

# Optimal Asymmetric Encryption and Signature Paddings

Benoît Chevallier-Mames<sup>1,2</sup>, Duong Hieu Phan<sup>2</sup>, and David Pointcheval<sup>2</sup>

<sup>1</sup> Gemplus, France – [benoit.chevallier-mames@gemplus.com](mailto:benoit.chevallier-mames@gemplus.com)

<sup>2</sup> ENS, Paris, France – [{david.pointcheval,duong.hieu.phan}@ens.fr](mailto:{david.pointcheval,duong.hieu.phan}@ens.fr)

**Abstract.** Strong security notions often introduce strong constraints on the construction of cryptographic schemes: semantic security implies probabilistic encryption, while the resistance to existential forgeries requires redundancy in signature schemes. Some paddings have thus been designed in order to provide these minimal requirements to each of them, in order to achieve secure primitives.

A few years ago, Coron et al. suggested the design of a common construction, a *universal padding*, which one could apply for both encryption and signature. As a consequence, such a padding has to introduce both randomness and redundancy, which does not lead to an optimal encryption nor an optimal signature.

In this paper, we refine this notion of universal padding, in which a part can be either a random string in order to introduce randomness or a zero-constant string in order to introduce some redundancy. This helps us to build, with a unique padding, optimal encryption and optimal signature: first, in the random-permutation model, and then in the random-oracle model. In both cases, we study the concrete sizes of the parameters, for a specific security level: The former achieves an optimal bandwidth.

## 1 Introduction

When one deals with public-key encryption, chosen-ciphertext security [23] is by now the basic required security notion. Similarly, for signatures, resistance to existential forgeries against adaptive chosen-message attacks [10] is also the minimal requirement. But strong security is not enough, it has to be achieved in an efficient way, according to various criteria: time, bandwidth, but also size of the code.

The first two above criteria are the most usual goals, and improvements are continuously proposed. When dealing with public-key cryptography, one can indeed note that fast paddings have been proposed for encryption [3, 19] and signature [4]. About the bandwidth, Phan and Pointcheval recently addressed this problem for encryption [21, 22], and proposed an optimal padding, w.r.t. this criteria, by avoiding redundancy. Most signatures with message-recovery [18, 16, 4] improve the bandwidth, but these solutions are not optimal, since redundancy and randomization are always added. The notable exception is the recent idea of Katz and Wang, that achieves tight security by using FDH, but also PSS-R, constructions [4] with only one additional bit, that is not random but dependent on the message [13].

The last criteria has been more recently considered, by Coron, Joye, Naccache and Paillier [5], with the so-called notion of *universal paddings*: the code size is reduced by using a common padding for both encryption and signature. For such a goal, they used a variant of PSS, called PSS-ES. Other solutions have thereafter been proposed, including those of Komano and Ohta [14]. But in all these constructions, the resulting encryption contains redundancy, and the signature is probabilistic.

### 1.1 Contribution

In this paper, we address this problem of efficiency, trying to optimize the three above criteria at the same time: for a time-efficient construction, we consider simple paddings; for a good bandwidth, we extend the work of [21, 22], by avoiding not only redundancy in encryption, but also randomization in signatures; additionally, we use the idea of the Katz-Wang construction [13] in order to achieve tight security in signature. Finally, about the size of the code, we optimize the common parts in the two paddings (for signature and encryption), by giving a relaxed version of *universal padding*. Furthermore, we analyze the security of these paddings, to be used for both encryption and signature, but in the extreme case where the same primitive (trapdoor one-way permutation which might optionally be assumed claw-free) is used for encryption and signature, at the same time, as already suggested in [12]: the same public/private key pair is used for encryption and signature.

More precisely, we study two paddings with the above *universal* property. The first one is based on the Full-Domain Permutation construction, studied in [11] for signature and in [21], for encryption, which can be proved optimal with the three above criteria in the random-permutation model. Hence the name of *Optimal Permutation-based Padding* (OPbP). Then, we also review the OAEP 3-rounds construction [21, 22] (OAEP3r), in the random-oracle model [2].

## 1.2 Redundancy and Randomness

A basic requirement for encryption, to achieve semantic security, is a probabilistic mechanism which is necessary to make distributions of ciphertexts indistinguishable. But until recently, chosen-ciphertext security was thought to furthermore imply redundancy in the ciphertext (for a kind of proof of knowledge/awareness of the plaintext [3, 1, 6].) However, this was not mandatory [21, 22], at least in the random-oracle model and in the ideal-cipher model. Existence of such schemes in the standard model is still an open problem.

Similarly, for signature, to prevent forgeries, some redundancy in the message-signature pair (or unique string in case of message-recovery feature) is required, which should be hard to satisfy without the signing key. But most of the signature schemes are probabilistic [25, 17, 4, 7], while it is not necessary (e.g. the FDH-signature, but with *loose* security). Recently, Katz and Wang proved that it was possible to achieve *tight* security with a deterministic construction very close to FDH-signature or PSS-R, by adding a single bit that is not random but dependent on the message [13]. More precisely, this additional bit should be not predictable by anyone else than the signer, and so Katz and Wang proposed that it results from a PRF computation.

## 1.3 Universal Paddings

The goal of universal padding is to design a padding which can not only be applied for signature and for encryption independently, but for both at the same time, with the same user's keys: the public key is used for both encryption and verification, while the private key is used for both decryption and signature.

In the security model, the adversaries (against either semantic security or existential unforgeability) are given access to both the signing and decryption oracles, which is not the security scenario considered when one deals with encryption and signature, independently. The decryption oracle may indeed help to forge signatures, and vice-versa.

# 2 Security Model

## 2.1 Signature Schemes

Digital signature schemes are the electronic version of handwritten signatures for digital documents: a user's signature on a message  $m$  is a string which depends on  $m$ , on public and secret data specific to the user and —possibly— on randomly chosen data, in such a way that anyone can check the validity of the signature by using public data only. In this section, we briefly review the main security notions [10].

**Definitions.** A signature scheme  $S = (\mathcal{K}, \mathcal{S}, \mathcal{V})$  is defined by the three following algorithms:

- The *key generation algorithm*  $\mathcal{K}$ . On input  $1^k$ , which is a formal notation for a machine with running time polynomial in  $k$  ( $1^k$  is indeed  $k$  in basis 1), the algorithm  $\mathcal{K}$  produces a pair  $(\text{pk}, \text{sk})$  of matching public and private keys. Algorithm  $\mathcal{K}$  is probabilistic. The input  $k$  is called the security parameter. The sizes of the keys, or of any problem involved in the cryptographic scheme, will depend on it, in order to achieve an appropriate security level (the expected minimal time complexity of any attack).
- The *signing algorithm*  $\mathcal{S}$ . Given a message  $m$  and a pair of matching public and private keys  $(\text{pk}, \text{sk})$   $\mathcal{S}$  produces a signature  $\sigma$ . The signing algorithm might be probabilistic.
- The *verification algorithm*  $\mathcal{V}$ . Given a signature  $\sigma$ , a message  $m$ , or just a part (possibly empty), and a public key  $\text{pk}$ ,  $\mathcal{V}$  possibly extracts the full message  $m$  and tests whether  $\sigma$  is a valid signature of  $m$  with respect to  $\text{pk}$ . In general, the verification algorithm need not be probabilistic.

**Forgeries and Attacks.** The simpler goal for an adversary is to build a new acceptable message-signature pair. This is called *existential forgery*. The corresponding security level is called *existential unforgeability* (EUF). On the other hand, the strongest scenario one usually considers is the so-called *adaptive chosen-message attack* (CMA), where the attacker can ask the signer to sign any message of its choice, in an adaptive way: it can adapt its queries according to previous answers. When signature generation is not deterministic, there may be several signatures corresponding to a given message. And then the notion of existential forgery may be ambiguous [26]: the original definition [10] says the adversary wins if it manages to forge a signature for a new message. Non-malleability [26] says the adversary wins if it manages to forge a new signature.

Thereafter, the security notion one wants to achieve is (at least) the resistance to existential forgeries under adaptive chosen-message attacks (EUF/CMA): one wants that the success probability of any adversary  $\mathcal{A}$  with a reasonable time is small, where

$$\text{Succ}_S^{\text{euf/cma}}(\mathcal{A}) = \Pr \left[ (\text{pk}, \text{sk}) \leftarrow \mathcal{K}(1^k), (m, \sigma) \leftarrow \mathcal{A}^{\mathcal{S}_{\text{sk}}}(\text{pk}) : \mathcal{V}(\text{pk}, m, \sigma) = 1 \right].$$

## 2.2 Public-Key Encryption

The aim of a public-key encryption scheme is to allow anybody who knows the public key of Alice to send her a message that she will be the only one able to recover, granted her private key.

**Definitions.** A public-key encryption scheme  $\mathbf{S} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$  is defined by the three following algorithms:

- The *key generation algorithm*  $\mathcal{K}$ . On input  $1^k$  where  $k$  is the security parameter, the algorithm  $\mathcal{K}$  produces a pair  $(\text{pk}, \text{sk})$  of matching public and private keys. Algorithm  $\mathcal{K}$  is probabilistic.
- The *encryption algorithm*  $\mathcal{E}$ . Given a message  $m$  and a public key  $\text{pk}$ ,  $\mathcal{E}$  produces a ciphertext  $c$  of  $m$ . This algorithm may be probabilistic. In the latter case, we write  $\mathcal{E}_{\text{pk}}(m; r)$  where  $r$  is the random input to  $\mathcal{E}$ .
- The *decryption algorithm*  $\mathcal{D}$ . Given a ciphertext  $c$  and the private key  $\text{sk}$ ,  $\mathcal{D}_{\text{sk}}(c)$  gives back the plaintext  $m$ . This algorithm is necessarily deterministic.

**Security Notions.** The most widely admitted goal of an adversary is the distinction of ciphertexts (IND). One thus wants to make it unable to distinguish between two messages, chosen by the adversary, which one has been encrypted, with a probability significantly better than one half. On the other hand, an attacker can play many kinds of attacks. The strongest scenario consists in giving a full access to the decryption oracle, which on any ciphertext answers the corresponding plaintext. There is of course the natural restriction not to ask the challenge ciphertext to that oracle. This scenario which allows adaptively chosen ciphertexts as queries to the decryption oracle is named the *chosen-ciphertext attack* (CCA). Therefore, for any adversary  $\mathcal{A}$ , seen as a 2-stage attacker  $(\mathcal{A}_1, \mathcal{A}_2)$ , its advantage  $\text{Adv}_S^{\text{ind/cca}}(\mathcal{A})$  should be negligible, where

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

## 2.3 Signature and Encryption

As already noticed, our motivation is to design a unified padding which one could use for both encryption and signature at the same time, and furthermore with the same asymmetric primitive. The goals of an adversary are thus the same as above: build an existential forgery (EUF) against the signature scheme, or distinguish ciphertexts (IND) against the encryption scheme. However, the means are the combination of the above attacks: it has access to both the signing oracle and the decryption oracle in a fully adaptive way, hence the CMA + CCA notation.

## 2.4 Claw-Free Permutations

In [13], Katz and Wang has shown that, by using trapdoor permutations induced by claw-free permutations, one can obtain a variant of FDH (just adding one more bit) with tight reduction. We can also use this technique for our construction. The existence of claw-free permutations seems be reasonable. In fact, any random self-reducible permutation can be seen as a trapdoor permutations induced by claw-free permutations [8] and almost all known examples of trapdoor permutations are self-reducible.

**Definition 1 (Claw-Free Permutations).** A family of claw-free permutations is a tuple of algorithms  $\{\text{Gen}; f_i; g_i | i \in I\}$  for an index set  $I$  such that:

- $\text{Gen}$  outputs a random index  $i$  and a trapdoor  $\text{td}$ .
- $f_i, g_i$  are both permutations over the same domain  $D_i$ .
- there is an efficient sampling algorithm which, on index  $i$ , outputs a random  $x \in D_i$ .
- $f_i^{-1}$  (the inverse of  $f_i$ ) and  $g_i^{-1}$  (the inverse of  $g_i$ ) are both efficiently computable given the trapdoor  $\text{td}$ .

A claw is a pair  $(x_0, x_1)$  such that  $f(x_0) = g(x_1)$ . Probabilistic algorithm  $\mathcal{A}$  is said to  $(t, \epsilon)$ -break a family of claw-free permutations if  $\mathcal{A}$  runs in time at most  $t$  and outputs a claw with probability greater than  $\epsilon$ :

$$\Pr \left[ (i, \text{td}) \leftarrow \text{Gen}(1^k), (x_0, x_1) \leftarrow \mathcal{A}(i) : f_i(x_0) = g_i(x_1) \right] \geq \epsilon$$

A family of claw-free permutations is  $(t, \epsilon)$ -secure if no algorithm can  $(t, \epsilon)$ -break it.

## 3 Optimal Permutation-based Padding

### 3.1 Our Optimal Proposal

In the following, we propose a universal padding, based on the construction from [21], in the random-permutation model. It is optimal both for signing and encrypting, *i.e.*, that uses only 82 bits of randomness for encrypting and only 82 bits of redundancy for signing. After the description, we show it is indeed secure, in the random-permutation model. In the next section, we provide another construction, based on the OAEP-3 rounds construction from the same paper [21], which is secure in the random-oracle model, but just near optimal (161 bits of overhead instead of 82).

The encryption and signature schemes use a permutation  $\mathcal{P}$ , that we assume to behave like a truly random permutation. Let  $k$  be a security parameter. Let  $\varphi_{\text{pk}} : \{0, 1\}^n \rightarrow \{0, 1\}^n$  be a trapdoor one-way permutation (whose inverse is called  $\psi_{\text{sk}}$ ). Messages to sign or to encrypt with our padding function will be of size  $\ell = n - k - 1$ . The symbol “ $\parallel$ ” denotes the bit-string concatenation and identifies  $\{0, 1\}^k \times \{0, 1\}^\ell \times \{0, 1\}$  to  $\{0, 1\}^n$ . Finally, in the following,  $\text{PRF}_\varrho()$  designs a PRF that uses a secret key  $\varrho$ .

**The Padding.** The padding is quite simple, since it takes as input a single bit  $\gamma$ , the message  $m$  and an additional data  $r$ , and  $\text{OPbP}(\gamma, m, r) = \mathcal{P}(\gamma \parallel m \parallel r) = t \parallel u$ . Thereafter, the reverse operation is natural:  $\text{OPbP}^{-1}(t, u) = \mathcal{P}^{-1}(t \parallel u) = \gamma \parallel m \parallel r$ .

**Encryption Algorithm.** The space of the plaintexts is  $\mathcal{M} = \{0, 1\}^\ell$ , the encryption algorithm uses a random coin from the set  $r \in \mathcal{R} = \{0, 1\}^k$ , a random bit  $\gamma$ , and outputs a ciphertext  $c$  into  $\{0, 1\}^n$ : on a plaintext  $m \in \mathcal{M}$ , one computes  $t \parallel u = \text{OPbP}(\gamma, m, r)$  and  $c = \varphi_{\text{pk}}(t \parallel u)$ .

**Decryption Algorithm.** On a ciphertext  $c$ , one first computes  $t \parallel u = \psi_{\text{sk}}(c)$ , where  $t \in \{0, 1\}^k$  and  $u \in \{0, 1\}^{\ell+1}$ , and then  $\gamma \parallel m \parallel r = \text{OPbP}^{-1}(t, u)$ . The answer is  $m$ .

**Signature Algorithm.** The space of the messages is  $\mathcal{M} = \{0, 1\}^\ell$ , the signature algorithm outputs a signature  $\sigma$  into  $\{0, 1\}^n$ : on a message  $m \in \mathcal{M}$ , one computes  $\gamma = \text{PRF}_\varrho(m)$ , and then  $t \parallel u = \text{OPbP}(\gamma, m, 0^k)$  and  $\sigma = \psi_{\text{sk}}(t \parallel u)$ .

**Verification Algorithm.** On a signature  $\sigma$ , one first computes  $t||u = \varphi_{\text{pk}}(\sigma)$ , where  $t \in \{0,1\}^k$  and  $u \in \{0,1\}^{\ell+1}$ , and then  $\gamma||m||r = \text{OPbP}^{-1}(t, u)$ . If  $r = 0^k$ , the verification outputs “Correct” and recovers  $m$ , otherwise outputs “Incorrect”.

### 3.2 Security Analysis

A variant of this padding has already been proved to lead to an IND/CCA secure encryption scheme [21], and to a EUF/CMA signature scheme [11], in the random-permutation model. However, there was not the additional bit of Katz and Wang, that just makes more randomness in the encryption. Here, we extend these results to IND/CMA + CCA and EUF/CMA + CCA:

**Theorem 2.** *Let  $\mathcal{A}$  and  $\mathcal{B}$  be both chosen-ciphertext (to the decryption oracle) and chosen-message (to the signing oracle) adversaries, against the encryption scheme (IND) and the signature scheme (EUF) respectively. Let us assume that  $\mathcal{A}$  can break the semantic security with an advantage  $\varepsilon_E$ , or  $\mathcal{B}$  can produce an existential forgery with success probability  $\varepsilon_S$  (within a time bound  $t$ , after  $q_p$ ,  $q_s$ ,  $q_d$  queries to the permutation oracles, signing oracle and decryption oracle respectively.) Then the permutation  $\varphi_{\text{pk}}$  can be inverted with probability  $\varepsilon'$  within time  $t'$  where either:*

$$\begin{aligned} \varepsilon' &\geq \varepsilon_E - \frac{(q_p + q_d + q_s + 1)^2}{2^{k+\ell+1}} - \frac{(q_d + 1)^2}{2^\ell} - \frac{2q_p + q_d + q_s + 2}{2^k}, \text{ or} \\ \varepsilon' &\geq \frac{1}{q_p + q_s + 1} \cdot \left( \varepsilon_S - \frac{(q_p + q_d + q_s + 1)^2}{2^{k+\ell+1}} - \frac{(q_d + 1)^2}{2^\ell} - \frac{2q_p + q_d + q_s + 2}{2^k} \right). \end{aligned}$$

Particularly, if the function  $\varphi_{\text{pk}}$  is induced by a  $(t', \varepsilon')$ -secure claw-free permutation, the latter can be rewritten by:

$$\varepsilon' \geq \frac{1}{2} \left( \varepsilon_S - \frac{(q_p + q_d + q_s + 1)^2}{2^{k+\ell+1}} - \frac{(q_d + 1)^2}{2^\ell} - \frac{2q_p + q_d + q_s + 2}{2^k} \right)$$

where  $t' \leq t + (q_p + q_d + q_s + 1)T_f$ , and  $T_f$  is the time for an evaluation of  $\varphi_{\text{pk}}$ .

*Proof.* We provide now the proof of this theorem, with incremental games, to reduce the inversion of the permutation  $\varphi_{\text{pk}}$  on a random instance  $y$  (i.e., find  $x$  such that  $y = \varphi_{\text{pk}}(x)$ ) to an attack against either the encryption or the signature. We show that either  $\mathcal{A}$  or  $\mathcal{B}$  can help us to invert  $\varphi_{\text{pk}}$ .

Some parts of this proof are similar to [21]. We anyway provide the proof without the similar parts.

**GAME  $\mathbf{G}_0$ :** This is the attack game, in the random-permutation model. Several oracles are thus available to the adversary: two random permutation oracles ( $\mathcal{P}$  and  $\mathcal{P}^{-1}$ ), the signing oracle  $\mathcal{S}_{\text{sk}}$ , and the decryption oracle  $\mathcal{D}_{\text{sk}}$ .

To break the encryption, the adversary  $\mathcal{A} = (A_1, A_2)$  runs its attack in two steps. First,  $A_1$  is given the public key  $\text{pk}$ , and outputs a pair of messages  $(m_0, m_1)$ . Next a challenge ciphertext is produced by the challenger, which flips a coin  $b$  and computes a ciphertext  $c^*$  of  $m^* = m_b$ . This ciphertext comes from a random  $r^* \xleftarrow{R} \{0,1\}^k$ , a bit  $\gamma^*$  and  $c^* = \mathcal{E}(\gamma^*, m_b, r^*) = \varphi_{\text{pk}}(\mathcal{P}(\gamma^*, m_b, r^*))$ . In the second step, on input  $c^*$ ,  $A_2$  outputs a bit  $b'$ . We denote by  $\text{Dist}_0$  the event  $b' = b$  and use the same notation  $\text{Dist}_n$  in any game  $\mathbf{G}_n$ .

To break the signature, the adversary  $\mathcal{B}$  outputs its forgery, one checks whether it is actually valid or not. We denote by  $\text{Forge}_0$  the event this forged signature is valid and use the same notation  $\text{Forge}_n$  in any game  $\mathbf{G}_n$ .

Note that the adversary is given access to the signing oracle  $\mathcal{S}_{\text{sk}}$  and the decryption oracle  $\mathcal{D}_{\text{sk}}$  at any time during the attack. Note also that if the adversary asks  $q_d$  queries to the decryption oracle,  $q_s$  queries to the signing oracle and  $q_p$  queries to the permutation oracles, at most  $q_d + q_s + q_p + 1$  queries are asked to the permutation oracles during this game, since each decryption query or signing query may

make such a new query, and the last verification step or the challenger step does too. By definition,

$$\begin{aligned}\varepsilon_E &= \text{Adv}_{\text{OPbP}}^{\text{ind/cma+cca}}(\mathcal{A}) = \Pr[\text{Dist}_0] - 1/2. \\ \varepsilon_S &= \text{Succ}_{\text{OPbP}}^{\text{euf/cma+cca}}(\mathcal{B}) = \Pr[\text{Forge}_0].\end{aligned}$$

We skip the easy steps, similar to [21] for the encryption part, and to [4] for the signature. Details can be found in the appendix A, which leads to the simulation in the game  $\mathbf{G}_8$  presented in Figure 1, which is statistically indistinguishable from the initial one since the distance is bounded by:

$$\Delta_G \leq \frac{(q_p + q_d + q_s + 1)^2}{2^{k+\ell+1}} + \frac{(q_d + 1)^2}{2^\ell} + \frac{2q_p + q_d + q_s + 2}{2^k}.$$

In the following, depending on the goal of the adversary, namely against encryption or against signature, we complete the reduction to the inversion of the function  $\psi_{\text{sk}}$  on the given instance  $y$ .

### Encryption attack.

GAME  $\mathbf{G}_{8.1}$ : We suppress the element  $(\gamma^*, m^*, r^*, \perp, \perp, c^*)$  from P-List during the generation of the challenge.

►Rule ChalAdd<sup>(8.1)</sup>

| Do nothing.

The two games  $\mathbf{G}_{8.1}$  and  $\mathbf{G}_8$  are perfectly indistinguishable unless  $(\gamma^*, m^*, r^*)$  is asked for  $\mathcal{P}$  (which event is included in event  $\text{BadP}_{8.1}$ , already excluded) or  $p^* = \psi_{\text{sk}}(c^*)$  is asked to  $\mathcal{P}^{-1}$ . We define the latter event  $\text{AskInvP}_{8.1}$ . We have:  $\Delta_{8.1} \leq \Pr[\text{AskInvP}_{8.1}]$ . Since  $(\gamma^*, m^*, r^*, \perp, \perp, c^*)$  does not appear in P-List, the adversary receives answers which are perfectly independent of the latter, and therefore, it has no advantage for guessing  $b$ :

$$\Pr[\text{Dist}_{8.1}] = \frac{1}{2}.$$

GAME  $\mathbf{G}_{8.2}$ : Instead of choosing  $c^* = \varphi_{\text{pk}}(p^*)$ , we choose  $c^* = y$ , uniformly at random.

►Rule Chal<sup>(8.2)</sup>

|  $c^* = y$ .

So, one implicitly defines  $p^* = \psi_{\text{sk}}(y)$ . Since the tuple  $(\gamma^*, m^*, r^*, \perp, \perp, c^*)$  is not used anywhere in the simulation, the two games  $\mathbf{G}_{8.2}$  and  $\mathbf{G}_{8.1}$  are perfectly indistinguishable:  $\Delta_{8.2} = 0$ .

Finally, it is clear that when the event  $\text{AskInvP}_{8.2}$  happens, one can easily compute  $\psi_{\text{sk}}$  on  $y$ : with a look up into P-List (which contains at most  $q_p + q_d + q_s + 1$  elements), one can extract  $p$  such that  $y = \varphi_{\text{pk}}(p)$ . Therefore,  $\Pr[\text{AskInvP}_{8.2}] \leq \text{Succ}_{\varphi}^{\text{ow}}(t')$ , where  $T_{\varphi}$  is the time for evaluating  $\varphi_{\text{pk}}$ , and  $t' \leq t + (q_p + q_d + q_s + 1) \times T_{\varphi}$  is the running time of the simulation in the current game. This completes the first part of the proof.

### Signature attack (general case).

GAME  $\mathbf{G}_{8.1}$ : In the following, we number calls to the permutation oracle, but only those which are of the form  $(\gamma, \star, 0^k)$ , which are those that are used for signature. We define a variable  $\nu$  which is initialized to 0.

►Rule EvalP<sup>(8.1)</sup>

$\mathcal{P}$ -Oracle	<p>A query <math>\mathcal{P}(\gamma, m, r)</math> is answered by <math>p</math>, where</p> <p>► <b>Rule EvalP<sup>(8)</sup></b></p> <ul style="list-style-type: none"> <li>– Look for <math>(\gamma, m, r, \alpha, \beta, c)</math> in P-List: <ul style="list-style-type: none"> <li>• if the record is found, <ul style="list-style-type: none"> <li>* if <math>\alpha \neq \perp</math>, <math>p = \alpha</math>;</li> <li>* otherwise, <b>Stop</b>.</li> </ul> </li> <li>• otherwise, choose a random element <math>s \in \{0, 1\}^n</math> and computes <math>p = \varphi_{pk}(s)</math>. The record <math>(\gamma, m, r, p, s, \varphi_{pk}(p))</math> is added to P-List.</li> </ul> </li> </ul> <p>Furthermore, if <math>(\gamma, m, r)</math> is a direct query from the adversary to <math>\mathcal{P}</math>, store the record <math>(\gamma, m, r, p, \perp, \varphi_{pk}(p))</math> in P-List.</p>
$\mathcal{P}^{-1}$ -Oracle	<p>A query <math>\mathcal{P}^{-1}(p)</math> is answered by <math>(\gamma, m, r)</math>, where</p> <p>► <b>Rule InvP<sup>(8)</sup></b></p> <ul style="list-style-type: none"> <li>– Compute <math>c = \varphi_{pk}(p)</math> and look for <math>(\gamma, m, r, \alpha, \beta, c)</math> in P-List: <ul style="list-style-type: none"> <li>– if the record is found, <math>(\gamma, m, r)</math> is defined,</li> <li>– otherwise we randomly choose <math>(\gamma, m, r)</math> in <math>\{0, 1\}^n</math>. If <math>r = 0^k</math>, <b>Stop</b>.</li> </ul> </li> </ul> <p>Furthermore, if <math>p</math> a direct query from the adversary to <math>\mathcal{P}^{-1}</math>, store the record <math>(\gamma, m, r, p, \perp, \varphi_{pk}(p))</math> in P-List.</p>
$\mathcal{D}$ -Oracle	<p>A query <math>\mathcal{D}_{sk}(c)</math> is answered by <math>m</math>, where</p> <p>► <b>Rule D<sup>(8)</sup></b></p> <ul style="list-style-type: none"> <li>– Look for <math>(\gamma, m, r, \alpha, \beta, c)</math> in P-List: <ol style="list-style-type: none"> <li>1. if the record is found, <math>(\gamma, m, r)</math> is defined,</li> <li>2. otherwise we randomly choose <math>(\gamma, m, r)</math> in <math>\{0, 1\}^n</math>.</li> </ol> </li> </ul> <p>Store <math>(\gamma, m, r, \perp, \perp, c)</math> in P-List.</p>
$\mathcal{S}$ -Oracle	<p>For a sign-query <math>\mathcal{S}_{sk}(m)</math>, one first computes <math>\gamma = \text{PRF}_e(m)</math>, then asks for <math>p = \mathcal{P}(\gamma, m, 0^k)</math> to the EvalP-oracle. The signature <math>\sigma</math> is then defined according to the following rule:</p> <p>► <b>Rule S<sup>(8)</sup></b></p> <ul style="list-style-type: none"> <li>– Look for <math>(\gamma, m, 0^k, p, s, c)</math> in P-List, and set <math>\sigma = s</math>.</li> </ul>
Challenger	<p>For two messages <math>(m_0, m_1)</math>, flip coins <math>\gamma^*</math> and <math>b</math>, set <math>m^* = m_b</math>, and randomly choose <math>r^*</math>.</p> <p>► <b>Rule Chal<sup>(8)</sup></b></p> <ul style="list-style-type: none"> <li>– <math>p^* = \mathcal{P}(\gamma^*, m^*, r^*)</math>; <math>c^* = \varphi_{pk}(p^*)</math>.</li> </ul> <p>► <b>Rule ChalAdd<sup>(8)</sup></b></p> <ul style="list-style-type: none"> <li>– Add <math>(\gamma^*, m^*, r^*, \perp, \perp, c^*)</math> in P-List.</li> </ul> <p>Answer <math>c^*</math></p>
$\mathcal{V}$ -Oracle	<p>The game ends with the verification of the output <math>(\sigma)</math> from the adversary. One first computes <math>t \  u = \varphi_{pk}(\sigma)</math>, then asks for <math>(\gamma, m, r) = \mathcal{P}^{-1}(t \  u)</math>. Then he checks whether <math>r = 0^k</math>, in which case the signature is a valid signature of <math>m</math>.</p>

**Fig. 1.** Simulation in the Game  $\mathbf{G}_8$

Look for  $(\gamma, m, r, \alpha, \beta, c)$  in P-List:

- if the record is found,
  - if  $\alpha \neq \perp$ ,  $p = \alpha$ ;
  - otherwise, **Stop**.
- otherwise,
  - if  $r = 0^k$ , increment  $\nu$
  - choose a random element  $s \in \{0, 1\}^n$  and computes  $p = \varphi_{\text{pk}}(s)$ . The record  $(\gamma, m, r, p, s, \varphi_{\text{pk}}(p))$  is added to P-List.

Clearly, this leaves the game indistinguishable from the game  $\mathbf{G}_8$ :  $\Delta_{8,1} = 0$ .

GAME  $\mathbf{G}_{8,2}$ : Since the verification process is included in the attack game, the output message is necessarily asked to the permutation oracle **EvalP**. Let us guess the index  $\nu_0$  of this (first) query. If the guess failed, we abort the game. Therefore, only a correct guess (event **GoodGuess**) may lead to a success.

$$\begin{aligned} \Pr[\text{Forge}_{8,2}] &= \Pr[\text{Forge}_{8,1} \wedge \text{GoodGuess}] = \Pr[\text{Forge}_{8,1} \mid \text{GoodGuess}] \times \Pr[\text{GoodGuess}] \\ &\geq \Pr[\text{Forge}_{8,1}] \times \frac{1}{q_p + q_s + 1}. \end{aligned}$$

GAME  $\mathbf{G}_{8,3}$ : We now incorporate the challenge  $y$  to the simulation of the permutation oracle. By this, we could extract the pre-image  $x$ . Our idea is to return  $y$  as the value of the guessed  $\nu$ -th query:

► **Rule EvalP<sup>(8.3)</sup>**

Look for  $(\gamma, m, r, \alpha, \beta, c)$  in P-List:

- if the record is found,
  - if  $\alpha \neq \perp$ ,  $p = \alpha$ ;
  - otherwise, **Stop**.
- otherwise,
  - if  $r = 0^k$ , increment  $\nu$
  - if  $\nu \neq \nu_0$  or if  $r \neq 0^k$ , choose a random element  $s \in \{0, 1\}^n$  and computes  $p = \varphi_{\text{pk}}(s)$ .
  - if  $\nu = \nu_0$  and  $r = 0^k$ , sets  $p = y$ .
  - The record  $(\gamma, m, r, y, s, \varphi_{\text{pk}}(p))$  is added to P-List.

Because of the random choice for the challenge  $y$ , this rule leaves the game indistinguishable from the previous one:  $\Delta_{8,3} = 0$ . It follows that the forgery leads to the pre-image of  $y$ :  $\Pr[\text{Forge}_{8,3}] = \text{Succ}_{\varphi}^{\text{ow}}(t + (q_p + q_d + q_s + 1)T_{\varphi})$ . This concludes the second part of the proof.

**Signature Attack (With  $(t', \varepsilon')$ -Secure Claw-Free Permutations).** We assume that  $(\varphi_{\text{pk}}, \lambda_{\text{pk}})$  are from a  $(t', \varepsilon')$ -secure claw-free permutations family.

GAME  $\mathbf{G}_{8,1}$ : We now exploit the bit  $\gamma$  to the simulation of the permutation oracle, as it was proposed firstly by Katz and Wang [13]. The idea is to use  $\varphi_{\text{pk}}$  in the **OPbP** output, for one and only one value of bit  $\gamma$ , and otherwise use  $\lambda_{\text{pk}}$ . As this value of  $\gamma$  is not predictable by the attacker, its forgery will, with a probability  $\frac{1}{2}$ , produce a claw.

► **Rule EvalP<sup>(8.1)</sup>**



Look for  $(\gamma, m, r, \alpha, \beta, c)$  in P-List:

- if the record is found,
  - if  $\alpha \neq \perp$ ,  $p = \alpha$ ;
  - otherwise, **Stop**.
- otherwise,
  - if  $r \neq 0^k$  or  $\gamma = \text{PRF}_\rho(m)$ , choose a random element  $s \in \{0, 1\}^n$  and compute  $p = \varphi_{\text{pk}}(s)$ .
  - if  $r = 0^k$  or  $\gamma \neq \text{PRF}_\rho(m)$ , choose a random element  $s \in \{0, 1\}^n$  and compute  $p = \lambda_{\text{pk}}(s)$ .
  - The record  $(m, r, p, s, \varphi_{\text{pk}}(p))$  is added to P-List.

Because of the random choice of  $s$  and so  $\lambda_{\text{pk}}(s)$ , this rule leaves the game indistinguishable from the previous one:  $\Delta_{8.1} = 0$ .

Using arguments as in [13], one can easily see that the forgery leads to a claw with probability  $\frac{1}{2}$ . In fact, let us assume that the adversary can forge a signature  $(\tilde{m}, \tilde{\sigma})$ , where  $(\tilde{m}, 0^k)$  has been asked to the permutation oracle  $\mathcal{P}$  either in a permutation query or in the verification step. Since the bit  $b_{\tilde{m}} = \text{PRF}_\rho(\tilde{m})$  is an unknown random bit in the view of the adversary, with probability of  $\frac{1}{2}$ , there exists an element  $(\tilde{m}, \tilde{r}, \tilde{p} = \lambda_{\text{pk}}(\tilde{s}), \tilde{s}, \varphi_{\text{pk}}(\tilde{p}))$  in the P-List. In that case, the simulator can output a claw  $\varphi_{\text{pk}}(\tilde{\sigma}) = \lambda_{\text{pk}}(\tilde{s})$ .

□

### 3.3 Proposed Sizes for the Parameters

We say that a scheme achieves a security level of  $2^\kappa$ , if the ratio between the running time  $t$  of the adversary, and its success probability  $\varepsilon$ , is at least  $2^\kappa$ : this is an approximation of the expected time of success. Or similarly, we want  $t/\varepsilon \leq 2^{-\kappa}$ , with a usual security bound set with  $\kappa = 80$ .

First, we can simplify the above security result. Indeed, for practical purpose, where  $\ell$  is the bit-size of the message, and  $k$  is the bit-size of the random/redundancy, the former is expected to be much larger than the latter: the quantity  $Q/2^\ell$ , or even  $Q^2/2^\ell$ , can be ignored in front of  $Q/2^k$  (since  $Q$ , the global number of queries is bounded by  $2^{80}$ ). Therefore, the above reduction cost provides that

$$\begin{aligned}
 \frac{\varepsilon_E}{t} &\leq \frac{\varepsilon'}{t} + \frac{2}{2^k} && \text{and} \\
 \frac{\varepsilon_S}{t} &\leq \frac{Q\varepsilon'}{t} + \frac{2}{2^k} && \text{in the general case} \\
 &\leq \frac{2\varepsilon'}{t} + \frac{2}{2^k} && \text{if the function } \varphi_{\text{pk}} \text{ is induced by a claw-free permutation}
 \end{aligned}$$

In the latter case (the most interesting case, where one uses RSA) we can assume the message length sufficiently large (and thus the RSA modulus) so that  $\varepsilon'/t$  is lower than  $2^{-82}$ . Due to the Lenstra-Verheul's estimation [15], for the case of RSA, we can use a 1024-bit modulus.

In the general case, we have to consider that the security parameter (and thus message length  $\ell$ ) large enough such that the ration between  $\varepsilon'/t$  is lower than  $2^{-161}$ . But then the overhead  $k = 82$  is enough too.

As a conclusion, for the general case, we can choose  $k = 82$  if the security level of the function  $\varphi$  is about  $2^{161}$ . For the particular case of RSA, we can use a 1024-bit modulus. We remark then that, with only 82 bits of redundancy, we obtain the same level of security than RSA-PSS [3], which, compared to our scheme, uses a lowest bandwidth. For the encryption security, we find again the result from [21]: 82 bits of randomness are enough to achieve semantic security, even under chosen-ciphertext and chosen-message attacks.

## 4 The OAEP-3 Rounds Construction

### 4.1 Description

In order to work in the more usual random-oracle model [2], we now consider the OAEP-3 rounds construction proposed in [21, 22]. As above, the security of this padding has already been studied for encryption, but without giving access to the signing oracle to the adversary. We thus extend the security model to deal with the two oracles access.

The encryption and signature schemes use three hash functions:  $\mathcal{F}$ ,  $\mathcal{G}$ ,  $\mathcal{H}$  (assumed to behave like random oracles in the security analysis) where the security parameters satisfy  $n = k + \ell + 1$ :

$$\mathcal{F} : \{0, 1\}^k \rightarrow \{0, 1\}^{\ell+1} \quad \mathcal{G} : \{0, 1\}^{\ell+1} \rightarrow \{0, 1\}^k \quad \mathcal{H} : \{0, 1\}^k \rightarrow \{0, 1\}^{\ell+1}.$$

The encryption and signature schemes use any permutation family  $(\varphi_{\text{pk}})_{\text{pk}}$  on the space  $\{0, 1\}^n$ , whose inverses are respectively denoted  $\psi_{\text{sk}}$ , where  $\text{sk}$  is the private key associated to the public key  $\text{pk}$ . The symbol “ $\parallel$ ” denotes the bit-string concatenation and identifies  $\{0, 1\}^k \times \{0, 1\}^\ell \times \{0, 1\}$  to  $\{0, 1\}^n$ .

### Padding OAEP3r and Unpadding OAEP3r<sup>-1</sup>

$$\begin{aligned} \text{OAEP3r}(\gamma, m, r) : s &= (\gamma \parallel m) \oplus \mathcal{F}(r) \quad t = r \oplus \mathcal{G}(s) \quad u = s \oplus \mathcal{H}(t) \\ \text{OAEP3r}(\gamma, m, r) &= t \parallel u \\ \text{OAEP3r}^{-1}(t, u) : s &= u \oplus \mathcal{H}(t) \quad r = t \oplus \mathcal{G}(s) \quad \gamma \parallel m = s \oplus \mathcal{F}(r) \\ \text{OAEP3r}^{-1}(t, u) &= \gamma \parallel m \parallel r \end{aligned}$$

**Encryption Algorithm.** The space of the plaintexts is  $\mathcal{M} = \{0, 1\}^\ell$ , the encryption algorithm uses a random coin from the set  $r \in \mathcal{R} = \{0, 1\}^k$ , a random bit  $\gamma$  and outputs a ciphertext  $c$  into  $\{0, 1\}^n$ : on a plaintext  $m \in \mathcal{M}$ , one computes  $t \parallel u = \text{OAEP3r}(\gamma, m, r)$  and  $c = \varphi_{\text{pk}}(t \parallel u)$ .

**Decryption Algorithm.** On a ciphertext  $c$ , one first computes  $t \parallel u = \psi_{\text{sk}}(c)$ , where  $t \in \{0, 1\}^k$  and  $u \in \{0, 1\}^{\ell+1}$ , and then  $\gamma \parallel m \parallel r = \text{OAEP3r}^{-1}(t, u)$ . The answer is  $m$ .

**Signature Algorithm.** The space of the plaintexts is  $\mathcal{M} = \{0, 1\}^\ell$ , the signature algorithm outputs a signature  $\sigma$  into  $\{0, 1\}^n$ : on a plaintext  $m \in \mathcal{M}$ , one computes  $\gamma = \text{PRF}_\varrho(m)$ , then computes  $t \parallel u = \text{OAEP3r}(\gamma, m, 0^k)$  and  $\sigma = \psi_{\text{sk}}(t \parallel u)$ .

**Verification Algorithm.** On a signature  $\sigma$ , one first computes  $t \parallel u = \varphi_{\text{pk}}(\sigma)$ , where  $t \in \{0, 1\}^k$  and  $u \in \{0, 1\}^{\ell+1}$ , and then  $\gamma \parallel m \parallel r = \text{OAEP3r}^{-1}(t, u)$ . If  $r = 0^k$ , the verification outputs “Correct” then recovers  $m$ , otherwise outputs “Incorrect”

### 4.2 Security Result

We extend the security result from [22] by the following theorem:

**Theorem 3.** *Let  $\mathcal{A}$  and  $\mathcal{B}$  be both chosen-ciphertext (to the decryption oracle) and chosen-message (to the signing oracle) adversaries, against the encryption scheme (IND) and the signature scheme (EUF) respectively. Let us assume that  $\mathcal{A}$  can break the semantic security with the advantage  $\varepsilon_E$ , or  $\mathcal{B}$  can produce an existential forgery with success probability  $\varepsilon_S$  (within a time bound  $t$ , after  $q_f$ ,  $q_g$ ,  $q_h$ ,  $q_s$ ,  $q_d$  queries to the oracles  $\mathcal{F}$ ,  $\mathcal{G}$ ,  $\mathcal{H}$ , signing oracle and decryption oracle respectively.) Then the permutation  $\varphi_{\text{pk}}$  can be inverted with probability  $\varepsilon'$  within time  $t'$  where either:*

$$\begin{aligned}\varepsilon' &\geq \varepsilon_E - \left( q_d^2 \times \left( \frac{1}{2^{\ell+1}} + \frac{6}{2^k} \right) + \frac{4q_dq_g + q_g}{2^{\ell+1}} + \frac{5q_dq_f + q_gq_h + q_f + q_d}{2^k} \right) \text{ or} \\ \varepsilon' &\geq \frac{1}{q_g + q_s + 1} \times \\ &\quad \left( \varepsilon_S - \left( q_d^2 \times \left( \frac{1}{2^{\ell+1}} + \frac{6}{2^k} \right) + \frac{4q_dq_g + q_g}{2^{\ell+1}} + \frac{5q_dq_f + q_gq_h + q_f + q_d}{2^k} \right) \right)\end{aligned}$$

Particularly, if the function  $\varphi_{\text{pk}}$  is induced by a  $(t', \varepsilon')$ -secure claw-free permutation, the latter can be rewritten by:

$$\varepsilon' \geq \frac{1}{2} \times \left( \varepsilon_S - \left( q_d^2 \times \left( \frac{1}{2^{\ell+1}} + \frac{6}{2^k} \right) + \frac{4q_dq_g + q_g}{2^{\ell+1}} + \frac{5q_dq_f + q_gq_h + q_f + q_d}{2^k} \right) \right)$$

with  $t' \leq t + (q_f + q_g + q_h + q_d)T_{lu} + q_d^2T_{lu} + (q_d + 1)q_gq_h(T_\varphi + T_{lu})$ , where  $T_\varphi$  is the time complexity for evaluating any function  $\varphi_{\text{pk}}$ , and  $T_{lu}$  is the time complexity for a look up in a list.

*Proof.* The full proof can be found in the appendix B. The simulation of the oracles as well as the simulation of the decryption are similar to the ones in [22]. The simulation of the signature (after all the oracles are well simulated) is quite the same as in the random-permutation model case.  $\square$

### 4.3 Proposed Sizes for the Parameters

Using similar arguments as in the previous construction, one can simplify the constraints on the security parameters:

- For encryption, one has:

$$\frac{\varepsilon_E}{t} \leq \frac{\varepsilon'}{t} + \frac{Q}{2^k}.$$

Then,  $k = 161$  is enough if the security parameters are large enough (*i.e.*, as soon as  $\varepsilon'/t < 2^{-81}$ ).

- For signature, in the general case:

$$\frac{\varepsilon_S}{t} \leq \frac{Q\varepsilon'}{t} + \frac{Q}{2^k}.$$

In the general case,  $k = 161$  is also valid, as soon as  $\varepsilon'/t < 2^{-161}$ .

- For signature, in case the function  $\varphi_{\text{pk}}$  is induced by a claw-free permutation:

$$\frac{\varepsilon_S}{t} \leq \frac{2\varepsilon'}{t} + \frac{Q}{2^k}.$$

We have a similar expression as in the above encryption case (the term  $\varepsilon'/t$  is replaced by  $2\varepsilon'/t$ , which allows shorter security parameters. Anyway,  $k = 161$  is required, as soon as  $\varepsilon'/t < 2^{-82}$ ).

To sum up, for the interesting case of the RSA, one can choose  $k = 161$ , with a security parameter chosen so that the security level of the function  $\varphi$  is about  $2^{82}$ , that is 1024-bit modulus.

### Acknowledgement

The work described in this paper has been supported in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

## References

1. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among Notions of Security for Public-Key Encryption Schemes. In *Crypto '98*, LNCS 1462, pages 26–45. Springer-Verlag, Berlin, 1998.
2. M. Bellare and P. Rogaway. Random Oracles Are Practical: a Paradigm for Designing Efficient Protocols. In *Proc. of the 1st CCS*, pages 62–73. ACM Press, New York, 1993.
3. M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. In *Eurocrypt '94*, LNCS 950, pages 92–111. Springer-Verlag, Berlin, 1995.
4. M. Bellare and P. Rogaway. The Exact Security of Digital Signatures – How to Sign with RSA and Rabin. In *Eurocrypt '96*, LNCS 1070, pages 399–416. Springer-Verlag, Berlin, 1996.
5. J.-S. Coron, M. Joye, D. Naccache, and P. Paillier. Universal Padding Schemes For RSA. In M. Yung, editor, *Advances in Cryptology – CRYPTO '02*, volume 2442 of *Lecture Notes in Computer Science*, pages 226–241. Springer-Verlag, Berlin, 2002.
6. R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack. In *Crypto '98*, LNCS 1462, pages 13–25. Springer-Verlag, Berlin, 1998.
7. R. Cramer and V. Shoup. Signature Scheme based on the Strong RSA Assumption. In *Proc. of the 6th CCS*, pages 46–51. ACM Press, New York, 1999.
8. Y. Dodis and L. Reyzin. On the power of claw-free permutation. In *Security in Communication Networks*, 2002.
9. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is Secure under the RSA Assumption. In *Crypto '01*, LNCS 2139, pages 260–274. Springer-Verlag, Berlin, 2001.
10. S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal of Computing*, 17(2):281–308, April 1988.
11. L. Granboulan. Short Signatures in the Random Oracle Model. In *Asiacrypt '02*, LNCS 2501, pages 364–378. Springer-Verlag, Berlin, 2002.
12. S. Haber and B. Pinkas. Combining Public Key Cryptosystems. In *Proc. of the 8th ACM CSS*, pages 215–224. ACM Press, New York, 2001.
13. J. Katz and N. Wang. Efficiency improvements for signature schemes with tight security reductions. In *Proc. of the 10th CCS*, pages 155–164. ACM Press, Washington, 2003.
14. Y. Komano and K. Ohta. Efficient Universal Padding Schemes for Multiplicative Trapdoor One-Way Permutation. In D. Boneh, editor, *Advances in Cryptology – CRYPTO '03*, volume 2729 of *Lecture Notes in Computer Science*, pages 366–382. Springer-Verlag, Berlin, 2003.
15. A. Lenstra and E. Verheul. Selecting Cryptographic Key Sizes. In *PKC '00*, LNCS 1751, pages 446–465. Springer-Verlag, Berlin, 2000.
16. D. Naccache and J. Stern. Signing on a Postcard. In *Financial Cryptography '00*, LNCS 1962. Springer-Verlag, Berlin, 2001.
17. NIST. Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186, November 1994.
18. K. Nyberg and R. A. Rueppel. Message Recovery for Signature Schemes Based on the Discrete Logarithm Problem. In *Eurocrypt '94*, LNCS 950, pages 182–193. Springer-Verlag, Berlin, 1995.
19. T. Okamoto and D. Pointcheval. The Gap-Problems: a New Class of Problems for the Security of Cryptographic Schemes. In *PKC '01*, LNCS 1992. Springer-Verlag, Berlin, 2001.
20. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt '99*, LNCS 1592, pages 223–238. Springer-Verlag, Berlin, 1999.
21. D. H. Phan and D. Pointcheval. Chosen-Ciphertext Security without Redundancy. In *Asiacrypt '03*, LNCS 2894, pages 1–18. Springer-Verlag, Berlin, 2003.
22. D. H. Phan and D. Pointcheval. OAEP 3-Round: A Generic and Secure Asymmetric Encryption Padding. In *Asiacrypt '04*, LNCS 3329, pages 63–77. Springer-Verlag, Berlin, 2004.
23. C. Rackoff and D. R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In *Crypto '91*, LNCS 576, pages 433–444. Springer-Verlag, Berlin, 1992.
24. R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
25. C. P. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.
26. J. Stern, D. Pointcheval, J. Malone-Lee, and N. Smart. Flaws in Applying Proof Methodologies to Signature Schemes. In *Crypto '02*, LNCS 2442, pages 93–110. Springer-Verlag, Berlin, 2002.

## A Security Proof of the Optimal Permutation-based Padding

**GAME  $\mathbf{G}_1$ :** A perfect simulation of the real attack game is described on the Figure 2. Actually, the rule  $\text{Chal}^{(1)}$  simulates the way the challenge  $c^*$  is generated, exactly as the challenger would do. Besides, we simulate the two random permutation oracles  $\mathcal{P}$  and  $\mathcal{P}^{-1}$ , the decryption oracle  $\mathcal{D}_{\text{sk}}$  and the signature oracle  $\mathcal{S}_{\text{sk}}$ , by maintaining a list P-List, using a truly random permutation  $P$  and its inverse  $P^{-1}$ , and finally a PRF  $\text{PRF}_\varrho$  for a random secret key  $\varrho$ .

$\mathcal{P}$ -Oracle	<p>A query <math>\mathcal{P}(\gamma, m, r)</math> is answered by <math>p</math>, where</p> <p>► <b>Rule EvalP<sup>(1)</sup></b>  <math>  p = P(\gamma, m, r)</math>.</p> <p>Furthermore, if <math>(\gamma, m, r)</math> is a direct query from the adversary to <math>\mathcal{P}</math>, store the record <math>(\gamma, m, r, p, \perp, \varphi_{pk}(p))</math> in P-List.</p>
$\mathcal{P}^{-1}$ -Oracle	<p>A query <math>\mathcal{P}^{-1}(p)</math> is answered by <math>(\gamma, m, r)</math>, where</p> <p>► <b>Rule InvP<sup>(1)</sup></b>  <math>  (\gamma, m, r) = P^{-1}(p)</math>.</p> <p>Furthermore, if <math>p</math> a direct query from the adversary to <math>\mathcal{P}^{-1}</math>, store the record <math>(\gamma, m, r, p, \perp, \varphi_{pk}(p))</math> in P-List.</p>
$\mathcal{D}$ -Oracle	<p>A query <math>\mathcal{D}_{sk}(c)</math> is answered by <math>m</math>, where</p> <p>► <b>Rule D<sup>(1)</sup></b>  <math>  p = \psi_{sk}(c)</math>, and <math>(\gamma, m, r) = \mathcal{P}^{-1}(p)</math>.</p> <p>Store <math>(\gamma, m, r, \perp, \perp, c)</math> in P-List.</p>
$\mathcal{S}$ -Oracle	<p>For a sign-query <math>\mathcal{S}_{sk}(m)</math>, one first computes <math>\gamma = \text{PRF}_g(m)</math>, then asks for <math>p = \mathcal{P}(\gamma, m, 0^k)</math> to the EvalP-oracle. The signature <math>\sigma</math> is then defined according to the following rule:</p> <p>► <b>Rule S<sup>(1)</sup></b>  <math> </math> Computes <math>\sigma = \psi_{sk}(p)</math>.</p> <p>Store <math>(\gamma, m, 0^k, p, \sigma, \varphi_{pk}(p))</math> in P-List.</p>
Challenger	<p>For two messages <math>(m_0, m_1)</math>, flip coins <math>\gamma^*</math> and <math>b</math>, set <math>m^* = m_b</math>, and randomly choose <math>r^*</math>.</p> <p>► <b>Rule Chal<sup>(1)</sup></b>  <math>  p^* = \mathcal{P}(\gamma^*, m^*, r^*); c^* = \varphi_{pk}(p^*)</math>.</p> <p>► <b>Rule ChalAdd<sup>(1)</sup></b>  <math> </math> Add <math>(m^*, \gamma^*, r^*, \perp, \perp, c^*)</math> in P-List.</p> <p>Answer <math>c^*</math></p>
$\mathcal{V}$ -Oracle	<p>The game ends with the verification of the output <math>\sigma</math> from the adversary. One first computes <math>t  u = \varphi_{pk}(\sigma)</math>, then asks for <math>(\gamma, m, r) = \mathcal{P}^{-1}(t  u)</math>. Then he checks whether <math>r = 0^k</math>, in which case the signature is a valid signature of <math>m</math>.</p>

**Fig. 2.** Simulation in the Game  $\mathbf{G}_1$

Note that we “store” elements of the form  $(\gamma, m, r, p, s, c)$  in P-List. Roughly speaking, we use  $p$  to store the value of  $\mathcal{P}(\gamma, m, r)$ ,  $c$  to store the value  $\varphi_{\text{pk}}(p)$ , and  $s$  to store the value of  $\psi_{\text{sk}}(p)$  (i.e.,  $p = \varphi_{\text{pk}}(s)$ ). The former will help us to simulate the decryption oracle, while the latter will help us to simulate the signing oracle. In the organization of the list P-List, when we store an element  $(\gamma, m, r, p, s, c)$  in P-List, if there is already an element  $(\gamma, m, r, \alpha, \beta, y)$  in P-List with  $\alpha = \perp$  or  $\beta = \perp$ , we replace the latter by the former.

Now, we denote by  $\Delta_n$  the statistical distance between the distribution of the adversary’s view in the game  $\mathbf{G}_n$  and in the game  $\mathbf{G}_{n-1}$ . We see that the simulation is perfect since P-List is not used. The latter list is only introduced for later simulation in the proof. Therefore:  $\Delta_1 = 0$ .

**GAME  $\mathbf{G}_2$ :** In this game, we modify the simulation of the oracle  $\mathcal{P}$ ,  $\mathcal{P}^{-1}$  so that the random permutation  $P$  and its inverse  $P^{-1}$  are called at most once for each input. We use the following rules:

► **Rule EvalP<sup>(2)</sup>**

Look for  $(\gamma, m, r, \alpha, \beta, c)$  in P-List:

- if the record is found,
  - if  $\alpha \neq \perp$ ,  $p = \alpha$ ;
  - otherwise, **Stop**.
- otherwise,  $p = P(\gamma, m, r)$ .

► **Rule InvP<sup>(2)</sup>**

Compute  $c = \varphi_{\text{pk}}(p)$  and look for  $(\gamma, m, r, \alpha, \beta, c)$  in P-List:

- if the record is found,  $(\gamma, m, r)$  is defined,
- otherwise we compute  $(\gamma, m, r) = P^{-1}(p)$ .

We would give a wrong answer in the case the oracle  $P$  is called and if the record is found and  $\alpha = \perp$ . In this case, we stop the game but the correct answer should be  $\psi_{\text{sk}}(c)$ . We define this event **BadP<sub>2</sub>**. In further games  $\mathbf{G}_n$ , this event is defined as **BadP<sub>n</sub>**. Unless this event happens, the two games  $\mathbf{G}_2$  and  $\mathbf{G}_1$  are perfectly indistinguishable:  $\Delta_2 \leq \Pr[\text{BadP}_2]$ .

**GAME  $\mathbf{G}_3$ :** In this game, we modify the simulation of the oracles  $\mathcal{P}$  and  $\mathcal{P}^{-1}$ , without asking any query at all to the random permutation  $P$  (and its inverse  $P^{-1}$ ), but answering a random value for any new query: for a new  $(\gamma, m, r)$ , we answer  $\mathcal{P}(\gamma, m, r)$  by a random  $p$ , and for a new  $p$  we answer  $\mathcal{P}^{-1}(p)$  by a random tuple  $(\gamma, m, r)$ . We rewrite the rules as follows:

► **Rule EvalP<sup>(3)</sup>**

Look for  $(\gamma, m, r, \alpha, \beta, c)$  in P-List:

- if the record is found,
  - if  $\alpha \neq \perp$ ,  $p = \alpha$ ;
  - otherwise, **Stop**.
- otherwise,  $p \xleftarrow{R} \{0, 1\}^n$ .

► **Rule InvP<sup>(3)</sup>**

Compute  $c = \varphi_{\text{pk}}(p)$  and look for  $(\gamma, m, r, \alpha, \beta, c)$  in P-List:

- if the record is found,  $(\gamma, m, r)$  is defined,
- otherwise we randomly choose  $(\gamma, m, r)$  in  $\{0, 1\}^n$ .

The two games  $\mathbf{G}_3$  and  $\mathbf{G}_2$  are perfectly indistinguishable unless a collision appears over  $(\gamma, m, r)$  or  $p$  in P-List, we define this event **CollP<sub>3</sub>**. So, we have:  $\Delta_3 \leq \Pr[\text{CollP}_3]$ . Since there are at most  $q_p + q_d + q_s + 1$  elements in the P-List, such a collision appears with probability bounded by  $(q_p + q_d + q_s + 1)^2 / 2^n$ :

$$\Pr[\text{CollP}_3] \leq \frac{(q_p + q_d + q_s + 1)^2}{2^{k+\ell+1}}.$$

**GAME  $\mathbf{G}_4$ :** Now, we can simulate the decryption oracle  $\mathcal{D}_{\text{sk}}$  without  $\psi_{\text{sk}}$ :

► **Rule  $\mathcal{D}^{(4)}$**

- | Look for  $(\gamma, m, r, \alpha, \beta, c)$  in P-List:
  1. if the record is found,  $(\gamma, m, r)$  is defined,
  2. otherwise we randomly choose  $(\gamma, m, r)$  in  $\{0, 1\}^n$ .

The two games  $\mathbf{G}_4$  and  $\mathbf{G}_3$  are perfectly indistinguishable since in case 2, we do exactly as  $\mathcal{P}^{-1}$  was simulated by the rule  $\text{InvP}^{(3)}$  in the previous game, including the storage in P-List:  $\Delta_4 = 0$ .

GAME  $\mathbf{G}_5$ : We consider the elements of the form  $(m, r, \perp, \beta, c)$  in P-List (there are at most  $q_d + 1$  elements). If there is a collision on  $m$ , we abort the game, which event is named  $\text{CollM}_5$ . Therefore,  $\Delta_5 \leq \Pr[\text{CollM}_5]$ . We see that for any  $(\gamma, m, r, \perp, \beta, c)$  except if  $m = m^*$ ,  $m$  is chosen randomly. As a consequence, a collision over  $m$  can be found with probability bounded by  $(q_d + 1)^2 / 2^\ell$ :

$$\Pr[\text{CollM}_5] \leq \frac{(q_d + 1)^2}{2^\ell}.$$

When collisions are excluded, for any  $m$ , there is at most one  $r$  (and a  $c$ ) such that  $(m, r, \perp, \beta, c) \in \text{P-List}$ . One can thus see that

$$\Pr[\text{BadP}_5] \leq \frac{q_p + q_s + 1}{2^k}.$$

GAME  $\mathbf{G}_6$ : We now simulate the permutation oracle, in a way that will help us not to use the function  $\psi_{\text{sk}}$  anymore in the simulation of signing oracle. For that, we choose to return some  $\varphi_{\text{pk}}(s)$  to permutation oracle:

► **Rule  $\text{EvalP}^{(6)}$**

- | Look for  $(\gamma, m, r, \alpha, \beta, c)$  in P-List:
  - if the record is found,
    - if  $\alpha \neq \perp$ ,  $p = \alpha$ ;
    - otherwise, **Stop**.
  - otherwise, choose a random element  $s \in \{0, 1\}^n$  and compute  $p = \varphi_{\text{pk}}(s)$ , The record  $(\gamma, m, r, p, s, \varphi_{\text{pk}}(p))$  is added to P-List.

Because  $\varphi_{\text{pk}}$  is a permutation,  $p$  is a random. Therefore, the two games  $\mathbf{G}_6$  and  $\mathbf{G}_5$  are perfectly indistinguishable:  $\Delta_6 = 0$ .

GAME  $\mathbf{G}_7$ : In this game, we verify that all known quantities  $\mathcal{P}(\gamma, m, 0^k)$  that are known by the attacker have been asked to the  $\mathcal{P}$  oracle, and not deduced by call to the  $\mathcal{P}^{-1}$  oracle. Indeed, for all  $p$ , if the attacker asked  $a = \mathcal{P}^{-1}(p)$ , he knows that  $\mathcal{P}(a) = p$ . Hence, if the returned  $a$  value was of the form  $\gamma \| m_0 \| 0^k$ , our simulator would not know any  $s$ , so that  $\mathcal{P}(a) = \varphi_{\text{pk}}(s)$ , and so would not be able to sign the message  $m_0$  without  $\psi_{\text{sk}}$ .

► **Rule  $\text{InvP}^{(7)}$**

- | Compute  $c = \varphi_{\text{pk}}(p)$  and look for  $(\gamma, m, r, \alpha, \beta, c)$  in P-List:
  - if the record is found,  $(\gamma, m, r)$  is defined,
  - otherwise we randomly choose  $(\gamma, m, r)$  in  $\{0, 1\}^n$ . If  $r = 0^k$ , **Stop**.

Because of the random choice of the returned value to (new)  $\mathcal{P}^{-1}$  queries, we stop the game with probability  $1/2^k$  :  $\Delta_7 \leq (q_p + q_d + 1)/2^k$ .

GAME  $\mathbf{G}_8$ : By now, each  $\mathcal{P}(\gamma, m, 0^k)$  that is known by the attacker can be written easily by the simulator:  $\mathcal{P}(\gamma, m, 0^k) = \varphi_{\text{pk}}(s)$ , by just looking for  $(\gamma, m, 0^k, p, s, c)$  in the P-List. Hence, pre-images are known.

One can thus simulate the signing oracle without querying  $\psi_{\text{sk}}$ :

► **Rule  $\mathcal{S}_{\text{sk}}^{(8)}$**

- | Look for  $(\gamma, m, 0^k, p, s, c)$  in P-List, and set  $\sigma = s$ .

Hence, this game is indistinguishable from the previous one:  $\Delta_8 = 0$ .

Until now, we have simulated all the queries that the adversary could make: permutation oracles, decryption oracle and signing oracle. These simulations are independent to whatever the adversary does and therefore, we can easily get the expected result.

## B Proof of the security for the case of OAEP-3 rounds

We first review the main steps, details can be found in the full version of [22].

**GAME  $\mathbf{G}_0$ :** This is the attack game, in the random-oracle model. The goal of the proof is to reduce the inversion of the permutation  $\varphi_{pk}$  on a random instance  $y$  (i.e., find  $x$  such that  $y = \varphi_{pk}(x)$ ) to an attack against the encryption or signature. As for the previous proof, we are interested in the events **Dist** and **Forge** which indicate the success of the adversary:

$$\begin{aligned}\Pr[\text{Dist}_0] &= \text{Adv}_{\text{OAEP3r}}^{\text{ind/cma+cca}}(\mathcal{A}) + \frac{1}{2} \\ \Pr[\text{Forge}_0] &= \text{Succ}_{\text{OAEP3r}}^{\text{euf/cma+cca}}(\mathcal{B}).\end{aligned}$$

Because it has a particular interest for signature, we name  $\delta_0$  the value  $\mathcal{F}(0^k)$ . We use in the following a PRF  $\text{PRF}_\varrho$  for a random secret key  $\varrho$ .

**Advantage Zero to an adversary against encryption.** The goal of the first two games is to build a game in which  $b$  is perfectly indistinguishable to any adversary.

**GAME  $\mathbf{G}_1$ :** The simulation in this game is presented in Figure 3. We simulate the random oracles  $\mathcal{F}$ ,  $\mathcal{G}$  and  $\mathcal{H}$ , as well as the decryption oracle  $\mathcal{D}_{sk}$ , by maintaining lists: **F-List**, **G-List** and **H-List** as usual for the random oracles, and **D-List** to deal with similar decryption queries. This perfect simulation does not modify any probability:  $\Delta_1 = 0$ .

**GAME  $\mathbf{G}_2$ :** In order to make the advantage of any (even powerful) adversary against encryption scheme exactly zero, we define the mask  $f^*$  so that it is totally independent of the view of the adversary:

► **Rule Chal<sup>(2)</sup>**

The three values  $\gamma^+ \xleftarrow{R} \{0,1\}$ ,  $r^+ \xleftarrow{R} \{0,1\}^k$  and  $f^+ \xleftarrow{R} \{0,1\}^{\ell+1}$  are given, then  $\gamma^* = \gamma^+$ ,  $r^* = r^+$ ,  $f^* = f^+$  and  $s^* = (\gamma^+ \| m^*) \oplus f^+$ ,  $g^* = \mathcal{G}(s^*)$ ,  $t^* = r^+ \oplus g^*$ ,  $h^* = \mathcal{H}(t^*)$ ,  $u^* = s^* \oplus h^*$ .

The two games  $\mathbf{G}_2$  and  $\mathbf{G}_1$  are perfectly indistinguishable unless  $r^*$  has been asked for  $\mathcal{F}$  (by the adversary or the decryption oracle). We thus define this event **AskF<sub>2</sub>**, and we have:  $\Delta_2 \leq \Pr[\text{AskF}_2]$ . As hoped, in this game,  $f^+$  is used in the generation of the challenge, but does not appear anywhere else since  $\mathcal{F}(r^+)$  is not defined to be equal to  $f^+$ . Thus, the output of  $\mathcal{A}_2$  follows a distribution that does not depend on  $b$ . Accordingly,  $\Pr[\text{Dist}_2] = 1/2$ . We thus obtain a first conclusion:  $\text{Adv}_{\text{OAEP3r}}^{\text{ind/cma+cca}}(t) \leq \Pr[\text{AskF}_2]$ . As in the proof in [22], we are now interested in the event **AskF**.

**No  $\mathcal{G}$  and  $\mathcal{H}$  Queries in the Decryption Simulation.** We modify the decryption process so that it makes no new query to  $\mathcal{G}$  and  $\mathcal{H}$ .

**GAME  $\mathbf{G}_3$ :** We begin to simulate the decryption oracle. First, we modify the rules **D-noT**, **D-TnoS** and **D-TSnoR** by outputting a random message, and choosing at random the  $\mathcal{F}$ ,  $\mathcal{G}$  and  $\mathcal{H}$  oracles outputs, without looking first in **F-List** and **G-List**:

► **Rule D-noT<sup>(3)</sup>**

Choose  $m \xleftarrow{R} \{0,1\}^\ell$ ,  $\gamma \xleftarrow{R} \{0,1\}$ ,  $g \xleftarrow{R} \{0,1\}^k$  and  $h \xleftarrow{R} \{0,1\}^{\ell+1}$ .  
Set  $s = u \oplus h$ ,  $r = t \oplus g$  and compute  $f = (\gamma \| m) \oplus s$ .  
Add  $(r, f)$  in **F-List**,  $(s, \perp, g)$  in **G-List**,  $(t, h)$  in **H-List**.



$\mathcal{F}, \mathcal{G}$ and $\mathcal{H}$ Oracles	<p>Query <math>\mathcal{F}(r)</math>: if a record <math>(r, f)</math> appears in F-List, the answer is <math>f</math>.  Otherwise the answer <math>f</math> is chosen randomly in <math>\{0, 1\}^{\ell+1}</math> and the record <math>(r, f)</math> is added in F-List.</p> <hr/> <p>Query <math>\mathcal{G}(s)</math>: if a record <math>(s, \star, g)</math> appears in G-List, the answer is <math>g</math>.  Otherwise:  <b>►Rule <math>\mathcal{G}^{(1)}</math></b>    the answer <math>g</math> is chosen randomly in <math>\{0, 1\}^k</math> and the record <math>(s, \perp, g)</math> is added in G-List.</p> <p><b>►Rule EvalGAdd<sup>(1)</sup></b>    Do nothing                      % To be defined later</p> <hr/> <p>Query <math>\mathcal{H}(t)</math>: if a record <math>(t, h)</math> appears in H-List, the answer is <math>h</math>.  Otherwise the answer <math>h</math> is chosen randomly in <math>\{0, 1\}^{\ell+1}</math> and the record <math>(t, h)</math> is added in H-List.</p>
$\mathcal{D}$ -Oracle	<p>Query <math>\mathcal{D}_{\text{sk}}(c)</math>: the answer <math>m</math> is defined according to the following rules:  <b>►Rule <math>\mathcal{D}\text{-Init}^{(1)}</math></b>    Compute <math>t\ u = \psi_{\text{sk}}(c)</math>;</p> <p>Look for <math>(t, h) \in \text{H-List}</math>:</p> <ul style="list-style-type: none"> <li>- if the record is found, compute <math>s = u \oplus h</math>. Look for <math>(s, \perp, g) \in \text{G-List}</math>: <ul style="list-style-type: none"> <li>• if the record is found, compute <math>r = t \oplus g</math>. Look for <math>(r, f) \in \text{F-List}</math>: <ul style="list-style-type: none"> <li>* if the record is found <b>►Rule <math>\mathcal{D}\text{-TSR}^{(1)}</math></b>   <math>h = \mathcal{H}(t),</math>  <math>s = u \oplus h, \quad g = \mathcal{G}(s),</math>  <math>r = t \oplus g, \quad f = \mathcal{F}(r),</math>  <math>\gamma\ m = s \oplus f.</math> </li> <li>* else <b>►Rule <math>\mathcal{D}\text{-TSnoR}^{(1)}</math></b>   same as rule <math>\mathcal{D}\text{-TSR}^{(1)}</math>. </li> </ul> </li> <li>• else <b>►Rule <math>\mathcal{D}\text{-TnoS}^{(1)}</math></b>   same as rule <math>\mathcal{D}\text{-TSR}^{(1)}</math>. </li> </ul> </li> <li>- else <b>►Rule <math>\mathcal{D}\text{-noT}^{(1)}</math></b>   same as rule <math>\mathcal{D}\text{-TSR}^{(1)}</math>. </li> </ul> <p>Answer <math>m</math> and add <math>(\gamma, m, c)</math> to <math>\mathcal{D}</math>-List.</p>
$\mathcal{S}_{\text{sk}}$ -Oracle	<p>For a sign-query <math>\mathcal{S}(m)</math>, one first computes <math>\gamma = \text{PRF}_e(m)</math>, then asks for <math>\delta_0 = \mathcal{F}(0^k)</math>, then asks for <math>t = \mathcal{G}((\gamma\ m) \oplus \delta_0)</math> to the <math>\mathcal{G}</math>-oracle, then asks for <math>h = \mathcal{H}(t)</math>, and then compute <math>u = (\gamma\ m) \oplus \delta_0 \oplus h</math>. The signature <math>\sigma</math> is defined according to the following rule:  <b>►Rule <math>\mathcal{S}_{\text{sk}}^{(1)}</math></b>    Computes <math>\sigma = \psi_{\text{sk}}(t\ u)</math>.</p>
Challenger	<p>For two messages <math>(m_0, m_1)</math>, flip a coin <math>b</math> and set <math>m^* = m_b</math>, choose randomly <math>r^*</math> and <math>\gamma^*</math>, and then answer <math>c^*</math> where  <b>►Rule Chal<sup>(1)</sup></b>    <math>f^* = \mathcal{F}(r^*), \quad s^* = (\gamma^*\ m^*) \oplus f^*,</math>  <math>g^* = \mathcal{G}(s^*), \quad t^* = r^* \oplus g^*,</math>  <math>h^* = \mathcal{H}(t^*), \quad u^* = s^* \oplus h^*.</math></p> <p><b>►Rule ChalC<sup>(1)</sup></b>    and <math>c^* = \varphi_{\text{pk}}(t^*\ u^*)</math></p>
$\mathcal{V}$ -Oracle	<p>The game ends with the verification of the output <math>\sigma</math> from the adversary. One first computes <math>t\ u = \varphi_{\text{pk}}(\sigma)</math>, then recovers <math>(\gamma\ m\ r) = \text{OAEP3r}^{-1}(t, u)</math>. Then he checks whether <math>r = 0^k</math>.</p>

Fig. 3. Formal Simulation of the IND-CCA Game

► **Rule  $\mathcal{D}\text{-TnoS}^{(3)}$**

Choose  $m \xleftarrow{R} \{0,1\}^\ell$ ,  $\gamma \xleftarrow{R} \{0,1\}$  and  $g \xleftarrow{R} \{0,1\}^k$ .  
 Set  $r = t \oplus g$  and compute  $f = (\gamma \| m) \oplus s$ .  
 Add  $(r, f)$  in F-List,  $(s, \perp, g)$  in G-List.

► **Rule  $\mathcal{D}\text{-TSnoR}^{(3)}$**

Choose  $m \xleftarrow{R} \{0,1\}^\ell$  and  $\gamma \xleftarrow{R} \{0,1\}$ .  
 Compute  $f = (\gamma \| m) \oplus s$ .  
 Add  $(r, f)$  in F-List.

The above rules are almost similar as before, except that inconsistencies may appear if some elements were already in the lists. But inputs are random, and thus the collisions are unlikely.

$$\Delta_3 \leq q_d \times \left( \frac{q_g + q_d}{2^{\ell+1}} + 2 \times \frac{q_f + q_d}{2^k} \right).$$

GAME  $\mathbf{G}_4$ : In the previous rules for the simulation of the decryption simulation, the random oracles were almost perfectly simulated, adding new relations to the corresponding lists. We now make more technical modifications, which differ a lot from previous proofs. Granted that, we can achieve a stronger result. We thus modify the above rules by not storing anymore the new relations  $(s, \perp, g)$  in G-List, defined during these simulations. Therefore, when  $g$  is no longer explicitly defined, we cannot compute  $r$  and thus we do not store  $(r, f)$  in F-List either. However, as soon as  $\mathcal{G}(s)$  is known, we must define  $\mathcal{F}(r)$  accordingly, and update the lists:

► **Rule  $\mathcal{D}\text{-TnoS}^{(4)}$**

Choose  $m \xleftarrow{R} \{0,1\}^\ell$  and  $\gamma \xleftarrow{R} \{0,1\}$ .

► **Rule  $\mathcal{D}\text{-noT}^{(4)}$**

Choose  $m \xleftarrow{R} \{0,1\}^\ell$ ,  $\gamma \xleftarrow{R} \{0,1\}$  and  $h \xleftarrow{R} \{0,1\}^\ell$ .  
 Add  $(t, h)$  in H-List.

► **Rule EvalGAdd<sup>(4)</sup>**

Look for  $(t, h) \in \text{H-List}$  and for  $(\gamma, m, c) \in \text{D-List}$  such that  $c = \varphi_{\text{pk}}(t \| h \oplus s)$ . If the record is found, we compute  $r = t \oplus g$  and  $f = (\gamma \| m) \oplus s$ , and finally add  $(r, f)$  in F-List.

With the above new rules, the answers in the two games  $\mathbf{G}_4$  and  $\mathbf{G}_3$  are perfectly indistinguishable unless  $r$  is asked to  $\mathcal{F}$  before  $s$  is asked to  $\mathcal{G}$ , which event is denoted by  $\text{AskRbS}_4$ . In fact, if  $r$  is asked after  $s$ , at the moment that  $s$  is asked, by the above simulation of  $\mathcal{G}$  (and the extra rule EvalGAdd), we will find out  $(t, h)$  and therefore  $(r, f)$  is computed in a consistent way, exactly as it would have been in the game  $\mathbf{G}_3$ , and added in F-List.

Note that for each ciphertext  $c$ , the value  $t$  is unique, and thus  $h$ , and consequently  $s$ : this rule is thus applied at most once for each ciphertext asked to the decryption oracle. However, until  $s$  is asked,  $g$  is a uniformly distributed random variable, and  $r$  is so too. Therefore, the probability that  $r$  has been asked to  $\mathcal{F}$  is  $\frac{q_f + q_d}{2^k}$ :

$$\Pr[\text{AskRbS}_4] \leq q_d \times \frac{q_f + q_d}{2^k}.$$

A more important gap may appear because of the removal of some elements  $(s, \perp, g)$  from G-List, and  $(r, f)$  from F-List, which may have some impact on the simulation of later decryption queries, but also on the event AskF itself:

- if we remove  $(r, f)$  for  $r = r^*$ , then the event AskF happened in the previous game but does not occur in the new game. Fortunately, since  $r = t \oplus g$  where  $g$  is randomly chosen, the probability of this event is  $1/2^k$ .

- in the simulation of a later decryption query  $c' = \varphi_{\text{pk}}(t' \| u')$ , the element  $s' = s$  might have been found in the previous game, while it is no longer in the list in the current game. A rule  $\mathcal{D}\text{--TSR}/\mathcal{D}\text{--TSnoR}$  is thus replaced by the rule  $\mathcal{D}\text{--TnoS}$ , which means that  $g$  was just defined during the first decryption, in the previous game, but never revealed later. Therefore, noting  $r' = t' \oplus g' = t' \oplus g$ , the probability that  $r'$  is in the F-List is  $(q_f + q_d)/2^k$  (modification of  $\mathcal{D}\text{--TSR}$  into  $\mathcal{D}\text{--TnoS}$ ). In the case that  $r'$  was not in the F-List,  $\gamma' \| m'$  was and is still random: modification of  $\mathcal{D}\text{--TSnoR}$  into  $\mathcal{D}\text{--TnoS}$ .

$$|\Pr[\text{AskF}_4] - \Pr[\text{AskF}_3]| \leq \Pr[\text{AskRbS}_4] + q_d \times \frac{q_f + q_d}{2^k} + q_d \times \frac{1}{2^k} \leq 2q_d \times \frac{q_f + q_d}{2^k} + q_d \times \frac{1}{2^k}.$$

Similarly, we also have:

$$|\Pr[\text{Forge}_4] - \Pr[\text{Forge}_3]| \leq 2q_d \times \frac{q_f + q_d}{2^k} + q_d \times \frac{1}{2^k}.$$

**GAME  $\mathbf{G}_5$ :** We follow in simplifying the simulation of the decryption, by not storing the new relations  $(t, h)$  in H-List either:

► **Rule  $\mathcal{D}\text{--noT}^{(5)}$**

Choose  $m \xleftarrow{R} \{0, 1\}^\ell$  and  $\gamma \xleftarrow{R} \{0, 1\}$ .

In the two games, the answers of the decryption simulations are identical, since they are random values. Nevertheless, the H-List has been changed, which may impact several other things:

- an  $\mathcal{F}$ -answer can be changed. Indeed, if  $s$  is asked to  $\mathcal{G}$  before  $t$  is asked to  $\mathcal{H}$ , which event is denoted by  $\text{AskSbT}_5$ , the rule  $\text{EvalGAdd}$  will not apply. Otherwise, when the event  $\text{AskSbT}_5$  does not happen, the F-List and the  $\mathcal{F}$  simulation are unchanged, after the  $\text{EvalGAdd}$  rule. Fortunately, until  $t$  is asked to  $\mathcal{H}$ ,  $h$  is a uniformly distributed random variable, and  $s = u \oplus h$  is so too. Therefore, the probability that  $s$  has been asked to  $\mathcal{G}$  is  $q_g/2^{\ell+1}$  (since no new  $\mathcal{G}$  relation is added by the decryption simulation):

$$\Pr[\text{AskSbT}_5] \leq q_d \times \frac{q_g}{2^{\ell+1}}.$$

- the removal of  $(t, h)$  from H-List, may have some impact on the simulation of a later decryption query  $c' = \varphi_{\text{pk}}(t' \| u')$ :
  - if  $s'$  is in the G-List and  $t' = t$ , it was found in the previous game, but it is no longer in the list. A rule  $\mathcal{D}\text{--TSR}/\mathcal{D}\text{--TSnoR}$  is thus replaced by the rule  $\mathcal{D}\text{--noT}$ . This event means that  $h' = h$  was just defined during the first decryption in the previous game, but never revealed later. The probability for  $s' = t' \oplus h$  to be in the G-List was less than  $q_g/2^{\ell+1}$ , which is an upper-bound of this case to appear.
  - if  $s'$  is not in the G-List but  $t' = t$  was found in the previous game, it may not be in the list any longer. A rule  $\mathcal{D}\text{--TnoS}$  is thus replaced by the rule  $\mathcal{D}\text{--noT}$ . In this case, the decryption is the same (it gives always a random plaintext and adds no element in the lists).

Summing up for all decryption queries, we get:

$$|\Pr[\text{AskF}_5] - \Pr[\text{AskF}_4]| \leq \Pr[\text{AskSbT}_5] + q_d \times \frac{q_g}{2^{\ell+1}} \leq 2q_d \times \frac{q_g}{2^{\ell+1}}.$$

Similarly, we also have:

$$|\Pr[\text{Forge}_5] - \Pr[\text{Forge}_4]| \leq 2q_d \times \frac{q_g}{2^{\ell+1}}.$$

Remark that the G-List and H-List contain now only the queries asked by the adversaries and by the generation of the challenge. The simulation of the decryption queries does not make/simulate any new query to  $\mathcal{G}$  or  $\mathcal{H}$ , but to  $\mathcal{F}$  only.

We denote by  $\text{AskGA}_5$  and  $\text{AskHA}_5$  the events that  $s^*$  and  $t^*$  (the values involved in the challenge), respectively, are asked by the adversary. The event  $\text{AskGHA}_5$  is also set to true when both  $\text{AskGA}_5$  and  $\text{AskHA}_5$  happen:  $\text{AskGHA} = \text{AskGA} \wedge \text{AskHA}$ . Note that these two queries,  $s^*$  and  $t^*$ , are also asked for the generation of the challenge  $c^*$ , but we do not consider them for the events  $\text{AskGA}_5$  and  $\text{AskHA}_5$ .

**The Classical Plaintext Extractor.** We now complete the modifications of the decryption process so that it behaves exactly as the classical plaintext extractor, briefly described in the sketch of the proof.

GAME  $\mathbf{G}_6$ : Before going on in some other modifications, we exclude some executions, with then a random output: the rule **Abort** is always checked. If it is true, we stop the game with a random output  $b'$ .

► **Rule Abort**<sup>(6)</sup>

| Abort and output a random bit: If  $\text{AskGA}_6 \wedge \neg \text{AskHA}_6$ , at the end.

When  $\neg \text{AskHA}_6$ ,  $\mathcal{H}(t^*) = u^* \oplus (\gamma^* \| m_b) \oplus f^+$  is never revealed, while  $f^+$  is a random value independent to the adversary's view. Therefore,  $\mathcal{H}(t^*)$  is a uniformly distributed random variable:  $s^* = u^* \oplus H(t^*)$  is so too. Consequently, the probability that  $s^*$  is queried is  $q_g/2^{\ell+1}$ . Consequently, the probability that this rule is applied is  $q_g/2^{\ell+1}$ :

$$\Delta_6 \leq \frac{q_g}{2^{\ell+1}}.$$

Furthermore,  $\Pr[\text{AskF}_6]$  can easily be upper-bounded with the following relation (by using the same classical argument as in [22]):

$$\Pr[\text{AskF}_6] \leq \frac{q_f}{2^k} + \Pr[\text{AskGHA}_6].$$

We can thus make another intermediate conclusion, which explains our interest in  $\Pr[\text{AskGHA}_6]$ :

$$\Pr[\text{AskF}_2] \leq q_d^2 \times \left( \frac{4}{2^k} + \frac{1}{2^{\ell+1}} \right) + \frac{4q_dq_f}{2^k} + \frac{3q_dq_g}{2^{\ell+1}} + \frac{q_g}{2^{\ell+1}} + \frac{q_f + q_d}{2^k} + \Pr[\text{AskGHA}_6].$$

GAME  $\mathbf{G}_7$ : We furthermore abort some games during the execution, if one of the following situations is met:

► **Rule Abort**<sup>(7)</sup>

| Abort and output a random bit:

- If  $\text{AskGA}_7 \wedge \neg \text{AskHA}_7$ , at the end.
- If a  $\mathcal{D}\text{-TSR}/\mathcal{D}\text{-TSnoR}$  rule has been applied with  $t = t^*$ , while  $\mathcal{H}(t^*)$  had not been asked by the adversary yet.
- If a  $\mathcal{D}\text{-TSR}$  rule has been applied with  $s = s^*$ , while  $\mathcal{G}(s^*)$  had not been asked by the adversary yet.

The gap between the two games is bounded by the proof of the following relation (by using the same classical argument as in [22]):

$$\Delta_7 \leq q_d \times \left( \frac{q_f + q_d}{2^k} + \frac{q_g}{2^{\ell+1}} \right).$$

GAME  $\mathbf{G}_8$ : In this game, we complete the simulation of the decryption oracle, so that it does not depend on the queries that the generation of the challenge makes. The decryption oracle does not use anymore the element  $(s^*, g^*)$  if the adversary did not ask for  $s^*$ .

► **Rule  $\mathcal{D}\text{-TSnoR}$** <sup>(8)</sup>

| If  $s = s^*$  but  $s^*$  has not been directly asked by the adversary yet:  $m \xleftarrow{R} \{0,1\}^\ell$  and  $\gamma \xleftarrow{R} \{0,1\}$ .  
 | Else, one chooses  $m \xleftarrow{R} \{0,1\}^\ell$  and  $\gamma \xleftarrow{R} \{0,1\}$ , computes  $f = (\gamma \| m) \oplus s$  and adds  $(r, f)$  in F-List.

If  $s = s^*$  but  $s^*$  has not been asked to  $\mathcal{G}$  by the adversary during a  $\mathcal{D}\text{-TSnoR}$  rule,  $f = \mathcal{F}(r)$  is a uniformly distributed random variable, therefore, we can give a random answer  $m$ . However, in this case, we do not store anymore  $(r, f)$  and this could make some problems: if a latter different decryption query  $c'$  involves  $\gamma' = \gamma$  and  $r' = r$ . In this case  $g^* \oplus t = g' \oplus t'$ . Since  $c' \neq c$  and  $r' = r$ ,  $s'$  must be not equal

to  $s$ . In this case,  $g = \mathcal{G}(s) = g^*$  is independent to  $g'$ . Moreover,  $s^*$  is not queried and thus  $r = g^* \oplus t$  is a uniformly distributed random variable: the probability that a later decryption query  $c'$  satisfies that is  $1/2^k$ :

$$\Delta_8 \leq \frac{q_d^2}{2^k}.$$

In the above game, one can remark that the simulation of the decryption does not use at all the queries asked to  $\mathcal{G}$  and  $\mathcal{H}$  by the generation of the challenge:

- $\mathcal{D}$ –TSR
  - with  $r = r^*$ : is not possible if the query  $r^*$  has not been queried directly by the adversary, since it has not been queried during the generation of the challenge;
  - with  $s = s^*$ : excluded in the game  $\mathbf{G}_7$ ;
  - with  $t = t^*$ : excluded in the game  $\mathbf{G}_7$ ;
- $\mathcal{D}$ –TSnoR
  - with  $s = s^*$ : similar to  $\mathcal{D}$ –TnoS since the game  $\mathbf{G}_8$ ;
  - with  $t = t^*$ : excluded in the game  $\mathbf{G}_7$ ;
- $\mathcal{D}$ –TnoS
  - with  $t = t^*$ : similar to  $\mathcal{D}$ –noT since the game  $\mathbf{G}_5$ ;

The simulation of the decryption of  $c$  is in fact the simple plaintext extractor [3, 9, 21] which looks up in the lists **G-List** and **H-List** (which only contain the queries directly asked by the adversary) to obtain the values  $(s, \star, g)$  and  $(t, h)$  which match with  $c = \varphi_{\text{pk}}(t \| s \oplus h)$  without using anymore  $\psi_{\text{sk}}$ :

► **Rule  $\mathcal{D}$ –Init<sup>(8)</sup>**

- |  |
|--|
| Look for $(t, h) \in \mathbf{H}\text{-List}$ and $(s, \star, g) \in \mathbf{G}\text{-List}$ such that $c = \varphi_{\text{pk}}(t \  s \oplus h)$ . |
| – if the record is found, we found out the corresponding $s$ and $t$ , and we furthermore define $u = s \oplus h$ .                                |
| – otherwise, we take $t = \perp$ and $u = \perp$ .   |

Note that the definitions  $t = \perp$  and  $u = \perp$  are just done to make the answer  $m$  to be random in the following of the simulation. The time complexity of one simulation is thus upper-bounded by  $q_g q_h \times (T_\varphi + T_{lu})$ , where  $T_\varphi$  is the time to evaluate one function in the  $\varphi$  family, and  $T_{lu}$  the time for looking up in the  $\mathcal{D}$ -List.

**GAME  $\mathbf{G}_9$ :** We now modify the simulator in order to deal with the signing oracle. We first modify the simulation of the hash oracle  $\mathcal{G}$ :

► **Rule  $\mathcal{G}$ <sup>(9)</sup>**

- |  |
|--|
| – Choose a random $\rho \in \{0, 1\}^n$ , compute $\alpha = \varphi_{\text{pk}}(\rho)$ , parse $\alpha$ into $\alpha = \alpha_1 \  \alpha_2$ , with $\alpha_1$ of $k$ bits and $\alpha_2$ of $(\ell + 1)$ bits. Then, $(s, \rho, \alpha_1)$ is added to <b>G-List</b> and $(\alpha_1, s \oplus \alpha_2)$ to the <b>H-List</b> . |
|--|

One remarks that the goal of this game is to return  $\varphi_{\text{pk}}(\rho)$  as the value of  $\text{OAEP3r}(\gamma, m, 0^k)$ : indeed,  $s = (\gamma \| m) \oplus \mathcal{F}(0^k) = (\gamma \| m) \oplus \delta_0$ ,  $t = \mathcal{G}(s) = \alpha_1$  and  $u = s \oplus \mathcal{H}(t) = s \oplus \mathcal{H}(\alpha_1) = \alpha_2$ , giving  $\text{OAEP3r}(\gamma, m, 0^k) = \varphi_{\text{pk}}(\rho)$ .

Because of the permutation property of  $\varphi_{\text{pk}}$ , and the random choice for  $\rho$ , this rule leaves the game indistinguishable from the previous one, except that there could exist an element  $h' \neq s \oplus \alpha_2$  such that  $(\alpha_1, h')$  is already in the **H-List**, *i.e.*, if  $\mathcal{H}(\alpha_1)$  is already defined. This is a failure that can happen for  $q_g$  queries. Because of the permutation property of  $\varphi_{\text{pk}}$ , and the random choice for  $\rho$ , each  $\alpha_1$  is random for the attacker, and so the difference between this game and the previous one is

$$\Delta_9 \leq \frac{q_g(q_h + q_s)}{2^k}.$$

**GAME  $\mathbf{G}_{10}$ :** We can now simulate the signing oracle without querying  $\psi_{\text{sk}}$ :

► **Rule  $\mathcal{S}_{\text{sk}}^{(10)}$**

| Look for  $((\gamma \| m) \oplus \delta_0, \rho, g)$  in **G-List**, and set  $\sigma = \rho$ .

This rule leaves the game indistinguishable from the previous one:  $\Delta_{10} = 0$ .

By now, we complete the simulation of all hash oracles as well as the decryption oracle and the signing oracle for any attack. In the following, depending on the goal of the attacker, we manage to invert the function  $\varphi_{\text{pk}}$  on the given instance  $y$  (or computing the function  $\psi_{\text{sk}}$  on the instance  $y$ ).

**Encryption attack.**

GAME  $\mathbf{G}_{10.1}$ : We can now modify the simulation of the challenge, without querying  $\mathcal{G}$  or  $\mathcal{H}$ :

► **Rule  $\text{Chal}^{(10.1)}$**

| The three values  $\gamma^+ \xleftarrow{R} \{0, 1\}$ ,  $r^+ \xleftarrow{R} \{0, 1\}^k$  and  $f^+ \xleftarrow{R} \{0, 1\}^{\ell+1}$  are given, as well as  $g^+ \xleftarrow{R} \{0, 1\}^k$  and  $h^+ \xleftarrow{R} \{0, 1\}^{\ell+1}$  then  $\gamma^* = \gamma^+$ ,  $r^* = r^+$ ,  $f^* = f^+$ ,  $s^* = (\gamma^* \| m^*) \oplus f^+$ ,  $g^* = g^+$ ,  $t^* = r^+ \oplus g^+$ ,  $h^* = h^+$  and  $u^* = s^* \oplus h^*$ .

As seen above, this does not impact at all the simulation of the decryption, and the rule **EvalGAdd** either since the modification had already been considered in the game  $\mathbf{G}_{10.5}$ . The probability distributions are thus unchanged.

The global running time is bounded by (including all the list look up):

$$\tau' \leq \tau + q_d q_g q_h \times (T_\varphi + T_{lu}) + (q_f + q_g + q_h + q_d) \times T_{lu}.$$

In the particular, one can improve it, using an extra list of size  $q_g q_h$ , which stores all the tuples  $(s, g = \mathcal{G}(s), t, h = \mathcal{H}(t), c = \varphi_{\text{pk}}(t \| s \oplus h))$ . The time complexity then falls down to  $\tau + q_g q_h \times T_\varphi + (q_f + q_g + q_h + q_d) \times T_{lu}$ .

**Conclusion.** The proof is almost finished, granted the permutation property of  $\varphi_{\text{pk}}$  from  $\{0, 1\}^k \times \{0, 1\}^\ell \times \{0, 1\}$  onto  $\{0, 1\}^n$ . Indeed using a classical argument [22], one easily gets the relation:

$$\Pr[\text{AskGHA}_{10.1}] \leq \text{Succ}_\varphi^{\text{ow}}(\tau' + q_g q_h (T_\varphi + T_{lu}), q_d q_g q_h + q_d^2),$$

where  $\tau'$  is the above running time of the simulation, which concludes the proof of the Theorem, since

$$\Pr[\text{AskGHA}_6] \leq \frac{2q_d^2}{2^k} + q_d \times \left( \frac{q_f}{2^k} + \frac{q_g}{2^{\ell+1}} \right) + \text{Succ}_\varphi^{\text{ow}}(\tau' + q_g q_h (T_\varphi + T_{lu}), q_d q_g q_h + q_d^2).$$

**Signature attack (general case).**

GAME  $\mathbf{G}_{10.1}$ : In the following, we number calls to the  $\mathcal{G}$  oracle. We thus define a variable  $\nu$  which is initialized to 0. Then:

► **Rule  $\mathcal{G}^{(10.1)}$**

| – Increment  $\nu$   
 | – Choose a random  $\rho \in \{0, 1\}^n$ , compute  $\alpha = \varphi_{\text{pk}}(\rho)$ , split  $\alpha$  into  $\alpha = \alpha_1 \| \alpha_2$ , with  $\alpha_1$  of  $k$  bits and  $\alpha_2$  of  $(\ell + 1)$  bits. Then,  $(s, \rho, \alpha_1)$  is added to **G-List** and  $(\alpha_1, s \oplus \alpha_2)$  to the **H-List**.

Clearly, this leaves the game indistinguishable from the previous one:  $\Delta_{10.1} = 0$ .

GAME  $\mathbf{G}_{10.2}$ : Recall that since the verification process is included in the attack game, the value  $s = (\gamma \| m) \oplus \delta_0$  (where  $m$  is the output message) is necessarily asked to the hash oracle  $\mathcal{G}$ . Let us guess the index  $\nu_0$  of this (first) query. If the guess failed, we abort the game. Therefore, only a correct guess (event **GoodGuess**) may lead to a success.

$$\begin{aligned}\Pr[\text{Forge}_{10.2}] &= \Pr[\text{Forge}_{10.1} \wedge \text{GoodGuess}] = \Pr[\text{Forge}_{10.1} \mid \text{GoodGuess}] \times \Pr[\text{GoodGuess}] \\ &\geq \Pr[\text{Forge}_{10.1}] \times \frac{1}{q_g + q_s + 1}.\end{aligned}$$

**GAME  $\mathbf{G}_{10.3}$ :** We can now simulate the hash oracle  $\mathcal{G}$ , incorporating the challenge  $y$ , for which we want to extract the pre-image  $x$  by  $\varphi_{\text{pk}}$ .

Our goal in this game is to return, at the guessed  $\nu_0$ -th query, a certain result, so that the value of  $\text{OAEP3r}(\gamma, m, 0^k)$ , where  $m$  is the message which is used in the forge of the signature attacker, is equal to  $y$ .

► **Rule  $\mathcal{G}^{(10.3)}$**

- Increment  $\nu$
- If  $\nu = \nu_0$ , parse  $y$  into  $y = y_1 \| y_2$ , with  $y_1$  of  $k$  bits and  $y_2$  of  $(\ell + 1)$  bits. Then,  $(s, \rho, y_1)$  is added to **G-List** and  $(y_1, s \oplus y_2)$  to the **H-List**.
- Otherwise, choose a random  $\rho \in \{0, 1\}^n$ , compute  $\alpha = \varphi_{\text{pk}}(\rho)$ , divide  $\alpha$  into  $\alpha = \alpha_1 \| \alpha_2$ , with  $\alpha_1$  of  $k$  bits and  $\alpha_2$  of  $(\ell + 1)$  bits. Then, add  $(s, \rho, \alpha_1)$  is added to **G-List** and  $(\alpha_1, s \oplus \alpha_2)$  to the **H-List**.

For the  $\nu_0$ -th query, we have  $s = (\gamma \| m) \oplus \mathcal{F}(0^k) = (\gamma \| m) \oplus \delta_0$ ,  $t = \mathcal{G}(s) = y_1$  and  $u = s \oplus \mathcal{H}(t) = s \oplus \mathcal{H}(y_1) = y_2$ , giving  $\text{OAEP3r}(\gamma, m, 0^k) = y$ .

Because of the random choice for the challenge  $y$ , and so of  $(y_1, y_2)$ , this rule leaves the game indistinguishable from the previous one:  $\Delta_{10.3} = 0$ . In this game, it's easy to see that the forgery leads to the pre-image of  $y$ :

$$\Pr[\text{Forge}_{10.3}] = \text{Succ}_{\varphi}^{\text{ow}}(\tau' + q_g q_h (T_{\varphi} + T_{lu}), q_d q_g q_h + q_d^2).$$

where  $\tau'$  is bounded by the same way as in the encryption attack :

$$\tau' \leq \tau + q_d q_g q_h \times (T_{\varphi} + T_{lu}) + (q_f + q_g + q_h + q_d) \times T_{lu}.$$

**Signature attack when  $\varphi_{\text{pk}}$  is induced by a  $(t, \varepsilon')$ -secure claw-free permutation  $(\varphi_{\text{pk}}, \lambda_{\text{pk}})$ .**

**GAME  $\mathbf{G}_{10.1}$ :** We now exploit the bit  $\gamma$  to the simulation of the permutation oracle, as it was proposed firstly by Katz and Wang [13]. The idea is to use  $\varphi_{\text{pk}}$  in the **OAEP3r** output, for one and only one value of the bit  $\gamma$ , and otherwise use  $\lambda_{\text{pk}}$ . As this value of  $\gamma$  is not predictable by the attacker, its forgery will, with a probability  $\frac{1}{2}$ , produce a claw.

► **Rule  $\mathcal{G}^{(10.1)}$**

- Compute  $\gamma \| m = s \oplus \delta_0$
- If  $\gamma = \text{PRF}_{\varrho}(m)$ , choose a random  $\rho \in \{0, 1\}^n$ , compute  $\alpha = \varphi_{\text{pk}}(\rho)$ , divide  $\alpha$  into  $\alpha = \alpha_1 \| \alpha_2$ , with  $\alpha_1$  of  $k$  bits and  $\alpha_2$  of  $(\ell + 1)$  bits. Then, add  $(s, \rho, \alpha_1)$  is added to **G-List** and  $(\alpha_1, s \oplus \alpha_2)$  to the **H-List**.
- If  $\gamma \neq \text{PRF}_{\varrho}(m)$ , choose a random  $\rho \in \{0, 1\}^n$ , compute  $\alpha = \lambda_{\text{pk}}(\rho)$ , divide  $\alpha$  into  $\alpha = \alpha_1 \| \alpha_2$ , with  $\alpha_1$  of  $k$  bits and  $\alpha_2$  of  $(\ell + 1)$  bits. Then, add  $(s, \rho, \alpha_1)$  is added to **G-List** and  $(\alpha_1, s \oplus \alpha_2)$  to the **H-List**.

Because of the random choice of  $\rho$  and so  $\lambda_{\text{pk}}(\rho)$ , this rule leaves the game indistinguishable from the previous one:  $\Delta_{10.1} = 0$ .

Using the arguments in [13], one can easily see that the forgery leads to a claw with probability  $\frac{1}{2}$ . In fact, let us assume that the adversary can forge a signature  $(\tilde{m}, \tilde{\sigma})$ , where  $((b_{\tilde{m}} \| \tilde{m}) \oplus \delta_0)$  has been asked to the oracle  $\mathcal{G}$  either in a hash query or in the verification step. Since the bit  $b_{\tilde{m}} = \text{PRF}_{\varrho}(\tilde{m})$  is an unknown random bit in the view of the adversary, with probability of  $\frac{1}{2}$ , there exists an element  $(\tilde{s}, \tilde{\rho}, \tilde{\alpha}_1)$

in the **G-List** and an element  $(\tilde{\alpha}_1, \tilde{s} \oplus \tilde{\alpha}_2)$  in the **H-List** such that  $\tilde{\alpha} = \tilde{\alpha}_1 \parallel \tilde{\alpha}_2 = \lambda_{\text{pk}}(\tilde{\rho})$ . In that case, the simulator can output a claw  $\varphi_{\text{pk}}(\tilde{\sigma}) = \lambda_{\text{pk}}(\tilde{\rho})$ .

$$\Pr[\text{Forge}_{10.1}] = \text{Succ}_{\varphi}^{\text{ow}}(\tau' + q_g q_h (T_{\varphi} + T_{lu}) + T_{\lambda}, q_d q_g q_h + q_d^2).$$

where  $\tau'$  is bounded by the same way as in the encryption attack :

$$\tau' \leq \tau + q_d q_g q_h \times (T_{\varphi} + T_{lu}) + (q_f + q_g + q_h + q_d) \times T_{lu}.$$

This easily concludes the proof. □