and G, H are ideal then the plaintext-aware scheme is a semantically secure, plaintext-aware encryption. In practice, again, G and H are derived from some standard cryptographic hash function.

1.3 Efficiency

The function f can be set to any candidate trapdoor permutation such as RSA [21] or modular squaring [19, 3]. In such a case the time for computing G and H is negligible compared to the time for computing f, f^{-1}. Thus complexity is discussed only in terms of f, f^{-1} computations. In this light our basic encryption scheme requires just a single application of f to encrypt, a single application of f^{-1} to decrypt, and the length of the ciphertext is k (as long as $k \geq n + k_0$). Our plaintext-aware scheme requires a single application of f to encrypt, a single application of f^{-1} to decrypt, and the length of the ciphertext is still k (as long as $k \geq n + k_0 + k_1$).

A concrete instantiation of our plaintext-aware scheme (using RSA for f and getting G, H from the Secure Hash Algorithm [18]) is given in Section 7.

1.4 The ideal hash function paradigm

As we indicated above, when proving security we take G, H to be random, and when we want a concrete scheme, G, H are instantiated by primitives derived from a cryptographic hash function. In this regard we are following the paradigm of [2] who argue that even though results which assume an ideal hash function do not provide provable security with respect to the standard model of computation, assuming an ideal hash function and doing proofs with respect to it provides much greater assurance benefit than purely *ad. hoc.* protocol design. We refer the reader to that paper for further discussion of the meaningfulness, motivation and history of this ideal hash approach.

1.5 Exact security

We want our results to be meaningful for practice. In particular, this means we should be able to say meaningful things about the security of our schemes for specific values of the security parameter (e.g., $k = 512$). This demands not only that we avoid asymptotics and address security "exactly," but also that we strive for security reductions which are as efficient as possible.[3]

Thus the theorem proving the security of our basic scheme quantifies the resources and success probability of a potential adversary: let her run for time t, make q_{gen} queries of G and q_{hash} queries of H, and suppose she could "break" the encryption with advantage ϵ. It then provides an algorithm M and numbers t', ϵ' such that M inverts the underlying trapdoor permutation f in time t' with probability ϵ'. The strength of the result is in the values of t', ϵ' which are specified as functions of $t, q_{\text{gen}}, q_{\text{hash}}, \epsilon$ and the underlying scheme parameters k, k_0, n ($k = k_0 + n$). Now a user with some idea of the (assumed) strength of a particular f (e.g., RSA on 512 bits) can get an idea of the resources necessary to break our encryption scheme.

1.6 Extensions

The assumption that $n = |x| \leq k - k_0 - k_1$ can be removed while retaining the bit optimality of the scheme: the ideas presented here can be extended to design an authenticated encryption scheme (provably secure in the ideal-hash model assuming an arbitrary trapdoor permutation) where encryption still requires one application of f on a k-bit input; decryption still requires one application of f^{-1} on a k-bit input; and now the length of the encrypted text will be $\max\{k, |x| + k_0 + k_1\}$.

1.7 Prior work in encryption

We briefly survey relevant prior art in encryption. In the following, f mapping k-bits to k-bits is the trapdoor permutation. As above, the following assumes the length n of the message to be encrypted is at most k. We begin by discussing work on attaining semantic security, and then move on to stronger goals.

Goldwasser and Micali [11] first suggested encrypting a message by probabilistically encrypting each of its bits: if B_f denotes a hard core predicate [5, 26, 10] for the trapdoor permutation f, then

[3] Exact security is not new: previous works which address it explicitly include [10, 14, 23, 16, 8, 1]. Moreover, although it is true that most theoretical works only provide asymptotic security guarantees of the form "the success probability of a polynomially bounded adversary is negligible" (everything measured as a function of the security parameter), the exact security can be derived from examination of the proof. (However, a lack of concern with the exactness means that in many cases the reductions are very inefficient, and the results are not useful for practice).

the encryption of $x = x_1 \ldots x_n$ is $\mathcal{E}_{\mathrm{GM}}(x) = f(r_1) \| \ldots \| f(r_n)$, where each r_i is randomly chosen from the domain of f subject to $B_f(r_i) = x_i$. This yields an encryption of length $O(nk)$ which requires n evaluations of f to encrypt and n evaluation of f^{-1} to decrypt, which is not practical.

The more efficient construction of Blum and Goldwasser [4] is based on the particular choice of f as the modular squaring function [19]. They achieve encryption size $n + k$. They require $O(nk^2/\log k)$ steps to encrypt and $O(k^3)$ steps to decrypt. The encryption is longer than ours by n bits. To compare the time complexities, take the function f in our scheme to also be squaring. Then their encryption time is a factor $O(n/\log k)$ more than ours. Their decryption time is a constant factor more than ours.

Of course the above two schemes have the advantage of being based only on standard assumptions, not the use of an ideal hash function.

The discrete log function simultaneously hides a constant fraction of the bits of its pre-image [24]. But it is not known to have a trapdoor and hence is not usable for the problem we are considering.

What we have called simple-embedding schemes are prevalent in computing practice. One example is the RSA Public Key Cryptography Standard #1 [22], where r_x in the embedding $x \mapsto r_x$ is essentially x in the low-order bit positions and a string of random non-zero bytes in the remaining bit positions. Another scheme is described in [15]; a simplified version of it is

$$\mathcal{E}^G_{\mathrm{IBM}}(x) \quad = \quad f((x0^{k_2} \oplus G(r)) \parallel r)\ .$$

Of concern with both of these schemes is that there is no compelling reason to believe that x is as hard to compute from $f(r_x)$ as r_x is hard to compute from $f(r_x)$—let alone that all interesting properties of x are well-hidden by $f(r_x)$. Indeed whether or not [22, 15] "work" depends on aspects of f beyond its being one-way, insofar as it is easy to show that if there exists a trapdoor permutation then there exists one for which encryption as above is completely insecure.[4]

In [2] we suggested the scheme

$$\mathcal{E}^G_{\mathrm{BR}}(x) \quad = \quad f(r) \parallel G(r) \oplus x\ .$$

and proved it semantically secure in the same ideal-hash model used here. In comparison with the schemes given here, the drawback is that the encryption size is $n + k$ rather than k.

Now we turn to stronger goals. Chosen-ciphertext security was provably achieved by [17], but the scheme is extremely inefficient. More practical encryption schemes which aimed at achieving chosen ciphertext security were proposed by Damgård [6] and Zheng and Seberry [27]. The latter scheme is

$$\mathcal{E}^{G,H}_{\mathrm{ZS}}(x) \quad = \quad f(r) \parallel (G(r) \oplus (x \parallel H(x))\ ,$$

matching our plaintext-aware scheme in computation but having bit complexity $n + k + k_1$. Non-malleability is provably achieved by [7], but the scheme is extremely inefficient. An efficient scheme proven in [2] to achieve both non-malleability and chosen-ciphertext security under the ideal-hash model is

$$\overline{\mathcal{E}}^{G,H}_{\mathrm{BR}}(x) \quad = \quad f(r) \parallel G(r) \oplus x \parallel H(rx)\ .$$

Again the drawback is a bit complexity of $n + k + k_1$.

[4]But f is mandated to be RSA in both of [22, 15].

2 Preliminaries

2.1 Probabilistic algorithms

We shall use notation of [13]. If A is a probabilistic algorithm then $A(x, y, \cdots)$ refers to to the probability space which to the string σ assigns the probability that A on inputs $x, y, \cdots$ outputs σ. If S is a probability space we denote its support (the set of elements of positive probability) by $[S]$. When S is a probability space, $x \leftarrow S$ denotes selecting a random sample from S. We use $x, y \leftarrow S$ as shorthand for $x \leftarrow S;\ y \leftarrow S$. For probability spaces $S, T, \ldots$, the notation $\Pr[\, x \leftarrow S;\ y \leftarrow T;\ \cdots : p(x, y, \cdots)\,]$ denotes the probability that the predicate $p(x, y, \cdots)$ is true after the (ordered) execution of the algorithms $x \leftarrow S$, $y \leftarrow T$, etc.. PPT is short for "probabilistic, polynomial time."

In evaluating the complexity of oracle machines we adopt the usual convention that all oracle queries receive their answer in unit time.

2.2 Random oracles

We will be discussing schemes which use functions G, H chosen at random from appropriate spaces (the input and output lengths for G and H depend on parameters of the scheme). When stating definitions it is convenient to not have to worry about exactly what these spaces may be and just write $G, H \leftarrow \Omega$, the latter being defined as the set of all maps from the set $\{0,1\}^*$ of finite strings to the set $\{0,1\}^\infty$ of infinite strings. The notation should be interpreted as appropriate to the context—for example, if the scheme says G maps $\{0,1\}^a$ to $\{0,1\}^b$ then we can interpret $G \leftarrow \Omega$ as meaning we choose G from Ω at random, restrict the domain to $\{0,1\}^a$, and drop all but the first b bits of output.

2.3 Trapdoor permutations and their security

Our encryption schemes require a *trapdoor permutation generator.* This is a PPT algorithm $\mathcal{F}$ such that $\mathcal{F}(1^k)$ outputs a pair of deterministic algorithms (f, f^{-1}) specifying a permutation and its inverse on $\{0,1\}^k$.

We associate to $\mathcal{F}$ an *evaluation time* $T_{\mathcal{F}}(\cdot)$: for all k, all $(f, f^{-1}) \in [\mathcal{F}(1^k)]$ and all $w \in \{0,1\}^k$, the time to compute $f(w)$ (given f and w) is $T_{\mathcal{F}}(k)$. Note the evaluation time depends on the setting: for example on whether or not there is hardware available to compute f.

We will be interested in two attributes of a (possibly non-uniform) algorithm M trying to invert $\mathcal{F}(1^k)$-distributed permutations; namely its running time and its success probability.

Definition 2.1 Let $\mathcal{F}$ be a trapdoor permutation generator. We say that algorithm M succeeds in (t, ϵ)-inverting $\mathcal{F}(1^k)$ if

$$\Pr[\, (f, f^{-1}) \leftarrow \mathcal{F}(1^k);\ w \leftarrow \{0,1\}^k;\ y \leftarrow f(w) :\ M(f, y) = w \,] \geq \epsilon\ ,$$

and, moreover, in the experiment above, M runs in at most t steps.

RSA [21] is a good candidate function as a secure trapdoor permutation.[5]

[5] Candidates like RSA [21] don't quite fit our definition, in that the domain of RSA is some Z_N^*, a proper subset of of $\{0,1\}^k$. Things can be patched in standard ways.

3 Semantically Secure Encryption

We extend the definition of semantic security [11] to the random oracle model in a way which enables us to discuss exact security.

3.1 Encryption schemes

An asymmetric (i.e. public key) encryption scheme is specified by a probabilistic *generator,* $\mathcal{G}$, and an associated *plaintext-length function,* $n(\cdot)$. On input 1^k, the generator $\mathcal{G}$ outputs a pair of algorithms $(\mathcal{E}, \mathcal{D})$, the first of which is probabilistic. Each of these algorithms has oracle-access to two functions, one called G and one called H. A user i runs $\mathcal{G}$ to get $(\mathcal{E}, \mathcal{D})$ and makes the former public while keeping the latter secret. To encrypt message $x \in \{0,1\}^{n(k)}$ using functions G, H, anyone can compute $y \leftarrow \mathcal{E}^{G,H}(x)$ and send it to i. To decrypt ciphertext y user i computes $x \leftarrow \mathcal{D}^{G,H}(y)$. We require $\mathcal{D}^{G,H}(y) = x$ for all $y \in [\mathcal{E}^{G,H}(x)]$. We further demand that $\mathcal{D}^{G,H}(y) = *$ if there is no x such that $y \in [\mathcal{E}^{G,H}(x)]$.

An *adversary* is a (possibly nonuniform) algorithm A with access to oracles G, H. We assume without loss of generality that an adversary makes no particular G-query more than once and no particular H-query more than once. For simplicity we assume that the number of G-queries and H-queries that an adversary makes don't depend on its coin tosses but only, say, on the length of its input.

3.2 Semantic security

The following definition will be used to discuss (exact) security. It captures the notion of semantic security [11] appropriately lifted to take into account the presence of G, H.

We consider an adversary who runs in two stages. In the find-stage it is given an encryption algorithm $\mathcal{E}$ and outputs a pair x_0, x_1 of messages. It also outputs a string c which could record, for example, its history and its inputs. Now we pick at random either x_0 or x_1 (the choice made according to a bit b) and encrypt it (under $\mathcal{E}$) to get y. In the guess-stage we provide A the output x_0, x_1, c of the previous stage, and y, and we ask it to guess b. (We assume wlog that $\mathcal{E}$ is included in c so that we don't need to explicitly provide it again.) Since even the algorithm which always outputs a fixed bit will be right half of the time, we measure how well A is doing by 1/2 less than the fraction of time that A correctly predicts b. We call twice this quantity the *advantage* which A has in predicting b. Multiplying by two makes the advantage fall in the range [0, 1] (0 for a worthless prediction and 1 for an always correct one), instead of [0, 0.5].

Definition 3.1 Let $\mathcal{G}$ be a generator for an encryption scheme having plaintext-length function $n(\cdot)$. An adversary A is said to succeed in $(t, q_{\rm gen}, q_{\rm hash}, \epsilon)$-$\mathsf{breaking}$ $\mathcal{G}(1^k)$ if

$$\begin{aligned} \epsilon \quad \leq \quad & 2 \cdot \Pr[\, (\mathcal{E}, \mathcal{D}) \leftarrow \mathcal{G}(1^k);\ G, H \leftarrow \Omega;\ (x_0, x_1, c) \leftarrow A^{G,H}(\mathcal{E}, \mathsf{find}); \\ & b \leftarrow \{0,1\};\ y \leftarrow \mathcal{E}^{G,H}(x_b) :\ A^{G,H}(y, x_0, x_1, c) = b \,] \ - \ 1 \ , \end{aligned}$$

and, moreover, in the experiment above, A runs for at most t steps, makes at most $q_{\rm gen}$ queries to G, and makes at most $q_{\rm hash}$ queries to H.

Note that t is the total running time; ie. the sum of the times in the two stages. Similarly $q_{\rm gen}, q_{\rm hash}$ are the total number of G and H queries, respectively.

4 The Basic Encryption Scheme

Let $\mathcal{F}$ be a trapdoor permutation generator and $k_0(\cdot)$ a positive integer valued function such that $k_0(k) < k$ for all $k \geq 1$. The *basic scheme* $\mathcal{G}$ *with parameters* $\mathcal{F}$ *and* $k_0(\cdot)$ has an associated plaintext-length function of $n(k) = k - k_0(k)$. On input 1^k, the generator $\mathcal{G}$ runs $\mathcal{F}(1^k)$ to obtain (f, f^{-1}). Then it outputs the pair of algorithms $(\mathcal{E}, \mathcal{D})$ determined as follows:

(1) On input x of length $n = n(k)$, algorithm $\mathcal{E}$ selects a random r of length $k_0 = k_0(k)$. It sets $s = x \oplus G(r)$ and $t = r \oplus H(s)$. It sets $w = s \parallel t$ and returns $y = f(w)$.

(2) On input y of length k, algorithm $\mathcal{D}$ computes $w = f^{-1}(y)$. Then it sets s to the first n bits of w and t to the last k_0 bits of w. It sets $r = t \oplus H(s)$, and returns the string $x = s \oplus G(r)$.

The oracles G and H which $\mathcal{E}$ and $\mathcal{D}$ reference above have input/output lengths of $G : \{0,1\}^{k_0} \to \{0,1\}^n$ and $H : \{0,1\}^n \to \{0,1\}^{k_0}$. We use the encoding of f as the encoding of $\mathcal{E}$ and the encoding of f^{-1} as the encoding of $\mathcal{D}$.

The intuition behind the (semantic) security of this scheme is as follows. We wish to guarantee that the adversary, given a point y in the range of f, must recover the complete preimage $w = r_x$ of y if she is to say anything meaningful about x itself. Well, if the adversary does not recover all of the first n bits of the preimage, s, then she will have no idea about the value $H(s)$ which is its hash; a failure to know anything about $H(s)$ implies a failure to know anything about $r = H(s) \oplus t$ (where t is the last k_0 bits of w), and therefore $G(r)$, and therefore $x = G(r) \oplus s$ itself. Now, assuming the adversary does recover s, a failure to completely recover t will again mean that the adversary fails to completely recover r, and, in the lack of complete knowledge about r, $x \oplus G(r)$ is uniformly distributed and so again the adversary can know nothing about x.

Yet the above discussion masks some subtleties and a formal proof of security is more complex than it might appear. This is particularly the case when one is interested, as we are here, in achieving the best possible exact security.

The following theorem says that if there is an adversary A who is able to break the encryption scheme with some success probability, then there is an algorithm M which can invert the underlying trapdoor permutation with comparable success probability and in comparable time. This implies that if the trapdoor permutations can't be inverted in reasonable time (which is the implicit assumption) then our scheme is secure. But the theorem says more: it specifies exactly how the resources and success of M relate to those of A and to the underlying scheme parameters k, n, k_0 ($k = n + k_0$).

The inverting algorithm M can by obtained from A in a "uniform" way; the theorem says there is a "universal" oracle machine U such that M can be implemented by U with oracle access to A. It is important for practice that the "description" of U is "small;" this is not made explicit in the theorem but is clear from the proof. The constant λ depends only on details of the underlying model of computation. We write n, k_0 for $n(k), k_0(k)$, respectively, when, as below, k is understood.

Theorem 4.1 Let $\mathcal{G}$ be the basic encryption scheme with parameters $\mathcal{F}$, k_0 and let n be the associated plaintext length. Then there exists an oracle machine U and a constant λ such that for each integer k the following is true. Suppose A succeeds in $(t, q_{\text{gen}}, q_{\text{hash}}, \epsilon)$-breaking $\mathcal{G}(1^k)$. Then $M = U^A$ succeeds in (t', ϵ')-inverting $\mathcal{F}(1^k)$, where

$$\begin{aligned} t' &= t + q_{\text{gen}} \cdot q_{\text{hash}} \cdot (T_{\mathcal{F}}(k) + \lambda k) \\ \epsilon' &= \epsilon \cdot (1 - q_{\text{gen}} 2^{-k_0} - q_{\text{hash}} 2^{-n}) \; - \; q_{\text{gen}} 2^{-k+1} \; . \end{aligned}$$

The proof of Theorem 4.1 is in Appendix A.

For reasonable values of k (e.g., $k \geq 512$) it will be the case that $k > n >> k_0$. Thus for reasonable values of $q_{\text{gen}}, q_{\text{hash}}$ we'll have $\epsilon' \approx \epsilon \cdot (1 - q_{\text{gen}} 2^{-k_0})$. Thus the success probability ϵ' achieved here is good in the sense that it is only slightly less than ϵ and close to optimal. Note also that the expression for ϵ' indicates that A will do best by favoring G-oracle queries over H-oracle queries.

The dominant factor in the time t' taken by the inverting algorithm to compute $f^{-1}(y)$ is the time to do $q_{\text{gen}} \cdot q_{\text{hash}}$ computations of the underlying f. An interesting open question is to find a scheme under which the number of computation of f is linear in $q_{\text{gen}} + q_{\text{hash}}$ while retaining a value of ϵ' similar to ours.

5 Plaintext-Aware Encryption

We introduce a new notion of an encryption being "plaintext aware." The idea is that an adversary is "aware" of the decryption of the messages which she encrypts in the sense that she cannot produce a ciphertext y without "knowing" the corresponding plaintext. In formalizing this we have relied on definitional ideas which begin with [12, 9, 25]. Our notion requires that some (universal) algorithm K (the "knowledge extractor") can usually decrypt whatever ciphertext an adversary B may output, just by watching the G, H-queries which B makes.

Let B be an adversary which given an encryption algorithm $\mathcal{E}$ outputs a string y (intuitively, the ciphertext). The notation $(y, \tau) \leftarrow \mathsf{run}\, B^{G,H}(\mathcal{E})$ means the following. We run the algorithm $B^{G,H}(\mathcal{E})$ which outputs y. We record in the process the transcripts of its interaction with its oracles. Thus there is a list τ_{gen} which for each G-oracle query g made by B records g and the answer $G(g)$; similarly for H:

$$\begin{aligned} \tau_{\text{gen}} &= (g_1, G(g_1)), \ldots, (g_{q_{\text{gen}}}, G(g_{q_{\text{gen}}})) \\ \tau_{\text{hash}} &= (h_1, H(h_1)), \ldots, (h_{q_{\text{hash}}}, H(h_{q_{\text{hash}}})) \,. \end{aligned}$$

The pair $(\tau_{\text{gen}}, \tau_{\text{hash}})$ constitutes τ.

Definition 5.1 Let $\mathcal{G}$ be a generator for an encryption scheme and let B be an adversary that outputs a string. An algorithm K is said to be a (t, ϵ)-$\mathsf{plaintext\ extractor}$ for $B, \mathcal{G}(1^k)$ if

$$\Pr[\, (\mathcal{E}, \mathcal{D}) \leftarrow \mathcal{G};\; G, H \leftarrow \Omega;\; (y, \tau) \leftarrow \mathsf{run}\, B^{G,H}(\mathcal{E}) :\; K(\mathcal{E}, y, \tau) \neq \mathcal{D}^{G,H}(y) \,] \;\leq\; \epsilon,$$

and K runs in at most t steps in the experiment above.

The information we provide K about B is only B's output y and the transcript of her oracle interactions τ. We could more generally also provide B's coin tosses; we omit to do this only because the stronger notion we define above is achieved by our scheme.

Note we don't give K oracle access to G, H: it is required to find the plaintext corresponding to y given only B's "view" of the oracle. The rest is random anyway so it makes no difference.

A complexity-theoretic notion for a plaintext-aware encryption can be easily created out of the exact definition given above. Also, a definition for the standard (random oracle devoid) model is easily obtained. But in this case, we would definitely allow K access to B's coin tosses.

As previously mentioned, demanding awareness of a secure encryption scheme is asking a lot. In the random oracle model, we can show that a plaintext-aware scheme is non-malleable and also secure against chosen-ciphertext attack. We omit proofs of this, but the intuition is quite clear. For example, a chosen-ciphertext attack will not help because the adversary already "knows" the plaintext of any ciphertext y whose decryption she might request from an available decryption box.

6 The Plaintext-Aware Encryption Scheme

Let $\mathcal{F}$ be a trapdoor permutation generator. Let $k_0(\cdot)$ and $k_1(\cdot)$ be positive integer valued functions such that $k_0(k) + k_1(k) < k$ for all $k \geq 1$. The *plaintext-aware scheme* $\mathcal{G}$ *with parameters* $\mathcal{F}, k_0, k_1$ has an associated plaintext-length function of $n(k) = k - k_0(k) - k_1(k)$. On input 1^k, the generator $\mathcal{G}$ runs $\mathcal{F}(1^k)$ to obtain (f, f^{-1}). Then it outputs the pair of algorithms $(\mathcal{E}, \mathcal{D})$ determined as follows:

(1) On input x of length $n = n(k)$, algorithm $\mathcal{E}$ selects a random r of length $k_0 = k_0(k)$. It sets $s = x0^{k_1} \oplus G(r)$ and $t = r \oplus H(s)$. It sets $w = s \parallel t$ and returns $y = f(w)$.

(2) On input y of length k, algorithm $\mathcal{D}$ computes $w = f^{-1}(y)$. Then it sets s to the first $n + k_1$ bits of w and t to the last k_0 bits of w. It sets $r = t \oplus H(s)$. It sets x to the first n bits of $s \oplus G(r)$ and z to the last k_1 bits of $s \oplus G(r)$. If $z = 0^{k_1}$ then it returns x, else it returns $*$.

The oracles G and H which $\mathcal{E}$ and $\mathcal{D}$ reference above have input/output lengths of G: $\{0,1\}^{k_0} \to \{0,1\}^{n+k_1}$ and H: $\{0,1\}^{n+k_1} \to \{0,1\}^{k_0}$.

The semantic security of this scheme as given by the following theorem is a consequence of Theorem 4.1.

Theorem 6.1 Let $\mathcal{G}$ be the plaintext-aware encryption scheme with parameters $\mathcal{F}$, k_0, k_1 and let n be the associated plaintext length. Then there exists an oracle machine U and a constant λ such that for each integer k the following is true. Suppose A succeeds in $(t, q_{\text{gen}}, q_{\text{hash}}, \epsilon)$-breaking $\mathcal{G}(1^k)$. Then $M = U^A$ succeeds in $(t'\epsilon')$-inverting $\mathcal{F}$, where

$$\begin{aligned} t' &= t + q_{\text{gen}} \cdot q_{\text{hash}} \cdot (T_{\mathcal{F}}(k) + \lambda k) \\ \epsilon' &= \epsilon \cdot (1 - q_{\text{gen}} 2^{-k_0} - q_{\text{hash}} 2^{-n-k_1}) - q_{\text{gen}} 2^{-k+1} . \end{aligned}$$

Proof: Let $\mathcal{G}'$ be the generator for the basic scheme with parameters $\mathcal{F}$ and k_0— the associated plaintext-length function is $n'(k) = k - k_0(k) = n(k) + k_1(k)$. Let A' be the adversary for $\mathcal{G}'$ who (i) in the find-stage runs A to get (x_0, x_1, c) and outputs $(x_0 0^{k_1}, x_1 0^{k_1}, c)$; and (ii) in the guess-stage removes the padded zeroes from the messages and runs A. Now apply Theorem 4.1 to A'. ■

The intuition for the plaintext awareness of our encryption scheme can be described as follows. Let y be the string output by B. If she hasn't asked $G(r)$, then almost certainly the first $n + k_1$ bits of the preimage of y won't end with the right substring 0^{k_1}; and if she hasn't asked $H(s)$, then she can't know r; but if the adversary does know s, then certainly she knows its first n bits, which is x.

To discuss exact security it is convenient to say that adversary $B(\cdot)$ is a $(t, q_{\text{gen}}, q_{\text{hash}})$-*adversary for* $\mathcal{G}(1^k)$ if for all $(\mathcal{E}, \mathcal{D}) \in [\mathcal{G}(1^k)]$, $B(\mathcal{E})$ runs in at most t steps, makes q_{gen} G-queries and makes q_{hash} H-queries.

Theorem 6.2 Let $\mathcal{G}$ be the plaintext-aware encryption scheme with parameters $\mathcal{F}$, k_0, k_1 and let n be the associated plaintext length. Then there exists an oracle machine K and a constant λ such that for each integer k the following is true. Suppose B is a $(t, q_{\rm gen}, q_{\rm hash})$-adversary for $\mathcal{G}(1^k)$. Then $K = U^B$ is a (t', ϵ')-plaintext extractor for $B, \mathcal{G}$, where

$$\begin{aligned} t' &= t + q_{\rm gen} \cdot q_{\rm hash} \cdot (T_{\mathcal{F}}(k) + \lambda k) \\ \epsilon' &= q_{\rm gen} 2^{-k_0} + 2^{-k_1} . \end{aligned}$$

As before, one interesting open question is to device a scheme with t' linear in $q_{\rm gen} + q_{\rm hash}$ rather than quadratic. Another nice open question is whether one can achieve plaintext-aware encryption in the standard (random oracle devoid) model given a standard complexity theoretic assumption.

7 Sample RSA-Based Instantiation

We provide here a concrete instantiation of our plaintext-aware encryption scheme (omitting only certain minor details). We use RSA as the trapdoor permutation and construct the functions G, H out of the (revised) NIST Secure Hash Algorithm [18]. (Other hash algorithms such as MD5 [20] would do as well).

Let f be the RSA function [21], so $f(x) = x^e \bmod N$ is specified by (e, N) where N is the k-bit product of two large primes and $(e, \varphi(N)) = 1$. We demand $k \geq 512$ bits (larger values are recommended). Our scheme will allow the encryption of any string *msg* whose length is at most $k - 320$ bits (thus the minimal permitted security parameter allows 192 bits (e.g., three 64-bit keys) to be encrypted.) Let $D = \{1 \leq i < N : \gcd(i, N) = 1\} \subseteq \{0,1\}^k$ be the set of valid domain points for f.

Our probabilistic encryption scheme depends on the message *msg* to encrypt, an arbitrary-length string *rand_coins*, the security parameter k, the function f, and a predicate $\textsc{ind}(x)$ which should return **true** if and only if $x \in D$. Our scheme further uses a 32-bit string *key_data* (whose use we do not specify here), and a string *desc* which provides a complete description of the function f (i.e., it says "This is RSA using N and e") encoded according to conventions not specified here.

We denote by $\mathrm{SHA}_\sigma(x)$ the 160-bit result of SHA (Secure Hash Algorithm) applied to x, except that the 160-bit "starting value" in the algorithm description is taken to be $ABCDE = \sigma$. Let $\mathrm{SHA}^\ell_\sigma(x)$ denote the first ℓ-bits of $\mathrm{SHA}_\sigma(x)$. Fix the notation $\langle\, i \,\rangle$ for i encoded as a binary 32-bit word. We define the function $H^\ell_\sigma(x)$ for string x, number ℓ, and 160-bit σ to be the ℓ-bit prefix of

$$\mathrm{SHA}^{80}_\sigma(\langle\, 0 \,\rangle.x) \;\|\; \mathrm{SHA}^{80}_\sigma(\langle\, 1 \,\rangle.x) \;\|\; \mathrm{SHA}^{80}_\sigma(\langle\, 2 \,\rangle.x) \;\|\; \cdots$$

Let K_0 be a fixed, randomly-chosen 160-bit string (which we do not specify here).

Our scheme is depicted in Figure 7. Basically, we augment the string *msg* which we want to encrypt by tacking on a word to indicate its length; including $k_1 = 128$ bits of redundancy; incorporating a 32-bit field *key_data* whose use we do not specify; and adding enough additional padding to fill out the length of the string we have made to $k - 128$ bits. The resulting string x now plays the same role as the x of our basic scheme, and a separate 128-bit r is then used to encrypt it.

We comment that in the concrete scheme shown in Figure 7 we have elected to make our generator and hash function sensitive both to our scheme itself (via K_0) and to the particular

$\textsc{Encrypt}$ (*msg*, *rand_coins*)

$$
\begin{array}{l}
\quad \sigma = \mathrm{SHA}_{K_0}(desc); \\
\quad \sigma_1 = \mathrm{SHA}_\sigma(\langle\, 1 \,\rangle); \\
\quad \sigma_2 = \mathrm{SHA}_\sigma(\langle\, 2 \,\rangle); \\
\quad \sigma_3 = \mathrm{SHA}_\sigma(\langle\, 3 \,\rangle); \\
\quad i \leftarrow 0; \\
\quad \textbf{repeat} \\
\qquad r \leftarrow H^{128}_{\sigma_1}(\langle\, i \,\rangle \;\|\; rand\_coins); \\
\qquad x \leftarrow key\_data \;\|\; \langle\, |msg| \,\rangle \;\|\; 0^{128} \;\|\; 0^{k-320-|msg|} \;\|\; msg; \\
\qquad \overline{x} \leftarrow x \oplus H^{|x|}_{\sigma_2}(r); \\
\qquad \overline{r} \leftarrow r \oplus H^{128}_{\sigma_3}(\overline{x}); \\
\qquad r_x = \overline{x} \;\|\; \overline{r}; \\
\qquad i \leftarrow i+1; \\
\quad \textbf{until } \textsc{ind}(r_x); \\
\quad \textbf{return } f(r_x);
\end{array}
$$

Figure 1: *A sample instantiation of the plaintext-aware encryption scheme.*

function f (via *desc*). Such "key separation" is a generally-useful heuristic to help ensure that when the same key is used in multiple (separately-secure) algorithms that the internals of these algorithms do not interact in such a way as to jointly compromise security. The use of "key variants" σ_1, σ_2 and σ_3 is motivated similarly. Our choice to only use half the bits of SHA has to do with a general "deficiency" in the use of SHA-like hash functions to instantiate random oracles; see [2] for a discussion.

Acknowledgments

We thank Don Johnson for an early discussion on this problem, where he described the method of [15]. We thank Silvio Micali for encouraging us to find and present the exact security of our constructions. Thanks also to (anonymous) Eurocrypt reviewers for their comments and corrections.

This work was carried out while the first author was at the IBM T. J. Watson Research Center, New York, and the second author worked for IBM in Austin, Texas (System Design, Jamil Bissar, PSP LAN Systems).

References

[1] M. Bellare, J. Kilian and P. Rogaway, "On the security of cipher-block chaining," *Advances in Cryptology – Crypto* 94 *Proceedings*, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994.

[2] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," *Proceedings of the First Annual Conference on Computer and Communications Security*, ACM, 1993.

[3] L. Blum, M. Blum and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator," *SIAM Journal on Computing* **15**(2), 364-383, May 1986.

[4] M. Blum and S. Goldwasser, "An efficient probabilistic public-key encryption scheme which hides all partial information," *Advances in Cryptology – Crypto* 84 *Proceedings*, Lecture Notes in Computer Science Vol. 196, R. Blakely ed., Springer-Verlag, 1984.

[5] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM Journal on Computing* **13**(4), 850-864, November 1984.

[6] I. Damgård, "Towards practical public key cryptosystems secure against chosen ciphertext attacks," *Advances in Cryptology – Crypto* 91 *Proceedings*, Lecture Notes in Computer Science Vol. 576, J. Feigenbaum ed., Springer-Verlag, 1991.

[7] D. Dolev, C. Dwork and M. Naor, "Non-malleable cryptography," Proceedings *of the* 23rd *Annual Symposium on Theory of Computing*, ACM, 1991.

[8] S. Even, O. Goldreich and S. Micali, "On-line/Off line digital signatures," Manuscript. Preliminary version in *Advances in Cryptology – Crypto* 89 *Proceedings*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[9] U. Feige, A. Fiat and A. Shamir, "Zero knowledge proofs of identity," *Journal of Cryptology*, Vol. 1, pp. 77–94, 1987.

[10] O. Goldreich and L. Levin, "A hard predicate for all one-way functions," Proceedings *of the* 21st *Annual Symposium on Theory of Computing*, ACM, 1989.

[11] S. Goldwasser and S. Micali, "Probabilistic Encryption," *Journal of Computer and System Sciences* **28**, 270-299, April 1984.

[12] S. Goldwasser, S. Micali and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM J. of Comp.*, Vol. 18, No. 1, 186–208, February 1989.

[13] S. Goldwasser, S. Micali and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal of Computing*, 17(2):281–308, April 1988.

[14] R. Impagliazzo, L. Levin and M. Luby, "Pseudo-random generation from one-way functions," Proceedings *of the* 21st *Annual Symposium on Theory of Computing*, ACM, 1989.

[15] D. Johnson, A. Lee, W. Martin, S. Matyas and J. Wilkins, "Hybrid key distribution scheme giving key record recovery," IBM Technical Disclosure Bulletin, 37(2A), 5–16, February 1994.

[16] T. Leighton and S. Micali, "Provably fast and secure digital signature algorithms based on secure hash functions," Manuscript, March 1993.

[17] M. Naor and M. Yung, "Public-key cryptosystems provably secure against chosen ciphertext attacks," Proceedings *of the* 22nd *Annual Symposium on Theory of Computing*, ACM, 1990.

[18] National Institute of Standards, FIPS Publication 180, "Secure Hash Standard," 1993.

[19] M. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," MIT Laboratory for Computer Science TR-212, January 1979.

[20] R. Rivest, "The MD5 message-digest algorithm," IETF Network Working Group, RFC 1321, April 1992.

[21] R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," CACM 21 (1978).

[22] RSA Data Security, Inc., "PKCS #1: RSA Encryption Standard," June 1991.

[23] C. Schnorr, "Efficient identification and signatures for smart cards," *Advances in Cryptology – Crypto* 89 *Proceedings*, Lecture Notes in Computer Science Vol. 435, G. Brassard ed., Springer-Verlag, 1989.

[24] A. Schrift and A. Shamir, "The discrete log is very discreet," Proceedings *of the* 22nd *Annual Symposium on Theory of Computing*, ACM, 1990.

[25] M. Tompa and H. Woll, "Random self-reducibility and zero-knowledge interactive proofs of possession of information," UCSD TR CS92-244, 1992.

[26] A. Yao, "Theory and applications of trapdoor functions," Proceedings *of the* 23rd *Symposium on Foundations of Computer Science*, IEEE, 1982.

[27] Y. Zheng and J. Seberry, "Practical approaches to attaining security against adaptively chosen ciphertext attacks," *Advances in Cryptology – Crypto* 92 *Proceedings*, Lecture Notes in Computer Science Vol. 740, E. Brickell ed., Springer-Verlag, 1992.

A Proof of Theorem 4.1

We first define the behavior of inverting algorithm M. M is given (an encoding of) a function $f : \{0,1\}^k \to \{0,1\}^k$ and a string $y \in \{0,1\}^k$. It is trying to find $w = f^{-1}(y)$.

(1) M begins by constructing $\mathcal{E}$ from f as specified by our basic scheme. It then initializes two lists, called its G-list and its H-list, to empty. It picks a bit $b \leftarrow \{0,1\}$ at random. Then it simulates the two stages of A as indicated in the next two steps.

(2) M simulates the find-stage of A by running A on input $(\mathcal{E}, \mathsf{find})$. M provides A with fair random coins and simulates A's random oracles G and H as follows. When A makes an oracle call h of H, machine M provides A with a random string H_h of length k_0, and adds h to the H-list. Similarly when A makes an oracle call g of G, machine M provides A with a random string G_g of length n and adds g to the G-list. Let (x_0, x_1, c) be the output with which A halts.

(3) Now M starts simulating the guess-stage of A. It runs A on input (y, x_0, x_1, c). It responds to oracle queries as follows.

  (3.1) Suppose A makes H-query h. M provides A with a random string H_h of length h and adds h to the H-list. Then for each g on the G-list M constructs $w_{h,g} = h \parallel g \oplus H_h$ and computes $y_{h,g} = f(w_{h,g})$. If there is some h, g such that $y_{h,g} = y$ then M sets $w^* = w_{h,g}$.

(3.2) Suppose A makes G-query g. Then for each h on the H-list M constructs the string $w_{h,g} = h \parallel g \oplus H_h$ and computes $y_{h,g} = f(w_{h,g})$.

(3.2.1) If there are h, g such that $y_{h,g} = y$ then M sets $w^* = w_{h,g}$. It sets $G_g = h \oplus x_b$, adds g to the G-list, and returns G_g to A.

(3.2.2) Else (ie. there are no h, g such that $y_{h,g} = y$) M provides A with a random string G_g of length n and adds g to the G-list.

The output of M is w^* if this string was defined in the above experiment, and fail otherwise. Note that the H-list and G-list include the queries of both the find and guess stages of A's execution.

It is easy to verify that the amount of time t' to carry out Game 1 is as claimed. It is also easy to verify that there is a universal machine U such that the computation of M can be done by U^A.

We note that as soon as M successfully finds a point $w^* = f^{-1}(y)$, it could stop and output w^*. Not only do we have it go on, but some variables and actions (such as the usage of the bit b in Step (3.2.1)) come into play only after w^* is found. These "unnecessary" actions do not affect the success probability of M but we put them in to simplify our exposition of the analysis of M's success probability. The intuition is that A in the above experiment is trying to predict b and M is trying to make the distribution provided to A look like that which A would expect were A running under the experiment which defines A's success in breaking the encryption scheme. Unfortunately, M does not provide A with a simulation which is quite perfect. Let us now proceed to the analysis.

We consider the probability space given by the above experiment. The inputs f, y to M are drawn at random according to $(f, f^{-1}) \leftarrow \mathcal{F}(1^k);\ y \leftarrow \{0,1\}^k$. We call this "Game 1" and we let $\Pr_1[\cdot]$ denote the corresponding probability.

Let $w = f^{-1}(y)$ and write it as $w = s \parallel t$ where $|s| = n$ and $|t| = k_0$. Let r be the random variable $t \oplus H(s)$. We consider the following events.

FBAD is true if:

- G-oracle query r was made in the find-stage, and
- $G_r \notin \{s \oplus x_0, s \oplus x_1\}$.

GBAD is true if:

- G-oracle query r was made in the guess stage, and
- at the point in time that it was made, the H-oracle query s was not on the H-list, and
- $G_r \notin \{s \oplus x_0, s \oplus x_1\}$.

$\mathsf{G} = \neg\mathsf{FBAD} \wedge \neg\mathsf{GBAD}$.

We let $\Pr_2[\cdot] = \Pr_1[\cdot \mid \mathsf{G}]$ denote the probability distribution, in Game 1, conditioned on G being true, and call this "Game 2."

Now consider the experiment which defines the advantage of A. Namely, first choose $(f_*, f_*^{-1}) \leftarrow \mathcal{F}(1^k)$ and let $\mathcal{E}^*$ be the corresponding encryption function under the basic scheme. Then choose

$$G_*, H_* \leftarrow \Omega;\ (x_0^*, x_1^*, c^*) \leftarrow A^{G_*,H_*}(\mathcal{E}^*, \mathsf{find});\ b_* \leftarrow \{0,1\};\ y^* \leftarrow \mathcal{E}^{G_*,H_*}(x_b^*)\,,$$

and run $A^{G_*,H_*}(y^*, x_0^*, x_1^*, c^*)$. Let $\Pr_1^*[\cdot]$ be the corresponding distribution and Game 1^* the game.

Now consider playing Game 1^* a little bit differently. As before, choose $(f_*, f_*^{-1}) \leftarrow \mathcal{F}(1^k)$ and let $\mathcal{E}^*$ be the corresponding encryption function. But now choose $y^* \leftarrow \{0,1\}^k$ uniformly at random first, and then select the rest according to the distribution which makes the outcome the same as in Game-1^*. (This is possible because the distribution on y^*-values in Game 1^* is indeed uniform). We let Game 2^* be this different way of playing Game 1^*.

We claim that Game 2 and Game 2^* are identical in the sense that the view of A at any point in these two games is the same. Indeed we have chosen the event G so that the oracle queries we are returning in Game 1 will mimic Game 2^* as long as G remains true.

We omit details to formally justify these claims, but a good way to get some intuition is to assume for simplicity that the find-stage is trivial and A always outputs the same strings x_0^*, x_1^*, c^*. Now if y^* is fixed then the conditional distribution on G_*, H_* can be described as follows: Pick H_* at random; pick $G_*(g)$ to be random whenever $g \neq t \oplus H_*(s)$. But $G_*(t \oplus H_*(s))$ must be constrained to be either $s \oplus x_0^*$ or $s \oplus x_1^*$, the choice of which being at random.

To proceed further with our analysis (of Game 1), let us introduce the following additional events:

FAskS is true if H-oracle query s was made in the find-stage.

AskR is true if, at the end of the guess-stage, r is on the G-list.

AskS is true if, at the end of the guess-stage, s is on the H-list.

$\mathsf{W} = \mathsf{AskR} \wedge \mathsf{AskS}$.

The first step is to show that the probability that the good event fails is low.

Lemma A.1 The probability that the good event fails is upper bounded by

$$\Pr_1[\neg\mathsf{G}] \quad \leq \quad q_{\rm gen} 2^{-k_0} + q_{\rm hash} 2^{-n} \ .$$

Proof: The intuition is that as long as H-query s has not been made, each G-query has probability only 2^{-k_0} of being r. Now, $\neg\mathsf{G} = \mathsf{FBAD} \vee \mathsf{GBAD}$. In GBAD is already included the fact that no H-query of s has been made before the G-query r. But in FBAD it could be that H-query s was made. But the probability of FAskS is small since $s \parallel t = f^{-1}(y)$ is determined at random after the find-stage. The proof that follows captures all this by conditioning on FAskS. We have:

$$\begin{aligned} \Pr_1[\neg\mathsf{G}] \quad &= \quad \Pr_1[\neg\mathsf{G} \mid \mathsf{FAskS}] \cdot \Pr_1[\mathsf{FAskS}] + \Pr_1[\neg\mathsf{G} \mid \neg\mathsf{FAskS}] \cdot \Pr_1[\neg\mathsf{FAskS}] \\ &\leq \quad \Pr_1[\mathsf{FAskS}] + \Pr_1[\neg\mathsf{G} \mid \neg\mathsf{FAskS}] \\ &\leq \quad \Pr_1[\mathsf{FAskS}] + \Pr_1[\mathsf{AskR} \mid \neg\mathsf{FAskS}] \ . \end{aligned}$$

The random choice of y implies that $\Pr_1[\mathsf{FAskS}] \leq q_{\rm hash} 2^{-n}$ while, on the other hand, we have $\Pr_1[\mathsf{AskR} \mid \neg\mathsf{FAskS}] \leq q_{\rm gen} 2^{-k_0}$. ■

We think of A in Game 1 as trying to predict b. With this in mind, let "$A = b$" denote the event that A is successful in predicting bit b. We analyze this probability to show that in Game 2 either W is true or A has little advantage in predicting b. Notice that if W is true then M successfully finds $w = f^{-1}(y)$. Following this we will use the equivalence with Game 2* to relate this to ϵ, and finally we will use Lemma A.1 to get a conclusion for Game 1.

Recall that $k = k_0 + n$ is the "security parameter" of the original trapdoor permutation.

Lemma A.2 The winning probability in Game 2 is bounded below by:

$$\Pr_2[\mathsf{W}] \quad \geq \quad 2 \cdot \Pr_2[A = b] \; - \; 1 \; - \; \frac{2q_{\text{gen}}2^{-k}}{\Pr_1[\mathsf{G}]} \; .$$

Proof: We upper bound $\Pr_2[A = b]$ by:

$$\begin{aligned}
\Pr_2[A = b] \quad &= \quad \Pr_2[A = b \mid \mathsf{W}] \cdot \Pr_2[\mathsf{W}] \; + \Pr_2[A = b \mid \neg\mathsf{AskR}] \cdot \Pr_2[\neg\mathsf{AskR}] \\
&\qquad + \Pr_2[A = b \mid \mathsf{AskR} \wedge \neg\mathsf{AskS}] \cdot \Pr_2[\mathsf{AskR} \wedge \neg\mathsf{AskS}] \\
&\leq \quad \Pr_2[\mathsf{W}] \; + \Pr_2[A = b \mid \neg\mathsf{AskR}] \cdot \Pr_2[\neg\mathsf{AskR}] \; + \Pr_2[\mathsf{AskR} \wedge \neg\mathsf{AskS}] \\
&= \quad \Pr_2[\mathsf{W}] \; + \Pr_2[A = b \mid \neg\mathsf{AskR}] \cdot (1 - \Pr_2[\mathsf{W}] - \Pr_2[\mathsf{AskR} \wedge \neg\mathsf{AskS}]) \\
&\qquad + \Pr_2[\mathsf{AskR} \wedge \neg\mathsf{AskS}] \; . \qquad (1)
\end{aligned}$$

Now observe that if $\neg\mathsf{AskR}$ then A has no advantage in predicting b:

$$\Pr_2[A = b \mid \neg\mathsf{AskR}] \leq 1/2 \; . \qquad (2)$$

In order to upper bound $\Pr_2[\mathsf{AskR} \wedge \neg\mathsf{AskS}]$, let RBS be the event that r is on the G-list and at the time it was put there, s was not on the H-list. Recall that $k = k_0 + n$. One can check that:

$$\begin{aligned}
\Pr_1[\mathsf{AskR} \wedge \neg\mathsf{AskS} \wedge \mathsf{G}] \quad &= \quad \Pr_1[\mathsf{RBS} \; \wedge \; G_r \in \{s \oplus x_0, s \oplus x_1\}] \\
&= \quad \Pr_1[\mathsf{RBS}] \cdot \Pr_1[G_r \in \{s \oplus x_0, s \oplus x_1\} \mid \mathsf{RBS}] \\
&\leq \quad q_{\text{gen}} 2^{-k_0} \cdot 2 \cdot 2^{-n} \\
&= \quad 2q_{\text{gen}}2^{-k} \; . \qquad (3)
\end{aligned}$$

Using (3) we have

$$\Pr_2[\mathsf{AskR} \wedge \neg\mathsf{AskS}] = \frac{\Pr_1[\mathsf{AskR} \wedge \neg\mathsf{AskS} \wedge \mathsf{G}]}{\Pr_1[\mathsf{G}]} \leq \frac{2q_{\text{gen}}2^{-k}}{\Pr_1[\mathsf{G}]} \; . \qquad (4)$$

Now put the bounds provided by (2) and (4) into (1) to get

$$\Pr_2[A = b] \quad \leq \quad \tfrac{1}{2}\Pr_2[\mathsf{W}] \; + \tfrac{1}{2} + \frac{q_{\text{gen}}2^{-k}}{\Pr_1[\mathsf{G}]}$$

and we may conclude the lemma. ■

The equivalence of Game 2 and Game 2* implies $\Pr_2[A = b] \leq \epsilon + 1/2$ so that from Lemma A.2 (making the conditioning on G explicit) we get

$$\Pr_1[\mathsf{W} \mid \mathsf{G}] \quad \geq \quad \epsilon \; - \; \frac{2q_{\text{gen}}2^{-k}}{\Pr_1[\mathsf{G}]} \; . \qquad (5)$$

Using (5) and Lemma A.1 we get

$$\begin{aligned}
\Pr_1[\mathsf{W}] &\geq \Pr_1[\mathsf{W} \mid \mathsf{G}] \cdot \Pr_1[\mathsf{G}] \\
&\geq \left(\epsilon - \frac{2q_{\text{gen}}2^{-k}}{\Pr_1[\mathsf{G}]}\right) \cdot \Pr_1[\mathsf{G}] \\
&= \epsilon \cdot \Pr_1[\mathsf{G}] \;-\; 2q_{\text{gen}}2^{-k} \\
&\geq \epsilon \cdot (1 - q_{\text{gen}}2^{-k_0} - q_{\text{hash}}2^{-n}) \;-\; 2q_{\text{gen}}2^{-k} \,.
\end{aligned}$$

However as we remarked earlier, $\epsilon' \geq \Pr_1[\mathsf{W}]$, so the proof is concluded.

B Proof of Theorem 6.2

We define the plaintext extractor K. Let $(f, f^{-1}) \in [\mathcal{F}(1^k)]$ and let $\mathcal{E}$ be the corresponding encryption function as constructed by our plaintext-aware scheme. Let $\tau = (\tau_{\text{gen}}, \tau_{\text{hash}})$ where

$$\begin{aligned}
\tau_{\text{gen}} &= (r_1, G_1), \ldots, (r_{q_{\text{gen}}}, G_{q_{\text{gen}}}) \\
\tau_{\text{hash}} &= (s_1, H_1), \ldots, (s_{q_{\text{hash}}}, H_{q_{\text{hash}}}) \,.
\end{aligned}$$

We call $r_1, \ldots, r_{q_{\text{gen}}}$ the G-list and $s_1, \ldots, s_{q_{\text{hash}}}$ the H-list. The inputs to K are $\mathcal{E}, y, \tau$. It proceeds as follows.

(1) For $i = 1, \ldots, q_{\text{gen}}$ and $j = 1, \ldots, q_{\text{hash}}$ machine K

  (1.1) Sets $x_{i,j}$ to the first n bits of $s_i \oplus G_j$ and $z_{i,j}$ to the remaining k_1 bits of $s_i \oplus G_j$

  (1.2) Sets $w_{i,j} = s_i \parallel r_j \oplus H_i$ and computes $y_{i,j} = f(w_{i,j})$.

(2) If there is an i, j such that $y_{i,j} = y$ and $z_{i,j} = 0^{k_1}$ then K outputs $x_{i,j}$; else it outputs $*$.

For the analysis let $w = f^{-1}(y)$ and write it as $w = s \parallel t$ where $|s| = n + k_1$ and $|t| = k_0$. Let r be the random variable $t \oplus H(s)$. Let x, z be the random variables defined by writing $s \oplus G(r) = x \parallel z$ where $|x| = n$ and $|z| = k_1$. We consider the following events.

FAIL is true if the output of K is different from $\mathcal{D}^{G,H}(y)$.

AskR is true if r is on the G-list.

AskS is true if s is on the H-list.

We now bound the failure probability.

$$\begin{aligned}
\Pr[\mathsf{FAIL}] &= \Pr[\mathsf{FAIL} \mid \neg\mathsf{AskR}] \cdot \Pr[\neg\mathsf{AskR}] \\
&\qquad + \Pr[\mathsf{FAIL} \mid \mathsf{AskR} \wedge \mathsf{AskS}] \cdot \Pr[\mathsf{AskR} \wedge \mathsf{AskS}] \\
&\qquad\quad + \Pr[\mathsf{FAIL} \mid \mathsf{AskR} \wedge \neg\mathsf{AskS}] \cdot \Pr[\mathsf{AskR} \wedge \neg\mathsf{AskS}] \\
&\leq \Pr[\mathsf{FAIL} \mid \neg\mathsf{AskR}] + \Pr[\mathsf{FAIL} \mid \mathsf{AskR} \wedge \mathsf{AskS}] + \Pr[\mathsf{AskR} \wedge \neg\mathsf{AskS}] \,.
\end{aligned}$$

If r is not on the G-list then the probability that $z = 0^{k_1}$ is at most 2^{-k_1}, so that in this case an output of $*$ is success. Thus $\Pr[\mathsf{FAIL} \mid \neg\mathsf{AskR}] \leq 2^{-k_1}$.

If r is on the G-list and s is on the H-list then there are i, j such that $w = w_{i,j}$. So K will decrypt correctly. That is, $\Pr[\mathsf{FAIL} \mid \mathsf{AskR} \wedge \mathsf{AskS}] = 0$.

If s is not on the H-list then $H(s)$ is uniformly distributed and hence so is r. So

$$\Pr[\mathsf{AskR} \wedge \neg\mathsf{AskS}] \leq \Pr[\mathsf{AskR} \mid \neg\mathsf{AskS}] \leq q_{\mathrm{gen}} 2^{-k_0} \ .$$

This concludes the proof.