

## OAEP Reconsidered\*

Victor Shoup

IBM Zurich Research Lab,  
Säumerstrasse 4, 8803 Rüschlikon, Switzerland  
sho@zurich.ibm.com

Communicated by Joan Feigenbaum

Received September 2001 and revised May 2002  
Online publication 12 August 2002

**Abstract.** The OAEP encryption scheme was introduced by Bellare and Rogaway at Eurocrypt '94. It converts any trapdoor permutation scheme into a public key encryption scheme. OAEP is widely believed to provide resistance against adaptive chosen ciphertext attack. The main justification for this belief is a supposed proof of security in the random oracle model, assuming the underlying trapdoor permutation scheme is one way.

This paper shows conclusively that this justification is invalid. First, it observes that there appears to be a non-trivial gap in the OAEP security proof. Second, it proves that this gap cannot be filled, in the sense that there can be no standard “black box” security reduction for OAEP. This is done by proving that there exists an oracle relative to which the general OAEP scheme is insecure.

The paper also presents a new scheme OAEP+, along with a complete proof of security in the random oracle model. OAEP+ is essentially just as efficient as OAEP.

It should be stressed that these results do not imply that a particular instantiation of OAEP, such as RSA-OAEP, is insecure. They simply undermine the original justification for its security. In fact, it turns out—essentially by accident, rather than by design—that RSA-OAEP is secure in the random oracle model; however, this fact relies on special algebraic properties of the RSA function, and not on the security of the general OAEP scheme.

**Key words.** Public-key encryption, Chosen ciphertext security, Random oracle model.

### 1. Introduction

It is generally agreed that the “right” definition of security for a public key encryption scheme is *security against adaptive chosen ciphertext attack*, as defined in [RS]. This notion of security is equivalent to other useful notions, such as the notion of *non-malleability*, as defined in [DDN1] and [DDN2].

---

\* This is a revision, with corrections and improvements, of the original November 16, 2000 version. An extended abstract of this paper appeared in *Crypto* 2001.

Dolev et al. [DDN1] proposed a scheme that is provably secure in this sense, based on standard intractability assumptions. While this scheme is useful as a proof of concept, it is quite impractical. Rackoff and Simon [RS] also propose a scheme that is also provably secure; however, it too is also quite impractical, and, moreover, it has special “public key infrastructure” requirements.

In 1993 Bellare and Rogaway proposed a method for converting any trapdoor permutation scheme into an encryption scheme [BR1]. They proved that this scheme is secure against adaptive chosen ciphertext attack in the *random oracle model*, provided the underlying trapdoor permutation scheme is one way.

In the random oracle model, one analyzes the security of the scheme by *pretending* that a cryptographic hash function is really a *random oracle*.

The encryption scheme in [BR1] is very efficient from the point of view of computation time. However, it has a “message expansion rate” that is not as good as some other encryption schemes.

In 1994 Bellare and Rogaway proposed another method for converting any trapdoor permutation scheme into an encryption scheme [BR2]. This scheme goes by the name OAEP. The scheme when instantiated with the RSA function [RSA] goes by the name RSA-OAEP, and is the industry-wide standard for RSA encryption (PKCS#1 version 2, IEEE P1363). It is just as efficient computationally as the scheme in [BR1], but it has a better message expansion rate. With RSA-OAEP, one can encrypt messages whose bit-length is up to just a few hundred bits less than the number of bits in the RSA modulus, yielding a ciphertext whose size is the same as that of the RSA modulus.

Besides its efficiency in terms of both time and message expansion, and its compatibility with more traditional implementations of RSA encryption, perhaps one of the reasons that OAEP is so popular is the widespread *belief* that the scheme is provably secure in the random oracle model, provided the underlying trapdoor permutation scheme is one way.

In this paper we argue that this belief is unjustified. Specifically, we argue that in fact, no *complete* proof of the general OAEP method has ever appeared in the literature. Moreover, we prove that no proof is attainable using standard “black box” reductions, even in the random oracle model.

More specifically, we show that there exists an oracle relative to which the general OAEP scheme is insecure. This rules out any “black box” reduction, as such a reduction would also work relative to any oracle.

We also present a variation, OAEP+, and a complete proof of security in the random oracle model. OAEP+ is essentially just as efficient as OAEP.

There is one more twist to this story: we observe that RSA-OAEP with encryption exponent 3 actually *is* provably secure in the random oracle model; the proof, of course, is not a “black box” reduction, but exploits special algebraic properties of the RSA function. These observations have been subsequently extended in [FOPS] to RSA-OAEP with arbitrary encryption exponent.

Note that although the precise specification of standards (PKCS#1 version 2, IEEE P1363) differ in a few minor points from the scheme described in [BR2], none of these minor changes affect the arguments we make here.

### 1.1. A Missing Proof of Security

[BR2] contains a valid proof that OAEP satisfies a certain technical property which they call “plaintext awareness.” We call this property *PA1*. However, it is claimed *without proof* that *PA1* implies security against chosen ciphertext attack and non-malleability. Moreover, it is not even clear if the authors mean adaptive chosen ciphertext attack (as in [RS]) or *indifferent* (a.k.a. *lunchtime*) chosen ciphertext attack (as in [NY]).

Later, in [BDPR], a new definition of “plaintext awareness” is given. We call this property *PA2*. It is claimed in [BDPR] that OAEP is “plaintext aware.” It is not clear if the authors mean to say that OAEP is *PA1* or *PA2*; in any event, they certainly do not prove anything new about OAEP in [BDPR]. Furthermore, [BDPR] contains a valid proof that *PA2* implies security against adaptive chosen ciphertext attack.

Notice that nowhere in this chain of reasoning is a proof that OAEP is secure against adaptive chosen ciphertext attack. What is missing is a proof that either OAEP is *PA2*, or that *PA1* implies security against adaptive chosen ciphertext attack.

We should point out, however, that *PA1* is trivially seen to imply security against *indifferent* chosen ciphertext attack, and thus OAEP is secure against indifferent chosen ciphertext attack. However, this is a strictly weaker and much less useful notion of security than security against adaptive chosen ciphertext attack.

### 1.2. Our Contributions

In Section 4 we give a rather informal argument that there is a non-trivial obstruction to obtaining a complete proof of security for OAEP against adaptive chosen ciphertext attack (in the random oracle model).

In Section 5 we give more formal and compelling evidence for this. Specifically, we prove that if one-way trapdoor permutation schemes with an additional special property exist, then OAEP when instantiated with such a one-way trapdoor permutation scheme is in fact *insecure*. We do not know how to prove the existence of such special one-way trapdoor permutation schemes (assuming, say, that one-way trapdoor permutation schemes exist at all). However, we prove that there exists an oracle, relative to which such special one-way trapdoor permutation schemes exist. It follows that relative to an oracle, the OAEP construction is not secure.

Actually, our proofs imply something slightly stronger: relative to an oracle, OAEP is *malleable* with respect to a *chosen plaintext* attack.

Of course, such relativized results do not necessarily imply anything about the ordinary, unrelativized security of OAEP. However, they do imply that standard proof techniques, in which the adversary and the trapdoor permutation are treated as “black boxes,” cannot possibly yield a proof of security, since they would relativize. Such “black box” reductions are by far the most common technique used for proving the security of cryptographic constructions. Certainly, all of the arguments in [BR2] and [BDPR] involve only “black box” reductions, and so they cannot possibly be modified to yield a proof of security.

In Section 6 we present a new scheme, called OAEP+. This is a variation of OAEP that is essentially just as efficient in all respects as OAEP, but for which we provide a complete, detailed proof of security against adaptive chosen ciphertext attack. Moreover,

the security reduction for OAEP+ is somewhat tighter than the security reduction for OAEP in [BR2] (which only shows security against indifferent chosen ciphertext attack).

We conclude the paper in Section 7 on a rather ironic note. After considering other variations of OAEP, we sketch a proof that RSA-OAEP with encryption exponent 3 actually *is* secure in the random oracle model. This fact, however, makes essential use of Coppersmith’s algorithm [Co] for solving low-degree modular equations. This proof of security does not generalize to large encryption exponents, and, in particular, it does not cover the popular encryption exponent  $2^{16} + 1$ .

Part of the irony of this observation is that Coppersmith viewed his own result as a reason *not* to use exponent 3, while here, it ostensibly gives one reason why one perhaps *should* use exponent 3.

It is also worth noting here that by using Coppersmith’s algorithm, one gets a fairly tight security reduction for exponent-3 RSA-OAEP, and an even tighter reduction for exponent-3 RSA-OAEP+. These reductions are much more efficient than either the reduction (proving the PA1 property) for OAEP in [BR2], or our general reduction for OAEP+. Indeed, these general reductions are so inefficient that they fail to provide any truly meaningful security guarantees for, say, 1024-bit RSA, whereas with the use of Coppersmith’s algorithm, the security guarantees are much more meaningful.

Subsequent to the distribution of the original version of this paper,<sup>1</sup> it was shown in [FOPS] that RSA-OAEP with an arbitrary encryption exponent is indeed secure against adaptive chosen ciphertext attack in the random oracle model. We remark, however, that the reduction in [FOPS] is significantly less efficient than our general reduction for OAEP+, and so it provides a less meaningful security guarantee for typical choices of security parameters. This may be a reason to consider using RSA-OAEP+ instead of RSA-OAEP.

We also mention the subsequent work of Boneh [Bo], which studies further variations of OAEP as applied to both the RSA and Rabin functions.

Let us be clear about the implications of our results. They do not imply an attack on RSA-OAEP. They only imply that the original *justification* for the *belief* that OAEP in general—and hence RSA-OAEP in particular—is resistant against adaptive chosen ciphertext attack was *invalid*. As it turns out, our observations on exponent-3 RSA-OAEP, and the more general results of [FOPS] on arbitrary-exponent RSA-OAEP, imply that RSA-OAEP is indeed secure against adaptive chosen ciphertext attack in the random oracle model. However, the security of RSA-OAEP does not follow from the security of OAEP in general, but, rather, relies on specific algebraic properties of the RSA function.

Before moving ahead, we recall some definitions in Section 2, and the OAEP scheme itself in Section 3.

## 2. Preliminaries

### 2.1. Security against Chosen Ciphertext Attack

We recall the definition of security against adaptive chosen ciphertext attack.

---

<sup>1</sup> Cryptology ePrint Archive, Report 2000/060, <http://eprint.iacr.org>.

We begin by describing the attack scenario.

- Stage 1.** The key generation algorithm is run, generating the public key and private key for the cryptosystem. The adversary, of course, obtains the public key, but not the private key.
- Stage 2.** The adversary makes a series of arbitrary queries to a *decryption oracle*. Each query is a ciphertext  $y$  that is decrypted by the decryption oracle, making use of the private key of the cryptosystem. The resulting decryption is given to the adversary. The adversary is free to construct the ciphertexts in an arbitrary way—it is certainly *not* required to compute them using the encryption algorithm.
- Stage 3.** The adversary prepares two messages  $x_0, x_1$ , and gives these to an *encryption oracle*. The encryption oracle chooses  $b \in \{0, 1\}$  at random, encrypts  $x_b$ , and gives the resulting “target” ciphertext  $y^*$  to the adversary. The adversary is free to choose  $x_0$  and  $x_1$  in an arbitrary way, except that if message lengths are not fixed by the cryptosystem, then these two messages must nevertheless be of the same length.
- Stage 4.** The adversary continues to submit ciphertexts  $y$  to the decryption oracle, subject only to the restriction that  $y \neq y^*$ .
- Stage 5.** The adversary outputs  $\hat{b} \in \{0, 1\}$ , representing its “guess” of  $b$ .

That completes the description of the attack scenario.

The adversary’s *advantage* in this attack scenario is defined to be  $|\Pr[\hat{b} = b] - \frac{1}{2}|$ .

A cryptosystem is defined to be *secure against adaptive chosen ciphertext attack* if for any efficient adversary, its advantage is negligible.

Of course, this is a complexity-theoretic definition, and the above description suppresses many details, e.g., there is an implicit security parameter which tends to infinity, and the terms “efficient” and “negligible” are technical terms, defined in the usual way. Also, we work in a *uniform* model of computation; i.e., we consider only computations that are carried out by Turing machines, as opposed to considering arbitrary circuit families.

The definition of security we have presented here is from [RS]. It is called *IND-CCA2* in [BDPR]. It is known to be equivalent to other notions, such as non-malleability [DDN1], [BDPR], [DDN2], which is called *NM-CCA2* in [BDPR].

It is fairly well understood and accepted that this notion of security is the “right” one, in the sense that a general-purpose cryptosystem that is to be deployed in a wide range of applications should satisfy this property. Indeed, with this property, one can typically establish the security of larger systems that use such a cryptosystem as a component.

There are other, weaker, notions of security against chosen ciphertext attack. For example, Naor and Yung [NY] define a notion that is sometimes called *security against indifferent chosen ciphertext attack*, or *security against lunchtime attack*. This definition of security is exactly the same as the one above, except that Stage 4 is omitted—that is, the adversary does not have access to the decryption oracle after it obtains the target ciphertext. This notion is called *IND-CCA1* in [BDPR]. While this notion of security may seem natural, it is actually not sufficient in many applications, such as protocols for key exchange [BCK], [Sh2] and atomic transactions [ASW].

An even weaker notion of security is called *semantic security*, defined in [GM]. This definition of security is equivalent to a definition that is the same as the one above, except that both Stages 2 and 4 are omitted—that is, the adversary does not have any access to the decryption oracle. This notion is called *IND-CPA* in [BDPR].

## 2.2. One-Way Trapdoor Permutations

We recall the notion of a trapdoor permutation scheme. This consists of a probabilistic *permutation generator* algorithm that outputs (descriptions of) two algorithms  $f$  and  $g$ , such that the function computed by  $f$  is a permutation on the set of  $k$ -bit strings, and the function computed by  $g$  is its inverse.

An attack on a trapdoor permutation scheme proceeds as follows. First the generator is run, yielding  $f$  and  $g$ . The adversary is given  $f$ , but not  $g$ . Additionally, the adversary is given a random  $y \in \{0, 1\}^k$ . The adversary then computes and outputs a string  $w \in \{0, 1\}^k$ .

The adversary's *success probability* is defined to  $\Pr[f(w) = y]$ .

The scheme is called a *one-way* trapdoor permutation scheme if for any efficient adversary, its success probability is negligible. As above, this is a complexity-theoretic definition, and we have suppressed a number of details, including a security parameter, which is input to the permutation generator; the parameter  $k$ , as well as the running times of  $f$  and  $g$ , should be bounded by a polynomial in this security parameter.

## 2.3. The Random Oracle Model

The random oracle model was introduced in [BR1] as a means of heuristically analyzing a cryptographic primitive or protocol. In this approach, one equips all of the algorithms associated with the primitive or protocol (including the adversary's algorithms) with oracle access to one or more functions. Each of these functions is a map from  $\{0, 1\}^a$  to  $\{0, 1\}^b$ , for some specified values  $a$  and  $b$ . One then reformulates the definition of security so that in the attack game, each of these functions is chosen at random from the set of all functions mapping  $\{0, 1\}^a$  to  $\{0, 1\}^b$ .

In an actual implementation, one typically instantiates these random oracles as cryptographic hash functions.

Now, a proof of security in the random oracle model does not necessarily imply *anything* about security in the “real world” where actual computation takes place (see [CGH]). Nevertheless, it seems that designing a scheme so that it is provably secure in the random oracle model is a good engineering principle, at least when all known schemes that are provably secure without the random oracle heuristic are too impractical. Subsequent to [BR1], many other papers have proposed and analyzed cryptographic schemes in the random oracle model.

## 3. OAEP

We now describe the OAEP encryption scheme, as described in Section 6 of [BR2].

The general scheme makes use of a one-way trapdoor permutation. Let  $f$  be the permutation, acting on  $k$ -bit strings, and let  $g$  be its inverse. The scheme also makes use

of two parameters  $k_0$  and  $k_1$ , which should satisfy  $k_0 + k_1 < k$ . It should also be the case that  $2^{-k_0}$  and  $2^{-k_1}$  are negligible quantities. The scheme encrypts messages  $x \in \{0, 1\}^n$ , where  $n = k - k_0 - k_1$ .

The scheme also makes use of two functions,  $G: \{0, 1\}^{k_0} \rightarrow \{0, 1\}^{n+k_1}$ , and  $H: \{0, 1\}^{n+k_1} \rightarrow \{0, 1\}^{k_0}$ . These two functions will be modeled as random oracles in the security analysis.

We describe the key generation, encryption, and decryption algorithms of the scheme.

*Key generation.* This simply runs the generator for the one-way trapdoor permutation scheme, obtaining  $f$  and  $g$ . The public key is  $f$ , and the private key is  $g$ .

*Encryption.* Given a plaintext  $x$ , the encryption algorithm randomly chooses  $r \in \{0, 1\}^{k_0}$ , and then computes

$$s \in \{0, 1\}^{n+k_1}, \quad t \in \{0, 1\}^{k_0}, \quad w \in \{0, 1\}^k, \quad y \in \{0, 1\}^k$$

as follows:

$$s = G(r) \oplus (x \parallel 0^{k_1}), \quad (1)$$

$$t = H(s) \oplus r, \quad (2)$$

$$w = s \parallel t, \quad (3)$$

$$y = f(w). \quad (4)$$

The ciphertext is  $y$ .

*Decryption.* Given a ciphertext  $y$ , the decryption algorithm computes

$$w \in \{0, 1\}^k, \quad s \in \{0, 1\}^{n+k_1}, \quad t \in \{0, 1\}^{k_0}, \quad r \in \{0, 1\}^{k_0}, \\ z \in \{0, 1\}^{n+k_1}, \quad x \in \{0, 1\}^n, \quad c \in \{0, 1\}^{k_1}$$

as follows:

$$w = g(y), \quad (5)$$

$$s = w[0 \cdots n + k_1 - 1], \quad (6)$$

$$t = w[n + k_1 \cdots k], \quad (7)$$

$$r = H(s) \oplus t, \quad (8)$$

$$z = G(r) \oplus s, \quad (9)$$

$$x = z[0 \cdots n - 1], \quad (10)$$

$$c = z[n \cdots n + k_1 - 1]. \quad (11)$$

If  $c = 0^{k_1}$ , then the algorithm outputs the cleartext  $x$ ; otherwise, the algorithm *rejects* the ciphertext, and does not output a cleartext.

#### 4. An Informal Argument that OAEP Cannot Be Proven Secure

In this section we discuss the gap in the proof in [BR2]. The reader may safely choose to skip this section upon first reading.

We first recall the main ideas of the proof in [BR2] that OAEP is “plaintext aware” in the random oracle model, where  $G$  and  $H$  are modeled as random oracles.

The argument shows how a decryption simulator, a.k.a. “plaintext extractor,” that has access to a table of input/output values for the points at which  $G$  and  $H$  were queried by the adversary, can effectively simulate the decryption oracle without knowing the private key. Since the information available to such a decryption simulator is already available to the adversary, it follows that the adversary can just run the decryption simulator, rather than invoke the decryption oracle; hence, security against chosen ciphertext attack follows from simple semantic security. Moreover, Bellare and Rogaway [BR2] do in fact give a correct proof of the semantic security of OAEP, assuming that the underlying trapdoor permutation scheme is one way.

As we shall see, one must distinguish between random oracle queries made by the adversary and random oracle queries made by the encryption oracle. This is a subtle point, but the failure to make this distinction is really at the heart of the flawed reasoning in [BR2].

To make our arguments clearer, we introduce some notational conventions. First, any ciphertext  $y$  implicitly defines values  $w, s, t, r, z, x, c$  via the decryption equations (5)–(11). Let  $y^*$  denote the target ciphertext, and let  $w^*, s^*, t^*, r^*, z^*, x^*, c^*$  be the corresponding implicitly defined values for  $y^*$ . Note that  $x^* = x_b$  and  $c^* = 0^{k_1}$ .

Let  $S_G$  be the set of values  $r$  at which  $G$  was queried by the adversary. Also, let  $S_H$  be the set of values  $s$  at which  $H$  was queried by the adversary. Further, let  $S_G^* = S_G \cup \{r^*\}$  and  $S_H^* = S_H \cup \{s^*\}$ , where  $r^*, s^*$  are the values implicitly defined by  $y^*$ , as described above. We view these sets as growing incrementally as the adversary’s attack proceeds—elements are added to these only when a random oracle is queried by the adversary or by the encryption oracle.

Suppose the decryption simulator is given a ciphertext  $y$  to decrypt. One can show that if  $r \notin S_G^*$ , then with overwhelming probability the actual decryption algorithm would reject  $y$ ; this is because in this case,  $s$  and  $G(r)$  are independent, and so the probability that  $c = 0^{k_1}$  is  $2^{-k_1}$ . Moreover, if  $s \notin S_H^*$ , then with overwhelming probability,  $r \notin S_G^*$ ; this is because in this case,  $t$  and  $H(s)$  are independent, and so  $r$  is independent of the adversary’s view. From this argument, it follows that the actual decryption algorithm would reject with overwhelming probability, unless  $r \in S_G^*$  and  $s \in S_H^*$ .

If the decryption simulator has access to  $S_G^*$  and  $S_H^*$ , as well as the corresponding outputs of  $G$  and  $H$ , then it can effectively simulate the decryption without knowing the secret key, as follows. It simply enumerates all  $r' \in S_G^*$  and  $s' \in S_H^*$ , and for each of these computes

$$t' = H(s') \oplus r', \quad w' = s' \parallel t', \quad y' = f(w').$$

If  $y'$  is equal to  $y$ , then it computes the corresponding  $x'$  and  $c'$  values, via (10) and (11); if  $c' = 0^{k_1}$ , it outputs  $x'$ , and otherwise rejects. If no  $y'$  equals  $y$ , then it simply outputs reject.



Given the above arguments, it is easy to see that this decryption simulator oracle behaves exactly like the actual decryption oracle, except with negligible probability. Certainly, if some  $y' = y$ , the decryption simulator's response is correct, and if no  $y' = y$ , then the above arguments imply that the real decryption oracle would have rejected  $y$  with overwhelming probability.

From this, one would like to conclude that the decryption oracle does not help the adversary. However, this reasoning is invalid. Indeed, the adversary in the actual attack has access to  $S_G$  and  $S_H$ , along with the corresponding outputs of  $G$  and  $H$ , but does not have direct access to  $r^*$ ,  $G(r^*)$ ,  $s^*$ ,  $H(s^*)$ . Thus, the above decryption simulator has access to information that is not directly available to the adversary, and so we cannot argue that the invocations of the decryption oracle can simply be replaced by the decryption simulator. Indeed, if we give the decryption simulator access to  $r^*$ ,  $G(r^*)$ ,  $s^*$ ,  $H(s^*)$ , then the proof that  $x^*$  is well hidden, unless the adversary can invert  $f$ , is doomed to failure: if the decryption simulator needs to “know”  $r^*$  and  $s^*$ , then it must already “know”  $w^*$ , and so one cannot hope to use the adversary to compute something that the decryption simulator did not already know.

On closer observation, it is clear that the decryption simulator does not need to know  $s^*$ ,  $G(s^*)$ : if it is queried with a ciphertext  $y \neq y^*$  for which  $s = s^*$ , then it must be the case that  $t \neq t^*$ , which implies that  $r \neq r^*$ , and so  $c = 0^{k_1}$  with negligible probability. Thus, it is safe to reject all ciphertexts  $y$  such that  $s = s^*$ .

If one could make an analogous argument that the decryption simulator does not need to know  $r^*$ ,  $G(r^*)$ , we would be done. This is unfortunately not the case, as the following example illustrates.

The arguments in [BR2] simply do not take into account the random oracle queries made by the decryption oracle. All these arguments really show is that OAEP is secure against indifferent chosen ciphertext attack (IND-CCA1). We leave this for the reader to verify.

#### 4.1. An Example

Suppose that we have an algorithm that actually can invert  $f$ . Now of course, in this case, we will not be able to construct a counter-example to the security of OAEP, but we will argue that the proof technique fails. In particular, we show how to build an adversary that uses the  $f$ -inverting algorithm to break the cryptosystem, but it does so in such a way that no simulator given black box access to the adversary and its random oracle queries can use our adversary to compute  $f^{-1}(y^*)$  for a given value of  $y^*$ .

We now describe adversary. Upon obtaining the target ciphertext  $y^*$ , the adversary computes  $w^*$  using the algorithm for inverting  $f$ , and then extracts the corresponding values  $s^*$  and  $t^*$ . The adversary then chooses an arbitrary, non-zero  $\Delta \in \{0, 1\}^n$ , and computes:

$$\begin{aligned} s &= s^* \oplus (\Delta \parallel 0^{k_1}), \\ t &= t^* \oplus H(s^*) \oplus H(s), \\ w &= s \parallel t, \\ y &= f(w). \end{aligned}$$

It is easily verified that  $y$  is a valid encryption of  $x = x^* \oplus \Delta$ , and clearly  $y \neq y^*$ . So if the adversary submits  $y$  to the decryption oracle, he obtains  $x$ , from which he can then easily compute  $x^*$ .

This adversary clearly breaks the cryptosystem—in fact, its advantage is  $\frac{1}{2}$ . However, note in this attack, the adversary only queries the oracle  $H$  at the points  $s$  and  $s^*$ . It never queries the oracle  $G$  at all. In fact  $r = r^*$ , and the attack succeeds just where the gap in the proof was identified above.

What information has a simulator learned by interacting with the adversary as a black box? It has only learned  $s^*$  and  $s$  (and hence  $\Delta$ ). So it has learned the first  $n + k_1$  bits of the pre-image of  $y^*$ , but the last  $k_0$  remain a complete mystery to the simulator, and in general, they will not be easily computable from the first  $n + k_1$  bits. The simulator also has seen the value  $y$  submitted to the decryption oracle, but it does not seem likely that this can be used by the simulator to any useful effect. Thus, the adversary breaks the encryption scheme, and yet it is not at all clear how to use this adversary as a “black box” to invert the permutation.

## 5. Formal Evidence that the OAEP Construction Is Not Sound

In this section we present strong evidence that the OAEP construction is not sound. First, we show that if a special type of one-way trapdoor permutation  $f_0$  exists, then, in fact, we can construct another one-way trapdoor permutation  $f$  such that OAEP using  $f$  is *insecure*. Although we do not know how to construct such a special  $f_0$  explicitly, we can show that there is an oracle relative to which one exists. Thus, there is an oracle relative to which OAEP is insecure. This in turn implies that there is no standard “black box” security reduction for OAEP.

**Definition 1.** Consider a trapdoor permutation scheme whose permutation generator outputs a permutation, which we denote by  $f_0$ , along with the inverse permutation. We call such a generator *XOR-malleable* if the following property holds. There exists an efficient algorithm  $U$ , such that for infinitely many values of the security parameter,  $U(f_0, f_0(t), \delta) = f_0(t \oplus \delta)$  with non-negligible probability. Here, probability is taken over the random bits of the permutation generator, and random bit strings  $t$  and  $\delta$  in the domain  $\{0, 1\}^{k_0}$  of the generated permutation  $f_0$ .

**Theorem 1.** *If there exists an XOR-malleable one-way trapdoor permutation scheme, then there exists a one-way trapdoor permutation scheme such that when OAEP is instantiated with this scheme, the resulting encryption scheme is insecure (in the random oracle model).*

**Proof.** The proof is based on the example presented in Section 4.1.

Assume we are given an XOR-malleable one-way trapdoor permutation scheme whose generator outputs a permutation  $f_0$  (along with its inverse) on  $k_0$ -bit strings. Choose  $n > 0$ ,  $k_1 > 0$ , and set  $k = n + k_0 + k_1$ . Let  $f$  be the permutation on  $k$ -bit strings defined as follows: for  $s \in \{0, 1\}^{n+k_1}$ ,  $t \in \{0, 1\}^{k_0}$ , let  $f(s \parallel t) = s \parallel f_0(t)$ . This clearly defines a generator for a one-way trapdoor permutation scheme.

Let  $U$  be the algorithm that computes  $f_0(t \oplus \delta)$  from  $(f_0, f_0(t), \delta)$ .

Now consider the OAEP scheme that uses this  $f$  as its one-way trapdoor permutation, and uses the parameters  $k, n, k_0, k_1$  for the padding scheme.

Recall our notational conventions: any ciphertext  $y$  implicitly defines values  $w, s, t, r, z, x, c$ , and the target ciphertext  $y^*$  implicitly defines  $w^*, s^*, t^*, r^*, z^*, x^*, c^*$ .

We now describe the adversary. Upon obtaining the target ciphertext  $y^*$ , the adversary decomposes  $y^*$  as  $y^* = s^* \parallel f_0(t^*)$ . The adversary then chooses an arbitrary, non-zero  $\Delta \in \{0, 1\}^n$ , and computes

$$\begin{aligned} s &= s^* \oplus (\Delta \parallel 0^{k_1}), \\ v &= U(f_0, f_0(t^*), H(s^*) \oplus H(s)), \\ y &= s \parallel v. \end{aligned}$$

It is easily verified that  $y$  is a valid encryption of  $x = x^* \oplus \Delta$ , provided  $v = f_0(t^* \oplus H(s^*) \oplus H(s))$ , which by our assumption of XOR-malleability occurs with non-negligible probability. Indeed, we have

$$\begin{aligned} t &= t^* \oplus H(s^*) \oplus H(s), \\ r &= H(s) \oplus t \\ &= H(s^*) \oplus t^* \\ &= r^*, \\ z &= G(r) \oplus s \\ &= G(r^*) \oplus s^* \oplus (\Delta \parallel 0^{k_1}) \\ &= (x^* \oplus \Delta) \parallel 0^{k_1}. \end{aligned}$$

So if the adversary submits  $y$  to the decryption oracle, he obtains  $x$ , from which he can then easily compute  $x^*$ .

This adversary clearly breaks the cryptosystem.  $\square$

Note that in the above attack,  $r = r^*$  and the adversary never explicitly queried  $G$  at  $r$ , but was able to “hijack”  $G(r)$  from the encryption oracle—this is the essence of the problem with OAEP.

Note that this attack also shows that the scheme is malleable with respect to chosen plaintext attack: the adversary is able to construct a ciphertext  $y$  that is an encryption of a message  $x$ , where  $x$  is related in some non-trivial way to  $x^*$ , and is able to do this without interacting with the decryption oracle at all. Note that in [BDPR], it was shown, using a rather artificial counter-example, that security in the sense of indistinguishability against indifferent chosen ciphertext attack (IND-CCA1) does not imply non-malleability against chosen plaintext attack (NM-CPA). Our results show that OAEP provides a more natural (but not so general) counter-example.

Of course, one might ask if it is at all reasonable to believe that XOR-malleable one-way trapdoor permutations exist at all. First, note that the standard RSA function is a one-way trapdoor permutation that is not XOR-malleable, but is still malleable in a very

similar way: given  $\alpha = (a^e \bmod N)$  and  $(b \bmod N)$ , we can compute  $((ab)^e \bmod N)$  as  $(\alpha \cdot (b^e \bmod N))$ . Thus, we can view the RSA function itself as a kind of malleable one-way trapdoor permutation, but where XOR is replaced by multiplication mod  $N$ . In fact, one could modify the OAEP scheme so that  $t$ ,  $H(s)$ , and  $r$  are numbers mod  $N$ , and instead of the relation  $t = H(s) \oplus r$ , we would use the relation  $t = H(s) \cdot r \bmod N$ . It would seem that if there were a proof of security for OAEP, then it should go through for this variant of OAEP as well. Yet, this variant of OAEP is clearly insecure, even though the underlying trapdoor permutation is presumably one way.

Another example is exponentiation in a finite abelian group. For a group element  $g$ , the function mapping  $a$  to  $g^a$  is malleable with respect to both addition *and* multiplication modulo the order of  $g$ . Although for appropriate choices of groups this function is a reasonable candidate for a one-way permutation, it does not have a trapdoor.

Beyond this, we prove a relativized result. We recall the usual machinery.

A *query machine*  $M$  is a Turing machine with a special *query tape*, and two special states *call* and *return*. An oracle  $\mathcal{O}$  is a function mapping bit strings to bit strings. The computation  $M^{\mathcal{O}}$  that machine  $M$  performs with oracle  $\mathcal{O}$  proceeds as follows: whenever the machine enters the *call* state, the function  $\mathcal{O}$  is evaluated on the contents on the query tape, the query tape is overwritten with the output of  $\mathcal{O}$ , and the machine is replaced in the *return* state. In defining notions of security in such a relativized setting, an “adversary” is a probabilistic, polynomial time query machine; note that we assume that such a machine *always* halts after a specific, polynomial number of steps.

**Theorem 2.** *There exists an oracle, relative to which XOR-malleable one-way trapdoor permutations exist.*

This theorem provides some evidence that the notion of an XOR-malleable one-way trapdoor permutation scheme is not a priori vacuous.

Also, Theorems 1 and 2 imply the following.

**Corollary 1.** *There exists an oracle, relative to which the OAEP construction is insecure.*

We should stress the implications of this corollary.

Normally, to prove the security of a cryptographic system, one proves this via a “black box” security reduction from solving the underlying “hard” problem to breaking the cryptographic system. Briefly, such a reduction for a cryptosystem based on a general trapdoor permutation scheme would be an efficient, probabilistic algorithm that inverts a permutation  $f$  on a random point, given oracle access to an adversary that successfully breaks the cryptosystem (instantiated with  $f$ ) and the permutation  $f$ . It should work for *all* adversaries and *all* permutations, even ones that are not efficiently computable, or even computable at all. Whatever the adversary’s advantage is in breaking the cryptosystem, the success probability of the inversion algorithm should not be too much smaller.

We do not attempt to make a more formal or precise definition of a black box security reduction, but it should be clear that any such reduction would imply security relative to any oracle. So Corollary 1 implies that there is no black box security reduction for OAEP.

**Proof of Theorem 2.** The rest of this section is devoted to a proof of this theorem.

We first begin by describing a *probability space* of oracles.

Let  $W$  be chosen at random from the set of all functions on  $\{0, 1\}^*$  such that for any  $n \geq 0$ ,  $W$  restricted to  $\{0, 1\}^n$  acts as a permutation on  $\{0, 1\}^n$ .

Let  $F$  be a family of permutations, such that for every positive integer  $k_0$ , and for every  $pk \in \{0, 1\}^{k_0}$ ,  $F_{pk}$  is a random permutation on  $\{0, 1\}^{k_0}$ .

Our oracle  $\mathcal{O}$  responds to four different types of queries:

- $\mathcal{O}_1$ : Given  $sk \in \{0, 1\}^*$ , return  $pk = W(sk)$ .
- $\mathcal{O}_2$ : Given  $pk, \delta \in \{0, 1\}^*$  with  $|pk| = |\delta|$ , return  $F_{pk}(\delta)$ .
- $\mathcal{O}_3$ : Given  $sk, v \in \{0, 1\}^*$  with  $|sk| = |v|$ , return  $F_{W(sk)}^{-1}(v)$ .
- $\mathcal{O}_4$ : Given  $pk, v, \delta \in \{0, 1\}^*$  with  $|pk| = |v| = |\delta|$ , return  $F_{pk}(F_{pk}^{-1}(v) \oplus \delta)$ .

The idea here is that the function  $W$  can be used to generate public key/secret key pairs for a *random* trapdoor permutation. If one chooses secret key  $sk^*$  at random, then  $pk^* = W(sk^*)$  is the corresponding public key. This can be accomplished using the  $\mathcal{O}_1$  query.

The  $\mathcal{O}_2$  query can be used to compute the permutation in the forward direction using  $pk^*$ . Using  $\mathcal{O}_3$  with the trapdoor  $sk^*$ , one can compute the inverse permutation.

Query  $\mathcal{O}_4$  is what makes our trapdoor permutation XOR-malleable.

Note that the above is not really a well-defined probability space—at least, in the sense of elementary probability theory—since the number of oracles is uncountable. To make this technically precise, we define a sequence of finite probability spaces  $\{\mathcal{S}_m\}_{m \geq 1}$ , where  $\mathcal{S}_m$  defines a distribution of oracles defined on a finite domain  $\mathcal{D}_m$ , such that for all  $m \geq 1$ ,  $\mathcal{D}_{m+1} \supseteq \mathcal{D}_m$ , and the distribution induced by sampling an oracle from  $\mathcal{S}_{m+1}$  and restricting to  $\mathcal{D}_m$  is the same as  $\mathcal{S}_m$ . If we want to examine the probabilistic behavior of a specific query machine  $M$  on a specific input size  $k_0$  with respect to a “random” oracle, we simply choose an  $m$  large enough so that this machine on this input size restricts its oracle queries to  $\mathcal{D}_m$ , and then choose an oracle from  $\mathcal{D}_m$ . The behavior is invariant of the particular choice of  $m$ .

To make a rigorous and precise proof, we state and prove the following very simple, but useful lemma, which we also use for some other proofs in the paper.

**Lemma 1.** *Let  $E, E'$ , and  $F$  be events defined on a probability space such that  $\Pr[E \wedge \neg F] = \Pr[E' \wedge \neg F]$ . Then we have*

$$|\Pr[E] - \Pr[E']| \leq \Pr[F].$$

**Proof.** We have

$$\Pr[E] = \Pr[E \wedge F] + \Pr[E \wedge \neg F]$$

and

$$\Pr[E'] = \Pr[E' \wedge F] + \Pr[E' \wedge \neg F].$$

Subtracting these two equations, we have

$$|\Pr[E] - \Pr[E']| = |\Pr[E \wedge F] - \Pr[E' \wedge F]| \leq \Pr[F]. \quad \square$$

Theorem 2 will now follow from the following lemma.

**Lemma 2.** *Any adversary that makes at most  $m$  oracle queries, succeeds in inverting a permutation on  $k_0$  bits with probability  $O(m^2/2^{k_0})$ . Here, the probability is taken over the random choice of the oracle, the random choice of the secret key, the random choice of the element to be inverted, and the random choices made by the adversary.*

**Proof.** We can assume that whenever the adversary makes an  $\mathcal{O}_3$  query with a given value of  $sk$ , he has previously made an  $\mathcal{O}_1$  query with the same value. Any adversary can be modified to conform to this convention, increasing  $m$  by a constant factor.

Let  $\mathbf{G}_0$  be the original attack game. Let  $(sk^*, pk^*)$  be secret key/public key of the generated trapdoor permutation. Let  $f_0 = F_{pk^*}$  denote this permutation, and assume that it is a permutation on  $\{0, 1\}^{k_0}$ . Let  $v^* \in \{0, 1\}^{k_0}$  be the string whose  $f_0$ -inverse the adversary is trying to compute, and let  $t^*$  be this inverse. Let  $S_0$  denote the event that the adversary succeeds.

We consider a modified attack game,  $\mathbf{G}_1$ , defined as follows. Game  $\mathbf{G}_1$  is exactly the same as  $\mathbf{G}_0$ , except that in game  $\mathbf{G}_1$ , if the adversary ever inputs  $sk^*$  to the  $\mathcal{O}_1$  oracle, which can be detected by testing if the output equals  $pk^*$ , it politely halts. Conceptually,  $\mathbf{G}_0$  and  $\mathbf{G}_1$  are games that operate on the same probability space, but the rules of the game are different. Let  $S_1$  be the event that the adversary succeeds in  $\mathbf{G}_1$ , and let  $F_0$  be the event that in game  $\mathbf{G}_0$  (or, equivalently,  $\mathbf{G}_1$ ), the adversary inputs  $sk^*$  to the  $\mathcal{O}_1$  oracle. Then Lemma 1 applies with  $(S_0, S_1, F_0)$ , and, moreover, it is clear that  $\Pr[F_0] = O(m/2^{k_0})$ . Therefore,

$$|\Pr[S_0] - \Pr[S_1]| = O(m/2^{k_0}). \quad (12)$$

By construction, the only information the adversary learns about  $f_0$  in game  $\mathbf{G}_1$  is through its initial input  $v^*$ , and calls to the oracles  $\mathcal{O}_2$  and  $\mathcal{O}_4$ .

We now define a game  $\mathbf{G}'_1$  that is completely equivalent to game  $\mathbf{G}_1$ , but formulated in a slightly different way. This is a rather technical “bridging” step that will facilitate more drastic modifications in game  $\mathbf{G}_2$  below. In this game we process  $\mathcal{O}_1$  and  $\mathcal{O}_3$  queries just as in game  $\mathbf{G}_1$ . Also, we process  $\mathcal{O}_2$  and  $\mathcal{O}_4$  queries with  $pk \neq pk^*$  just as in game  $\mathbf{G}_1$ . However, we process  $\mathcal{O}_2$  and  $\mathcal{O}_4$  queries with  $pk = pk^*$  differently.

At the outset of the game, we generate a vector  $(v_1, \dots, v_{2^{k_0}})$  that is a random permutation of  $\{0, 1\}^{k_0}$ . We also generate a sequence  $(s_1, \dots, s_m)$ , where each  $s_i$  is uniformly and independently drawn from  $\{0, 1\}^{k_0}$ .

Now, we also define sequences  $(t_0, t_1, t_2, \dots)$  and  $(D_1, D_2, \dots)$  incrementally as the game proceeds. Each  $t_i$  is a bit string of length  $k_0$ . Each  $D_i$  is a pair  $(j, \delta)$ , where  $j$  is an integer and  $\delta$  is a bit string of length  $k_0$ . We maintain two counters,  $a$  and  $b$ , and it will always be the case that  $D_i$  is defined for  $1 \leq i \leq a$ , and that  $t_j$  is defined for  $0 \leq j \leq b$ .

Conceptually, the permutation  $f_0$  is implicitly defined by  $f_0(t_j \oplus \delta) = v_i$ , where  $D_i = (j, \delta)$ ,  $1 \leq i \leq a$  and  $0 \leq j \leq b$ ; moreover,  $\delta$  is a value that is “known” by the adversary, while the values  $t_j$  for  $j > 0$  are not “known” by the adversary.

At the outset of the game, we set  $a = 1$ ,  $b = 1$ ,  $t_0 = 0^{k_0}$ ,  $t_1 = s_1$ , and  $D_1 = (1, 0^{k_0})$ . We also set  $v^* = v_1$  and  $t^* = t_1$ . So conceptually, this defines  $f_0(t^*) = v^*$ . We give  $v^*$  to the adversary—this is the element whose inverse the adversary is supposed to compute. At the end of the game, the adversary succeeds if its output is equal to  $t^*$ .

Now consider an  $\mathcal{O}_2$  query with input  $(pk^*, \delta)$ . We first test if there is an  $i$  with  $1 \leq i \leq a$ , such that  $D_i = (0, \delta)$ . If so, we let the oracle output the corresponding  $v_i$ . Otherwise, we test if there exists an  $i$  with  $1 \leq i \leq a$ ,  $D_i = (j, \tilde{\delta})$ , and  $1 \leq j \leq b$ , such that

$$\delta = t_j \oplus \tilde{\delta}. \quad (13)$$

If so, we let the oracle output the corresponding  $v_i$ . Otherwise, we increment  $a$ , and set  $D_a = (0, \delta)$ , and output  $v_a$ .

Now consider an  $\mathcal{O}_4$  query with input  $(pk^*, v, \delta)$ .

*Case 1:  $v = v_i$  for some  $1 \leq i \leq a$ .*

- If  $D_i = (0, \tilde{\delta})$  for some  $\tilde{\delta}$ , then we process this  $\mathcal{O}_4$  query just like an  $\mathcal{O}_2$  query with input  $(pk^*, \delta \oplus \tilde{\delta})$ .
- Otherwise,  $D_i = (j, \tilde{\delta})$  for some  $1 \leq j \leq b$  and some  $\tilde{\delta}$ . If there exists an  $i'$ , with  $1 \leq i' \leq a$  such that  $D_{i'} = (j, \delta \oplus \tilde{\delta})$ , then we output  $v_{i'}$ .
- Otherwise, we test there exists an  $i'$ , with  $1 \leq i' \leq a$ ,  $D_{i'} = (j', \tilde{\delta}')$ ,  $0 \leq j' \leq b$ , and  $j' \neq j$ , such that

$$t_j \oplus \tilde{\delta} \oplus \delta = t_{j'} \oplus \tilde{\delta}'. \quad (14)$$

(Note that we allow  $j' = 0$ .) If so, we output  $v_{i'}$ .

- Otherwise, we increment  $a$ , set  $D_a = (j, \delta \oplus \tilde{\delta})$ , and output  $v_a$ .

*Case 2:  $v \neq v_i$  for all  $1 \leq i \leq a$ .*

- Let

$$T = \{t_j \oplus \tilde{\delta} : D_i = (j, \tilde{\delta}), 1 \leq i \leq a\}.$$

We increment  $b$ , and then we define  $t_b$  as follows. First, we test if

$$s_b \in T. \quad (15)$$

If so, we choose  $t_b$  at random from  $\{0, 1\}^{k_0} \setminus T$ ; otherwise, we set  $t_b = s_b$ .

- Next, we increment  $a$ . Let  $a'$  be such that  $v_{a'} = v$ . By construction, we have  $a' \geq a$ . We now swap  $v_a$  and  $v_{a'}$ . Next, we define  $D_a = (b, 0^{k_0})$ , and then perform the actions in case 1.

Let  $S'_1$  be the event that the adversary succeeds in game  $\mathbf{G}'_1$ . It is straightforward to verify that

$$\Pr[S_1] = \Pr[S'_1]. \quad (16)$$

Now we define a game  $\mathbf{G}_2$  that is just like  $\mathbf{G}'_1$ , except that we simply behave *as if* the tests (13)–(15) *always fail*. Conceptually, we view  $\mathbf{G}'_1$  and  $\mathbf{G}_2$  as operating on the same probability space; in particular, the vectors  $(v_1, \dots, v_{2^{k_0}})$  and  $(s_1, \dots, s_m)$  are the same in both games. Note that in game  $\mathbf{G}_2$  it no longer makes sense to speak of an implicitly defined permutation  $f_0$ ; however, we can still define the event  $S_2$  that the adversary outputs  $t^*$ . Let  $F_2$  be the event that in game  $\mathbf{G}_2$ , one of these tests (13)–(15) passes (even though this is ignored in game  $\mathbf{G}_2$ ).

A key observation is that in game  $\mathbf{G}_2$  the values  $v_i$  seen by the adversary, and hence the inputs to the oracles, are *independent* of the values  $s_j$ , and that  $t_j = s_j$  for  $0 < j \leq b$ . This follows by simple inspection, keeping in mind that the only places where game  $\mathbf{G}'_1$  was influenced at all by the  $s_j$  or  $t_j$  values were in the tests (13)–(15), and in game  $\mathbf{G}_2$  these tests are defined to fail unconditionally. So to bound  $\Pr[F_2]$ , it suffices to bound  $\Pr[F_2]$ , conditioning on arbitrary, fixed values of  $(v_1, \dots, v_{2^{k_0}})$  and arbitrary, fixed values of the adversary's coin tosses. In this conditional space, the event  $F_2$  is equivalent to the event that one of  $O(m^2)$  equations holds, where each equation is of the form  $s_j = \delta$ , or of the form  $s_j \oplus s_{j'} = \delta$ , where  $j \neq j'$ . In this conditional space, all the values  $\delta$  are constants, but because of the above observation, while the values  $s_j$  are uniformly and independently distributed. Each equation is satisfied with probability  $1/2^{k_0}$ , and hence  $\Pr[F_2] = O(m^2/2^{k_0})$ .

One also sees that  $\Pr[S_2 \wedge \neg F_2] = \Pr[S'_1 \wedge \neg F_2]$ , since both games proceed *identically* up until the first point where  $F_2$  occurs. So we apply Lemma 1 with  $(S'_1, S_2, F_2)$ , and we obtain

$$|\Pr[S'_1] - \Pr[S_2]| = O(m^2/2^{k_0}). \quad (17)$$

Finally, observe that

$$\Pr[S_2] = O(1/2^{k_0}), \quad (18)$$

since in this game, the value  $t^*$  is independent of the adversary's view.

So, finally, Lemma 2 follows from (12), (16), (17), and (18).  $\square$

There are a few more details required to finish the proof of Theorem 2. The reason we are not done is that we want to show that there exists a fixed oracle relative to which the implied permutation is one way. These details are mostly straightforward, but slightly tedious.

Let  $\{M_i\}_{i \geq 1}$  be a complete enumeration of probabilistic, polynomial time query machines. Fix numbers  $0 < \beta < \alpha < 1$ . Lemma 2 implies that for each  $i \geq 1$ , there exists an integer  $k(i)$  such that whenever  $k_0 \geq k(i)$ , machine  $M_i^{\mathcal{O}}$  wins the inversion game on  $k_0$ -bit permutations with probability at most  $2^{-\alpha k_0}$ . Here, the probability space includes the random choice of oracle  $\mathcal{O}$ ; note that to be technically correct,  $\mathcal{O}$  is sampled from an appropriate space  $\mathcal{S}_m$ , where  $m$  depends on  $i$  and  $k_0$ .

Let  $c, d$  be positive numbers whose values will be determined below, and for  $i \geq 1$  define  $k'(i) = \max\{k(i), \lceil c \log i + d \rceil\}$ .

For positive integers  $i$  and  $k_0$ , we say that oracle  $\mathcal{O}$  is  $(i, k_0)$ -good if machine  $M_i^{\mathcal{O}}$  wins the inversion game on  $k_0$ -bit permutations with probability at most  $2^{-\beta k_0}$ , where the probability is conditioned on the choice of  $\mathcal{O}$ ; otherwise, we say that  $\mathcal{O}$  is  $(i, k_0)$ -bad. By Markov's inequality, for any  $i \geq 1$  and any  $k_0 \geq k'(i)$ , the probability that a random  $\mathcal{O}$  is  $(i, k_0)$ -bad is at most  $2^{-\gamma k_0}$ , where  $\gamma = \alpha - \beta$ .

We say that a globally defined oracle  $\mathcal{O}$  is *good* if it is  $(i, k_0)$ -good for all  $i \geq 1$  and  $k_0 \geq k'(i)$ ; otherwise, we say that  $\mathcal{O}$  is *bad*. Note that we cannot talk about the probability that a random oracle is bad, since we have not defined a probability space over all globally defined oracles.

For a positive integer  $B$ , consider all integers  $i, k_0$  with  $1 \leq i \leq B$  and  $1 \leq k_0 \leq B$ . For this  $B$ , there is a domain  $\mathcal{D}_{m(B)}$  which is sufficiently large so that all the oracle queries



made by all machines  $M_i$  running on  $k_0$ -bit permutations for  $i, k_0$  bounded by  $B$  lie in  $\mathcal{D}_{m(B)}$ . We can assume that  $m(B) \leq m(B+1)$  for all  $B \geq 1$ . We then say that an oracle defined on  $\mathcal{D}_{m(B)}$  is *B-good* if it is  $(i, k_0)$ -good for all  $1 \leq i \leq B$  and  $k'(i) \leq k_0 \leq B$ ; otherwise, we say that it is *B-bad*. Let  $p(B)$  be the probability that a random oracle  $\mathcal{O}$  chosen from  $\mathcal{S}_{m(B)}$  is *B-bad*. Then we have

$$p(B) \leq \sum_{i \geq 1} \sum_{k_0 \geq k'(i)} 2^{-\gamma k_0} \leq (1/(1 - 2^{-\gamma})) \sum_{i \geq 1} 2^{-\gamma(c \log i + d)}.$$

From this, one sees that  $p(B) < 1$  for appropriate choices of  $c$  and  $d$ , and for all  $B \geq 1$ .

So we know that for all  $B \geq 1$ , there exist *B-good* oracles. From this, we want to show there exists a globally defined good oracle. To do this, we need the following technical lemma.

**Lemma 3.** *Let  $\{\mathcal{G}_i\}_{i \geq 1}$  be a sequence of non-empty, finite collections of sets with the following property: for all  $i \geq 1$ , all  $S \in \mathcal{G}_i$ , and all  $1 \leq j \leq i$ , there exists a unique  $S(j) \in \mathcal{G}_j$  such that  $S \supseteq S(j)$ . Then there exists an increasing chain*

$$S_1 \subseteq S_2 \subseteq S_3 \subseteq \dots,$$

with  $S_i \in \mathcal{G}_i$  for each  $i \geq 1$ .

**Proof.** We begin by proving the following claim:

There exists  $S \in \mathcal{G}_1$  such that for all  $i > 1$ , there exists  $S' \in \mathcal{G}_i$  with  $S' \supseteq S$ .

To prove this claim, assume that it is false. Then for all  $S \in \mathcal{G}_1$ , there exists  $i(S)$  such that  $\mathcal{G}_{i(S)}$  does not contain a superset of  $S$ . Since  $\mathcal{G}_1$  is finite, we can set  $i_{\max} = \max\{i(S) : S \in \mathcal{G}_1\}$ . Now,  $\mathcal{G}_{i_{\max}}$  is non-empty, so let  $S'$  be an element of  $\mathcal{G}_{i_{\max}}$ . We have a decreasing chain

$$S' = S'(i_{\max}) \supseteq S'(i_{\max} - 1) \supseteq \dots \supseteq S'(1) = S,$$

with  $S'(j) \in \mathcal{G}_j$  for  $1 \leq j \leq i_{\max}$ . However, then it follows that  $\mathcal{G}_{i(S)}$  contains a superset, namely  $S'(i(S))$ , of  $S$ . That proves the claim.

We now prove the lemma using this claim. Choose an  $S_1 \in \mathcal{G}_1$  whose existence is guaranteed by the claim. Now consider the sequence  $\{\mathcal{G}'_i\}_{i \geq 2}$  of collections of sets, where for all  $i \geq 2$ ,  $\mathcal{G}'_i$  consists of those elements of  $\mathcal{G}_i$  that contain  $S_1$ . By the claim,  $\mathcal{G}'_i$  is non-empty for all  $i \geq 2$ . Moreover, it is easily verified that  $\{\mathcal{G}'_i\}_{i \geq 2}$  satisfies the other hypotheses of the lemma (this relies on the uniqueness of the restriction map sending  $S \in \mathcal{G}_i$  to  $S(j) \in \mathcal{G}_j$  for all  $1 \leq j \leq i$ ). So we can apply the claim to  $\{\mathcal{G}'_i\}_{i \geq 2}$ , obtaining an  $S_2 \in \mathcal{G}'_2$ . Iterating this procedure, we can build up an increasing chain

$$S_1 \subseteq S_2 \subseteq S_3 \subseteq \dots,$$

with  $S_i \in \mathcal{G}_i$  for each  $i \geq 1$ . □

Now, for  $B \geq 1$ , let  $\mathcal{G}_B$  be the set of *B-good* oracles defined on  $\mathcal{D}_{m(B)}$ . It is easy to verify that the sequence  $\{\mathcal{G}_B\}_{B \geq 1}$  satisfies the hypothesis of Lemma 3, and so there

is a sequence of oracles  $\{\mathcal{O}^{(B)}\}_{B \geq 1}$ , where for each  $B \geq 1$ ,  $\mathcal{O}^{(B)} \in \mathcal{G}_B$  and  $\mathcal{O}^{(B+1)}$  is an extension of  $\mathcal{O}^{(B)}$ . We then set  $\mathcal{O} = \bigcup_{B \geq 1} \mathcal{O}^{(B)}$ , and it is easy to see that this is a globally defined good oracle.

Thus, there exists a good oracle  $\mathcal{O}$ . Such a good oracle then satisfies the conditions of Theorem 2; namely, for all  $i \geq 1$ , for all  $k_0 \geq k'(i)$ , the probability that  $M_i^{\mathcal{O}}$  wins the inversion game on  $k_0$ -bit permutations is at most  $2^{-\beta k_0}$ .

That completes the proof of Theorem 2.

*Remark.* The proof of Lemma 2 is quite similar to proofs of lower bounds for the discrete logarithm problem presented in [Sh1] (in particular, Theorem 2 in that paper). There are a few technical differences, and the proof we have presented here is much more complete. We should also point out that Lemma 2 is fairly tight, in the sense that the well-known baby step/giant step attack for the discrete logarithm problem (see Section 3.6.2 of [MvOV]) can be easily adapted to inverting XOR-malleable permutations, provided the algorithm  $U$  is highly reliable.

*Remark.* The argument showing that Lemma 2 implies Theorem 2 seems overly technical, especially since one cannot use elementary probability theory to model properly the notion of a “random” oracle in this context. We could have made a syntactically simpler argument if we used more advanced notions of measure, but we chose to make a completely self-contained, elementary argument.

## 6. OAEP+

We now describe the OAEP+ encryption scheme, which is just a slight modification of the OAEP scheme.

The general scheme makes use of a one-way trapdoor permutation. Let  $f$  be the permutation, acting on  $k$ -bit strings, and let  $g$  be its inverse. The scheme also makes use of two parameters  $k_0$  and  $k_1$ , which should satisfy  $k_0 + k_1 < k$ . It should also be the case that  $2^{-k_0}$  and  $2^{-k_1}$  are negligible quantities. The scheme encrypts messages  $x \in \{0, 1\}^n$ , where  $n = k - k_0 - k_1$ .

The scheme also makes use of three functions:

$$\begin{aligned} G: \{0, 1\}^{k_0} &\rightarrow \{0, 1\}^n, \\ H': \{0, 1\}^{n+k_0} &\rightarrow \{0, 1\}^{k_1}, \\ H: \{0, 1\}^{n+k_1} &\rightarrow \{0, 1\}^{k_0}. \end{aligned}$$

These three functions will be modeled as independent random oracles in the security analysis.

We describe the key generation, encryption, and decryption algorithms of the scheme.

*Key Generation.* This simply runs the generator for the one-way trapdoor permutation scheme, obtaining  $f$  and  $g$ . The public key is  $f$ , and the private key is  $g$ .

*Encryption.* Given a plaintext  $x$ , the encryption algorithm randomly chooses  $r \in \{0, 1\}^{k_0}$ , and then computes

$$s \in \{0, 1\}^{n+k_1}, \quad t \in \{0, 1\}^{k_0}, \quad w \in \{0, 1\}^k, \quad y \in \{0, 1\}^k$$

as follows:

$$s = (G(r) \oplus x) \parallel H'(r \parallel x), \quad (19)$$

$$t = H(s) \oplus r, \quad (20)$$

$$w = s \parallel t, \quad (21)$$

$$y = f(w). \quad (22)$$

The ciphertext is  $y$ .

*Decryption.* Given a ciphertext  $y$ , the decryption algorithm computes

$$\begin{aligned} w \in \{0, 1\}^k, \quad s \in \{0, 1\}^{n+k_1}, \quad t \in \{0, 1\}^{k_0}, \\ r \in \{0, 1\}^{k_0}, \quad x \in \{0, 1\}^n, \quad c \in \{0, 1\}^{k_1} \end{aligned}$$

as follows:

$$w = g(y), \quad (23)$$

$$s = w[0 \cdots n + k_1 - 1], \quad (24)$$

$$t = w[n + k_1 \cdots k], \quad (25)$$

$$r = H(s) \oplus t, \quad (26)$$

$$x = G(r) \oplus s[0 \cdots n - 1], \quad (27)$$

$$c = s[n \cdots n + k_1 - 1]. \quad (28)$$

If  $c = H'(r \parallel x)$ , then the algorithm outputs the cleartext  $x$ ; otherwise, the algorithm *rejects* the ciphertext, and does not output a cleartext.

**Theorem 3.** *If the underlying trapdoor permutation scheme is one way, then OAEP+ is secure against adaptive chosen ciphertext attack in the random oracle model.*

**Proof.** The remainder of this section is devoted to the proof of this theorem. We start with some notations and conventions.

Let  $A$  be an adversary, and let  $\mathbf{G}_0$  be the original attack game. Let  $b$  and  $\hat{b}$  be as defined in Section 2.1, and let  $S_0$  be the event that  $b = \hat{b}$ . To prove the theorem, it will suffice to show that  $|\Pr[S_0] - \frac{1}{2}|$  is negligible.

Let  $q_G$ ,  $q_H$ , and  $q_{H'}$  bound the number of queries made by  $A$  to the oracles  $G$ ,  $H$ , and  $H'$ , respectively, and let  $q_D$  bound the number of decryption oracle queries.

We assume without loss of generality that whenever  $A$  makes a query of the form  $H'(r \parallel x)$ , for any  $r \in \{0, 1\}^{k_0}$ ,  $x \in \{0, 1\}^n$ , then  $A$  has previously made the query  $G(r)$ .

We shall prove the theorem by showing that

$$|\Pr[S_0] - \frac{1}{2}| \leq \text{InvAdv}(A') + (q_{H'} + q_D)/2^{k_1} + (q_D + 1)q_G/2^{k_0}, \quad (29)$$

where  $\text{InvAdv}(A')$  is the success probability that a particular adversary  $A'$  has in breaking the one-way trapdoor permutation scheme on  $k$ -bit inputs. The time and space requirements of  $A'$  are related to those of  $A$  as follows:

$$\text{Time}(A') = O(\text{Time}(A) + q_G q_H T_f + (q_G + q_{H'} + q_H + q_D)k); \quad (30)$$

$$\text{Space}(A') = O(\text{Space}(A) + (q_G + q_{H'} + q_H)k). \quad (31)$$

Here,  $T_f$  is the time required to compute  $f$ , and space is measured in bits of storage. These complexity estimates assume a standard random-access model of computation.

Any ciphertext  $y$  implicitly defines values  $w, s, t, r, x, c$  via the decryption equations (23)–(28). Let  $y^*$  denote the target ciphertext, and let  $w^*, s^*, t^*, r^*, x^*, c^*$  be the corresponding implicitly defined values for  $y^*$ . Note that  $x^* = x_b$  and  $c^* = H'(r^* \parallel x^*)$ .

We define sets  $S_G$  and  $S_H$ , as in Section 4, as follows. Let  $S_G$  be the set of values  $r$  at which  $G$  was queried by  $A$ . Also, let  $S_H$  be the set of values  $s$  at which  $H$  was queried by  $A$ . Additionally, define  $S_{H'}$  to be the set of pairs  $(r, x)$  such that  $H'$  was queried at  $r \parallel x$  by  $A$ . We view these sets as growing incrementally as  $A$ 's attack proceeds—elements are added to these only when a random oracle is queried by  $A$ .

We also define  $A$ 's *view* as the sequence of random variables

$$\text{View} = \langle X_0, X_1, \dots, X_{q_G + q_{H'} + q_H + q_D + 1} \rangle,$$

where  $X_0$  consists of  $A$ 's coin tosses and the public key of the encryption scheme, and where each  $X_i$  for  $i \geq 1$  consists of a response to either a random oracle query, a decryption oracle query, or the encryption oracle query. The  $i$ th such query is a function of  $\langle X_0, \dots, X_{i-1} \rangle$ . The adversary's final output  $\hat{b}$  is a function of  $\text{View}$ . At any fixed point in time,  $A$  has made some number, say  $m$ , queries, and we define

$$\text{CurrentView} = \langle X_0, \dots, X_m \rangle.$$

Our overall strategy for the proof is as follows. We define a sequence  $\mathbf{G}_1, \mathbf{G}_2, \dots, \mathbf{G}_5$  of modified attack games. Each of the games  $\mathbf{G}_0, \mathbf{G}_1, \dots, \mathbf{G}_5$  operate on the same underlying probability space. In particular, the public key and private key of the cryptosystem, the coin tosses of  $A$ , the values of the random oracles  $G, H', H$ , and the hidden bit  $b$  take on *identical* values across all games. Only some of the rules defining how the view is computed differ from game to game. For any  $1 \leq i \leq 5$ , we let  $S_i$  be the event that  $b = \hat{b}$  in game  $\mathbf{G}_i$ . Our strategy is to show that for  $1 \leq i \leq 5$ , the quantity  $|\Pr[S_{i-1}] - \Pr[S_i]|$  is negligible. Also, it will be evident from the definition of game  $\mathbf{G}_5$  that  $\Pr[S_5] = \frac{1}{2}$ , which will imply that  $|\Pr[S_0] - \frac{1}{2}|$  is negligible.

In games  $\mathbf{G}_1, \mathbf{G}_2$ , and  $\mathbf{G}_3$  we incrementally modify the decryption oracle, so that in game  $\mathbf{G}_3$  the modified decryption oracle operates without using the trapdoor for  $f$  at all. In games  $\mathbf{G}_4$  and  $\mathbf{G}_5$  we modify the encryption oracle, so that in game  $\mathbf{G}_5$  the hidden bit  $b$  is completely independent of  $\text{View}$ .

**Game  $\mathbf{G}_1$ .** Now we modify game  $\mathbf{G}_0$  to define a new game  $\mathbf{G}_1$ .

We modify the decryption oracle as follows. Given a ciphertext  $y$ , the new decryption oracle computes  $w, s, t, r, x, c$  as usual. If the old decryption oracle rejects, so does the new one. However, the new decryption oracle also rejects if  $(r, x) \notin S_{H'}$ . More precisely, if the new decryption oracle computes  $r$  via (26), and finds that  $r \notin S_G$ , then it rejects right away, without ever querying  $G(r)$ ; if  $r \in S_G$ , then  $x$  is computed, but if  $(r, x) \notin S_{H'}$ , it rejects without querying  $H'(r \parallel x)$ . Recall that by convention, if  $A$  queried  $H'(r \parallel x)$ , it already queried  $G(r)$ . One sees that in game  $\mathbf{G}_1$  the decryption oracle never queries  $G$  or  $H'$  at points other than those at which  $A$  did.

Let  $F_1$  be the event that a ciphertext is rejected in  $\mathbf{G}_1$  that would not have been rejected under the rules of game  $\mathbf{G}_0$ .

Consider a ciphertext  $y$  submitted to the decryption oracle. Suppose that the encryption oracle has already been queried before this decryption oracle query (and  $y \neq y^*$ , of course). If  $r = r^*$  and  $x = x^*$ , then we must have  $c \neq c^*$ ; in this case, however, we will surely reject under the rules of game  $\mathbf{G}_0$ . So we assume that  $r \neq r^*$  or  $x \neq x^*$ . Now, the encryption oracle has made the query  $H'(r^* \parallel x^*)$ , but not  $H'(r \parallel x)$ , since  $(r, x) \neq (r^*, x^*)$ .

From the above considerations, we conclude that regardless of whether the encryption oracle has already been queried or not, if  $A$  has not made the query  $H'(r \parallel x)$ , the value of  $H'(r \parallel x)$  is independent of  $\text{CurrentView}$ , and, hence, is independent of  $c$ , which is a function of  $\text{CurrentView}$  and  $H$ . Therefore, the probability that  $c = H'(r \parallel x)$  is  $1/2^{k_1}$ .

It follows that  $\Pr[F_1] \leq q_D/2^{k_1}$ . Moreover, it is clear by construction that  $\Pr[S_0 \wedge \neg F_1] = \Pr[S_1 \wedge \neg F_1]$ , since the two games proceed identically unless the event  $F_1$  occurs; that is, the value of  $\text{View}$  is the same in both games, provided  $F_1$  does not occur. So applying Lemma 1 with  $(S_0, S_1, F_1)$ , we have

$$|\Pr[S_0] - \Pr[S_1]| \leq q_D/2^{k_1}. \quad (32)$$

**Game  $\mathbf{G}_2$ .** Now we modify game  $\mathbf{G}_1$  to obtain a new game  $\mathbf{G}_2$ . In this new game we modify the decryption oracle yet again. Given a ciphertext  $y$ , the new decryption oracle computes  $w, s, t, r, x, c$  as usual. If the old decryption oracle rejects, so does the new one. However, the new decryption oracle also rejects if  $s \notin S_H$ . More precisely, if the new decryption oracle computes  $s$  via (24), and finds that  $s \notin S_H$ , then it rejects right away, without ever querying  $H(s)$ . Thus, in game  $\mathbf{G}_2$ , the decryption oracle never queries  $G$ ,  $H'$ , or  $H$  at points other than those at which  $A$  did.

Let  $F_2$  be the event that a ciphertext is rejected in  $\mathbf{G}_2$  that would not have been rejected under the rules of game  $\mathbf{G}_1$ .

Consider a ciphertext  $y$  with  $s \notin S_H$  submitted to the decryption oracle. We consider two cases.

*Case 1: The encryption oracle has already been queried and  $s = s^*$ .* Now,  $s = s^*$  and  $y \neq y^*$  implies  $t \neq t^*$ . Moreover,  $s = s^*$  and  $t \neq t^*$  implies that  $r \neq r^*$ . If this ciphertext is rejected in game  $\mathbf{G}_2$  but would not be under the rules in game  $\mathbf{G}_1$ , it must be the case that  $H'(r^* \parallel x^*) = H'(r \parallel x)$ . The probability that such a collision can be found over the course of the attack is  $q_H/2^{k_1}$ . Note that  $r^*$  is fixed by the encryption oracle, and so “birthday attacks” are not possible.

*Case 2: The encryption oracle has not already been queried, or it has and  $s \neq s^*$ .* In this case, the oracle  $H$  was never queried at  $s$  by either  $A$ , the encryption oracle, or the decryption oracle. Since  $t = H(s) \oplus r$ , the value  $r$  is independent of *CurrentView*. It follows that the probability that  $r \in S_G$  is at most  $q_G/2^{k_0}$ . Over the course of the entire attack, these probabilities sum to  $q_D q_G/2^{k_0}$ .

It follows that  $\Pr[F_2] \leq q_{H'}/2^{k_1} + q_D q_G/2^{k_0}$ . Moreover, it is clear by construction that  $\Pr[S_1 \wedge \neg F_2] = \Pr[S_2 \wedge \neg F_2]$ , since the two games proceed identically unless  $F_2$  occurs. So applying Lemma 1 with  $(S_1, S_2, F_2)$ , we have

$$|\Pr[S_1] - \Pr[S_2]| \leq q_{H'}/2^{k_1} + q_D q_G/2^{k_0}. \quad (33)$$

**Game  $G_3$ .** Now we modify game  $G_2$  to obtain an equivalent game  $G_3$ . We modify the decryption oracle so that it does not make use of the trapdoor for  $f$  at all.

Conceptually, this new decryption oracle iterates through all pairs  $(r', x') \in S_{H'}$ . For each of these, it does the following. First, it sets  $s' = (G(r') \oplus x') \parallel H'(r' \parallel x')$ . Note that both  $G$  and  $H'$  have already been queried at the given points. Second, if  $s' \in S_H$ , it then computes

$$t' = H(s') \oplus r', \quad w' = s' \parallel t', \quad y' = f(w').$$

If  $y'$  is equal to  $y$ , it stops and outputs  $x'$ .

If the above iteration terminates without having found some  $y' = y$ , then the new decryption oracle simply rejects.

It is clear that games  $G_3$  and  $G_2$  are identical, and so

$$\Pr[S_3] = \Pr[S_2]. \quad (34)$$

To implement this idea, one would build up a table, with one entry for each  $(r', x') \in S_{H'}$ . Each entry in the table would contain the corresponding value  $s'$ , along with  $y'$  if  $s'$  is currently in  $S_H$ . If  $s'$  is currently not in  $S_H$ , we place  $y'$  in the table entry if and when  $A$  eventually queries  $H(s')$ . When a ciphertext  $y$  is submitted to the decryption oracle, we simply perform a table lookup to see if there is a  $y'$  in the table that is equal to  $y$ . These tables can all be implemented using standard data structures and algorithms. Using search tries to implement the table lookup, the total running time of the simulated decryption oracle over the course of game  $G_3$  is

$$O(\min(q_{H'}, q_H)T_f + (q_G + q_{H'} + q_H + q_D)k).$$

Note also that the space needed is essentially linear:  $O((q_G + q_{H'} + q_H)k)$  bits.

*Remark.* Let us summarize the modifications made so far. We have modified the decryption oracle so that it does not make use of the trapdoor for  $f$  at all; moreover, the decryption oracle never queries  $G$ ,  $H'$ , or  $H$  at points other than those at which  $A$  did.

**Game  $G_4$ .** In this game we modify the random oracles and slightly modify the encryption oracle. The resulting game  $G_4$  is equivalent to game  $G_3$ ; however, this rather technical “bridging” step will facilitate the analysis of more drastic modifications of the encryption oracle in games  $G_5$  and  $G'_5$  below.

We introduce random bit strings  $r^+ \in \{0, 1\}^{k_0}$  and  $g^+ \in \{0, 1\}^n$ . We also introduce a new random oracle

$$h^+: \{0, 1\}^n \rightarrow \{0, 1\}^{k_1}.$$

Game  $\mathbf{G}_4$  is the same as game  $\mathbf{G}_3$ , except that we apply the following special rules:

**R1:** In the encryption oracle we compute

$$y^* = f(s^* \parallel (H(s^*) \oplus r^*)),$$

where

$$r^* = r^+ \quad \text{and} \quad s^* = (g^+ \oplus x_b) \parallel h^+(x_b).$$

**R2:** Whenever the random oracle  $G$  is queried at  $r^+$ , we respond with the value  $g^+$ , instead of  $G(r^+)$ .

**R3:** Whenever the random oracle  $H'$  is queried at a point  $r^+ \parallel x$  for some  $x \in \{0, 1\}^n$ , we respond with the value  $h^+(x)$ , instead of  $H'(r^+ \parallel x)$ .

That completes the description of game  $\mathbf{G}_4$ . It is a simple matter to verify that the the random variable  $\langle \text{View}, b \rangle$  has the same distribution in both games  $\mathbf{G}_3$  and  $\mathbf{G}_4$ , since we have simply replaced one set of random variables by a different, but identically distributed, set of random variables. In particular,

$$\Pr[S_4] = \Pr[S_3]. \quad (35)$$

In this and the following game, we can consider  $r^*$  to be defined from the very beginning of the game.

**Game  $\mathbf{G}_5$ .** This game is identical to game  $\mathbf{G}_4$ , except that we drop rules **R2** and **R3**, while retaining rule **R1**.

In game  $\mathbf{G}_5$  it will not in general hold that  $x^* = x_b$  or that  $H(r^* \parallel x^*) = c^*$ . Moreover, since the values  $g^+$  and  $h^+(x_b)$  are not used anywhere else in game  $\mathbf{G}_5$  other than in the encryption oracle, we have

$$\Pr[S_5] = \frac{1}{2}. \quad (36)$$

Despite the above differences, games  $\mathbf{G}_4$  and  $\mathbf{G}_5$  proceed identically unless  $A$  queries  $G$  at  $r^*$  or  $H'$  at  $r^* \parallel x$  for some  $x \in \{0, 1\}^n$ . Recall that by our convention, whenever  $A$  queries  $H'$  at  $r^* \parallel x$  for some  $x \in \{0, 1\}^n$ , then  $G$  has already been queried at  $r^*$ . Let  $F_5$  be the event that in game  $\mathbf{G}_5$   $A$  queries  $G$  at  $r^*$ . We have  $\Pr[S_4 \wedge \neg F_5] = \Pr[S_5 \wedge \neg F_5]$ , and so by Lemma 1 applied to  $(S_4, S_5, F_5)$ ,

$$|\Pr[S_4] - \Pr[S_5]| \leq \Pr[F_5]. \quad (37)$$

**Game  $\mathbf{G}'_5$ .** We introduce an auxiliary game  $\mathbf{G}'_5$  in order to bound  $\Pr[F_5]$ . In game  $\mathbf{G}'_5$  we modify the encryption oracle once again. Let  $y^+ \in \{0, 1\}^k$  be a random bit string. Then in the encryption oracle, we simply set  $y^* = y^+$ , ignoring the encryption algorithm altogether.

In this game we can consider the value  $y^*$ , and the associated values  $y^*, w^*, s^*, t^*, r^*$ , etc., as being defined from the very start of the attack game.

It is not too hard to see that the random variable  $\langle \text{View}, r^* \rangle$  has the same distribution in both games  $\mathbf{G}_5$  and  $\mathbf{G}'_5$ . Indeed, the distribution of  $\langle \text{View}, r^* \rangle$  in game  $\mathbf{G}_5$  clearly remains the same if we instead choose  $r^*$  and  $s^*$  at random, and compute  $y^* = f(s^* \parallel (H(s^*) \oplus r^*))$ . Simply choosing  $y^*$  at random clearly induces the same distribution on  $\langle \text{View}, r^* \rangle$ . In particular, if we define  $F'_5$  to be the event that in game  $\mathbf{G}'_5$   $A$  queries  $G$  at  $r^*$ , then

$$\Pr[F_5] = \Pr[F'_5]. \quad (38)$$

So our goal now is to bound  $\Pr[F'_5]$ . To this end, let  $F''_5$  be the event that  $A$  queries  $H$  at  $s^*$  in game  $\mathbf{G}'_5$ . Then we have

$$\Pr[F'_5] = \Pr[F'_5 \wedge F''_5] + \Pr[F'_5 \wedge \neg F''_5]. \quad (39)$$

First, we claim that

$$\Pr[F'_5 \wedge F''_5] \leq \text{InvAdv}(A'), \quad (40)$$

where  $\text{InvAdv}(A')$  is the success probability of an inverting algorithm  $A'$  whose time and space requirements are bounded as in (30) and (31). To see this, observe that if  $A$  queries  $G$  at  $r^*$  and  $H$  at  $s^*$ , then we can easily convert the attack into an algorithm  $A'$  that computes  $f^{-1}(y^+)$  on input  $y^+$ .  $A'$  simply runs  $A$  against game  $\mathbf{G}'_5$ . When  $A$  terminates,  $A'$  enumerates all  $r' \in S_G$  and  $s' \in S_H$ , and for each of these computes

$$t' = H(s') \oplus r', \quad w' = s' \parallel t', \quad y' = f(w').$$

If  $y'$  is equal to  $y^+$ , then  $A'$  outputs  $w'$  and terminates.

Although game  $\mathbf{G}'_5$  is defined with respect to random oracles, there are no random oracles in  $A'$ . To implement  $A'$ , one simulates the random oracles that appear in game  $\mathbf{G}'_5$  in the “natural” way. That is, whenever  $A$  queries a random oracle at a new point,  $A'$  generates an output for the oracle at random and puts this into a lookup table keyed by the input to the oracle. If  $A$  has previously queried the oracle at a point,  $A'$  takes the output value from the lookup table. Again, using standard algorithms and data structures, such as search tries, the running time and space complexity of  $A'$  are easily seen to be bounded as claimed in (30) and (31).

Unfortunately, the running time of  $A'$  is much worse than that of the simulated decryption oracle described in game  $\mathbf{G}_3$ . However, at least the space remains essentially linear in the total number of oracle queries.

We also claim that

$$\Pr[F'_5 \wedge \neg F''_5] \leq q_G/2^{k_0}. \quad (41)$$

To see this, consider a query of  $G$  at  $r$ , prior to which  $H$  has not been queried at  $s^*$ . Since  $t^* = H(s^*) \oplus r^*$ , the value  $r^*$  is independent of  $\text{CurrentView}$ , and so  $\Pr[r = r^*] = 1/2^{k_0}$ . The bound (41) now follows.

Equations (39)–(41) together imply

$$\Pr[F'_5] \leq \text{InvAdv}(A') + q_G/2^{k_0}. \quad (42)$$



Equations (32)–(38) and (42) together imply (29).

That completes the proof of Theorem 3.  $\square$

*Remark.* Our reduction from inverting  $f$  to breaking OAEP+ is tighter than the corresponding reduction for OAEP in [BR2] (which only shows security against indifferent chosen ciphertext attack). In particular, the OAEP+ construction facilitates a much more efficient “plaintext extractor” than the OAEP construction. The latter appears to require inherently either

- time proportional to  $q_D q_G q_H$  and space linear in the number of oracle queries, or
- time proportional to  $q_D + q_G q_H$  and space proportional to  $q_G q_H$  (if one builds a look-up table).

For OAEP+, the total time and space complexity of the plaintext extractor in game  $\mathbf{G}_3$  is linear in the number of oracle queries. Unfortunately, our inversion algorithm for OAEP+ in game  $\mathbf{G}'_5$  still requires time proportional to  $q_G q_H$ , although its space complexity is linear in the number of oracle queries. We should remark that as things now stand, the reductions for OAEP+ are not tight enough actually to imply that an algorithm that breaks, say, 1024-bit RSA-OAEP+ in a “reasonable” amount of time implies an algorithm that solves the RSA problem in time faster than the best known factoring algorithms. However, as we shall see in Section 7.2, for exponent-3 RSA-OAEP+, one can in fact get a very tight reduction. An interesting open problem is to get a tighter reduction for OAEP+ or a variant thereof.

## 7. Further Observations

### 7.1. Other Variations of OAEP

Instead of modifying OAEP as we did, one could also modify OAEP so that instead of adding the data-independent redundancy  $0^{k_1}$  in (1), one added the data-dependent redundancy  $H''(x)$ , where  $H''$  is a hash function mapping  $n$ -bit strings to  $k_1$ -bit strings. This variant of OAEP—call it OAEP'—suffers from the same problem from which OAEP suffers. Indeed, Theorem 1 holds also for OAEP'.

### 7.2. RSA-OAEP with Exponent 3 Is Provably Secure

Consider RSA-OAEP. Let  $N$  be the modulus and let  $e$  be the encryption exponent. Then this scheme actually *is* secure in the random oracle model, provided  $k_0 \leq \log_2 N/e$ . This condition is satisfied by typical implementations of RSA-OAEP with  $e = 3$ .

We sketch very briefly why this is so.

We first remind the reader of the attempted proof of security of OAEP in Section 4, and we adopt all the notation specified there.

Suppose an adversary submits a ciphertext  $y$  to the decryption oracle. We observed in Section 4 that if the adversary never explicitly queried  $H(s)$ , then with overwhelming probability, the actual decryption oracle would reject. The only problem was, we could not always say the same thing about  $G(r)$  (specifically, when  $r = r^*$ ).

For a bit string  $v$ , let  $I(v)$  denote the unique integer such that  $v$  is a binary representation of  $I(v)$ .

If a simulated decryption oracle knows  $s$  (it will be one of the adversary's  $H$ -queries), then  $X = I(t)$  is a solution to the equation

$$(X + 2^{k_0} I(s))^e \equiv y \pmod{N}.$$

To find  $I(t)$ , we can apply Coppersmith's algorithm [Co]. This algorithm works provided  $I(t) < N^{1/e}$ , which is guaranteed by our assumption that  $k_0 \leq \log_2 N/e$ .

More precisely, for all  $s' \in S_H$ , the simulated decryption oracle tries to find a corresponding solution  $t'$  using Coppersmith's algorithm. If all of these attempts fail, then the simulator rejects  $y$ . Otherwise, knowing  $s$  and  $t$ , it decrypts  $y$  in the usual way.

We can also apply Coppersmith's algorithm in the step of the proof where we use the adversary to help us extract a challenge instance of the RSA problem.

Not only does this prove security, but we get a more efficient reduction—the implied inverting algorithm has a running time roughly equal to that of the adversary, plus  $O(q_D q_H T_C)$ , where  $T_C$  is the running time of Coppersmith's algorithm.

We can also use the same observation to speed up the reduction for exponent-3 RSA-OAEP+. The total running time of the implied inversion algorithm would be roughly equal to that of the adversary, plus  $O(q_H T_C)$ ; that is, a factor of  $q_D$  faster than the inversion algorithm implied by RSA-OAEP. Unlike the generic security reduction for OAEP+, this security reduction is essentially tight, and so it has much more meaningful implications for the security of the scheme when used with a typical, say, 1024-bit RSA modulus.

### 7.3. RSA-OAEP with Large Exponent

In our example in Section 4.1, as well as in our proof of Theorem 1, the adversary is able to create a valid ciphertext  $y$  without ever querying  $G(r)$ . However, this adversary queries both  $H(s)$  and  $H(s^*)$ . As we already noted, the adversary must query  $H(s)$ , but it turns out that if the adversary avoids querying  $G(r)$ , he must query  $H(s^*)$ . This observation was made by Fujisaki et al. [FOPS], who then further observed that this implies the security of RSA-OAEP with an arbitrary encryption exponent in the random oracle model. We remark, however, that the reduction in [FOPS] is significantly less efficient than our general reduction for OAEP+. In particular, their reduction only implies that if an adversary has advantage  $\varepsilon$  in breaking RSA-OAEP, then there is an algorithm that solves the RSA inversion problem with probability about  $\varepsilon^2$ . Moreover, their inversion algorithm is even somewhat slower than that of the (incorrect) inversion algorithm for OAEP in [BR2]. There is still the possibility, however, that a more efficient reduction for RSA-OAEP can be found.

### Acknowledgments

Thanks to Jean-Sebastien Coron for pointing out an error in an early draft of this paper. Namely, it was claimed that the the variant OAEP' briefly discussed in Section 7.1 could also be proven secure, but this is not so.

## References

- [ASW] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000. Extended abstract in *Advances in Cryptology—Eurocrypt '98*.
- [BCK] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 419–428, 1998.
- [BDPR] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology—Crypto '98*, pages 26–45, 1998.
- [Bo] D. Boneh. Simplified OAEP for the RSA and Rabin functions. In *Advances in Cryptology—Crypto 2001*, pages 275–291, 2001.
- [BR1] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
- [BR2] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology—Eurocrypt '94*, pages 92–111, 1994.
- [CGH] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 209–218, 1998.
- [Co] D. Coppersmith. Finding a small root of a univariate modular equation. In *Advances in Cryptology—Eurocrypt '96*, pages 155–165, 1996.
- [DDN1] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 542–552, 1991.
- [DDN2] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.
- [FOPS] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. In *Advances in Cryptology—Crypto 2001*, pages 260–274, 2001. An earlier version appeared as Cryptology ePrint Archive, Report 2000/061, <http://eprint.iacr.org>.
- [GM] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
- [MvOV] A. Meneses, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1997.
- [NY] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 427–437, 1990.
- [RS] C. Rackoff and D. Simon. Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology—Crypto '91*, pages 433–444, 1991.
- [RSA] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [Sh1] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology—Eurocrypt '97*, pages 256–266, 1997.
- [Sh2] V. Shoup. On formal models for secure key exchange. Cryptology ePrint Archive, Report 1999/012, 1999. <http://eprint.iacr.org>.