OAEP++ : A Very Simple Way to Apply OAEP to Deterministic OW-CPA Primitives

Kazukuni Kobara and Hideki Imai

Institute of Industrial Science, The University of Tokyo
4-6-1, Komaba, Meguro-ku, Tokyo, 153-8505 Japan
Tel: +81-3-5452-6232
FAX: +81-3-5452-6631
E-mail: {kobara,imai}@iis.u-tokyo.ac.jp

Abstract. We prove in the random oracle model that OAEP++, which was proposed by us at the rump session of Asiacrypt 2000, can generate IND-CCA2 ciphers using deterministic OW-CPA cryptographic primitives. Note that OAEP++ differs from OAEP^{++} proposed by Jonsson in [4]. While OAEP^{++} requires a non-malleable block cipher, OAEP++ does not require such additional functions. The security reduction of OAEP++ is as tight as that of OAEP^{++}.

Keywords random oracle model, provable security, OAEP, IND-CCA2, OW-CPA

1 Introduction

In [6], Shoup showed that OAEP [2] is not a sufficient conversion, even in the random oracle model [1], to generate an IND-CCA2 cipher from a deterministic cryptographic primitive satisfying OW-CPA. Currently, it is known that PDOW-CPA (Partial-Domain One-Wayness against CPA) is a sufficient condition for OAEP to generate an IND-CCA2 cipher from a deterministic cryptographic primitive in the random oracle model [3].

Since PDOW-CPA is a stronger assumption[1] than OW-CPA, it is worthwhile to loosen the assumption to OW-CPA with small costs. A couple of solutions have already obtained by modifying OAEP slightly. One is OAEP+ proposed by Shoup [6]. Another is OAEP^{++} proposed by Jonsson [4]. The other is OAEP++ [5]. The advantages of OAEP++ are: (1) It does not require any additional functions such as non-malleable block ciphers. (2) It can encrypt any long message. (3) The security reduction is as tight as that of OAEP^{++}.

In this paper, we give a security proof that OAEP++ can generate IND-CCA2 ciphers in the random oracle model using deterministic OW-CPA primitives.

[1] In some primitives, such as RSA, the gap between PDOW-CPA and OW-CPA is small [3].

2 Notations

We use the following notations in this paper:

$Prep(m)$: Preprocessing to a message m, such as data-compression, data-padding and so on. Its inverse is represented as $Prep^{-1}()$.
$Hash(x)$: One-way hash function of an arbitrary length binary string x to a fixed length binary string.
$Gen(x)$: Generator of a cryptographically secure pseudo random sequences of arbitrary length from a fixed length seed x.
$Len(x)$: Bit-length of x.
$Msb_{x_1}(x_2)$: The left x_1 bits of x_2.
$Lsb_{x_1}(x_2)$: The right x_1 bits of x_2.
$Const$: Predetermined public constant.
$Rand$: Random source which generates a truly random (or computationally indistinguishable pseudo random) sequence.
$\mathcal{E}(x)$: Encryption of x using a deterministic OW-CPA primitive function.
$\mathcal{D}(x)$: Decryption of x using a deterministic OW-CPA primitive function.
k_1 : Bit-length of the full domain input of $\mathcal{E}(x)$.
k_1' : Bit-length of the output of $\mathcal{D}(x)$.

3 OAEP++ Conversion

OAEP++ is a slightly extended version of OAEP, proposed in [5] to fix the bug in OAEP. The description of OAEP++ is given in Fig. 1. When $k_1 = Len(y_1||y_2)$, i.e. $Len(y_4) = 0$, it is equivalent to OAEP. Thus IND-CCA2 is satisfied in the random oracle model under the assumption of PDOW-CPA of $\mathcal{E}()$ [3]. Even when $Len(y_1||y_2) > k_1 > Len(y_1)$, IND-CCA2 is satisfied under the same assumption as PDOW-CPA since we can see that the underlying primitive function $\mathcal{E}'()$ takes $(y_3||y_4)$ as its input and then outputs $(\mathcal{E}(y_3)||y_4)$.

When $k_1 \leq Len(y_1)$, IND-CCA2 is satisfied under the assumption of OW-CPA. Note that one can always satisfy $k_1 \leq Len(y_1)$ by increasing $Len(y_1)$, i.e. by increasing either $Len(Const)$ or $Lem(\bar{m})$. The corresponding security proof is given is the next section.

4 Security Proof

The following theorem holds on the OAEP++:

Theorem 1 *To break the indistinguishability of encryption of OAEP++ using CCA2 is polynomial equivalent, in the random oracle model, to break OW-CPA of the underlying deterministic function when $k_1 \leq Len(y_1)$ holds.*

Encryption of m:

$$\begin{aligned} r &:= Rand \\ \bar{m} &:= Prep(m) \\ y_1 &:= (\bar{m}||Const) \oplus Gen(r) \\ y_2 &:= r \oplus Hash(y_1) \\ (y_3||y_4) &:= (y_1||y_2) \\ c &:= (\mathcal{E}(y_3)||y_4) \\ \text{return} & \quad c \end{aligned}$$

Decryption of c:

$$\begin{aligned} (y_5||y_4) &:= c \\ (y_1||y_2) &:= (\mathcal{D}(y_5)||y_4) \\ r &:= y_2 \oplus Hash(y_1) \\ (\bar{m}'||Const') &:= y_1 \oplus Gen(r) \\ \text{If } Const' &= Const, \\ \text{return} & \quad Prep^{-1}(\bar{m}') \\ \text{Otherwise} & \quad \text{return } \perp \end{aligned} \tag{1}$$

Fig. 1. OAEP++ with a deterministic encryption function where $Len(y_2) = Len(r)$, $Len(y_3) = k_1$, $Len(y_5) = k_1'$ and $\perp$ means the given cipher is invalid.

This theorem can be proven by showing Lemma 3 is true. Before we show it, we describe the one-wayness, the indistinguishability of encryption, random oracles, and Lemma 2, respectively.

4.1 One-Wayness

In the notion of the one-wayness, an adversary $\mathcal{A}$ is given a ciphertext of a random plaintext, and then tries to find the whole preimage of the ciphertext. If $\mathcal{A}$ only has access to encryption oracles, experiment is in the adaptive-chosen-plaintext scenario.

4.2 Indistinguishability of Encryption

In the notion of the indistinguishability of encryption, an adversary $\mathcal{A}$ selects two distinct plaintexts m_0 and m_1 of the same length in the find stage, and then, in the guess stage, $\mathcal{A}$ is given c which is the encryption of m_b where b is either 0 or 1 with the probability of $1/2$. Then $\mathcal{A}$ tries to guess b. The advantage of $\mathcal{A}$ is defined by $2Pr(\text{Win}) - 1$ where $Pr(\text{Win})$ denotes the expected probability of $\mathcal{A}$ guessing b correctly. If $\mathcal{A}$ has a decryption oracle D (which rejects invalid ciphertexts or decrypts any other valid ones than the target one c), it is called that this experiment is in the adaptive-chosen-ciphertext scenario. Otherwise, if $\mathcal{A}$ does not have it, it is called that this experiment is in the adaptive-chosen-plaintext scenario.

4.3 Random Oracle

A random oracle is an ideal hash or an ideal generator which returns truly random numbers distributed uniformly over the output region for a new query, but it returns the same value for the same query. On such random oracles, the following lemma is true.

Lemma 1 *Suppose that f is a random oracle. Then it is impossible to get any significant information on $f(x)$ without asking x to the oracle, even if one knows all the other input-output pairs of f except x.*

It is obvious that Lemma 1 is true since the output value of f is determined truly at random.

4.4 Adaptive-Chosen-Ciphertext Security

Lemma 2 (Adaptive-Chosen-Plaintext Security) *Suppose that there exists, for any $Hash$ and any Gen, an algorithm $\mathcal{A}$ which accepts m_0, m_1 and c of conversion OAEP++ where c is the ciphertext of m_b and $b \in \{0,1\}$, asks at most q_G queries to Gen, asks at most q_H queries to $Hash$, runs in at most τ steps and guesses b with advantage of ϵ. Then one can design an algorithm $\mathcal{B}$ which accepts a ciphertext $\bar{c}$ of the primitive PKC, runs in τ' steps and decrypts it with probability ϵ' where*

$$\epsilon' \geq \epsilon - \frac{q_G}{2^{Len(r)}},$$
$$\tau' = \tau + Poly(n, q_G, q_H)$$

and $Poly(n, q_G, q_H)$ denotes a polynomial of n, q_G and q_H.

Proof.

The algorithm $\mathcal{B}$ can be constructed as follows. First the algorithm $\mathcal{B}$ simulates both Gen and $Hash$ referred by the algorithm $\mathcal{A}$. From the assumption of $\mathcal{A}$ in Lemma 2, $\mathcal{A}$ must be able to distinguish b with the advantage of ϵ for any Gen and any $Hash$ as long as the algorithm $\mathcal{B}$ simulates them correctly.

$\mathcal{B}$ begins by initializing two lists, G-list and H-list, to empty. These G-list and H-list are the tables of inputs and the corresponding outputs for describing Gen and $Hash$, respectively. It runs $\mathcal{A}$ as the find-stage mode simulating $\mathcal{A}'s$ oracles as follows. When $\mathcal{A}$ makes an oracle call h of $Hash$, $\mathcal{B}$ provides it with a random string H, and adds h and H to the H-list. Similarly when $\mathcal{A}$ makes an oracle call g of Gen, $\mathcal{B}$ provides it with a random string G, and adds g and G to the G-list. Let (m_0, m_1) be the output of $\mathcal{A}$.

Let $y_4 = (y_5||y_2)$, i.e. $(y_1||y_2) = (y_3||y_5||y_2)$. $\mathcal{B}$ chooses $b \in \{0,1\}$, r and $(y_5||y_2)$ at random, and then defines both $Hash$ and Gen so that the ciphertext of m_b should be $(\bar{c}||y_5||y_2)$ where $\bar{c}$ is a ciphertext of the primitive PKC which $\mathcal{B}$ wants to decrypt. That is,

$$Gen(r) \stackrel{\text{def}}{=} (Prep(m_b) \ || \ Const) \oplus (y_3||y_5) \tag{2}$$
$$Hash(y_3||y_5) \stackrel{\text{def}}{=} y_2 \oplus r. \tag{3}$$

and $\mathcal{B}$ adds r and $(Prep(m_b)||Const) \oplus (y_3||y_5)$ to the G-list, and $(y_3||y_5)$ and $y_2 \oplus r$ to the H-list. $\mathcal{B}$ runs $\mathcal{A}$ as the guess-stage mode. For these Gen and $Hash$,

$\mathcal{A}$ must be able to distinguish b with the advantage of ϵ from the assumption in Lemma 2 as long as $\mathcal{B}$ simulates them correctly.[2]

Can $\mathcal{B}$ simulate them correctly for any queries? The answer is "no" since $\mathcal{B}$ does not know y_3, and thus $\mathcal{B}$ cannot simulate Gen correctly when r is asked to it. We consider the following two events:

- **AskH** denotes the event that $(y_3||*)$ is asked to $Hash$ among the q_H queries to $Hash$ and that this query is performed before r is asked to Gen where $*$ denotes any string.
- **AskG** denotes the event that r is asked to Gen among the q_G queries to Gen and that this query is performed before $(y_3||*)$ is asked to $Hash$.

Since $Pr(\text{AskG} \wedge \text{AskH}) = 0$ in the above definition, the following holds

$$\begin{aligned} &Pr(\text{AskG} \vee \text{AskH}) \\ &= Pr(\text{AskG}) + Pr(\text{AskH}). \end{aligned} \tag{4}$$

Next, we estimate the upper-limit of $Pr(\text{Win})$, the probability of $\mathcal{A}$ guessing b correctly. Since the mapping from $(r||Prep(m_b)||Const)$ to $(y_3||y_5||y_2)$ is bijective defined by Gen and $Hash$ where Lemma 1 holds, one cannot get any information on the connectivity between $(y_3||y_5||y_2)$ and $(r||Prep(m_b)||Const)$ without asking r to Gen or asking $(y_3||y_5)$ to $Hash$. That is, one cannot guess b with a significant probability after the event $(\neg\text{AskG} \wedge \neg\text{AskH})$. After the other event, i.e. after the event $(\text{AskG} \vee \text{AskH})$, $\mathcal{A}$ might guess b with more significant probability. By assuming this probability to be 1, the upper-limit of $Pr(\text{Win})$ is obtained as follows:

$$\begin{aligned} Pr(\text{Win}) &\le Pr(\text{AskG} \vee \text{AskH}) \\ &\quad + \frac{(1 - Pr(\text{AskG} \vee \text{AskH}))}{2} \\ &\le \frac{Pr(\text{AskG} \vee \text{AskH}) + 1}{2}. \end{aligned} \tag{5}$$

From the definition of advantage, i.e. $Pr(\text{Win}) = (\epsilon + 1)/2$, the following relationship holds

$$Pr(\text{AskG} \vee \text{AskH}) \ge \epsilon. \tag{6}$$

Since r is chosen at random by $\mathcal{B}$, $\mathcal{A}$ cannot know it (without asking $(y_3||y_5)$ to $Hash$). Thus the probability of one query to Gen accidentally being r is $1/2^{Len(r)}$, and then that of at most q_G queries is given by

$$\begin{aligned} &Pr(\text{AskG}) \\ &\le 1 - \left(1 - \frac{1}{2^{Len(r)}}\right)^{q_G} \le \frac{q_G}{2^{Len(r)}}. \end{aligned} \tag{7}$$

[2] If $\mathcal{A}$ distinguishes b only for certain combinations of $Hash$ and Gen, then the fault must be in either Gen or $Hash$, or in both. This implies this fault can be easily removed just avoiding these combinations of Gen and $Hash$. Otherwise, i.e. if $\mathcal{A}$ distinguishes b for any $Hash$ and any Gen, the fault must be in the conversion structure itself.

The algorithm $\mathcal{B}$ can simulate both Gen and $Hash$ correctly unless the event AskG happens. And then, after the event AskH, $\mathcal{B}$ can recover the whole plaintext of the target ciphertext $\bar{c}$ of the primitive PKC. From (4), (6) and (7), the lower-limit of this probability is given by

$$\begin{aligned}
&Pr(\neg\text{AskG} \wedge \text{AskH}) \\
&= Pr(\text{AskH}) \\
&= Pr(\text{AskG} \vee \text{AskH}) - Pr(\text{AskG}) \\
&\geq \epsilon - \frac{q_G}{2^{Len(r)}}.
\end{aligned} \tag{8}$$

The number of steps of $\mathcal{B}$ is at most $\tau + (T_{Enc} + T_H) \cdot q_H + T_G \cdot q_G$ where T_G is both for checking whether a query to Gen is new or not and for returning the corresponding value, and then T_H is that of $Hash$. T_{Enc} is the number of steps for checking whether a new query h_j to $Hash$ satisfies the event AskH by checking whether the Hamming weight of $z := \bar{c} \oplus Lsb_k(h_j)G'$ is τ or not. Since these parameters, T_{Enc}, T_G and T_H can be written in a polynomial of n, q_G and q_H, the total number of steps of $\mathcal{B}$ is also written in a a polynomial of them.

□

Lemma 3 (Adaptive-Chosen-Ciphertext Security) *Suppose that there exists, for any $Hash$ and Gen, an algorithm $\mathcal{A}$ which accepts m_0, m_1 and c of OAEP++, asks at most q_G queries to Gen, asks at most q_H queries to $Hash$, asks at most q_D queries to a decryption oracle D, runs in at most τ steps and guesses b with advantage of ϵ. Then one can design an algorithm $\mathcal{B}$ which accepts a ciphertext $\bar{c}$ of the primitive PKC, runs in τ' steps and decrypts it with probability ϵ' where*

$$\begin{aligned}
\epsilon' &\geq \epsilon - \frac{q_G}{2^{Len(r)}} - \frac{q_D(q_G+1)}{2^{Len(r)}} - \frac{q_D}{2^{Len(Const)}}, \\
\tau' &= \tau + Poly(n, q_G, q_H, q_D)
\end{aligned}$$

and $Poly(n, q_G, q_H, q_D)$ denotes a polynomial of n, q_G, q_H and q_D.

Proof.

From the assumption of $\mathcal{A}$ in Lemma 3, $\mathcal{A}$ must be able to distinguish the given ciphertext with advantage of ϵ as long as $\mathcal{B}$ simulates them correctly. How to simulate both Gen and $Hash$ is the same as in the proof of Lemma 2. The decryption oracle D can be simulated using the following plaintext-extractor. It accepts a ciphertext, say $(\bar{c}'||y_5'||y_2')$, and then either outputs the corresponding plaintext or rejects it as an invalid ciphertext.

It works as follows. Let g_i and G_i denote the i-th pair of query and its answer for Gen. And then let h_j and H_j denote the j-th pair of query and its answer for $Hash$. From the queries and the answers obtained while simulating Gen and $Hash$, the plaintext-extractor finds y_3' satisfying below:

$$y_3' = Msb_{k_1}(h_j) \tag{9}$$

$$\bar{c}' = \mathcal{E}(y_3') \tag{10}$$

If found, it evaluates $H' := Hash(y_3'||y_5')$, $G' := Gen(H' \oplus y_2')$ and then checks whether $Lsb_{Len(Const)}(G' \oplus y_2') \oplus (y_3'||y_5')) = Const$. If so it outputs $Msb_{Len(m')}(G' \oplus y_2') \oplus (y_3'||y_5'))$. Otherwise, it rejects $(\bar{c}'||y_5'||y_2')$.

If $\mathcal{A}$ asks a valid ciphertext to D without asking $(y_3'||*)$ to $Hash$, it rejects the valid ciphertext, and therefore does not simulate D correctly. However it is a small chance for $\mathcal{A}$ to generate it without asking it. Since the definition of "valid" is to satisfy

$$\begin{aligned} &Lsb_{Len(Const)}(Gen(Hash(y_3'||y_5') \oplus y_2') \\ &= Const \oplus Lsb_{Len(Const)}(y_3'||y_5') \end{aligned} \tag{11}$$

and, from Lemma 1, it is impossible for $\mathcal{A}$ to know whether (11) is true or not without asking $(y_3'||y_5')$ to $Hash$.

We evaluate the possibility that one ciphertext $c' = (\bar{c}'||y_5'||y_2')$ can be valid without asking $(y_3'||y_5')$ to $Hash$. We consider the following events

- **AskG'** denotes the event that $(Hash(y_3'||y_5') \oplus y_2')$ is asked to Gen among at most q_G queries from $\mathcal{A}$.
- **AskH'** denotes the event that $(y_3'||*)$ is asked to $Hash$ among at most q_H queries from $\mathcal{A}$.
- **ValidR1** denotes the event that the given ciphertext satisfies

$$\begin{aligned} &Hash(y_3'||y_5') \oplus y_2' \\ &= Hash(y_3||y_5) \oplus y_2, \end{aligned} \tag{12}$$

$$\begin{aligned} &Lsb_{Len(Const)}(y_3'||y_5') \\ &= Lsb_{Len(Const)}(y_3||y_5), \end{aligned} \tag{13}$$

$$\begin{aligned} &(y_2', Msb_{Len(m')}(y_3'||y_5')) \\ &\neq (y_2, Msb_{Len(Prep(m_b))}(y_3||y_5)) \end{aligned} \tag{14}$$

  and thus (11) where y_2, y_3 and y_5 are variables of a valid challenge ciphertext satisfying (11).
- **ValidC1** denotes the event that the given ciphertext satisfies both (11) and

$$Hash(y_3'||y_5') \oplus y_2' \neq Hash(y_3||y_5) \oplus y_2. \tag{15}$$

- **Valid1** denotes the event that the given ciphertext satisfies (11). Note that

$$\begin{aligned} &Pr(\text{Valid1}) \\ &= Pr(\text{ValidR1} \vee \text{ValidC1}) \\ &= Pr(\text{ValidR1}) + Pr(\text{ValidC1}|\neg\text{ValidR1}) \\ &\quad \cdot Pr(\neg\text{ValidR1}). \end{aligned} \tag{16}$$

- **Fail1** denotes the event that the above plaintext-extractor outputs a wrong answer against one given ciphertext to D.

Since it does not return any plaintext from an invalid ciphertext, and also it returns the correct answer after the events AskH'. Thus

$$
\begin{aligned}
Pr(\text{Fail1}) = {} & Pr(\text{Valid1}|\text{AskG'} \wedge \neg\text{AskH'}) \\
& \cdot Pr(\text{AskG'} \wedge \neg\text{AskH'}) \\
& + Pr(\text{Valid1}|\neg\text{AskG'} \wedge \neg\text{AskH'}) \\
& \cdot Pr(\neg\text{AskG'} \wedge \neg\text{AskH'})
\end{aligned} \tag{17}
$$

where

$$
Pr(\text{AskG'} \wedge \neg\text{AskH'}) \leq \frac{q_G}{2^{Len(r')}} \tag{18}
$$

$$
Pr(\text{Valid1}|\text{AskG'} \wedge \neg\text{AskH'}) \leq 1 \tag{19}
$$

$$
Pr(\neg\text{AskG'} \wedge \neg\text{AskH'}) \leq 1 \tag{20}
$$

and

$$
\begin{aligned}
& Pr(\text{Valid1}|\neg\text{AskG'} \wedge \neg\text{AskH'}) \\
& = Pr(\text{ValidR1}|\neg\text{AskG'} \wedge \neg\text{AskH'}) \\
& \quad + Pr(\text{ValidC1}|\neg\text{ValidR1} \wedge \neg\text{AskG'} \wedge \neg\text{AskH'}) \\
& \quad \cdot Pr(\neg\text{ValidR1}|\neg\text{AskG'} \wedge \neg\text{AskH'}).
\end{aligned} \tag{21}
$$

Since $Pr(\text{ValidR1}|\neg\text{AskG'} \wedge \neg\text{AskH'}) = \frac{1}{2^{Len(r')}}$ and $Pr(\text{ValidC1}|\neg\text{ValidR1} \wedge \neg\text{AskG'} \wedge \neg\text{AskH'}) = \frac{1}{2^{Len(Const)}}$, the upper-limit of $Pr(\text{Fail1})$ is given by

$$
Pr(\text{Fail1}) \leq \frac{q_G + 1}{2^{Len(r')}} + \frac{1}{2^{Len(Const)}}. \tag{22}
$$

Next, we consider the following event Fail where

- **Fail** denotes the event that the above plaintext-extractor outputs at least one wrong answer against at most q_D queries to D.

The upper-limit of $Pr(\text{Fail})$ is given by

$$
\begin{aligned}
Pr(\text{Fail}) & \leq 1 - (1 - Pr(\text{Fail1}))^{q_D} \\
& \leq \frac{q_D(q_G + 1)}{2^{Len(r')}} + \frac{q_D}{2^{Len(Const)}}.
\end{aligned} \tag{23}
$$

Unless either Fail or AskG happens, $\mathcal{B}$ can correctly simulate the oracles referred by $\mathcal{A}$. In addition, when AskH happens, $\mathcal{B}$ can recover the whole plaintext of $\bar{c}$, the ciphertext of the primitive PKC. The lower-limit of this probability $Pr(\text{AskH} \wedge \neg\text{AskG} \wedge \neg\text{Fail})$ is given by

$$
\begin{aligned}
& Pr(\text{AskH} \wedge \neg\text{AskG} \wedge \neg\text{Fail}) \\
& = Pr(\text{AskH} \wedge \neg\text{AskG}) \\
& \quad - Pr(\text{AskH} \wedge \neg\text{AskG} \wedge \text{Fail}) \\
& \geq Pr(\text{AskH} \wedge \neg\text{AskG}) - Pr(\text{Fail}) \\
& \geq \epsilon - \frac{q_G}{2^{Len(r)}} - \frac{q_D(q_G + 1)}{2^{Len(r)}} - \frac{q_D}{2^{Len(Const)}}.
\end{aligned} \tag{24}
$$

The number of steps of $\mathcal{B}$ is at most $\tau + (T_{Enc} + T_H) \cdot q_H + T_G \cdot q_G + T_D \cdot q_D$ where T_{Enc}, T_G and T_H are the same as the parameters in the proof of Lemma 2. The number of steps T_D is that of the knowledge-extractor to verify whether (11) holds and then to return the result. Since these parameters, T_{Enc}, T_G, T_H and T_D can be written in a polynomial of n, q_G, q_H and q_D, the total number of steps of $\mathcal{B}$ is also written in a polynomial of them.

$\square$

5 Conclusion

We proved in the random oracle model that a slightly extended version of OAEP, i.e. OAEP++, can generate IND-CCA2 ciphers using deterministic OW-CPA cryptographic primitives when $k_1 \leq Len(y_1)$ holds. OAEP++ has the following advantages: (1) It does not require any additional functions such as non-malleable block ciphers. (2) It can encrypt any long message. (3) The security reduction is as tight as that of OAEP^{++}.

OAEP++ is also easily extensible to multiple encryption of any combination of both deterministic and probabilistic cryptosystems. Extension is performed as follows, first one enlarges the length of either $Const$ or $\bar{m}$ so that y_1 can be divided into n pieces of full domains of all the n underlying cryptosystems. For the probabilistic cryptosystems, randomness is deterministically generated from $Hash(r||index)$ where $index$ is a unique description of each probabilistic cryptosystem, and then the integrity of the randomness is checked after recovering r in the decoding process.

References

1. M. Bellare and P. Rogaway. "Random oracles are practical: A paradigm for designing efficient protocols". In *Proc. of the First ACM CCCS*, pages 62–73, 1993.
2. M. Bellare and P. Rogaway. "Optimal asymmetric encryption". In *Proc. of EUROCRYPT '94, LNCS 950*, pages 92–111, 1995.
3. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. "RSA-OAEP is secure under the RSA assumption". In *Proc. of CRYPTO 2001, LNCS 2139*, pages 260–274, 2001.
4. J. Jonsson. "An OAEP variant with a tight security proof draft 1.0". In *IACR ePrint archive 2002/034: http://eprint.iacr.org/2002/034*, 2002.
5. K. Kobara and H. Imai. "OAEP++ – another simple bug fix in OAEP –". In *Rump Session at Asiacrypt 2000*, 2000.
6. V. Shoup. "OAEP Reconsidered". In *Proc. of CRYPTO 2001*, pages 239–259, 2001.