Simplified OAEP for the RSA and Rabin Functions

Dan Boneh*

Computer Science Department, Stanford University
dabo@cs.stanford.edu

Abstract. Optimal Asymmetric Encryption Padding (OAEP) is a technique for converting the RSA trapdoor permutation into a chosen ciphertext secure system in the random oracle model. OAEP padding can be viewed as two rounds of a Feistel network. We show that for the Rabin and RSA trapdoor functions a much simpler padding scheme is sufficient for chosen ciphertext security in the random oracle model. We show that only one round of a Feistel network is sufficient. The proof of security uses the algebraic properties of the RSA and Rabin functions.

1 Introduction

In an influential paper Bellare and Rogaway [2] introduced the Optimal Asymmetric Encryption Padding (OAEP) system. OAEP is most commonly used for strengthening the RSA and Rabin encryption schemes. OAEP is widely deployed and appears in several standards. Shoup [11] recently described a modification to OAEP called OAEP+ that provably converts any trapdoor permutation into a chosen ciphertext secure system in the random oracle model. Shoup also showed that applying OAEP to the RSA permutation with public exponent $e = 3$ gives a chosen ciphertext secure system in the random oracle model. Fujisaki et al.[8] were able to extend the result and prove that the same holds for the RSA permutation with any RSA public exponent e.

We show that for the RSA and Rabin systems, much simpler padding schemes can be shown to be chosen ciphertext secure in the random oracle model. We introduce two simple padding schemes. The first is called Simple-OAEP, or SAEP for short. The second is called SAEP$^+$. We note that simplifying the padding scheme makes the system easier to describe and easier to implement, and thus is more elegant. Simplifying the padding scheme has little bearing on performance since padding time is negligible compared to public key operations.

We begin by describing SAEP and SAEP$^+$ padding (see Figure 1). Let M be a message $M \in \{0,1\}^m$ and let r be a random string $r \in \{0,1\}^{s_1}$. Let H be a hash function from $\{0,1\}^{s_1}$ to $\{0,1\}^{m+s_0}$. Let G be a hash function from $\{0,1\}^{m+s_1}$ to $\{0,1\}^{s_0}$. Define the new padding schemes SAEP and SAEP$^+$ as follows:

* Supported by NSF and the Packard Foundation.

$$\mathsf{SAEP}(M,r) = \big(\,(M \,\|\, \quad 0^{s_0} \quad\ ) \oplus H(r)\big) \;\big\|\; r$$
$$\mathsf{SAEP}^{+}(M,r) = \big(\,(M \,\|\, G(M\|r)\ ) \oplus H(r)\big) \;\big\|\; r$$

These padding schemes are to be used as preprocessing functions with the Rabin or RSA trapdoor functions. To encrypt a message $M \in \{0,1\}^m$ first pick a random $r \in \{0,1\}^{s_1}$, compute $y = \mathsf{SAEP}(M,r)$, and set $C = y^2 \bmod N$ or $C = y^e \bmod N$ for some RSA exponent e.

Both schemes provide security against an adaptive chosen ciphertext attack in the random oracle model for appropriate values of m, s_0, s_1. Let N be an n-bit modulus. We prove the following results for the Rabin and RSA functions:

SAEP: Let Rabin-SAEP be the encryption scheme resulting from combining SAEP with the Rabin trapdoor function, $f(x) = x^2 \bmod N$ (as described in the next section). We show that Rabin-SAEP provides chosen ciphertext security whenever $m + s_0 < n/2$ and $m < n/4$. Security is based on the hardness of factoring large RSA composites. The reduction is very efficient. It is based entirely on applying Coppersmith's algorithm [6] to quadratic and quartic polynomials. SAEP works well with the Rabin function, but is hard to use with RSA, as explained in Section 4.

SAEP$^+$: Both RSA-SAEP$^+$ (for any RSA exponent e) and Rabin-SAEP$^+$ can be shown to be chosen ciphertext secure whenever $m + s_0 < n/2$. The reduction to factoring for Rabin-SAEP$^+$ is extremely efficient. The proof is based on Coppersmith's algorithm. For RSA-SAEP$^+$ the reduction to breaking RSA is less efficient. Its running time is similar to the running time of the reduction in the proof of security for RSA-OAEP [8].

SAEP$^+$ is more flexible than SAEP in a number of ways. First, SAEP$^+$ can be used with both Rabin and RSA (although Rabin is preferred). Second, SAEP$^+$ can encrypt messages of longer size. For example, when using a 1024 bit modulus ($n = 1024$) one often takes $s_0 = 128$ for proper security. In this case, the maximum message length in SAEP is 256 bits. In SAEP$^+$ the maximum length is 384 bits. Note that since a 1024-bit modulus is often used for transporting a 128-bit session-key, both SAEP and SAEP$^+$ are adequate for this purpose.

In some cases it might be desirable to allow for longer messages to be encrypted with SAEP$^+$. In Section 5 we note that the proof of security for RSA-SAEP$^+$ can be extended so that the scheme is secure whenever $m + s_0 < n(1-\delta)$ for any fixed $\delta > 0$. This means M could be almost as long as the modulus. However, the efficiency of the reduction to breaking RSA degrades exponentially in $\frac{1}{\delta}$. Hence, throughout the paper we stick with $\delta = 1/2$. The extended proof is based on solutions to the Hidden Number Problem [4] modulo a composite.

Both SAEP and SAEP$^+$ work best with the Rabin function. The resulting systems are better than their RSA counterparts in all aspects: (1) encryption is slightly faster, (2) the reduction given in the security proof is more efficient, and (3) security relies on the difficulty of factoring rather than the difficulty of inverting the RSA permutation.

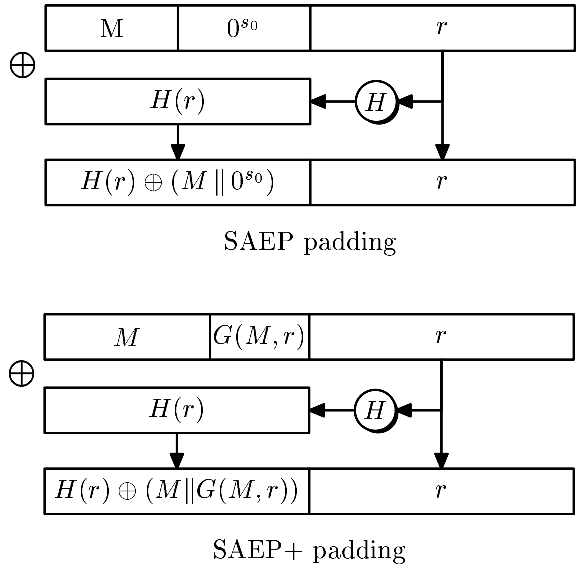

Fig. 1. SAEP and SAEP$^+$ padding

Comparison of OAEP *and* SAEP. OAEP, presented by Bellare and Rogaway, and OAEP+, presented by Shoup, both provide chosen ciphertext security for the RSA trapdoor permutation (although OAEP+ has a more efficient security proof). These padding schemes are defined as follows:

$$\mathsf{OAEP}(M,r) = \big(\quad (M\|0^{s_0}) \,\oplus\, H(r) \quad \big) \,\big\|\, \big(r \oplus G(\quad (M\|0^{s_0}) \,\oplus\, H(r) \quad)\big)$$

$$\mathsf{OAEP+}(M,r) = \big(\, (M \oplus H(r)) \,\|\, W(M,r)\,\big) \,\big\|\, \big(r \oplus G(\, (M \oplus H(r)) \,\|\, W(M,r)\,)\big)$$

where H, G, W are hash functions. Schematically both OAEP and OAEP+ look like two rounds of a Feistel network. Clearly the new padding schemes, SAEP and SAEP$^+$ are simpler. These new schemes are only a single round of a Feistel network.

Although the new padding schemes are simpler than OAEP, they are slightly more restrictive. Using OAEP and OAEP+ one can encrypt messages that are almost as long as the modulus. For example, for a 1024-bit modulus it is safe to encrypt messages that are 768-bits long. In contrast, using the same modulus size, SAEP$^+$ can only encrypt 384-bit messages. This difference is irrelevant for common applications (e.g. key transport), but is worth pointing out.

1.1 Chosen ciphertext security

Adaptive chosen ciphertext security is the accepted notion for secure encryption. We have confidence in this notion since it captures a wide range of attacks, and is equivalent to several other useful security notions [7,3]. We present the definition due to Rackoff and Simon [12]. Define a (t, q_D) chosen ciphertext attack algorithm $\mathcal{A}$ as a t-time algorithm that interacts with a challenger as follows:

Setup: The challenger generates a public/private key pair. It gives the public key to the attacker $\mathcal{A}$ and keeps the private key to itself.

Phase I: The attacker $\mathcal{A}$ issues decryption queries for various ciphertexts C. The challenger responds with the decryption of all valid ciphertexts.

Challenge: At some point algorithm $\mathcal{A}$ outputs two messages M_0, M_1. The challenger responds with a ciphertext C^* which is the encryption of M_b where b is randomly chosen in $\{0,1\}$.

Phase II: The attacker $\mathcal{A}$ continues to issue decryption requests C, subject to the constraint $C \neq C^*$. Finally algorithm $\mathcal{A}$ terminates and outputs $b' \in \{0,1\}$.

We say that the attacker is successful if $b = b'$. During the attack the attacker is allowed to make at most q_D decryption queries. We define the adversary's advantage as: $\mathrm{adv}(\mathcal{A}) = \left|\Pr[b = b'] - \frac{1}{2}\right|$
We say that a system is (t, ϵ, q_D) secure if no (t, q_D) attacker has advantage more than ϵ.

Random oracles: To analyze the security of certain natural constructions Bellare and Rogaway introduced an idealized world called the random oracle model [1]. A system that has chosen ciphertext security in this idealized world is said to be chosen ciphertext secure in the random oracle model. Security in the random oracle model does not imply security in the real world [5]. Nevertheless, the random oracle model is a useful tool for validating natural constructions. Given an encryption scheme using hash functions $H_1, \ldots, H_n$ we use $(t, q_D, q_{H_1}, \ldots, q_{H_n})$ to denote a (t, q_D) chosen ciphertext attacker that makes at most q_{H_i} queries to the hash function H_i.

1.2 Coppersmith's algorithm

The proofs of security for SAEP and SAEP$^+$ are based on an important result due to Coppersmith [6]. Coppersmith proved the following theorem:

Theorem 1 (Coppersmith). *Let N be an integer and let $f(x) \in \mathbb{Z}_N[x]$ be a monic polynomial of degree d. Then there is an efficient algorithm to find all $x_0 \in \mathbb{Z}$ such that $f(x_0) = 0 \bmod N$ and $|x_0| < N^{1/d}$.*

We denote by $T_C(N, d)$ the running time of Coppersmith's algorithm when finding roots of a polynomial $f \in \mathbb{Z}[x]$ of degree d. In our proofs we only apply Coppersmith's algorithm to quadratic and quartic polynomials.

2 Full description of SAEP and SAEP$^+$

We now give a full description of the SAEP and SAEP$^+$ systems for RSA and Rabin. We first describe these schemes as they apply to the Rabin function.

In doing so we deal with complications that arise from the fact that $f(x) = x^2 \bmod N$ is not a permutation of $\mathbb{Z}_N^*$. Let m, s_0, s_1 be security parameters. Set $n = m + s_0 + s_1$. We will make use of a hash function $H : \{0,1\}^{s_1} \to \{0,1\}^{m+s_0}$. The Rabin-SAEP system is composed of three algorithms: key-gen, encrypt, decrypt. We describe each of these algorithms in turn:

key-gen: The key generation algorithm takes a security parameter n and produces an $(n+2)$-bit RSA modulus $N = pq$ where p and q are $(n/2+1)$-bit primes. We require that $p = q = 3 \bmod 4$. We also require that $N \in [2^{n+1}, 2^{n+1} + 2^n)$, i.e. that the two most significant bits of N are '10'. Any of the standardized methods can be used to generate p and q [9]. The public key is N. The private key is the factorization of N, namely $\langle p, q \rangle$.

encrypt: We wish to encrypt a message $M \in \{0,1\}^m$:

Step 1: Pick a random $r \in \{0,1\}^{s_1}$.

Step 2: Set $t = 0^{s_0}$.

Step 3: Set $v = M \| t \in \{0,1\}^{m+s_0}$.

Step 4: Set $x = v \oplus H(r)$.

Step 5: Set $y = x \| r \in \{0,1\}^n$. We view y as an n-bit integer. Note that $y < N/2$.

Step 6: Define the ciphertext C as $C = y^2 \bmod N$.

decrypt: Given a ciphertext $C \in \mathbb{Z}_N$ we decrypt using the steps below. We let A and B be the Chinese Remainder coefficients, i.e. A is $1 \bmod p$ and $0 \bmod q$, and B is $0 \bmod p$ and $1 \bmod q$.

Step 1: Compute $z_p = C^{\frac{p+1}{4}} \bmod p$ and $z_q = C^{\frac{q+1}{4}} \bmod q$. Since $p = q = 3 \bmod 4$ it follows that z_p, z_q are square roots of C in $\mathbb{Z}_p, \mathbb{Z}_q$ respectively.

Step 2: Test that $z_p^2 = C \bmod p$ and $z_q^2 = C \bmod q$. If either condition does not hold, then C is not a quadratic residue in $\mathbb{Z}_N$. Reject this C as an invalid ciphertext.

Step 3: Set $y_1 = A \cdot z_p + B \cdot z_q \bmod N$ and $y_2 = A \cdot z_p - B \cdot z_q \bmod N$. The four square roots of $C \bmod N$ are $\pm y_1$ and $\pm y_2$. Two of these four roots must be greater than $N/2$ and hence can be discarded. Let y_1, y_2 be the two remaining square roots. If neither of y_1, y_2 is in $[0, 2^n)$ then reject C as an invalid ciphertext. Without loss of generality we assume both y_1, y_2 are in $[0, 2^n)$.

Step 4: View both y_1 and y_2 as strings in $\{0,1\}^n$. Write $y_1 = x_1 \| r_1$ and $y_2 = x_2 \| r_2$ with $x_1, x_2 \in \{0,1\}^{m+s_0}$ and $r_1, r_2 \in \{0,1\}^{s_1}$.

Step 5: Set $v_1 = x_1 \oplus H(r_1)$ and $v_2 = x_2 \oplus H(r_2)$.

Step 6: Write $v_1 = M_1 \| t_1$ and $v_2 = M_2 \| t_2$ where $M_1, M_2 \in \{0,1\}^m$ and $t_1, t_2 \in \{0,1\}^{s_0}$.

Step 7: For $i = 1, 2$ test if t_i is equal to 0^{s_0}. If this condition holds for either none or both of v_1, v_2 then reject C as an invalid ciphertext.

Step 8: Let $i \in \{1, 2\}$ be the unique i for which the condition of Step 7 holds. Output M_i as the decryption of C.

Note that in Step 7, if both t_1 and t_2 are equal to 0^{s_0} the decryptor cannot choose between them. Hence, in this case the ciphertext is rejected. This means that with very low probability, namely 2^{-s_0}, a valid ciphertext might be rejected by the decryptor (recall that typically $s_0 \geq 128$). For most applications such low error probabilities can be ignored. One concern is whether a malicious encryptor can create a valid ciphertext that will be rejected by the decryptor in Step 7. It is easy to show that in the random oracle model the encryptor would have to spend expected time $O(2^{s_0})$ to create such a ciphertext. This is sufficient for most applications. We note that if a negligible error probability is unacceptable then the encryptor could keep choosing random r's until y has Jacobi symbol 1. This enables the decryptor to select the correct square root by choosing the unique root $y_i \in [0, 2^n)$ with Jacobi symbol 1. However, this is unnecessary and makes the scheme less efficient.

During decryption invalid ciphertexts can be rejected in Steps 2 and 3 as well as in Step 7. Manger [10] points out the importance of preventing an attacker from distinguishing between rejections at the various steps, say, using timing analysis. Implementors must ensure that the reason a ciphertext is rejected is hidden from the outside world. Indeed, our proof of security fails if this is not the case.

*Description of Rabin-*SAEP$^+$*:* The description of Rabin-SAEP$^+$ is very similar to Rabin-SAEP. SAEP$^+$ makes use of an additional hash function $G : \{0,1\}^{m+s_1} \to \{0,1\}^{s_0}$. Key generation for Rabin-SAEP$^+$ is identical to key generation for Rabin-SAEP. Encryption differs only in Step 2 where t is defined as $t = G(M, r) \in \{0,1\}^{s_0}$. Decryption differs only in Step 7 where the condition tested is whether t_i is equal to $G(M_i, r_i)$.

The description of RSA-SAEP$^+$ is analogous to the one given above. Decryption is a bit simpler since one does not have to worry about multiple preimages to the RSA trapdoor permutation.

2.1 Complexity assumptions

Throughout the paper we use the following standard complexity assumptions:

Factoring assumption: We say that a t-time algorithm $\mathcal{B}$ is an (n, t) factoring algorithm with advantage ϵ if $\mathcal{B}$ succeeds with probability at least ϵ in factoring n-bit integers generated by the key-gen algorithm. The probability is over the random bits used by algorithms key-gen and $\mathcal{B}$. We write $\mathrm{adv}(\mathcal{B}) = \epsilon$. We say that the (n, t, ϵ) factoring assumption holds if there is no (n, t) factoring algorithm with advantage ϵ.

RSA assumption: We say that a t-time algorithm $\mathcal{B}$ is an (n, e, t) algorithm for computing e'th roots in $\mathbb{Z}_N$ with advantage ϵ if $\mathcal{B}$ succeeds with probability at least ϵ in computing $x^{1/e} \bmod N$ for an n-bit integer N generated by the key-gen algorithm and a random $x \in \mathbb{Z}_N$. The probability is over x and the random bits used by algorithms key-gen, $\mathcal{B}$. We write $\mathrm{adv}(\mathcal{B}) = \epsilon$. We say that the (n, e, t, ϵ) RSA assumption holds if there is no (n, e, t) algorithm with advantage ϵ.

3 Two simple facts

We state two simple facts that will be useful in the proof of security.

Fact 2. *Let $N = pq$ be an $n + 2$-bit integer generated by the* key-gen *algorithm, i.e. $N \in [2^{n+1}, 2^{n+1} + 2^n)$. Let α be a random integer in $[0, 2^n)$ and $C^* = \alpha^2$. Then with probability at least $1/3$ (over the choice of α) there exist two distinct integers $y_1^*, y_2^* \in [0, 2^n)$ such that $(y_1^*)^2 = (y_2^*)^2 = C^* \bmod N$.*

Proof. The condition $N \in [2^{n+1}, 2^{n+1} + 2^n)$ implies that $2^n < N/2$ and that $N/2^{n+1} < 3/2$. Let $\alpha \in [0, 2^n)$. Since $2^n < N/2$ we know that $C^* = \alpha^2 \bmod N$ always has either one or two square roots in $[0, 2^n)$. Let A be the number of $\alpha \in [0, 2^n)$ so that $\alpha^2 \bmod N$ has one root in $[0, 2^n)$. Let B be the number of $\alpha \in [0, 2^n)$ so that $\alpha^2 \bmod N$ has two roots in $[0, 2^n)$. We know $A + B = 2^n$. Furthermore, we know that for every $\alpha \in [0, 2^n)$ relatively prime to N we have that $\alpha^2 \bmod N$ has exactly two roots in $[0, N/2)$. The number of α not relatively prime to N is at most $p + q$. Therefore, $A < (N/2 - 2^n) + p + q$ and hence $B > 2^n - (N/2 - 2^n) - p - q = 2^{n+1} - N/2 - p - q$. We get that:

$$\frac{B}{2^n} > 2 - \frac{N}{2^{n+1}} - \frac{p+q}{2^n} > \frac{1}{2} - \frac{p+q}{2^n} > \frac{1}{3}$$

□

Fact 3. *Let $N = pq$ be an $n + 2$-bit integer generated by the* key-gen *algorithm. Let α be a random integer in $[0, 2^n)$ and set $C^* = \alpha^2$. Let $y_1^*, y_2^* \in [0, 2^n)$ be two integers such that $(y_1^*)^2 = (y_2^*)^2 = C^* \bmod N$. When C^* has two distinct roots in $[0, 2^n)$ we assume $y_1^* \neq y_2^*$, otherwise set $y_1^* = y_2^*$. Let c be a random bit in $\{1, 2\}$. Then y_c^* is a uniform random variable in $[0, 2^n)$ over the choice of (α, c).*

The proof of Fact 3 is immediate.

4 Proof of security of Rabin-SAEP

We show that an attacker capable of mounting a successful adaptive chosen ciphertext attack on Rabin-SAEP in the random oracle model can be used to efficiently factor large integers. We use m, s_0, s_1 as the security parameters of SAEP and set $n = m + s_0 + s_1$. Recall that the SAEP key-gen algorithm generates an $(n + 2)$-bit modulus N.

Theorem 4. *Let $N = pq$ be an integer generated by the Rabin-*SAEP key-gen *algorithm given the security parameter n. We assume $m < n/4$ and $m + s_0 < n/2$. Let $\mathcal{A}$ be a (t, q_D, q_H) chosen ciphertext attack algorithm in the random oracle model. Suppose $\mathcal{A}$ has advantage ϵ when attacking Rabin-*SAEP *modulo N. Then there is a uniform algorithm $\mathcal{B}$ for factoring N with the following parameters:*

$$time(\mathcal{B}) = time(\mathcal{A}) + O\big(q_D q_H T_C + q_D T_C'\big)$$
$$adv(\mathcal{B}) \geq \tfrac{1}{6} \cdot adv(\mathcal{A}) \cdot (1 - \frac{2q_D}{2^{s_0}} - \frac{2q_D}{2^{s_1}})$$

Here $T_C = T_C(n, 2)$ and $T_C' = T_C(n, 4)$.

Proof of Theorem 4. Let N be an $(n+2)$-bit integer generated by the key-gen algorithm. To factor N algorithm $\mathcal{B}$ begins by picking a random $\alpha \in [0, 2^n)$ and computing $C^* = \alpha^2 \bmod N$. We show an algorithm that takes C^* as input, interacts with $\mathcal{A}$, and outputs a square root $\alpha' \in [0, 2^n)$ of $C^* \bmod N$ with probability at least $\epsilon' = \epsilon \cdot (1 - 2q_D/2^{s_0} - 2q_D/2^{s_1})$. By Fact 2 we know that C^* has two distinct square roots $\gamma, \gamma' \in [0, 2^n)$ with probability at least $1/3$. Therefore, $\alpha \neq \alpha'$ with probability $1/6$. When this happens, we can factor N by computing $\gcd(N, \alpha - \alpha')$. Since both $0 \leq \alpha, \alpha' < N/2$ this is guaranteed to give a non-trivial factor of N. Overall, we succeed in factoring N with probability at least $\frac{1}{6}\epsilon'$ as required.

The rest of the proof focuses on computing a square root α' of C^*. We construct a simulator that given C^* interacts with algorithm $\mathcal{A}$ and produces a root. The simulator responds to $\mathcal{A}$'s decryption queries and H hash queries, and provides algorithm $\mathcal{A}$ with the challenge ciphertext. We first give a high level description of the simulator (the simulator is described in detail below). The simulator gives C^* as the challenge ciphertext to the attacker $\mathcal{A}$. Suppose $C^* = (y_1^*)^2 = (y_2^*)^2 \bmod N$ for some $y_1^*, y_2^* \in [0, 2^n)$ (unknown to the simulator). For $i = 1, 2$ write $y_i^* = x_i^* \| r_i^*$ with $r_i^* \in \{0,1\}^{s_1}$ and $x_i^* \in \{0,1\}^{m+s_0}$. If $\mathcal{A}$ is to have any information about the decryption of C^* we will show that it must either query the function H at a point r_i^* or issue a decryption query involving one of r_1^*, r_2^* as described below. First, we show that once the simulator receives a query for one of $H(r_1^*)$ or $H(r_2^*)$ it can easily deduce a square root of C^*. Given r_i^* we know that x_i^* is a root of $f(x) = (2^{s_1}x + r_i^*)^2 - C^* \bmod N$. Since $x_i^* < 2^{m+s_0} < \sqrt{N}$, the simulator can use Coppersmith's algorithm to find x_i^*. Then $y^* = x_i^* \| r_i^*$ is a square root of C^* as required.

Next, we give a high level description of how the simulator responds to $\mathcal{A}$'s decryption queries. Suppose the attacker issues a decryption query for the ciphertext C. Let $C = y^2 \bmod N$ for some $y \in [0, 2^n)$ and let r be the s_1 least significant bits of y. We will show that if C is a valid ciphertext, then $H(r)$ must already be defined (otherwise, with high probability, the string 0^{s_0} will not be found when unpadding y). Hence, the r used to create C must satisfy one of the following: (1) the attacker queried $H(r)$ prior to issuing the decryption query, or (2) $r = r_1^*$ or $r = r_2^*$. Suppose method (1) is used. Then when the decryption query is issued, the simulator already has r, which enables it to find the square root of C, as above. Suppose method (2) is used, i.e. $r = r_i^*$ for some $i \in \{1, 2\}$. In this case, assuming C is a valid ciphertext, we know that $y = y_i^* + 2^{s_0+s_1}\Delta$ for some $|\Delta| < 2^m < N^{1/4}$. Hence, define the two polynomials:

$$f(z) = z^2 - C^* \quad \text{and} \quad g(z, \Delta) = (z + 2^{s_0+s_1}\Delta)^2 - C$$

Then $f(y_i^*) = g(y_i^*, \Delta) = 0 \bmod N$. Therefore, Δ must be a root of the resultant $h = \mathrm{Res}_z(f, g)$ which is a quartic polynomial in Δ. Since $|\Delta| < N^{1/4}$ we can use Coppersmith's algorithm to find Δ. Using Δ the simulator easily finds y_i^* which is a square root of C^* as required. Hence, decryption queries for valid ciphertexts are either correctly answered or they lead directly to a square root of C^*.

We are now ready to describe the complete simulator for computing square roots. It works as follows:

Setup: The simulator gives $\mathcal{A}$ the value N as the public key to be attacked. It also gives $\mathcal{A}$ the security parameters m, s_0, s_1.

H-queries: At any time $\mathcal{A}$ can query H at $r \in \{0,1\}^{s_1}$. The simulator needs to respond with $H(r)$. To respond to such H-queries the simulator maintains a list, called the H_{list}. The H_{list} is a list of tuples of the form $\langle z, H(z)\rangle$ that records all responses to previous H-queries. The H_{list} is initially empty. To respond to the query r the simulator works as follows:

Step 1: If r already appears as the left hand side of some tuple $\langle z, H(z)\rangle$ in the H_{list} then respond to $\mathcal{A}$ with $H(r) = H(z)$.

Step 2: Consider the polynomial $f(x) = (2^{s_1}x + r)^2 - C^*$. The simulator runs Coppersmith's algorithm to try to find a solution $|x_0| < 2^{m+s_0} < \sqrt{N}$ satisfying $f(x_0) = 0 \bmod N$. If a solution is found, the simulator outputs $2^{s_1}x_0 + r$ as the square root of C^* and terminates the simulation.

Step 3: Otherwise, the simulator picks a random $w \in \{0,1\}^{m+s_0}$ and sets $H(r) = w$. It adds the tuple $\langle r, w\rangle$ to the H_{list} and responds to $\mathcal{A}$ by saying $H(r) = w$.

Challenge: At some point $\mathcal{A}$ produces two plaintexts $M_0, M_1 \in \{0,1\}^m$ where it wishes to be challenged. The simulator responds with C^* as the challenge ciphertext.

Decryption queries: Let $C \in \mathbb{Z}_N$ be a ciphertext output by $\mathcal{A}$. The simulator must decrypt C or reject it as an invalid ciphertext. We construct a plaintext extractor to decrypt C. The plaintext extractor takes C, H_{list}, C^* as input and works as follows:

Step 1: For each tuple $\langle r, H(r)\rangle$ on the H_{list} consider the polynomial $f_r(X) = (2^{s_1}x + r)^2 - C$. The simulator runs Coppersmith's algorithm on each $f_r(x)$ to try to find an $|x_0| < \sqrt{N}$ satisfying $f_r(x_0) = 0 \bmod N$. Suppose an x_0 is found for some r_0 on the H_{list}. In this case, the simulator found a square root of C, namely $2^{s_1}x_0 + r_0$. Using $H(r_0)$ from the H_{list} the simulator checks that x_0 is a properly padded SAEP message. If so, it gives $\mathcal{A}$ the plaintext. If not, the simulator rejects C as an invalid ciphertext.

Step 2: Suppose no r_0 on the H_{list} is found. Consider the two polynomials

$$f(z) = z^2 - C^* \quad \text{and} \quad g(z, \Delta) = (z + 2^{s_0+s_1}\Delta)^2 - C$$

Let $h(\Delta)$ be the resultant of the two polynomials with respect to z. Then $h(\Delta)$ is a quartic polynomial. Use Coppersmith's algorithm to try to find a $\Delta_0 < 2^m < N^{1/4}$ such that $h(\Delta_0) = 0 \bmod N$. If such a Δ_0 is found then we know $f(y^*) = g(y^*, \Delta_0) = 0 \bmod N$ where y^* is some square root of C^*. Then the simulator can easily find y^* by computing the gcd of the univariate polynomials $f(z)$ and $g(z, \Delta_0)$. Since these two monic quadratic polynomials must be different (since $C \neq C^*$) their gcd must be a linear polynomial having y^* as a root. The simulator outputs y^* as the square root of C^* and terminates the simulation.

Step 3: If both Step 1 and Step 2 fail to resolve the decryption query, the ciphertext C is rejected as an invalid ciphertext. Note that Step 2 is only done in Phase 2 of the attack.

This completes the description of the simulator. The simulator's running time is as stated in the statement of Theorem 4. It remains to calculate the success probability of computing a square root of C^*. Let y_1^*, y_2^* be the two square roots of C^* mod N in $[0, 2^n)$. If C^* only has one such square root then set $y_1^* = y_2^*$. Let r_1^*, r_2^* be the s_1 least significant bits of y_1^*, y_2^* respectively. We are successful if during the simulation either: (1) $\mathcal{A}$ issues a query for one of $H(r_1^*), H(r_2^*)$, or (2) $\mathcal{A}$ issues a decryption query for a valid ciphertext $C \neq C^*$ where the s_1 least significant bits of some $\sqrt{C} \in [0, 2^n)$ equal r_1^* or r_2^*. If either one of these queries occurs during the attack we say that $\mathcal{A}$ issued an r^* query. We denote by $\mathcal{A}(r^*)$ the event that $\mathcal{A}$ issues an r^* query during the attack. Our goal is to show that during the simulation $\Pr_{sim}[\mathcal{A}(r^*)]$ is non-negligible.

Lemma 1. *Let $\mathcal{A}$ be a (t, q_D, q_H) chosen ciphertext attacker with $adv(A) \geq \epsilon$. Then* $\Pr_{sim}[\mathcal{A}(r^*)] \geq \epsilon(1 - \frac{2q_D}{2^{s_0}} - \frac{2q_D}{2^{s_1}})$.

Proof. We first note that during the real attack we have $\Pr_{real}[\mathcal{A}(r^*)] \geq \epsilon$. To see this observe that if $\mathcal{A}$ does not issue an r^* query during the real attack then the decryption of the challenge C^* is independent of $\mathcal{A}$'s view (since $H(r_1^*), H(r_2^*)$ are independent of $\mathcal{A}$'s view). Hence, since $\text{adv}(\mathcal{A}) \geq \epsilon$, it follows that in the real attack $\mathcal{A}$ must make an r^* query with probability at least ϵ, i.e. $\Pr_{real}[\mathcal{A}(r^*)] \geq \epsilon$.

Next, we show that with high probability $\mathcal{A}$ cannot distinguish the real attack from the simulation until it issues an r^* query. We say that the event $\mathsf{GoodSim}$ occurred if the following two events happen:

– The simulator never rejects a valid decryption query issued by $\mathcal{A}$ (the validity of a query is determined relative to the oracle H at the end of the simulation), and

– During phase I of the attack (i.e. prior to being given the challenge) algorithm $\mathcal{A}$ did not issue a decryption query for C where $C = y^2 \bmod N$ and the s_1 least significant bits of $y \in [0, 2^n)$ are equal to r_1^* or r_2^*.

We show that when $\mathsf{GoodSim}$ occurs the simulation and the real attack are indistinguishable. We then show that $\mathsf{GoodSim}$ occurs with high probability.

Claim 1: $\Pr_{real}[\mathcal{A}(r^*)] = \Pr_{sim}[\mathcal{A}(r^*) | \mathsf{GoodSim}]$.

Proof: We show that when $\mathsf{GoodSim}$ occurs $\mathcal{A}$'s view during the simulation is sampled from the same distribution as $\mathcal{A}$'s view during the real attack. By construction, all responses to H queries are as in a real attack. Similarly, when $\mathsf{GoodSim}$ occurs all responses to decryption queries are as in a real attack. Hence, the only thing to show is that the challenge C^* given by the simulator is sampled from the same distribution as in a real attack. Recall that C^* is generated by picking a random $\alpha \in [0, 2^n)$ and computing $C^* = \alpha^2 \bmod N$. For C^* to be an encryption of M_0 or M_1 we must introduce an implicit constraint on H, namely $H(r^*) = w^*$ for some $\langle r^*, w^* \rangle$. We show that w^* is uniform in $\{0,1\}^{m+s_0}$ and that $w^*, H(r^*)$ are both independent of the attacker's view at the end of phase I. Hence, setting $H(r^*) = w^*$ is consistent with a real attack. Proving this requires some care for the Rabin function.

Let $c \in \{1, 2\}$ be a random bit. If C^* has two square roots in $[0, 2^n)$ we use the bit c to pick one of them at random. Let y^* be the chosen square root (unknown to the simulator). By Fact 3 we know that y^* is uniform in $\{0,1\}^n$ (over the probability space induced by $\langle \alpha, c \rangle$). Write $y^* = x^* \| r^*$ with $x^* \in \{0,1\}^{m+s_0}$ and $r^* \in \{0,1\}^{s_1}$. Choose a random $b \in \{0,1\}$ and set $v^* = M_b \| 0^{s_0}$. The random bit b indicates whether C^* is an encryption of M_0 or M_1. Finally, set $H(r^*) = v^* \oplus x^*$. Since y^* is uniform in $[0, 2^n)$ we know that x^* is uniformly distributed in $\{0,1\}^{m+s_0}$. Hence, $v^* \oplus x^* \in \{0,1\}^{m+s_0}$ is a uniform random string. It is independent of $\mathcal{A}$'s view at the end of phase I as required since at that time C^* has not yet been used to answer any queries.

Next, we show that at the end of phase I (just before $\mathcal{A}$ receives the challenge) $H(r^*)$ is independent of $\mathcal{A}$'s view (otherwise we cannot set $H(r^*) = v^* \oplus x^*$). This is immediate by the following facts: (1) we may assume that during phase I the attacker does not issue a query for $H(r^*)$ since otherwise the event $\mathcal{A}(r^*)$ has already occurred and there is nothing more to prove. (2) the second part of $\mathsf{GoodSim}$ implies that during phase I the attacker did not issue a decryption query that restricts $H(r^*)$. Hence, at the end of phase I we know that $H(r^*)$ is independent of the attacker's view. This completes the proof of Claim 1.

Claim 2: $\Pr[\mathsf{GoodSim}] \geq 1 - \frac{2q_D}{2^{s_0}} - \frac{2q_D}{2^{s_1}}$.

Proof: Let C be a decryption query issued by the attacker and rejected by the simulator (i.e. C fails steps 1 and 2 of response to decryption queries). We show that the probability that C is valid is at most $2/2^{s_0}$. Let y_1, y_2 be the square roots of C in $[0, 2^n)$. Let M_1, r_1, x_1, t_1, v_1 and M_2, r_2, x_2, t_2, v_2 be the unpadding of y_1, y_2 as defined in Section 2. Then C is a valid ciphertext only if either $t_1 = 0^{s_0}$ or $t_2 = 0^{s_0}$. Since C failed to satisfy the condition of Step 1 we know that $\mathcal{A}$ has not yet issued a query for $H(r_1)$ or $H(r_2)$. Since C failed to satisfy Step 2 we know that $r_1, r_2 \neq r_1^*$ and $r_1, r_2 \neq r_2^*$. Hence, $H(r_1)$ and $H(r_2)$ are independent of the attacker's current view. Therefore, the probability that $t_1 = 0^{s_0}$ or $t_2 = 0^{s_0}$ is at most $2/2^{s_0}$. Since the attacker makes at most q_D queries, the probability that any of these queries are incorrectly rejected is at most $2q_D/2^{s_0}$.

To bound the probability for the second part of $\mathsf{GoodSim}$ observe that during phase I the challenge C^* is independent of the attacker's view. Therefore, the probability that a decryption query during phase I happened to use r_1^* or r_2^* is at most $2/2^{s_1}$. Therefore, the probability that any of the queries during phase I use r_1^* or r_2^* is at most $2q_D/2^{s_1}$. To conclude we have that $\Pr[\mathsf{GoodSim}] \geq 1 - 2q_D/2^{s_0} - 2q_D/2^{s_1}$ as required. This completes the proof of Claim 2.

The proof of the lemma now follows from Claims 1 and 2:

$$\Pr_{sim}\left[\mathcal{A}(r^*)\right] \geq \Pr_{sim}\left[\mathcal{A}(r^*) \middle| \mathsf{GoodSim}\right] \cdot \Pr\left[\mathsf{GoodSim}\right] =$$

$$\Pr_{real}[\mathcal{A}(r^*)] \cdot \Pr\left[\mathsf{GoodSim}\right] \geq \epsilon\left(1 - \frac{2q_D}{2^{s_0}} - \frac{2q_D}{2^{s_1}}\right)$$

As required. This concludes the proof of Lemma 1 and Theorem 4. $\square$

Extensions. SAEP is not known to be secure for the general RSA trapdoor permutation, $f(x) = x^e \bmod N$. For very small RSA exponents one can show some limited security. For example, for $e = 3$ SAEP has chosen ciphertext security whenever $m + s_0 < n/3$ and $m < n/9$. For typical RSA modulus sizes, these restrictions on the message length make it difficult to use this system.

5 Proof of security for RSA-SAEP$^+$ and Rabin-SAEP$^+$

The proof of security for SAEP$^+$ holds in a more general settings than the proof of SAEP. As in the previous section, we use m, s_0, s_1 as the security parameters of SAEP$^+$ and set $n = m + s_0 + s_1$.

Let $f(x, r)$ be a trapdoor permutation acting on strings in $\{0,1\}^{m+s_0} \times \{0,1\}^{s_1}$. As usual we assume f is selected from a family $\mathcal{F}$ of such trapdoor permutations. Following the notation of [8] we define the set partial one-wayness problem as follows:

Set partial one-wayness: We say that an algorithm $\mathcal{A}$ solves the (f, k) partial one-wayness problem if given $f(x, r)$ the algorithm produces a set $S = \{r_1, \ldots, r_k\} \subseteq \{0,1\}^{s_1}$ such that $r \in S$. More precisely, we say that $\mathcal{A}$ has advantage ϵ if

$$\mathsf{adv}^{p-ow}(\mathcal{A}) = Pr_{x,r}[r \in \mathcal{A}(f(x, r))] \geq \epsilon$$

Consider the $f-$SAEP$^+$ cryptosystem obtained by padding the message M with SAEP$^+$ prior to encrypting with f. We first show that a successful chosen ciphertext attacker on $f-$SAEP$^+$ can be used to solve the set partial one-wayness problem for f. We then discuss the applications to the RSA and Rabin functions.

Theorem 5. *Let $\mathcal{A}$ be a (t, q_D, q_H, q_G) chosen ciphertext attack algorithm in the random oracle model. Suppose $\mathcal{A}$ has advantage ϵ when attacking $f -$ SAEP$^+$. Then there is a uniform algorithm $\mathcal{B}$ for solving the (f, q_H) set partial one-wayness problem with the following parameters:*

$$time(\mathcal{B}) \leq time(\mathcal{A}) + O(q_H + q_G + q_D)$$
$$adv^{p-ow}(\mathcal{B}) \geq adv(\mathcal{A})(1 - q_D/2^{s_0} - q_D/2^{s_1})$$

Proof. Algorithm $\mathcal{B}$ is given $C^* = f(x^*, r^*)$ for some random $x^* \| r^* \in \{0,1\}^n$. Our goal is to output a list of size q_H containing r^*. We construct a simulator that interacts with algorithm $\mathcal{A}$ and produces the required output. Note that since f is a permutation, $x^* \| r^*$ is unique given C^*.

We first give a high level description of the simulator. During the simulation, $\mathcal{A}$ outputs two plaintexts M_0, M_1 where it wishes to be challenged. The simulator responds with C^* as the challenge ciphertext. We view C^* as the encryption of M^*, where M^* is one of the two challenge plaintexts M_0, M_1. We will show that if $\mathcal{A}$ is to have any information about the decryption of C^* it must query the function H at the point r^*. Therefore, if we place all of $\mathcal{A}$'s queries to H in a list, called the H_{list}, then with non-negligible probability the H_{list} is a solution to the set partial one-wayness problem.

Next, we show how to respond to decryption queries. Say the attacker wishes to decrypt the ciphertext C. Suppose C is a valid ciphertext, and is the encryption of some message M. Furthermore, let $C = f(x, r)$. We will show that if C is a valid ciphertext, then both $G(M,r)$ and $H(r)$ are already defined. Hence, the r used to create C must satisfy one of the following: (1) the attacker queried $G(M,r)$ and $H(r)$ prior to issuing the decryption query, or (2) $r = r^*$ and $M = M^*$. Suppose method (1) is used. Then when the decryption query is issued, the simulator has already been queried on $G(M,r)$. Hence, to decrypt C the simulator simply checks to see which pair $\langle M, r\rangle$ on the list of queries to G is the decryption of C. Suppose method (2) is used, i.e. $r = r^*$ and $M = M^*$. In this case $C = C^*$ and hence this is an invalid decryption query since it matches the challenge ciphertext. Consequently, all decryption queries can be correctly answered.

We now give the detailed description of the simulator $\mathcal{B}$.

Setup: The simulator gives $\mathcal{A}$ the security parameters m, s_0, s_1, and identifies the function f within the family of trapdoor permutations $\mathcal{F}$.

H-queries: At any time $\mathcal{A}$ can query H at $r \in \{0,1\}^{s_1}$. The simulator needs to respond with $H(r)$. To respond to such H-queries the simulator maintains a list, called the H_{list}. The H_{list} is a list of tuples of the form $\langle z, H(z)\rangle$ that records all responses to previous H-queries. The H_{list} is initially empty. To respond to the query r the simulator works as follows:

Step 1: If r already appears as the left hand side of some tuple $\langle z, H(z)\rangle$ in the H_{list} then respond to $\mathcal{A}$ with $H(r) = H(z)$.

Step 2: Otherwise, the simulator picks a random $w \in \{0,1\}^{m+s_0}$ and sets $H(r) = w$. It adds the tuple $\langle r, w\rangle$ to the H_{list} and responds to $\mathcal{A}$ by saying $H(r) = w$.

G-queries: At any time $\mathcal{A}$ can query G at $G(M_0, r_0)$ where $M_0 \in \{0,1\}^m$ and $r_0 \in \{0,1\}^{s_1}$. The simulator needs to produce $G(M_0, r_0)$. To respond to such G-queries the simulator maintains a list, called the G_{list}. It is a list of tuples of the form $\langle M, r, G(M,r), C\rangle$ that records all responses to previous G-queries. The last entry, C, is the ciphertext that results from encrypting M using the random string r (see Step 2 below). The G_{list} is initially empty. To respond to the query (M_0, r_0) the simulator works as follows:

Step 1: If (M_0, r_0) appears as the left hand side of some tuple $\langle M_0, r_0, u, C\rangle$ in the G_{list} then respond to $\mathcal{A}$ with $G(M_0, r_0) = u$.

Step 2: Otherwise, the simulator picks a random $u \in \{0,1\}^{s_0}$ and sets $G(M_0, r_0) = u$. It then runs the algorithm for responding to an H query to obtain the value of $H(r_0)$. The simulator then computes $C_0 = f(\mathsf{SAEP}^+(M_0, r_0))$, which is the ciphertext obtained from encrypting M_0 using r_0. Note that at this point $H(r_0)$ and $G(M_0, r_0)$ are well defined, so that C_0 is well defined. The simulator adds $\langle M_0, r_0, u, C_0\rangle$ to the G_{list} and responds to $\mathcal{A}$ by saying $G(M_0, r_0) = u$.

Challenge: At some point $\mathcal{A}$ produces two plaintexts $M_0, M_1 \in \{0,1\}^m$ where it wishes to be challenged. The simulator responds with C^* as the challenge ciphertext.

Decryption queries: Let $C \in \mathbb{Z}_N$ be a ciphertext output by $\mathcal{A}$. The simulator must decrypt C or reject it as an invalid ciphertext. We construct a plaintext extractor to decrypt C. The plaintext extractor is very simple: search the G_{list} to see if it contains a tuple $\langle M, r, u, C\rangle$ with C as the last entry. If so, respond with M as the decryption of C. Otherwise, reject the ciphertext as an invalid ciphertext.

This completes the description of the simulator. Algorithm $\mathcal{B}$ outputs the H_{list} at the end of the simulation as its solution to the given set partial one-wayness problem. One can easily verify that the running time of $\mathcal{B}$ is as stated in the statement of the theorem. We are assuming that searching the H_{list} and G_{list} takes constant time.

It remains to calculate the probability that r^* is contained in one of the tuples on the final H_{list}. This happens if $\mathcal{A}$ issues a query for $H(r^*)$ or a query for $G(-, r^*)$. We denote the probability of this event by $\Pr_{sim}[r^* \in H_{list}]$. We note that once the attacker queries $H(r^*)$ it can easily distinguish the simulation from a real attack: the simulator defines $H(r^*)$ to be a random string, but then C^* is unlikely to be the encryption of M_0 or M_1. Hence, the attacker may choose to abort the attack. However, at that point r^* is already in the H_{list} as required. The next lemma shows that $\Pr_{sim}[r^* \in H_{list}]$ is sufficiently large.

Lemma 2. *Let $\mathcal{A}$ be a (t, q_D, q_H, q_G) chosen ciphertext attacker for $f - \mathsf{SAEP}^+$ with advantage ϵ.*
Then $\quad \Pr_{sim}[r^* \in H_{list}] \geq \epsilon(1 - q_D/2^{s_0} - q_D/2^{s_1})$.

Proof As in the proof of Lemma 1 we have that in the real attack $\Pr_{real}[r^* \in H_{list}] \geq \epsilon$. It remains to show that with high probability $\mathcal{A}$ cannot distinguish the simulation from the real attack until it issues a query for $H(r^*)$ or $G(-, r^*)$. Let $\mathsf{GoodSim}$ be the event defined as in the proof of Lemma 1, namely we say that the event $\mathsf{GoodSim}$ occurred if the following two events happen:

- The simulator never rejects a valid decryption query issued by $\mathcal{A}$ (the validity of a query is determined relative to the oracle H at the end of the simulation), and
- During phase I of the attack (i.e. prior to being given the challenge) algorithm $\mathcal{A}$ did not issue a decryption query for C where $C = f(x, r^*)$ for some $x \in \{0,1\}^{m+s_0}$.

Claim 1: $\quad \Pr_{real}[r^* \in H_{list}] = \Pr_{sim}[r^* \in H_{list} | \mathsf{GoodSim}]$.

Proof: We show that when $\mathsf{GoodSim}$ occurs $\mathcal{A}$'s view during the simulation is sampled from the same distribution as $\mathcal{A}$'s view during the real attack. Observe that the simulator provides a perfect simulation of the H and G oracles. Also, when $\mathsf{GoodSim}$ occurs all decryption queries are answered correctly. Next we show that the challenge ciphertext C^* given to $\mathcal{A}$ is distributed as in the real attack. Recall that x^*, r^* are chosen at random. Let M_0, M_1 be the messages on which $\mathcal{A}$ wishes to be challenged. Pick a random $b \in \{0,1\}$. We make C^* be the encryption of M_b. To do so, pick a random $t^* \in \{0,1\}^{s_0}$ and define $G(M_b, r^*) = t^*$. Set $v^* = M_b \| t^*$ and define $H(r^*) = v^* \oplus x^*$. Then C^* is the encryption of M_b. Furthermore, t^* and $v^* \oplus x^*$ are random strings independent of $\mathcal{A}$'s view at the end of phase I as required. To complete the proof we need to argue that at the end of phase I the hash values $G(M_b, r^*)$ and $H(r^*)$ are

independent of the attacker's view (otherwise we cannot set $G(M_b, r^*) = t^*$ and $H(r^*) = v^* \oplus x^*$). We do so in the same way as at the end of Claim 1 of Lemma 1.
Claim 2: $\Pr[\mathsf{GoodSim}] \geq 1 - \frac{q_D}{2^{s_0}} - \frac{q_D}{2^{s_1}}$.
Proof: Let C be a decryption query issued by the attacker and rejected by the simulator. Let $C = f(x, r)$, and let M, t, v be the unpadding of $x \| r$ as described in Section 2. Then C is a valid ciphertext only if $t = G(M, r)$. Since C is rejected by the simulator we know that the attacker did not issue a query for $G(M, r)$. Similarly, since $C \neq C^*$ we know that $\langle M, r \rangle$ is not equal to $\langle M_b, r^* \rangle$. Hence, $G(M, r)$ is independent of the attacker's current view. Therefore, the probability that $t = G(M, r)$ is $1/2^{s_0}$. Since the attacker makes at most q_D queries, the probability that any decryption query is incorrectly rejected is at most $q_D/2^{s_0}$. We bound the probability for the second part of GoodSim as we did in the proof of Claim 2 of Lemma 1. Overall, we get that $\Pr[\mathsf{GoodSim}] \geq 1 - q_D/2^{s_0} - q_D/2^{s_1}$ as required. This concludes the proof of Claim 2.
The proof of the lemma now follows from Claims 1 and 2 as in the calculation at the end of Lemma 1. This concludes the proof of Theorem 5. □

We now describe how Theorem 5 applies to the Rabin and RSA functions. For the Rabin function we obtain an extremely efficient reduction to factoring. For the RSA permutation we obtain a reduction to breaking RSA, but the reduction is not as efficient. Since the Rabin function is not a permutation on $\mathbb{Z}_N^*$ one needs to extend the proof of Theorem 5 to this case. The extension is done using the same techniques as in Theorem 4. Theorem 5 remains unchanged.

Corollary 1 (Rabin-SAEP$^+$). *Consider the Rabin-SAEP$^+$ scheme, with $m + s_0 < n/2$. Suppose the (n, t, ϵ) factoring assumption holds. Then Rabin-SAEP$^+$ is $(t', \epsilon', q_D, q_H, q_G)$ chosen ciphertext secure in the random oracle model for t', ϵ' satisfying:*

$$t' \leq t - O(q_D + q_G + q_H T_C), \quad \textit{and}$$
$$\tfrac{1}{6}\epsilon' \geq \epsilon + q_D/2^{s_0} + q_D/2^{s_1}$$

where $T_C = T_C(n, 2)$.

Proof Suppose $\mathcal{A}$ is a (t', q_D, q_H, q_G) chosen ciphertext attacker on Rabin-SAEP$^+$ with advantage ϵ'. Let f_N be the function $f_N(x) = x^2 \bmod N$ for some N generated by the Rabin-SAEP$^+$ key-gen algorithm. By Theorem 5 there exists a t_0-time algorithm $\mathcal{B}$ that solves the (f_N, q_H) set partial one-wayness problem with advantage ϵ_0 for some t_0, ϵ_0.

We construct an algorithm $\mathcal{C}$ for factoring N. The algorithm starts by picking a random $\alpha \in [0, 2^n)$ and computing $C^* = \alpha^2 \bmod N$. It then runs $\mathcal{B}$ on input C^*. With probability at least ϵ_0 we obtain a set $S = \{r_1, \ldots, r_{q_H}\} \subseteq \{0,1\}^{s_1}$ of size q_H with the following property: there exists an integer $x \in [0, 2^{m+s_0})$ and $r \in S$ such that $(2^{s_1}x + r)^2 = C^* \bmod N$. Since $x < \sqrt{N}$ we can then find x, r by running Coppersmith's algorithm on all q_H candidates for r. Once x, r are found, we obtain a square root $\alpha' \in [0, 2^n)$ of $C^* \bmod N$. Then the factorization of N is revealed with probability at least $1/6$ by computing $\gcd(N, \alpha - \alpha')$. To see this observe that by Fact 2, C^* mod N has two square roots in $[0, 2^n)$ with probability at least $1/3$. Therefore, $\alpha \neq \alpha'$ with probability $1/6$. Since $0 \leq \alpha, \alpha' < N/2$ the GCD gives a non-trivial factor of N. The resulting factoring algorithm $\mathcal{C}$ has running time: $\text{time}(\mathcal{C}) = t_0 + q_H T_C = t' + O(q_D + q_G + q_H T_C)$ and success probability at least $\text{adv}(\mathcal{C}) = \frac{1}{6}\epsilon_0 = \frac{1}{6}\epsilon'(1 - q_D/2^{s_0} - q_D/2^{s_1})$. The corollary now follows. □

Corollary 2 (RSA-SAEP$^+$). *Consider the RSA-SAEP$^+$ scheme, with $m + s_0 < n/2$. Suppose the (n, e, t, ϵ) RSA assumption holds for some $e > 0$. Then RSA-SAEP$^+$ is $(t', \epsilon', q_D, q_H, q_G)$ chosen ciphertext secure in the random oracle model for t', ϵ' satisfying:*

$$t' \leq t/2 - O(q_D + q_G + q_H^2), \quad \textit{and}$$
$$\epsilon' \geq \epsilon^{1/2} + q_D/2^{s_0} + q_D/2^{s_1}$$

Proof Suppose $\mathcal{A}$ is a (t', q_D, q_H, q_G) chosen ciphertext attacker with advantage ϵ'. Let f_N be the function $f_N(x) = x^e$ mod N for some N generated by the RSA-SAEP$^+$ key-gen algorithm. By Theorem 5 there exists a t_0-time algorithm $\mathcal{B}$ that solves the (f_N, q_H) set partial one-wayness problem with advantage ϵ_0 for some t_0, ϵ_0. Fujisaki et al. [8] show that, when $m + s_0 < n/2$, such an algorithm can be used to compute the e'th root of C^* modulo N. They do so by running algorithm $\mathcal{B}$ on both C^* and αC^* for a random $\alpha \in \mathbb{Z}_N$. The resulting sets S and S_α expose the e'th root of C^* in time $O(q_H^2)$. Hence, we obtain an algorithm for breaking RSA in time $2t_0 = 2t' + O(q_D + q_G + q_H^2)$ and success probability $\epsilon_0^2 = (\epsilon'(1 - q_D/2^{s_0} - q_D/2^{s_1}))^2 \geq (\epsilon' - q_D/2^{s_0} - q_D/2^{s_1})^2$. The corollary now follows. □

Note that the reduction time for RSA-SAEP$^+$ is quadratic in q_H and the success probability is quadratic in ϵ. This is not as efficient as the reduction for Rabin-SAEP$^+$ which is linear time.

Accommodating large messages in RSA-SAEP$^+$. Note that in Corollary 2 the message length must satisfy $m + s_0 < n/2$. We briefly note that the corollary remains true even if $m + s_0 < (1 - \delta)n$ for any fixed $\delta > 0$. To do so run algorithm $\mathcal{B}$ on $c = 1/\delta$ random values $\alpha_1 C^*, \ldots, \alpha_c C^*$. We obtain c lists of size q_H each. Suppose we find a c-tuple $c^* = \langle r_1^*, \ldots r_c^* \rangle$ (one entry from each list) that is the correct solution to these c partial one-wayness problems. Then we obtain the δn least significant bits of each $\alpha_i C^*$ mod N where the α_i are random in $\mathbb{Z}_N$. Finding C^* from this tuple is a standard Hidden Number Problem (HNP) modulo N. We can use the algorithm in [4] to efficiently find C^*. The analysis in [4], which applies to HNP modulo primes, extends to handle RSA composites $N = pq$ as well. The resulting algorithm for breaking RSA has a running time of $O(q_H^c)$, since we must try all c-tuples c^*, and a success probability of $O(\epsilon^c)$, since $\mathcal{B}$ must succeed on all c iterations. Consequently, this reduction becomes very inefficient for small δ.

6 Conclusions

We showed that OAEP can be simplified significantly when applied to the Rabin and RSA functions. OAEP can be viewed as two rounds of a Feistel network. The simplified schemes, SAEP and SAEP$^+$, require only one round of Feistel. The proof of security for the two schemes is based on the algebraic properties of the Rabin and RSA functions. When using an n-bit modulus Rabin-SAEP is secure whenever $m + s_0 < n/2$ and $m < n/4$. SAEP$^+$ is secure whenever $m + s_0 < n/2$. The proof of security for RSA-SAEP$^+$ has the same efficiency as the proof for RSA-OAEP [8]. For Rabin-SAEP$^+$ the proof is as efficient as the proof for Rabin-OAEP+ [11].

The padding SAEP$^+$ is superior to SAEP both in terms of the reduction efficiency and in terms of the weaker restriction on the message length. For practical purposes one is most likely to use SAEP$^+$ rather than SAEP. Nevertheless, it is useful to know that Rabin-SAEP, which is a slightly simpler construction, also provides chosen ciphertext security when appropriate parameters are used.

Acknowledgments

The author thanks David Pointcheval, Jacques Stern, Victor Shoup, and Phong Nguyen for helpful discussions.

References

1. M. Bellare, P. Rogaway, “Random oracles are practical: a paradigm for designing efficient protocols”, In ACM conference on Computers and Communication Security, pp. 62–73, 1993.
2. M. Bellare, P. Rogaway, “Optimal asymmetric encryption”, Eurocrypt ’94, pp. 92–111, 1994.
3. M. Bellare, A. Desai, D. Pointcheval, P. Rogaway, “Relations among notions of security for public-key encryption schemes”, in proc. Crypto ’98, pp. 26–45, 1998.
4. D. Boneh, R. Venkatesan, “Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes”, in proc. Crypto ’96, pp. 129–142, 1996.
5. R. Canetti, O. Goldreich, S. Halevi, “The random oracle model, revisited”, in proc. STOC ’98.
6. D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, vol. 10, pp. 233–260, 1997.
7. D. Dolev, C. Dwork, M. Naor, “Non-malleable cryptography”, SIAM J. of Computing, Vol. 30(2), pp. 391–437, 2000.
8. E. Fujisaki, T. Okamoto, D. Pointcheval, J. Stern, “RSA-OAEP is secure under the RSA assumption”, In proc. Crypto ’2001, Springer-Verlag, 2001.
9. A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
10. J. Manger, “A chosen ciphertext attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as standardized in PKCS #1”, In proc. Crypto ’2001.
11. V. Shoup, “OAEP reconsidered”, In proc. Crypto ’2001, Springer-Verlag, 2001.
12. C. Rackoff, D. Simon, “Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack”, in proc. Crypto ’91, pp. 433–444, 1991.