An OAEP Variant With a Tight Security Proof – Draft 1.0

Jakob Jonsson

RSA Laboratories

March 18, 2002

Abstract

We introduce the OAEP^{++} encoding method, which is an adaptation of the OAEP encoding method, replacing the last step of the encoding operation with an application of a block cipher such as AES. We demonstrate that if f is a one-way trapdoor function that is hard to invert, then OAEP^{++} combined with f is secure against an IND-CCA2 adversary in the random oracle model. Moreover, the security reduction is tight; an adversary against f-OAEP^{++} can be extended to an f-inverter with a running time linear in the number of oracle queries. [1]

1 Introduction

The purpose of this paper is to introduce a new encoding method, which will be denoted OAEP^{++}. This method is a minor tweak of the OAEP encoding method [3], replacing the last step in the latter encoding operation with an application of a block cipher (i.e., a pseudo-random permutation). The OAEP^{++} encoding method can be combined with any one-way trapdoor function f, for example RSA [31]. The components of OAEP^{++} are a mask generation function similar to the one used in the first round of OAEP and a block cipher such as AES [25]. In addition, a block cipher key derivation function is required if the block cipher does not support long key sizes.

OAEP^{++} takes as input a message m of a fixed length, generates a random seed r, and forms a padded message $p\|m$, where p is a fixed string. The mask generation function is applied to r to form an output s' of the

[1] The information in this document is provided as is, and no warranty is given or implied that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

same length as $p\|m$; put $s = s' \oplus (p\|m)$. Finally, the seed r is encrypted with the block cipher using s as the key to form an output t. The OAEP^{++} encoded message is the string $t\|s$. If the block cipher does not support keys of length $|s|$ (this is typically the case), s is transformed into an encryption key of appropriate length via the key derivation function. To encrypt a message with f-OAEP^{++}, encode the message with OAEP^{++} and apply f to the OAEP^{++} encoded message. The OAEP^{++} encoding operation is illustrated at the end of this paper.

The only difference between OAEP^{++} and OAEP is that the latter method defines t as the xor sum of r and a string derived from s via the mask generation function; OAEP encrypts the seed with a stream cipher, whereas OAEP^{++} encrypts it with a block cipher.

A typical parameter choice for OAEP^{++} would be $|r| = |p| = 128$ with AES as the underlying block cipher and with the mask generation function and the key derivation function based on SHA-256 [24] (this means that we use AES with 256-bit keys). A potential variant of OAEP^{++} defined entirely in terms of AES is discussed in Appendix C.

The main advantage of the OAEP^{++} encoding method is that the security of the combination f-OAEP^{++} can be tightly related to the hardness of inverting f. For example, this holds for $f = \text{RSA}$. While the reduction requires non-standard "random oracle" assumptions on the components, we claim that these assumptions are not stronger than the corresponding assumptions needed to prove the security of RSA-OAEP and the variant RSA-OAEP^{+} [35]. Recall from [35] and [13] that the best known security reductions for RSA-OAEP and RSA-OAEP^{+} are not tight (see Section 2).

Another advantage of OAEP^{++} is that it provides compact ciphertexts; with parameters as above, a plaintext of size $k - 256$ can be encrypted, resulting in a ciphertext of size k, where k is the bit length of the input to f. Similar properties hold for RSA-OAEP and RSA-OAEP^{+} as well, while encryption schemes such as [37, 2, 36, 28] based on RSA-KEM ("simple-RSA"; see Section 2) have a ciphertext overhead compared to RSA-OAEP^{++} of size at least the length of the message to be encrypted. While RSA-KEM-based schemes are particularly attractive choices for key agreement and key establishment applications, RSA-OAEP^{++} might be a preferable alternative in applications where rather small secrets need to be transmitted securely.

The running time for the OAEP^{++} encoding operation is the same as for OAEP plus the time needed to perform one block encryption. OAEP^{++} and OAEP^{+} require roughly the same amount of operations; there is no block cipher within the latter mechanism, but instead one application of a hash function is required. The conclusion is that neither variant is particularly advantageous or disadvantageous in terms of performance; one application of the trapdoor function (or its inverse in case of decryption) is typically several magnitudes more time-consuming than any of the encoding methods discussed in this section.

Organization of the paper In Section 2 we give a brief overview of the history of RSA encryption schemes, concentrating on schemes based on the encode-and-encrypt paradigm. OAEP^{++} design rationale is provided in Section 3; the formal definition of OAEP^{++} is given in Section 4. In Section 5 we provide a heuristic argument for the tight security of OAEP^{++}; a formal security analysis is given in Section 6. Finally, Section 7 elaborates on an extended version of f-OAEP^{++} applicable to messages of arbitrary length.

2 Background

Ever since the RSA public-key encryption scheme [31] was introduced by Rivest, Shamir, and Adleman in the late seventies, one of the most important and most widely studied problems is how to properly pad a message to be encrypted. Celebrated cryptanalytic results such as [16], [6], and [7] stress the need for a nontrivial padding method rather than just applying RSA to the message directly. Specifically, the larger part of the RSA inverse of the ciphertext must be unpredictable for an adversary even if the plaintext is known. The obvious solution is to introduce some randomness; to encrypt a message, mix the message and the generated randomness somehow before applying the RSA encryption function.

The most widely used encoding method as of today is the PKCS #1 v1.5 padding method introduced in [33]. This method, from now on denoted P1, generates a random string r (with some prespecified structure) and concatenates r with the message m to be encrypted.

We believe that RSA-P1 is adequate for key agreement protocols such as the handshake protocol in TLS [11], where the plaintext is a random and unpredictable secret. Yet, as an ordinary encryption scheme, the technique suffers from the lack of a convincing security argument. Indeed, Bleichenbacher's "Million Messages Attack" [4] implies that RSA-P1 is not secure against adaptive chosen-ciphertext attack without additional countermeasures. In addition, the results in [8] indicate that there is little hope to find a security proof even in the chosen-plaintext attack model without countermeasures.

One further concern about P1 is that an adversary who knows how to extract, say, the last few bits of the RSA inverse from the ciphertext will be able to derive information about the plaintext. Ideally, an encryption scheme based on RSA should have a security that can be provably related to the hardness of fully (rather than partially) inverting RSA on a random input. While there are polynomial-time reductions to a "partial" RSA inverter from a full RSA inverter, these reductions are typically far from tight and are therefore not very useful for concrete parameters. See [17] for a survey of different reduction techniques; a related technique for large partial blocks can be found in [13].

RSA-OAEP, introduced by Bellare and Rogaway in 1994 [3], is a scheme that turns out to have many of the desired security properties. The seminal paper [3] provides a security proof for RSA-OAEP in terms of RSA in a chosen-ciphertext attack model. The proof is based on the assumption that the underlying mask generation function g is uniformly chosen from the set of all possible functions and unpredictable for the adversary; if the adversary wants to compute g on a certain value r, she must send r to an external *oracle*, which responds with a uniformly random value $g(r)$.

In the attack model in [3], decryption queries from the adversary are accepted only before the adversary is given the target ciphertext to be examined. For several years, it was widely believed that the security proof for OAEP could be extended to the more general model in which the adversary is allowed to send decryption queries during the entire attack (i.e., the attack is *adaptive*). However, Shoup [35] demonstrated that this cannot be true for OAEP combined with a general trapdoor function. To address this concern, Shoup introduced OAEP$^+$, which is an adapted version of OAEP that is provably secure against adaptive chosen-ciphertext attack when combined with a secure trapdoor function. Fortunately however, Fujisaki, Okamoto, Pointcheval, and Stern [13] were able to provide a security proof also for OAEP when combined with RSA.

Some other members in the OAEP family worth mentioning are the SAEP [18] and SAEP+ [5] methods. SAEP is similar to OAEP except that the second application of the mask generation function is omitted. SAEP+ is derived from OAEP$^+$ in the same manner. Boneh [5] has demonstrated that both schemes are secure when combined with the Rabin [29] trapdoor function. Also, SAEP+ can be securely combined with RSA.

Thus, we have an entire family of encoding methods that are provably secure in a random oracle model. However, none of these methods has a known tight security reduction (except for when combined with certain trapdoor functions such as Rabin and RSA with exponent 3). More precisely, the reductions are much more time-consuming than the running time of the underlying adversary. What has been proven is roughly that if RSA cannot be inverted in time T, then the corresponding encryption scheme cannot be broken in time $O(\sqrt{T})$. This is not a very useful result for practical parameters such as $T \approx 2^{80}$ or even $T \approx 2^{128}$.

However, it has been known for years how to provide an RSA-based encryption scheme with a tight reduction; Zheng and Seberry [37] introduced such a scheme as early as in 1992. The idea is to generate a random integer r between 0 and $N-1$ (N is the RSA modulus), encrypt r with RSA, and encrypt and authenticate the message with a symmetric key (or a key stream) derived from r. The procedure of deriving a symmetric key using RSA and a key derivation function in this manner is denoted RSA-KEM [36].

The most well-known encryption schemes based on RSA-KEM are RSA-

REACT [28] and RSA-KEM-DEM1 [36]. Among numerous attractive features of these constructions, we mention that they can handle messages of arbitrary length and that they can be defined in terms of any public-key encryption primitive, not only trapdoor functions such as RSA. This property is true also for a third variant called RSA-GEM [9], which has a slightly smaller ciphertext overhead (the integrity check is embedded in the input to RSA). However, this scheme suffers from a weaker security reduction.

The only significant drawback of schemes based on RSA-KEM compared to schemes in the OAEP family is the ciphertext overhead when the message to be encrypted is smaller than the RSA modulus length. The problem of finding a practical encoding method with a tight security reduction *and* no ciphertext overhead appears to have been unsettled thus far. In this paper we provide a solution that we believe satisfies those two conditions.

3 Design goals and OAEP^{++} design rationale

We want to define an encoding method in terms of random oracles such that an adversary must be able to determine the entirety of $x = f^{-1}(y)$ for a target ciphertext y to be able to say anything about the corresponding plaintext m. More precisely, if the adversary is able to determine nontrivial information about m, then she must have leaked the entirety of x in her correspondence with the oracles. An *inverter* simulating the oracles is then able to determine x from the communication.

This is true for RSA-OAEP and RSA-OAEP$^+$. Yet, the best known reductions for these schemes are not tight. The drawback of these schemes is that a combination of two oracle queries is needed to recover the whole of x. Thus the inverter must check many *pairs* of queries before he finds the correct pair, which makes the time bound for the inverter proportional to the square of the number of oracle queries rather than linear in the number of queries. The obvious design goal is hence to construct an encoding method such that the plaintext cannot be recovered unless the entire encoded message is included in one single oracle query.

As a comparison, we analyze the OAEP encoding method in greater detail. After the first Feistel half-round, the encoded message consists of the seed r and a string s that is the xor sum of the padded message and a pseudo-random string generated from r. In the second round, a mask generation function h is applied to s and xored to r; the result is a new string $t = h(s) \oplus r$. To recover the original message, we need to send first s to one oracle and then r to another oracle. However, neither the query s nor the response $h(s)$ leak any *a priori* information about r or $h(s) \oplus r$. This is the core reason why RSA-OAEP (as well as RSA-OAEP$^+$) does not have a tight security reduction; each pair of queries (r, s) corresponds to a possible encoded message, which may well be the f inverse of the target ciphertext.

Yet, look at the second step of the OAEP encoding operation again; we transform r by xoring $h(s)$ to it. This means that we *encrypt* r with the *stream cipher* h using the *encryption key* s. A stronger encryption method is to apply a well-trusted *block cipher* E to r using s as the encryption key. Assuming that E_s is a random permutation for each s, we obtain a scheme where an adversary cannot determine any non-negligible information about the decryption of a ciphertext $y = f(t\|s)$ unless she asks for the block cipher decryption of t under the key s. This implies that the adversary must know the entire inverse of y under f. Of course, this argument is very heuristic, but we will demonstrate in Section 6 that the security of this scheme is indeed tightly related to the hardness of inverting f.

There are two remaining concerns regarding this construction, which will be denoted OAEP^{++}; see Section 4 for a formal description.

- While the length k_r of r may well be a standard block length such as 128 (which means that AES [25] might be used), the length $k - k_r$ of the key s is typically not a standard block cipher key length. While there are block ciphers that support variable key length (for example, RC6 [32]), the construction would be more versatile if the key length were fixed to a certain value (for example, 256). The solution is to apply a key derivation function h (idealized as a random oracle) to s and use the result $h(s)$ as the key.

  Now it seems that we violate the property that each block cipher decryption query ("D-oracle" query) should correspond to a unique ciphertext $y = f(t\|s)$. Namely, the query corresponding to y contains the string t to be decrypted and the key $h(s)$, but not s. Yet, an adversary cannot guess $h(s)$ with non-negligible probability unless she has queried s before at the h-oracle. In particular, queries to the h-oracle and the D-oracle are tightly related; to each D-oracle (and E-oracle) query corresponds at most one relevant h-oracle query unless there are known h-collisions. See Lemma 6.1 for a formal demonstration of this fact.

- It may seem that the random oracle assumption on a block cipher is less reasonable than the corresponding random oracle assumption on a hash function or a mask generation function.[2] Namely, a hash function is one-way and does not have a simple description of how to recover the original message from the hash. A block cipher however must be reversible; given the key, it should be straightforward to recover the plaintext. Intuitively, this seems to imply that an block cipher has more "structure" than a hash function.

[2]We stress that the random oracle assumption is an ideal assumption that cannot be interpreted in the real world.

However, there are quite a few efficient constructions for deriving a block cipher from a pseudo-random function (PRF); see [20, 23] (a hash function is easily transformed into a PRF). These constructions may not have very good security bounds for concrete security parameters, but they indicate that the above intuition is not entirely accurate.

In addition, standard hash functions such as the ones in the SHA family are typically defined in terms of a compression function, following the Merkle-Damgård [10, 22] paradigm. In certain cases (including the SHA family), this compression function (or a slight variant thereof), is a pseudo-random permutation. This has been observed by Handschuh and Naccache, who submitted the compression function in SHA-1, denoted SHACAL [15], as a block cipher to the NESSIE project [27]. If the random oracle assumption on a block cipher is unreasonable, then so is clearly the corresponding assumption on a hash function that is based on a block cipher in a very straightforward manner.

4 Scheme description

Let k, k_r, k_p, and k_u be parameters such that $k_r + k_p < k$. The OAEP^{++} encoding operation transforms a message of bit length $k_m = k - k_r - k_p$ into an encoded message of bit length k. This is done via a *mask generation function*

$$g : \{0,1\}^{k_r} \to \{0,1\}^{k-k_r},$$

a *key derivation function*

$$h : \{0,1\}^{k-k_r} \to \{0,1\}^{k_u},$$

and a symmetric *block cipher* (pseudo-random permutation)

$$E : \{0,1\}^{k_u} \times \{0,1\}^{k_r} \to \{0,1\}^{k_r}.$$

A block cipher E has the property that $r \to E(u,r)$ is a permutation for each fixed key u. To E corresponds a decryption operation D such that $D(u, E(u,r)) = r$ for all $(u,r) \in \{0,1\}^{k_u} \times \{0,1\}^{k_r}$. Put $E_u(r) = E(u,r)$ and $D_u(t) = D(u,t)$; u is the *(encryption) key*. Note that $D_u = E_u^{-1}$. We assume that g, h, E, and D are all "efficient" functions.

The OAEP^{++} encoding operation is defined as follows. The operation takes as input a message m of length k_m and a parameter string p of length k_p.

OAEP^{++}-Encode(m, p)
- $m' \leftarrow p \| m$;
- $r \xleftarrow{R} \{0,1\}^{k_r}$;

- $s \leftarrow m' \oplus g(r)$;
- $u \leftarrow h(s)$;
- $t \leftarrow E_u(r)$;
- $x \leftarrow t \| s$;
- Output x.

Note that we have fixed the length of m. An alternative that is probably more useful in practice is to redefine the scheme such that it can handle messages m of variable length. For example, if we replace the second step in the above description with

$$m' \leftarrow p \| 0^{k_m - |m| - 1} \| 1 \| m, \tag{1}$$

then the resulting scheme can handle any message of length less than k_m.

We want to combine OAEP^{++} with a one-way trapdoor permutation $f : X \to X$ such that $X \supseteq \{0,1\}^k$. We define the concept of one-way trapdoor permutations formally in Section 6. Define the combined scheme f-OAEP^{++} as follows. First, fix an encoding parameter string p of length k_p. The f-OAEP^{++} encryption operation takes as input the one-way trapdoor permutation f and a message m of length k_m.

f-OAEP^{++}-ENCRYPT(f, m)
- $x \leftarrow$ OAEP^{++}-ENCODE(m, p);
- $y \leftarrow f(x)$;
- Output y.

The mathematical description of the reverse operations OAEP^{++}-DECODE and f-OAEP^{++}-DECRYPT are easily derived from the encoding and encryption operations. Yet, there are some concerns that need to be addressed. Namely, since $\{0,1\}^k \subsetneq X$ for typical trapdoor functions such as RSA, not all elements in X are possible outputs from OAEP^{++}-ENCODE. This means that the decoding operation should report an error if its input is not an element in $\{0,1\}^k$. Yet, attacks by Bleichenbacher [4] and Manger [21] on RSA-PKCS #1 v1.5 and RSA-OAEP, respectively, demonstrate that such errors must not be distinguishable from other kinds of errors.

To avoid potential implementation weaknesses, we will need to deal with elements $x \in X \setminus \{0,1\}^k$. For this reason, introduce a function $\omega : X \to \{0,1\}^k$ such that $\omega(x) = x$ if $x \in \{0,1\}^k$. For example, if $X = \mathbb{Z}_N$, then we can define $\omega(x) = x \bmod 2^k$.

To decode $x \in X$, proceed as follows.

OAEP^{++}-DECODE(x, p)
- Put $x' = \omega(x)$;
- Write $x' = t \| s$ ($|t| = k_r$);

- $u \leftarrow h(s)$;
- $r \leftarrow D_u(t)$;
- $m' \leftarrow s \oplus g(r)$;
- Write $m' \leftarrow p' \| m$;
- If $p' \neq p$ or $x \notin \{0,1\}^k$; then output "Decoding error" and exit;
- Output m.

Note that all errors are reported simultaneously (if a padding scheme such as (1) is used, then there might be additional potential errors that should be reported in the very same step).

The f-OAEP^{++} decryption operation takes as input the inverse f^{-1} of the one-way trapdoor function f and a ciphertext $y \in X$.

f-OAEP^{++}-Decrypt(f^{-1}, y)
- $x \leftarrow f^{-1}(y)$;
- $m \leftarrow$ OAEP^{++}-Decode(x, p);
- In case of decoding error, output "Decryption error" and exit;
- Output m.

5 A heuristic security argument

Before proceeding with formal definitions and proofs, we provide a (very) heuristic argument for the security of f-OAEP^{++}.

We assume that the block cipher E is a random oracle; for each key u, E_u is a random permutation chosen uniformly at random from the set of all permutations. This means that an adversary who wants to compute $E_u(r)$ must submit both u and r to the oracle simulating E. Even more important, to compute $D_u(t)$, she must submit both u and t to the oracle simulating D. We assume that the other functions g and h are random oracles as well.

Let y^* be a ciphertext, and let s^*, t^*, u^*, r^*, and m^* be the values corresponding to y^*. An adversary is given y^* and wants to determine information about m^*. It seems intuitively clear that if the adversary is able to achieve this goal (using a strategy better than pure guessing), then she must have submitted (u^*, t^*) to the D-oracle. Namely, to recover any information about m^*, the adversary must know $g(r^*)$, because otherwise $g(r^*)$ would be a completely random string, which implies that $g(r^*) \oplus s^*$ would be completely random as well. Thus r^* must be a previous g-oracle query. Since r^* is (almost) completely random unless (u^*, t^*) is a previous query to the D-oracle, we conclude that (u^*, t^*) must indeed be a D-oracle query with overwhelming probability.

Also, without having queried s^* at h, the adversary has no information about u^*. Thus if (u^*, t^*) is a D-oracle query, then, with overwhelming probability, s^* is an h-oracle query. Unless there are h-collisions, these two

queries are tightly related; there is only one h-oracle query that is relevant for each E- and D-oracle query. As we will see, this implies that an f-inverter defined in terms of the adversary will be able to determine the inverse of y^* in time linear in the number of queries.

6 Security analysis of OAEP^{++}

6.1 Trapdoor permutations

We want to analyze the security of f-OAEP^{++} for a given one-way trapdoor permutation f. First we define the concept of trapdoor permutations. Let $\mathcal{F}$ be a finite family of pairs (f, f^{-1}) of permutations $f, f^{-1} : X_f \to X_f$ such that f^{-1} is the inverse of f. This means that $f^{-1}(f(x)) = x$ for any $x \in X_f$. We assume that $f(y)$ and $f^{-1}(y)$ are "easy" to compute on any inputs (f, y) and (f^{-1}, y), respectively.

Let $\mathcal{G}$ be a probabilistic polynomial-time (PPT) algorithm that outputs a pair $(f, f^{-1}) \in \mathcal{F}$. $\mathcal{G}$ is a *trapdoor permutation generator*. An *f-inverter* $\mathcal{I}$ is an algorithm that on input (f, y) tries to compute $f^{-1}(y)$ for a random $y \in Y$. [3] $\mathcal{I}$ has success probability ϵ and running time T if

$$\Pr\left((f, f^{-1}) \leftarrow \mathcal{G}; x \xleftarrow{\mathsf{R}} X_f; y \leftarrow f(x) : \mathcal{I}(f, y) = x\right) \geq \epsilon$$

and the running time for $\mathcal{I}$ is at most T. In words, $\mathcal{I}$ should be able to compute $f^{-1}(y)$ with probability ϵ within time T, where (f, f^{-1}) is derived via the trapdoor permutation generator and y is random.

6.2 Attack model

We define an attack model employing an adversary against f-OAEP^{++} who is given free access to a *decryption oracle*; hence we consider the family of adaptive chosen-ciphertext attacks (CCA2; see [14]). The task for the adversary is to distinguish a plaintext m_0^* corresponding to a certain ciphertext y^* from another plaintext m_1^*.

The decryption oracle responds to a query y with the corresponding plaintext $m = f$-OAEP^{++}-DECRYPT(f^{-1}, y) unless there is a decryption error, in which case the oracle responds with a generic error message. The decryption oracle accepts any query, except that y^* is not a valid query after the challenge generator (see below) has been triggered.

The functions g and h are instantiated as random oracles. Thus the adversary has no information about $h(r)$ unless she sends the *query* r to an oracle instantiating h (and similar for g). A random oracle responds

[3] f^{-1} not being an input to $\mathcal{I}$ means that $\mathcal{I}$ must find out himself how to compute $f^{-1}(y)$; he is not provided with an *explicit* description of f^{-1}, only the implicit description given by f.

to queries with strings chosen uniformly at random and independent from earlier queries and responses, except that a string that is repeatedly queried to the oracle should have the same response every time.

Similarly, the block cipher E is assumed to be ideal. This means that E_u is a permutation chosen uniformly at random from the set of all random permutations for each $u \in \{0,1\}^{k_u}$ and that all permutations E_u are mutually independent. There is one oracle simulating E and one oracle simulating the inverse D (obviously, those two oracles are tightly related).

The attack experiment runs as follows. First, the adversary is given a trapdoor permutation f generated via a trapdoor permutation generator $\mathcal{G}$. The adversary is allowed to send queries to the random oracles and the decryption oracle during the entire attack. At any time of the attack – but only once – the adversary triggers a *challenge generator* with an input (m_0^*, m_1^*); $m_i^* \in \{0,1\}^{k_m}$. The challenge generator flips a fair coin b and applies the f-OAEP^{++}-Encrypt operation to m_b^*, producing a ciphertext y^*. The generator returns y^*.

At the end, the adversary outputs a bit b'. The *distinguishing advantage* of the adversary is defined as

$$\big(\Pr(b' = b) - \Pr(b' \neq b)\big) = 2\Pr(b' = b) - 1; \tag{2}$$

the probability is computed over all possible trapdoor mappings. The adversary is an *IND-CCA2* adversary [14, 30] (IND = indistinguishability).

6.3 A preliminary reduction

Due to the excessive number of oracles, we want to get rid of some oracle before proceeding; in this manner, the security analysis is simplified. More precisely, we will exclude the h-oracle from the model and replace E with a block cipher $\hat{E}$ using a string $s \in \{0,1\}^{k-k_r}$ instead of $h(s) \in \{0,1\}^{k_u}$ as the encryption key. Let S-OAEP^{++} ("simplified" OAEP^{++}) be the encoding operation obtained in this manner; S-OAEP^{++}-Encode is defined as follows.

S-OAEP^{++}-Encode(m, p)
- $m' \leftarrow p \| m$;
- $r \stackrel{R}{\leftarrow} \{0,1\}^{k_r}$;
- $s \leftarrow m' \oplus g(r)$;
- $t \leftarrow \hat{E}_s(r)$;
- $x \leftarrow t \| s$;
- Output x.

Our first goal is to reduce an adversary against f-S-OAEP^{++} to an adversary against f-OAEP^{++}. This is done via the following lemma.

Lemma 6.1 *Let $\mathcal{A}$ be an IND-CCA2 adversary against f-OAEP^{++} with advantage ϵ' within running time T'. Assume that $\mathcal{A}$ makes at most q_g number of g-oracle queries, q_h number of h-oracle queries, q_E number of E- and D-oracle queries, and q_f number of decryption queries. Then there is an IND-CCA2 adversary $\hat{\mathcal{A}}$ against f-S-OAEP^{++} with advantage ϵ within running time T making q_g number of g-oracle queries, q_E number of $\hat{E}$- and $\hat{D}$-oracle queries, and q_f number of decryption queries. Here,*

$$\epsilon = \epsilon' - (q_h + q_f + 1) \cdot (2q_E + q_h + q_f) \cdot 2^{-k_u - 1}$$

and

$$T = T' + O(q_h + q_E).$$

The proof is given in Appendix A.

Remark. The additional failure probability for the f-S-OAEP^{++} adversary in Lemma 6.1 will be non-negligible if the square of the number of queries is close to 2^{k_u}. For this reason, k_u must be fairly large; $k_u = 128$ is way too small for most practical applications. We recommend $k_u = 256$ for applications requiring up to approximately 128 bits of security.

6.4 Relating f-OAEP^{++} to f

We will now give a security proof for f-S-OAEP^{++}, relating the scheme to the underlying trapdoor permutation f. Using Lemma 6.1, we obtain a proof for f-OAEP^{++}. To simplify notation, let E (rather than $\hat{E}$) denote the block cipher in S-OAEP^{++}.

Theorem 6.2 *Let q_g, q_E, q_f, k_r, k_p be parameters. Put $q_0 = q_g + q_f + 2q_E$ and assume that $2q_0 \leq \max\{2^{k_p}, 2^{k_r}\}$; put*

$$\varphi = \left\lceil \frac{k_p + \log_2 q_0}{k_p - \log_2 q_0} \right\rceil$$

(see the remark below). Assume that there is an IND-CCA2 adversary breaking f-S-OAEP^{++} with advantage ϵ' within time T' and making q_g number of g-oracle queries, q_E number of E- and D-oracle queries, and q_f number of decryption queries. Then there is an f-inverter with success probability ϵ within time T, where

$$\epsilon = \frac{2^k}{|X|} \left(\epsilon' - ((2\varphi + 1)q_f + q_g + 2q_E)(2^{-k_r} + 2^{-k_p}) \right).$$

and

$$T = T' + O(q_f + (q_g + q_E)T_f);$$

T_f is the running time for f.

The proof of Theorem 6.2 is given in Appendix B.

Remark. Note that $\varphi = \left\lceil \frac{k_p + \log_2 q_0}{k_p - \log_2 q_0} \right\rceil$ implies

$$q_0^{\varphi+1} \leq 2^{(\varphi-1)k_p}.$$

This unorthodox assumption is to facilitate analysis of the special case that $k_r = 128$. Such a small k_r means that there will be E-collisions (values (s, r) and (s', r') such that $E_s(r) = E_{s'}(r')$) and D-collisions with a large probability. To be able to handle this situation, we need k_p to be slightly larger than the security parameter $\log_2 q$; φ is a probabilistic bound on the maximal number of collisions on the k_p first positions among all known outputs from g.

Combining Theorem 6.2 and Lemma 6.1, we obtain the following security result for f-OAEP^{++}.

Corollary 6.3 *Let $q_g, q_h, q_E, q_f, k_r, k_p, k_u$ be parameters. Put $q_0 = q_g + q_f + 2q_E$ and assume that $2q_0 \leq \max\{2^{k_p}, 2^{k_r}\}$; put*

$$\varphi = \left\lceil \frac{k_p + \log_2 q_0}{k_p - \log_2 q_0} \right\rceil.$$

Put

$$q = \max\{(2\varphi + 1)q_f, q_h, q_E, q_g\}.$$

Assume that there is an IND-CCA2 adversary breaking f-OAEP^{++} with advantage ϵ' within time T' and making q_g number of g-oracle queries, q_h number of h-oracle queries, q_E number of E- and D-oracle queries, and q_f number of decryption queries. Then there is an f-inverter with success probability ϵ within time T, where

$$\epsilon = \frac{2^k}{|X|}\left(\epsilon - 4q \cdot (2^{-k_r} + 2^{-k_p}) - 2q(2q+1) \cdot 2^{-k_u}\right)$$

and

$$T = T' + O(q) + O(qT_f);$$

T_f is the running time for f. □

7 Encrypting messages of arbitrary length with OAEP^{++} and f

One concern about f-OAEP^{++} (as well as f-OAEP and f-OAEP^{+}) is that the size of a message that can be encrypted is bounded by $k - k_r - k_p$. In case we want to encrypt a long message, only a small part can be input to the encryption scheme while the remaining part must be handled in some other way, including means for authentication of the encrypted message.

In this section we will address this concern. The application we have in mind is a simple one-pass protocol where a single message is to be transmitted. This is the one situation where an "encode-and-encrypt" scheme based on RSA might be more attractive than a scheme based on RSA-KEM. In other more complex multi-pass settings, RSA-KEM would be more beneficial.

We proceed as follows on a message m of bit length $k_m \geq k - k_r - k_p$ (shorter messages can be padded in the appropriate manner as in (1)). What we do is that we encode the entire message m using the OAEP^{++} encoding operation and then encrypt only the last k bits of the encoded message with f. The reason why this works is very simple: If $f : X \to X$ is a one-way trapdoor function, then so is $f_l : \{0,1\}^l \times X \to \{0,1\}^l \times X$ for every l, where

$$f_l(u, x) = (u, f(x)).$$

Namely, a function being one-way means that it is hard to determine *the entirety* of $f^{-1}(y)$ with y randomly chosen. An adversary may easily determine the first l bits of ${f_l}^{-1}(u, y)$ (they are simply the string u), but determining the remaining part is just as hard as inverting the original function f.

The security proof for this variant of f-OAEP^{++} is a straightforward consequence of Corollary 6.3, except that we now have a situation where the plaintext length k_m is variable (with a fixed k_m, Corollary 6.3 would apply directly). We need to analyze the OAEP^{++} components h, E, and g:

- The key derivation function h must be extended to accept inputs of any size; in the random oracle model, all outputs are assumed to be independent and uniformly random in the usual manner. Note that each input s to the key derivation function h is tightly related to a certain message length; if $|s| = k_s$, then the corresponding message length is $k_s - k_p$.

- Assuming that h is collision-resistant (meaning, in the random oracle model, that the output length is large enough), each key input u to E will correspond to at most one known s such that $h(s) = u$ with overwhelming probability; thus u is tightly related to at most one message length with overwhelming probability.

- Before applicable to this extended scheme, the mask generation function g needs to be updated to be able to handle outputs of different lengths; hence let g be a function $\{0,1\}^{k_r} \times \mathbb{Z} \to \{0,1\}^*$ ($\{0,1\}^*$ is the set of all bitstrings) such that $g(r, l)$ is a bit string of length l for each positive integer l (l might be upper-bounded by some large constant). Assuming that g is a random oracle in both arguments, we obtain that each input to g is tightly related to a certain output length; the value $g(r, l)$ does not leak any information about any other output from g

in the random oracle model. We stress that this property does not hold for the MGF in standards such as [34]; the first $l' < l$ bits of $g(r, l)$ coincide with $g(r, l')$. However, it is straightforward to update this MGF in the appropriate manner.

We define the extended X-OAEP^{++} encoding method as follows; the input message m is a string of bit length $k_m \geq k - k_p - k_r$.

X-OAEP^{++}-Encode(m, p)
- $m' \leftarrow p \| m$;
- $r \stackrel{R}{\leftarrow} \{0,1\}^{k_r}$;
- $s \leftarrow m' \oplus g(r, k_p + k_m)$;
- $u \leftarrow h(s)$;
- $t \leftarrow E_u(r)$;
- $x \leftarrow t \| s$;
- Output x.

Thus the only difference to the original OAEP^{++} encoding operation is that the X-OAEP^{++} operation accepts messages of different lengths. The decoding operation is defined in the obvious way. The encryption operation is defined as follows; again, the message is a string of bit length at least $k - k_p - k_r$.

f-X-OAEP^{++}-Encrypt(f, m)
- $x \leftarrow$ X-OAEP^{++}-Encode(m, p);
- Write $x = y_0 \| x_1$, where $|x_1| = k$;
- $y_1 \leftarrow f(x_1)$;
- Output $y = y_0 \| y_1$.

With assumptions on the components as above, it is a straightforward exercise to check that Corollary 6.3 holds for f-X-OAEP^{++}; modify the proofs in Appendices A and B in the appropriate manner.

Remark. In the same manner as described in this section, one may define an extended version of f-OAEP. With k as above, if we require that $k_m \geq k - k_p$, then the resulting scheme f-X-OAEP is provably secure. The argument is as follows. In [13] a reduction from "set-partial-f" to f-OAEP is provided. Given a target ciphertext $f(t \| s)$ ($|t| = k_r$), the set-partial-f problem is to output a set of strings including s. In f-X-OAEP, the encrypted seed t is left unmodified. Thus f-X-OAEP can be related to "set-f"; for a given target ciphertext $y = f(x)$, the set-f problem is to output a set X of strings including x. Since f is deterministic, we may easily check for each x' in the set X whether $y = f(x')$; thus we obtain a reduction from an ordinary f-inverter to an f-X-OAEP adversary. In fact, it turns out that

the security reduction in [13] can be substantially tightened for f-X-OAEP, yielding a reduction time linear in the number of queries [19]. Thus the only drawback of f-X-OAEP compared to f-X-OAEP^{++} is an additional ciphertext overhead of k_r bits for small messages.

References

[1] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway. Relations among Notions of Security for Public-Key Encryption Schemes. *Advances in Cryptology – Crypto '98*, pp. 26 – 45. Springer Verlag, 1998.

[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] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. *Advances in Cryptology – Eurocrypt '94*, pp. 92 – 111. Springer Verlag, 1994.

[4] D. Bleichenbacher. Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS #1. *Advances in Cryptology – Crypto '98*, pp. 1 – 12. Springer Verlag, 1998.

[5] D. Boneh. Simplified OAEP for the RSA and Rabin Functions. *Advances in Cryptology – Crypto 2001*, pp. 275 – 291. Springer Verlag, 2001.

[6] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10, pp. 233 – 260, 1997.

[7] D. Coppersmith, M. Franklin, J. Patarin and M. Reiter. Low-Exponent RSA with Related Messages. *Advances in Cryptology – Eurocrypt '96*, pp. 1 – 9. Springer Verlag, 1996.

[8] J.-S. Coron, M. Joye, D. Naccache and P. Paillier. New Attacks on PKCS #1 v1.5 Encryption. *Advances in Cryptology – Eurocrypt 2000*, pp. 369 – 379. Springer Verlag, 2000.

[9] J.-S. Coron, H. Handschuh, M. Joye, P. Paillier, D. Pointcheval, C. Tymen. GEM: A Generic Chosen-Ciphertext Secure Encryption Method. *Cryptographers' Track – RSA Conference 2002*, pp. ?? – ??. Springer Verlag, 2002.

[10] I. Damgard. A Design Principle For Hash Functions. *Advances in Cryptology – Crypto '89*, pp. 416–427. Springer-Verlag, 1990.

[11] T. Dierks and C. Allen. *IETF RFC 2246: The TLS Protocol Version 1.0*. January 1999.

[12] W. Diffie and M. Hellman. Privacy and Authentication: An Introduction to Cryptography. *Proceedings of the IEEE*, 67, pp. 397 – 427, 1979.

[13] E. Fujisaki, T. Okamoto, D. Pointcheval and J. Stern. RSA-OAEP Is Secure under the RSA Assumption. *Advances in Cryptology – Crypto 2001*, pp. 260 – 274. Springer Verlag, 2001.

[14] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28 (2). April 1984.

[15] H. Handschuh and D. Naccache. SHACAL. Submission to the NESSIE project, 2000.

[16] J. Håstad. Solving Simultaneous Modular Equations of Low Degree. *SIAM Journal of Computing*, 17(2), pp. 336 – 341., 1988.

[17] J. Håstad and M. Näslund. The Security of Individual RSA Bits. *39th Annual Symposium on Foundations of Computer Science, FOCS'98*, pp. 510 – 521. IEEE Computer Society, 1998.

[18] 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), pp. 5 – 16, 1994.

[19] J. Jonsson. *How to Combine OAEP With an Arbitrary Trapdoor Permutation.* In preparation.

[20] M. Luby, C. Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM Journal of Computing*, 17(2), pp. 373 – 386, 1988)

[21] J. Manger. A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0. *Advances in Cryptology – Crypto 2001*, pp. 260 – 274. Springer Verlag, 2001.

[22] R.C. Merkle. One Way Hash Functions And DES. *Advances in Cryptology – Crypto '89*, pp. 428 – 446. Springer-Verlag, 1990.

[23] M. Naor and O. Reingold. On the Construction of Pseudo-Random Permutations: Luby-Rackoff Revisited. Journal of Cryptology, pp. 29 – 66, 1999.

[24] National Institute of Standards and Technology (NIST). *Draft FIPS 180-2: Secure Hash Standard.* Draft, May 2001.

[25] National Institute of Standards and Technology (NIST). *FIPS 197: Advanced Encryption Standard (AES).* November 2001.

[26] National Institute of Standards and Technology (NIST). *AES Key Wrap Specification.* November 2001.
Available at www.nist.gov/kms/key-wrap.pdf.

[27] *NESSIE Project.* Information is available at www.cryptonessie.org.

[28] T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. *CT – RSA'2001*, pp. 159 – 175. Springer Verlag, 2001.

[29] M. O. Rabin. *Digitalized Signatures and Public-Key Functions as Intractable as Factorization.* MIT Laboratory for Computer Science Technical Report 212, 1979.

[30] C. Rackoff and D. R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. *Advances in Cryptology – Crypto '91*, pp. 433 – 444. Springer-Verlag, 1992.

[31] R. Rivest, A. Shamir and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2), pp. 120 – 126. February 1978.

[32] R. Rivest, M. Robshaw, R. Sidney, and L. Yin. *The RC6 Block Cipher.* v1.1, August 1998. Available at `www.rsasecurity.com/rsalabs/rc6/`.

[33] RSA Laboratories. *PKCS #1 v1.5: RSA Encryption Standard.* November 1993.

[34] RSA Laboratories. *PKCS #1 v2.0: RSA Encryption Standard.* October 1, 1998.

[35] V. Shoup. OAEP Reconsidered. *Advances in Cryptology – Crypto 2001*, pp. 239 – 259. Springer Verlag, 2001.

[36] V. Shoup. *A Proposal for an ISO Standard for Public Key Encryption.* Preprint, December 2001. Available from eprint.iacr.org/2001/112.

[37] Y. Zheng and J. Seberry. Practical Approaches to Attaining Security Against Adaptively Chosen Ciphertext Attacks. *In Advances in Cryptology – Crypto 2001.* Springer Verlag, 1992.

A Proof of Lemma 6.1

We are given an adversary $\mathcal{A}$ against f-OAEP^{++} and want to define an adversary $\hat{\mathcal{A}}$ against f-S-OAEP^{++} in terms of $\mathcal{A}$. This is done in the following manner. $\hat{\mathcal{A}}$ stores information on an "h-list", an "E-list", and a "U-list". All lists are empty at the beginning of the experiment.

The h-oracle is simulated as follows on input s.

H1 Checks whether s is on the h-list. If this is the case, then $u = h(s)$ is already defined; output u and exit.

H2 Generate a uniformly random string u and output u.

The E-oracle is simulated as follows on input (u, r).

E1 Check whether u is on the h-list. If this is the case, then there is an s such that $h(s) = u$; send (s, r) to the $\hat{E}$-oracle, output the response and exit.

E2 If u is not on the U-list, then add u to the U-list.

E3 Check whether (u, r) is on the E-list. If this is true, then $t = E_u(r)$ is already defined; output t and exit.

E4 Generate a string t uniformly at random such that (u, r', t) is not an entry on the E-list for any r'; add (u, r, t) to the list, output t, and exit.

Note that step 1 is ambiguous if there are several s such that $h(s) = u$. However, we will account for this ambiguity in the security analysis below. The D-oracle is simulated in a manner analogous to the E-oracle simulation, with E and $\hat{E}$ replaced with D and $\hat{D}$ and with r and t swapped.

The other oracles are easy to simulate. For each decryption query y, $\hat{\mathcal{A}}$ applies her own decryption oracle to y and returns the response; note that this oracle applies the $\hat{D}$-oracle and the g-oracle. When $\mathcal{A}$ triggers the challenge generator with a pair (m_0^*, m_1^*), $\hat{\mathcal{A}}$ triggers her own generator with the very same pair and returns the response to $\mathcal{A}$; this oracle applies the g-oracle and the $\hat{E}$-oracle. Finally, g-oracle queries are just passed on to $\hat{\mathcal{A}}$'s g-oracle.

We need to compute the probability that this simulation fails. The part that may go wrong is that $\hat{\mathcal{A}}$ may not provide a perfect simulation of E and D. More precisely, if $h(s) = u$, then we must have that $\hat{E}_s = E_u$. This must hold for *all* s and u that ever occur in the simulation, including those s (within the decryption oracle and target generator computations) that are never queried to the h-oracle.

Specifically, let S be the set of all elements $s \in \{0,1\}^{k-k_r}$ that are part of an h-oracle query or an intermediate step in a decryption oracle or target generator computation (i.e., the underlying ciphertext is equal to $f(t\|s)$ for

some t). Let U be the set of all elements u on the U-list. An error occurs only if $h(s) = h(s')$ for some distinct $s, s' \in S$ or if $h(s) = u$ for some $s \in S$ and $u \in U$. Namely, otherwise $\hat{E}_s$ will provide a perfect simulation of $E_{h(s)}$ for each $s \in S$, and $\hat{\mathcal{A}}$ will provide a perfect simulation of E_u for each $u \in U$.

There are at most $q_f + q_h + 1$ elements in S and at most q_E elements in U. First consider elements in S. The probability of an h-collision among the elements in S is bounded by

$$(q_h + q_f + 1)(q_h + q_f) \cdot 2^{-k_u - 1}.$$

Next consider elements in U. By definition, an element u is in U if u is part of an E- or D-query before being the response to an h-oracle query. The probability that $u = h(s)$ for some later h-oracle query s is bounded by $q_h \cdot 2^{-k_u}$. The probability that $u = h(s)$ for some $s \in S$ that is not queried to the h-oracle is bounded by $(q_f + 1) \cdot 2^{-k_u}$. Namely, such an s appears only within the decryption oracle or the target generator; since $h(s)$ is never computed, the whole simulation is independent of $h(s)$.

Summing over all $u \in U$, we yield the bound

$$q_E(q_h + q_f + 1) \cdot 2^{-k_u}.$$

Thus the total failure probability is bounded by

$$(q_h + q_f + 1)(q_h + q_f + 2q_E) \cdot 2^{-k_u - 1}.$$

At the end, when $\mathcal{A}$ outputs a bit b', $\hat{\mathcal{A}}$ outputs the very same bit. Clearly, the success probability is the same for both adversaries, except that $\mathcal{A}$ may be successful in the true simulation on coin flips leading to a simulation failure in the modified simulation. □

B Proof of Theorem 6.2

We are given an IND-CCA2 adversary $\mathcal{A}$ against f-S-OAEP^{++}; our goal is to define an f-inverter $\mathcal{I}$ in terms of $\mathcal{A}$. To achieve this goal, we must provide a simulation of $\mathcal{A}$'s oracles. Since there are as many as four oracles, it seems prudent to somehow reduce the number of oracles before proceeding. This is done as follows. Define $P : \{0,1\}^k \to \{0,1\}^k$ on input $r \| m'$ ($m' \in \{0,1\}^{k-k_r}$) as

$$P(r \| m') = E_{m' \oplus g(r)}(r) \| m' \oplus g(r).$$

Note that

$$P^{-1}(t \| s) = D_s(t) \| s \oplus g(D_s(t)).$$

We want to define an adversary $\mathcal{B}$ in terms of $\mathcal{A}$ such that $\mathcal{B}$ has access to oracles for P and P^{-1}, but not for the components E, D, and g. Proceed as follows.

In case of a g-oracle query r, $\mathcal{B}$ generates a random string m', sends $r\|m'$ to the P-oracle, and gets as response a string $t\|s$. By definition, $s = g(r) \oplus m'$; $\mathcal{B}$ returns $s \oplus m'$ to $\mathcal{A}$.

In case of an E-oracle query (s, r), $\mathcal{B}$ again generates a random string m', sends $r\|m'$ to the P-oracle, and gets as response a string $t'\|s'$. Next, $\mathcal{B}$ sends the string $r\|(s \oplus s' \oplus m')$ to the P-oracle. Since $g(r) = s' \oplus m'$, the response is $E_s(r)\|s$, from which $\mathcal{B}$ easily extracts $E_s(r)$, which she returns to $\mathcal{A}$.

In case of a D-oracle query (s, t), $\mathcal{B}$ sends $t\|s$ to the P^{-1}-oracle and gets as response a string $D_s(t)\|m'$. $\mathcal{B}$ extracts $r = D_s(t)$ and returns r to $\mathcal{A}$.

In case of a decryption query y, $\mathcal{B}$ sends y to her decryption oracle, who computes $f^{-1}(y)$, applies P^{-1} to the result, extracts the padded plaintext $m' = p'\|m$, and checks whether $p' = p$. If yes, m is returned; otherwise "Decryption Error" is returned.

It is clear that this simulation is perfect; $\mathcal{B}$ does not generate anything herself but uses only her own oracles. The number of P- and P^{-1}-oracle queries from $\mathcal{B}$ is at most $q_g + 2q_E$.

Now, we want to define an inverter $\mathcal{I}$ in terms of $\mathcal{B}$. Let y^* be the target ciphertext; with probability $|X|/2^k$, $f^{-1}(y^*) \notin \{0,1\}^k$ and $\mathcal{I}$ fails. From now on, assume that $f^{-1}(y^*) \in \{0,1\}^k$. Write $f^{-1}(y^*) = t^*\|s^*$.

$\mathcal{I}$ has to simulate the P-oracle, the P^{-1}-oracle, the decryption oracle, and the challenge generator. During the simulation, he stores information on a list. Each entry on the list is a 5-tuple (r, s, t, m', y) such that $E_s(r) = t$, $g(r) = m' \oplus s$, and $f(t\|s) = y$. Whenever we say that a certain value r (or r') is or is not contained in an entry on the list, we are referring to the first position in the entry (not the third position, which may well be equal to r in some entry) and analogous for s, t, and m'.

The P-oracle is simulated as follows on input $r\|m'$.

P1 If there is an entry on the list containing r and m', then $s = g(r) \oplus m'$ and $t = E_s(r)$ are already defined; output $t\|s$ and exit.

P2 If there is an entry on the list containing r, then $g(r)$ is defined; put $s = g(r) \oplus m'$. Otherwise, generate a uniformly random string s.

P3 There is no entry on the list containing both r and s. Generate a random string t such that (s, t) is not part of any entry on the list and define $E_s(r) = t$.

P4 Compute $y = f(t\|s)$ and add (r, s, t, m', y) to the list.

P5 Output $t\|s$.

The P^{-1}-oracle is simulated as follows on input $t\|s$.

PInv1 Compute $y = f(t\|s)$.

PInv2 If there is an entry on the list containing s and t, then $r = D_s(t)$ and $m' = g(r) \oplus s$ are already defined; output $r\|m'$ and exit.

PInv3 There is no entry on the list containing both s and t. Generate a random string r such that (r, s) is not part of any entry on the list and define $D_s(t) = r$.

PInv4 If there is an entry on the list containing r, then $g(r)$ is defined; put $m' = g(r) \oplus s$. Otherwise, generate a uniformly random string m'.

PInv5 Add (r, s, t, m', y) to the list.

PInv6 Output $r\|m'$.

The decryption oracle is simulated as follows on input y.

F1 If y is on the list next to some (r, s, t, m'), then write $p'\|m = m' = g(r) \oplus s$. If $p' = p$, then output m and exit.

F2 Output "Decryption Error".

During the first phase of the algorithm (before the challenge generator is triggered), there is nothing that can go wrong with the simulation of the P- and P^{-1}-oracles since they act in a completely random manner. Yet, the decryption oracle may reject a ciphertext that is actually valid. Let BadReject1 be this event.

After the first phase, $\mathcal{B}$ sends two messages m_0^* and m_1^* to the challenge generator. Clearly, those messages are independent from y^*. $\mathcal{I}$ flips a coin b, generates a random string $r^* \in \{0,1\}^{k_r}$ and defines (implicitly) $g(r^*) = s^* \oplus (p\|m_b^*)$ and $E_{s^*}(r^*) = t^*$. $\mathcal{I}$ adds the entry $(r^*, ?, ?, m_b^*, y^*)$ to the list (question marks since $\mathcal{I}$ does not know s^* and t^*). Since the first phase is independent from s^*, m_b^* is independent from s^*, which implies that the string $s^* \oplus (p\|m_b^*)$ is completely random from the view of $\mathcal{B}$. In particular, this is a perfect simulation of the challenge generator in terms of unpredictability.

Yet, there are a few possible errors that may occur in this step. Namely, the definitions $g(r^*) = s^* \oplus (p\|m_b^*)$ and $E_{s^*}(r^*) = t^*$ must be consistent with earlier oracle queries. This is true if rQuery1 or sQuery1 have not occurred, where rQuery1 is the event that r^* is part of some step of the above simulations (implicitly in the case of the decryption oracle queries) and sQuery1 is defined analogously for s^*.

Now, the experiment continues with the second phase with oracles simulated in the same manner as before with two exceptions:

- The decryption oracle refuses to decrypt the ciphertext y^*.
- In step **PInv1** of the P^{-1}-oracle simulation, if $y = y^*$, then replace the list entry $(r^*, ?, ?, m^*, y^*)$ with (r^*, s, t, m^*, y^*). Note that $\mathcal{I}$ knows the inverse of y^* if this happens.

Again, there is a possibility that a decryption oracle query is erroneously rejected; let BadReject2 be this event and let BadReject be the event that either of BadReject1 or BadReject2 occurs. Also, we consider as bad the event that a P^{-1}-oracle query different from (s^*, t^*) has response r^* or that a P-oracle query contains r^* before (s^*, t^*) is a P^{-1}-oracle query; let rQuery2 be this event and let rQuery be the event that either of rQuery1 or rQuery2 occurs. In this event we include the case that r^* turns up as an intermediate value in a decryption query (though in practice the decryption oracle as simulated by $\mathcal{I}$ aborts without consulting the P^{-1}-oracle).

In the end, $\mathcal{B}$ outputs a bit b'. In case r^* is not part of any query, $\mathcal{B}$ cannot guess b with probability better (or worse) than $1/2$ ($g(r^*) = s^* \oplus (p \| m^*_{b'})$ with probability $1/2$). Thus the advantage of $\mathcal{B}$ is 0 in this case. Assume that sQuery1 or rQuery have not occurred but that $\mathcal{B}$ knows $g(r^*)$. The only possibility is that (s^*, t^*) is a P^{-1}-oracle query in the second phase. However, this implies that $\mathcal{I}$ knows the f-inverse of y^*. We conclude that the success probability for $\mathcal{I}$ is at least

$$\epsilon - \Pr(\mathsf{sQuery1}) - \Pr(\mathsf{rQuery}) - \Pr(\mathsf{BadReject}). \tag{3}$$

The probability that rQuery occurs is at most $(q_f + q_g + 2q_E)2^{-k_r}$ (there are at most $q_f + q_g + 2q_E$ different r^* that can turn up in P- and P^{-1}-oracle queries from $\mathcal{A}$ or the decryption oracle). Similarly, the probability that sQuery1 occurs is at most $(q_f + q_g + 2q_E)2^{-(k-k_r)}$. Thus

$$\Pr(\mathsf{sQuery1}) + \Pr(\mathsf{rQuery}) < (q_f + q_g + 2q_E)\left(2^{-k_r} + 2^{-k_p}\right). \tag{4}$$

It remains to analyze the event BadReject. Let y be the first failed decryption query (a rejected but valid ciphertext). Write $f^{-1}(y) = t \| s$. By construction, (s, t) is not included in the list by the time of the query. Thus $r = D_s(t)$ is a completely random string, except that we know that r cannot be equal to $r' = D_s(t')$ for previously known values r'.

For each earlier rejected decryption query y', there are implicit values r', s', t', m'' that are never computed; add the 5-tuple (r', s', t', m'', y') to the list of entries. Also, add the target 5-tuple $(r^*, s^*, t^*, m^*_b, y^*)$ to the list. Let $q \leq (q_f - 1) + 1 + q_g + 2q_E = q_0$ be the number of entries on this list at the time of the decryption query under consideration. Let q_1 be the number of entries including s; for each such entry, the corresponding r' cannot be equal to r (namely, this would imply either that $\mathcal{I}$ would be able to simulate a correct response to the decryption query or that y is an earlier decryption query). This means that there are $2^{k_r} - q_1$ possibilities for r.

We need to compute the probability that $g(r) \oplus s$ starts with the string p. In case there are at most φ values r' on the list such that $g(r') \oplus s$ starts with p, this probability is at most

$$\frac{2^{k_r} - (q - q_1)}{2^{k_r} - q_1} \cdot 2^{-k_p} + \frac{\varphi}{2^{k_r} - q_1} < 2^{-k_p} + 2\varphi \cdot 2^{-k_r}.$$

Namely, either r is not on the list, in which case $g(r)$ is unknown and the probability of decryption failure is 2^{-k_p}, or r is on the list, in which case there are at most φ possibilities for $g(r)$ to be bad. The inequality follows from the assumption that $2q_0 \leq 2^{k_r}$; note that $q_1 \leq q_0$.

The probability that there are more than φ elements r' on the list such that $g(r') \oplus s$ starts with p is at most the probability of an $(\varphi+1)$-collision among q random strings of length k_p (note that the adversary may form a decryption query from an optimal s). This probability is less than $q^{\varphi+1} \cdot 2^{-\varphi k_p}$ (there are less than $q^{\varphi+1}$ subsets with $\varphi+1$ elements and each subset has the desired property with probability $2^{-\varphi k_p}$). By assumption,

$$q^{\varphi+1} \cdot 2^{-\varphi k_p} \leq 2^{-k_p},$$

which implies that the probability that $g(r) \oplus s$ starts with the string p is bounded by

$$2^{-k_p} + 2\varphi \cdot 2^{-k_r} + 2^{-k_p} = 2 \cdot 2^{-k_p} + 2\varphi \cdot 2^{-k_r}.$$

The total probability of BadReject is hence less than

$$2q_f \cdot 2^{-k_p} + 2q_f\varphi \cdot 2^{-k_r}. \tag{5}$$

Combining (3), (4), and (5), we obtain that the probability that the inverter fails but not the adversary is bounded by

$$\begin{aligned} & (q_f + q_g + 2q_E)\left(2^{-k_r} + 2^{-k_p}\right) + 2q_f \cdot 2^{-k_p} + 2q_f\varphi \cdot 2^{-k_r} \\ < \quad & ((2\varphi+1)q_f + q_g + 2q_E)\left(2^{-k_r} + 2^{-k_p}\right). \end{aligned}$$

This concludes the proof. □

C AES-based instantiation of OAEP^{++}

Note that OAEP and OAEP$^+$ can be defined in terms of one single hash function such as SHA-1 or SHA-256 [24]; there are standardized mask generation functions defined in terms of a hash function (see [34]). It would be possible to define also OAEP^{++} in terms of SHA-1 or SHA-256; use the underlying compression function as the block cipher E (compare to [15]).

However, this construction works only for hash functions that are based on a block cipher and is in addition highly non-standard. It seems more natural to go in the other direction, defining a mask generation function and a key derivation function in terms of the block cipher E. Here, AES [25] seems to have attractive properties; recall that there are three possible key sizes (128, 192, and 256 bits). For independence between the three functions, one may base the mask generation function on 128-bit AES and

the key derivation function on 192-bit AES, and use 256-bit AES as the block cipher.

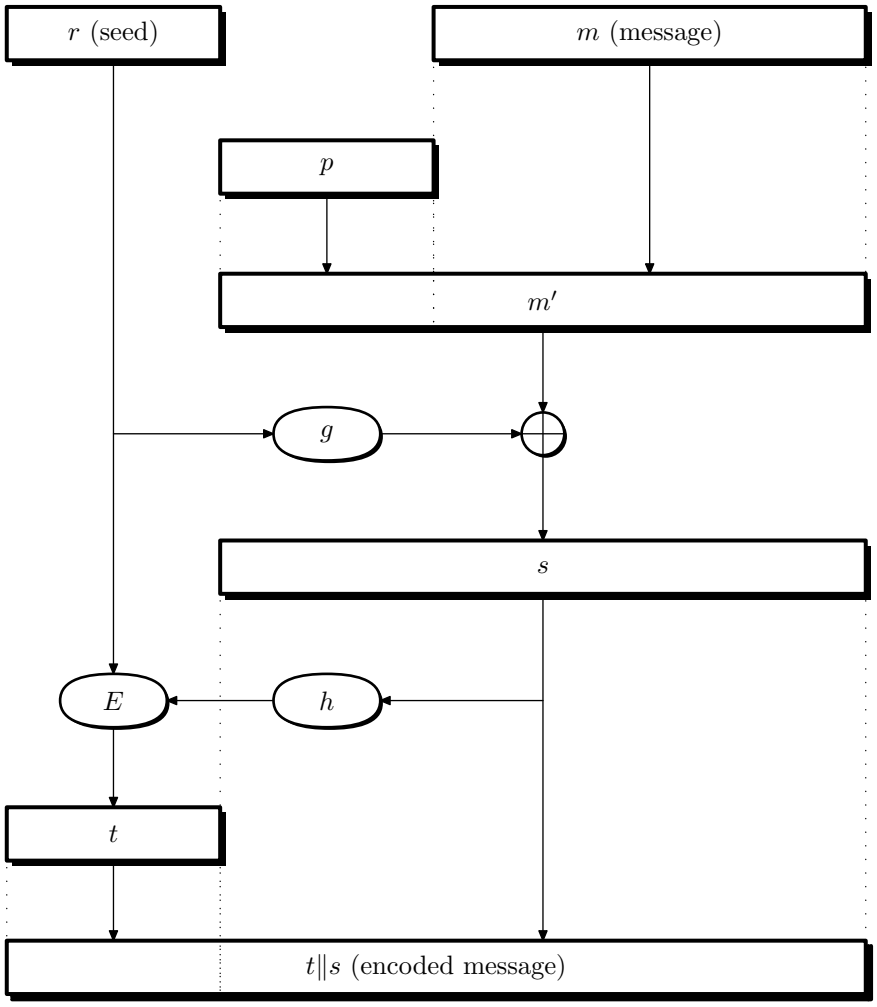
Defining a mask generation function with input length the block length $k_r = 128$ of AES is not very hard. The most obvious approach would be to apply CTR mode, thus defining $g(r)$ as the first $k - k_r$ bits of

$$E_r(a+0)||E_r(a+1)||E_r(a+2)||\dots$$

(addition modulo 2^{k_r}). $E_r(i)$ is the encryption of the k_r-bit representation of the integer i, while a is a fixed initialization vector chosen in the appropriate manner. For example, the 32 most significant bits of a might be a 32-bit representation of the desired output length $k - k_r$. Assuming that E is an ideal block cipher and $(k - k_r)/k_r$ is significantly smaller than $2^{k_r/2}$, this construction is almost as secure as a random oracle producing uniformly random outputs. Namely, the output will be close to indistinguishable from uniformly random output if $k - k_r$ is not very large; due to the birthday paradox, the lack of colliding output blocks from E_r would be a concern for very large values of $k - k_r$.

Defining a key derivation function (KDF) appears to be harder, especially if E is a block cipher such as AES with block length only 128 bits and the desired KDF output length is 256 bits.

AUTHOR'S NOTE. In an attempt to define an AES-based version of RSA-KEM [36], RSA Laboratories is currently examining different potential AES-based KDF constructions; our hope is to find a function with a security that can be tightly related to the security of the underlying block cipher. Clearly, such a KDF would be useful within OAEP^{++} as well.



OAEP^{++} encoding operation.