Optimal security proofs for PSS and other signature schemes

Jean-Sébastien Coron
Gemplus Card International
34 rue Guynemer
Issy-les-Moulineaux, F-92447, France
jean-sebastien.coron@gemplus.com

Abstract. The Probabilistic Signature Scheme (PSS) designed by Bellare and Rogaway is a signature scheme provably secure against chosen message attacks in the random oracle model, with a security level equivalent to RSA. In this paper, we derive a new security proof for PSS in which a much shorter random salt is used to achieve the same security level, namely we show that $\log_2 q_{sig}$ bits suffice, where q_{sig} is the number of signature queries made by the attacker. When PSS is used with message recovery, a better bandwidth is obtained because longer messages can now be recovered. Moreover, we show that this size is optimal: if less than $\log_2 q_{sig}$ bits of random salt are used, PSS is still provably secure but no security proof can be tight. This result is based on a new technique which shows that other signature schemes such as the Full Domain Hash scheme and Gennaro-Halevi-Rabin's scheme have optimal security proofs.

Key-words: Probabilistic Signature Scheme, provable security, random oracle model.

1 Introduction

Since the invention of public key cryptography in the seminal Diffie-Hellman paper [8], significant research endeavors were devoted to the design of practical and provably secure schemes. A proof of security is usually a computational reduction from solving a well established problem to breaking the cryptosystem. Well established problems of cryptographic relevance include factoring large integers, computing discrete logarithms in prime order groups, or extracting roots modulo a composite integer.

For digital signature schemes, the strongest security notion was defined by Goldwasser, Micali and Rivest in [12], as *existential unforgeability under an adaptive chosen message attack*. This notion captures the property that an attacker cannot produce a valid signature, even after obtaining the signature of (polynomially many) messages of his choice.

Goldwasser, Micali and Rivest proposed in [12] a signature scheme based on signature trees which provably meets this definition. The efficiency of the scheme was later improved by Dwork and Naor [9], and Cramer and Damgård [6]. A significant drawback of those signature schemes is that the signature of a message depends on previously signed messages: the signer must thus store information relative to the signatures he generates as time goes by. Gennaro, Halevi and Rabin presented in [11] a new hash-and-sign scheme provably secure against adaptive chosen message attacks which is both state-free and efficient. Its security is based on the strong-RSA assumption. Cramer and Shoup presented in [7] a signature scheme provably secure against adaptive chosen message attacks, which is also state-free, efficient, and based on the strong-RSA assumption.

The random oracle model, introduced by Bellare and Rogaway in [1], is a theoretical framework allowing to prove the security of hash-and-sign signature schemes. In this model, the hash function is seen as an oracle which outputs a random value for each new query.

Bellare and Rogaway defined in [2] the Full Domain Hash (FDH) signature scheme, which is provably secure in the random oracle model assuming that inverting RSA is hard. [2] also introduced the Probabilistic Signature Scheme (PSS), which offers better security guarantees than FDH. Similarly, Pointcheval and Stern [18] proved the security of discrete-log based signature schemes in the random oracle model (see also [15] for a concrete treatment). However, security proofs in the random oracle are not real proofs, since the random oracle is replaced by a well defined hash function in practice; actually, Canetti, Goldreich and Halevi [4] showed that a security proof in the random oracle model does not necessarily imply that a scheme is secure in the real world.

For practical applications of provably secure schemes, the tightness of the security reduction must be taken into account. A security reduction is tight when breaking the signature scheme leads to solving the well established problem with probability close to one. In this case, the signature scheme is almost as secure as the well established problem. On the contrary, if the above probability is too small, the guarantee on the signature scheme will be weak; in which case larger security parameters must be used, thereby decreasing the efficiency of the scheme.

The security reduction of [2] for Full Domain Hash bounds the probability ε of breaking FDH in time t by $(q_{hash} + q_{sig}) \cdot \varepsilon'$ where ε' is the probability of inverting RSA in time t' close to t and where q_{hash} and q_{sig} are the number of hash queries and signature queries performed by the forger. This was later improved in [5] to $\varepsilon \simeq q_{sig} \cdot \varepsilon'$, which is a significant improvement since in practice q_{sig} happens to be much smaller than q_{hash}. However, FDH's security reduction is still not tight, and FDH is still not as secure as inverting RSA.

On the contrary, PSS is almost as secure as inverting RSA ($\varepsilon \simeq \varepsilon'$). Additionally, for PSS to have a tight security proof in [2], the random salt used to generate the signature must be of length at least $k_0 \simeq 2 \cdot \log_2 q_{hash} + \log_2 1/\varepsilon'$, where q_{hash} is the number of hash queries requested by the attacker and ε' the probability of inverting RSA within a given time bound. Taking $q_{hash} = 2^{60}$ and $\varepsilon' = 2^{-60}$ as in [2], we obtain a random salt of size $k_0 = 180$ bits. In this paper, we show that PSS has actually a tight security proof for a random salt as short as $\log_2 q_{sig}$ bits, where q_{sig} is the number of signature queries made by the attacker. For example, for an application in which at most one billion signatures will be generated, $k_0 = 30$ bits of random salt are actually sufficient to guarantee the same level of security as RSA, and taking a longer salt *does not* increase the security level. When PSS is used with message recovery, we obtain a better bandwidth because a larger message can now be recovered when verifying the signature.

Moreover, we show that this size is optimal: if less than $\log_2 q_{sig}$ bits of random salt are used, PSS is still provably secure, but PSS cannot have exactly the same security level as RSA. First, using a new technique, we derive an upper bound for the security of FDH, which shows that the security proof in [5] with $\varepsilon \simeq q_{sig} \cdot \varepsilon'$ is optimal. In other words, it is not possible to further improve the security proof of FDH in order to obtain a security level equivalent to RSA. This answers the open question raised by Bellare and Rogaway in [2], about the existence of a better security proof for FDH: as opposed to PSS, FDH *cannot* be proven as secure as inverting RSA. The technique also applies to other signature schemes such as Gennaro-Halevi-Rabin's scheme [11] and Paillier's signature scheme [16]. To our knowledge, this is the first result concerning optimal security proofs. Then, using the upper bound for the security of FDH, we show that our size k_0 for the random salt in PSS is optimal: if less than $\log_2 q_{sig}$ bits are used, no security proof for PSS can be tight.

2 Definitions

In this section we briefly present some notations and definitions used throughout the paper. We start by recalling the definition of a signature scheme.

Definition 1 (signature scheme). *A signature scheme* $(\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$ *is defined as follows:*

- The key generation algorithm Gen *is a probabilistic algorithm which given* 1^k*, outputs a pair of matching public and private keys,* (pk, sk)*.*

- The signing algorithm Sign *takes the message* M *to be signed, the public key* pk *and the private key* sk*, and returns a signature* $x = \texttt{Sign}_{pk,sk}(M)$*. The signing algorithm may be probabilistic.*

- The verification algorithm Verify *takes a message* M*, a candidate signature* x' *and* pk*. It returns a bit* $\texttt{Verify}_{pk}(M, x')$*, equal to one if the signature is accepted, and zero otherwise. We require that if* $x \leftarrow \texttt{Sign}_{pk,sk}(M)$*, then* $\texttt{Verify}_{pk}(M, x) = 1$*.*

In the previously introduced *existential unforgeability under an adaptive chosen message attack* scenario, the forger can dynamically obtain signatures of messages of his choice and attempts to output a valid forgery. A *valid forgery* is a message/signature pair (M, x) such that $\texttt{Verify}_{pk}(M, x) = 1$ whereas the signature of M was never requested by the forger.

A significant line of research for proving the security of signature schemes is the previously introduced random oracle model, where resistance against adaptive chosen message attacks is defined as follows [1]:

Definition 2. *A forger* $\mathcal{F}$ *is said to* $(t, q_{hash}, q_{sig}, \varepsilon)$*-break the signature scheme* (Gen, Sign, Verify) *if after at most* $q_{hash}(k)$ *queries to the hash oracle,* $q_{sig}(k)$ *signatures queries and* $t(k)$ *processing time, it outputs a valid forgery with probability at least* $\varepsilon(k)$ *for all* $k \in \mathbb{N}$*.*

and quite naturally:

Definition 3. *A signature scheme* (Gen, Sign, Verify) *is* $(t, q_{sig}, q_{hash}, \varepsilon)$*-secure if there is no forger who* $(t, q_{hash}, q_{sig}, \varepsilon)$*-breaks the scheme.*

The RSA cryptosystem, invented by Rivest, Shamir and Adleman [19], is the most widely used cryptosystem today:

Definition 4 (The RSA cryptosystem). *The RSA cryptosystem is a family of trapdoor permutations, specified by:*

- The RSA generator $\mathcal{RSA}$*, which on input* 1^k*, randomly selects two distinct* $k/2$*-bit primes* p *and* q *and computes the modulus* $N = p \cdot q$*. It randomly picks an encryption exponent* $e \in \mathbb{Z}^*_{\phi(N)}$ *and computes the corresponding decryption exponent* d *such that* $e \cdot d = 1 \bmod \phi(N)$*. The generator returns* (N, e, d)*.*

- The encryption function $f : \mathbb{Z}^*_N \to \mathbb{Z}^*_N$ *defined by* $f(x) = x^e \bmod N$*.*

- The decryption function $f^{-1} : \mathbb{Z}^*_N \to \mathbb{Z}^*_N$ *defined by* $f^{-1}(y) = y^d \bmod N$*.*

FDH was the first practical and provably secure signature scheme based on RSA. It is defined as follows: the key generation algorithm, on input 1^k, runs $\mathcal{RSA}(1^k)$ to obtain (N, e, d). It outputs (pk, sk), where the public key pk is (N, e) and the private key sk is (N, d). The signing and verifying algorithms use a hash function $H : \{0,1\}^* \to \mathbb{Z}^*_N$ which maps bit strings of arbitrary length to the set of invertible integers modulo N.

$\texttt{SignFDH}_{N,d}(M)$	$\texttt{VerifyFDH}_{N,e}(M, x)$
$y \leftarrow H(M)$	$y \leftarrow x^e \bmod N$
return $y^d \bmod N$	if $y = H(M)$ then return 1 else return 0.

FDH is provably secure in the random oracle model, assuming that inverting RSA is hard. An *inverting algorithm* $\mathcal{I}$ for RSA gets as input (N, e, y) and tries to find $y^d \bmod N$. Its success probability is the probability to output $y^d \bmod N$ when (N, e, d) are obtained by running $\mathcal{RSA}(1^k)$ and y is set to $x^e \bmod N$ for some x chosen at random in $\mathbb{Z}_N^*$.

Definition 5. *An inverting algorithm $\mathcal{I}$ is said to (t, ε)-break RSA if after at most $t(k)$ processing time its success probability is at least $\varepsilon(k)$ for all $k \in \mathbb{N}$.*

Definition 6. *RSA is said to be (t, ε)-secure if there is no inverter which (t, ε)-breaks RSA.*

The following theorem [5] proves the security of FDH in the random oracle model. We include the proof in appendix A for further reference in the paper.

Theorem 1. *Assuming that RSA is (t_I, ε_I)-secure, FDH is $(t_F,\ q_{hash},\ q_{sig},\ \varepsilon_F)$-secure, with:*

$$t_I = t_F + (q_{hash} + q_{sig} + 1) \cdot \mathcal{O}(k^3) \tag{1}$$

$$\varepsilon_I = \frac{\varepsilon_F}{q_{sig}} \cdot \left(1 - \frac{1}{q_{sig} + 1}\right)^{q_{sig}+1} \tag{2}$$

The same method can be used to obtain an improved security proof for Gennaro-Halevi-Rabin's signature scheme [11] in the random oracle model and for Paillier's signature scheme [16]. From a forger which outputs a forgery with probability ε_F, the reduction succeeds in solving the hard problem with probability roughly ε_F / q_{sig}, in approximately the same time bound.

For example, if we assume that, for a given security parameter k, the probability of inverting RSA is less than 2^{-60} for a given time bound t, and if the forger is allowed to make at most 2^{60} hash queries and 2^{30} signature queries, then the probability of breaking FDH is less than 2^{-28} for a time bound close to t.

The security reduction of FDH is not tight: the probability ε_F of breaking FDH is smaller than roughly $q_{sig} \cdot \varepsilon_I$ where ε_I is the probability of inverting RSA, whereas the security reduction of PSS is tight: the probability of breaking PSS is almost the same as the probability of inverting RSA ($\varepsilon_F \simeq \varepsilon_I$).

3 New security proof for PSS

Several standards include PSS, among these are IEEE P1363a [13], a revision of ISO/IEC 9796-2, and the upcoming PKCS#1 v2.1 [17]. In this section we obtain a better security proof for PSS, in which a shorter random salt is used to generate the signature. We consider first a variant of PSS for which the security proof is simpler.

3.1 A variant of PSS

In this section we describe a variant of PSS, which we call PFDH, for Probabilistic Full Domain Hash. The scheme is similar to Full Domain Hash except that a random *salt* of k_0 bits is concatenated to the message M before hashing it. The difference with PSS is that the random salt is not recovered when verifying the signature; instead the random salt is transmitted separately. As FDH, the scheme uses a hash function $H : \{0,1\}^* \to \mathbb{Z}_N^*$.

$\texttt{SignPFDH}(M)$:	$\texttt{VerifyPFDH}(M, s, r)$:
$r \xleftarrow{R} \{0,1\}^{k_0}$	$y \leftarrow s^e \bmod N$
$y \leftarrow H(M\|r)$	if $y = H(M\|r)$ then return 1
return $(y^d \bmod N, r)$	else return 0

The following theorem proves the security of PFDH in the random oracle model, assuming that inverting RSA is hard. It shows that PFDH has a tight security proof for a random salt of length $k_0 = \log_2 q_{sig}$ bits.

Theorem 2. *Suppose that RSA is (t', ε')-secure. Then the signature scheme $\texttt{PFDH}[k_0]$ is $(t, q_{hash}, q_{sig}, \varepsilon)$-secure, where:*

$$t = t' - (q_{hash} + q_{sig}) \cdot \mathcal{O}(k^3) \tag{3}$$

$$\varepsilon = \varepsilon' \cdot \left(1 + 6 \cdot q_{sig} \cdot 2^{-k_0}\right) \tag{4}$$

Proof. Let $\mathcal{F}$ be a forger which $(t, q_{sig}, q_{hash}, \varepsilon)$-breaks PFDH. We construct an inverter I which (t', ε')-breaks RSA. The inverter receives as input (N, e, η) and must output $\eta^d \bmod N$. We assume that the forger never repeats a hash query. However, the forger may repeat a signature query, in order to obtain the signature of M with distinct integers r. The inverter $\mathcal{I}$ maintains a counter i, initially set to zero.

When a message M appears for the first time in a hash query or a signature query, the inverter increments the counter i and sets $M_i \leftarrow M$. Then, the inverter generates a list L_i of q_{sig} random integers in $\{0,1\}^{k_0}$.

When the forger makes a hash query for $M_i\|r$, we distinguish two cases. If r belongs to the list L_i, the inverter generates a random $x \in \mathbb{Z}_N^*$ and returns $H(M_i\|r) = x^e \bmod N$. Otherwise, the inverter generates a random $x \in \mathbb{Z}_N^*$ and returns $\eta \cdot x^e \bmod N$. Consequently, for each message M_i, the list L_i contains the integers $r \in \{0,1\}^{k_0}$ such that the inverter knows the signature x corresponding to $M_i\|r$.

When the forger makes a signature query for M_i, the inverter picks a random r in L_i and discards it from the list. Since the list contains initially q_{sig} integers and there are at most q_{sig} signature queries, this is always possible. If there was already a hash query for $M_i\|r$, we have $H(M_i\|r) = x^e \bmod N$ and the inverter returns the signature x. Otherwise the inverter generates a random $x \in \mathbb{Z}_N^*$, sets $H(M_i\|r) = x^e \bmod N$ and returns the signature x.

When the forger outputs a forgery (M, s, r), we assume that it has already made a hash query for M, so $M = M_i$ for a given i. Otherwise, the inverter goes ahead and makes the hash query for $M\|r$. Then if r does not belong to the list L_i, we have $H(M_i\|r) = \eta \cdot x^e \bmod N$. From $s = H(M_i\|r)^d = \eta^d \cdot x \bmod N$, we obtain $\eta^d = s/x \bmod N$ and the inverter succeeds in outputting $\eta^d \bmod N$.

Since the forger has not made any signature query for the message M_i in the forgery (M_i, s, r), the forger has no information about the q_{sig} random integers in the list L_i. Therefore, the probability that r does not belong to L_i is $(1-2^{-k_0})^{q_{sig}}$. If the size k_0 of the random salt is greater than $\log_2 q_{sig}$, we obtain if $q_{sig} \geq 2$:

$$\left(1 - 2^{-k_0}\right)^{q_{sig}} \geq \left(1 - \frac{1}{q_{sig}}\right)^{q_{sig}} \geq \frac{1}{4}$$

Since the forger outputs a forgery with probability ε, the success probability ε' of the inverter is then at least $\varepsilon/4$, which shows that for $k_0 \geq \log_2 q_{sig}$ the probability of breaking PFDH is almost the same as the probability of inverting RSA.

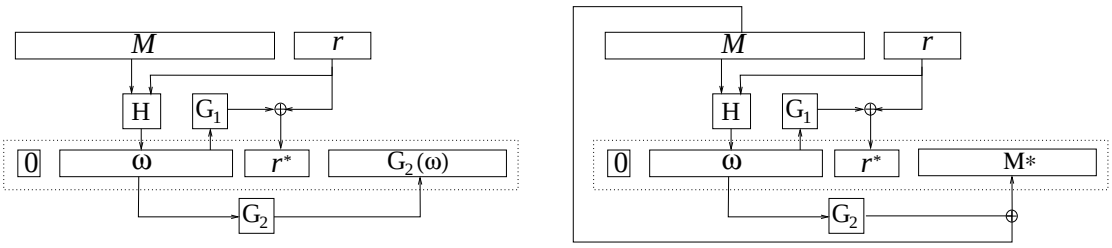

Fig. 1. PSS (left) and PSS-R (right)

For the general case, *i.e.* if we do not assume $k_0 \geq \log_2 q_{sig}$, we generate fewer than q_{sig} random integers in the list L_i, so that the salt r in the forgery (M_i, s, r) belongs to L_i with lower probability. More precisely, starting from an empty list L_i, the inverter generates with probability β a random $r \leftarrow \{0,1\}^{k_0}$, adds it to L_i, and starts again until the list L_i contains q_{sig} elements. Otherwise (so with probability $1-\beta$) the inverter stops adding integers to the list. The number a_i of integers in L_i is then a random variable following a geometric law of parameter β:

$$\Pr[a_i = j] = \begin{cases} (1-\beta)\cdot\beta^j & \text{if } j < q_{sig} \\ \beta^{q_{sig}} & \text{if } j = q_{sig} \end{cases} \tag{5}$$

The inverter answers a signature query for M_i if the corresponding list L_i contains one more integer, which happens with probability β (otherwise the inverter must abort). Consequently, the inverter answers all the signature queries with probability greater than $\beta^{q_{sig}}$. Note that if $\beta = 1$, the setting boils down to the previous case: all the lists L_i contain exactly q_{sig} integers, and the inverter answers all the signature queries with probability one.

The probability that r in the forgery (M_i, s, r) does not belong to the list L_i is then $(1-2^{-k_0})^j$, when the length a_i of L_i is equal to j. The probability that r does not belong to L_i is then:

$$f(\beta) = \sum_{j=0}^{q_{sig}} \Pr[a_i = j] \cdot \left(1 - 2^{-k_0}\right)^j \tag{6}$$

Since the forger outputs a forgery with probability ε, the success probability of the inverter is at least $\varepsilon \cdot \beta^{q_{sig}} \cdot f(\beta)$. We select a value of β which maximizes this success probability; in appendix B we show that for any (q_{sig}, k_0), there exists β_0 such that:

$$\beta_0^{q_{sig}} \cdot f(\beta_0) \geq \frac{1}{1 + 6 \cdot q_{sig} \cdot 2^{-k_0}} \tag{7}$$

which gives (4). The running time of $\mathcal{I}$ is the running time of $\mathcal{F}$ plus the time necessary to compute the integers $x^e \bmod N$, which gives (3).

3.2 Application to PSS

The signature scheme PSS is parameterized by the integers k_0 and k_1. The key generation is identical to FDH. The signing and verifying algorithms use two hash function $H : \{0,1\}^* \to \{0,1\}^{k_1}$ and $G : \{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$. Let G_1 be the function which on input $\omega \in \{0,1\}^{k_1}$ returns the first k_0 bits of $G(\omega)$, whereas G_2 is the function returning the remaining $k - k_0 - k_1 - 1$ bits of $G(\omega)$. The scheme is illustrated in figure 1.

$\texttt{SignPSS}(M)$:	$\texttt{VerifyPSS}(M, x)$:
$r \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$	$y \leftarrow x^e \bmod N$
$\omega \leftarrow H(M\|r)$	Break up y as $b\|\omega\|r^*\|\gamma$
$r^* \leftarrow G_1(\omega) \oplus r$	Let $r \leftarrow r^* \oplus G_1(\omega)$
$y \leftarrow 0\|\omega\|r^*\|G_2(\omega)$	if $H(M\|r) = \omega$ and $G_2(\omega) = \gamma$ and $b = 1$
return $y^d \bmod N$	then return 1 else return 0

The following theorem [2] proves the security of PSS in the random oracle model:

Theorem 3. *Assuming that RSA is (t', ε')-secure, the signature scheme $\texttt{PSS}[k_0, k_1]$ is $(t, q_{sig}, q_{hash}, \varepsilon)$-secure, where :*

$$t = t' - (q_{hash} + q_{sig} + 1) \cdot k_0 \cdot \mathcal{O}(k^3)$$
$$\varepsilon = \varepsilon' + 3 \cdot (q_{sig} + q_{hash})^2 \cdot \left(2^{-k_0} + 2^{-k_1}\right)$$

Theorem 3 shows that for PSS to be as secure as RSA (*i.e.* $\varepsilon' \simeq \varepsilon$), it must be the case that $(q_{sig} + q_{hash})^2 \cdot \left(2^{-k_0} + 2^{-k_1}\right) < \varepsilon'$, which gives $k_0 \geq k_{min}$ and $k_1 \geq k_{min}$, where:

$$k_{min} = 2 \cdot \log_2(q_{hash} + q_{sig}) + \log_2 \frac{1}{\varepsilon'} \tag{8}$$

Taking $q_{hash} = 2^{60}$, $q_{sig} = 2^{30}$ and $\varepsilon' = 2^{-60}$ as in [2], we obtain that k_0 and k_1 must be greater than $k_{min} = 180$ bits.

The following theorem shows that PSS can be proven as secure as RSA for a much shorter random salt, namely $k_0 = \log_2 q_{sig}$ bits, which for $q_{sig} = 2^{30}$ gives $k_0 = 30$ bits. The minimum value for k_1 remains unchanged. The proof is very similar to the proof of theorem 2 for PFDH and is given in appendix C.

Theorem 4. *Assuming that RSA is (t', ε')-secure, the signature scheme $\texttt{PSS}[k_0, k_1]$ is $(t, q_{sig}, q_{hash}, \varepsilon)$-secure, where :*

$$t = t' - (q_{hash} + q_{sig}) \cdot k_1 \cdot \mathcal{O}(k^3) \tag{9}$$
$$\varepsilon = \varepsilon' \cdot \left(1 + 6 \cdot q_{sig} \cdot 2^{-k_0}\right) + 2 \cdot (q_{hash} + q_{sig})^2 \cdot 2^{-k_1} \tag{10}$$

3.3 Discussion

In figure 2 we plot $\log_2 \varepsilon'/\varepsilon$ as a function of the size k_0 of the salt, which depicts the relative security of PSS compared to RSA, for $q_{sig} = 2^{30}$, and $k_1 > k_{min}$. For $k_0 = 0$, we reach the security level of FDH, where approximately $\log_2 q_{sig}$ bits of security are lost compared to RSA. For k_0 comprised between zero and $\log_2 q_{sig}$, we gain one bit of security when k_0 increases by one bit. And for k_0 greater than $\log_2 q_{sig}$, the security level of PSS is almost the same as inverting RSA. This shows that PSS has a tight security proof as soon as the salt size reaches $\log_2 q_{sig}$, and using larger salts does not further improve security. For the signer, q_{sig} represents the maximal number of signatures which can be generated for a given public-key. For example, for an application in which at most one billion signatures will be generated, $k_0 = 30$ bits of random salt are actually sufficient to guarantee the same level of security as RSA, and taking a larger salt does not increase the security level.

More precisely, taking $k_0 = \log_2 q_{sig}$ and $k_1 = k_{min}$ where k_{min} is given by (8), we obtain that the probability of breaking PSS in time less than t, is less than $\varepsilon = 9 \cdot \varepsilon'$, where ε' is the probability of inverting RSA in time close to t. Therefore with those parameters PSS

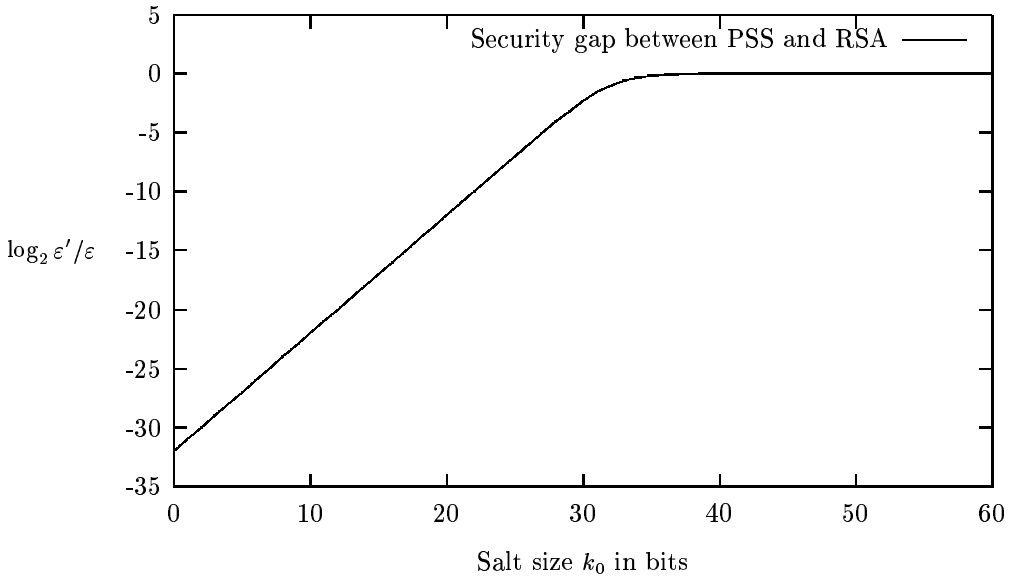

Fig. 2. Security gap between PSS and RSA: $\log_2 \varepsilon'/\varepsilon$ as a function of the salt size k_0 for $q_{sig} = 2^{30}$ signature queries.

is almost as secure as inverting RSA[1]. Taking $q_{hash} = 2^{60}$, $q_{sig} = 2^{30}$ and $\varepsilon' = 2^{-60}$ for a 1024-bit modulus as in [2], we can take $k_1 = k_{min} = 180$ bits and $k_0 = \log_2 q_{sig} = 30$ bits.

PSS-R is a variant of PSS which provides message recovery; the scheme is illustrated in figure 1. The goal is to save on the bandwidth: instead of transmitting the message separately, the message is recovered when verifying the signature.The security proof for PSS-R is almost identical to the security proof of PSS, and PSS-R achieves the same security level as PSS. Consequently, using the same parameters as for PSS with a 1024-bits RSA modulus, 813 bits of message can now be recovered when verifying the signature (instead of 663 bits with the previous security proof).

4 Optimal security proof for FDH

In section 2 we have seen that the security proof of theorem 1 for FDH is still not tight: the probability ε_F of breaking FDH is smaller than roughly $q_{sig} \cdot \varepsilon_I$ where ε_I is the probability of inverting RSA, whereas the security reduction of PSS is tight: the probability of breaking PSS is almost the same as the probability of inverting RSA ($\varepsilon_F \simeq \varepsilon_I$). An interesting question is whether it is possible to obtain a better security bound for FDH. In particular, is it possible to show that FDH is as secure as inverting RSA ?

In this section we show that the security proof of theorem 1 for FDH is optimal, *i.e.* there is no better reduction from inverting RSA to breaking FDH, and one cannot avoid loosing the q_{sig} factor in the probability bound. A possible direction would be to demonstrate an attack against FDH which would not apply to inverting RSA. More precisely, if we could prove that the best possible attack against FDH is q_{sig} times faster than the best possible attack against RSA, this would show that FDH is indeed less secure than RSA and that the previous security proof for FDH is optimal. But actually we don't know any attack on FDH, faster than factoring N.

[1] The factor 9 is not relevant here, because it represents less than 4 bits of security. To obtain $\varepsilon' \simeq \varepsilon$, we can take $k_0 = \log_2 q_{sig} + 8$ and $k_1 = k_{min} + 8$, which gives $\varepsilon' = 1.04 \cdot \varepsilon$.

Instead, in order to show that there is no better reduction from inverting RSA to breaking FDH, we will use a similar approach as Boneh and Venkatesan in [3] for disproving the equivalence between inverting low-exponent RSA and factoring. They show that any efficient algebraic reduction from factoring to inverting low-exponent RSA can be converted into an efficient factoring algorithm. Such reduction is an algorithm $\mathcal{A}$ which factors N using an e-th root oracle for N. They show how to convert algorithm $\mathcal{A}$ into an algorithm $\mathcal{B}$ that factors integers without using the e-th root oracle. Thus, unless factoring is easy, inverting low-exponent RSA cannot be equivalent to factoring under algebraic reductions.

Similarly, we show that any better reduction from inverting RSA to breaking FDH can be converted into an efficient RSA inverting algorithm. Such reduction is an algorithm $\mathcal{R}$ which uses a forger as an oracle in order to invert RSA. We show how to convert $\mathcal{R}$ into an algorithm $\mathcal{I}$ which inverts RSA without using the oracle forger. Consequently, if inverting RSA is hard, there is no such better reduction for FDH, and the reduction of theorem 1 must be optimal.

Our technique is the following. Recall that resistance against adaptive chosen message attacks is considered, so the forger is allowed to make signature queries for messages of its choice, which must be answered by the reduction $\mathcal{R}$. Eventually the forger outputs a forgery, and the reduction must invert RSA. Therefore we first ask the reduction to sign a message M and receive its signature s, then we rewind the reduction to the state in which it was before the signature query, and we send s as a forgery for M. This is a true forgery for the reduction, because after the rewind there was no signature query for M, so eventually the reduction inverts RSA. Consequently, we have constructed from $\mathcal{R}$ an algorithm $\mathcal{I}$ which inverts RSA without using any forger. Actually, this technique allows to simulate a forger with respect to $\mathcal{R}$, without being able to break FDH. However, the simulation is not perfect, because it outputs a forgery only for messages which can be signed by the reduction, whereas a real forger outputs the forgery of a message which the reduction may or may not be able to sign.

We quantify the efficiency of the reduction by giving the probability that the reduction inverts RSA using a forger that $(t_F, q_{hash}, q_{sig}, \varepsilon_F)$-breaks the signature scheme, within an additional running time of t_R:

Definition 7. *We say that a reduction algorithm $\mathcal{R}$ $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R)$-reduces inverting RSA to breaking FDH if upon input (N, e, y) and after running any forger that $(t_F, q_{hash}, q_{sig}, \varepsilon_F)$-breaks FDH, the reduction outputs $y^d \bmod N$ with probability greater than ε_R, within an additional running time of t_R.*

In the above definition, t_R is the running time of the reduction algorithm only and does not include the running time of the forger. Eventually, the time needed to invert RSA is $t_F + t_R$, where t_F is the running time of the forger. For example, the reduction of theorem 1 for FDH $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R)$-reduces inverting RSA to breaking FDH with $t_R(k) = (q_{hash} + q_{sig}) \cdot \mathcal{O}(k^3)$ and $\varepsilon_R = \varepsilon_F/(4 \cdot q_{sig})$.

The following theorem shows that from any such reduction $\mathcal{R}$ we can invert RSA with probability greater than roughly $\varepsilon_R - \varepsilon_F/q_{sig}$, in roughly the same time bound. The term ε_F/q_{sig} is due to the fact that our simulation of a forger is not perfect. This also corresponds to the success probability of the reduction in theorem 1. This means that if the success probability ε_R of the reduction is greater than ε_F/q_{sig}, we obtain an algorithm which inverts RSA without using the forger. Therefore, if inverting RSA is hard, the success probability of the reduction cannot be greater than roughly ε_F/q_{sig}, and the reduction of theorem 1 must be optimal.

Theorem 5. *Let $\mathcal{R}$ be a reduction which $(t_R, q_{hash}, q_{sig}, \varepsilon_R, \varepsilon_F)$-reduces inverting RSA to breaking FDH. $\mathcal{R}$ runs the forger only once. From $\mathcal{R}$ we can construct an algorithm which (t_I, ε_I)-inverts RSA, with:*

$$t_I = 2 \cdot t_R \tag{11}$$

$$\varepsilon_I = \varepsilon_R - \varepsilon_F \cdot \frac{\exp(-1)}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{q_{hash}}\right)^{-1} \tag{12}$$

Proof. From $\mathcal{R}$ we build an algorithm $\mathcal{I}$ which inverts RSA, without using a forger for FDH. We receive as input (N, e, y) and our goal is to output $y^d \bmod N$ using $\mathcal{R}$. We select q_{hash} distinct messages $M_1, \ldots, M_{q_{hash}}$ of length $\mathcal{O}(k)$ and starts running $\mathcal{R}$ with (N, e, y).

First we ask $\mathcal{R}$ to hash the q_{hash} messages $M_1, \ldots, M_{q_{hash}}$, and obtain the hash values $h_1, \ldots, h_{q_{hash}}$. We select a random integer $\beta \in [1, q_{hash}]$ and a random sequence α of q_{sig} integers in $[1, q_{hash}] \setminus \{\beta\}$, which we denote $\alpha = (\alpha_1, \ldots, \alpha_{q_{sig}})$. We select a random integer $i \in [1, q_{sig}]$ and define the sequence of i integers $\alpha' = (\alpha_1, \ldots, \alpha_{i-1}, \beta)$. Then we make the i signature queries corresponding to α' to $\mathcal{R}$ and receive from $\mathcal{R}$ the corresponding signatures, the last one being the signature s_β of M_β. For example, if $\alpha' = (3, 2)$, this corresponds to making a signature query for M_3 first, and then for M_2.

Then we rewind $\mathcal{R}$ to the state it was after the hash queries, and this time, we make the q_{sig} signature queries corresponding to α. If $\mathcal{R}$ has answered all the signature queries, then with probability ε_F, we send (M_β, s_β) as a forgery to $\mathcal{R}$. This is a true forgery for $\mathcal{R}$ because after the rewind of $\mathcal{R}$, there was no signature query for M_β. Eventually $\mathcal{R}$ inverts RSA and outputs $y^d \bmod N$.

We denote by $\mathcal{Q}$ the set of sequences of signature queries which are correctly answered by $\mathcal{R}$ after the hash queries, in time less than t_R. If a sequence of signature queries is correctly answered by $\mathcal{R}$, then the same sequence without the last signature query is also correctly answered, so for any $(\alpha_1, \ldots, \alpha_j) \in \mathcal{Q}$, we have $(\alpha_1, \ldots, \alpha_{j-1}) \in \mathcal{Q}$. Let us denote by `ans` the event $\alpha \in \mathcal{Q}$, which corresponds to $\mathcal{R}$ answering all the signature queries after the rewind, and by `ans'` the event $\alpha' \in \mathcal{Q}$, which corresponds to $\mathcal{R}$ answering all the signature queries before the rewind.

Let us consider a forger which makes the same hash queries, the same signature queries corresponding to α, and outputs a forgery for M_β with probability ε_F. By definition, when interacting with such a forger, $\mathcal{R}$ would output $y^d \bmod N$ with probability at least ε_R. After the rewind, $\mathcal{R}$ sees exactly the same transcript as when interacting with this forger, except if event `ans` is true and `ans'` is false: in this case, the forger outputs a forgery with probability ε_F, whereas our simulation does not output a forgery. Consequently, when interacting with our simulation of a forger, $\mathcal{R}$ outputs $y^d \bmod N$ with probability at least:

$$\varepsilon_R - \varepsilon_F \cdot \Pr[\texttt{ans} \wedge \neg \texttt{ans'}] \tag{13}$$

Lemma 1. *Let $\mathcal{Q}$ be a set of sequences of at most n integers in $[1, k]$, such that for any sequence $(\alpha_1, \ldots, \alpha_j) \in \mathcal{Q}$, we have $(\alpha_1, \ldots, \alpha_{j-1}) \in \mathcal{Q}$. Then the following holds:*

$$\Pr_{\substack{i \leftarrow [1,n] \\ (\alpha_1, \ldots, \alpha_n, \beta) \leftarrow [1,k]^{n+1}}} [(\alpha_1, \ldots, \alpha_n) \in \mathcal{Q} \wedge (\alpha_1, \ldots, \alpha_{i-1}, \beta) \notin \mathcal{Q}] \leq \frac{\exp(-1)}{n}$$

Proof. The proof is given in appendix D.

Using lemma 1 with $n = q_{sig}$ and $k = q_{hash}$, we obtain:

$$\Pr[\texttt{ans} \wedge \neg \texttt{ans'}] \leq \frac{\exp(-1)}{q_{sig}} \left(1 - \frac{q_{sig}}{q_{hash}}\right)^{-1} \tag{14}$$

The term $(1 - q_{sig}/q_{hash})$ in equation (14) is due to the fact that we select $\alpha_1, \ldots, \alpha_{q_{sig}}$ in $[1, q_{hash}] \setminus \{\beta\}$ whereas in lemma 1 the integers are selected in $[1, q_{hash}]$. From equations (13) and (14) we obtain that $\mathcal{I}$ succeeds with probability greater than ε_I given by (12). Because of the rewind, the running time of $\mathcal{I}$ is at most twice the running time of $\mathcal{R}$, which gives (11).

4.1 Discussion

The previous theorem shows that from any reduction $\mathcal{R}$ which inverts RSA with probability ε_R when interacting with a forger which outputs a forgery with probability ε_F, we can invert RSA with probability roughly $\varepsilon_R - \varepsilon_F/q_{sig}$, in roughly the same time bound, without using a forger. For simplicity, we neglect here the factors $\exp(-1)$ and $(1 - q_{sig}/q_{hash})$. Moreover we consider a forger which makes q_{sig} signature queries, and with probability $\varepsilon_F = 1$ outputs a forgery[2]. We begin by providing an asymptotic analysis, and then we illustrate the theorem with a concrete analysis, *i.e.* for a fixed size of the modulus.

Theorem 5 implies that from a polynomial time reduction $\mathcal{R}$ which succeeds with probability ε_R when interacting with this forger, we obtain a polynomial time RSA inverter $\mathcal{I}$ which succeed with probability $\varepsilon_I = \varepsilon_R - 1/q_{sig}$, without using the forger. If inverting RSA is hard, the success probability ε_I of the polynomial time inverter must be negligible. Consequently, the success probability ε_R of the reduction must be less than $1/q_{sig} + \mathtt{negl}$. This shows that from a forger which outputs a forgery with probability one, a polynomial time reduction cannot succeed with probability greater than $1/q_{sig} + \mathtt{negl}$. On the contrary, a tight security reduction would invert RSA with probability close to one. Here we cannot avoid the q_{sig} factor in the security proof: the security level of FDH cannot be proven equivalent to RSA.

For the concrete analysis, we need to assume a lower bound for the complexity of breaking RSA for a given key size. The running time of the best factoring algorithm known (NFS [14]) for factoring a modulus N is about

$$T_{NFS}(k) = \exp(C \cdot (\log N)^{1/3} \cdot (\log \log N)^{2/3})$$

where $C \simeq 1.923$. Therefore we might assume that RSA is (t, ε)-secure for any (t, ε) satisfying $t(k)/\varepsilon(k) < T_{NFS}(k)$. For a 1024-bit modulus, we obtain that RSA is $(t, t \cdot 2^{-86})$-secure for all $t \leq 2^{86}$. For example, the probability of inverting RSA in time 2^{26} is less than 2^{-60}.

Using the previous forger, the reduction of theorem 1 outputs $y^d \bmod N$ with probability about $1/q_{sig} = 2^{-20}$ in additional time $(q_{hash} + q_{sig}) \cdot \mathcal{O}(k^3)$. Taking $q_{hash} = 2^{40}$ and $q_{sig} = 2^{20}$, and assuming that the modular exponentiations corresponding to the term $\mathcal{O}(k^3)$ are done in unit time for a 1024-bit modulus, we get an additional running time of 2^{40}.

Let us consider another reduction $\mathcal{R}$ which, using the same forger, outputs $y^d \bmod N$ with probability ε_R in additional time t_R. Theorem 5 shows that from $\mathcal{R}$ and without using the forger, we can invert RSA in time $t_I = 2 \cdot t_R$, with probability at least $\varepsilon_R - 1/q_{sig}$. Conversely, if RSA is (t_I, ε_I)-secure, the reduction $\mathcal{R}$ cannot invert RSA with probability greater than $1/q_{sig} + \varepsilon_I$. Assume that $\mathcal{R}$ is as efficient as the reduction of theorem 1, *i.e.* its running time t_R is less than 2^{40}. This gives $t_I = 2^{41}$, and since RSA is $(2^{41}, 2^{-45})$-secure for a 1024-bit modulus, the probability that $\mathcal{R}$ outputs $y^d \bmod N$ using the previous forger cannot be greater than $1/q_{sig} + 2^{-45} \simeq 2^{-20}$. From a forger which outputs a forgery with probability one, the reduction cannot invert RSA with probability greater than roughly

[2] Such forger can be constructed by first factoring the modulus N, then computing a forgery using the factorisation of N.

$1/q_{sig} = 2^{-20}$, if the running time of the reduction is less than 2^{40}. Again, this shows that we cannot avoid the q_{sig} factor in the security proof: the security proof of theorem 1 for FDH is optimal and the security level of FDH cannot be proven equivalent to RSA.

5 Extension to any signature scheme with unique signature

Actually, our technique which consists in making a signature query for M, rewinding the forger, then sending the signature of M as a forgery, stretches beyond FDH and can be generalized and applied to any signature scheme. However, the technique works only for signature schemes in which each message has a unique signature, because otherwise the forger cannot be simulated. Namely if M has many possible signatures, our simulation sends as a forgery for M a signature s that was received from $\mathcal{R}$, whereas a real forger has no information about s (since it has not queried M for signature to $\mathcal{R}$) and can output any signature $s' \neq s$ for M. For signature schemes with unique signature, our technique shows that the reduction cannot succeed with probability greater than roughly ε_F/q_{sig}, using a forger which outputs a forgery with probability ε_F. Signature schemes with unique signature include FDH, Gennaro-Halevi-Rabin's signature scheme and Paillier's signature scheme. Note that PSS is not a signature scheme with unique signature.

However, we have so far considered reductions running a forger only once. If the reduction of theorem 1 for FDH runs the forger r times, its success probability will be roughly $r \cdot \varepsilon_F/q_{sig}$, and the total running time will be roughly r times the running time of the forger. But there might be a better reduction which would yield a better time/probability trade-off. For example, a reduction for FDH could succeed with probability almost ε_F when running a forger only twice. In this case, FDH would be almost as secure as inverting RSA. Additionally, the reduction might rewind the forger with different inputs, as for proof-of-knowledge based signature schemes [15, 18].

The following theorem shows that there is no better time/probability trade-off: for a hash-and-sign signature scheme with unique signature, a reduction allowed to run or rewind a forger at most r times cannot succeed with probability greater than roughly $r \cdot \varepsilon_F/q_{sig}$. The definitions are in appendix E and the proof of the theorem is given in appendix F.

Theorem 6. *Let $\mathcal{R}$ be a reduction which $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R)$-reduces solving a problem Π to breaking a hash-and-sign signature scheme with unique signature. $\mathcal{R}$ is allowed to run or rewind a forger at most r times. From $\mathcal{R}$ we can construct an algorithm which (t_A, ε_A)-solves Π, with:*

$$t_A = (r+1) \cdot t_R \tag{15}$$

$$\varepsilon_A = \varepsilon_R - \varepsilon_F \cdot \frac{\exp(-1) \cdot r}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{q_{hash}}\right)^{-1} \tag{16}$$

6 Security proofs for signature schemes in the standard model

The same technique can be applied for security reductions in the standard model, and we obtain the same upper bound in $1/q_{sig}$ for signature schemes with unique signature. The definitions of security against adaptive chosen message attacks are analogous in the standard model and can be found in appendix G.

The following theorem is analogous to theorem 6. It proves that for any signature scheme with unique signature, assuming the hardness of a given problem Π, any security reduction running or rewinding a forger at most r times cannot be tighter than roughly $r \cdot \varepsilon_F/q_{sig}$.

Namely a better reduction can be converted into an algorithm for solving Π, in approximately the same time bound. The proof is similar to the proof of theorem 6 and is given in appendix H

Theorem 7. *Let $\mathcal{R}$ be a reduction which $(t_R, q_{sig}, \varepsilon_F, \varepsilon_R)$-reduces solving Π to breaking a signature scheme with unique signature. $\mathcal{R}$ can run or rewind the forger at most r times. Assume that the size of the message space is at least 2^ℓ. From $\mathcal{R}$ we can construct an algorithm which (t_A, ε_A)-solves Π, with:*

$$t_A = (r+1) \cdot t_R \tag{17}$$

$$\varepsilon_A = \varepsilon_R - \varepsilon_F \cdot \frac{\exp(-1) \cdot r}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{2^\ell}\right)^{-1} \tag{18}$$

Gennaro-Halevi-Rabin's signature scheme has a tight ($\varepsilon_F \simeq \varepsilon_R$) security reduction in the standard model, but the above theorem does not apply here because the reduction of [11] requires that a message has many possible signatures. This is also the case for the Cramer-Shoup signature scheme [7]. However, we show in appendix I that the above bound in $1/q_{sig}$ is reached for a variant of Gennaro-Halevi-Rabin's scheme with unique signature, provably secure in the standard model. The variant is provably secure for short messages only (say, less than 40 bits). We do not know if there exists a *practical* signature scheme with unique signature, provably secure in the standard model and reaching the above bound.

7 Optimal security proof for PSS

In section 3.2 we have seen that $k_0 = \log_2 q_{sig}$ bits of random salt are sufficient for PSS to have a security level equivalent to RSA, and taking a larger salt does not further improve the security. An interesting question is that of knowing whether this size is optimal or not. For k_1, the output size of the hash function H, the minimum value k_{min} given by equation (8) is clearly optimal, because an attacker making q_{hash} hash queries can find a collision $H(M||0) = H(M'||0)$ with probability roughly $(q_{hash})^2 \cdot 2^{-k_1}/2$ and then forge the signature of M' using the signature of M. However, there might be a better security proof for PSS which would be tight for a shorter size k_0 of the random salt. Actually, if this size is equal to zero, the scheme becomes with unique signature, and we know from section 5 that one must loose $\log_2 q_{sig}$ bits of security compared to the security of RSA. So it seems natural to think that we need at least $\log_2 q_{sig}$ bits of random to make PSS as secure as RSA, because normally we should gain at most one bit of security for each added bit of random salt.

In this section, we show that this is indeed the case: if a shorter random salt is used, the security of PSS cannot be proven equivalent to RSA. Our technique described in section 4 does not apply directly because PSS is not a signature scheme with unique signature. However, we show in appendix J how to extend to PSS the previous upper bound for FDH. More precisely, we show that from a reduction $\mathcal{R}$ which inverts RSA in time t_R with probability ε_R when running at most r times a forger which breaks $\mathtt{PSS}[k_0, k_1]$ with probability ε_F, one can invert RSA without using the forger, with probability $\varepsilon_I = \varepsilon_R - r \cdot \varepsilon_F \cdot 2^{k_0+2}/q_{sig}$, in time $t_I = (r+1) \cdot t_R$.

Theorem 8. *Let $\mathcal{R}$ a reduction which $(t, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R)$-reduces inverting RSA to breaking $\mathtt{PSS}[k_0, k_1]$, with $q_{hash} \geq 2 \cdot q_{sig}$. The reduction can run or rewind the forger at most r times. From $\mathcal{R}$ we can construct an inverting algorithm for RSA which (t_I, ε_I)-inverts RSA,*

with:

$$t_I = (r+1) \cdot t_R \tag{19}$$

$$\varepsilon_I = \varepsilon_R - r \cdot \varepsilon_F \cdot \frac{2^{k_0+2}}{q_{sig}} \tag{20}$$

Proof. The proof is given in appendix J.

7.1 Discussion

Let consider as in section 4.1 a forger for $\mathtt{PSS}[k_0, k_1]$ which makes q_{sig} signature queries and outputs a forgery with probability $\varepsilon_F = 1/2$. Then, from a polynomial time reduction $\mathcal{R}$ which succeeds with probability ε_R when running once this forger, we obtain a polynomial time inverter which succeeds with probability $\varepsilon_I = \varepsilon_R - 2^{k_0+1}/q_{sig}$, without using the forger. If inverting RSA is hard, the success probability ε_I of the polynomial time inverter must be negligible, and therefore the success probability ε_R of the reduction must be less than $2^{k_0+1}/q_{sig} + \mathtt{negl}$. Consequently, in order to have a tight security reduction ($\varepsilon_R \simeq \varepsilon_R$), we must have $k_0 \simeq \log_2 q_{sig}$. The reduction of theorem 3.2 is consequently optimal.

Let us illustrate the theorem with concrete values. Using the previous forger, the reduction for PSS of section 3.2 inverts RSA with probability (we assume that $k_1 > k_{min}$):

$$\varepsilon_R = \frac{\varepsilon_F}{1 + 6 \cdot q_{sig} \cdot 2^{-k_0}}$$

Taking $k_0 = \log_2 q_{sig}$, we obtain that the reduction inverts RSA with probability at least $1/14$. Assuming as in section 4.1 that the modular exponentiations are performed in unit time for a 1024-bit modulus, the running time of the reduction is less than 2^{50}.

Let us consider another reduction $\mathcal{R}$ from inverting RSA to braking $\mathtt{PSS}[k_0, k_1]$, with the same running time 2^{50}, and which succeeds with probability at least ε_R using the previous forger. From theorem 8 we can construct an algorithm which inverts RSA in time 2^{51} with probability $\varepsilon_I = \varepsilon_R - 2^{k_0+1}/q_{sig}$. Assuming as in section 4.1 that RSA is $(2^{51}, 2^{-35})$-secure, the success probability of the reduction cannot be greater than $2^{k_0+1}/q_{sig} + 2^{-35}$. Consequently, to obtain the same success probability as the reduction of section 3.2, we must have $2^{k_0+1}/q_{sig} + 2^{-35} \geq 1/14$, which gives $k_0 \geq \log_2 q_{sig} - 5$. With $k_0 = \log_2 q_{sig}$, the reduction of section 3.2 is consequently optimal, up to a constant factor. To summarize, if the size k_0 of the random salt is smaller than $\log_2 q_{sig}$, PSS is still provably secure as shown in section 3.2, but the security level of PSS can not be proven equivalent to RSA.

8 Conclusion

We have described a new technique for analyzing the security proofs of signature schemes. The technique is both general and very simple and allows to derive upper bounds for security reductions using a forger as a black box, both in the random oracle model and in the standard model, for signature schemes with unique signature. We have also obtained a new criterion for a security reduction to be optimal, which may be of independent interest: we say that a security reduction is optimal if from a better reduction one can solve a difficult problem, such as inverting RSA. Our technique enables to show that the Full Domain Hash scheme, Gennaro-Halevi-Rabin's scheme and Paillier's signature scheme have an optimal security reduction in that sense. In other words, we have a matching lower and upper bound for the security reduction of those signature schemes: one cannot do better than losing a factor of q_{sig} in the security reduction.

Moreover, we have described a better security proof for PSS, in which a much shorter random salt is sufficient to achieve the same security level. This is of practical interest, since when PSS is used with message recovery, a better bandwidth is obtained because larger messages can be embedded inside the signature. Eventually, we have shown that this security proof for PSS is optimal: if a smaller random salt is used, PSS remains provably secure, but it cannot have the same level of security as RSA.

References

1. M. Bellare and P. Rogaway, *Random oracles are practical: a paradigm for designing efficient protocols*. Proceedings of the First Annual Conference on Computer and Commmunications Security, ACM, 1993.
2. M. Bellare and P. Rogaway, *The exact security of digital signatures - How to sign with RSA and Rabin*. Proceedings of Eurocrypt'96, LNCS vol. 1070, Springer-Verlag, 1996, pp. 399-416.
3. D. Boneh and R. Venkatesan, *Breaking RSA may not be equivalent to factoring*. Proceedings of Eurocrypt' 98, LNCS vol. 1403, Springer-Verlag, 1998, pp. 59–71.
4. R. Canetti, O. Goldreich and S. Halevi, *The random oracle methodology, revisited*, STOC' 98, ACM, 1998.
5. J.S. Coron, *On the exact security of Full Domain Hash*, Proceedings of Crypto'2000, LNCS vol. 1880, Springer-Verlag, 2000, pp. 229-235.
6. R. Cramer and I. Damgård, *New generation of secure and practical RSA-based signatures*, Proceedings of Crypto'96, LNCS vol. 1109, Springer-Verlag, 1996, pp. 173-185.
7. R. Cramer and V. Shoup, *Signature schemes based on the Strong RSA Assumption*, May 9, 2000, revision of the extended abstract in Proc. 6th ACM Conf. on Computer and Communications Security, 1999; To appear, ACM Transactions on Information and System Security (ACM TISSEC). Available at `http://www.shoup.net/`
8. W. Diffie and M. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory, IT-22, 6, pp. 644-654, 1976.
9. C. Dwork and M. Naor, *An efficient existentially unforgeable signature scheme and its applications*, In J. of Cryptology, 11 (3), Summer 1998, pp. 187-208.
10. FIPS 186, *Digital signature standard*, Federal Information Processing Standards Publication 186, U.S. Department of Commerce/NIST, 1994.
11. R. Gennaro, S. Halevi and T. Rabin, *Secure hash-and-sign signatures without the random oracle*, proceedings of Eurocrypt '99, LNCS vol. 1592, Springer-Verlag, 1999, pp. 123-139.
12. S. Goldwasser, S. Micali and R. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM Journal of computing, 17(2), pp. 281-308, April 1988.
13. IEEE P1363a, *Standard Specifications For Public Key Cryptography: Additional Techniques*, available at `http://www.manta.ieee.org/groups/1363`
14. A. Lenstra and H. Lenstra (eds.), *The development of the number field sieve*, Lecture Notes in Mathematics, vol 1554, Springer-Verlag, 1993.
15. K. Ohta and T. Okamoto, *On concrete security treatment of signatures derived from identification*. Prooceedings of Crypto '98, Lecture Notes in Computer Science vol. 1462, Springer-Verlag, 1998, pp. 354-369.
16. P. Paillier, *Public-key cryptosystems based on composite degree residuosity classes*. Proceedings of Eurocrypt'99, LNCS vol. 1592, Springer-Verlag, 1999, pp. 223-238.
17. PKCS #1 v2.1, *RSA Cryptography Standard (draft)*, available at `http://www.rsasecurity.com /rsalabs/pkcs`.
18. D. Pointcheval and J. Stern, *Security proofs for signature schemes*. Proceedings of Eurocrypt'96, LNCS vol. 1070, Springer-Verlag, pp. 387-398.
19. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, CACM 21, 1978.

A Proof of theorem 1

We construct an algorithm $\mathcal{R}$ which inverts RSA using a forger $\mathcal{F}$. The reduction $\mathcal{R}$ will answer by itself the hash queries and signature queries of $\mathcal{F}$. We assume that when the forger makes a signature query he has already made the corresponding hash query. If not, the reduction goes ahead and makes the corresponding hash query. Similarly, we assume that the message M which signature is forged by the forger, has already been queried for hashing. Otherwise the reduction makes the corresponding hash query and proceeds.

Algorithm for $\mathcal{R}$:
Input: (N, e, y) and (q_{hash}, q_{sig}), where $(N, e) \leftarrow RSA(1^k)$ and $y \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$.
Output: $y^d \bmod N$.

1. Set $i \leftarrow 0$
2. Send (N, e) to $\mathcal{F}$.
3. If $\mathcal{F}$ makes a hash query for M:
   $i \leftarrow i + 1$; $M_i \leftarrow M$; $r_i \stackrel{R}{\leftarrow} \mathbb{Z}_N^*$
   Flip a coin c_i with bias γ.
   $c_i = 0$ with probability γ and $c_i = 1$ with probability $1 - \gamma$.
   Return $H(M) = y^{c_i} \cdot r_i^e \bmod N$
4. If $\mathcal{F}$ makes a signature query for M_i:
   Return r_i if $H(M_i) = r_i^e \bmod N$. Otherwise stop.
5. If $\mathcal{F}$ outputs a forgery (M, x):
   If $H(M) = y \cdot r_i^e \bmod N$ then output $y^d = x/r_i \bmod N$. Otherwise stop.
6. Go to step 3

$\mathcal{R}$ answers a signature query at step 4 with probability γ; the probability that $\mathcal{R}$ answers all the signature queries is greater than $\gamma^{q_{sig}}$. Eventually $\mathcal{F}$ outputs a forgery with probability ε_F; $\mathcal{R}$ can use this forgery at step 5 with probability $1-\gamma$ to output $y^d \bmod N$. Consequently $\mathcal{R}$ outputs $y^d \bmod N$ with probability $\gamma^{q_{sig}} \cdot (1 - \gamma) \cdot \varepsilon_F$, which is maximal for $\gamma = 1 - 1/(q_{sig} + 1)$ and gives (2).

B Proof of inequality (7)

Let

$$g(\beta) = \beta^{q_{sig}} \cdot \sum_{j=0}^{q_{sig}} \Pr[a_i = j] \cdot \left(1 - 2^{-k_0}\right)^j \tag{21}$$

with

$$\Pr[a_i = j] = \begin{cases} (1 - \beta) \cdot \beta^j & \text{if } j < q_{sig} \\ \beta^{q_{sig}} & \text{if } j = q_{sig} \end{cases} \tag{22}$$

We denote $g_0 = \max\{g(\beta); \beta \in [0, 1]\}$ and want to prove that

$$g_0 \geq \frac{1}{1 + 6 \cdot q_{sig} \cdot 2^{-k_0}}$$

Denoting $\gamma = 2^{-k_0}$, we obtain from (21) and (22):

$$g(\beta) = \frac{\beta^{q_{sig}}}{1 - (1 - \gamma) \cdot \beta} \cdot \left(1 - \beta + \gamma \cdot (1 - \gamma)^{q_{sig}} \cdot \beta^{q_{sig}+1}\right) \tag{23}$$

from which we derive:

$$g(\beta) \geq \beta^{q_{sig}} \cdot \frac{1-\beta}{1-\beta+\gamma}$$

If $\gamma \cdot q_{sig} \geq 1/2$, we take $\beta = 1 - 1/(2 \cdot q_{sig})$ and obtain:

$$g_0 \geq \left(1 - \frac{1}{2 \cdot q_{sig}}\right)^{q_{sig}} \cdot \frac{1}{1 + 2 \cdot \gamma \cdot q_{sig}}$$

For $q_{sig} \geq 1$ we have

$$\left(1 - \frac{1}{2 \cdot q_{sig}}\right)^{q_{sig}} \geq \frac{1}{2}$$

Using $\gamma \cdot q_{sig} \geq 1/2$, we obtain

$$g_0 \geq \frac{1}{2 \cdot (1 + 2 \cdot \gamma \cdot q_{sig})} \geq \frac{1}{1 + 6 \cdot \gamma \cdot q_{sig}}$$

For $\gamma \cdot q_{sig} \leq 1/2$, we take $\beta = 1$ and obtain using (23):

$$g_0 \geq (1-\gamma)^{q_{sig}} \geq 1 - \gamma \cdot q_{sig} \geq \frac{1}{1 + 6 \cdot \gamma \cdot q_{sig}} \quad \text{for } \gamma \cdot q_{sig} \leq 1/2$$

C Proof of theorem 4

Let $\mathcal{F}$ be a forger which $(t, q_{sig}, q_{hash}, \varepsilon)$-breaks PSS. We construct an inverter I which (t', ε')-breaks RSA. The inverter receives as input (N, e, η) and must output $\eta^d \bmod N$. The inverter $\mathcal{I}$ maintains a counter i, initially 0.

The proof is very similar to the proof of theorem 2 and to the original security proof of PSS in [2]. To answer a hash query $M||r$ in theorem 2, we generated a random $x \in \mathbb{Z}_N$ and $y = x^e \cdot \eta^b$ with $b = 0$ or $b = 1$, and defined $H(M||r) = y$. The only difference here is that we write y as $0||\omega||r^*||\gamma$, where the size of ω is k_1 bits, the size of r^* is k_0 bits and the size of γ is the remaining $k - k_0 - k_1 - 1$ bits. We define $H(M||r) = \omega$ and $G(\omega) = r^* \oplus r||\gamma$. Moreover we must make sure that the same ω never appears twice otherwise we would be re-defining $G(\omega)$.

When a message M appears for the first time in a hash query or a signature query, the inverter increments the counter i and sets $M_i \leftarrow M$. Then, the inverter generates a list L_i of q_{sig} random integers in $\{0,1\}^{k_0}$.

When the forger makes a H-oracle query for $M_i\|r$, we distinguish two cases. If r belongs to the list L_i, the inverter sets $b = 0$, else it sets $b = 1$. Then the inverter generates a random $x \in \mathbb{Z}_N^*$ until the first bit of $y = x^e \cdot \eta^b \bmod N$ is 0. Then it writes y as $0\|\omega\|r^*\|\gamma$ and sets $H(M_i\|r) = \omega$. The inverter aborts if ω has already appeared before. Eventually the inverter sets $G(\omega) = r^* \oplus r\|\gamma$ and returns ω as the answer to the H-oracle query $M_i\|r$.

When the forger makes a G-oracle query for ω, the inverter returns $G(\omega)$ if ω appeared before. Otherwise it generates a random string $\alpha \leftarrow \{0,1\}^{k-k_1-1}$, sets $G(\omega) = \alpha$, and returns α.

When the forger makes a signature query for M_i, the inverter picks up a random r in L_i and discards it from the list. If there was already a H-oracle query for $M_i\|r$, the inverter knows x, y, ω, r^* and γ such that $y = x^e \bmod N$ and $y = 0\|\omega\|r^*\|\gamma$ where $H(M_i||r) = \omega$ and $G(\omega) = r^* \oplus r\|\gamma$, so the inverter returns x as a signature for M_i. Otherwise, the inverter generates a random $x \in \mathbb{Z}_N^*$ until the first bit of $y = x^e \bmod N$ is 0. Then it writes y as

$0\|\omega\|r^*\|\gamma$ and sets $H(M_i\|r) = \omega$. The inverter aborts if ω has already appeared before. Then the inverter sets $G(\omega) = r^* \oplus r\|\gamma$, and returns x as a signature for M_i.

Since there are at most q_{hash} hash queries and q_{sig} signature queries, the number of distinct ω which can appear is less than $q_{hash} + q_{sig}$. The probability that the inverter aborts after generating a random ω is then less than $(q_{hash} + q_{sig}) \cdot 2^{-k_1}$. Therefore, the inverter aborts when answering the hash and signature queries with probability less than $\delta = (q_{hash} + q_{sig})^2 \cdot 2^{-k_1}$. Consequently, the forger outputs a forgery with probability at least $\varepsilon - \delta$.

When the forger outputs a forgery (M, s), we compute $y = s^e \bmod N$ and write y as $0\|\omega\|r^*\|\gamma$. Let $r = r^* \oplus G_1(\omega)$, where G_1 denotes the first k_0 bits of G. If there was no H-oracle query for $M\|r$ before, the probability that $\omega = H(M\|r)$ is at most 2^{-k_1}. Therefore, with probability at least $\varepsilon - \delta - 2^{-k_1}$, the forger outputs a forgery and there exists an integer i such that there has been a H-oracle query for $M_i\|r$. Then if r does not belong to the list L_i, the inverter knows x such that $y = x^e \cdot \eta$, which gives $\eta^d = s/x \bmod N$ and the inverter succeeds in outputting $\eta^d \bmod N$.

As in theorem 2, the probability that r does not belong to the list L_i of q_{sig} random integers is $(1 - 2^{-k_0})^{q_{sig}}$. If $k_0 \geq \log_2 q_{sig}$ and for $q_{sig} \geq 2$, this gives

$$\left(1 - 2^{-k_0}\right)^{q_{sig}} \geq \left(1 - \frac{1}{q_{sig}}\right)^{q_{sig}} \geq \frac{1}{4}$$

Consequently, the success probability ε' of the inverter is at least $(\varepsilon - \delta - 2^{-k_1})/4$, which shows that for $k_0 \geq \log_2 q_{sig}$ the probability of breaking $\mathtt{PSS}[k_0, k_1]$ is almost the same as the probability of inverting RSA.

For smaller values of k_0, we apply the same trick as in theorem 2: we generate fewer than q_{sig} random integer in the lists L_i, according to the same distribution with parameter β. As in theorem 2, the success probability of the inverter is at least:

$$\left(\varepsilon - \delta - 2^{-k_1}\right) \cdot \beta^{q_{sig}} \cdot f(\beta)$$

where $f(\beta)$ is given by equation (6). As in theorem 2, we select a value of β which maximizes this success probability; we obtain that the inverter succeeds with probability at least:

$$\frac{\varepsilon - \delta - 2^{-k_1}}{1 + 6 \cdot q_{sig} \cdot 2^{-k_0}}$$

Moreover, when answering the hash and signature queries, the probability that the first bit of $x^e \cdot \eta^b \bmod N$ is 0 for a random $x \in \mathbb{Z}_N$ is at least $1/2$. Therefore we stop the loop after $1 + k_1$ steps[3], which adds a failure probability of 2^{-k_1} per hash or signature query. Eventually, the success probability ε' of the inverter is at least:

$$\varepsilon' = \frac{\varepsilon - 2 \cdot (q_{hash} + q_{sig})^2 \cdot 2^{-k_1}}{1 + 6 \cdot q_{sig} \cdot 2^{-k_0}}$$

which gives equation (10). The running time of the inverter is the running time of the forger plus the time to generate the $x^e \cdot \eta^b \bmod N$, which gives (9).

[3] otherwise the running time could not be bounded.

D Proof of lemma 1

We show inductively over n that, letting D_n be the following distribution

$$D_n = \begin{cases} i \leftarrow [1,n] \\ (\alpha_1, \ldots, \alpha_n) \leftarrow [1,k]^n \\ \beta \leftarrow [1,k] \end{cases}$$

and denoting for any $j \in [1,n]$ the events:

$$A_j : (\alpha_1, \ldots, \alpha_{j-1}, \alpha_j) \in Q$$
$$B_j : (\alpha_1, \ldots, \alpha_{j-1}, \beta) \in Q$$

with $A_j \Rightarrow A_{j-1}$ for all $j \in [2,n]$, then the following holds:

$$\Pr_{D_n}[A_n \wedge B_i] \geq \Pr_{D_n}[A_n]^{1+\frac{1}{n}} \tag{24}$$

Inequality (24) clearly holds for $n = 1$. Assuming that inequality (24) holds for $n-1$, we show that it holds for n. In the following, unless specified otherwise, probabilities are taken according to the distribution D_n. Since i is randomly selected in $[1,n]$, we have:

$$\Pr[A_n \wedge B_i] = \frac{1}{n}\Pr[A_n \wedge B_1] + \frac{n-1}{n}\Pr[A_n \wedge B_i | i \geq 2] \tag{25}$$

The events A_n and B_1 are independent, which gives:

$$\Pr[A_n \wedge B_1] = \Pr[A_n] \cdot \Pr[B_1] = \Pr[A_n] \cdot \Pr[A_1] \tag{26}$$

We have:

$$\Pr[A_n] = \frac{1}{k} \sum_{a_1 \in [1,k]} \Pr[A_n | \alpha_1 = a_1]$$

and

$$\Pr[A_n \wedge B_i | i \geq 2] = \frac{1}{k} \sum_{a_1 \in [1,k]} \Pr\left[A_n \wedge B_i | (\alpha_1 = a_1) \wedge (i \geq 2)\right]$$

Letting $L_1 = \{a_1 \in [1,k] \mid (a_1) \in Q\}$, we have using $\Pr[A_1] = \#L_1/k$ and $A_n \Rightarrow A_1$:

$$\Pr[A_n | A_1] = \frac{\Pr[A_n \wedge A_1]}{\Pr[A_1]} = \frac{\Pr[A_n]}{\Pr[A_1]} = \frac{1}{\#L_1} \sum_{a_1 \in L_1} \Pr[A_n | \alpha_1 = a_1] \tag{27}$$

and

$$\Pr[A_n \wedge B_i | i \geq 2] = \Pr[A_1] \cdot \frac{1}{\#L_1} \sum_{a_1 \in L_1} \Pr\left[A_n \wedge B_i | (\alpha_1 = a_1) \wedge (i \geq 2)\right] \tag{28}$$

For all $j \in [2,n]$, let $A'_{j-1} = A_j \wedge (\alpha_1 = a_1)$ and $B'_{j-1} = B_j \wedge (\alpha_1 = a_1)$, and let D'_{n-1} be the following distribution:

$$D'_{n-1} = \begin{cases} i' \leftarrow [1,n-1] \\ (\alpha_2, \ldots, \alpha_n) \leftarrow [1,k]^{n-1} \\ \beta \leftarrow [1,k] \end{cases}$$

We have:

$$\Pr_{D'_{n-1}}[A'_{n-1}] = \Pr[A_n | \alpha_1 = a_1] \tag{29}$$

and

$$\Pr\left[A_n \wedge B_i | (\alpha_1 = a_1) \wedge (i \geq 2)\right] = \Pr_{D'_{n-1}}\left[A'_{n-1} \wedge B'_{i'}\right] \tag{30}$$

Applying inequality (24) for $n-1$, we obtain:

$$\Pr_{D'_{n-1}}\left[A'_{n-1} \wedge B'_{i'}\right] \geq \Pr_{D'_{n-1}}\left[A'_{n-1}\right]^{\frac{n}{n-1}}$$

which gives using equations (28), (29) and (30):

$$\Pr\left[A_n \wedge B_i | i \geq 2\right] \geq \Pr[A_1] \cdot \frac{1}{\#L_1} \sum_{a_1 \in L_1} \Pr[A_n | \alpha_1 = a_1]^{\frac{n}{n-1}} \tag{31}$$

From the inequality

$$\frac{1}{t}\sum_{i=1}^{t} x_i^r \geq \left(\frac{1}{t}\sum_{i=1}^{t} x_i\right)^r \quad \text{for} \quad r \geq 1$$

we obtain:

$$\Pr[A_n \wedge B_i | i \geq 2] \geq \Pr[A_1] \cdot \left(\frac{1}{\#L_1} \sum_{a_1 \in L_1} \Pr[A_n | \alpha_1 = a_1]\right)^{\frac{n}{n-1}}$$

which gives using (27):

$$\Pr[A_n \wedge B_i | i \geq 2] \geq \Pr[A_1] \cdot \Pr[A_n | A_1]^{\frac{n}{n-1}} = \Pr[A_n] \cdot \Pr[A_n | A_1]^{\frac{1}{n-1}}$$

Then using equations (25) and (26), we obtain:

$$\Pr[A_n \wedge B_i] \geq \Pr[A_n]\left(\frac{\Pr[A_1]}{n} + \frac{n-1}{n}\Pr[A_n | A_1]^{\frac{1}{n-1}}\right)$$

Using the well known inequality $S \geq P$ between the arithmetic mean S and the geometric mean P, we obtain:

$$\frac{1}{n}\left(\Pr[A_1] + (n-1) \cdot \Pr[A_n | A_1]^{\frac{1}{n-1}}\right) \geq \left(\Pr[A_1] \cdot \Pr[A_n | A_1]\right)^{\frac{1}{n}} = \Pr[A_n]^{\frac{1}{n}}$$

and eventually

$$\Pr[A_n \wedge B_i] \geq \Pr[A_n]^{1+\frac{1}{n}}$$

which shows that equation (24) holds for n and terminates the proof by induction.

Inequality (24) gives:

$$\Pr[A_n \wedge \neg B_i] = \Pr[A_n] - \Pr[A_n \wedge B_i] \leq \Pr[A_n] \cdot \left(1 - \Pr[A_n]^{1/n}\right)$$

Denoting $x = \Pr[A_n]^{1/n}$ and using the inequality $x^n \cdot (1-x) \leq \exp(-1)/n$ for $x \in [0,1]$, we obtain:

$$\Pr[A_n \wedge \neg B_i] \leq \frac{\exp(-1)}{n}$$

E Definitions for security proofs in the random oracle

In this section, we consider a signature scheme provably secure in the random oracle model. In the random oracle model, the hash function is replaced by a random function.

Definition 8 (random oracle). *For any constant k, a random oracle is a function H selected uniformly at random in the set $\mathcal{H}_k$ of the functions from $\{0,1\}^*$ to $\{0,1\}^k$.*

We say that a signature scheme is a hash-and-sign signature scheme if the signature generation algorithm first hashes the message and then signs the hash value using the private key.

Definition 9 (hash-and-sign scheme). *A signature scheme* (`Gen`, `Sign`, `Verify`) *is said to be a hash-and-sign signature scheme if* `Sign` *takes as input the message M, the public key pk and the private key sk, runs* `Hashing` *with M and pk, obtains h, then runs* `Signing` *with h and sk, obtains and returns the signature x, where:*
- `Hashing` *is an algorithm taking as input the message M to be signed and the public key pk and returning a string h.* `Hashing` *may have access to a random oracle.*
- `Signing` *is an algorithm taking as input h and the private key sk and returning the signature x.* `Signing` *does not have access to a random oracle.*

Examples of hash-and-sign signature schemes include the FDH scheme, PSS, Gennaro-Halevi-Rabin's scheme (GHR) in the random oracle model [11], Paillier's signature scheme [16] and DSA [10].

The `hashing` algorithm may require multiple hash oracle queries, for example two hash queries as in PSS. For simplicity, we say in the following that a forger can make q_{hash} hash queries if he can apply `Hashing` to q_{hash} messages. The actual number of hash queries q'_{hash} will then be a multiple of q_{hash} (for PSS, we have $q'_{hash} = 2 \cdot q_{hash}$).

We say that a signature scheme is with unique signature if each message has a unique signature, given the random oracle $H \in \mathcal{H}_k$; formally:

Definition 10 (signature scheme with unique signature). *A signature scheme is said to be with unique signature if for any public key pk, any message M and any random oracle H in $\mathcal{H}_k$, there is a unique x such that* $\mathtt{Verify}_{pk}(M,x) = 1$.

Hash-and-sign signature schemes with unique signature include FDH, GHR in the random oracle model and Paillier's signature scheme. PSS and DSA are *not* signature schemes with unique signature.

Lemma 2. *Let $\mathcal{S}$ be a hash-and-sign signature scheme with unique signature. Let $h \leftarrow \mathtt{Hashing}_{pk}(M)$. The signature x of M is then a function of h and the public key pk only.*

Proof. We denote $\mathtt{Sign}^H_{pk,sk}$, $\mathtt{Hashing}^H_{pk}$ and $\mathtt{Verify}^H_{pk}$ the algorithms `Sign`, `Hashing` and `Verify` with oracle access to $H \in \mathcal{H}_k$.

Let x be the signature of M with random oracle $H \in \mathcal{H}_k$ and public key pk. Let (pk', sk') be another public/private key pair, M' another message, and $H' \in \mathcal{H}_k$ another random oracle. Let $h' \leftarrow \mathtt{Hashing}^{H'}_{pk'}(M')$ and $x' \leftarrow \mathtt{Signing}_{pk',sk'}(h')$. We must show that if $pk = pk'$ and $h = h'$, then $x = x'$.

From $pk = pk'$ and $h = h'$, we deduce $h' \leftarrow \mathtt{Hashing}^H_{pk}(M)$ and $x' \leftarrow \mathtt{Signing}_{pk,sk'}(h')$, which implies that x' is a signature of M under the public key pk with random oracle H. Since $\mathcal{S}$ is a signature scheme with unique signature, we must have $x = x'$.

□

The security of the signature scheme that we consider is not necessarily based on the hardness of inverting RSA; it can be based on the hardness of any search problem Π, defined as follows:

Definition 11. *A search problem Π is a triple $(\mathtt{Gen}\Pi, D, S)$ where D is a set of finite objects called instances, and for each instance $I \in D$, $S[I]$ is a set of finite objects called solutions for I. $\mathtt{Gen}\Pi$ is an algorithm which, on input 1^k, randomly selects an instance $I \in D$ such that $|I| = k$.*

Definition 12. *An algorithm $\mathcal{A}$ is said to (t_A, ε_A)-solve Π if after receiving an instance I generated using $\mathtt{Gen}\Pi(1^k)$ and $t_A(k)$ processing time it outputs a solution z for I with probability greater than $\varepsilon_A(k)$ for all $k \in \mathbb{N}$.*

Definition 13. *A problem Π is said to be (t_A, ε_A)-hard if there is no algorithm $\mathcal{A}$ which (t_A, ε_A)-solves Π.*

In the following we consider a hash-and-sign signature scheme with unique signature provably secure in the random oracle model, assuming that solving a given problem Π is hard. This means that there exists a reduction from solving the hard problem Π to breaking the signature scheme $\mathcal{S}$. A reduction from solving Π to breaking $\mathcal{S}$ is an algorithm which uses a forger for $\mathcal{S}$ in order to solve Π. Resistance against adaptive chosen message attacks is considered, so the forger is allowed to make signature queries for messages of his choice. Moreover, in the random oracle model, the forger cannot compute the hash function by itself: the forger must make a hash query. Consequently, when interacting with the forger, the reduction algorithm must answer the hash queries and the signature queries made by the forger.

Definition 14. *A reduction R in the random oracle model from solving $(Gen\Pi, D, S)$ to breaking* (`Gen`, `Sign`, `Verify`) *is a probabilistic algorithm taking as input an instance I and (q_{hash}, q_{sig}), where $I \leftarrow Gen\Pi(1^k)$, and outputting a solution z for I. The reduction algorithm interacts with a forger $\mathcal{F}$ for* (`Gen`, `Sign`, `Verify`) *which outputs a forgery after at most q_{hash} hash queries and q_{sig} signature queries. The reduction algorithm answers the hash queries and the signature queries made by $\mathcal{F}$.*

We quantify the efficiency of the reduction by giving the probability that the reduction algorithm outputs a solution of the problem Π using a forger that $(t_F, q_{hash}, q_{sig}, \varepsilon_F)$-breaks the signature scheme, within an additional running time of t_R.

Definition 15. *We say that a reduction algorithm $\mathcal{R}$ $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R)$-reduces solving $(\mathtt{Gen}\Pi, D, S)$ to breaking the signature scheme* (`Gen`, `Sign`, `Verify`) *if after running any forger that $(t_F, q_{hash}, q_{sig}, \varepsilon_F)$-breaks* (`Gen`, `Sign`, `Verify`), *the reduction outputs a solution of Π with probability greater than ε_R, within an additional running time of t_R.*

In this section we consider reductions running a forger only once, as the reduction of theorem 1 for FDH. Reductions running a forger more than once will be considered in the next section. The following theorem shows that for any hash-and-sign signature scheme with unique signature provably secure in the random oracle model, assuming the hardness of a given problem Π, the security reduction cannot be tighter than roughly ε_F / q_{sig}. Namely we show that from $\mathcal{R}$ we can solve the problem Π with probability greater than roughly $\varepsilon_R - \varepsilon_F / q_{sig}$, in roughly the same time bound. Thus, if solving Π is hard, the success probability ε_R of $\mathcal{R}$ cannot be greater than roughly ε_F / q_{sig}.

Theorem 9. *Let $\mathcal{R}$ be a reduction which $(t_R, q_{hash}, q_{sig}, \varepsilon_R, \varepsilon_F)$-reduces solving Π to breaking a hash-and-sign signature scheme with unique signature. $\mathcal{R}$ runs the forger only once. From $\mathcal{R}$ we can construct an algorithm which (t_A, ε_A)-solves Π, with:*

$$t_A = 2 \cdot t_R \tag{32}$$

$$\varepsilon_A = \varepsilon_R - \varepsilon_F \cdot \frac{\exp(-1)}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{q_{hash}}\right)^{-1} \tag{33}$$

Proof. From a reduction $\mathcal{R}$ that $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R)$-reduces solving Π to breaking the signature scheme $(\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$, we build an algorithm $\mathcal{A}$ that (t_A, ε_A)-solves Π using $\mathcal{R}$. The algorithm $\mathcal{A}$ receives as input an instance I of the problem Π and must output a solution z of I using $\mathcal{R}$. As in the proof of theorem 5, the algorithm $\mathcal{A}$ will simulate a forger with respect to $\mathcal{R}$. $\mathcal{A}$ arbitrarily selects q_{hash} distinct messages $M_1, \ldots, M_{q_{hash}}$ of length $\mathcal{O}(k)$.

First $\mathcal{A}$ receives from $\mathcal{R}$ the public key pk. Then $\mathcal{A}$ runs `Hashing` for the q_{hash} messages $M_1, \ldots, M_{q_{hash}}$, performs the corresponding hash queries to $\mathcal{R}$, and obtain the corresponding strings $h_1, \ldots, h_{q_{hash}}$. $\mathcal{A}$ selects a random integer $\beta \in [1, q_{hash}]$ and a random sequence α of q_{sig} integers in $[1, q_{hash}] \backslash \{\beta\}$, which we denote $\alpha = (\alpha_1, \ldots, \alpha_{q_{sig}})$. $\mathcal{A}$ selects a random integer $i \in [1, q_{sig}]$ and define the sequence of i integers $\alpha' = (\alpha_1, \ldots, \alpha_{i-1}, \beta)$. Then $\mathcal{A}$ makes the i signature queries corresponding to α' to $\mathcal{R}$ and receive from $\mathcal{R}$ the corresponding signatures, the last one being the signature s_β of M_β.

Then the reduction $\mathcal{R}$ is rewound to the state in which it was after the hash queries, and this time, $\mathcal{A}$ makes the q_{sig} signature queries corresponding to α. If $\mathcal{R}$ has answered all the signature queries, then with probability ε_F, $\mathcal{A}$ sends (M_β, s_β) as a forgery to $\mathcal{R}$. From lemma 2 the signature s_β of M_β is a function of h_β and pk only, so s_β is still a valid signature of M_β after $\mathcal{R}$ has been rewound. This is a true forgery for $\mathcal{R}$ because after the rewind of $\mathcal{R}$, there was no signature query for M_β. Eventually $\mathcal{R}$ outputs a solution z of instance I.

We denote by $\mathcal{Q}$ the set of sequences of signature queries which are correctly answered by $\mathcal{R}$ after the hash queries, in time less than t_R. If a sequence of signature queries is correctly answered by $\mathcal{R}$, then the same sequence without the last signature query is also correctly answered, so for any $(\alpha_1, \ldots, \alpha_j) \in \mathcal{Q}$, we have $(\alpha_1, \ldots, \alpha_{j-1}) \in \mathcal{Q}$. Let denote by `ans` the event $\alpha \in \mathcal{Q}$, which corresponds to $\mathcal{R}$ answering all the signature queries after the rewind, and by `ans'` the event $\alpha' \in \mathcal{Q}$, which corresponds to $\mathcal{R}$ answering all the signature queries before the rewind.

Let consider a forger which makes the same hash queries, the same signature queries corresponding to α, and outputs a forgery for M_β with probability ε_F. By definition, when interacting with such a forger, $\mathcal{R}$ would output $y^d \bmod N$ with probability at least ε_R. After the rewind, $\mathcal{R}$ sees exactly the same transcript as when interacting with this forger, except if event `ans` is true and `ans'` is false: in this case, the forger outputs a forgery with probability ε_F, whereas our simulation does not output a forgery. Consequently, when interacting with our simulation of a forger, $\mathcal{R}$ outputs $y^d \bmod N$ with probability at least:

$$\varepsilon_R - \varepsilon_F \cdot \Pr[\texttt{ans} \wedge \neg\texttt{ans'}] \tag{34}$$

Using lemma 1 with $n = q_{sig}$ and $k = q_{hash}$, we obtain:

$$\Pr[\texttt{ans} \wedge \neg\texttt{ans'}] \leq \frac{\exp(-1)}{q_{sig}} \left(1 - \frac{q_{sig}}{q_{hash}}\right)^{-1} \tag{35}$$

The term $(1 - q_{sig}/q_{hash})$ in equation (35) is due to the fact that we select $\alpha_1, \ldots, \alpha_{q_{sig}}$ in $[1, q_{hash}] \setminus \{\beta\}$ whereas in lemma 1 the integers are selected in $[1, q_{hash}]$. From equations (34)

and (35) we obtain that $\mathcal{I}$ succeeds with probability greater than ε_I given by (33). Because of the rewind, the running time of $\mathcal{I}$ is at most twice the running time of $\mathcal{R}$, which gives (32).

F Proof of theorem 6

In this section, we consider reductions running a forger more than once, as opposed to section E in which the forger was run only once. The reduction can run or rewind the forger at most r times. Using the same technique as previously, we show that from a reduction $\mathcal{R}$ allowed to run or rewind the forger at most r times, we can solve the problem Π with probability greater than roughly $\varepsilon_R - \varepsilon_F \cdot r/q_{sig}$ in roughly the same time bound. Thus, if solving Π is hard, the success probability of $\mathcal{R}$ cannot be greater than roughly $\varepsilon_F \cdot r/q_{sig}$.

The proof is very similar to the proof of theorem 9. Assume first that the reduction is not allowed to rewind the forger. The reduction is only allowed to run the forger at most r times. We say that the reduction is in the j-th round if the reduction has already run the forger $j-1$ times. Thus there are at most r rounds.

In the first round of the reduction, we apply exactly the same technique as previously: we make the q_{hash} hash queries, then we select a random $\beta_1 \in [1, q_{hash}]$ and a random sequence α_1 of q_{sig} integers in $[1, q_{hash}] \setminus \{\beta_1\}$. We select a random integer $i_1 \in [1, q_{sig}]$ and define the sequence α'_1 as the first $i_1 - 1$ integers of α_1 plus the integer β_1. Then we make the i_1 signature queries corresponding to α'_1 to $\mathcal{R}$ and obtain the signature s_{β_1} of M_{β_1}. Then we rewind $\mathcal{R}$ to the state it was after the hash queries, and this time, we make the q_{sig} signature queries corresponding to α_1. If $\mathcal{R}$ has answered all the signature queries, then with probability ε_F, we send $(M_{\beta_1}, s_{\beta_1})$ as a forgery to $\mathcal{R}$.

Then the reduction is in the second round, and starts interacting with a forger for the second time. We proceed recursively for the remaining rounds: at the j-th round, we make the same q_{hash} hash queries and select β_j, α_j and i_j as previously. We obtain the signature of M_{β_j}, then we rewind $\mathcal{R}$ to the state it was after the hash queries, then make the signature queries corresponding to α_j, and with probability ε_R output the signature of M_{β_j} as a forgery. Using this technique, we are able to simulate a forger being run at most r times by the reduction.

Let us denote $\mathtt{ans}'_j$ the event in which the reduction in the j-th round answers the signature queries before it is rewound and $\mathtt{ans}_j$ the event in which the reduction answers the signature queries after it is has been rewound.

Let consider a forger which at each round makes the same hash queries and signature queries corresponding to α_j, and outputs a forgery for M_{β_j} with probability ε_F. By definition, when running at most r times this forger, $\mathcal{R}$ succeeds with probability at least ε_R.

After the rewind of the j-th round, $\mathcal{R}$ sees exactly the same transcript as when interacting with this forger, except if event $\mathtt{ans}_j$ is true and $\mathtt{ans}'_j$ is false: in this case, this forger outputs a forgery with probability ε_F, whereas our simulation does not output a forgery. This happens with probability:

$$\varepsilon_F \cdot \Pr[\mathtt{ans}_j \wedge \neg \mathtt{ans}'_j]$$

Since there are at most r rounds, $\mathcal{A}$ succeeds with probability greater than:

$$\varepsilon_R - \sum_{j=1}^{r} \varepsilon_F \cdot \Pr[\mathtt{ans}_j \wedge \neg \mathtt{ans}'_j]$$

Using lemma 1, we obtain for all j:

$$\Pr[\mathtt{ans}_j \wedge \neg\mathtt{ans}'_j] \leq \frac{\exp(-1)}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{q_{hash}}\right)^{-1}$$

Consequently, $\mathcal{A}$ succeeds with probability greater than:

$$\varepsilon_A = \varepsilon_R - \varepsilon_F \cdot \frac{\exp(-1) \cdot r}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{q_{hash}}\right)^{-1} \tag{36}$$

The reduction $\mathcal{R}$ is rewound at most r times, so the running time of $\mathcal{A}$ is at most $r+1$ times the running time of $\mathcal{R}$, which gives:

$$t_A = (r+1) \cdot t_R \tag{37}$$

Now assume that the reduction $\mathcal{R}$ is allowed to rewind the forger to a previous state $\mathcal{S}$. Equivalently we assume that the reduction actually restarts the forger with the same random tape and provides the same input to the forger until the same state $\mathcal{S}$ is reached. If $\mathcal{R}$ restarts the forger at the $j-1$-th round, the reduction is now in the j-th round. We distinguish two cases: if the reduction sends the same public key and provides the same answer to the hash queries, the forger will make the same signature queries and forgery as in the $j-1$-th round. Therefore our simulation will make the same signature queries and forgery as in the $j-1$-th round. At the j-th round the reduction sees exactly the same transcript when interacting with the forger as when interacting with our simulation, except with probability:

$$\varepsilon_F \cdot \Pr[\mathtt{ans}_j \wedge \neg\mathtt{ans}'_j]$$

Otherwise if the reduction sends a different public key or provides a different answer to the hash queries, the forger makes signature queries for random messages and forge the signature of a randomly chosen message, and so our simulation makes signature queries for random messages and forge the signature of a randomly chosen message. Consequently, at the j-th round the reduction sees exactly the same transcript when interacting with the forger as when interacting with our simulation, except with probability:

$$\varepsilon_F \cdot \Pr[\mathtt{ans}_j \wedge \neg\mathtt{ans}'_j]$$

Consequently, we obtain the same result as previously: $\mathcal{A}$ succeeds with probability at least ε_A given by equation (36).

G Security definitions in the standard model

Definition 16. *A forger $\mathcal{F}$ is said to $(t, q_{sig}, \varepsilon)$-break the signature scheme* (`Gen`, `Sign`, `Verify`) *if after at most $q_{sig}(k)$ signature queries and $t(k)$ processing time, it outputs a forgery with probability at least $\varepsilon(k)$ for all $k \in \mathbb{N}$.*

Definition 17. *A signature scheme* (`Gen`, `Sign`, `Verify`) *is $(t, q_{sig}, \varepsilon)$-secure if there is no forger who $(t, q_{sig}, \varepsilon)$-breaks the scheme.*

Definition 18 (signature scheme with unique signature). *A signature scheme is said to be with unique signature if for any public key pk and any message M, there is a unique signature x such that $\mathtt{Verify}_{pk}(M, x) = 1$.*

Note that a signature scheme with unique signature is necessarily state-free: the signature of a message does not depend on previously signed messages.

We assume that the security of $(\texttt{Gen}, \texttt{Sign}, \texttt{Verify})$ is based on the hardness of the problem Π, so there exists a reduction from solving Π to breaking the signature scheme in the standard model.

Definition 19. *A reduction algorithm R in the standard model from solving $(Gen\Pi, D, S)$ to breaking* (Gen, Sign, Verify) *is a probabilistic algorithm taking as input an instance I and q_{sig}, where $I \leftarrow Gen\Pi(1^k)$, and outputting a solution z for I. The reduction algorithm interacts with a forger $\mathcal{F}$ for* (Gen, Sign, Verify) *which outputs a forgery after at most q_{sig} signature queries. The reduction algorithm answers the signature queries made by $\mathcal{F}$.*

Definition 20. *A reduction algorithm $\mathcal{R}$ is said to $(t_R, q_{sig}, \varepsilon_F, \varepsilon_R)$-reduce solving Π to breaking the signature scheme* (Gen, Sign, Verify) *if after receiving an instance I such that $I \leftarrow Gen\Pi(1^k)$ and running any forger that $(t_F, q_{sig}, \varepsilon_F)$-breaks the signature scheme, the reduction $\mathcal{R}$ outputs a solution z for I with probability at least $\varepsilon_R(k)$ after at most $t_R(k)$ additional processing time for all $k \in \mathbb{N}$.*

H Proof of theorem 7

The proof is very similar to the proof of theorem 6. The only difference is that there are no hash queries. Moreover, we replace in algorithm $\mathcal{A}$ the number of hash queries q_{hash} by the lower bound 2^ℓ on the size of the message space; instead of selecting q_{hash} distinct messages, we select 2^ℓ distinct messages $M_1, \ldots, M_{2^\ell}$. The rest of algorithm $\mathcal{A}$ is the same, and the same analysis shows that from a reduction with running time of t_R, which succeeds with probability at least ε_R after running or rewinding at most r times a forger that breaks the signature scheme with probability at least ε_F, we can build an algorithm which (t_A, ε_A)-solves the problem Π, with:

$$t_A = (r+1) \cdot t_R$$
$$\varepsilon_A = \varepsilon_R - \varepsilon_F \cdot \frac{\exp(-1) \cdot r}{q_{sig}} \cdot \left(1 - \frac{q_{sig}}{2^\ell}\right)^{-1}$$

I A variant of GHR's scheme provably secure in the standard model

Let us consider Gennaro-Halevi-Rabin's signature scheme [11]. The public key is $N = p \cdot q$ and a random $y \in \mathbb{Z}_N^*$, where p and q are random $k/2$-bit primes and $(p-1)/2$ and $(q-1)/2$ are also primes. The private key is (p, q). The scheme uses a hash function H which outputs odd integers of length k_0 bits. To sign a message m, the signer obtains $e = H(m)$ and computes the signature σ as the e-th root of y modulo N, using p, q. To verify a signature σ, one computes $e = H(m)$ and checks that $\sigma^e = y \bmod N$.

The security of Gennaro-Halevi-Rabin's signature scheme is based on the hardness of the strong RSA problem.

Definition 21 (Strong RSA problem). *Given a randomly chosen RSA modulus N and a random element $s \in \mathbb{Z}_N^*$, find a pair (e, r) with $e > 1$ such that $r^e = s \bmod N$.*

In this section we illustrate theorem 7 with a variant of GHR's scheme provably secure in the standard model, with unique signature. The hash function H is replaced by an injective function Ψ which maps any string from $\{0,1\}^\ell$ to the set of prime integers, so that Ψ is

easy to compute. Such a function is constructed in [11]. We obtain a signature scheme with unique signature provably secure in the standard model. However the scheme is provably secure for short messages only (say, less than 40 bits); this is due to the 2^ℓ factor in the time bound t_F of the forger. We denote by $t(\ell)$ the time necessary to compute Ψ.

Theorem 10. *Assume that the strong RSA problem is (t_I, ε_I)-hard. Then the previous GHR variant is $(t_F, q_{sig}, \varepsilon_F)$-secure, where:*

$$t_I = t_F + \mathtt{poly}\left(2^\ell, k, t(\ell)\right) \tag{38}$$

$$\varepsilon_I = \frac{\varepsilon_F}{q_{sig}} \cdot \left(1 - \frac{1}{q_{sig}+1}\right)^{q_{sig}+1} \tag{39}$$

Proof. Assume that there exists a forger $\mathcal{F}$ that $(t_F,\ q_{sig},\ \varepsilon_F)$-breaks the signature scheme (`Gen`, `Sign`, `Verify`). We construct an algorithm $\mathcal{A}$ that solves the strong RSA problem using $\mathcal{F}$. $\mathcal{A}$ will answer the signature queries of the forger itself. The message space is $\{0,1\}^\ell$.

Algorithm for $\mathcal{A}$.
Input: (N, s) and (ℓ, q_{sig}), where $N \leftarrow RSA(1^k)$ and $s \xleftarrow{R} \mathbb{Z}_N^*$.
Output: (r, e) with $e > 1$ such that $r^e = s \bmod N$.

1. Let $E \leftarrow 1$.
2. For all messages $M_i \in \{0,1\}^\ell$, do the following:
   Flip a coin c_i with bias γ.
   $c_i = 0$ with probability γ and $c_i = 1$ with probability $1 - \gamma$.
   If $c_i = 0$ then compute $E \leftarrow E \cdot \Psi(M_i)$.
3. Let $y \leftarrow s^E \bmod N$.
4. Send the public key (N, y) to $\mathcal{F}$.
5. If $\mathcal{F}$ makes a signature query for M_i:
   If $c_i = 0$ then return $s^{E/\Psi(M_i)} \bmod N$. Otherwise stop.
6. If $\mathcal{F}$ outputs a forgery (M_i, x):
   If $c_i = 0$ then stop.
   Otherwise $\Psi(M_i) \wedge E = 1$ so let $a, b \in \mathbb{Z}$ such that $a \cdot \Psi(M_i) + b \cdot E = 1$.
   Let $r \leftarrow x^b \cdot s^a \bmod N$ and $e \leftarrow \Psi(M_i)$ and output (r, e).

The probability that $\mathcal{A}$ answers to all the signature queries is greater than $\gamma^{q_{sig}}$. Eventually $\mathcal{F}$ outputs a forgery with probability ε_F which $\mathcal{A}$ can use with probability $1 - \gamma$ to output (r, e). Consequently $\mathcal{A}$ outputs (r, e) with probability $\gamma^{q_{sig}} \cdot (1 - \gamma) \cdot \varepsilon_F$, which is maximal for $\gamma = 1 - 1/(q_{sig} + 1)$ and gives (39). □

J Proof of theorem 8

We use the following method: we consider PSS in which the random salt is fixed to 0^{k_0}, and we denote this signature scheme $\mathtt{PSS0}[k_0, k_1]$. Consequently, $\mathtt{PSS0}[k_0, k_1]$ is a signature scheme with unique signature. First, we show how to convert a forger for $\mathtt{PSS0}[k_0, k_1]$ into a forger for $\mathtt{PSS}[k_0, k_1]$. Then, any reduction $\mathcal{R}$ from inverting RSA to breaking $\mathtt{PSS}[k_0, k_1]$ will use this forger for $\mathtt{PSS}[k_0, k_1]$ in order to invert RSA. Consequently, from a forger for $\mathtt{PSS0}[k_0, k_1]$, we can invert RSA using the reduction $\mathcal{R}$. In other words, from $\mathcal{R}$ we can construct a reduction $\mathcal{R}_0$ from inverting RSA to breaking $\mathtt{PSS0}[k_0, k_1]$. Since $\mathtt{PSS0}[k_0, k_1]$ is a signature scheme with unique signature, theorem 6 gives an upper bound for the success probability of $\mathcal{R}_0$, from which we derive an upper bound for the success probability of $\mathcal{R}$.

This upper bound shows that the size k_0 of the random salt must be at least $\log_2 q_{sig}$ for $\mathtt{PSS}[k_0, k_1]$ to have a security level equivalent to RSA, and so our security proof of section 3.2 is optimal.

Lemma 3. *Let $\mathcal{F}_0$ be a forger which $(t_F^0, q_{hash}^0, q_{sig}^0, \varepsilon_F^0)$-breaks $\mathtt{PSS0}[k_0, k_1]$. From $\mathcal{F}_0$ we can construct a forger $\mathcal{F}$ which $(t_F, q_{hash}, q_{sig}, \varepsilon_F)$-breaks $\mathtt{PSS}[k_0, k_1]$, with:*

$$q_{hash} = q_{hash}^0 \qquad q_{sig} = 2^{k_0+1} \cdot q_{sig}^0 \qquad \varepsilon_F = \varepsilon_F^0/2$$

Proof. From $\mathcal{F}_0$ we construct a forger $\mathcal{F}$ for $\mathtt{PSS}[k_0, k_1]$. When the forger $\mathcal{F}_0$ makes a hash query, the forger $\mathcal{F}$ makes the same hash query and forwards the result to $\mathcal{F}_0$. When the forger $\mathcal{F}_0$ makes a signature query for a message M, the forger $\mathcal{F}$ makes signature queries for M until the random salt used to generate the signature is 0^{k_0}. Then it forwards the signature to $\mathcal{F}_0$. Eventually the forger $\mathcal{F}_0$ outputs a forgery for $\mathtt{PSS0}[k_0, k_1]$, which is also a forgery for $\mathtt{PSS}[k_0, k_1]$.

When $\mathcal{F}$ makes a signature query, the random salt used to generate the signature is equal to 0^{k_0} with probability 2^{-k_0}. Therefore $\mathcal{F}$ must perform on average 2^{k_0} signature queries for each signature query of $\mathcal{F}_0$. More precisely, let Y_i be the number of signature queries made by $\mathcal{F}$ for the i-th signature query of $\mathcal{F}_0$, and let Y be the total number of signature queries made by $\mathcal{F}$. Since $\mathcal{F}$ is limited to q_{sig} signature queries, the probability that all the signature queries of $\mathcal{F}_0$ are answered is $\Pr[Y \leq q_{sig}]$. In this case, the forger $\mathcal{F}_0$ outputs a forgery with probability at least ε_F^0. Therefore, the forger $\mathcal{F}$ outputs a forgery with probability at least $\varepsilon_F^0 \cdot \Pr[Y \leq q_{sig}]$.

The distribution of Y_i follows a geometric law of parameter $1 - 2^{-k_0}$:

$$\Pr[Y_i = j] = 2^{-k_0} \cdot (1 - 2^{-k_0})^{j-1} \quad \text{for } j \geq 1$$

The expectancy and variance of Y_i are given by:

$$\mathrm{E}[Y_i] = 2^{k_0} \qquad \mathrm{Var}[Y_i] = 2^{k_0} \cdot \left(2^{k_0} - 1\right)$$

We assume that $\mathcal{F}_0$ makes exactly q_{sig}^0 signature queries[4]. Since Y is the sum of q_{sig}^0 independent random variables, we obtain:

$$\mathrm{E}[Y] = 2^{k_0} \cdot q_{sig}^0 \qquad \mathrm{Var}[Y] = q_{sig}^0 \cdot 2^{k_0} \left(2^{k_0} - 1\right)$$

Then, using Chebyshev's inequality, we have for any δ:

$$\Pr\left[|Y - \mathrm{E}[Y]| \geq \delta\right] \leq \frac{\mathrm{Var}[Y]}{\delta^2}$$

and taking $\delta = E[Y]$, we obtain for $q_{sig}^0 \geq 2$:

$$\Pr\left[Y \geq 2 \cdot E[Y]\right] \leq \frac{1}{q_{sig}^0} \leq \frac{1}{2}$$

If $q_{sig}^0 = 1$, then $Y = Y_1$ and

$$\Pr[Y \geq 2 \cdot E[Y]] = \Pr[Y_1 \geq 2^{k_0+1}] = \left(1 - 2^{-k_0}\right)^{2^{k_0+1}-1}$$

[4] Otherwise we can simulate the remaining signature queries with previously queried messages. If $\mathcal{F}_0$ makes no signature queries, then $\mathcal{F}$ outputs a forgery with the same probability as $\mathcal{F}_0$.

Using the inequality $(1-1/x)^x \leq 1/2$ for $x \geq 1$, we obtain as previously:

$$\Pr[Y \geq 2 \cdot E[Y]] \leq \left(1 - 2^{-k_0}\right)^{2^{k_0}} \leq \frac{1}{2}$$

So letting

$$q_{sig} = 2^{k_0+1} \cdot q^0_{sig}$$

this gives $\Pr[Y \geq q_{sig}] \leq 1/2$ and thus $\Pr[Y \leq q_{sig}] \geq 1/2$. Consequently, the forger $\mathcal{F}$ outputs a forgery for $\mathtt{PSS}[k_0, k_1]$ with probability at least

$$\varepsilon_F = \varepsilon^0_F/2$$

after at most q_{sig} signature queries. □

Lemma 4. *Let $\mathcal{R}$ be a reduction which $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R)$-reduces inverting RSA to breaking $\mathtt{PSS}[k_0, k_1]$. From $\mathcal{R}$ we can construct a reduction $\mathcal{R}_0$ which $(t^0_R, q^0_{hash}, q^0_{sig}, \varepsilon^0_F, \varepsilon^0_R)$-reduces inverting RSA to breaking $\mathtt{PSS0}[k_0, k_1]$, with:*

$$\begin{aligned} q_{sig} &= q^0_{sig} \cdot 2^{k_0+1} & q_{hash} &= q^0_{hash} \\ \varepsilon_F &= \varepsilon^0_F/2 & \varepsilon_R &= \varepsilon^0_R \end{aligned} \tag{40}$$

$$t^0_R = t_R \tag{41}$$

Proof. Let $\mathcal{F}_0$ be a forger which $(t^0_F, q^0_{hash}, q^0_{sig}, \varepsilon^0_F)$-breaks $\mathtt{PSS0}[k_0, k_1]$. Using the previous lemma we construct from $\mathcal{F}_0$ a forger $\mathcal{F}$ which $(t_F, q_{hash}, q_{sig}, \varepsilon_F)$-breaks $\mathtt{PSS}[k_0, k_1]$, where q_{hash}, q_{sig} and ε_F are given by equation (40). Then from $\mathcal{F}$ using $\mathcal{R}$ we can invert RSA with probability at least ε_R.

Therefore, from $\mathcal{F}_0$ which $(t^0_F, q^0_{hash}, q^0_{sig}, \varepsilon^0_F)$-breaks $\mathtt{PSS0}[k_0, k_1]$, and using $\mathcal{R}$, we can invert RSA with probability at least ε_R. Consequently, from $\mathcal{R}$ we can construct a reduction $\mathcal{R}_0$ which $(t^0_R, q^0_{hash}, q^0_{sig}, \varepsilon^0_F, \varepsilon^0_R)$-reduces inverting RSA to breaking $\mathtt{PSS0}[k_0, k_1]$, where $\varepsilon^0_R = \varepsilon_R$ and $t^0_R = t_R$. □

Let $\mathcal{R}$ be a reduction which $(t_R, q_{hash}, q_{sig}, \varepsilon_F, \varepsilon_R)$-reduces inverting RSA to breaking $\mathtt{PSS}[k_0, k_1]$. From lemma 4, we construct from $\mathcal{R}$ an algorithm $\mathcal{R}_0$ which $(t^0_R, q^0_{hash}, q^0_{sig}, \varepsilon^0_F, \varepsilon^0_R)$-reduces inverting RSA to breaking $\mathtt{PSS0}[k_0, k_1]$, where t^0_R, q^0_{hash}, q^0_{sig}, ε^0_F and ε^0_R are given by equations (40) and (41). The reduction $\mathcal{R}$ can run or rewind the forger at most r times, so $\mathcal{R}_0$ runs or rewinds the forger at most r times. Then from $\mathcal{R}_0$ using theorem 6 we construct an algorithm $\mathcal{I}$ which (t_I, ε_I)-inverts RSA, with:

$$\begin{aligned} t_I &= (r+1) \cdot t^0_R \\ \varepsilon_I &= \varepsilon^0_R - r \cdot \varepsilon^0_F \cdot \frac{\exp(-1)}{q^0_{sig}} \cdot \left(1 - \frac{q^0_{sig}}{q^0_{hash}}\right)^{-1} \end{aligned}$$

Using equations (40) and (41) with $q_{hash} \geq 2 \cdot q_{sig}$ and $\exp(-1) \leq 1/2$, we obtain:

$$r \cdot \varepsilon^0_F \cdot \frac{\exp(-1)}{q^0_{sig}} \cdot \left(1 - \frac{q^0_{sig}}{q^0_{hash}}\right)^{-1} \leq r \cdot \varepsilon_F \cdot \frac{2^{k_0+2}}{q_{sig}}$$

which shows that the inverter succeeds with probability at least:

$$\varepsilon_R - r \cdot \varepsilon_F \cdot \frac{2^{k_0+2}}{q_{sig}}$$

and gives equations (19) and (20).