Provable Security in Cryptography

Thomas Baignères

EPFL
`http://lasecwww.epfl.ch`

May 29, 2007 (ver. 25)

These lecture notes are a compilation of some of my readings while I was preparing two lectures given at EPFL on provable security in cryptography. They are essentially based on a book chapter from David Pointcheval called "Provable Security for Public Key Schemes" [24], on Victor Shoup's tutorial on game playing techniques [30], on Coron's CRYPTO'00 paper on the exact security of the Full Domain Hash [9], and on Victor Shoup's Journal of Cryptology paper on OAEP+ [28,29].

1 Provable Security

Although the origin of cryptography seems to date back to the invention of writing, no provably secure cryptosystem (a notion that will be made clearer later) was known before Rabin's cryptosystem, published in 1979 [18,25]. Yet, several cryptosystems designed during the past 30 years provide very little (not to say no) security proofs. Some of these algorithms are widely used in nowadays secure applications. For example, if it was not for the work of Keliher [15], the AES [10] (the block cipher adopted as an encryption standard by the U.S. government) would not provide any (convincing) security proof against linear cryptanalysis [20] (a very powerful, yet very specific attack). The strongest argument in favor of the security of the AES is that, until now, none of the smart cryptanalytic attempts to break it was successful. This fact, added to the very nice design rationales on which the AES relies, are often considered as sufficient from a security perspective. Are we done then? Not quite. It sometimes takes time to break a cryptographic scheme. For example the Chor-Rivest cryptosystem [8] resisted to almost 15 years of cryptanalytic efforts, until it was completely broken by Vaudenay [32]. Obviously, the lack of successful cryptanalytic attack shall not replace a security proof.

But what do cryptographers exactly mean by *provable security*? Informally, a scheme is *provably secure* if it comes with a rigorous logical argument that shows that if the security of this scheme is compromised then

– either some simple logical contradiction occurs (Information theoretic security or security against computationally unbounded adversaries),
– or some well-studied problem can be solved efficiently (security against computationally bounded adversaries).

In the latter case, one must first assume the hardness of some problem (such as factorization of large integers) or the existence of some primitive (such as a one-way function f, for which it is easy to compute $f(x)$, but given $y = f(x)$ it is computationally intractable to recover x). In order to prove the security of a cryptographic scheme, one shows that a potential adversary against the scheme (i.e., an algorithm that breaks the scheme) can be used as a subroutine in order to *efficiently* break the computational assumption. We say that the cryptographic scheme *reduces to* the computational assumption (a notion borrowed from complexity theory). The reduction is considered to be *efficient* when both the time and space complexities of the routine against the computational assumption are bounded by a polynomial in the size of some security input (e.g., the size an RSA modulus). However, even the existence of such a proof may have little impact on *practical* security. To illustrate this fact we borrow an example from Koblitz and Menezes [17].

The Blum-Blum-Shub generator [6] is a cryptographically secure pseudorandom bit generator. Let $N = p \cdot q$ be the n-bit product of two large primes both congruent to $3 \bmod 4$. Let $x_0 < N$ be a random integer and define $x_i = x_{i-1}^2 \bmod N$ for $i = 1, \ldots, k$. The pseudorandom bit sequence made of the $\mathcal{O}(\log \log N)$ least significant bits of the x_i's is cryptographically secure [33], i.e., no polynomial-time statistical test can distinguish it from a perfectly random bitstring of the same length under the assumption that factoring N is intractable. More precisely, if the running time of the statistical test is bounded by T and its advantage[1] is bounded by ϵ, then it was shown by Sidorenko and Schoenmakers [31] that one can securely extract the j least significant bits of each x_i, provided that

$$T \leq \frac{L(n)}{36n(\log n)\delta^{-2}} - 2^{2j+9}n\delta^{-4}, \tag{1}$$

where $\delta = (2^j - 1)^{-1}(\epsilon/(kj))$ and $L(n) = 2.8 \cdot 10^{-3} \exp\left(1.9229(n \ln 2)^{1/3}(\ln(n \ln 2))^{2/3}\right)$ which is the heuristic expected running time of the number field sieve to factor a random n-bit Blum integer. For $n = 1024$, $j = 10$, $k = 10^6$, and $\epsilon = 0.01$, the bound given by (1) is close to -2^{-200}, which is quite meaningless. To obtain a positive bound with this specific choice of j,k, and ϵ, n must be larger than 20000 whereas in practice the typical size for a modulus would be 2048 bits.

2 From Provable Security to Practical Security or the Need for Idealized Models.

The problem illustrated in the last section with the BBS pseudorandom generator finds a solution in the notion of *exact security* [5] or *concrete security* [23]. The adversary against the underlying problem should almost be *as efficient* (both in time and space) than the adversary against the cryptographic scheme it relies on, and should almost reach the same success probability. A scheme that comes with such security arguments achieves *practical security.*

Unfortunately, practical security seems to lead to inefficient cryptographic schemes. To compensate, *idealized models* have been introduced. Among them, one may cite the *random oracle model*, informally introduced by Fiat and Shamir [11] and formalized by Bellare and Rogaway [3], where random functions replace hash functions, and the *ideal cipher model* [2], where block ciphers are replaced by random permutations.

Given the fact that providing security proofs in such idealized models indeed leads to efficient cryptographic schemes, it is natural to wonder whether there is a gap between practical security in these models and practical security in the real world (e.g., where a hash function is SHA-1 [22]), also known as the *standard model.* The first counter example was provided by Canetti, Goldreich, and Halevi [7] who show that there exists encryption and signature schemes which are secure in the random oracle model but that have no secure implementation in the standard model. In other words, a real implementation of the secure ideal schemes would result in an insecure scheme. As far as the author of these notes can judge, the question whether the random oracle model should be preferred to the standard model with strong security assumptions on the underlying primitives is essentially a matter of taste. Yet, it is often stated that the constructions whose purpose is to refute the validity of the random oracle model are not natural and that it would be very unlikely to come up with a "real" construction that would suffer from the same pathology [18, 24].

3 Structuring Convincing Security Proofs using Sequences of Games

A proof of security (just like any kind of proof) should be clear and easy to follow. If it is not elementary, being convincing about the validity of that which is to be demonstrated can be very challenging. Structuring security proofs as a sequence of games is one possibility to provide such proofs.

The notion of security for a cryptographic scheme is usually defined via the description of a game between an *adversary* and a *challenger.* If the adversary wins the game, the security of the scheme is compromised. Both the adversary and the challenger are modeled as probabilistic processes, so that the whole game is modeled as a probability space. Consequently, the fact that the game is won by the adversary corresponds to a specific event S and the scheme is secure when $\Pr[S]$ is close to some target probability (such as 0 or $\frac{1}{2}$). Providing a *tight* bound between $\Pr[S]$ and the target probability is fundamental to provide practical security. Usually, providing such a bound given the sole description of the initial game is hard. One thus constructs a sequence of games Game 0, Game 1,..., Game n, where Game 0 is the original game between the adversary and the challenger. Just as Game 0 defines an event

[1] The advantage of such a test is the absolute value of the probability that it outputs 1 when fed with a random bitstring minus the probability that it outputs 1 when fed with a BBS pseudorandom sequence.

$S_0 = S$, each Game i defines an event S_i such that $\Pr[S_i]$ is negligibly close to $\Pr[S_{i-1}]$ for $i = 1, \ldots, n$. Provided that $\Pr[S_n]$ is easy to compute and negligibly close to the target probability, we are done. Note that if we only consider provable security, "negligibly close to" means "bounded by the inverse of some polynomial in the security parameter". When considering practical security, the final bound should moreover be of practical interest (i.e., be meaningful for practical values of the security parameter).

Transitions between the games should be small to keep the analysis as simple as possible. Transitions are mainly of 3 types [30]:

Transitions based on indistinguishability. Here, if the adversary is able to distinguish between the two games, then it is easy to derive a distinguishing algorithm between two probability distributions that were assumed to be indistinguishable (either computationally or statistically, in the case of information theoretic security), hence a contradiction. The security proof the ElGamal encryption (see Section 7) is the first example in this notes that uses this kind of transition.

Transitions based on failure events. In such a transition, Game i and Game $i+1$ proceed identically *unless* some failure event F occurs, i.e.,

$$S_i \wedge \overline{F} \Longleftrightarrow S_{i+1} \wedge \overline{F}.$$

The following fact is then (almost) inevitably used.

Lemma 1 (Difference Lemma). *Let A, B, and F be three probabilistic events such that $A \wedge \overline{F} \Leftrightarrow B \wedge \overline{F}$. Then $|\Pr[A] - \Pr[B]| \leq \Pr[F]$.*

Proof.

$$|\Pr[A] - \Pr[B]| = |\Pr[A \wedge F] + \Pr[A \wedge \overline{F}] - \Pr[B \wedge F] - \Pr[B \wedge \overline{F}]| = |\Pr[A \wedge F] - \Pr[B \wedge F]| \leq \Pr[F],$$

where the second equality comes from the fact that $\Pr[A \wedge \overline{F}] = \Pr[B \wedge \overline{F}]$ and the third from the fact that both $\Pr[A \wedge F]$ and $\Pr[B \wedge F]$ are real numbers between 0 and $\Pr[F]$. □

Thus, to show that $|\Pr[S_i] - \Pr[S_{i+1}]|$ is negligible (i.e., negligibly close to 0), it is sufficient to show that $\Pr[F]$ is negligible. The computation of the RF/RP-advantage (see Section 8) is the first example in this notes that uses this kind of transition.

Bridging step. This is the most simple kind of steps in which the game is just formulated in a different way, but such that $\Pr[S_i] = \Pr[S_{i+1}]$. The objective is to obtain an equivalent game, but easier to analyse. The computation of the RF/RP-advantage (see Section 8) is the first example in this notes that uses this kind of transition.

4 About of the Rest of this Paper

Sections 5 and 6 respectively introduce the main security scenarios for public key encryption and for digital signatures. Next, each of the following sections describes a secure scheme and provides a proof of its security using sequences of games. The sections are ordered by increasing level difficulty of the proofs, the first examples being fairly simple (toy examples, yet, important results), the last two examples being more technical (as they are based on recent research papers). All the security notions introduced in sections 5 and 6 are illustrated by at least one example in the following sections.

5 Security of Public-Key Encryption Schemes

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 [24]. A public-key encryption scheme is a triplet $(\mathcal{K}, f, f^{-1})$ where

- $\mathcal{K}$ is a key generation algorithm, which on input 1^k (where k is the security parameter) outputs a pair $(\mathsf{pk}, \mathsf{sk})$ of matching public and private keys. The algorithm is probabilistic.
- f is an encryption algorithm that, given a message m and the public key pk outputs a ciphertext $c = f_{\mathsf{pk}}(m)$. The algorithm may be probabilistic.
- f^{-1} is a decryption algorithm that, given a ciphertext c and the secret key sk outputs a plaintext $m = f_{\mathsf{sk}}^{-1}(c)$. The algorithm is deterministic.

The decryption shall "undo" the encryption, i.e., for any message m and any valid public/private key pair $(\mathsf{pk}, \mathsf{sk})$, it should hold that $f_{\mathsf{sk}}^{-1}(f_{\mathsf{pk}}(m)) = m$.

One-Wayness. This is the most basic security notion for a public-key encryption scheme which informally means that only the legitimate secret key holder should be able to decrypt. We describe this notion more formally in Algorithm 1. In this algorithm the adversary $\mathcal{A}$ is a deterministic algorithm that takes as input "random coins" r sampled uniformly from some set R.

```
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}(1^k)$
$r \xleftarrow{\mathsf{r}} R$, $\mathsf{view} \leftarrow \{r, \mathsf{pk}\}$
$m \xleftarrow{\mathsf{r}} \mathcal{M}$, $c \leftarrow f_{\mathsf{pk}}(m)$,$\mathsf{view} \leftarrow \mathsf{view} \cup \{c\}$
$\widehat{m} \leftarrow \mathcal{A}(\mathsf{view})$
if $\widehat{m} = m$ then return 1 else return 0
```

Algorithm 1: One-wayness of a Public-key Encryption Scheme

Denoting S the event that Algorithm 1 returns 1, the success of an adversary $\mathcal{A}$ of breaking the one-wayness (ow) of the public-key scheme S is defined by

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{S}}(\mathcal{A}) = \Pr[S]$$

where the probability holds over the random coins used by the encryption scheme, the internal coins r used by the adversary, and the message m.

Semantic Security. This notion, introduced by Goldwasser and Micali in [13], guarantees that the adversary should not be able to obtain any information about a message given its encryption, *even* if the adversary knows that the plaintext was chosen among a finite set of texts (e.g., if the plaintext is just the encryption of "yes" or "no"). This notion[2] is described in Algorithm 2.

```
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}(1^k)$
$r \xleftarrow{\mathsf{r}} R$, $\mathsf{view} \leftarrow \{r, \mathsf{pk}\}$
$(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{view})$
$b \xleftarrow{\mathsf{r}} \{0,1\}$, $c \leftarrow f_{\mathsf{pk}}(m_b)$,$\mathsf{view} \leftarrow \mathsf{view} \cup \{c\}$
$\widehat{b} \leftarrow \mathcal{A}(\mathsf{view})$
if $\widehat{b} = b$ then return 1 else return 0
```

Algorithm 2: Semantic Security of a Public-key Encryption Scheme

Denoting S the event that Algorithm 2 returns 1, the success of an adversary $\mathcal{A}$ of breaking the semantic security (ss) of the public-key encryption scheme S is defined by

$$\mathsf{Succ}^{\mathsf{ss}}_{\mathsf{S}}(\mathcal{A}) = |\Pr[S] - 1/2|.$$

Adaptive Chosen Ciphertext Attacks (CCA). Semantic security is not sufficient when considering *active* adversaries (i.e, that don't only eavesdrop but also injects messages). To deal with active adversaries, Rackoff and Simon introduced the notion of adaptive chosen ciphertext attack [26]. Here, the adversary has access to a "decryption oracle" that she/he can query to obtain the decryption of any ciphertext. Given a "target" ciphertext (different from those that were submitted to the decryption oracle), the adversary must not be able to extract any information about the corresponding plaintext. The attack is adaptive in the sense that the adversary is allowed to query the decryption oracle even *after* she/he obtained the target ciphertext (provided, of course, that this target ciphertext is not submitted to the decryption oracle). This notion is described in Algorithm 3.

Denoting S the event that Algorithm 3 returns 1, the success of an adversary $\mathcal{A}$ of breaking the CCA security (cca) of the public-key encryption scheme S is defined by

$$\mathsf{Succ}^{\mathsf{cca}}_{\mathsf{S}}(\mathcal{A}) = |\Pr[S] - 1/2|.$$

In Algorithm 3 it is understood that the adversary is bounded in the number of oracle queries (i.e., the loop in the `Oracle_Queries` function eventually ends). In other words, when studying the security of a particular scheme, there will be some explicit bound on this number of queries, on which the final security bound will depend.

[2] The definition we give for semantic security is actually that of a different (but equivalent) one, called ciphertext indistinguishability, also introduced by Goldwasser and Micali in [13].

```
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}(1^k)$ /* Global Vars */
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r, \mathsf{pk}\}$
Oracle_Queries($\mathcal{A}, \mathsf{view}, \perp$)
$(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{view})$, $(y^\star, \mathsf{bit}) \leftarrow$ Encryption_Oracle($m_0, m_1$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{y^\star\}$
Oracle_Queries($\mathcal{A}, \mathsf{view}, y^\star$)
$\widehat{\mathsf{bit}} \leftarrow \mathcal{A}(\mathsf{view})$
if $\widehat{\mathsf{bit}} = \mathsf{bit}$ then return 1 else return 0

function Oracle_Queries($\mathcal{A}, \mathsf{view}, y^\star$)
    loop
        $y \leftarrow \mathcal{A}(\mathsf{view})$ such that $y \neq y^\star$, $m \leftarrow$ Decryption_Oracle($y$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{m\}$
    end

function Decryption_Oracle($y$)
    $m \leftarrow f_{\mathsf{sk}}^{-1}(y)$, return $m$

function Encryption_Oracle($m_0, m_1$)
    $\mathsf{bit} \xleftarrow{r} \{0,1\}$, $m^\star \leftarrow m_{\mathsf{bit}}$, $y^\star \leftarrow f_{\mathsf{pk}}(m^\star)$, return $(y^\star, \mathsf{bit})$
```
Algorithm 3: CCA Security of a Public-key Encryption Scheme

Other Security Notions. For an in-depth study of the security relations between different security criteria of public-key encryption schemes, we refer to [1].

6 Security of Digital Signature Schemes

The aim of a digital signature scheme is to allow Alice to sign any document with her private key, the correctness of this signature being verifiable by anybody using Alice's public key. Intuitively, it should be impossible to *forge* a signature, i.e., without the knowledge of Alice's private key, it should not be possible to sign messages on behalf of her. A digital signature scheme is a triplet $(\mathcal{K}, \mathsf{sig}, \mathsf{ver})$ where

– $\mathcal{K}$ is a key generation algorithm, which on input 1^k (where k is the security parameter) outputs a pair $(\mathsf{pk}, \mathsf{sk})$ of matching public and private keys. The algorithm is probabilistic.

– sig is the signing algorithm that, given a message m and the secret key sk outputs a signature $\sigma = \mathsf{sig}_{\mathsf{sk}}(m)$ of the message m. The algorithm may be probabilistic.

– ver is verification algorithm that, given a message m, a signature σ, and the public key pk, checks whether the signature is valid (in that case $\mathsf{ver}_{\mathsf{pk}}(m, \sigma)$ returns 1) or not (in that case $\mathsf{ver}_{\mathsf{pk}}(m, \sigma)$ returns 0). This algorithm is deterministic.

For any valid public/private key pair $(\mathsf{pk}, \mathsf{sk})$, any message m, and any signature $\sigma = \mathsf{sig}_{\mathsf{sk}}(m)$, it should *always* hold that $\mathsf{ver}_{\mathsf{pk}}(m, \sigma) = 1$. Moreover, it should be impossible for an adversary to compute a valid signature on Alice's behalf, without the knowledge of her private key. Several kind of adversaries can be considered, with different goals. In these notes we will only consider one kind of security notion, namely existential unforgeability (euf) under chosen-message attack (cma) [14], defined by Algorithm 4.

```
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathcal{K}(1^k)$ /* Global Vars */
$r \xleftarrow{\mathsf{r}} R$, $\mathsf{view} \leftarrow \{r, \mathsf{pk}\}$
Oracle_Queries($\mathcal{A}, \mathsf{view}$)
$(m^\star, \sigma^\star) \leftarrow \mathcal{A}(\mathsf{view})$
return $\mathsf{ver}_{\mathsf{pk}}(m^\star, \sigma^\star)$ /* returns 1 if the signature is valid, 0 otherwise */

function Oracle_Queries($\mathcal{A}, \mathsf{view}$)
    loop
        $m \leftarrow \mathcal{A}(\mathsf{view})$, $\sigma \leftarrow$ Signing_Oracle($m$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{\sigma\}$
    end

function Signing_Oracle($m$)
    $\sigma \leftarrow \mathsf{sig}_{\mathsf{sk}}(m)$, return $\sigma$
```

Algorithm 4: Existential unforgeability (euf) against chosen-message attack (cma) of a digital signature scheme.

Denoting S the event that Algorithm 4 returns 1, the success of an adversary $\mathcal{A}$ of forging a valid message/signature pair (euf) under a chosen-message attack against the digital signature scheme S is

$$\mathsf{Succ}^{\mathsf{euf}}_{\mathsf{S}}(\mathcal{A}) = \Pr[S].$$

In Algorithm 4 it is understood that $m^\star$ should not have been queried to the signing oracle[3], and that the number of signing queries of the adversary is upper bounded. The security bound on the success of a particular adversary will depend on this bound.

7 ElGamal Encryption

We show in this section that the ElGamal public-key encryption scheme is semantically secure under the decisional Diffie-Hellman (DDH) assumption. In this section, G is a cyclic group of prime order q, and γ is an arbitrary generator of G.

7.1 Preliminaries

The Decisional Diffie-Hellman (DDH) assumption. Let $\mathcal{D}$ be a distinguishing algorithm that takes a triplet of elements of G as an input and outputs a bit. The DDH advantage of $\mathcal{D}$ is defined by

$$\mathsf{DDHAdv}(\mathcal{D}) = |\Pr_{x,y}[\mathcal{D}(\gamma^x, \gamma^y, \gamma^{xy}) = 1] - \Pr_{x,y,z}[\mathcal{D}(\gamma^x, \gamma^y, \gamma^z) = 1]|,$$

where x, y, z are random elements of $\mathbf{Z}_q$. The DDH assumption (for G) is the assumption that $\mathsf{DDHAdv}(\mathcal{D})$ is negligible for any $\mathcal{D}$.

The ElGamal public-key encryption scheme. The key generation algorithm computes the public/private key pair as follows:

$$x \xleftarrow{\mathsf{r}} \mathbf{Z}_q,\ \alpha \leftarrow \gamma^x,\ \mathsf{pk} \leftarrow \alpha,\ \mathsf{sk} \leftarrow x.$$

Given a message $m \in G$, the encryption algorithm computes the ciphertext c as follows:

$$y \xleftarrow{\mathsf{r}} \mathbf{Z}_q,\ \beta \leftarrow \gamma^y,\ \delta \leftarrow \alpha^y,\ \chi \leftarrow \delta \cdot m, c \leftarrow (\beta, \chi).$$

Given a ciphertext $c = (\beta, \chi) \in G^2$, the decryption algorithm recovers the plaintext m as follows:

$$m \leftarrow \chi/\beta^x.$$

The decryption "undoes" the encryption as

$$\chi/\beta^x = (\delta \cdot m)/(\gamma^y)^x = (\alpha^y \cdot m)/\gamma^{xy} = ((\gamma^x)^y \cdot m)/\gamma^{xy} = (\gamma^{xy} \cdot m)/\gamma^{xy} = m.$$

Finally, note that the security parameter corresponds to the bit-length of the group order q.

[3] A stronger security notion, called non-malleability, requires that the signature was not obtained from the signing oracle.

7.2 Security Proof

Game 0: This game corresponds to the definition of an adversary $\mathcal{A}$ against the semantic security of the ElGamal encryption scheme.

```
$x \xleftarrow{r} \mathbf{Z}_q$, $\alpha = \gamma^x$, $\mathsf{pk} \leftarrow \alpha$, $\mathsf{sk} \leftarrow x$
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r, \mathsf{pk}\}$
$(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{view})$
$b \xleftarrow{r} \{0,1\}$, $y \xleftarrow{r} \mathbf{Z}_q$, $\beta \leftarrow \gamma^y$, $\delta \leftarrow \alpha^y$, $\chi \leftarrow \delta \cdot m_b$, $c \leftarrow (\beta, \chi)$, $\mathsf{view} \leftarrow \mathsf{view} \cup \{c\}$
$\widehat{b} \leftarrow \mathcal{A}(\mathsf{view})$
if $\widehat{b} = b$ then return 1 else return 0
```

Denoting S_0 the event that Game 0 returns 1, we have

$$\mathsf{Succ}^{\mathsf{ss}}_{\mathsf{ElGamal}}(\mathcal{A}) = |\Pr[S_0] - 1/2|. \tag{2}$$

Game 1: [Transition based on indistinguishability.] Instead of computing δ as $\alpha^y = \gamma^{xy}$, we now compute it as γ^z, where z is sampled uniformly from $\mathbf{Z}_q$.

```
$x \xleftarrow{r} \mathbf{Z}_q$, $\alpha = \gamma^x$, $\mathsf{pk} \leftarrow \alpha$, $\mathsf{sk} \leftarrow x$
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r, \mathsf{pk}\}$
$(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{view})$
$b \xleftarrow{r} \{0,1\}$, $y \xleftarrow{r} \mathbf{Z}_q$, $\beta \leftarrow \gamma^y$, $\boxed{z \xleftarrow{r} \mathbf{Z}_q,\ \delta \leftarrow \gamma^z,}$ $\chi \leftarrow \delta \cdot m_b$, $c \leftarrow (\beta, \chi)$, $\mathsf{view} \leftarrow \mathsf{view} \cup \{c\}$
$\widehat{b} \leftarrow \mathcal{A}(\mathsf{view})$
if $\widehat{b} = b$ then return 1 else return 0
```

Denoting S_1 the event that Game 0 returns 1, we claim that

$$|\Pr[S_1] - \Pr[S_0]| = \mathsf{DDHAdv}(\mathcal{D}), \tag{3}$$

for some distinguishing algorithm $\mathcal{D}$. To prove this claim, let us first define $\mathcal{D}$ as follows:

```
Distinguisher $\mathcal{D}(\alpha, \beta, \delta)$
    $\mathsf{pk} \leftarrow \alpha$
    $r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r, \mathsf{pk}\}$
    $(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{view})$
    $b \xleftarrow{r} \{0,1\}$, $\chi \leftarrow \delta \cdot m_b$, $c \leftarrow (\beta, \chi)$, $\mathsf{view} \leftarrow \mathsf{view} \cup \{c\}$
    $\widehat{b} \leftarrow \mathcal{A}(\mathsf{view})$
    if $\widehat{b} = b$ then return 1 else return 0
```

If the input of the previous algorithm is of the form $(\gamma^x, \gamma^y, \gamma^{xy})$, then computation proceeds just as in Game 0, so that

$$\Pr[\mathcal{D}(\gamma^x, \gamma^y, \gamma^{xy}) = 1] = \Pr[S_0].$$

If the input is of the form $(\gamma^x, \gamma^y, \gamma^z)$, then computation proceeds just as in Game 1, so that

$$\Pr[\mathcal{D}(\gamma^x, \gamma^y, \gamma^z) = 1] = \Pr[S_1].$$

Based on the adversary $\mathcal{A}$, we obtained the description of a distinguisher $\mathcal{D}$ against the DDH assumption such that

$$|\Pr[S_1] - \Pr[S_0]| = \mathsf{DDHAdv}(\mathcal{D}).$$

Moreover, it is easy to see that $\Pr[S_1] = 1/2$: we first note that (in Game 1) $b, r, \mathsf{pk}, \beta, \delta$ are uniformly distributed random variables that are mutually independent, and thus, so are $b, r, \mathsf{pk}, \beta, \chi$ (the argument is identical to the one that shows that the distribution of a ciphertext produced by the one-time pad is uniform, regardless of the plaintext distribution). Consequently b and $\widehat{b} \leftarrow \mathcal{A}(\mathtt{view})$ are mutually independent so that $\Pr[S_1] = 1/2$. From this, and from equations (2) and (3), we conclude that

$$\mathsf{Succ}^{\mathsf{ss}}_{\mathsf{ElGamal}}(\mathcal{A}) = \mathsf{DDHAdv}(\mathcal{D})$$

for some distinguisher $\mathcal{D}$, which is negligible under the DDH assumption. This completes the proof.

8 Random Function vs. Random Permutation

In the ideal cipher model [2], block ciphers are replaced by random permutations. In certain circumstances, it is easier to study the security of a cryptographic scheme by replacing the random *permutations* by random *functions*. The RF/RP-Lemma allows to evaluate in which cases such a switch is acceptable by computing the advantage of an adversary trying to distinguish between a random permutation and a random function of the same (co)domain.

8.1 Preliminaries

We denote by Γ_ℓ the set of all functions from $\{0,1\}^\ell$ to $\{0,1\}^\ell$ and by Π_ℓ the set of all permutations on $\{0,1\}^\ell$. We consider an adversary $\mathcal{A}$ who is either given a black-box (oracle) access to a uniformly distributed random function $F \in \Gamma_\ell$ or to a uniformly distributed random permutation $P \in \Pi_\ell$. We respectively denote this adversary $\mathcal{A}^F$ and $\mathcal{A}^P$. The RF/RP-advantage of this adversary is defined by

$$|\mathrm{Pr}_F[\mathcal{A}^F = 1] - \mathrm{Pr}_P[\mathcal{A}^P = 1]|.$$

We will show that the advantage of an adversary that makes at most q queries to the black box is bounded by

$$\frac{q^2}{2} \cdot 2^{-\ell}.$$

Without loss of generality, we assume that the adversary makes *exactly* q queries and that all of them are distinct (a duplicate query would be useless as it would not provide any additional information that the adversary would not already have).

8.2 Security Proof

Game 0: This game represents the computations made by an adversary $\mathcal{A}$ having a black-box (oracle) access to a random permutation $P \in \Pi_\ell$.

```
$P \xleftarrow{r} \Pi_\ell$ /* Global Var */
$r \xleftarrow{r} R$, view $\leftarrow \{r\}$
for $i = 1, \ldots, q$ do
    $x_i \leftarrow \mathcal{A}$(view), $y_i \leftarrow$ Oracle($x_i$), view $\leftarrow$ view $\cup \{y_i\}$
end
bit $\leftarrow \mathcal{A}$(view), return bit

function Oracle($x$)
    $y \leftarrow P(x)$
    return $y$
```

Denoting S_0 the event that the previous algorithm returns 1, we have

$$\mathrm{Pr}_P[\mathcal{A}^P = 1] = \Pr[S_0]. \tag{4}$$

Game 1: [Bridging Step.] We now introduce a technique that will be overused in the rest of these notes. We replace the random permutation by a table that grows while the adversary makes queries to the oracle. The table, initially empty, keeps track of the input/output values of the simulated permutation. Given an input value x on which the oracle has not been queried yet, a random value is chosen for y, the pair (x, y) is inserted in the table, and y is returned. If the input value x matches some entry in the table, the corresponding y is returned. This would perfectly simulate a random *function*. As we are dealing with a random *permutation*, we should also make sure that two distinct queries receive two distinct answers. This is described more formally in Game 1 (recall that we assumed that the adversary never makes the same query twice, so that we do not actually need to keep track of x in the list here). This technique is sometimes called “lazy sampling”.

```
$\boxed{\mathsf{List} \leftarrow \emptyset}$ /* Global Var */
$r \xleftarrow{r} R$, view $\leftarrow \{r\}$
for $i = 1, \ldots, q$ do
    $x_i \leftarrow \mathcal{A}(\mathsf{view})$, $y_i \leftarrow \mathtt{Oracle}(x_i)$, view $\leftarrow$ view $\cup \{y_i\}$
end
bit $\leftarrow \mathcal{A}(\mathsf{view})$, return bit

function Oracle($x$)
    | $Y \xleftarrow{r} \{0,1\}^\ell$
    | if $Y \in$ List then $y \xleftarrow{r} \{0,1\}^\ell \setminus$List else $y \leftarrow Y$
    | List $\leftarrow$ List $\cup \{y\}$
    return $y$
```

It should be clear that this change is purely conceptual and doesn't change anything from the point of view of the adversary. Consequently, if we denote by S_1 the event the Game 1 returns 1,

$$\Pr[S_1] = \Pr[S_0]. \tag{5}$$

Game 2: [Transition based on a failure event.] We now drop the consistency check in the simulation of the random permutation, so that it actually becomes a simulation of a random function, i.e., letting S_2 be the event that Game 2 returns 1,

$$\Pr[S_2] = \Pr_F[\mathcal{A}^F = 1], \tag{6}$$

where $F \in \Gamma_\ell$ is a random function.

```
List $\leftarrow \emptyset$ /* Global Var */
$r \xleftarrow{r} R$, view $\leftarrow \{r\}$
for $i = 1, \ldots, q$ do
    $x_i \leftarrow \mathcal{A}(\mathsf{view})$, $y_i \leftarrow \mathtt{Oracle}(x_i)$, view $\leftarrow$ view $\cup \{y_i\}$
end
bit $\leftarrow \mathcal{A}(\mathsf{view})$, return bit

function Oracle($x$)
    $Y \xleftarrow{r} \{0,1\}^\ell$
    $\boxed{y \leftarrow Y}$
    List $\leftarrow$ List $\cup \{y\}$
    return $y$
```

Let F be the event that $Y \in$ `List` at least once during the execution of Game 1. It is obvious that Game 1 and Game 2 proceed identically *unless* the event F occurs. By the difference lemma,

$$|\Pr[S_2] - \Pr[S_1]| \le \Pr[F]. \tag{7}$$

We denote Y_i the value sampled by `Oracle` to answer the ith query in Game 1. Let $\{y_1, y_2, \ldots, y_q\}$ be a list of q *distinct* elements of $\{0,1\}^\ell$ at let $Y \in \{0,1\}^\ell$ be a uniformly distributed random variable. We have that

$$\begin{aligned}
\Pr[F] &= \Pr[Y_2 \in \{Y_1\} \vee Y_3 \in \{Y_1, Y_2\} \vee \cdots \vee Y_q \in \{Y_1, \ldots, Y_{q-1}\}] \\
&\le \Pr[Y_2 \in \{Y_1\}] + \Pr[Y_3 \in \{Y_1, Y_2\}] + \cdots + \Pr[Y_q \in \{Y_1, \ldots, Y_{q-1}\}] \\
&\le \Pr_Y[Y \in \{y_1\}] + \Pr_Y[Y \in \{y_1, y_2\}] + \cdots + \Pr_Y[Y \in \{y_1, y_2, \ldots, y_{q-1}\}] \\
&= \frac{1}{2^\ell} + \frac{2}{2^\ell} + \cdots + \frac{q-1}{2^\ell} = \frac{q(q-1)}{2} \cdot 2^{-\ell} \le \frac{q^2}{2} \cdot 2^{-\ell}.
\end{aligned}$$

The first inequality comes from the union bound, the second from the fact that $\Pr[Y_i \in \{Y_1, \ldots, Y_{i-1}\}] \le \Pr[Y_i \in \{y_1, \ldots, y_{i-1}\}]$ as the $y_1, \ldots, y_{i-1}$ are distinct values (which might not be the case for the Y_i's). From this bound and from equations (4), (5), (6), and (7) we deduce the announced result.

9 The Luby-Rackoff Construction

When studying the security of the Data Encryption Standard (DES) [21], Luby and Rackoff proved that a 3 rounds Feistel network (on which the DES is based) can generate a pseudo-random permutation out of three mutually independent random functions.

9.1 Preliminaries

A family $\mathcal{P} = \{P_k\}_{k\in K} \subset \Pi_\ell$ is said to be *pseudo-random* if it is hard to distinguish between a random instance of that family and a random instance of Π_ℓ. More formally, let $\mathcal{A}$ be an adversary who is given an oracle access to a random permutation P_k of $\mathcal{P}$ or to a random permutation P^* of Π_ℓ. The PRP-advantage of $\mathcal{A}$ is defined by

$$|\Pr[\mathcal{A}^{P_k} = 1] - \Pr[\mathcal{A}^{P^*} = 1]|.$$

The family $\mathcal{P}$ is said to be *pseudo-random* if any adversary's PRP-advantage is negligible.

A Feistel scheme allows to build a permutation from a round functions. A three round Feistel scheme based on the functions $f_1, f_2, f_3 \in \Gamma_\ell$ is a permutation in $\Pi_{2\ell}$, usually denoted $\Psi(f_1, f_2, f_3)$, and defined as follows: On input $(u, v) \in \{0,1\}^\ell \times \{0,1\}^\ell$, let

$$\begin{aligned} w &\leftarrow u \oplus f_1(v) \\ x &\leftarrow v \oplus f_2(w) \\ y &\leftarrow w \oplus f_3(x), \end{aligned}$$

the output is $(x, y) \in \{0,1\}^\ell \times \{0,1\}^\ell$. It is easy to see that $\Psi(f_1, f_2, f_3)$ is permutation by checking that $\Psi^{-1}(f_1, f_2, f_3) = \Psi(f_3, f_2, f_1)$. The Luby-Rackoff construction consists in a three round Feistel scheme with 3 mutually independent random functions. In the next section, we show that the family $\mathcal{LR} = \{\Psi(F, G, H) : F, G, H \in \Gamma_\ell\} \subset \Pi_{2\ell}$ is pseudo-random, provided that $2^{-\ell}$ is negligible. More precisely, we show that the PRP-advantage of any adversary $\mathcal{A}$ who is given access to a random permutation $P \in \mathcal{LR}$ or to a random permutation $P^* \in \Pi_\ell$ is bounded by $\frac{3}{2} \cdot q^2 \cdot 2^{-\ell}$.

9.2 Security Proof

Game 0: This game represents the computation of an adversary $\mathcal{A}$ who is given an oracle access to a random Luby-Rackoff instance. As in Section 8.1, we assume that the adversary makes *exactly q* encryption queries, and that all the queries are *distinct* from each other. In the following game, this translates in $(u_i, v_i) \neq (u_j, v_j)$ for $i \neq j$.

```
$F, G, H \xleftarrow{r} \Gamma_\ell$ /* Global Var */
$r \xleftarrow{r} R$, view $\leftarrow \{r\}$
for $i = 1, \ldots, q$ do
    $(u_i, v_i) \leftarrow \mathcal{A}(\text{view})$, $(x_i, y_i) \leftarrow$ Oracle$(u_i, v_i)$, view $\leftarrow$ view $\cup \{(x_i, y_i)\}$
end
bit $\leftarrow \mathcal{A}(\text{view})$, return bit

function Oracle$(u, v)$
    $w \leftarrow u \oplus F(v)$
    $x \leftarrow v \oplus G(w)$
    $y \leftarrow w \oplus H(x)$
    return $(x, y)$
```

Denoting S_0 the event that Game 0 returns 1, we have

$$\Pr[S_0] = \Pr[\mathcal{A}^P = 1], \tag{8}$$

where P is a random instance of Luby-Rackoff construction.

Game 1: [Bridging Step.] In this game, we adopt the lazy-sampling technique for both random functions G and H (similarly to what we did in Game 1 in Section 8.2, except that we deal with a simple case here, as we consider random *functions*).

```
$\boxed{\mathsf{GList}, \mathsf{HList} \leftarrow \emptyset,\ F \xleftarrow{r} \Gamma_\ell}$ /* Global Var */
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r\}$
for $i = 1, \ldots, q$ do
    $(u_i, v_i) \leftarrow \mathcal{A}(\mathsf{view})$, $(x_i, y_i) \leftarrow \mathtt{Oracle}(u_i, v_i)$, $\mathsf{view} \leftarrow \mathsf{view} \cup \{(x_i, y_i)\}$
end
$\mathsf{bit} \leftarrow \mathcal{A}(\mathsf{view})$, return $\mathsf{bit}$

function Oracle$(u, v)$
    $w \leftarrow u \oplus F(v)$
    if $(w, g) \in \mathsf{GList}$ then $x \leftarrow v \oplus g$ else $g \xleftarrow{r} \{0,1\}^\ell$, $\mathsf{GList} \leftarrow \mathsf{GList} \cup \{(w, g)\}$, $x \leftarrow v \oplus g$
    if $(x, h) \in \mathsf{HList}$ then $y \leftarrow w \oplus h$ else $h \xleftarrow{r} \{0,1\}^\ell$, $\mathsf{HList} \leftarrow \mathsf{HList} \cup \{(x, h)\}$, $y \leftarrow w \oplus h$
    return $(x, y)$
```

To save us some space in the algorithm, we do not explicitly write that searching for $(w, g) \in$ GList or for $(x, h) \in$ HList is respectively done "for some g" and "for some h". Denoting S_1 the event that Game 0 returns 1, we have

$$\Pr[S_1] = \Pr[S_0]. \tag{9}$$

Game 2: [Transition based on a failure event.] We eliminate the consistency checks in the encryption oracle. As we do not need GList nor HList in this case, we remove them from the game description to simplify it a little bit.

```
$\boxed{F \xleftarrow{r} \Gamma_\ell}$ /* Global Var */
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r\}$
for $i = 1, \ldots, q$ do
    $(u_i, v_i) \leftarrow \mathcal{A}(\mathsf{view})$, $(x_i, y_i) \leftarrow \mathtt{Oracle}(u_i, v_i)$, $\mathsf{view} \leftarrow \mathsf{view} \cup \{(x_i, y_i)\}$
end
$\mathsf{bit} \leftarrow \mathcal{A}(\mathsf{view})$, return $\mathsf{bit}$

function Oracle$(u, v)$
    $w \leftarrow u \oplus F(v)$
    $g \xleftarrow{r} \{0,1\}^\ell$, $x \leftarrow v \oplus g$
    $h \xleftarrow{r} \{0,1\}^\ell$, $y \leftarrow w \oplus h$
    return $(x, y)$
```

In this new game, the oracle simply outputs a fresh random value of $\{0,1\}^{2\ell}$ for each query. As we assumed that the q queries of the adversary are distinct from each other, this means that the oracle behaves like a random function of Γ_ℓ. Denoting S_2 the event that Game 2 outputs 1, we thus have

$$\Pr[S_2] = \Pr[\mathcal{A}^{F^*} = 1], \tag{10}$$

where $F^* \in \Gamma_\ell$ is a random function.

We denote by $(u_1, v_1), \ldots, (u_q, v_q)$ the q queries made to the oracle, and similarly denote w_i, g_i, h_i, x_i, y_i the values corresponding to the query (u_i, v_i). Let F_1 be the event that $w_i = w_j$ for some $i \neq j$ in Game 2. Let F_2 be the event that $x_i = x_j$ for some $i \neq j$ in Game 2. It is simple to see that Game 2 proceed identically to Game 1, *unless* the event $F_1 \vee F_2$ occurs. By the difference lemma and the union bound, we have

$$|\Pr[S_2] - \Pr[S_1]| \leq \Pr[F_1 \vee F_2] \leq \Pr[F_1] + \Pr[F_2] \leq \sum_{i<j} (\Pr[w_i = w_j] + \Pr[x_i = x_j]). \tag{11}$$

For any $i \neq j$ we have $\Pr[w_i = w_j] = \Pr[u_i \oplus F(v_i) = u_j \oplus F(v_j)]$. As we assumed that the adversary does not ask the same query twice,

- either $u_i \neq u_j$ and $v_i = v_j$, in such a case we have $\Pr[u_i \oplus F(v_i) = u_j \oplus F(v_j)] = 0$,
- or $v_i \neq v_j$, and we have $\Pr[u_i \oplus F(v_i) = u_j \oplus F(v_j)] = 2^{-\ell}$.

Thus $\Pr[w_i = w_j] \leq 2^{-\ell}$. For any $i \neq j$, we moreover have $\Pr[x_i = x_j] = \Pr[v_i \oplus g_i = v_j \oplus g_j] = 2^{-\ell}$, as g_i, g_j are mutually independent random variables. With this and from (11) we obtain that

$$|\Pr[S_2] - \Pr[S_1]| \leq q^2 \cdot 2^{-\ell}. \tag{12}$$

From equations (8), (9), (10), (11), and (12), we deduce that

$$|\Pr[\mathcal{A}^P = 1] - \Pr[\mathcal{A}^{F^*} = 1]| \leq q^2 \cdot 2^{-\ell}.$$

Finalizing the proof using the RF/RP-Lemma: To conclude, we make use of the result of Section 8. The advantage of $\mathcal{A}$ of distinguishing P from P^* is

$$|\Pr[\mathcal{A}^P = 1] - \Pr[\mathcal{A}^{P^*} = 1]| \leq |\Pr[\mathcal{A}^P = 1] - \Pr[\mathcal{A}^{F^*} = 1]| + |\Pr[\mathcal{A}^{F^*} = 1] - \Pr[\mathcal{A}^{P^*} = 1]| \leq \frac{3}{2} \cdot q^2 \cdot 2^{-\ell}.$$

This completes the proof.

10 Full-Domain Hash

The Full-Domain Hash (FDH) [3] is a provably secure signature scheme in the random oracle model. The first security proof was initially proposed by Bellare and Rogaway in [5] and was later improved by Coron in [9] (see Section 11 for Coron's proof).

10.1 Preliminaries

The Full Domain Hash (FDH) signature scheme is defined as follows. On input 1^k (where k is the security parameter), the key generation algorithm computes RSA parameters $n = p \cdot q, e, d$ where p, q are $k/2$-bit primes and where $e \cdot d \equiv 1 \pmod{\varphi(n)}$. It outputs $(\mathsf{pk}, \mathsf{sk})$ where $\mathsf{pk} = (n, e)$ and $\mathsf{sk} = (n, d)$. Both the signing and the verifying algorithm have oracle access to a hash function $H : \{0,1\}^* \to \mathbf{Z}_n^*$. On input m, the signing algorithm outputs the signature $\sigma = H(m)^d \bmod n$. On input (m, σ), the verifying algorithm outputs 1 if $\sigma^e \bmod n = H(m)$ and 0 otherwise.

Theorem 2. *Let $\mathcal{A}$ be an adversary performing a chosen-message attack against the Full Domain Hash in the random oracle model, with security parameter k. Let q_s and q_h denote the number of queries made by $\mathcal{A}$ to the signing oracle and to the hash oracle respectively. Let $\mathsf{Succ}^{\mathsf{euf}}_{\mathsf{fdh}}(\mathcal{A})$ be the success probability of $\mathcal{A}$ to produce an existential forgery in time t. Then there exists an adversary $\mathcal{A}'$ that breaks the one-wayness of RSA with probability of success $\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{rsa}}(\mathcal{A}')$ in time t' where*

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{rsa}}(\mathcal{A}') = \frac{1}{q_h} \cdot \mathsf{Succ}^{\mathsf{euf}}_{\mathsf{fdh}}(\mathcal{A}) \quad \text{and} \quad t' = t + q_h \cdot \mathcal{O}(k^3).$$

A proof of this result is provided in Section 10.3.

10.2 Discussion

As noted by authors themselves, the bound is not satisfactory. Indeed, whereas it is easy in practice to limit the number of signing query, it is not possible to limit the number of hash queries. We should assume that $q_h \gg q_s$. If the adversary is allowed to ask, say, $q_h = 2^{60}$ hash queries (in practice, this corresponds to hash 2^{60} times with SHA-1 [22] or MD5 [27]), and if the success probability of inverting RSA is 2^{-61}, Theorem 2 says that the forging probability is 1/2, which is too much. If we only had this security result available, we would have to increase the size of the security parameter k to lower the probability of inverting RSA. The drawback of this solution is that it would decrease the efficiency of the scheme.

At first sight, it might be surprising that the bound given in Theorem 2 does not depend on q_s. In fact, the original bound proposed in [5] does, as it shows how to construct an adversary $\mathcal{A}'$ such that

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{rsa}}(\mathcal{A}') = \frac{1}{q_h + q_s + 1} \cdot \mathsf{Succ}^{\mathsf{euf}}_{\mathsf{fdh}}(\mathcal{A}) \quad \text{and} \quad t' = t + (q_h + q_s + 1) \cdot \mathcal{O}(k^3).$$

Therefore, our bound slightly improves on that of the original proof, but the gain is negligible. For reasons mentioned in the previous paragraph, we should consider that $q_h \gg q_s$ so that both bounds are equivalent. Coron's proof [9] is a real improvement in the sense that it replaces the factor q_h by q_s in the bound on the success probability of $\mathcal{A}'$ (see Section 11 for a full treatment of Coron's proof).

10.3 Proof of Theorem 2

Throughout this proof, we denote by S_i the probability that Game i returns 1.

Game 0: This game exactly corresponds to the *existential unforgeability* (euf) game under chosen-message attack performed by an adversary $\mathcal{A}$. In the game, we denote by $\mathcal{H}$ the set of all functions from $\{0,1\}^*$ to $\mathbf{Z}_n^*$. We have

$$\mathsf{Succ}^{\mathsf{euf}}_{\mathsf{fdh}}(\mathcal{A}) = \Pr[S_0]. \tag{13}$$

Note that as the hash oracle is queried $q_h + q_s + 1$ times in total (q_h times by the adversary, q_s by the signing oracle, and one last time at the end of the game).

```
$(n, e, d) \leftarrow \mathsf{RSA}(1^k)$, $H \xleftarrow{r} \mathcal{H}$, $i, j \leftarrow 0$ /* Global Vars */
$r \xleftarrow{r} R$, view $\leftarrow \{r, n, e\}$
Oracle_Queries($\mathcal{A}$, view)
$(m^\star, \sigma^\star) \leftarrow \mathcal{A}$(view)
$h \leftarrow H(m^\star)$
if $\sigma^\star = h^d \bmod n$ then return 1 else return 0

function Oracle_Queries($\mathcal{A}$, view)
    loop
        $m_{i+j} \leftarrow \mathcal{A}$(view)
        do either
            if $i < q_h$ then  $h_{i+j} \leftarrow$ Hash_Oracle($m_{i+j}$), view $\leftarrow$ view $\cup \{h_{i+j}\}$, $i \leftarrow i + 1$
        or
            if $j < q_s$ then  $\sigma_{i+j} \leftarrow$ Signing_Oracle($m_{i+j}$), view $\leftarrow$ view $\cup \{\sigma_{i+j}\}$, $j \leftarrow j + 1$
        done
    end

function Hash_Oracle($m$)
    $h \leftarrow H(m)$, return $h$

function Signing_Oracle($m$)
    $h \leftarrow$ Hash_Oracle($m$), $\sigma \leftarrow h^d \bmod n$
    return $\sigma$
```

Game 1: [Bridging Step.] In this game, we adopt the lazy-sampling technique for the random function H. Note that searching for $(*, m, *, h) \in$ `HList` is done "for some h". We have

$$\Pr[S_1] = \Pr[S_0]. \tag{14}$$

```
$(n,e,d) \leftarrow \mathsf{RSA}(1^k)$, $\boxed{\mathsf{HList} \leftarrow \emptyset,}$ $i,j \leftarrow 0$ /* Global Vars */
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r,n,e\}$
Oracle_Queries($\mathcal{A}$, $\mathsf{view}$)
$(m^\star,\sigma^\star) \leftarrow \mathcal{A}(\mathsf{view})$
$\boxed{\textbf{if } (*,m^\star,*,h^\star) \in \mathsf{HList} \textbf{ then } h \leftarrow h^\star \textbf{ else } h \xleftarrow{r} \mathbf{Z}_n^*}$
if $\sigma^\star = h^d \bmod n$ then return 1 else return 0

function Oracle_Queries($\mathcal{A}$, $\mathsf{view}$)
    loop
        $m_{i+j} \leftarrow \mathcal{A}(\mathsf{view})$
        do either
            if $i < q_h$ then  $h_{i+j} \leftarrow$ Hash_Oracle($m_{i+j}$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{h_{i+j}\}$, $i \leftarrow i+1$
        or
            if $j < q_s$ then  $\sigma_{i+j} \leftarrow$ Signing_Oracle($m_{i+j}$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{\sigma_{i+j}\}$, $j \leftarrow j+1$
        done
    end

function Hash_Oracle($m$)
    ┌────────────────────────────────────────────────────────────────────────────┐
    │ if $(*,m,*,h) \in \mathsf{HList}$ then return $h$                          │
    │ else  $h \xleftarrow{r} \mathbf{Z}_n^*$, $\mathsf{HList} \leftarrow \mathsf{HList} \cup \{(i+j,m,\perp,h)\}$, return $h$ │
    └────────────────────────────────────────────────────────────────────────────┘

function Signing_Oracle($m$)
    $h \leftarrow$ Hash_Oracle($m$), $\sigma \leftarrow h^d \bmod n$
    return $\sigma$
```

Game 2: [Bridging Step.]

```
$(n,e,d) \leftarrow \mathsf{RSA}(1^k)$, $\mathsf{HList} \leftarrow \emptyset$, $i,j \leftarrow 0$ /* Global Vars */
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r,n,e\}$
Oracle_Queries($\mathcal{A}$, $\mathsf{view}$)
$(m^\star,\sigma^\star) \leftarrow \mathcal{A}(\mathsf{view})$
if $(*,m^\star,*,h^\star) \in \mathsf{HList}$ then $h \leftarrow h^\star$ else $h \xleftarrow{r} \mathbf{Z}_n^*$
if $\sigma^\star = h^d \bmod n$ then return 1 else return 0

function Oracle_Queries($\mathcal{A}$, $\mathsf{view}$)
    ┌────────────────────────────────────────────────────────────────────────────┐
    │ for $i = 1,\ldots,q_h$ do                                                  │
    │     $m_i \leftarrow \mathcal{A}(\mathsf{view})$                            │
    │     $h_i \leftarrow$ Hash_Oracle($m_i$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{h_i\}$ │
    │     if $j < q_s$ then possibly do $\sigma_i \leftarrow$ Signing_Oracle($m_i$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{\sigma_i\}$, $j \leftarrow j+1$ │
    │ end                                                                        │
    └────────────────────────────────────────────────────────────────────────────┘

function Hash_Oracle($m$)
    if $(*,m,*,h) \in \mathsf{HList}$ then return $h$
    else  $h \xleftarrow{r} \mathbf{Z}_n^*$, $\mathsf{HList} \leftarrow \mathsf{HList} \cup \{(\boxed{i},m,\perp,h)\}$, return $h$

function Signing_Oracle($m$)
    $h \leftarrow$ Hash_Oracle($m$), $\sigma \leftarrow h^d \bmod n$
    return $\sigma$
```

This game comes from two observations. This first is that we can assume without loss of generality that the adversary does not ask twice the same query to the signing oracle (as she/he would necessarily obtain the same answer twice). We can thus assume that the signing queries are distinct from each other.

The second observation is that if the adversary performs a signing query at a point m that she/he never queries to the hash oracle during the game, then the signature $\sigma \in \mathbf{Z}_n^*$ is a uniformly distributed random variable independent from the rest of the game at any time, and in particular from the rest of `view` (as it is the encryption of a random bitstring that is never included in `view`). We can thus assume that the adversary never queries the signing oracle at point she/he has not submitted to the hash oracle yet. Note that this implies that $q_h \geq q_s$. We have

$$\Pr[S_2] = \Pr[S_1]. \tag{15}$$

Game 3: Using a similar argument than in the previous game, if $m^\star$ does not match any entry in the list (i.e., if the adversary did not query the hash oracle at the point $m^\star$), then the success probability of the adversary is necessarily negligible since the guess $\sigma^\star$ is compared to a random value independent from the value of `view` at the end of the game. We can assume that there exists some index $1 \leq c < q_h +$ such that $m^\star = m_c$. In Game 3 we try to *guess* the value of c (we denote the guess $\widehat{c}$). If the guess is correct, Game 3 proceeds just as Game 2. If it is not, the game is aborted. As both games proceed identically unless $\widehat{c} \neq c$ in Game 3, we have

$$\Pr[S_3] = \Pr[S_2 \ \wedge \ \widehat{c} = c] = \Pr[S_2] \Pr[\widehat{c} = c] = \frac{1}{q_h} \Pr[S_2], \tag{16}$$

where the second equality comes from the fact that the events S_2 and $\widehat{c} = c$ are independent (they concern two distinct games), and where the last equality comes from the fact that $\widehat{c}$ is sampled uniformly at random.

```
$(n, e, d) \leftarrow \mathsf{RSA}(1^k)$, HList $\leftarrow \emptyset$, $i, j \leftarrow 0$ /* Global Vars */
$r \xleftarrow{r} R$, view $\leftarrow \{r, n, e\}$
$\boxed{\widehat{c} \xleftarrow{r} \{1, 2, \ldots, q_h\} \text{ /* Global Var */}}$
Oracle_Queries($\mathcal{A}$, view)
$(m^\star, \sigma^\star) \leftarrow \mathcal{A}$(view)
| Search for $(c, m^\star, *, h^\star) \in$ HList for some $c$ and $h^\star$ /* We assumed that this entry exists */
| if $\widehat{c} \neq c$ then abort else $h \leftarrow h^\star$
if $\sigma^\star = h^d \bmod n$ then return 1 else return 0

function Oracle_Queries($\mathcal{A}$, view)
    for $i = 1, \ldots, q_h$ do
        $m_i \leftarrow \mathcal{A}$(view)
        $h_i \leftarrow$ Hash_Oracle($m_i$), view $\leftarrow$ view $\cup \{h_i\}$
        if $j < q_s$ then possibly do $\sigma_i \leftarrow$ Signing_Oracle($m_i$), view $\leftarrow$ view $\cup \{\sigma_i\}$, $j \leftarrow j + 1$
    end

function Hash_Oracle($m$)
    if $(*, m, *, h) \in$ HList then return $h$
    else $h \xleftarrow{r} \mathbf{Z}_n^*$, HList $\leftarrow$ HList $\cup \{(i, m, \perp, h)\}$, return $h$

function Signing_Oracle($m$)
    $h \leftarrow$ Hash_Oracle($m$), $\sigma \leftarrow h^d \bmod n$
    return $\sigma$
```

Game 4: In this game we incorporate the challenge ciphertext (of the ow game) as follows: at the cth query to the hash oracle, we set the hash value to y, without querying the hash oracle.

```
$(n, e, d) \leftarrow \mathsf{RSA}(1^k)$, $\mathsf{HList} \leftarrow \emptyset$, $i, j \leftarrow 0$ /* Global Vars */
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r, n, e\}$
$\widehat{c} \xleftarrow{r} \{1, 2, \ldots, q_h\}$, $\boxed{y \xleftarrow{r} \mathbf{Z}_n^*}$ /* Global Vars */
Oracle_Queries($\mathcal{A}$, view)
$(m^\star, \sigma^\star) \leftarrow \mathcal{A}(\mathsf{view})$
Search for $(c, m^\star, *, h^\star) \in \mathsf{HList}$ for some $c$ and $h^\star$ /* We assumed that this entry exists */
if $\widehat{c} \neq c$ then abort else $h \leftarrow h^\star$
if $\sigma^\star = h^d \bmod n$ then return 1 else return 0

function Oracle_Queries($\mathcal{A}$, view)
    for $i = 1, \ldots, q_h$ do
        $m_i \leftarrow \mathcal{A}(\mathsf{view})$
        if $i = \widehat{c}$ then $\mathsf{HList} \leftarrow \mathsf{HList} \cup \{(i, m_i, \perp, y)\}$, $\mathsf{view} \leftarrow \mathsf{view} \cup \{y\}$
        else $h_i \leftarrow$ Hash_Oracle($m_i$) $\mathsf{view} \leftarrow \mathsf{view} \cup \{h_i\}$
        if $j < q_s$ then possibly do $\sigma_i \leftarrow$ Signing_Oracle($m_i$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{\sigma_i\}$, $j \leftarrow j + 1$
    end

function Hash_Oracle($m$)
    if $(*, m, *, h) \in \mathsf{HList}$ then return $h$
    else $h \xleftarrow{r} \mathbf{Z}_n^*$, $\mathsf{HList} \leftarrow \mathsf{HList} \cup \{(i, m, \perp, h)\}$, return $h$

function Signing_Oracle($m$)
    $h \leftarrow$ Hash_Oracle($m$), $\sigma \leftarrow h^d \bmod n$
    return $\sigma$
```

Since we assumed that the adversary never asks the same hash query twice, this game performs just as the previous one from the point of view of the adversary (i.e., the respective distributions of `view` are identical). Consequently,

$$\Pr[S_4] = \Pr[S_3]. \tag{17}$$

Game 5: In this game, instead of generating a random value for each hash query, we generate random signatures that we encrypt (using the public key) to get equivalent random hash values. Also note that we keep track of the plaintexts in the list. As the encryption is bijective, this makes no difference from the point of view of the adversary, and thus

$$\Pr[S_5] = \Pr[S_4]. \tag{18}$$

```
$(n,e,d) \leftarrow \mathsf{RSA}(1^k)$, $\mathsf{HList} \leftarrow \emptyset$, $i, j \leftarrow 0$ /* Global Vars */
$r \xleftarrow{\mathsf{r}} R$, $\mathsf{view} \leftarrow \{r, n, e\}$
$\widehat{c} \xleftarrow{\mathsf{r}} \{1, 2, \ldots, q_h\}$, $y \xleftarrow{\mathsf{r}} \mathbf{Z}_n^*$ /* Global Vars */
Oracle_Queries($\mathcal{A}$, view)
$(m^\star, \sigma^\star) \leftarrow \mathcal{A}(\mathsf{view})$
Search for $(c, m^\star, *, h^\star) \in \mathsf{HList}$ for some $c$ and $h^\star$ /* We assumed that this entry exists */
if $\widehat{c} \neq c$ then abort else $h \leftarrow h^\star$
if $\sigma^\star = h^d \bmod n$ then return 1 else return 0

function Oracle_Queries($\mathcal{A}$, view)
    for $i = 1, \ldots, q_h$ do
        $m_i \leftarrow \mathcal{A}(\mathsf{view})$
        if $i = \widehat{c}$ then  $\mathsf{HList} \leftarrow \mathsf{HList} \cup \{(i, m_i, \perp, y)\}$, $\mathsf{view} \leftarrow \mathsf{view} \cup \{y\}$
        else  $h_i \leftarrow$ Hash_Oracle($m_i$) $\mathsf{view} \leftarrow \mathsf{view} \cup \{h_i\}$
        if $j < q_s$ then possibly do $\sigma_i \leftarrow$ Signing_Oracle($m_i$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{\sigma_i\}$, $j \leftarrow j + 1$
    end

function Hash_Oracle($m$)
    if $(*, m, *, h) \in \mathsf{HList}$ then return $h$
    else $\boxed{x \xleftarrow{\mathsf{r}} \mathbf{Z}_n^*,\ h \leftarrow x^e \bmod n,\ \mathsf{HList} \leftarrow \mathsf{HList} \cup \{(i, m, x, h)\},}$ return $h$

function Signing_Oracle($m$)
    $h \leftarrow$ Hash_Oracle($m$), $\sigma \leftarrow h^d \bmod n$
    return $\sigma$
```

Game 6: Since the preimage of each hash query (except the cth one) is known, we can simulate the signing oracle without the need for the secret key. From the point of view of the adversary, this game is identical to the previous one:

$$\Pr[S_6] = \Pr[S_5]. \tag{19}$$

In Game 6, one can note that a valid forgery actually corresponds to a preimage of y. Moreover, the simulation of this game does not require to have access to the signing (decryption) oracle as it can be simulated. Indeed, the secret key is never used during the game (except for the final verification step of course). This game thus provides a description of a *valid* adversary $\mathcal{A}'$ that tries to break the one-wayness (ow) of the public key scheme. This adversary needs to perform q_h RSA encryptions (with the public key). Consequently,

$$\Pr[S_6] = \mathsf{Succ}^{\mathsf{ow}}_{\mathsf{rsa}}(\mathcal{A}'), \tag{20}$$

where $\mathcal{A}'$ performs in time $t' = t + q_h \cdot \mathcal{O}(k^3)$. From equations (13), (14), (15), (16), (17), (18), (19), and (20) we obtain

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{rsa}}(\mathcal{A}') = \frac{1}{q_h} \cdot \mathsf{Succ}^{\mathsf{euf}}_{\mathsf{fdh}}(\mathcal{A}).$$

```
$(n, e, d) \leftarrow \mathsf{RSA}(1^k)$, $\mathsf{HList} \leftarrow \emptyset$, $i, j \leftarrow 0$ /* Global Vars */
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r, n, e\}$
$\widehat{c} \xleftarrow{r} \{1, 2, \ldots, q_h\}$, $y \xleftarrow{r} \mathbf{Z}_n^*$ /* Global Vars */
Oracle_Queries($\mathcal{A}$, view)
$(m^\star, \sigma^\star) \leftarrow \mathcal{A}(\mathsf{view})$
Search for $(c, m^\star, *, h^\star) \in \mathsf{HList}$ for some $c$ and $h^\star$ /* We assumed that this entry exists */
if $\widehat{c} \neq c$ then abort else $h \leftarrow h^\star$
if $\sigma^\star = h^d \bmod n$ then return 1 else return 0

function Oracle_Queries($\mathcal{A}$, view)
    for $i = 1, \ldots, q_h$ do
        $m_i \leftarrow \mathcal{A}(\mathsf{view})$
        if $i = \widehat{c}$ then  $\mathsf{HList} \leftarrow \mathsf{HList} \cup \{(i, m_i, \perp, y)\}$, $\mathsf{view} \leftarrow \mathsf{view} \cup \{y\}$
        else  $h_i \leftarrow$ Hash_Oracle($m_i$) $\mathsf{view} \leftarrow \mathsf{view} \cup \{h_i\}$
        if $j < q_s$ then possibly do $\sigma_i \leftarrow$ Signing_Oracle($m_i$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{\sigma_i\}$, $j \leftarrow j + 1$
    end

function Hash_Oracle($m$)
    if $(*, m, *, h) \in \mathsf{HList}$ then return $h$
    else  $x \xleftarrow{r} \mathbf{Z}_n^*$, $h \leftarrow x^e \bmod n$, $\mathsf{HList} \leftarrow \mathsf{HList} \cup \{(i, m, x, h)\}$, return $h$

function Signing_Oracle($m$)
    $h \leftarrow$ Hash_Oracle($m$), $\boxed{\text{Search for } (*, *, x, h) \in \mathsf{HList},\ \sigma \leftarrow x}$
    return $\sigma$
```

11 A Better Security Bound for the Full-Domain Hash

11.1 Preliminaries

In Section 10 we described the FDH signature scheme and proved its security. For reasons detailed in Section 10.2 the bound was not satisfactory. In this section we introduce a new bound, proposed by Coron in [9].

Theorem 3. *Let $\mathcal{A}$ be an adversary performing a chosen-message attack against the Full Domain Hash in the random oracle model, with security parameter k. Let q_s and q_h denote the number of queries made by $\mathcal{A}$ to the signing oracle and to the hash oracle respectively. Let $\mathsf{Succ}^{\mathsf{euf}}_{\mathsf{fdh}}(\mathcal{A})$ be the success probability of $\mathcal{A}$ to produce an existential forgery in time t. Then there exists an adversary $\mathcal{A}'$ that breaks the one-wayness of RSA with probability of success $\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{rsa}}(\mathcal{A}')$ in time t' where*

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{rsa}}(\mathcal{A}') = \frac{1}{q_s \cdot e} \cdot \mathsf{Succ}^{\mathsf{euf}}_{\mathsf{fdh}}(\mathcal{A}) \quad \textit{and} \quad t' = t + q_h \cdot \mathcal{O}(k^3).$$

11.2 Proof of Theorem 3

Games 0,1, and 2 are almost the same in this proof than in the one of Theorem 2. The only difference is that the list HList contains elements that are different from those that were stored in the previous proof. The two first elements will naturally correspond to a message and to its image by H. The next to will be clarified in Game 3. These last two values are always set to $\perp$ in games 0,1, and 2.

Game 3: In the previous proof, the challenge ciphertext y was introduced only once, at a specific index. Here, for each query, we introduce it with probability p (that we will precised later). With probability $1 - p$ we introduce a value with a known preimage. In both cases, the value returned by Hash_Oracle is a uniformly distributed random value of $\mathbf{Z}_n^\star$, just as in Game 2. Consequently,

$$\Pr[S_3] = \Pr[S_2]. \tag{21}$$

```
$(n,e,d) \leftarrow \mathsf{RSA}(1^k)$, $\mathsf{HList} \leftarrow \emptyset$, $i, j \leftarrow 0$, $\boxed{y \xleftarrow{r} \mathbf{Z}_n^\star}$ /* Global Vars */
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r, n, e\}$
Oracle_Queries($\mathcal{A}$, view)
$(m^\star, \sigma^\star) \leftarrow \mathcal{A}(\mathsf{view})$
if $(m^\star, h^\star, *, *) \in \mathsf{HList}$ then $h \leftarrow h^\star$ else $h \xleftarrow{r} \mathbf{Z}_n^*$
if $\sigma^\star = h^d \bmod n$ then return 1 else return 0

function Oracle_Queries($\mathcal{A}$, view)
    for $i = 1, \ldots, q_h$ do
        $m_i \leftarrow \mathcal{A}(\mathsf{view})$
        $h_i \leftarrow$ Hash_Oracle($m_i$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{h_i\}$
        if $j < q_s$ then possibly do $\sigma_i \leftarrow$ Signing_Oracle($m_i$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{\sigma_i\}$, $j \leftarrow j + 1$
    end

function Hash_Oracle($m$)
    if $(m, h, *, *) \in \mathsf{HList}$ then return $h$
    else
        $s \xleftarrow{r} \mathbf{Z}_n^\star$, With probability $p$, $h \leftarrow y \cdot s^e$ and $t \leftarrow 1$, otherwise $h \leftarrow s^e$ and $t \leftarrow 0$
        $\mathsf{HList} \leftarrow \mathsf{HList} \cup \{(m, h, s, t)\}$,
    end

function Signing_Oracle($m$)
    $h \leftarrow$ Hash_Oracle($m$), $\sigma \leftarrow h^d \bmod n$
    return $\sigma$
```

Game 4: We first note that if $m^\star$ cannot be found in HList, then the advantage of the adversary is necessarily negligible as her/his guess $\sigma^\star$ is compared to a fresh random value. Consequently, we assume in this game that $m^\star$ can always be found in the list. This will simplify the analysis of Game 5. Now the tricky part.

For a proportion $1-p$ of the signing queries, it is now possible to simulate the signing oracle without the knowledge of the secret key. Games 3 and 4 are identical unless Game 4 abort (an event that we denote F), i.e., unless $t = 1$ for one of the q_s signing oracle queries. As $t = 1$ with probability p we have

$$\Pr[S_4] = \Pr[\overline{F} \wedge S_3] = \Pr[\overline{F}] \Pr[S_3] = (1-p)^{q_s} \cdot \Pr[S_3]. \tag{22}$$

```
$(n, e, d) \leftarrow \mathsf{RSA}(1^k)$, $\mathsf{HList} \leftarrow \emptyset$, $i, j \leftarrow 0$, $y \xleftarrow{r} \mathbf{Z}_n^\star$ /* Global Vars */
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r, n, e\}$
Oracle_Queries($\mathcal{A}$, view)
$(m^\star, \sigma^\star) \leftarrow \mathcal{A}(\mathsf{view})$
$\boxed{\text{Search for } (m^\star, h^\star, *, *) \in \mathsf{HList} \text{ and set } h \leftarrow h^\star}$
if $\sigma^\star = h^d \bmod n$ then return 1 else return 0

function Oracle_Queries($\mathcal{A}$, view)
    for $i = 1, \ldots, q_h$ do
        $m_i \leftarrow \mathcal{A}(\mathsf{view})$
        $h_i \leftarrow$ Hash_Oracle($m_i$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{h_i\}$
        if $j < q_s$ then possibly do $\sigma_i \leftarrow$ Signing_Oracle($m_i$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{\sigma_i\}$, $j \leftarrow j + 1$
    end

function Hash_Oracle($m$)
    if $(m, h, *, *) \in \mathsf{HList}$ then return $h$
    else
        $s \xleftarrow{r} \mathbf{Z}_n^\star$, With probability $p$, $h \leftarrow y \cdot s^e$ and $t \leftarrow 1$, otherwise $h \leftarrow s^e$ and $t \leftarrow 0$
        $\mathsf{HList} \leftarrow \mathsf{HList} \cup \{(m, h, s, t)\}$,
    end

function Signing_Oracle($m$)
    $\boxed{\text{Search for } (m, h, s, t) \in \mathsf{HList} \text{ for some } h, s, t. \textbf{ if } t = 1 \textbf{ then abort else } \sigma \leftarrow s}$
    return $\sigma$
```

Game 5: We add a last modification at the end of the game. If the last lookup returns a tuple such that $t = 0$ we abort the game (an event that we denote F'). As $t = 0$ with probability $1 - p$,

$$\Pr[S_5] = \Pr[\overline{F'} \ \wedge \ S_4] = \Pr[\overline{F'}] \cdot \Pr[S_4] = p \cdot \Pr[S_4]. \tag{23}$$

In this last game, the simulation can be performed without the knowledge of the secret key (except for the last verification step of course). Note also that when Game 5 outputs one, it is easy to find the preimage x of y as in that case we have $\sigma^\star = h^d$ and $h = y \cdot s^e$, so that $x = y^d = h^d / s^{e \cdot d} = \sigma^\star / s$. This game thus provides a description of a *valid* adversary $\mathcal{A}'$ that breaks the one-wayness (ow) of the public key scheme by use of $\mathcal{A}$ so that we can denote

$$\Pr[S_5] = \mathsf{Succ}^{\mathsf{ow}}_{\mathsf{rsa}}(\mathcal{A}'). \tag{24}$$

This adversary performs q_h encryptions (using the public key) and thus, it performs in time $t' = t + q_h \cdot \mathcal{O}(k^3)$. From equations (13), (14), (15), (21), (22), (23), and (24), we obtain

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{rsa}}(\mathcal{A}') = p(1-p)^{q_s} \cdot \mathsf{Succ}^{\mathsf{euf}}_{\mathsf{fdh}}(\mathcal{A}).$$

To best success probability of adversary $\mathcal{A}'$ is obtained by choosing $p = \frac{1}{1+q_s}$, in which case we obtain

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{rsa}}(\mathcal{A}') = \frac{1}{1+q_s}\left(1 - \frac{1}{1+q_s}\right)^{q_s} \cdot \mathsf{Succ}^{\mathsf{euf}}_{\mathsf{fdh}}(\mathcal{A}).$$

When q_s is large, this can be approximated by

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{rsa}}(\mathcal{A}') = \frac{1}{q_s \cdot e} \cdot \mathsf{Succ}^{\mathsf{euf}}_{\mathsf{fdh}}(\mathcal{A}),$$

where $e = \exp(1)$.

```
$(n,e,d) \leftarrow \mathsf{RSA}(1^k)$, $\mathsf{HList} \leftarrow \emptyset$, $i,j \leftarrow 0$, $y \xleftarrow{r} \mathbf{Z}_n^\star$ /* Global Vars */
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r,n,e\}$
Oracle_Queries($\mathcal{A}$, view)
$(m^\star, \sigma^\star) \leftarrow \mathcal{A}(\mathsf{view})$
$\boxed{\text{Search for } (m^\star, h^\star, s, t) \in \mathsf{HList}, \textbf{ if } t = 0 \textbf{ then } \mathsf{abort} \textbf{ else } h \leftarrow h^\star}$
if $\sigma^\star = h^d \bmod n$ then return 1 else return 0

function Oracle_Queries($\mathcal{A}$, view)
    for $i = 1, \ldots, q_h$ do
        $m_i \leftarrow \mathcal{A}(\mathsf{view})$
        $h_i \leftarrow$ Hash_Oracle($m_i$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{h_i\}$
        if $j < q_s$ then possibly do $\sigma_i \leftarrow$ Signing_Oracle($m_i$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{\sigma_i\}$, $j \leftarrow j+1$
    end

function Hash_Oracle($m$)
    if $(m, h, *, *) \in \mathsf{HList}$ then return $h$
    else
        $s \xleftarrow{r} \mathbf{Z}_n^\star$, With probability $p$, $h \leftarrow y \cdot s^e$ and $t \leftarrow 1$, otherwise $h \leftarrow s^e$ and $t \leftarrow 0$
        $\mathsf{HList} \leftarrow \mathsf{HList} \cup \{(m, h, s, t)\}$,
    end

function Signing_Oracle($m$)
    Search for $(m, h, s, t) \in \mathsf{HList}$ for some $s, h, t$. if $t = 1$ then abort else $\sigma \leftarrow s$
    return $\sigma$
```

12 OAEP+

OAEP is a public-key encryption scheme introduced by Bellare and Rogaway in [4]. Although very efficient, this scheme suffers from the fact that it is not provably secure against adaptive chosen ciphertext attacks. Shoup shows in [28, 29] that no proof is attainable for the general OAEP scheme by only assuming the one-wayness of the underlying trapdoor permutation, even in the random oracle model. Yet, he proves that when the underlying trapdoor permutation is RSA with a public exponent equal to 3, then the construction is secure (a result that is extended in [12] to any public exponent). To obtain a provably secure scheme under the one-wayness assumption of the underlying trapdoor permutation, Shoup introduces the OAEP+ public-key encryption scheme.

12.1 Preliminaries

OAEP+ is based on a one-way trapdoor permutation $f_{\mathsf{pk}} : \{0,1\}^k \to \{0,1\}^k$, its inverse being f_{sk}^{-1}. Let k_0 and k_1 be two parameters that satisfy $k_0 + k_1 < k$ and such that 2^{-k_0} and 2^{-k_1} are negligible. The scheme encrypts messages $x \in \{0,1\}^n$ where $n = k - k_0 - k_1$. It makes use of three hash functions (that will be modeled in the proofs as random oracles):

$$\begin{aligned} G : \{0,1\}^{k_0} &\longrightarrow \{0,1\}^n, \\ H' : \{0,1\}^{n+k_0} &\longrightarrow \{0,1\}^{k_1}, \\ H : \{0,1\}^{n+k_1} &\longrightarrow \{0,1\}^{k_0}. \end{aligned}$$

Key Generation: On input the security parameter, the key generation algorithm produces a public/private key pair $(\mathsf{pk}, \mathsf{sk})$, defining the public permutation f_{pk} and its inverse f_{sk}^{-1}.

Encryption: Given a plaintext $x \in \{0,1\}^n$, the encryption algorithm chooses $r \xleftarrow{r} \{0,1\}^{k_0}$ and computes

$$\begin{array}{ll} s \leftarrow (G(r) \oplus x) \| H'(r\|x), & (s \in \{0,1\}^{n+k_1}), \\ t \leftarrow H(s) \oplus r, & (t \in \{0,1\}^{k_0}), \\ w \leftarrow s\|t, & (w \in \{0,1\}^{k}), \\ y \leftarrow f_{\mathsf{pk}}(w) & (y\{0,1\}^{k}). \end{array}$$

The ciphertext is y.

Decryption: On input $y \in \{0,1\}^k$, the decryption algorithm performs the following computations:

$$\begin{array}{lll} w & \leftarrow f_{\mathsf{sk}}^{-1}(y) & (w \in \{0,1\}^{k}), \\ s\|t & \leftarrow w & (s \in \{0,1\}^{n+k_1}, t \in \{0,1\}^{k_0}), \\ r & \leftarrow H(s) \oplus t & (r \in \{0,1\}^{k_0}), \\ x & \leftarrow G(r) \oplus s[0 \cdots n-1] & (x \in \{0,1\}^{n}), \\ c & \leftarrow s[n \cdots n+k_1-1] & (c \in \{0,1\}^{k_1}). \end{array}$$

If $c = H'(r\|x)$, the algorithm outputs the cleartext x; otherwise, the algorithm rejects the ciphertext and does not output a cleartext.

Theorem 4. *If the underlying trapdoor permutation f is one-way* (ow)*, then* $\mathsf{OAEP+}$ *is secure against adaptive chosen ciphertext attack in the random oracle model.*

12.2 Proof of Theorem 4

Throughout this proof the event S_i always denotes the probability that Game i returns 1. Moreover we note that, during the decryption process, H' is always queried at points of the form $r\|x$ where $x = G(r) \oplus s[0 \cdots n-1]$. Consequently, an efficient adversary would *not* query H' at a point $r\|x$ without at least querying $G(r)$ first (otherwise, the probability that the H' query makes any sense would be negligible). In the proof, we thus assume that whenever a query of the form $H'(r\|x)$ is made by the adversary $\mathcal{A}$, then $\mathcal{A}$ has previously made the query $G(r)$.

Game 0: This is the original attack game against the encryption scheme. It is represented on page 23. In this game we let $\mathcal{G} = \{g : \{0,1\}^{k_0} \to \{0,1\}^n\}$, $\mathcal{H}' = \{h : \{0,1\}^{n+k_0} \to \{0,1\}^{k_1}\}$, and $\mathcal{H} = \{h : \{0,1\}^{n+k_1} \to \{0,1\}^{k_0}\}$. We have

$$\mathsf{Succ}^{\mathsf{cca}}_{\mathsf{oaep+}}(\mathcal{A}) = |\Pr[S_0] - 1/2|. \tag{25}$$

Game 0' : We adopt the (by now) well known lazy-sampling technique. Obviously,

$$\Pr[S_{0'}] = \Pr[S_0]. \tag{26}$$

```
$(\mathsf{pk},\mathsf{sk}) \overset{r}{\longleftarrow} \mathcal{K}(1^k)$, $G \overset{r}{\longleftarrow} \mathcal{G}$, $H' \overset{r}{\longleftarrow} \mathcal{H}'$ $H \overset{r}{\longleftarrow} \mathcal{H}$, $i, j, k, \ell \leftarrow 0$ /* Global Vars */
$r \overset{r}{\longleftarrow} R$, $\mathsf{view} \leftarrow \{r, \mathsf{pk}\}$
Oracle_Queries($\mathcal{A}, \mathsf{view}, \perp$)
$(x_0, x_1) \leftarrow \mathcal{A}(\mathsf{view})$, $(y^\star, \mathsf{bit}) \leftarrow$ Encryption_Oracle($x_0, x_1$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{y^\star\}$
Oracle_Queries($\mathcal{A}, \mathsf{view}, y^\star$)
$\widehat{\mathsf{bit}} \leftarrow \mathcal{A}(\mathsf{view})$
if $\widehat{\mathsf{bit}} = \mathsf{bit}$ then return 1 else return 0

function Oracle_Queries($\mathcal{A}, \mathsf{view}, y^\star$)
    loop
        do either
            /* Hash Oracle query to G */
            if $i < q_g$ then  $r \leftarrow \mathcal{A}(\mathsf{view})$, $g \leftarrow \mathtt{G}(r)$, $\mathsf{view} \leftarrow \mathsf{view} \cup \{g\}$, $i \leftarrow i + 1$
        or
            /* Hash Oracle query to H' */
            if $j < q_{h'}$ then  $r, x \leftarrow \mathcal{A}(\mathsf{view})$, $h' \leftarrow \mathtt{H'}(r \| x)$, $\mathsf{view} \leftarrow \mathsf{view} \cup \{h'\}$, $j \leftarrow j + 1$
        or
            /* Hash Oracle query to H */
            if $k < q_h$ then  $s \leftarrow \mathcal{A}(\mathsf{view})$, $h \leftarrow \mathtt{H}(s)$, $\mathsf{view} \leftarrow \mathsf{view} \cup \{h\}$, $k \leftarrow k + 1$
        or
            /* Decryption Oracle query */
            if $\ell < q_d$ then
                $y \leftarrow \mathcal{A}(\mathsf{view})$ such that $y \neq y^\star$
                $x \leftarrow$ Decryption_Oracle($y$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{x\}$, $\ell \leftarrow \ell + 1$
        done
    end

function G($r$)
    $g \leftarrow G(r)$, return $g$

function H'($r \| x$)
    $h' \leftarrow H'(r \| x)$, return $h'$

function H($s$)
    $h \leftarrow H(s)$, return $h$

function Decryption_Oracle($y$)
    $w \leftarrow f_{\mathsf{sk}}^{-1}(y)$, $s \| t \leftarrow w$, $r \leftarrow \mathtt{H}(s) \oplus t$, $x \leftarrow \mathtt{G}(r) \oplus s[0 \cdots n-1]$, $c \leftarrow s[n \cdots n + k_1 - 1]$
    if $c = \mathtt{H'}(r \| x)$ then return $x$ else return reject

function Encryption_Oracle($x_1, x_2$)
    $\mathsf{bit} \overset{r}{\longleftarrow} \{0, 1\}$, $x^\star \leftarrow x_{\mathsf{bit}}$
    $r^\star \overset{r}{\longleftarrow} \{0, 1\}^{k_0}$
    $s^\star \leftarrow (\mathtt{G}(r^\star) \oplus x^\star) \| \mathtt{H'}(r^\star \| x^\star)$, $t^\star \leftarrow \mathtt{H}(s^\star) \oplus r^\star$, $w^\star \leftarrow s^\star \| t^\star$, $y^\star \leftarrow f_{\mathsf{pk}}(w^\star)$
    return $(y^\star, \mathsf{bit})$
```

```
$(\mathsf{pk},\mathsf{sk}) \xleftarrow{r} \mathcal{K}(1^k)$, $\boxed{\mathsf{Glist} \leftarrow \emptyset,\ \mathsf{H'list} \leftarrow \emptyset,\ \mathsf{Hlist} \leftarrow \emptyset,}$ $i, j, k, \ell \leftarrow 0$ /* Global Vars */
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r, \mathsf{pk}\}$
Oracle_Queries($\mathcal{A}$, $\mathsf{view}$, $\perp$)
$(x_0, x_1) \leftarrow \mathcal{A}(\mathsf{view})$, $(y^\star, \mathsf{bit}) \leftarrow$ Encryption_Oracle($x_0, x_1$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{y^\star\}$
Oracle_Queries($\mathcal{A}$, $\mathsf{view}$, $y^\star$)
$\widehat{\mathsf{bit}} \leftarrow \mathcal{A}(\mathsf{view})$
if $\widehat{\mathsf{bit}} = \mathsf{bit}$ then return 1 else return 0

function G($r$)
    $\boxed{\textbf{if } (r, g) \in \mathsf{Glist} \textbf{ then return } g \textbf{ else } g \xleftarrow{r} \{0,1\}^n, \mathsf{Glist} \leftarrow \mathsf{Glist} \cup \{(r, g)\}, \text{return } g}$

function H'($r\|x$)
    $\boxed{\textbf{if } (r, x, h') \in \mathsf{H'list} \textbf{ then return } h' \textbf{ else } h' \xleftarrow{r} \{0,1\}^{k_1}, \mathsf{H'list} \leftarrow \mathsf{H'list} \cup \{(r, x, h')\}, \text{return } h'}$

function H($s$)
    $\boxed{\textbf{if } (s, h) \in \mathsf{Hlist} \textbf{ then return } h \textbf{ else } h \xleftarrow{r} \{0,1\}^{k_0}, \mathsf{Hlist} \leftarrow \mathsf{Hlist} \cup \{(s, h)\}, \text{return } h'}$
```

Game 1: We modify the decryption oracle so that it never queries G nor H' at points that were not previously queried by the adversary.

```
function Decryption_Oracle($y$)
    $w \leftarrow f_{\mathsf{sk}}^{-1}(y)$, $s\|t \leftarrow w$, $r \leftarrow$ H($s$) $\oplus t$
    $\boxed{\textbf{if } (r, *) \notin \mathsf{Glist} \textbf{ then reject}}$
    $x \leftarrow$ G($r$) $\oplus s[0 \cdots n-1]$, $c \leftarrow s[n \cdots n + k_1 - 1]$
    $\boxed{\textbf{if } (r, x, *) \notin \mathsf{H'list} \textbf{ then reject}}$
    if $c =$ H'($r\|x$) then return $x$ else return reject
```

Let F_1 be the event that a ciphertext is rejected in Game 1 that would not have been rejected in Game 0'. Consider a ciphertext y submitted to the decryption oracle (denote $w, s, t, \ldots$ the values computed in Game 0'), and denote by F_1' the event that y is rejected in Game 1 but would not in Game 0'. We have $F_1' \Leftrightarrow (H'(r\|x) = c \wedge (r, x, *) \notin \mathsf{H'list})$ (recall that we assumed that if the adversary has queried $H'(r\|x)$ then he has already queried $G(r)$), and thus

$$\Pr[F_1'] \leq \Pr[H'(r\|x) = c \mid (r, x, *) \notin \mathsf{H'list}] = 2^{-k_1}.$$

From the union bound we conclude that $\Pr[F_1] \leq q_d/2^{k_1}$. Since both games proceed identically unless F_1 occurs, the difference lemma gives

$$|\Pr[S_1] - \Pr[S_{0'}]| \leq q_d/2^{k_1}. \tag{27}$$

Game 2: We modify the decryption oracle again. Let y be a ciphertext submitted to the decryption oracle. If the previous decryption oracle rejects y, so does the new one. However, the new decryption oracle also reject y when $H(s)$ has not been queried yet. Note that in Game 2, the decryption oracle never queries G, H, or H' at point other than at which the adversary did.

```
function Decryption_Oracle($y$)
    $w \leftarrow f_{\mathsf{sk}}^{-1}(y)$, $s\|t \leftarrow w$
    $\boxed{\textbf{if } (s,*) \notin \mathsf{Hlist} \textbf{ then } \mathsf{reject}}$
    $r \leftarrow \mathtt{H}(s) \oplus t$
    if $(r,*) \notin \mathsf{Glist}$ then reject
    $x \leftarrow \mathtt{G}(r) \oplus s[0 \cdots n-1]$, $c \leftarrow s[n \cdots n+k_1-1]$
    if $(r,x,*) \notin \mathsf{H'list}$ then reject
    if $c = \mathtt{H'}(r\|x)$ then return $x$ else return reject
```

Let F_2 be the event that a ciphertext is rejected in Game 2 that would not have been rejected in Game 1. To compute $\Pr[F_2]$, we consider a ciphertext y submitted to the decryption oracle such that $H(s)$ has not been queried yet (so that y is rejected by Game 2) and evaluate the probability that y is accepted in Game 1:

- If the encryption oracle has not been queried yet or if $s \neq s^\star$, then $H(s)$ has never been queried before, either by the adversary or the encryption oracle. Thus, $H(s)$ is a fresh random value independent of the adversary's view. As $r \leftarrow H(s) \oplus t$, this is also the case of r. If y is not rejected by Game 1, it must be the case that $G(r)$ has been queried by the adversary, which occurs with a probability at most $q_g/2^{k_0}$. Over the course of the entire attack, the probabilities sum to $q_d q_g/2^{k_0}$.
- If the encryption oracle has already been queried and $s = s^\star$, it must be the case that $t \neq t^\star$ (as $y \neq y^\star$). Moreover, $s = s^\star$ and $t \neq t^\star$ implies $r \neq r^\star$ and thus $r\|x \neq r^\star\|x^\star$. If y is accepted in Game 1, it must be the case that $H'(r\|x) = H'(r^\star\|x^\star)$ (as $s = s^\star$ implies $c = c^\star$). This last event occurs with probability 2^{-k_1} (note that no birthday attack can be possible here as $r^\star\|x^\star$ are fixed by the challenger). Overall, the probability that such a collision occur over the course of the attack is $q_{h'}/2^{k_1}$.

We conclude that $\Pr[F_2] \leq q_{h'}/2^{k_1} + q_d q_g/2^{k_0}$, so that

$$|\Pr[S_2] - \Pr[S_1]| \leq q_{h'}/2^{k_1} + q_d q_g/2^{k_0}. \tag{28}$$

Game 3: We modify the decryption oracle so that it does not make use of the secret key anymore. Both games will be equivalent. Recall that for each triplet (r, x, h') found in $\mathsf{H'list}$, we know for sure that there exists some g such that $(r, g) \in \mathsf{Glist}$.

Note that the time and space complexities of this algorithm are linear with respect to the total number of queries made to the oracles. Note also that any ciphertext rejected by Game 3 would also be rejected by Game 2. It follows that

$$\Pr[S_3] = \Pr[S_2]. \tag{29}$$

```
function Decryption_Oracle($y$)
    foreach $(r,x,h') \in \mathsf{H'list}$ do
        Look for $(r,g) \in \mathsf{Glist}$
        $s \leftarrow (g \oplus x)\|h'$
        if $(s,h) \in \mathsf{Hlist}$ then
            $t \leftarrow h \oplus r$, $w \leftarrow s\|t$, $\widehat{y} = f_{\mathsf{pk}}(w)$
            if $\widehat{y} = y$ then return $x$
        end
    end
    return reject
```

Game 4 : We only perform a bridging step here. We first modify the encryption oracle as follows: instead of choosing $r^\star$ at random at the time of encryption, we choose it at the beginning of the game. This makes no difference from the adversary point of view. Similarly, as $r^\star$ is known from the beginning, we can set the value of $G(r^\star)$ (still at random) right away. Once this is done, we add a new random oracle $H^\star : \{0,1\}^n \to \{0,1\}^{k_1}$ which will sometimes answer instead of H': if H' is queried at a point $r^\star\|x$ for some x, then the answer is $H^\star(x)$. As the distribution of the triplet $(r, x, H'(r\|x))$ remains unchanged, adding this new random oracle makes no difference from the adversary point of view. Finally,

$$\Pr[S_4] = \Pr[S_3]. \tag{30}$$

```
(pk, sk) $\xleftarrow{r} \mathcal{K}(1^k)$, $\boxed{r^\star \xleftarrow{r} \{0,1\}^{k_0},\ g^\star \xleftarrow{r} \{0,1\}^n,\ \mathsf{Glist} \leftarrow \{(r^\star, g^\star)\},\ H^\star \xleftarrow{r} \Gamma_{n,k_1}}$, H'list $\leftarrow \emptyset$,
Hlist $\leftarrow \emptyset$, $i, j, k, \ell \leftarrow 0$ /* Global Vars */
$r \xleftarrow{r} R$, view $\leftarrow \{r, \mathsf{pk}\}$
Oracle_Queries($\mathcal{A}$, view, $\perp$)
$(x_0, x_1) \leftarrow \mathcal{A}$(view), $(y^\star, \mathsf{bit}) \leftarrow$ Encryption_Oracle($x_0, x_1$), view $\leftarrow$ view $\cup \{y^\star\}$
Oracle_Queries($\mathcal{A}$, view, $y^\star$)
$\widehat{\mathsf{bit}} \leftarrow \mathcal{A}$(view)
if $\widehat{\mathsf{bit}} = \mathsf{bit}$ then return 1 else return 0

function H'($r\|x$)
    $\boxed{\textbf{if } r = r^\star \textbf{ then return } H^\star(x)}$
    if $(r, x, h') \in$ H'list then return $h'$ else $h' \xleftarrow{r} \{0,1\}^{k_1}$, H'list $\leftarrow$ H'list $\cup \{(r, x, h')\}$, return $h'$

function Encryption_Oracle($x_1, x_2$)
    bit $\xleftarrow{r} \{0,1\}$, $x^\star \leftarrow x_{\mathsf{bit}}$
    $\boxed{s^\star \leftarrow (g^\star \oplus x^\star)\|H^\star(x^\star)}$, $t^\star \leftarrow \mathtt{H}(s^\star) \oplus r^\star$, $w^\star \leftarrow s^\star\|t^\star$, $y^\star \leftarrow f_{\mathsf{pk}}(w^\star)$
    return $(y^\star, \mathsf{bit})$
```

Game 5: In this game we drop two rules so that $y^\star$ is a random string independent from `view`.

```
(pk, sk) $\xleftarrow{r} \mathcal{K}(1^k)$, $r^\star \xleftarrow{r} \{0,1\}^{k_0}$, $g^\star \xleftarrow{r} \{0,1\}^n$, $\boxed{\mathsf{Glist} \leftarrow \emptyset}$, $H^\star \xleftarrow{r} \Gamma_{n,k_1}$, H'list $\leftarrow \emptyset$, Hlist $\leftarrow \emptyset$,
$i, j, k, \ell \leftarrow 0$ /* Global Vars */
$r \xleftarrow{r} R$, view $\leftarrow \{r, \mathsf{pk}\}$
Oracle_Queries($\mathcal{A}$, view, $\perp$)
$(x_0, x_1) \leftarrow \mathcal{A}$(view), $(y^\star, \mathsf{bit}) \leftarrow$ Encryption_Oracle($x_0, x_1$), view $\leftarrow$ view $\cup \{y^\star\}$
Oracle_Queries($\mathcal{A}$, view, $y^\star$)
$\widehat{\mathsf{bit}} \leftarrow \mathcal{A}$(view)
if $\widehat{\mathsf{bit}} = \mathsf{bit}$ then return 1 else return 0

function H'($r\|x$)
    $\boxed{\texttt{/* rule removed */}}$
    if $(r, x, h') \in$ H'list then return $h'$ else $h' \xleftarrow{r} \{0,1\}^{k_1}$, H'list $\leftarrow$ H'list $\cup \{(r, x, h')\}$, return $h'$
```

Here, as both $g^\star$ and $H^\star$ (and thus $H^\star(x^\star)$) are not used anywhere else than in the encryption oracle, then both $s^\star$ and $t^\star$ (and thus $w^\star$) are independent of $x^\star$. Consequently, in Game 5 we have

$$\Pr[S_5] = \frac{1}{2}. \tag{31}$$

Games 4 and 5 proceed identically unless the adversary queries G at point $r^\star$ or H' at point $r^\star\|x$ for some $x \in \{0,1\}^n$ (note that if the latter case occurs then, by assumption, G has already been queried at point $r^\star$). Let F_5 be the event that in Game 5, the adversary queries G at point $r^\star$. As both games proceed identically unless F_5 occurs,

$$|\Pr[S_5] - \Pr[S_4]| \le \Pr[F_5]. \tag{32}$$

All we need to do now is to bound $\Pr[F_5]$. To do this we introduce a last game, equivalent to this one, but easier to study.

Game 5' : As the encryption oracle outputs a bitstring independent from the adversary view (and in particular from $x^\star$ and thus from both x_1 and x_2), we can set $y^\star$ (and $w^\star$, $s^\star$, and $t^\star$) right from the beginning of the game. When the encryption oracle is queried, it simply returns $y^\star$. These modifications do not affect the value of $\Pr[F_5]$.

```
$(\mathsf{pk},\mathsf{sk}) \xleftarrow{r} \mathcal{K}(1^k)$, $r^\star \xleftarrow{r} \{0,1\}^{k_0}$, $g^\star \xleftarrow{r} \{0,1\}^n$, $\mathsf{Glist} \leftarrow \emptyset$, $H^\star \xleftarrow{r} \Gamma_{n,k_1}$, $\mathsf{H'list} \leftarrow \emptyset$, $\mathsf{Hlist} \leftarrow \emptyset$,
$i, j, k, \ell \leftarrow 0$ /* Global Vars */
$\boxed{y^\star \xleftarrow{r} \{0,1\}^k,\ w^\star \leftarrow f_{\mathsf{sk}}^{-1}(y^\star),\ s^\star \| t^\star \leftarrow w^\star,\ r^\star \leftarrow \mathtt{H}(s^\star) \oplus t^\star \text{ /* Global Vars */}}$
$r \xleftarrow{r} R$, $\mathsf{view} \leftarrow \{r, \mathsf{pk}\}$
Oracle_Queries($\mathcal{A}$, view, $\perp$)
$(x_0, x_1) \leftarrow \mathcal{A}(\mathsf{view})$, $(y^\star, \mathsf{bit}) \leftarrow$ Encryption_Oracle($x_0, x_1$), $\mathsf{view} \leftarrow \mathsf{view} \cup \{y^\star\}$
Oracle_Queries($\mathcal{A}$, view, $y^\star$)
$\widehat{\mathsf{bit}} \leftarrow \mathcal{A}(\mathsf{view})$
if $\widehat{\mathsf{bit}} = \mathsf{bit}$ then return 1 else return 0

function Encryption_Oracle($x_1, x_2$)
    $\mathsf{bit} \xleftarrow{r} \{0,1\}$, $x^\star \leftarrow x_{\mathsf{bit}}$
    $\boxed{\text{/* rules removed */}}$
    return $(y^\star, \mathsf{bit})$
```

Let F_5' the event that the adversary queries H at $s^\star$ in this new game. We have

$$\Pr[F_5] = \Pr[F_5 \wedge F_5'] + \Pr[F_5 \wedge \overline{F_5'}].$$

- We first bound $\Pr[F_5 \wedge F_5']$: if the adversary has queried $r^\star$ to G and $s^\star$ to H, it is possible to define and *inverting* adversary $\mathcal{A}'$ (that recovers $f_{\mathsf{sk}}^{-1}(y^\star)$ successfully) as follows: $\mathcal{A}'$ first runs $\mathcal{A}$ and then enumerates all $(r, g) \in \mathsf{Glist}$ and $(s, h) \in \mathsf{Hlist}$ and for each of these computes:

  $$t \leftarrow h \oplus r, \quad w \leftarrow s \| t, \quad y \leftarrow f_{\mathsf{pk}}(w).$$

  If $y = y^\star$ (which eventually happens in this case), $\mathcal{A}'$ outputs w and thus successfully inverts f_{pk}. Consequently,

  $$\Pr[F_5 \wedge F_5'] \leq \mathsf{Succ}^{\mathsf{ow}}_{f_{\mathsf{pk}}}(\mathcal{A}').$$

  At this point, it is important to note that $\mathcal{A}'$ is *efficient*. Indeed, the running time of $\mathcal{A}'$ is equal to that of $\mathcal{A}$ (which was shown to be linear with respect to the total number of queries made to the oracles) plus a factor proportional to $q_g q_h T_f$ where T_f is the time required to evaluate $f_{\mathsf{pk}}(\cdot)$. The space requirements of $\mathcal{A}'$ are essentially the same as those of $\mathcal{A}$.
- We bound $\Pr[F_5 \wedge \overline{F_5'}]$: In that case, H has *not* been queried at point $s^\star$. As $r^\star \leftarrow H(s^\star) \oplus t^\star$, $r^\star$ is independent from `view` and thus the probability that $r^\star$ is queried to G over the course of the entire attack is bounded by $q_g/2^{k_0}$. We obtain

  $$\Pr[F_5 \wedge \overline{F_5'}] \leq q_g/2^{k_0}.$$

From these two cases we obtain that

$$\Pr[F_5] \leq \mathsf{Succ}^{\mathsf{ow}}_{f_{\mathsf{pk}}}(\mathcal{A}') + q_g/2^{k_0}. \tag{33}$$

Conclusion: From equations (25), (26), (27), (28), (29), (30), (31), (32), and (33), we conclude that

$$\mathsf{Succ}^{\mathsf{cca}}_{\mathsf{oaep+}}(\mathcal{A}) \leq \mathsf{Succ}^{\mathsf{ow}}_{f_{\mathsf{pk}}}(\mathcal{A}') + q_g(1 + q_d)/2^{k_0} + (q_{h'} + q_d)/2^{k_1}.$$

References

1. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In Krawczyk [19], pages 26–45.

2. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *Advances in Cryptology* - EUROCRYPT *'00*, volume 1807 of *LNCS*, pages 139–155. Springer-Verlag, 2000.
3. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
4. M. Bellare and P. Rogaway. Optimal asymmetric encryption. In A. De Santis, editor, *Advances in Cryptology* - EUROCRYPT *'94*, volume 950 of *LNCS*, pages 92–111. Springer-Verlag, 1995.
5. M. Bellare and P. Rogaway. The exact security of digital signatures - how to sign with RSA and Rabin. In U. Maurer, editor, *Advances in Cryptology* - EUROCRYPT *'96*, volume 1070 of *LNCS*, pages 399–416. Springer-Verlag, 1996.
6. L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, May 1986.
7. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proc. of the 30th STOC*, pages 209–218. ACM Press, 1998.
8. B. Chor and R.L. Rivest. A knapsack type public key cryptosystem based on arithmetic in finite fields. In G. Blakley and D. Chaum, editors, *Advances in Cryptology* - CRYPTO *'84*, volume 196 of *LNCS*, pages 54–65. Springer-Verlag, 1985.
9. J.-S. Coron. On the exact security of full domain hash. In M. Bellare, editor, *Advances in Cryptology* - CRYPTO *'00*, number 1880 in LNCS, pages 229–235. Springer-Verlag, 2000.
10. J. Daemen and V. Rijmen. *The Design of Rijndael*. Information Security and Cryptography. Springer-Verlag, 2002.
11. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Advances in Cryptology* - CRYPTO *'86*, volume 263 of *LNCS*, pages 186–194. Springer-Verlag, 1987.
12. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. In Kilian [16], pages 260–274.
13. S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.
14. S. Goldwasser, S. Micali, and R.L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
15. L. Keliher. Refined analysis of bounds related to linear and differential cryptanalysis for the AES. In H. Dobbertin, V. Rijmen, and A. Sowa, editors, *Fourth Conference on the Advanced Encryption Standard* - AES*4*, volume 3373 of *LNCS*, pages 42–57. Springer-Verlag, 2005.
16. J. Kilian, editor. *Advances in Cryptology* - CRYPTO *2001*, volume 2139 of *LNCS*. Springer-Verlag, 2001.
17. N. Koblitz and A. Menezes. Another look at "provable security". II. Available on the IACR eprint archive: http://eprint.iacr.org/2006/229.pdf, July 2006.
18. N. Koblitz and A. Menezes. Another look at "provable security". *Journal of Cryptology*, 20(1):3–37, 2007.
19. H. Krawczyk, editor. *Advances in Cryptology* - CRYPTO *'98*, volume 1462 of *LNCS*. Springer-Verlag, 1998.
20. M. Matsui. Linear cryptanalysis method for DES cipher. In T. Helleseth, editor, *Advances in Cryptology* - EUROCRYPT *'93*, volume 765 of *LNCS*, pages 386–397. Springer-Verlag, 1994.
21. National Bureau of Standards, U. S. Department of Commerce. *Data Encryption Standard*, 1977.
22. National Insitute of Standards and Technology. *FIPS Publication 180-1: Secure Hash Standard*, 1995.
23. K. Ohta and T. Okamoto. On concrete security treatment of signatures derived from identification. In Krawczyk [19], pages 354–369.
24. D. Pointcheval. *Advanced Course on Contemporary Cryptology - Provable Security for Public Key Schemes*, pages 133–189. Birkhäuser Publishers, June 2005. Available at http://www.di.ens.fr/∼pointche/.
25. M.O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. MIT/LCS/TR-212,MIT Laboratory for Computer Science, 1979.
26. C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In J. Feigenbaum, editor, *Advances in Cryptology* - CRYPTO *'91*, volume 576 of *LNCS*, pages 433–444. Springer-Verlag, 1991.
27. R.L. Rivest. The MD5 message-digest algorithm. Request for Comments (RFC) 1321, April 1992. Presented at the rump session of CRYPTO'91.
28. V. Shoup. OAEP reconsidered. In Kilian [16], pages 239–259.
29. V. Shoup. OAEP reconsidered. *Journal of Cryptology*, 15(4):223–249, 2002.
30. V. Shoup. Sequences of games: A tool for taming complexity of security proofs, 2006. Available on `http://shoup.net`.
31. A. Sidorenko and B. Schoenmakers. Concrete security of the Blum-Blum-Shub pseudorandom generator. In N.P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *LNCS*, pages 355–375. Springer-Verlag, 2005.
32. S. Vaudenay. Cryptanalysis of the Chor-Rivest cryptosystem. In Krawczyk [19], pages 243–256.
33. U. Vazirani and V. Vazirani. Efficient and secure pseudo-random number generation (extended abstract). In *Proceedings of FOCS'84*, pages 458–463. IEEE, 1985.