ISSN 1342-2812

Research Reports on Mathematical and Computing Sciences

A Criterion and Schemes on the Random Oracle Model

Manabu Suzuki and Keisuke Tanaka

December 2004, C–205

Department of
Mathematical and
Computing Sciences
Tokyo Institute of Technology

SERIES C: Computer Science

A Criterion and Schemes on the Random Oracle Model

Manabu Suzuki Keisuke Tanaka *

Dept. of Mathematical and Computing Sciences
Tokyo Institute of Technology
W8-55, 2-12-1 Ookayama, Meguro-ku, Tokyo 152-8552, Japan
{suzuki1, keisuke}@is.titech.ac.jp

December 24, 2004

Abstract

A study of the random oracle model seems to be concentrated to showing the gap between the schemes in the random oracle model and the schemes whose random oracles are replaced with functions chosen at random from some function ensembles. We consider a different direction on the study of the schemes in the random oracle model. We focus on the size of the tables necessary to describe all of the entries to be potentially queried in the random oracle model. We show how to reduce the table sizes of the schemes for encryption and signature in the random oracle model. In particular, we apply this idea to PSS-R and OAEP and show the security of our schemes.

Keywords: Random oracle, PSS-R, OAEP

1 Introduction

The random oracle model is an idealization of cryptographic hash functions, which assumes that all parties including the adversary have oracle access to truly random functions. Bellare and Rogaway [1] introduced the random oracle methodology. They argued that the random oracle model provides a bridge between theory and practice. It consists of two steps. First, design a secure scheme in the random oracle model. Then, replace the random oracles with functions chosen at random from some function ensemble and provide all parties including the adversary with a succinct description of the function. This gives an implementation of the idealized scheme in the real world.

Canetti, Goldreich, and Halevi [4] showed that there exist signature and encryption schemes that are secure in the random oracle model, but for which any implementation of the random oracle results in insecure schemes. Each of their schemes has an adversary that when given as input the description of an implementation of the oracle breaks the scheme that uses this implementation.

Pointcheval and Stern [8] proved that for every 3-round public-coin identification protocol, the signature scheme obtained by applying the Fiat-Shamir transformation is secure in the random oracle model. However, Goldwasser and Taumann [6] proved that the Fiat-Shamir paradigm for designing signature schemes does not always lead to those of security. In particular, they demonstrated the existence of a secure 3-round public-coin identification scheme for which the corresponding signature scheme obtained by applying the Fiat-Shamir paradigm is not secure with respect to any function ensemble implementing the public function.

*Supported in part by NTT Information Sharing Platform Laboratories and Grant-in-Aid for Scientific Research, Ministry of Education, Culture, Sports, Science, and Technology, 14780190, 16092206.

Our contribution

A study of the random oracle model seems to be concentrated to showing the gap between the schemes in the random oracle model and the schemes whose random oracles are replaced with functions chosen at random from some function ensembles. We consider a different direction on the study of the schemes in the random oracle model. We focus on the size of the tables necessary to describe all of the entries to be potentially queried in the random oracle model. We show how to reduce the table sizes of the schemes for encryption and signature in the random oracle model.

Let us consider the function f whose input size is n and output size is m. In order to describe the function for a random oracle, we have to prepare a random table whose size is $2^n \times m$.

Our idea to reduce the table size of a random oracle is to replace one random oracle f with new two random oracles f_1, f_2 such as $f_1 : \{0,1\}^n \to \{0,1\}^{(\log m)^c}$ and $f_2 : \{0,1\}^{(\log m)^c} \to \{0,1\}^m$. This reduces the table size from $2^n \times m$ to $2^n \times (\log m)^c + 2^{(\log m)^c} \times m$. Note that the security of the scheme is no longer preserved by applying our idea.

In particular, we apply this idea to PSS-R and OAEP and show the security of our schemes. PSS-R is an RSA-based signature scheme proposed by Bellare and Rogaway [3]. This signature scheme uses two random oracles and provides unforgeability against the adaptive chosen message attack. OAEP was introduced by Bellare and Rogaway [2]. This scheme uses two random oracles. Fujisaki, Okamoto, Pointcheval and Stern [5] proved that OAEP provides IND-CCA2.

Organization

The rest of this paper is organized as follows. In Section 2, we provide some definitions. We also review PSS-R and OAEP. We apply our idea described above to PSS-R in Section 3, and to OAEP in Section 4. We conclude in Section 5.

2 Preliminaries

In this section, we provide some definitions.

2.1 RSA

Definition 1 (The RSA family). *RSA is a family of trapdoor permutations. It is specified by the RSA generator RSA, which on input 1^k, picks a pair of random distinct $(k/2)$ bit primes and multiplies them to produce a modulus N. It also picks at random an encryption exponent $e \in Z^*_{\varphi(N)}$ and computes the corresponding decryption exponent d so that $ed \equiv 1 \bmod N$. The generator returns N, e, d. These specify $f : Z^*_N \to Z^*_N$ and $f^{-1} : Z^*_N \to Z^*_N$, which are defined by $f(x) = x^e \bmod N$ and $f^{-1}(y) = y^d \bmod N$.*

An inverting algorithm for RSA named I, gets an input N, e, y and tries to find $f^{-1}(y)$. Its success probability is defined by the probability it outputs $f^{-1}(y)$ when N, e, d are obtained by running RSA and y is set to $f(x)$ for an x chosen at random from Z^*_N. The standard definition of security asks that the success probability of any PPT algorithm is a negligible function of k. However, in this paper, we are interested in how much time an inverting algorithm uses and how much probability it achieves.

Definition 2 (Exact security of the RSA family). *An inverting algorithm is said to be a t inverter, where $t : N \to N$, if its running time plus the size of its description is bounded by $t(k)$, in some fixed standard model of computation. We say that I (t, ε)-breaks RSA, where $\varepsilon \in [0,1]$, if I is a t inverter and for each k the success probability of I is at least $\varepsilon(k)$. Finally, we say that RSA is (t, ε)-secure if there is no inverting algorithm which (t, ε)-breaks RSA.*

2.2 Signature schemes with message recovery

In this paper, we discuss PSS-R proposed by Bellare and Rogaway [3], which is the signature scheme with message recovery. We describe the definitions of signature schemes with message recovery and the security of signature schemes.

Definition 3 (Signature schemes with message recovery). *A signature scheme* $\Pi[k_0, k_1]$=(Gen, Sign, Rec) *is specified by a key generation algorithm* Gen*, a signing algorithm* Sign*, and a recovering algorithm* Rec*. The first two should be probabilistic, and all three run in expected polynomial time. Given* 1^k*, where* k *is the security parameter, the key generation algorithm outputs a pair of matching public and secret keys* (pk, sk)*. The signing algorithm takes the message* M *and the secret key* sk *and returns a signature* $x = \mathsf{Sign}_{sk}(M)$ *of* M*. The recovering algorithm takes a candidate signature* x' *and the public key* pk *and returns* $\mathsf{Rec}_{pk}(x') \in \{0,1\}^* \cup \{\text{REJECT}\}$*. The distinguished point* REJECT *is used to indicate that the recipient rejected the signature. A returned value of* $M \in \{0,1\}^*$ *indicates that the verifier accepts the message* M *as an authentic one. We demand that if* x *is produced as* $x \leftarrow \mathsf{Sign}_{sk}(M)$ *then* $\mathsf{Rec}_{pk}(x) = M$*.*

Definition 4 (Security of signature schemes). *A forger takes as input a public key* pk*, where* $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$*, and tries to forge signatures with respect to* pk*. The forger is allowed a chosen message attack in which it can request and obtain signatures of messages of its choice. This is modeled by allowing the forger oracle access to the signing algorithm. The forger is deemed successful if it outputs a valid forgery, namely a message/signature pair* (M, x) *such that* $\mathsf{Rec}_{pk}(x) = M$ *where* M *has never been requested to the signing oracle. The forger is said to be a* (t, q_{sig}, q_{hash})*-forger if its running time plus description size is bounded by* $t(k)$*, and it makes at most* $q_{sig}(k)$ *and* $q_{hash}(k)$ *queries of its signing oracle and hash oracles, respectively. Such a forger* F *is said to* $(t, q_{sig}, q_{hash}, \varepsilon)$*-break the signature scheme if for every* k*, the probability that* F *outputs a valid forgery is at least* $\varepsilon(k)$*. Finally we say that the signature scheme* (Gen, Sign, Rec) *is* $(t, q_{sig}, q_{hash}, \varepsilon)$*-secure if there is no forger who* $(t, q_{sig}, q_{hash}, \varepsilon)$*-breaks the scheme.*

2.3 Encryption schemes

We describe the definitions of public key encryption schemes and indistinguishability of encryption schemes.

Definition 5 (Public key encryption schemes). *A encryption scheme* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *is defined by the three following algorithms. Given* 1^k*, where* k *is the security parameter, the key generation algorithm* $\mathcal{K}$ *outputs a pair of* (pk, sk) *of matching public and secret keys. This algorithm is probabilistic. Given a message* m *and a public key* pk*, the encryption algorithm* $\mathcal{E}$ *produces a ciphertext* c *of* m*. This algorithm may be probabilistic. Given a ciphertext* c *and the secret key* sk*, the decryption algorithm* $\mathcal{D}$ *returns the plaintext* m*. This algorithm is deterministic.*

Definition 6 (Indistinguishability of encryption schemes). *This security notion requires computational impossibility to distinguish between two messages, chosen by the adversary, one of which has been encrypted, with a probability significantly better than one half. Her advantage* $Adv^{ind}(\mathcal{A})$*, where the adversary* $\mathcal{A}$ *is seen as a 2-stage Turing machine* $(\mathcal{A}_1, \mathcal{A}_2)$*, should be negligible, where* $Adv^{ind}(\mathcal{A})$ *is formally defined as.*

$$Adv^{ind}(\mathcal{A}) = 2 \times \Pr_{b,r}[(pk, sk) \leftarrow \mathcal{K}(1^k), (m_0, m_1, s) \leftarrow \mathcal{A}_1(pk), c = \mathcal{E}_{pk}(m_b; r) : \mathcal{A}_2(m_0, m_1, s, c) = b] - 1.$$

2.4 PSS-R

In this section, we describe PSS-R. PSS-R is an RSA-based signature scheme proposed by Bellare and Rogaway [3].

The signature scheme $\Pi[k_0, k_1]$ =(Gen, Sign, Rec) is parameterized by k_0 and k_1, which are numbers between 1 and k satisfying $k_0 + k_1 \leq k - 1$.

The key generation algorithm Gen, on input 1^k, runs $RSA(1^k)$ to obtain (N, e, d) and output (pk, sk), where $pk = (N, e)$ and $sk = (N, d)$.

The signing and recovering algorithms make use of two hash functions. The first H called the compressor maps as $H : \{0,1\}^{k-k_1-1} \to \{0,1\}^{k_1}$ and G called the generator maps as $G : \{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$. For simplicity of explication, we assume that the length of messages to be signed is $n = k - k_0 - k_1 - 1$.

We now describe how to sign and verify. In the description, $[M]_n$ denote the n least significant bits of M, while $[M]^n$ denotes the n most significant bits of M.

Sign(M)

1. $r \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$; $w \leftarrow H(M \parallel r)$
2. $r^* \leftarrow [G(w)]^{k_0} \oplus r$; $M^* \leftarrow [G(w)]_{k-k_0-k_1-1} \oplus M$
3. $y \leftarrow 0 \parallel w \parallel r^* \parallel M^*$
4. $x \leftarrow y^d \bmod N$
5. Return x.

Rec(x)

1. $y \leftarrow x^e \bmod N$
2. Break up y as $b \parallel w \parallel r^* \parallel M^*$.
3. $r \leftarrow r^* \oplus [G(w)]^{k_0}$
4. $M \leftarrow M^* \oplus [G(w)]_{k-k_0-k_1-1}$
5. If $H(M \parallel r) = w$ and $b = 0$, then return M, else return REJECT.

The step $r \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$ indicates that the signer picks at random a seed r of k_0 bits. The reason we add 0 to y is to guarantee that y is in Z_N^*.

In this scheme, the algorithm Sign is probabilistic. Therefore, a given message has many possible signatures, depending on the value of r chosen by the signer.

The following proposition describes the security of PSS-R. The proof of the proposition is in Appendix A.

Proposition 1. *Suppose that RSA is (t', ε')-secure. Then for any q_{sig}, q_{hash}, the above signature scheme $\Pi[k_0, k_1]$ =(Gen, Sign, Rec) is $(t, q_{sig}, q_{hash}, \varepsilon)$-secure, where $t(k) = t'(k) - [q_{sig}(k) + q_{hash}(k) + 1] \cdot k_0 \cdot O(k^3)$ and $\varepsilon(k) = \varepsilon'(k) + [2(q_{sig}(k) + q_{hash}(k))^2](2^{-k_0} + 2^{-k_1}) + 2^{-k_0}$.*

2.5 OAEP

In this section, we describe OAEP. OAEP was introduced by Bellare and Rogaway [2]. Fujisaki, Okamoto, Pointcheval and Stern [5] proved that OAEP provides IND-CCA2 under the partial-domain one-wayness of the underlying trapdoor permutation. They also uses two random oracles in the scheme.

We define the partial-domain one-wayness and the set partial-domain one-wayness of permutation f.

Definition 7 ((τ, ε)-partial-domain one-wayness of f). *We say a function f is (τ, ε)-partial-domain one-wayness if for any adversary $\mathcal{A}$ whose running time is bounded by τ, the success probability $Succ^{pd\text{-}ow}(\mathcal{A})$ is upper bounded by ε, where*

$$Succ^{pd\text{-}ow}(\mathcal{A}) = \Pr_{s,t}[\mathcal{A}(f(s,t)) = s].$$

Definition 8 ((τ,ε)-set partial-domain one-wayness of f). *We say a function f is (τ,ε)-set partial-domain one-wayness if for any adversary $\mathcal{A}$ that outputs a set of ℓ elements within time bound τ the success probability $Succ^{s\text{-}pd\text{-}ow}(\mathcal{A}) = \Pr_{s,t}[\mathcal{A}(f(s,t)) = s]$ is upper bounded by ε, where*

$$Succ^{s\text{-}pd\text{-}ow}(\mathcal{A}) = \Pr_{s,t}[\mathcal{A}(f(s,t)) = s].$$

We briefly describe OAEP cryptosystem $(\mathcal{K},\mathcal{E},\mathcal{D})$ obtained from any trapdoor permutation generator $f : \{0,1\}^k \to \{0,1\}^k$ whose inverse is denoted by g.

We need two hash functions G and H. We assume both G and H are the random oracles such as $G : \{0,1\}^{k_0} \to \{0,1\}^{n+k_1}$ and $H : \{0,1\}^{n+k_1} \to \{0,1\}^{k_0}$, respectively.

We now describe how to generate keys. We also describe how to encrypt and decrypt. The size of the plaintext m is n. In the description below, $[M]_n$ denote the n least significant bits of M, while $[M]^n$ denotes the n most significant bits of M.

$\mathcal{K}(1^k)$

It specifies an instance of the function f, and of its inverse g. The public key pk is therefore f and the secret key sk is g.

$\mathcal{E}^{G,H}(pk, m; r)$

1. $s \leftarrow (m \parallel 0^{k_1}) \oplus G(r)$
2. $t \leftarrow r \oplus H(s)$
3. $w \leftarrow s \parallel t$
4. $c \leftarrow f(w)$
5. Return c.

$\mathcal{D}^{G,H}(sk, c)$

1. $w \leftarrow g(c)$
2. $s \leftarrow [w]^{n+k_1}$, $t \leftarrow [w]_{k_0}$
3. $r \leftarrow t \oplus H(s)$
4. $m \leftarrow [s \oplus G(r)]^n$, $z \leftarrow [s \oplus G(r)]_{k_1}$
5. If $z = 0^{k_1}$, then return m else return REJECT.

Fujisaki, Okamoto, Pointcheval and Stern [5] proved that OAEP provides IND-CCA2 under the partial-domain one-wayness of the underlying trapdoor permutation. They also uses two random oracles in the scheme. More precisely, the following security result holds. The proof of the theorem is in Appendix B.

Proposition 2. *Let $\mathcal{A}$ be a CCA2-adversary against the semantic security of OAEP conversion $(\mathcal{K},\mathcal{E},\mathcal{D})$, with advantage ε and running time t, making q_D, q_G and q_H queries to the decryption oracle, and the hash functions G and H, respectively. Then, there exists an algorithm $\mathcal{B}$ such that $Succ^{pd-ow}(\mathcal{B})$ is greater than*

$$\frac{1}{q_H}(\varepsilon - \frac{10q_Dq_G + 5q_D + q_G}{2^{k_0}} - \frac{10q_D}{2^{k_1}}),$$

and whose running time $t' \leq t + q_Gq_H(T_f + O(1))$ where T_f denotes the time complexity of function f.

3 Our variant of PSS-R

In this section, we apply our idea to PSS-R. We describe our variant of PSS-R and prove the security. Then we compare our variant and PSS-R with respect to the security and the table size of the random oracle.

3.1 The scheme

We describe our variant of PSS-R. We describe $\Pi[k_0, k_1] =$(Gen, Sign, Rec).

The key generation algorithm Gen, on input 1^k, runs $RSA(1^k)$ to obtain (N, e, d) and output (pk, sk), where $pk = (N, e)$ and $sk = (N, d)$.

We use two random oracles H^1 and H^2 instead of H in PSS-R. We also use two random oracles G^1 and G^2 instead of G in PSS-R. Let $H^1 : \{0,1\}^{k-k_1-1} \to \{0,1\}^{c_1}$, $H^2 : \{0,1\}^{c_1} \to \{0,1\}^{k_1}$, $G^1 : \{0,1\}^{k_1} \to \{0,1\}^{c_2}$, and $G^2 : \{0,1\}^{c_2} \to \{0,1\}^{k-k_1-1}$. In the above description, let $c_1 = O((\log k_1)^c)$ and $c_2 = O((\log(k - k_1))^c)$, where c is a constant number.

We now describe how to sign and recover. We assume that the length of the message to be signed is $n = k - k_0 - k_1 - 1$.

Sign(M)

1. $r \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$; $w^1 \leftarrow H^1(M \parallel r)$; $w^2 \leftarrow H^2(w^1)$; $w^3 \leftarrow G^1(w^2)$
2. $r^* \leftarrow [G^2(w^3)]^{k_0} \oplus r$; $M^* \leftarrow [G^2(w^3)]_{k-k_1-k_0-1} \oplus M$
3. $y \leftarrow 0 \parallel w^2 \parallel r^* \parallel M^*$
4. $x \leftarrow y^d \bmod N$
5. Return x.

Rec(x)

1. $y \leftarrow x^e \bmod N$
2. Break up y as $b \parallel w^2 \parallel r^* \parallel M^*$.
3. $w^3 \leftarrow G^1(w^2)$; $r \leftarrow r^* \oplus [G^2(w^3)]^{k_0}$
4. $M \leftarrow M^* \oplus [G^2(w^3)]_{k-k_1-k_0-1}$
5. If $H^2(H^1(M \parallel r)) = w^2$ and $b = 0$, then return M else return REJECT.

3.2 Security

We show the security of our variant of PSS-R.

Theorem 1. *Suppose that RSA is (t', ε')-secure. Then for any q_{sig}, q_{hash} the signature scheme $\Pi[k_0, k_1] =$(Gen, Sign, Rec) is $(t, q_{sig}, q_{hash}, \varepsilon)$-secure, where $t(k) = t'(k) - [q_{sig}(k) + q_{hash}(k) + 1] \cdot k_0 \cdot O(k^3)$, $\varepsilon(k) = \varepsilon'(k) + [2(q_{sig}(k) + q_{hash}(k))^2](2^{-k_0} + 2^{-c_1} + 2^{-k_1} + 2^{-c_2}) + 2^{-k_0}$.*

Proof. Let F be a forger which $(t, q_{sig}, q_{hash}, \varepsilon)$-breaks our variant. We present an inverter I which (t', ε')-breaks the trapdoor permutation family RSA.

The simulation

The input to I is N, e and η, where η is chosen at random from Z_N^*, and N, e, d are chosen by running the generator RSA. We let $f : Z_N^* \to Z_N^*$ be $f(x) = x^e \bmod N$. The inverter I wants to compute $f^{-1}(\eta) = \eta^d \bmod N$. It forms the public key N, e, and starts running F on input this key. The forger F will make oracle queries. The inverter I must answer itself. We let $Q_1, \ldots, Q_{q_{sig}+q_{hash}}$

denote the sequence of oracle queries that F makes. In the process of answering these queries, the inverter I will build or define the function H^1, H^2, G^1, G^2. The inverter I maintain a counter i, initially 0, which is incremented for each query and prepare empty lists named List G^1, List G^2, List H^1 and List H^2.

Answering signing queries

1. Increment i and let $M_i = Q_i$.
2. Pick $r \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$.
3. If $r_j = r_i$ for some $j \leq i$, then abort.
4. Pick $w_i^1 \stackrel{R}{\leftarrow} \{0,1\}^{c_1}$.
5. If $w_j^1 = w_i^1$ for some $j \leq i$, then abort.
6. Set $H^1(M_i \parallel r_i) = w_i^1$ into the List H^1.
7. Repeat $x_i \stackrel{R}{\leftarrow} Z_N^*$; $y_i \leftarrow f(x_i)$ until the first bit of y_i is 0.
8. Break up y_i to write it as $0 \parallel w_i^2 \parallel {r_i}^* \parallel {M_i}^*$.
9. Set $H^2(w_i^1) = w_i^2$ into the List H^2.
10. If $w_j^2 = w_i^2$ for some $j \leq i$, then abort.
11. Pick $w_i^3 \stackrel{R}{\leftarrow} \{0,1\}^{c_2}$.
12. If $w_j^3 = w_i^3$ for some $j \leq i$, then abort.
13. Set $G^1(w_i^2) = w_i^3$ into the List G_1.
14. Set $G^2(w_i^3) = {r_i}^* \oplus r_i \parallel M_i \oplus {M_i}^*$ into the List G^2.
15. Return x_i to F as the answer to signing query $Q_i = M_i$.

Answering H^1 oracle queries

1. Increment i and break up Q_i as $M_i \parallel r_i$.
2. Say Q_i is old if $M_j \parallel r_j = M_i \parallel r_i$ for some $j \leq i$ and new otherwise. Now if Q_i is old then set $(w_i^1, {r_i}^*, {M_i}^*) = (w_j^1, {r_j}^*, {M_j}^*)$ and return w_j^1. Else go to next step.
3. Repeat $x_i \stackrel{R}{\leftarrow} Z_N^*$; $z_i \leftarrow f(x_i)$; $y_i \leftarrow \eta z_i \bmod N$ until the first bit of y_i is 0.
4. Break up y_i to write it as $0 \parallel w_i^2 \parallel {r_i}^* \parallel {M_i}^*$.
5. Pick $w_i^1 \stackrel{R}{\leftarrow} \{0,1\}^{c_1}$.
6. If $w_j^1 = w_i^1$ for some $j \leq i$, then abort.
7. Set $H^1(M_i \parallel r_i) = w_i^1$ into the List H^1.
8. Set $H^2(w_i^1) = w_i^2$ into the List H^1.
9. Pick $w_i^3 \stackrel{R}{\leftarrow} \{0,1\}^{c_2}$.
10. If $w_j^1 = w_i^1$ for some $j \leq i$, then abort.
11. Set $G^1(w_i^2) = w_i^3$ into the List G^1.
12. Set $G^2(w_i^3) = {r_i}^* \oplus r_i \parallel M_i \oplus {M_i}^*$ into the List G^2.
13. Return w_i^1 to F as the answer to the H^1 oracle query $Q_i = M_i \parallel r_i$.

Answering H^2 oracle queries

1. Increment i and let $w_i^1 = Q_i$.

2. If $w_i^1 = w_j^1$ for some $j \leq i$,then return $w_i^2 = H^2(w_j^1)$. Else pick a string $\alpha \stackrel{R}{\leftarrow} \{0,1\}^{k_1}$, set $w_i^2 = H^2(w_i^1)$ into the List H^2 and return α to F as the answer to the H^2 oracle query Q_i.

Answering G^1 oracle queries

1. Increment i and let $w_i^2 = Q_i$.
2. If $w_i^2 = w_j^2$ for some $j \leq i$, then return $w_i^3 = G^1(w_j^2)$. Else pick a string $\alpha \stackrel{R}{\leftarrow} \{0,1\}^{c^2}$, set $w_i^3 = G^1(w_i^2)$ into the List G^1 and return α to F as the answer to the G^1 oracle query Q_i.

Answering G^2 oracle queries

1. Increment i and let $w_i^3 = Q_i$.
2. If $w_i^3 = w_j^3$ for some $j \leq i$, then return $G^1(w_j^3)$. Else pick a string $\alpha \stackrel{R}{\leftarrow} \{0,1\}^{k-k_1-1}$, set $G^1(w_i^3)$ into the List G^2 and return α to F as the answer to the G oracle query Q_i.

Analysis

We first show that the validity of the simulation. We insist that the forger must use the random oracles G_1, G_2, H_1, H_2 in order to output the signature.

We define some events to show the validity of the simulation. We denote (M^+, x^+) as a correct pair of the message and the signature.

E_1 : Forger outputs (M^+, x^+) without asking corresponding $M^+ \parallel r^+$ to the H_1 oracle queries, but asking corresponding w^{1+}, w^{2+}, w^{3+} to the H^2, G^1, G^2 oracle queries.

E_2 : Forger outputs (M^+, x^+) without asking corresponding w^{1+} to the H^2 oracle queries but asking corresponding $M^+ \parallel r^+, w^{2+}, w^{3+}$ to the H^1, G^1, G^2 oracle queries.

E_3 : Forger outputs (M^+, x^+) without asking corresponding w^{2+} to the G^1 oracle queries but asking corresponding $M^+ \parallel r^+, w^{1+}, w^{3+}$ to the H^1, H^2, G^2 oracle queries.

E_4 : Forger outputs (M^+, x^+) without asking corresponding w^{3+} to the G^2 oracle queries but asking corresponding $M^+ \parallel r^+, w^{1+}, w^{2+}$ to the H^1, H^2, G^1 oracle queries.

E_5 : Forger uses the random oracle H^1, H^2, G^1, G^2 and the signing oracle in order to output (M^+, x^+) asking any oracle.

Our goal is to estimate the probability that E_5 occurs. It is not so hard to figure out $\Pr[E_1]$, $\Pr[E_2]$, $\Pr[E_3]$ and $\Pr[E_4]$. Since $H^1(M \parallel r)$ is unpredictable, the probability that E_1 occurs is at most 2^{-c_1}. Similarly we can figure out that $\Pr[E_2] \leq 2^{-k_1}$, $\Pr[E_3] \leq 2^{-c_2}$, and $\Pr[E_4] \leq 2^{-(k-k_1-1)}$.

We can estimate that $\Pr[E_5] \geq 1 - (\Pr[E_1] + \Pr[E_2] + \Pr[E_3] + \Pr[E_4])$.

It shows that the forger must use the random oracle to forge a signature.

We now prove the theorem. We define the following events to analyze the security probability.

E_6 : The inverter I never abort in step 3, 5, 10, and 12 in answering signing queries and step 6 and 10 in answering H^1 oracle queries.

E_7 : The inverter I cannot stop in step 7 in answering signing queries and step 3 in answering H^1 oracle queries within $1 + k_0$ repetition.

	Table size	Security : $(t, q_{sig}, q_{hash}, \varepsilon)$-secure
PSS-R	$2^{k-k_1-1} \times k_1 + 2^{k_1} \times (k-k_1-1)$	$\varepsilon'(k) + [2(q_{sig}(k)+q_{hash}(k))^2](2^{-k_0}+2^{-k_1}) + 2^{-k_0}$ $t'(k) - [q_{sig}(k)+q_{hash}(k)+1] \cdot k_0 \cdot O(k^3)$
Our variant	$2^{k-k_1-1} \times c_1 + 2^{c_1} \times k_1 + 2^{k_1} \times c_2 + 2^{c_2} \times (k-k_1-1)$	$\varepsilon'(k) + [2(q_{sig}(k)+q_{hash}(k))^2](2^{-k_0}+2^{-c_1}+2^{-k_1}+2^{-c_2}) + 2^{-k_0}$ $t'(k) - [q_{sig}(k)+q_{hash}(k)+1] \cdot k_0 \cdot O(k^3)$

Figure 1: The comparison of PSS-R and our variant

We can estimate that $\Pr[E_6] \leq 2(q_{sig}+q_{hash})^2(2^{-k_0}+2^{-c_1}+2^{-k_1}+2^{-c_2})$.

Let us consider E_7. The time for step 7 in answering signing queries and step 3 in answering H^1 oracle queries can not be bounded. However, the expected time is two execution of the loop. Then we make it a rule to stop the loop after $1+k_0$ steps. Then, we can estimate that $\Pr[E_7] \leq 2^{-k_0}$.

Now we can figure out that $\varepsilon(k) = \varepsilon'(k) + [2(q_{sig}(k)+q_{hash}(k))^2](2^{-k_0}+2^{-c_1}+2^{-k_1}+2^{-c_2}) + 2^{-k_0}$.

The running time of I is that of F plus the time to choose the y_i's. The main thing here is one RSA computation for each y_i, which is cubic time (or better). We can say that $t(k) = t'(k) - [q_{sig}(k)+q_{hash}(k)+1] \cdot k_0 \cdot O(k^3)$.

□

3.3 Comparison

We now compare our variant and PSS-R with respect to the table size of the random oracle.

In PSS-R, the signing and verifying algorithms make use of two random oracles H and G. $H : \{0,1\}^{k-k_1-1} \to \{0,1\}^{k_1}$ and $G : \{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$. We need $2^{k-k_1-1} \times k_1$ table size to realize the random oracle H, and $2^{k_1} \times (k-k_1-1)$ to random oracle G. Therefore, we need $2^{k-k_1-1} \times k_1 + 2^{k_1} \times (k-k_1-1)$ table size in order to realize PSS-R.

In our variant, we use two random oracles H^1 and H^2 instead of H in PSS-R. We also use two random oracles G^1 and G^2 instead of G in PSS-R. Let $H^1 : \{0,1\}^{k-k_1-1} \to \{0,1\}^{c_1}$, $H^2 : \{0,1\}^{c_1} \to \{0,1\}^{k_1}$, $G^1 : \{0,1\}^{k_1} \to \{0,1\}^{c_2}$, and $G^2 : \{0,1\}^{c_2} \to \{0,1\}^{k-k_1-1}$. In the above description, let $c_1 = O((\log k_1)^c)$ and $c_2 = O((\log(k-k_1))^c)$, where c is a constant number. We need $2^{k-k_1-1} \times c_1$ table size to realize the random oracle H^1, $2^{c_1} \times k_1$ to H^2, $2^{k_1} \times c_2$ to G^1 and $2^{c_2} \times (k-k_1-1)$ to G^2. Thus, we need $2^{k-k_1-1} \times c_1 + 2^{c_1} \times k_1 + 2^{k_1} \times c_2 + 2^{c_2} \times (k-k_1-1)$ table size in order to realize our variant.

We show the table sizes and the security of PSS-R and our variant in Figure 1. We can certainly reduce the table size of PSS-R by applying our idea.

4 Our variant of OAEP

In this section, we apply our idea described to OAEP. We describe our variant of OAEP and prove the security. Then we compare our variant and OAEP with respect to the security and the table size of the random oracle.

4.1 The scheme

We describe our variant of OAEP.

We use two random oracles H^1 and H^2 instead of H in OAEP. We also use two random oracles G^1 and G^2 instead of G in OAEP. Let $G^1 : \{0,1\}^{k_0} \to \{0,1\}^{c_1}$, $G^2 : \{0,1\}^{c_1} \to \{0,1\}^{k-k_0}$, $H^1 : \{0,1\}^{k-k_0} \to \{0,1\}^{c_2}$, and $H^2 : \{0,1\}^{c_2} \to \{0,1\}^{k_0}$.

In the above description, let $c_1 = O((\log(k-k_0))^c)$ and $c_2 = O((\log k_0)^c)$, where c is a constant number.

We now describe how to generate keys. We also describe how to encrypt and decrypt. The size of the plaintext m is n.

$\mathcal{K}(1^k)$

It specifies an instance of the function f, and of its inverse g. The public key pk is therefore f and the secret key sk is g.

$\mathcal{E}^{G^1,G^2,H^1,H^2}(pk, m; r_1)$

1. $r_2 \leftarrow G^1(r_1)$
2. $s_1 \leftarrow (m \parallel 0^{k_1}) \oplus G^2(r_2)$
3. $s_2 \leftarrow H^1(s_1)$
4. $t \leftarrow r_1 \oplus H^2(s_2)$
5. $w \leftarrow s_1 \parallel t$
6. $c \leftarrow f(w)$
7. Return c.

$\mathcal{D}^{G^1,G^2,H^1,H^2}(sk, c)$

1. $w \leftarrow g(c)$
2. $s_1 \leftarrow [w]^{n+k_1}$, $t \leftarrow [w]_{k_0}$
3. $s_2 \leftarrow H^1(s_1)$
4. $r_1 \leftarrow t \oplus H^2(s_2)$
5. $r_2 \leftarrow G^1(r_1)$
6. $m \leftarrow [s_1 \oplus G^2(r_2)]^n, z \leftarrow [s_1 \oplus G^2(r_2)]_{k_1}$
7. If $z = 0^{k_1}$, then return m else return REJECT.

4.2 The security result

In the following, we prove that the scheme is IND-CCA2 in the random oracle model, relative to the partial-domain one-wayness of function f. More precisely, the following exact security result holds.

Theorem 2. *Let $\mathcal{A}$ be a CCA2-adversary against the semantic security of our variant scheme of OAEP conversion $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, with advantage ε and running time t, making q_D, q'_G and q'_H queries to the decryption oracle, and the hash functions G^1, G^2, H^1,and H^2, respectively. Then, there exist an algorithm $\mathcal{B}$ such that $Succ^{pd-ow}(\mathcal{B})$ is greater than*

$$\frac{1}{q_{H'}}(\varepsilon - \frac{10q_Dq'_G + 5q_D + q'_G}{2^{k_0}} - \frac{5q_Dq'_G + q'_G}{2^{c_2}} - \frac{10q_D}{2^{k_1}} - \frac{9q_Dq_G + 7q_D}{2^{c_1}}),$$

and whose running time $t' \leq t + q'_G \cdot q'_H \cdot (T_f + O(1))$, where T_f denotes the time complexity of function f.

In the above description, $q'_G = q^1_G + q^2_G$ and $q'_H = q^1_H + q^2_H$.

In order to prove this thorem relative to the partial-domain one-wayness of the permutation, we prove the following lemma.

Lemma 1. *Let $\mathcal{A}$ be a CCA2-adversary against the semantic security of our variant of OAEP conversion $\mathcal{D}^{G^1,G^2,H^1,H^2}(sk, c)$, with advantage ε and running time t, making q_D, q'_G and q'_H queries to the decryption oracle, and the hash functions G^1, G^2, H^1,and H^2, respectively. Then, there exists an algorithm $\mathcal{B}$ such that $Succ^{s-pd-ow}(\mathcal{B})$ is greater than*

$$\varepsilon - \frac{10q_Dq'_G + 5q_D + q'_G}{2^{k_0}} - \frac{5q_Dq'_G + q'_G}{2^{c_2}} - \frac{10q_D}{2^{k_1}} - \frac{9q_Dq_G + 7q_D}{2^{c_1}},$$

and whose running time $t' \leq t + q_{G'}q_{H'}(T_f + O(1))$, where T_f denotes the time complexity of function f.

In the above description, $q'_G = q^1_G + q^2_G$ and $q'_H = q^1_H + q^2_H$.

4.3 The proof of the theorem

We prove lemma in three stages.

The first we present the reduction of the CCA2-adversary $\mathcal{A}$ attacking indistinguishability to the algorithm $\mathcal{B}$ for breaking the partial-domain one-wayness of f. Note that, in the present proof, we are interested in the security under the partial-domain one-wayness of f.

The second we present the simulation of the algorithm $\mathcal{B}$. We present how $\mathcal{B}$ simulates the random oracles G, H and the decryption oracle.

Finally, we analyze the success probability of our reduction in total. We analyze the decryption oracle simulation employed in this reduction works correctly with overwhelming probability under the partial-domain one-wayness of f and insist the lemma and the theorem.

4.3.1 The top level description of the reduction

In this first part, we recall how the reduction operates. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against the semantic security of $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, under the chosen ciphertext attack. Within time bound t, $\mathcal{A}$ asks q_D, q'_G, q'_H queries to the decryption oracle and the random oracles G^1, G^2, H^1 and H^2, and distinguishes the right plaintext with an advantage greater that ε. Let us describe the reduction of $\mathcal{B}$.

1. Algorithm $\mathcal{B}$ is given a function f which is defined by the public key and $c^* \leftarrow f(s_1^*, t^*)$, for $(s_1^*, t^*) \stackrel{R}{\leftarrow} \{0,1\}^{k-k_0} \times \{0,1\}^{k_0}$. The aim of $\mathcal{B}$ is to recover the partial pre-image s_1^* to c^*.

2. Algorithm $\mathcal{B}$ runs $\mathcal{A}_1$ on the public data, and gets a pair of messages $\{m_0, m_1\}$ as well as states information st. It chooses a random bit b, and then gives c^* to $\mathcal{A}_1$, as the cipher text of m_b. Algorithm $\mathcal{B}$ simulates the answers to the queries of $\mathcal{A}_1$ to the decryption oracle and the random oracles G^1, G^2, H^1 and H^2, respectively. See the description of these simulations below.

3. Algorithm $\mathcal{B}$ runs $\mathcal{A}_2(c^*, st)$ and finally gets answer b'. Algorithm $\mathcal{B}$ simulates the answers to the queries of $\mathcal{A}_2$ to the random oracles G^1, G^2, H^1, H^2 and the decryption oracle, respectively. See the description of these simulations below. Algorithm $\mathcal{B}$ then outputs the partial pre-image s_1^* of c^*, if one has been found among the queries asked to H^1, or the list of queries asked to H^1.

4.3.2 The simulation

In this section, we explain how the algorithm $\mathcal{B}$ simulates the random oracle G^1, G^2, H^1, H^2 and the decryption oracle.

The random oracles

On the simulation of the random oracles, $\mathcal{B}$ has to simulate the random oracle answers, managing query answer lists List H^1, List H^2, List G^1 and List G^2 for the oracles G^1, G^2, H^1, and H^2 respectively, they are initially set to empty lists;

Answering G^1 oracle queries

- If γ_1 is asked before, return corresponding γ_2 that $\mathcal{B}$ answered before.
- For a fresh query γ_1 to G^1, look at the List H^1 and List H^2 and for any query δ_1, δ_2 asked to H^1, H^2 with answer $H^1(\delta_1)$, $H^2(\delta_2)$ one build $z = \gamma_1 \oplus H^2(\delta_2)$, and check whether $c^* = f(\delta_1, z)$.
  - If for some δ_1, δ_2 that relation holds, choose γ_2 at random and we can still correctly simulate G^1, by answering $G^2(\gamma_2) = \delta_1 \oplus (m_b \parallel 0^{k_1})$. Then put (γ_1, γ_2) into the List G^1 and put $(\gamma_2, G^2(\gamma_2))$ into the List G^2.

– If there is no such δ_1, δ_2, choose γ_2 at random and put (γ_1, γ_2) into the List-G^1.

Answering G^2 oracle queries

- If γ_2 is asked before, return corresponding $G^2(\gamma_2)$ that $\mathcal{B}$ answered before.
- For a fresh query γ_2 to G^2, output a random value $G^2(\gamma_2)$ and the pair $(\gamma_2, G^2(\gamma_2))$ is put to the List G^2.

Answering H^1 oracle queries

- If δ_1 is asked before, return corresponding $H^1(\delta_1)$ that $\mathcal{B}$ answered before.
- For a fresh query δ_1 to H^1, output a random value $\delta_2 = H^1(\delta_1)$ and the pair (δ_1, δ_2) is put to the List H^1.

Answering H^2 oracle queries

- If δ_2 is asked before, returns corresponding $H^2(\delta_2)$ that $\mathcal{B}$ answered before.
- For a fresh query δ_2 to H^2, one outputs a random value $H^2(\delta_2)$ and the pair $(\delta_2, H^2(\delta_2))$ is put to the List H^2.

The decryption oracle

On query $c = f(s_1, t)$ to the decryption oracle, decryption oracle simulation DS looks at each query-answer $(\gamma_1, G^1(\gamma_1)) \in$ List G^1, $(\gamma_2, G^2(\gamma_2)) \in$ List G^2, $(\delta_1, H^1(\delta_1)) \in$ List H^1, and $(\delta_2, H^2(\delta_2)) \in$ List H^2.

For each pair taken from these lists, check whether $G^1(\gamma_1) = \gamma_2$, $H^1(\delta_1) = \delta_2$, $c = f(\delta_1, \gamma_1 \oplus H^2(\delta_2))$ and $[\delta_1 \oplus G^2(\gamma_2)]_{k_1} = 0^{k_1}$.

As soon as both equalities hold, DS outputs $[\delta_1 \oplus G^2(\gamma_2)]^n$. If no such pair is found, REJECT is returned.

4.3.3 Analysis

In this section, we analyze the success probability of the reduction.

Notation

When we have found the pre-image of c^* and thus inverted f, we could output the expected results s_1^* and stop the reduction. However, for this analysis, we assume the reduction goes on and that $\mathcal{B}$ only outputs it or the list of queries asked to H^1 once $\mathcal{A}_2$ has answered b'.

In order to proceed to the analysis of the success probability of the above-mentioned reduction, one needs to set up notations. First, we still denote with a star $*$ all variables related to the challenge ciphertext c^*, obtained from the encryption oracle. Indeed this ciphertext of either m_0 or m_1 implicitly defined hash values, but the corresponding pairs may not appear in the G^1, G^2, H^1, H^2 lists. All other variables refer to the decryption query c, asked by the adversary to the decryption oracle, and thus to be decrypted by this simulation. We consider several further events about a ciphertext queried to the decryption oracle.

- $\mathsf{AskH'} = \mathsf{AskH}^1 \wedge \mathsf{AskH}^2$
  - AskH^1 : the events that query s_1^* has been asked to H^1.
  - AskH^2 : the events that query s_2^* has been asked to H^2.

- $\mathsf{AskG'} = \mathsf{AskG}^1 \wedge \mathsf{AskG}^2$
  - AskG^1 : the event that r_1^* has been asked to G^1.
  - AskG^2 : the event that r_2^* has been asked to G^2.
- $\mathsf{Bad} = \mathsf{G}^1\mathsf{Bad} \vee \mathsf{DBad}$
  - $\mathsf{G}^1\mathsf{Bad}$: the event that r_1^* has been asked to G^1, but the answer is something other than $s_1^* \oplus (m_b \parallel 0^{k_1})$. Note that the event $\mathsf{G}^1\mathsf{Bad}$ implies AskG^1.
  - DBad : the event that decryption simulator fails.
- $\mathsf{CBad} = \mathsf{R'Bad} \vee \mathsf{S'Bad}$
  - $\mathsf{S'Bad} = \mathsf{S}^1\mathsf{Bad} \vee \mathsf{S}^2\mathsf{Bad}$
    - $\mathsf{S}^1\mathsf{Bad}$: the event that $s_1 = s_1^*$
    - $\mathsf{S}^2\mathsf{Bad}$: the event that $s_2 = s_2^*$
  - $\mathsf{R'Bad} = \mathsf{R}^1\mathsf{Bad} \vee \mathsf{R}^2\mathsf{Bad}$
    - $\mathsf{R}^1\mathsf{Bad}$: the event that $r_1 = r_1^*$
    - $\mathsf{R}^2\mathsf{Bad}$: the event that $r_2 = r_2^*$
- $\mathsf{AskR'S'} = \mathsf{AskR'} \vee \mathsf{AskS'}$
  - $\mathsf{AskR'} = \mathsf{AskR}^1 \wedge \mathsf{AskR}^2$
    - AskR^1 : the event that $r_1 = t \oplus H^2(s_2)$ has been asked to G^1.
    - AskR^2 : the event that $r_2 = G^1(r_1)$ has been asked to G^2.
  - $\mathsf{AskS'} = \mathsf{AskS}^1 \wedge \mathsf{AskS}^2$
    - AskS^1 : the event that s_1 has been asked to H^1.
    - AskS^2 : the event that s_2 has been asked to H^2.
- Fail : the event that the above decryption oracle simulator outputs a wrong decryption answer to query c. Therefore in the global reduction, the event DBad will be set to true as soon as one decryption simulation fails.

Analysis of the decryption oracle

We analyze the success probability of the decryption oracle simulator DS. We claim the following computational assumption.

Lemma 2. *When at most one ciphertext $c^* = f(s_1^*, t^*)$ has been directly obtained from the encryption oracle, but s_1^* has not been asked to H^1, the decryption oracle simulation DS can correctly produce the decryption oracle's output on query $c(\neq c^*)$ with probability greater than ε', within time bound t', where*

$$\varepsilon' \geq 1 - \left(\frac{2}{2^{k_1}} + \frac{2}{2^{c_1}} + \frac{2q'_G + 1}{2^{k_0}} + \frac{2q'_G + 1}{2^{c_2}}\right) \text{ and } t' \leq q'_G \cdot q'_H \cdot (T_f + O(1)).$$

Since our goal is to prove the security relative to partial domain one-wayness of f, we are only interested in the probability of the event Fail while $\neg\mathsf{AskH}^1$ occured which may be split according to other events. Granted $\neg\mathsf{CBad} \wedge \mathsf{AskR'S'}$, the simulation is perfect and cannot fail. Thus we have to consider the complementary events,

$$\begin{aligned}\Pr[\mathsf{Fail}|\neg\mathsf{AskH}^1] &= \Pr[\mathsf{Fail}\wedge\mathsf{CBad}|\neg\mathsf{AskH}^1] + \Pr[\mathsf{Fail}\wedge\neg\mathsf{CBad}|\neg\mathsf{AskH}^1]\\ &= \Pr[\mathsf{Fail}\wedge\mathsf{CBad}|\neg\mathsf{AskH}^1] + \Pr[\mathsf{Fail}\wedge\neg\mathsf{CBad}\wedge\neg\mathsf{AskR'S'}|\neg\mathsf{AskH}^1].\end{aligned}$$

Concerning the latter contribution to the right hand side, we first note that

$$\neg\mathsf{AskR'S'} = \neg\mathsf{AskR'}\vee\neg\mathsf{AskS'} = (\neg\mathsf{AskR'})\vee(\neg\mathsf{AskS'}\wedge\mathsf{AskR'}).$$

Forgetting $\neg\mathsf{AskH}^1$ for a while, one gets that

$$\begin{aligned}\Pr[\mathsf{Fail}\wedge\neg\mathsf{CBad}\wedge\neg\mathsf{AskR'S'}] &= \Pr[\mathsf{Fail}\wedge\neg\mathsf{CBad}\wedge((\neg\mathsf{AskR'})\vee(\neg\mathsf{AskS'}\wedge\mathsf{AskR'}))]\\ &= \Pr[\mathsf{Fail}\wedge\neg\mathsf{CBad}\wedge\neg\mathsf{AskR'}] + \Pr[\mathsf{Fail}\wedge\neg\mathsf{CBad}\wedge\neg\mathsf{AskS'}\wedge\mathsf{AskR'}]\\ &\leq \Pr[\mathsf{Fail}\wedge\neg\mathsf{R'Bad}\wedge\neg\mathsf{AskR'}] + \Pr[\mathsf{Fail}\wedge\neg\mathsf{S'Bad}\wedge\neg\mathsf{AskS'}\wedge\mathsf{AskR'}]\\ &\leq \Pr[\mathsf{Fail}|\neg\mathsf{AskR'}\wedge\neg\mathsf{R'Bad}] + \Pr[\mathsf{AskR'}|\neg\mathsf{AskS'}\wedge\neg\mathsf{S'Bad}].\end{aligned}$$

Let consider the $\Pr[\mathsf{Fail}|\neg\mathsf{AskR'}\wedge\neg\mathsf{R'Bad}]$. Taking into account the events $\neg$R'Bad, $G^2(r_2)$ is unpredictable, and thus the probability that $[s_1\oplus G^2(r_2)]_{k_1} = 0^{k_1}$ is less than 2^{-k_1}. We should also consider the probability that one does not ask r_1 to G^1 but asks r_2 to G^2 by chance. This probability is less than 2^{-c_1}. We can estimate that $\Pr[\mathsf{Fail}|\neg\mathsf{AskR'}\wedge\neg\mathsf{R'Bad}]\leq 2^{-k_1}+2^{c_1}$.

On the other hand, the probability of having asked r_1, r_2 to G^1, G^2, without any information about $H^1(s_1)$ or $H^2(s_2)$ and thus $\Pr[\mathsf{AskR'}|\neg\mathsf{AskS'}\wedge\neg\mathsf{S'Bad}]\leq q'_G\cdot(2^{-k_0}+2^{-c_2})$.

Furthermore, this event is independent of AskH^1, which yield

$$\Pr[\mathsf{Fail}\wedge\neg\mathsf{CBad}\wedge\neg\mathsf{AskR'S'}|\neg\mathsf{AskH}^1]\leq 2^{-k_1}+2^{-c_1}+q'_G\cdot(2^{-k_0}+2^{-c_2}).$$

We now focus on the former term, $\mathsf{Fail}\wedge\mathsf{CBad}$ while $\neg\mathsf{AskH}^1$. It can be split according to the disjoint sub-cases of CBad, which are S'Bad and $\neg$S'Bad $\wedge$ R'Bad. Then

$$\begin{aligned}\Pr[\mathsf{Fail}\wedge\mathsf{CBad}|\neg\mathsf{AskH}^1] &= \Pr[\mathsf{Fail}\wedge(\mathsf{S'Bad}\vee(\mathsf{R'Bad}\wedge\neg\mathsf{S'Bad}))|\neg\mathsf{AskH}^1]\\ &\leq \Pr[\mathsf{Fail}|\mathsf{S'Bad}\wedge\neg\mathsf{AskH}^1] + \Pr[\mathsf{R'Bad}|\neg\mathsf{S'Bad}\wedge\neg\mathsf{AskH}^1].\end{aligned}$$

The latter event means that R'Bad occurs provided $s_1\neq s_1^*$ and $s_2\neq s_2^*$ and the adversary has not queried s_1^* from H^1. When s_1^* has not been asked to H^1 and $s_1\neq s_1^*$ and $s_2\neq s_2^*$, $H^1(s_1^*)$ and $H^2(s_2^*)$ is unpredictable. Then event R'Bad occurs with probability at most $2^{-c_1}+2^{-k_0}$.

The former event can be further split according to AskR', it is upper-bounded by

$$\Pr[\mathsf{Fail}|\mathsf{S'Bad}\wedge\neg\mathsf{AskH}^1]\leq\Pr[\mathsf{AskR'}|\mathsf{S'Bad}\wedge\neg\mathsf{AskH}^1]+\Pr[\mathsf{Fail}|\neg\mathsf{AskR'}\wedge\mathsf{S'Bad}\wedge\neg\mathsf{AskH}^1].$$

The former event means that r_1, r_2 are asked to G^1, G^2, respectively, whereas $H^2(s_2)$ is unpredictable. Since r_1, r_2 is unpredictable, the probability of this event is at most $q'_G\cdot(2^{-k_0}+2^{-c_1})$. On the other hand, the latter event cannot hold with probability greater than $2^{-c_1}+2^{-k_1}$. To sum up,

$$\begin{aligned}&\Pr[\mathsf{Fail}|\mathsf{S'Bad}\wedge\neg\mathsf{AskH}^1]\leq 2^{-c_1}+2^{-k_1}+q'_G\cdot(2^{-k_0}+2^{-c_1}),\\ &\Pr[\mathsf{Fail}\wedge\mathsf{C'Bad}|\neg\mathsf{AskH}^1]\leq 2^{-c_1}+2^{-k_1}+(q'_G+1)\cdot(2^{-k_0}+2^{-c_1}),\\ &\Pr[\mathsf{Fail}|\neg\mathsf{AskH}^1]\leq\frac{2}{2^{k_1}}+\frac{q'_G+3}{2^{c_1}}+\frac{2q'_G+1}{2^{k_0}}+\frac{q'_G}{2^{c_2}}.\end{aligned}$$

The running time of this simulator includes just the computation of $c=f(\delta_1,\gamma_1\oplus H^2(\delta_2))$ and $[\delta_1\oplus G^2(\gamma_2)]_{k_1}=0^{k_1}$ for all possible pairs and is thus bounded by $q'_G\cdot q'_H\cdot(T_f+O(1))$.

Analysis of the success probability

This section analyzes the success probability of our reduction with respect to the advantage of the CCA2 adversary. The goal of the reduction is given $c^* = f(s_1^*, t^*)$ to obtain s_1^*. Therefore, the success probability is obtained by the probability that event AskH^1 occurs during the reduction.

We thus evaluate $\Pr[\mathsf{AskH}^1]$ by splitting event AskH^1 according to event Bad.

$$\Pr[\mathsf{AskH}^1] = \Pr[\mathsf{AskH}^1 \wedge \mathsf{Bad}] + \Pr[\mathsf{AskH}^1 \wedge \neg\mathsf{Bad}].$$

First let us evaluate the first term.

$$\begin{aligned}
\Pr[\mathsf{AskH}^1 \wedge \mathsf{Bad}] &= \Pr[\mathsf{Bad}] - \Pr[\neg\mathsf{AskH}^1 \wedge \mathsf{Bad}] \\
&\geq \Pr[\mathsf{Bad}] - \Pr[\neg\mathsf{AskH}^1 \wedge \mathsf{G}^1\mathsf{Bad}] - \Pr[\neg\mathsf{AskH}^1 \wedge \mathsf{DBad}] \\
&\geq \Pr[\mathsf{Bad}] - \Pr[\mathsf{G}^1\mathsf{Bad}|\neg\mathsf{AskH}^1] - \Pr[\mathsf{DBad}|\neg\mathsf{AskH}^1] \\
&\geq \Pr[\mathsf{Bad}] - \Pr[\mathsf{AskG}^1|\neg\mathsf{AskH}^1] - \Pr[\mathsf{DBad}|\neg\mathsf{AskH}^1] \\
&\geq \Pr[\mathsf{Bad}] - \frac{q'_G}{2^{k_0}} - \frac{q'_G}{2^{c_2}} - q_D\left(\frac{2}{2^{k_1}} + \frac{q'_G+3}{2^{c_1}} + \frac{2q'_G+1}{2^{k_0}} + \frac{q'_G}{2^{c_2}}\right) \\
&= \Pr[\mathsf{Bad}] - \frac{2q_Dq'_G + q_D + q'_G}{2^{k_0}} + \frac{q_Dq'_G + q'_G}{2^{c_2}} - \frac{2q_D}{2^{k_1}} - \frac{2q_Dq'_G + 3q_D}{2^{c_1}}.
\end{aligned}$$

Here $\Pr[\mathsf{DBad}|\neg\mathsf{AskH}]$ is directly obtained from lemma, and $\Pr[\mathsf{AskG}^1|\neg\mathsf{AskH}^1] \leq q'_G \cdot (2^{-k_0} + 2^{-c_2})$. We then evaluate the second term.

$$\begin{aligned}
\Pr[\mathsf{AskH}^1 \wedge \neg\mathsf{Bad}] &\geq \Pr[A = b \wedge \mathsf{AskH}^1 \wedge \neg\mathsf{Bad}] \\
&= \Pr[A = b \wedge \neg\mathsf{Bad}] - \Pr[A = b \wedge \neg\mathsf{AskH}^1 \wedge \neg\mathsf{Bad}].
\end{aligned}$$

The first term can be evaluated

$$\begin{aligned}
\Pr[A = b \wedge \neg\mathsf{Bad}] &= \Pr[A = b] - \Pr[A = b \wedge \mathsf{Bad}] \\
&\geq \Pr[A = b] - \Pr[\mathsf{Bad}] \\
&\geq \frac{\varepsilon}{2} + \frac{1}{2} - 2\Pr[\mathsf{DBad}] - \Pr[\mathsf{Bad}] \\
&\geq \frac{\varepsilon}{2} + \frac{1}{2} - 2q_D\left(\frac{2}{2^{k_1}} + \frac{q'_G+3}{2^{c_1}} + \frac{2q'_G+1}{2^{k_0}} + \frac{q'_G}{2^{c_2}}\right) - \Pr[\mathsf{Bad}].
\end{aligned}$$

Then we evaluate the second term. Here when $\neg\mathsf{AskH}^1$ occurs, $H^1(s_1^*)$ is unpredictable, thus $r_1^* = t^* \oplus H^2(s_2^*)$ is unpredictable and so is b as well. This fact is independent from event $\neg\mathsf{Bad}$. Hence $\Pr[A = b|\neg\mathsf{AskH}^1 \wedge \neg\mathsf{Bad}] = 1/2$. We can also estimate that

$$\Pr[\mathsf{Bad}] + (\Pr[\mathsf{AskH}^1 \wedge \neg\mathsf{Bad}] + \Pr[\neg\mathsf{AskH}^1 \wedge \neg\mathsf{Bad}]) = 1.$$

We have

$$\begin{aligned}
\Pr[\mathsf{AskH}^1 \wedge \neg\mathsf{Bad}] &\geq \frac{\varepsilon}{2} + \frac{1}{2} - 2q_D\left(\frac{2}{2^{k_1}} + \frac{q'_G+3}{2^{c_1}} + \frac{2q'_G+1}{2^{k_0}} + \frac{q'_G}{2^{c_2}}\right) - \Pr[\mathsf{Bad}] \\
&\qquad -(1 - \Pr[\mathsf{AskH}^1 \wedge \neg\mathsf{Bad}] - \Pr[\mathsf{Bad}]) \cdot \frac{1}{2} \\
&= \frac{\varepsilon + \Pr[\mathsf{AskH}^1 \wedge \neg\mathsf{Bad}] - \Pr[\mathsf{Bad}]}{2} - 2q_D\left(\frac{2}{2^{k_1}} + \frac{q'_G+3}{2^{c_1}} + \frac{2q'_G+1}{2^{k_0}} + \frac{q'_G}{2^{c_2}}\right).
\end{aligned}$$

	Table size	Security
OAEP	$2^{k-k_0} \times k_0 + 2^{k_0} \times (k-k_0)$	$\varepsilon - (10q_Dq'_G + 5q_D + q'_G) \cdot 2^{-k_0} - (10q_D) \cdot 2^{-k_1}$ $t' \leq t + q'_G \cdot q'_H \cdot (T_f + O(1))$
Our variant	$2^{k-k_0} \times c_2 + 2^{c_2} \times k_0 + 2^{k_0} \times c_1 + 2^{c_1} \times (k-k_0)$	$\varepsilon - (10q_Dq'_G + 5q_D + q'_G) \cdot 2^{-k_0} - (5q_Dq'_G + q'_G) \cdot 2^{-c_2} - (10q_D) \cdot 2^{-k_1} - (9q_Dq_G + 7q_D) \cdot 2^{-c_1}$ $t' \leq t + q'_G \cdot q'_H \cdot (T_f + O(1))$

Figure 2: The comparison of OAEP and our variant

and we can figure out $\Pr[\mathsf{AskH}^1 \wedge \neg\mathsf{Bad}]$ as

$$\Pr[\mathsf{AskH}^1 \wedge \neg\mathsf{Bad}] = \Pr[\mathsf{AskH}^1 \wedge \neg\mathsf{Bad}] \geq \varepsilon - \Pr[\mathsf{Bad}] - 4q_D \left(\frac{2}{2^{k_1}} + \frac{q'_G + 3}{2^{c_1}} + \frac{2q'_G + 1}{2^{k_0}} + \frac{q'_G}{2^{c_2}} \right).$$

In the conclusion, we have

$$\Pr[\mathsf{AskH}^1] \quad \geq \quad \varepsilon - \frac{10q_Dq'_G + 5q_D + q'_G}{2^{k_0}} - \frac{5q_Dq'_G + q'_G}{2^{c_2}} - \frac{10q_D}{2^{k_1}} - \frac{9q_Dq_G + 7q_D}{2^{c_1}}.$$

Analysis of the complexity

Note that during the execution of $\mathcal{B}$ for any new G^1 query γ_1, one has to look at all query answers and checks whether they satisfy $c = f(\delta_1, \gamma_1 \oplus H^2(\delta_2))$ and $[\delta_1 \oplus G^2(\gamma_2)]_{k_1} = 0^{k_1}$. Proper bookkeeping allows the computation to be done once for each pair, when the query is asked to the hash function. Thus, the time complexity of the overall reduction is $t' \leq t + q'_G \cdot q'_H \cdot (T_f + O(1))$.

4.4 Comparison

We now compare our variant and OAEP with respect to the table size of the random oracle.

In OAEP, the encryption and decryption algorithms make use of two random oracles H and G. Let $H : \{0,1\}^{k-k_0} \to \{0,1\}^{k_0}$ and $G : \{0,1\}^{k_0} \to \{0,1\}^{k-k_0}$. We need $2^{k-k_0} \times k_0$ table size to realize the random oracle H and $2^{k_0} \times (k-k_0)$ to G. Therefore, we need $2^{k-k_0} \times k_0 + 2^{k_0} \times (k-k_0)$ table size in order to realize OAEP.

In our variant, we use two random oracles H^1 and H^2 instead of H in OAEP. We also use two random oracles G^1 and G^2 instead of G in OAEP. Let $G^1 : \{0,1\}^{k_0} \to \{0,1\}^{c_1}$, $G^2 : \{0,1\}^{c_1} \to \{0,1\}^{k-k_0}$, $H^1 : \{0,1\}^{k-k_0} \to \{0,1\}^{c_2}$, and $H^2 : \{0,1\}^{c_2} \to \{0,1\}^{k_1}$. In the above description, let $c_1 = O((\log(k-k_0))^c)$ and $c_2 = O((\log k_1)^c)$, where c is a constant number. We need $2^{k-k_0} \times c_2$ table size to realize the random oracle H^1, $2^{c_2} \times k_0$ to H^2, $2^{k_0} \times c_1$ to G^1 and $2^{c_1} \times (k-k_0)$ to G^2. Thus, we need $2^{k-k_0} \times c_2 + 2^{c_2} \times k_0 + 2^{k_0} \times c_1 + 2^{c_1} \times (k-k_0)$ table size in order to realize our variant.

We show the table sizes and the security of OAEP and our variant in Figure 2. We can certainly reduce the table size of OAEP by applying our idea.

5 Concluding remarks

We have considered a different direction on the study of the schemes in the random oracle model. We have focused on the size of the tables necessary to describe all of the entries to be potentially queried in the random oracle model. We have shown how to reduce the table sizes of the schemes for encryption and signature in the random oracle model. In particular, we have applied this idea to PSS-R and OAEP and shown the security of our schemes.

References

[1] Bellare, M., and Rogaway, P. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *1st ACM Conference on Computer and Communications Security* (Fairfax, Virginia, USA, 1993), ACM, pp. 62–73.

[2] Bellare, M., and Rogaway, P. Optimal Asymmetric Encryption - How to Encrypt with RSA. In *Advances in Cryptology – EUROCRYPT '94* (Perugia, Italy, May 1994), A. De Santis, Ed., vol. 950 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 92–111.

[3] Bellare, M., and Rogaway, P. The Exact Security of Digital Signatures - How to Sign with RSA and Rabin. In Maurer [7], pp. 399–416.

[4] Canetti, R., Goldreich, O., and Halevi, S. The Random Oracle Methodology, Revisited. In *32nd Annual ACM Wymposium on Theory of Computing* (Dalllas, USA, 1998), ACM, pp. 209–218.

[5] Fujisaki, E., Okamoto, T., Pointcheval, D., and Stern, J. RSA–OAEP Is Secure under the RSA Assumption. In *Advances in Cryptology – CRYPTO 2001* (Santa Barbara, California, USA, August 2001), J. Kilian, Ed., vol. 2139 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 260–274.

[6] Goldwasser, S., and Taumann, Y. On the (In)security of the Fiat-Shamir Paradigm. In *Advances in Cryptology – EUROCRYPT 2003* (Warsaw, Poland, May 2003), E. Biham, Ed., vol. 2656 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 102–114.

[7] Maurer, U., Ed. *Advances in Cryptology – EUROCRYPT '96* (Saragossa, Spain, May 1996), vol. 1070 of *Lecture Notes in Computer Science*, Springer-Verlag.

[8] Pointcheval, D., and Stern, J. Security Proofs for Signature Schemes. In Maurer [7], pp. 387–398.

A PSS-R

A.1 Security

The following theorem proves the security of PSS-R based on the security of RSA.

Proposition 3. *Suppose that RSA is (t', ε')-secure. Then for any q_{sig}, q_{hash} the signature scheme PSS-R$[k_0, k_1]$ is $(t, q_{sig}, q_{hash}, \varepsilon)$-secure, where $t(k) = t'(k) - [q_{sig}(k) + q_{hash}(k) + 1] \cdot k_0 \cdot O(k^3)$ and $\varepsilon(k) = \varepsilon'(k) + [2(q_{sig}(k) + q_{hash}(k))^2](2^{-k_0} + 2^{-k_1}) + 2^{-k_0}$.*

Proof. Let F be a forger which $(t, q_{sig}, q_{hash}, \varepsilon)$-breaks PSS-R. We present an inverter I which (t', ε')-breaks the RSA family.

The simulation

The input to I is N, e and η, where η is chosen at random from Z_N^*, and N, e, d are chosen by running the generator $RSA(1^k)$. We let $f : Z_N^* \to Z_N^*$ be $f(x) = x^e \bmod N$. The inverter I want to compute $f^{-1}(\eta) = \eta^d \bmod N$. It forms the public key N, e, and starts running F on input this key. The forger F will make oracle queries. The inverter I must answer itself. We let $Q_1, ..., Q_{q_{sig}+q_{hash}}$ denote the sequence of oracle queries that F makes. In the process of answering these queries, the inverter I will build or define the function H, G. The inverter I prepare two empty lists named List G and List H.

I maintain a counter i, initially 0, which is incremented for each query.

Answering signing queries

1. Increment i and let $M_i = Q_i$.
2. Pick $r \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$.
3. If $r_j = r_i$ for some $j \leq i$, then abort.
4. Repeat $x_i \stackrel{R}{\leftarrow} Z_N^*$; $y_i \leftarrow f(x_i)$ until the first bit of y_i is 0.
5. Break up y_i to write it as $0 \parallel w_i \parallel {r_i}^* \parallel {M_i}^*$.
6. Set $H(M_i \parallel r_i) = w_i$ in the List H.
7. If $w_j = w_i$ for some $j \leq i$, then abort.
8. Set $G(w_i) = {r_i}^* \oplus r_i \parallel M_i \oplus {M_i}^*$ into the List G.
9. Return x_i to F as the answer to the signing query $Q_i = M_i$.

Answering H oracle queries

1. Increment i and break up Q_i as $M_i \parallel r_i$.
2. Say Q_i is old if $M_j \parallel r_j = M_i \parallel r_i$ for some $j \leq i$ and new otherwise. Now if Q_i is old then set $(w_i, {r_i}^*, {M_i}^*) = (w_j, {r_j}^*, {M_j}^*)$ and return w_j. Else go to next step.
3. Repeat $x_i \stackrel{R}{\leftarrow} Z_N^*$; $z_i \leftarrow f(x_i)$; $y_i \leftarrow \eta z_i \bmod N$ until the first bit of y_i is 0.
4. Break up y_i to write it as $0 \parallel w_i \parallel {r_i}^* \parallel {M_i}^*$.
5. Set $H(M_i \parallel r_i) = w_i$ into the List H.
6. If $w_j = w_i$ for some $j \leq i$, then abort.
7. Set $G(w_i) = {r_i}^* \oplus r_i \parallel M_i \oplus {M_i}^*$ into the List G.
8. Return w_i to F as the answer to the H oracle query $Q_i = M_i \parallel r_i$.

Answering G oracle queries

1. Increment i and let $w_i = Q_i$.
2. If $w_i = w_j$ for some $j \leq i$, then return $G(w_j)$. Else pick a string $\alpha \stackrel{R}{\leftarrow} \{0,1\}^{k-k_1-1}$, set $G(w_i) = \alpha$ into the List G and return α to F as the answer to the G oracle query Q_i.

We want to arrange that, if F later forges a signature of M using a seed r then we invert f at η. To arrange this, we will associate to query $M \parallel r$ an image of the form $\eta {x_i}^e$, where x_i is random. Thus if F later comes up with $f^{-1}(\eta {x_i}^e) = x_i \eta^d$, then I can divide out x_i and recover $\eta^d = f^{-1}(\eta)$.

Analysis

We first show that the validity of the simulation. We insist that the forger must use the random oracles G, H in order to output the correct signature.

We define some events to show the validity of the simulation. We denote (M^+, x^+) as a correct pair of the message and the signature.

E_1 : Forger outputs (M^+, x^+) without asking corresponding $M^+ \parallel r^+$ to the H oracle queries, but asking corresponding w^+ to the G oracle queries.

E_2 : Forger outputs (M^+, x^+) without asking corresponding w^+ to the G oracle queries but asking corresponding $M^+ \parallel r^+$ to the H oracle queries.

E_3 : Forger uses the random oracles G, H in order to output (M^+, x^+) asking any oracle.

Our goal is to estimate the probability that E_3 occurs. It is not so hard to figure out $\Pr[E_1]$, $\Pr[E_2]$, and $\Pr[E_3]$. Since $H(M \parallel r)$ is unpredictable, the probability that E_1 occurs is at most 2^{-k_1}.Similarly, we can figure out that $\Pr[E_2] \leq 2^{-(k-k_1-1)}$.

We can estimate that $\Pr[E_3] \geq 1 - (\Pr[E_1] + \Pr[E_2])$.

It shows that the forger must use the random oracle G, H to forge a signature.

We now prove the theorem. We define the following events to analyze the security probability.

E_4 : The inverter I never abort in step 3 and 7 in answering signing queries and step 6 in answering H oracle queries.

E_5 : The inverter I cannot stop in step 4 in answering signing queries and step 3 in answering H oracle queries within $1 + k_0$ repetition.

We can estimate that $\Pr[E_4] \leq 2(q_{sig} + q_{hash})^2(2^{-k_0} + 2^{-k_1})$.

Let us consider E_5. The time for step 4 in answering signing queries and step 3 in answering H oracle queries can not be bounded. However, the expected time is two execution of the loop. Then we make it a rule to stop the loop after $1 + k_0$ steps. Then, we can estimate that $\Pr[E_5] \leq 2^{-k_0}$.

We now claim that with propability at least $\varepsilon - \Pr[E_4] - \Pr[E_5]$, we can break the RSA.

Now we can figure out that

$\varepsilon(k) = \varepsilon'(k) + [2(q_{sig}(k) + q_{hash}(k))^2](2^{-k_0} + 2^{-k_1}) + 2^{-k_0}$.

The running time of I is that of F plus the time to choose the y_i's. The main thing here is one RSA computation for each y_i, which is cubic time (or better). We can say that

$t(k) = t'(k) - [q_{sig}(k) + q_{hash}(k) + 1] \cdot k_0 \cdot O(k^3)$.

□

B OAEP

B.1 The security result

In the following, we prove that the scheme is IND-CCA2 in the random oracle model, relative to the partial-domain one-wayness of function f. More precisely, the following exact security result holds.

Proposition 4. *Let $\mathcal{A}$ be a CCA2-adversary against the semantic security of OAEP conversion $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, with advatage ε and running time t, making q_D, q_G and q_H queries to the decryption oracle, and the hash function G and H respectively. Then, there exist an algorithm $\mathcal{B}$ such that $Succ^{pd-ow}(\mathcal{B})$ is greater than*

$$\frac{1}{q_H}(\varepsilon - \frac{10q_Dq_G + 5q_D + q_G}{2^{k_0}} - \frac{10q_D}{2^{k_1}}),$$

and whose running time $t' \leq t + q_Gq_H(T_f + O(1))$, where T_f denotes the time complexity of function f.

In order to prove this proposition relative to the partial-domain one-wayness of the permutation, we prove the following lemma.

Lemma 3. *Let $\mathcal{A}$ be a CCA2-adversary against the semantic security of $OAEP$ conversion $\mathcal{D}^{G,H}(sk, c)$, with advantage ε and running time t, making q_D, q_G and q_H queries to the decryption oracle, and the hash function G and H respectively. Then, there exist an algorithm $\mathcal{B}$ such that $Succ^{s-pd-ow}(\mathcal{B})$ is greater than*

$$\varepsilon - \frac{10q_Dq_G + 5q_D + q_G}{2^{k_0}} - \frac{10q_D}{2^{k_1}},$$

and whose running time $t' \leq t + q_Gq_H(T_f + O(1))$, where T_f denotes the time complexity of function f.

B.2 The proof of the proposition

We prove lemma in three stages.

The first we present the reduction of the CCA2-adversary $\mathcal{A}$ attacking indistinguishability to algorithm $\mathcal{B}$ for breaking the partial-domain one-wayness of f. Note that, in the present proof, we are interested in the security under the partial-domain one-wayness of f.

The second we present the simulation of the algorithm $\mathcal{B}$. We present how $\mathcal{B}$ simulates the random oracles G, H and the decryption oracle.

Finally, we analyze the success probability of our reduction in total. We analyze the decryption oracle simulation employed in this reduction works correctly with overwhelming probability under the partial-domain one-wayness of f and insist the lemma and the proposition.

B.2.1 The top level description of the reduction

In this first part, we recall how the reduction operates. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary against the semantic security of $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, under the chosen ciphertext attack. Within time bound t, $\mathcal{A}$ asks q_D, q_G, q_H queries to the decryption oracle and the random oracles G and H respectively, and distinguishes the right plaintext with an advantage greater that ε. Let us describe the reduction of $\mathcal{B}$.

1. Algorithm $\mathcal{B}$ is given a function f which is defined by the public key and $c^* \leftarrow f(s^*, t^*)$, for $(s^*, t^*) \stackrel{R}{\leftarrow} \{0,1\}^{k-k_0} \times \{0,1\}^{k_0}$. The aim of $\mathcal{B}$ is to recover the partial pre-image s^* to c^*.
2. Algorithm $\mathcal{B}$ runs $\mathcal{A}_1$ on the public data, and gets a pair of messages $\{m_0, m_1\}$ as well as states information st. It chooses a random bit b, and then gives c^* to $\mathcal{A}_1$, as the cipher text of m_b. Algorithm $\mathcal{B}$ simulates the answers to the queries of $\mathcal{A}_1$ to the decryption oracle and random oracles G, H respectively. See the description of these simulations below.
3. Algorithm $\mathcal{B}$ runs $\mathcal{A}_2(c^*, st)$ and finally gets answer b'. Algorithm $\mathcal{B}$ simulates the answers to the queries of $\mathcal{A}_2$ to the decryption oracle and random oracle G, H respectively. See the description of these simulations below. Algorithm $\mathcal{B}$ then outputs the partial pre-image s^* of c^*, if one has been found among the queries asked to H, or the list of queries asked to H.

B.2.2 The simulation

In this section, we explain how the algorithm $\mathcal{B}$ simulates the random oracle G, H and the decryption oracle.

The random oracles

On the simulation of the random oracles, $\mathcal{B}$ has to simulate the random oracle answers, managing query answer lists List H and List G for the oracles G, H, respectively. They are initially set to empty lists;

Answering G oracle queries

- If there is a γ asked before, return corresponding $G(\gamma)$ that $\mathcal{B}$ answered before.
- For a fresh query γ to G, look at the List H and for any query δ asked to H with answer $H(\delta)$, one build $z = \gamma \oplus H(\delta)$ and check whether $c^* = f(\delta, z)$.
  - If for some δ that relation holds, function f has been inverted, and we can still correctly simulate G, by answering $G(\gamma) = \delta \oplus (m_b \parallel 0^{k_1})$. Note that $G(\gamma)$ is a uniformly distributed value since $\delta = s^*$, and the latter is uniformly distributed.
  - If there is no such δ, choose $G(\gamma)$ at random and put $(\gamma, G(\gamma))$ into the List G.

Answering H oracle queries

- If there is a δ asked before, return corresponding $H(\delta)$ that $\mathcal{B}$ answered before.
- For a fresh query δ to H, output a random value $H(\delta)$ and the pair $(\delta, H(\delta))$ is concatenated to the List H. Note that once again for any $(\gamma, G(\gamma)) \in$ List G, one may build $z = \gamma \oplus H(\delta)$, and checks whether $c^* = f(\delta, t)$. If for some γ that relation holds, we have inverted the function f.

The decryption oracle

On query $c = f(s,t)$ to the decryption oracle, decryption oracle simulation DS looks at each query-answer $(\gamma, G(\gamma)) \in$ List G and $(\delta, H(\delta)) \in$ List H.

For each pair taken from both lists, check whether $c = f(\delta, \gamma \oplus H(\delta))$ and $[\delta \oplus G(\gamma)]_{k_1} = 0^{k_1}$.

As soon as both equalities hold, DS outputs $[\delta \oplus G(\gamma)]^n$. If no such pair is found, return REJECT.

B.2.3 Analysis

In this section, we analyze the success probability of the reduction.

Notation

When we have found the pre-image of c^* and thus inverted f, we could output the expected results s^* and stop the reduction. However, for this analysis, we assume the reduction goes on and that $\mathcal{B}$ only outputs it or the list of queries asked to H once $\mathcal{A}_2$ has answered b'.

Even if no answer is explicitly specified, expect by a random value for new queries, some are implicitly defined. Indeed c is defined to be a ciphertext of m_b with random tape r^* :

$r^* \leftarrow H(s^*) \oplus t^*$ and $G(r^*) \leftarrow s^* \oplus (m_b \parallel 0^{k_1})$.

Since $H(s^*)$ is randomly defined, r^* can be seen as a random variable.

In order to proceed to the analysis of the success probability of the above-mentioned reduction, one need to set up notations. First, we still denote with a star $*$ all variables related to the challenge ciphertext c^*, obtained from the encryption oracle. Indeed this ciphertext of either m_0 or m_1 implicitly defined hash values, but the corresponding pairs may not appear in the G or H lists. All other variables refer to the decryption query c, asked by the adversary to the decryption oracle, and thus to be decrypted by this simulation. We consider several further events about a ciphertext queried to the decryption oracle.

- AskH : the events that query s^* has been asked to H.
- AskG : the event that r^* has been asked to G.
- Bad= GBad $\vee$ DBad
  - GBad : the event that r^* has been asked to G, but the answer is something other than $s^* \oplus (m_b \parallel 0^{k_1})$. Note that the event GBad implies AskG.
  - DBad : the event that decryption simulator fails.
- CBad = RBad $\vee$ SBad
  - SBad : the event that $s = s^*$,
  - RBad : the event that $r = r^*$ and thus $H(s) \oplus t = H(s^*) \oplus t^*$
- AskRS = AskR $\vee$ AskS

- AskR : the event that $r = t \oplus H(s)$ has been asked to G.
- AskS : the event that s has been asked to H.

- Fail : the event that the above decryption oracle simulator outputs a wrong decryption answer to query c. Therefore in the global reduction, the event DBad will be set to true as soon as one decryption simulation fails.

Analysis of the decryption oracle

We analyze the success probability of the decryption oracle simulator DS. We claim the following computational assumption.

Lemma 4. *When at most one ciphertext $c^* = f(s^*, t^*)$ has been directly obtained from the encryption oracle, but s^* has not been asked to H, the decryption oracle simulation DS can correctly produce the decryption oracle's output on query $c(\neq c^*)$ with probability greater than ε', within time bound t', where*

$$\varepsilon' \geq 1 - (\frac{2}{2^{k_1}} + \frac{2q_G + 1}{2^{k_0}}) \text{ and } t' \leq q_G \cdot q_H \cdot (T_f + O(1)).$$

Since our goal is to prove the security relative to partial domain one-wayness of f, we are only interested in the probability of the event Fail while $\neg\mathsf{AskH}$ occured which may be split according to other events. Granted $\neg\mathsf{CBad} \wedge \mathsf{AskRS}$, the simulation is perfect and cannot fail. Thus we have to consider the complementary events.

$$\begin{aligned}\Pr[\mathsf{Fail}|\neg\mathsf{AskH}] &= \Pr[\mathsf{Fail} \wedge \mathsf{CBad}|\neg\mathsf{AskH}] + \Pr[\mathsf{Fail} \wedge \neg\mathsf{CBad}|\neg\mathsf{AskH}] \\ &= \Pr[\mathsf{Fail} \wedge \mathsf{CBad}|\neg\mathsf{AskH}] + \Pr[\mathsf{Fail} \wedge \neg\mathsf{CBad} \wedge \neg\mathsf{AskRS}|\neg\mathsf{AskH}].\end{aligned}$$

Concerning the latter contribution to the right hand side, we first note that

$$\neg\mathsf{AskRS} = \neg\mathsf{AskR} \vee \neg\mathsf{AskS} = (\neg\mathsf{AskR}) \vee (\neg\mathsf{AskS} \wedge \mathsf{AskR}).$$

Forgetting $\neg\mathsf{AskH}$ for a while, one gets that

$$\begin{aligned}\Pr[\mathsf{Fail} \wedge \neg\mathsf{CBad} \wedge \neg\mathsf{AskRS}] &= \Pr[\mathsf{Fail} \wedge \neg\mathsf{CBad} \wedge ((\neg\mathsf{AskR}) \vee (\neg\mathsf{AskS} \wedge \mathsf{AskR}))] \\ &= \Pr[\mathsf{Fail} \wedge \neg\mathsf{CBad} \wedge \neg\mathsf{AskR}] + \Pr[\mathsf{Fail} \wedge \neg\mathsf{CBad} \wedge \neg\mathsf{AskS} \wedge \mathsf{AskR}] \\ &\leq \Pr[\mathsf{Fail} \wedge \neg\mathsf{RBad} \wedge \neg\mathsf{AskR}] + \Pr[\mathsf{Fail} \wedge \neg\mathsf{SBad} \wedge \neg\mathsf{AskS} \wedge \mathsf{AskR}] \\ &\leq \Pr[\mathsf{Fail}|\neg\mathsf{AskR} \wedge \neg\mathsf{RBad}] + \Pr[\mathsf{AskR}|\neg\mathsf{AskS} \wedge \neg\mathsf{SBad}].\end{aligned}$$

However, without having asked r to G, taking into account the further events $\neg\mathsf{RBad}$, $G(r)$ is unpredictable, and thus the probability that $[s \oplus G(r)]_{k_1} = 0^{k_1}$ is less than 2^{-k_0}. On the other hand, the probability of having asked r to G, without any information about $H(s)$ and thus about r is less than $q_G \cdot 2^{-k_0}$. Furthermore, this event is independent of AskH, which yield

$$\Pr[\mathsf{Fail} \wedge \neg\mathsf{CBad} \wedge \neg\mathsf{AskRS}|\neg\mathsf{AskH}] \leq 2^{-k_0} + q_G \cdot 2^{-k_0}.$$

We now focus on the former term, $\mathsf{Fail} \wedge \mathsf{CBad}$ while $\neg\mathsf{AskH}$. It can be split according to the disjoint sub-cases of CBad, which are SBad and $\neg\mathsf{SBad} \wedge \mathsf{RBad}$. Then

$$\begin{aligned}\Pr[\mathsf{Fail} \wedge \mathsf{CBad}|\neg\mathsf{AskH}] &= \Pr[\mathsf{Fail} \wedge (\mathsf{SBad} \vee (\mathsf{RBad} \wedge \neg\mathsf{SBad}))|\neg\mathsf{AskH}] \\ &\leq \Pr[\mathsf{Fail}|\mathsf{SBad} \wedge \neg\mathsf{AskH}] + \Pr[\mathsf{RBad}|\neg\mathsf{SBad} \wedge \neg\mathsf{AskH}].\end{aligned}$$

The latter event means that RBad occurs provided $s \neq s^*$ and the adversary has not queried s^* from H. When s^* has not been asked to H and $s \neq s^*$, $H(s^*)$ is unpredictable and independent of $H(s)$ as well as t and t^*. Then event RBad $H(s^*) = H(s) \oplus t \oplus t^*$ occurs with probability at most 2^{k_0}. The former event can be further split according to AskR, it is upper-bounded by

$$\Pr[\mathsf{Fail}|\mathsf{SBad} \wedge \neg\mathsf{AskH}] \leq \Pr[\mathsf{AskR}|\mathsf{SBad} \wedge \neg\mathsf{AskH}] + \Pr[\mathsf{Fail}|\neg\mathsf{AskR} \wedge \mathsf{SBad} \wedge \neg\mathsf{AskH}].$$

The former event means that r is asked to G whereas $s = s^*$ and $H(s^*)$ is unpredictable thus $H(s)$ is unpredictable. Since r is unpredictable, the probability of this event is at most $q_G \cdot 2^{-k_0}$. On the other hand, the latter event means that the simulator rejects the valid ciphertext c whereas $H(s)$ is unpredictable and r is not asked to G. It follows from $s = s^*$ that $r = r^*$, and thus $G(r)$ is unpredictable. Then the redundancy cannot hold with probability greater than 2^{-k_1}. To sum up,

$$\begin{aligned}
&\Pr[\mathsf{Fail}|\mathsf{SBad} \wedge \neg\mathsf{AskH}] \leq 2^{-k_1} + q_G \cdot 2^{-k_0},\\
&\Pr[\mathsf{Fail} \wedge \mathsf{CBad}|\neg\mathsf{AskH}] \leq 2^{-k_1} + (q_G + 1) \cdot 2^{-k_0},\\
&\Pr[\mathsf{Fail}|\neg\mathsf{AskH}] \leq \frac{2}{2^{k_1}} + \frac{2q_G + 1}{2^{k_0}}.
\end{aligned}$$

The running time of this simulator includes just the computation of $c = f(\delta, \gamma \oplus H(\delta))$ and $[\delta \oplus G(\gamma)]_{k_1} = 0^{k_1}$ for all possible pairs and is thus bounded by $q_G \cdot q_H \cdot (T_f + O(1))$.

Analysis of the success probability

This section analyzes the success probability of our reduction with respect to the advantage of the CCA2 adversary. The goal of the reduction is given $c^* = f(s^*, t^*)$ to obtain s^*. Therefore, the success probability is obtained by the probability that event AskH occurs during the reduction.

We thus evaluate $\Pr[\mathsf{AskH}]$ by splitting event AskH according to event Bad.

$$\Pr[\mathsf{AskH}] = \Pr[\mathsf{AskH} \wedge \mathsf{Bad}] + \Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}].$$

First let us evaluate the first term.

$$\begin{aligned}
\Pr[\mathsf{AskH} \wedge \mathsf{Bad}] &= \Pr[\mathsf{Bad}] - \Pr[\neg\mathsf{AskH} \wedge \mathsf{Bad}]\\
&\geq \Pr[\mathsf{Bad}] - \Pr[\neg\mathsf{AskH} \wedge \mathsf{GBad}] - \Pr[\neg\mathsf{AskH} \wedge \mathsf{DBad}]\\
&\geq \Pr[\mathsf{Bad}] - \Pr[\mathsf{GBad}|\neg\mathsf{AskH}] - \Pr[\mathsf{DBad}|\neg\mathsf{AskH}]\\
&\geq \Pr[\mathsf{Bad}] - \Pr[\mathsf{AskG}|\neg\mathsf{AskH}] - \Pr[\mathsf{DBad}|\neg\mathsf{AskH}]\\
&\geq \Pr[\mathsf{Bad}] - \frac{q_G}{2^{k_0}} - q_D\left(\frac{2}{2^{k_1}} + \frac{2q_G + 1}{2^{k_0}}\right)\\
&= \Pr[\mathsf{Bad}] - \frac{2q_Dq_G + q_D + q_G}{2^{k_0}} - \frac{2q_D}{2^{k_1}}.
\end{aligned}$$

Here $\Pr[\mathsf{DBad}|\neg\mathsf{AskH}] \leq q_D \cdot (2 \cdot 2^{-k_1} + (2q_G + 1) \cdot 2^{-k_0})$ is directly obtained from lemma, and $\Pr[\mathsf{AskG}|\neg\mathsf{AskH}] \leq q_G \cdot 2^{-k_0}$.

We then evaluate the second term.

$$\begin{aligned}
\Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}] &\geq \Pr[A = b \wedge \mathsf{AskH} \wedge \neg\mathsf{Bad}]\\
&= \Pr[A = b \wedge \neg\mathsf{Bad}] - \Pr[A = b \wedge \neg\mathsf{AskH} \wedge \neg\mathsf{Bad}].
\end{aligned}$$

The first term can be evaluated

$$\begin{aligned}
\Pr[A = b \wedge \neg\mathsf{Bad}] &= \Pr[A = b] - \Pr[A = b \wedge \mathsf{Bad}]\\
&\geq \Pr[A = b] - \Pr[\mathsf{Bad}]\\
&\geq \frac{\varepsilon}{2} + \frac{1}{2} - 2\Pr[\mathsf{DBad}] - \Pr[\mathsf{Bad}]\\
&\geq \frac{\varepsilon}{2} + \frac{1}{2} - 2q_D\left(\frac{2}{2^{k_1}} + \frac{2q_G + 1}{2^{k_0}}\right) - \Pr[\mathsf{Bad}].
\end{aligned}$$

Then we evaluate the second term. Here when $\neg\mathsf{AskH}$ occurs, $H(s^*)$ is unpredictable, thus $r^* = t^* \oplus H(s^*)$ is unpredictable and so is b as well. This fact is independent from event $\neg\mathsf{Bad}$. Hence $\Pr[A = b|\neg\mathsf{AskH} \wedge \neg\mathsf{Bad}] = 1/2$. We can also estimate that

$$\Pr[\mathsf{Bad}] + (\Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}] + \Pr[\neg\mathsf{AskH} \wedge \neg\mathsf{Bad}]) = 1.$$

We have

$$\begin{aligned}\Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}] &\geq \frac{\varepsilon}{2} + \frac{1}{2} - 2q_D\left(\frac{2}{2^{k_1}} + \frac{2q_G+1}{2^{k_0}}\right) - \Pr[\mathsf{Bad}] - (1 - \Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}] - \Pr[\mathsf{Bad}]) \cdot \frac{1}{2} \\ &= \frac{\varepsilon + \Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}] - \Pr[\mathsf{Bad}]}{2} - 2q_D\left(\frac{2}{2^{k_1}} + \frac{2q_G+1}{2^{k_0}}\right).\end{aligned}$$

and we can figure out $\Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}]$ as

$$\Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}] = \Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}] \geq \varepsilon - \Pr[\mathsf{Bad}] - 4q_D\left(\frac{2}{2^{k_1}} + \frac{2q_G+1}{2^{k_0}}\right).$$

In the conclusion, we have

$$\begin{aligned}\Pr[\mathsf{AskH}] &\geq \Pr[\mathsf{Bad}] - \frac{2q_Dq_G + q_D + q_G}{2^{k_0}} - \frac{2}{q_D}2^{k_1} + \Pr[\mathsf{AskH} \wedge \neg\mathsf{Bad}] \\ &= \varepsilon - \frac{10q_Dq_G + 5q_D + q_G}{2^{k_0}} - \frac{10q_D}{2^{k_1}}.\end{aligned}$$

Analysis of the complexity

Note that during the execution of $\mathcal{B}$ for any new G query γ, one has to look at all query answer pairs $(\delta, H(\delta))$ in the H List and checks whether it satisfies $c = f(\delta, \gamma \oplus H(\delta))$ and $[\delta \oplus G(\gamma)]_{k_1} = 0^{k_1}$. Proper bookkeeping allows the computation to be done once for each pair, when the query is asked to the hash function. Thus the time complexity of the overall reduction is $t' \leq t + q_G \cdot q_H \cdot (T_f + O(1))$.