Appears in *Advances in Cryptology – Eurocrypt* 96 *Proceedings*, Lecture Notes in Computer Science Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.

The **Exact** Security of Digital Signatures— How to Sign with **RSA** and **Rabin**

Mihir Bellare[*] Phillip Rogaway[†]

March 14, 1996

Abstract

We describe an RSA-based signing scheme which combines essentially optimal efficiency with attractive security properties. Signing takes one RSA decryption plus some hashing, verification takes one RSA encryption plus some hashing, and the size of the signature is the size of the modulus. Assuming the underlying hash functions are ideal, our schemes are not only provably secure, but are so in a *tight* way— an ability to forge signatures with a certain amount of computational resources implies the ability to invert RSA (on the same size modulus) with about the same computational effort. Furthermore, we provide a second scheme which maintains all of the above features and in addition provides message recovery. These ideas extend to provide schemes for Rabin signatures with analogous properties; in particular their security can be tightly related to the hardness of factoring.

[*] Department of Computer Science and Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA. E-mail: mihir@cs.ucsd.edu ; Web page: http://www-cse.ucsd.edu/users/mihir

[†] Department of Computer Science, University of California at Davis, Davis, CA 95616, USA. E-mail: rogaway@cs.ucdavis.edu ; Web page: http://wwwcsif.cs.ucdavis.edu/~rogaway/homepage.html

1 Introduction

A widely employed paradigm for signing with RSA is to first "hash" the message into a domain point of RSA and then decrypt (ie. exponentiate with the RSA decryption exponent). In particular, this is the basis of several existing standards. Unfortunately, the security of the standardized schemes cannot be justified under standard assumptions about RSA, even assuming the underlying hash functions are ideal.

We propose new schemes, both for signing and for signing with message recovery. They are as simple and efficient as the standardized ones. (In particular, signing takes one RSA decryption plus some hashing, verification takes one RSA encryption plus some hashing, and the size of the signature is the size of the modulus.) But, assuming the underlying hash function is ideal, our methods are not only provably secure, but provably secure in a strong sense: the security of our schemes can be *tightly* related to the security of the RSA function.

Besides providing concrete new schemes for signing with RSA, this work highlights the importance, for practical applications of provable security, of consideration of the tightness of the security reduction, and also provides a rare example of modifying one provably-good scheme in order to obtain another which has a better security bound.

Let us now expand on all of the above. We begin by looking at current practice. Then we consider the full domain hash scheme of [3] which is provable, and discuss its exact security. Finally we come to our new schemes, PSS and PSS-R, and their exact security.

1.1 Signing with RSA– Current practice

THE RSA SYSTEM. In the RSA public key system [15] a party has public key (N, e) and secret key (N, d), where N is a k-bit modulus, the product of two $(k/2)$-bit primes, and $e, d \in \mathsf{Z}^*_{\phi(N)}$ satisfy $ed \equiv 1 \bmod \varphi(N)$. (Think of $k = 1024$, a recommended modulus size these days.) Recall that the RSA function $f\colon \mathsf{Z}^*_N \to \mathsf{Z}^*_N$ is defined by $f(x) = x^e \bmod N$ and its inverse $f^{-1}\colon \mathsf{Z}^*_N \to \mathsf{Z}^*_N$ is defined by $f^{-1}(y) = y^d \bmod N$ $(x, y \in \mathsf{Z}^*_N)$. The generally-made assumption is that f is trapdoor one-way— roughly, if you don't know d (or the prime factors of N) then it is hard to compute $x = f^{-1}(y)$ for a y drawn randomly from Z^*_N.

HASH-THEN-DECRYPT SCHEMES. A widely employed paradigm to sign a document M is to first compute some "hash" $y = \mathit{Hash}(M)$ and then set the signature to $x = f^{-1}(y) = y^d \bmod N$. (To verify that x is a signature of M, compute $f(x) = x^e \bmod N$ and check this equals $\mathit{Hash}(M)$.) In particular, this is the basis for several existing standards. A necessary requirement on *Hash* in such a scheme is that it be collision-intractable and produce a k-bit output in Z^*_N. Accordingly, *Hash* is most often implemented via a cryptographic hash function like $H = \mathsf{MD5}$ (which yields a 128 bit output and is assumed to be collision-intractable) and some padding. A concrete example of such a scheme is [16, 17], where the hash is

$$\mathit{Hash}_{\mathrm{PKCS}}(M) \quad = \quad \mathsf{0x\ 00\ 01\ FF\ FF} \cdots \mathsf{FF\ FF\ 00} \,\|\, H(M)\,.$$

Here $\|$ denotes concatenation, and enough 0xFF-bytes are used so as to make the length of $\mathit{Hash}_{\mathrm{PKCS}}(M)$ equal to k bits.

SECURITY. We draw attention to the fact that the security of a hash-then-decrypt signature depends very much on how exactly one implements *Hash*. In particular, it is important to recognize that the security of a signature scheme like $\mathit{Sign}_{\mathrm{PKCS}}(M) = f^{-1}(\mathit{Hash}_{\mathrm{PKCS}}(M))$ can't be justified given (only) that RSA is trapdoor one-way, even under the assumption that hash function H is ideal. (The reason is that the set of points $\{ \mathit{Hash}_{\mathrm{PKCS}}(M) : M \in \{0,1\}^* \}$ has size at most 2^{128} and hence is a very sparse, and a very structured, subset of Z_N^*.) We consider this to be a disadvantage. We stress that we don't know of any *attack* on this scheme. But we prefer, for such important primitives, to have some proof of security rather than just an absence of known attacks.

The same situation holds for other standards, including ISO 9796 [10]. (There the function *Hash* involves no cryptographic hashing, and the message M is easily recovered from $\mathit{Hash}(M)$. This doesn't effect the points we've just made.)

The above discussion highlights that *collision-intractability is not enough.* The function $\mathit{Hash}_{\mathrm{PKCS}}$ is guaranteed to be collision-intractable if we use a collision-intractable H. But this won't suffice to get a proof of security.

1.2 FDH and its exact security

THE FDH SCHEME. In earlier work [3] we suggested to hash M onto the full domain Z_N^* of the RSA function before decrypting. That is, $\mathit{Hash}_{\mathrm{FDH}}\colon \{0,1\}^* \to \mathsf{Z}_N^*$ is understood to hash strings "uniformly" into Z_N^*, and the signature of M is $\mathit{Sign}_{\mathrm{FDH}}(M) = f^{-1}(\mathit{Hash}_{\mathrm{FDH}}(M))$. (Candidates for suitable functions $\mathit{Hash}_{\mathrm{FDH}}$ can easily be constructed out of MD5 or similar hash functions, as described in [3].) We call this the Full-Domain-Hash scheme (FDH).

PROVABLE SECURITY OF FDH. Assuming $\mathit{Hash}_{\mathrm{FDH}}$ is ideal (ie. it behaves like a random function of the specified domain and range) the security of FDH can be proven assuming only that RSA is a trapdoor permutation. (This is a special case of [3, Section 4], which considers this construction with an arbitrary trapdoor permutation.) This makes the security guarantee of the FDH scheme superior to those of the schemes we discussed in Section 1.1.

Now we want to go further. We will explain how, within the class of provable schemes, quality depends on the quantifiable notion of *exact security.* In this paper we compute the exact security of the FDH scheme, and then we offer a new scheme which has better exact security.

EXACT SECURITY. We quantify the security of RSA as a trapdoor permutation. We say it is (t', ϵ')-secure if an attacker, given y drawn randomly from Z_N^* and limited to running in time $t'(k)$, succeeds in finding $f^{-1}(y)$ with probability at most $\epsilon'(k)$. Values of t', ϵ' for which it is safe to assume RSA is (t', ϵ')-secure can be provided based on the perceived cryptanalytic strength of RSA.

Next we quantify the security of a signature scheme. A signature scheme is said to be $(t, q_{\mathrm{sig}}, q_{\mathrm{hash}}, \epsilon)$-secure if an attacker, provided the public key, allowed to run for time $t(k)$, allowed a chosen-message attack in which she can see up to $q_{\mathrm{sig}}(k)$ legitimate message-signature pairs, and allowed q_{hash} invocations of the (ideal) hash function, is successful in forging the signature of a new message with probability at most $\epsilon(k)$.

EXACT SECURITY OF FDH. The "exact security" of the reduction of [3] used to prove the security of the FDH signature scheme is analyzed in Theorem 3.1. It says that if RSA is (t', ϵ')-secure and $q_{\mathrm{sig}}, q_{\mathrm{hash}}$ are given then the FDH signature scheme is $(t, q_{\mathrm{sig}}, q_{\mathrm{hash}}, \epsilon)$-secure for $t = t' - \mathrm{poly}(q_{\mathrm{sig}}, q_{\mathrm{hash}}, k)$ and $\epsilon = (q_{\mathrm{sig}} + q_{\mathrm{hash}}) \cdot \epsilon'$. Here poly is some small polynomial explicitly specified in Theorem 3.1.

We note that ϵ could thus be considerably larger than ϵ'. This means that even if RSA is quite strong, the guarantee on the signature scheme could be quite weak. To see this, say we would like to allow the forger to see at least $q_{\mathrm{sig}}(k) = 2^{30}$ example signatures and compute hashes on, say, $q_{\mathrm{hash}} = 2^{60}$ strings. Then even if the RSA inversion probability was originally as low as 2^{-61}, all we can say is that the forging probability is now at most $1/2$, which is not good enough. To compensate, we will have to be able to assume that $\epsilon'(k)$ is very, very low, like 2^{-100}. This means that we must have a fairly large value of k, ie. a larger modulus. But this affects the efficiency of the scheme, because the time to do the underlying modular exponentiation grows (and rather quickly) as the modulus size increases. We prefer to avoid this.

We reiterate the crucial point: if the reduction proving security is "loose," like the one above, the efficiency of the scheme is impacted, because we must move to a larger security parameter. Thus, it would be nice to have "tighter" reductions, meaning ones in which ϵ is almost the same as ϵ', with the relations amongst the other parameters staying about the same as they are now.

One might suggest that it is possible to prove a better security bound for FDH than that outlined above. Perhaps, but we don't know how. Instead, we will strengthen the scheme so that a better security bound can be proven.

CLARIFICATION. Before going on, let us clarify our assessments of scheme quality. We are *not* saying the FDH scheme is bad. Indeed, since it is provable, it is ahead of schemes discussed in Section 1.1, and a viable alternative to them. What we are saying is that it is possible to do *even better* than FDH. That is, it is possible to get a scheme which is not only proven secure, but has strong exact security. This successor to FDH is the scheme we discuss next.

1.3 New schemes: PSS and PSS-R

PSS. We introduce a new scheme which we call the *probabilistic signature scheme* (PSS). It is fully specified in Section 4.

The idea is to strengthen the FDH scheme by making the hashing probabilistic. In order to sign message M, the signer first picks a random seed r of length k_0, where $k_0 < k$ is a parameter of the scheme. Then using some hashing, in a specific way we specify, the signer produces from M and r an image point $y = \mathit{Hash}_{\mathsf{PSS}}(M, r) \in \mathsf{Z}_N^*$. As usual, the signature is $x = f^{-1}(y) = y^d \bmod N$. (Verification is a bit more tricky than usual, since one cannot simple "re-compute" this probabilistic hash, but still takes only one RSA encryption and some hashing. See Section 4.) In particular, our scheme is as efficient as the schemes discussed above. But Theorem 4.1 shows that the security can be *tightly* related to that of RSA. Roughly, it says that if RSA is (t', ϵ')-secure then, given $q_{\text{sig}}, q_{\text{hash}}$, scheme PSS is $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$-secure for $t = t' - \text{poly}(q_{\text{sig}}, q_{\text{hash}}, k)$ and $\epsilon = \epsilon' - o(1)$. Here $o(1)$ denotes a function exponentially small in k_0 and k_1 (another parameter of the scheme) and poly denotes a specific polynomial, both of these explicitly specified in the theorem.

Continuing the above example, if the RSA inversion probability was originally as low as 2^{-61}, the probability of forgery for the signature scheme is almost equally low, regardless of the number of sign and hash queries the adversary makes!

PSS WITH RECOVERY. We also have a variant of PSS, called PSS-R, which provides message recovery. The goal is to save on bandwidth. Rather than transmit the message M and its signature x, a single "enhanced signature"τ, of length less than $|M| + |x|$, is transmitted. The verifier will be able to recover M from τ and simultaneously check the authenticity. With security parameter $k = 1024$, our scheme enables one to authenticate a message of up to $n = 767$ bits by transmitting only a total of k bits. PSS-R accomplishes this by appropriately "folding" the message into the signature in such a way that the verifier can recover it. The efficiency and security are the same as for PSS. See Section 5.

RABIN SIGNATURES. The same ideas apply for the Rabin function, and, in particular, we have both a basic Rabin scheme and a variant which provides for message recovery, with security tightly related to the hardness of factoring. See Section 6.

1.4 Discussion

The above illustrates that to fairly compare the efficiency of two provably-secure schemes one needs to look at more than just computation time for a k-bit key. Schemes FDH and PSS have essentially the same computation time when k is fixed. But since PSS has tighter provable security one can safely use a smaller modulus size and thus, ultimately, get greater efficiency.

A numerical example may help to make this clear. Let us again assume that the forger F can compute the hash of at most 2^{60} strings and that she can obtain the signatures of at most 2^{30} messages. Assume that it takes time $Ce^{1.923(\log N)^{1/3}(\log\log N)^{2/3}}$ to invert RSA [12]. Then, our theorems imply that if you use FDH then you must select a modulus of 3447 bits in order to get the same degree of guaranteed-security as you would have gotten had you selected a modulus of 1024 bits and used PSS.

1.5 Related work

We have already discussed the PKCS standards [16, 17] and the ISO standard [10] and seen that their security cannot be justified based on the assumption that RSA is trapdoor one-way. Other standards, such as [1], are similar to [16], and the same statement applies.

The schemes we discuss in the remainder of this section do not use the hash-then-decrypt paradigm.

Signature schemes whose security can be provably based on the RSA assumption include [9, 2, 11, 20, 6]. The major plus of these works is that they do not use an ideal hash function (random oracle) model— the provable security is in the standard sense. On the other hand, the security reductions are quite loose for each of those schemes. On the efficiency front, the efficiency of the schemes of [9, 2, 11, 20] is too poor to seriously consider them for practice. The Dwork-Naor scheme [6], on the other hand, is computationally quite efficient, taking two to six RSA computations, although there is some storage overhead and the signatures are longer than a single RSA modulus. This scheme is the best current choice if one is willing to allow some extra computation and storage, and one wants well-justified security *without* assuming an ideal hash function.

Back among signature schemes which assume an ideal hash, a great many have been proposed, based on the hardness of factoring or other assumptions. Most of these schemes are derived from identification schemes, as was first done by [8]. Some of these methods are provable (in the ideal hash model), some not. In some of the proven schemes exact security is analyzed; usually it is not. In no case that we know of is the security tight. The efficiency varies. The computational requirements are often lower than a hash-then-decrypt RSA signature, although key sizes are typically larger.

The paradigm of protocol design with ideal hash functions (aka random oracles) is developed in [3] and continued in [4]. The current paper is in some ways the analogue, for digital signatures, of our earlier work on encryption [4]. Further work on signing in the random oracle model includes Pointcheval and Stern [13]. (They do not consider exact security, and it may be helpful to do so in their context.)

2 Definitions

We provide definitions for an exact security treatment of RSA, basic signature schemes, and signing with recovery.

2.1 An exact treatment of RSA

THE RSA FAMILY. RSA is a family of *trapdoor permutations.* It is specified by the RSA generator, $\mathcal{RSA}$, which, on input 1^k, picks a pair of random distinct $(k/2)$-bit primes and multiplies them to produce a modulus N. It also picks, at random, an encryption exponent $e \in \mathsf{Z}^*_{\varphi(N)}$ and computes the corresponding decryption exponent d so that $ed \equiv 1 \bmod \varphi(N)$. The generator returns N, e, d, these specifying $f\colon \mathsf{Z}^*_N \to \mathsf{Z}^*_N$ and $f^{-1}\colon \mathsf{Z}^*_N \to \mathsf{Z}^*_N$, which are defined by $f(x) = x^e \bmod N$ and $f^{-1}(y) = y^d \bmod N$. Recall that both functions are permutations, and, as the notation indicates, inverses of each other.

The trapdoor permutation generator $\mathcal{RSA}$–3 is identical to $\mathcal{RSA}$ except that the encryption exponent e is fixed to be 3. More generally, $\mathcal{RSA}$–e provides an encryption exponent of the specified constant. Other variants of $\mathcal{RSA}$ use a somewhat different distribution on the modulus N. Our results, though stated for $\mathcal{RSA}$, also hold for these other variants.

EXACT SECURITY OF THE RSA FAMILY. An *inverting algorithm* for $\mathcal{RSA}$, I, gets input N, e, y and tries to find $f^{-1}(y)$. Its success probability is the probability it outputs $f^{-1}(y)$ when N, e, d are obtained by running $\mathcal{RSA}(1^k)$ and y is set to $f(x)$ for an x chosen at random from Z^*_N. The standard asymptotic definition of security asks that the success probability of any PPT (probabilistic, polynomial time) algorithm be a negligible function of k. We want to go further. We are interested in exactly how much time an inverting algorithm uses and what success probability it achieves in this time. Formally an inverting algorithms is said to be a t-inverter, where $t\colon \mathsf{N} \to \mathsf{N}$, if its running time plus the size of its description is bounded by $t(k)$, in some fixed standard model of computation. We say that I (t, ϵ)-breaks $\mathcal{RSA}$, where $\epsilon\colon \mathsf{N} \to [0, 1]$, if I is a t-inverter and for each k the success probability of I is at least $\epsilon(k)$. Finally, we say that $\mathcal{RSA}$ is (t, ϵ)-secure if there is no inverter which (t, ϵ)-breaks $\mathcal{RSA}$.

EXAMPLE. The asymptotically best factoring algorithm known (NFS) takes time which seems to be about $e^{1.9k^{1/3}(\log k)^{2/3}}$ to factor a k-bit modulus. So one might be willing to assume that the trapdoor permutation family $\mathcal{RSA}$ is (t, ϵ)-secure for any (t, ϵ) satisfying $t(k)/\epsilon(k) \le Ce^{k^{1/4}}$, for some particular constant C.

2.2 Signature schemes and their exact security

SIGNATURE SCHEMES. A *digital signature scheme* $\Pi = (\mathit{Gen}, \mathit{Sign}, \mathit{Verify})$ is specified by a key generation algorithm, *Gen*, a signing algorithm, *Sign*, and a verifying algorithm, *Verify*. The first two are probabilistic, and all three should run in expected polynomial time. Given 1^k, the key generation algorithm outputs a pair of matching public and secret keys, (pk, sk). The signing algorithm takes the message M to be signed and the secret key sk, and it returns a signature $x = \mathit{Sign}_{sk}(M)$. The verifying algorithm takes a message M, a candidate signature x', and the public key pk, and it returns a bit $\mathit{Verify}_{pk}(M, x')$, with 1 signifying "accept" and 0 signifying "reject." We demand that if x was produced via $x \leftarrow \mathit{Sign}_{sk}(M)$ then $\mathit{Verify}_{pk}(M, x) = 1$.

One or more strong hash functions will usually be available to the algorithms *Sign* and *Verify*, their domain and range depending on the scheme. We model them as ideal, meaning that if hash function h is invoked on some input, the output is a uniformly distributed point of the range. (But if invoked twice on the same input, the same thing is returned both times.) Formally, h is a random oracle. It is called a hash oracle and it is accessed via oracle queries: an algorithm can write a string z and get back $h(z)$ in time $|z|$.

SECURITY OF SIGNATURE SCHEMES. Definitions for the security of signatures in the asymptotic setting were provided by Goldwasser, Micali and Rivest [9], and enhanced to take into account the presence of an ideal hash function in [3]. Here we provide an exact version of these definitions.

A forger takes as input a public key pk, where $(pk, sk) \stackrel{R}{\leftarrow} Gen(1^k)$, and tries to forge signatures with respect to pk. The forger is allowed a chosen message attack in which it can request, and obtain, signatures of messages of its choice. This is modeled by allowing the forger oracle access to the signing algorithm. The forger is deemed *successful* if it outputs a *valid forgery*—namely, a message/signature pair (M, x) such that $Verify_{pk}(M, x) = 1$ but M was not a message of which a signature was requested earlier of the signer. The forger is said to be a $(t, q_{\text{sig}}, q_{\text{hash}})$-forger if its running time plus description size is bounded by $t(k)$; it makes at most $q_{\text{sig}}(k)$ queries of its signing oracle; and it makes a total of at most $q_{\text{hash}}(k)$ queries of its various hash oracles. As a convention, the time $t(k)$ includes the time to answer the signing queries. Such a forger F is said to $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$-break the signature scheme if, for every k, the probability that F outputs a valid forgery is at least $\epsilon(k)$. Finally we say that the signature scheme $(Gen, Sign, Verify)$ is $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$-secure if there is no forger who $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$-breaks the scheme.

For simplicity we will assume that a forger does any necessary book-keeping so that it never repeats a hash query. (It might repeat a signing query. If the scheme is probabilistic, this might help it.)

2.3 Quantifying the quality of reductions

Our theorems will have the form: If $\mathcal{RSA}$ is (t', ϵ')-secure, then some signature scheme $\Pi = (Gen, Sign, Verify)$ is $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$-secure. The proof will take a forger F who $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$-breaks Π and produce from F an inverter I who (t', ϵ')-breaks $\mathcal{RSA}$. The quality of the reduction is in how the primed variables depend on the unprimed ones. We will typically view $q_{\text{sig}}, q_{\text{hash}}$ as given, these being query bounds we are willing to allow. (For example, $q_{\text{sig}} = 2^{30}$ and $q_{\text{hash}} = 2^{60}$ are reasonable possibilities.) Obviously we want t' to be as large as possible and we want ϵ' to be as small as possible. We are usually satisfied when $t' = t - \text{poly}(q_{\text{hash}}, q_{\text{sig}}, k)$ and $\epsilon' \approx \epsilon$.

3 The Full-Domain-Hash Scheme – FDH

THE SCHEME. Signature scheme $\mathsf{FDH} = (GenFDH, SignFDH, VerifyFDH)$ is defined as follows [3]. The key generation algorithm, on input 1^k, runs $\mathcal{RSA}(1^k)$ to obtain (N, e, d). It outputs (pk, sk), where $pk = (N, e)$ and $sk = (N, d)$. The signing and verifying algorithms have oracle access to a hash function $H_{\text{FDH}}\colon \{0,1\}^* \rightarrow \mathsf{Z}_N^*$. (In the security analysis it is assumed to be ideal. In practice it can be implemented on top of a cryptographic hash function such as SHA-1.) Signature generation and verification are as follows:

$SignFDH_{N,d}(M)$
  $y \leftarrow H_{\text{FDH}}(M)$
  return $y^d \bmod N$

$VerifyFDH_{N,e}(M, x)$
  $y \leftarrow x^e \bmod N$; $y' \leftarrow H_{\text{FDH}}(M)$
  if $y = y'$ **then return** 1 **else return** 0

SECURITY. The following theorem summarizes the exact security of the FDH scheme as provided by the reduction of [3]. The proof is straightforward, but it is instructive all the same, so we include it. The disadvantage of the result, from our point of view, is that ϵ' could be much smaller than ϵ.

Theorem 3.1 *Suppose $\mathcal{RSA}$ is a (t', ϵ')-secure. Then, for any $q_{\rm sig}, q_{\rm hash}$, signature scheme* FDH *is $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-secure, where*

$$\begin{aligned} t(k) &= t'(k) - [q_{\rm hash}(k) + q_{\rm sig}(k) + 1] \cdot \Theta(k^3) \quad \textit{and} \\ \epsilon(k) &= [q_{\rm sig}(k) + q_{\rm hash}(k) + 1] \cdot \epsilon'(k) \,. \end{aligned}$$

Proof: Let F be a forger which $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-breaks FDH. We present an inverter I which (t', ϵ')-breaks $\mathcal{RSA}$.

Inverting algorithm I is given as input (N, e, y) where N, e, d were obtained by running the generator $\mathcal{RSA}(1^k)$, and y was chosen at random from Z_N^*. It is trying to find $x = f^{-1}(y)$, where f is the RSA function described by N, e. It forms the public key N, e of the Full-Domain-Hash signature scheme, and starts running F on input of this key. Forger F will make two kinds of oracle queries: hash oracle queries and signing queries. Inverter I must answer these queries itself. For simplicity we assume that if F makes sign query M then it has already made hash oracle query M. (We will argue later that this is wlog.) Let $q = q_{\rm sig} + q_{\rm hash}$. Inverter I picks at random an integer j from $\{1, \ldots, q\}$. Now we describe how I answers oracle queries. Here i is a counter, initially 0.

Suppose F makes hash oracle query M. Inverter I increments i and sets $M_i = M$. If $i = j$ then it sets $y_i = y$ and returns y_i. Else it picks r_i at random in Z_N^*, sets $y_i = f(r_i)$, and returns y_i.

Alternatively, suppose F makes signing query M. By assumption, there was already a hash query of M, so $M = M_i$ for some i. Let I return the corresponding r_i as the signature.

Eventually, F halts, outputting some (attempted forgery) (M, x). Let inverting algorithm I output x. Without loss of generality (see below) we may assume that $M = M_i$ for some i. In that case, if (M, x) is a valid forgery, then, with probability at least $1/q$, we have $i = j$ and $x = f^{-1}(y_i) = f^{-1}(y)$ was the correct inverse for f.

The running time of I is that of F plus the time to choose the y_i-values. The main thing here is one RSA computation for each y_i, which is cubic time (or better). This explains the formula for t.

It remains to justify the assumptions. Recall that I is running F. So if the latter makes a sign query without having made the corresponding hash query, I at once goes ahead and makes the hash query itself. Similarly for the output forgery. All this means that the effective number of hash queries is at most $q_{\rm hash} + q_{\rm sig} + 1$, which is the number we used in the time bound above. ▌

Is there a different proof which would achieve a translation in which t is like the above but ϵ is $\Omega(\epsilon')$? We don't believe so. Instead we will modify the scheme to get the security we want. We do this by making the hashing probabilistic.

4 The Probabilistic Signature Scheme – PSS

Here we propose a new scheme—a probabilistic generalization of FDH. It preserves the efficiency and provable security of FDH but achieves the latter with a much better security bound.

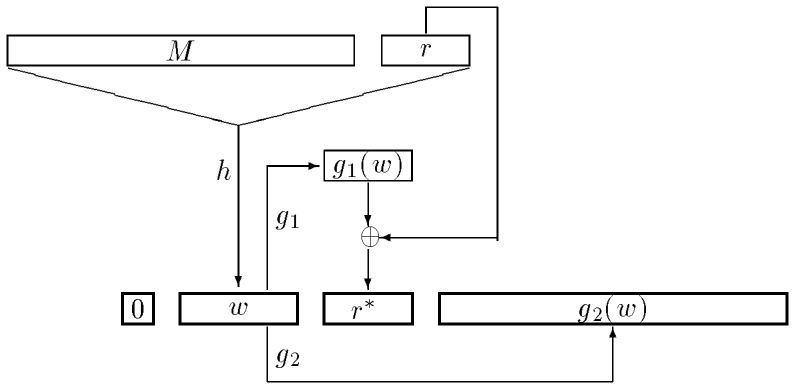

Figure 1: PSS: Components of image $y = 0 \parallel w \parallel r^* \parallel g_2(w)$ are darkened. The signature of M is $y^d \bmod N$.

4.1 Description of the PSS

Signature scheme $\mathsf{PSS}[k_0, k_1] = (\mathit{GenPSS}, \mathit{SignPSS}, \mathit{VerifyPSS})$ is parameterized by k_0 and k_1, which are numbers between 1 and k satisfying $k_0 + k_1 \leq k - 1$. To be concrete, the reader may like to imagine $k = 1024$, $k_0 = k_1 = 128$.

The key generation algorithm *GenPSS* is identical to *GenFDH*: on input 1^k, run $\mathcal{RSA}(1^k)$ to obtain (N, e, d), and output (pk, sk), where $pk = (N, e)$ and $sk = (N, d)$.

The signing and verifying algorithms make use of two hash functions. The first, h, called the compressor, maps as h: $\{0,1\}^* \to \{0,1\}^{k_1}$ and the second, g, called the generator, maps as g: $\{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$. (The analysis assumes these to be ideal. In practice they can be implemented in simple ways out of cryptographic hash functions like MD5, as discussed in Appendix A.) Let g_1 be the function which on input $w \in \{0,1\}^{k_1}$ returns the first k_0 bits of $g(w)$, and let g_2 be the function which on input $w \in \{0,1\}^{k_1}$ returns the remaining $k - k_0 - k_1 - 1$ bits of $g(w)$. We now describe how to sign and verify. Refer to Figure 1 for a picture.

SignPSS (M)
  $r \xleftarrow{R} \{0,1\}^{k_0}$; $w \leftarrow h(M \parallel r)$; $r^* \leftarrow g_1(w) \oplus r$
  $y \leftarrow 0 \parallel w \parallel r^* \parallel g_2(w)$
  return $y^d \bmod N$

VerifyPSS (M, x)
  $y \leftarrow x^e \bmod N$
  Break up y as $b \parallel w \parallel r^* \parallel \gamma$. (That is, let b be the first bit of y, w the next k_1 bits, r^* the next k_0 bits, and γ the remaining bits.)
  $r \leftarrow r^* \oplus g_1(w)$
  if ($h(M \parallel r) = w$ and $g_2(w) = \gamma$ and $b = 0$) **then return** 1
  else return 0

The step $r \xleftarrow{R} \{0,1\}^{k_0}$ indicates that the signer picks at random a seed r of k_0 bits. He then concatenates this seed to the message M, effectively "randomizing" the message, and hashes this down, via the "compressing" function, to a k_1 bit string w. Then the generator g is applied to w to yield a k_0 bit string $r^* = g_1(w)$ and a $k - k_0 - k_1 - 1$ bit string $g_2(w)$. The first is used to "mask" the k_0-bit seed r, resulting in the masked seed r^*. Now $w \parallel r^*$ is pre-pended with a 0 bit and appended with $g_2(w)$ to create the image point y which is decrypted under the RSA function to define the signature. (The 0-bit is to guarantee that y is in Z_N^*.)

Notice that a new seed is chosen for each message. In particular, a given message has many possible signatures, depending on the value of r chosen by the signer.

Given (M, x), the verifier first computes $y = x^e \bmod N$ and recovers r^*, w, r. These are used to check that y was correctly constructed, and the verifier only accepts if all the checks succeed.

Note the efficiency of the scheme is as claimed. Signing takes one application of h, one application of g, and one RSA decryption, while verification takes one application of h, one application of g, and one RSA encryption.

4.2 Security of the PSS

The following theorem proves the security of the PSS based on the security of RSA, but with a relation between the two securities that is much tighter than the one we saw for the FDH scheme. The key difference is that $\epsilon(k)$ is within an additive, rather than multiplicative, factor of $\epsilon'(k)$, and this additive factor decreases exponentially with k_0, k_1. The relation between t and t' is about the same as in Theorem 3.1.

Theorem 4.1 *Suppose that $\mathcal{RSA}$ is (t', ϵ')-secure. Then for any $q_{\text{sig}}, q_{\text{hash}}$ the signature scheme* PSS$[k_0, k_1]$ *is $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$-secure, where*

$$\begin{aligned} t(k) &= t'(k) - [q_{\text{sig}}(k) + q_{\text{hash}}(k) + 1] \cdot k_0 \cdot \Theta(k^3), \quad \textit{and} \\ \epsilon(k) &= \epsilon'(k) + [3(q_{\text{sig}}(k) + q_{\text{hash}}(k))^2] \cdot (2^{-k_0} + 2^{-k_1}) \,. \end{aligned}$$

The rest of this section is devoted to a sketch of the proof of this theorem.

Proof Sketch: Let F be a forger which $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$-breaks the PSS. We present an inverter I which (t', ϵ')-breaks the trapdoor permutation family $\mathcal{RSA}$.

The input to I is N, e and η where η was chosen at random from Z_N^*, and N, e, d were chosen by running the generator $\mathcal{RSA}(1^k)$. (But d is not provided to I!) We let f: $\mathsf{Z}_N^* \to \mathsf{Z}_N^*$ be $f(x) = x^e \bmod N$. I wants to compute $f^{-1}(\eta) = \eta^d \bmod N$. It forms the public key N, e, and starts running F on input this key. F will make oracle queries (signing queries, h-oracle queries, and g-oracle queries), which I must answer itself. We assume no hash query (h or g) is repeated (but a signing query might be repeated). We let $Q_1, \ldots, Q_{q_{\text{sig}}+q_{\text{hash}}}$ denote the sequence of oracle queries that F makes. (This is a sequence of random variables.) This list includes all queries, and we implicitly assume that along with each Q_i is an indication of whether it is a signing oracle query, an h-oracle query or a g-oracle query. In the process of answering these queries, I will "build" or "define" the functions h, g.

I maintains a counter i, initially 0, which is incremented for each query. We now indicate how the queries are answered. It depends on the type of query.

Answering signing queries. First, suppose $Q_i = M$ is a signing query. Let us first try to give some intuition, and then the precise instructions for I to answer the query.

The problem is that I cannot answer a signing query as the signer would since it doesn't know f^{-1}. So, it first picks a point $x \in Z_N^*$, and then arranges that $y = f(x)$ be the image point of a signature of M. (It does this by viewing y as $0 \parallel w \parallel r^* \parallel \gamma$, and then defining $h(M \parallel r) = w$ and $g(w) = r^* {\oplus} r \parallel \gamma$, for some random r.) At this point, x can be returned as a legitimate signature of M. Some technicalities include making sure there are no conflicts (re-defining h or g on points where their values were already assigned) and making sure y has first bit 0. These are attended to in the following full description of the instructions for I to answer signing query Q_i:

(1) Increment i and let $M_i = Q_i$.

(2) Pick $r_i \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$. (Recall this notation means r_i is chosen at random from $\{0,1\}^{k_0}$.)

(3) If $\exists\, j : j < i : r_j = r_i$ then `abort`.

(4) `Repeat` $x_i \stackrel{R}{\leftarrow} \mathsf{Z}_N^*$; $y_i \leftarrow f(x_i)$ `until` the first bit of y_i is 0.

(5) Break up y_i to write it as $0 \| w_i \| r_i^* \| \gamma_i$. (That is, let w_i be the k_1 bits following the 0, let r_i^* be the next k_0 bits, and let γ_i be the last $k - k_0 - k_1 - 1$ bits.)

(6) Set $h(M_i \| r_i) = w_i$.

(7) If $\exists\, j : j < i : w_j = w_i$ then `abort`.

(8) Set $g_1(w_i) = r_i^* \oplus r_i$; Set $g_2(w_i) = \gamma_i$; Set $g(w_i) = g_1(w_i) \| g_2(w_i)$.

(9) Return x_i to F as the answer to signing query $Q_i = M_i$.

Answering h-oracle queries. Next, suppose Q_i is an h-oracle query. We may assume it has length at least k_0 since otherwise it doesn't help the adversary to make this query. Again, before the precise instructions, here is the intuition. The query looks like $M \| r$. We want to arrange that, if F later forges a signature of M using seed r then[1] we invert f at η. To arrange this, we will associate to query $M \| r$ an image of the form ηx_i^e, where x_i is random. (Thus if F later comes up with $f^{-1}(\eta x_i^e) = x_i \eta^d$, then I can divide out x_i and recover $\eta^d = f^{-1}(\eta)$.) This is done by choosing a random x_i, viewing ηx_i^e as $0 \| w \| r^* \| \gamma$, and, as before, defining $h(M \| r) = w$ and $g(w) = r^* \oplus r \| \gamma$. The detailed instructions for I to answer h-oracle query Q_i (taking into account technicalities similar to the above) are:

(1) Increment i and break up Q_i as $M_i \| r_i$. (That is, let r_i be the last k_0 bits of Q_i and let M_i be the rest).

(2) Say Q_i is *old* if $\exists\, j : j < i : M_j \| r_j = M_i \| r_i$, and *new* otherwise. (Since h-queries are not repeated, Q_i is old iff M_i was signing query M_j and in the process of answering it above we picked $r_j = r_i$.) Now if Q_i is old then set $(w_i, r_i^*, \gamma_i) = (w_j, r_j^*, \gamma_j)$ and return w_j (which is $h(M_j \| r_j)$); Else go on to next step.

(3) `Repeat` $x_i \stackrel{R}{\leftarrow} \mathsf{Z}_N^*$; $z_i \leftarrow f(x_i)$; $y_i \leftarrow \eta z_i \bmod N$ `until` the first bit of y_i is 0.

(4) Break up y_i to write it as $0 \| w_i \| r_i^* \| \gamma_i$.

(5) Set $h(M_i \| r_i) = w_i$.

(6) If $\exists\, j : j < i : w_j = w_i$ then `abort`.

(7) Set $g_1(w_i) = r_i^* \oplus r_i$; Set $g_2(w_i) = \gamma_i$; Set $g(w_i) = g_1(w_i) \| g_2(w_i)$.

(8) Return w_i to F as the answer to h-oracle query $Q_i = M_i \| r_i$.

Answering g-oracle queries. Last, suppose Q_i is a g-oracle query. We may assume it has length k_1 since otherwise it doesn't help the adversary to make this query. This time, there is not much to do:

(1) Increment i and let $w_i = Q_i$.

(2) If $\exists\, j : j < i : w_j = w_i$ then return $g(w_j)$. Else pick a string $\alpha \stackrel{R}{\leftarrow} \{0,1\}^{k-k_1-1}$, set $g(w_i) = \alpha$, and return α.

Analysis. Let $\mathsf{Distinct}$ be the event that we never abort in Steps (3) or (7) in answering signing queries or Step (6) in answering h-oracle queries. The reader can verify that $\Pr[\neg\mathsf{Distinct}] \le p$ where $p = 2(q_{\mathrm{sig}} + q_{\mathrm{hash}})^2 \cdot (2^{-k_0} + 2^{-k_1})$. So with probability $\epsilon - p$, $\mathsf{Distinct}$ holds and F halts and outputs

[1] F might forge a signature of M with a seed r' such that h-query $M \| r'$ was never made. But the probability of this is very low.

a valid forgery (M, x). Assume we are in this situation, and let $y = f(x) = x^e \bmod N$. If the first bit of y is not 0 then the forgery is invalid, so assume this bit is 0. So we can break y up to view it as $0 \parallel w \parallel r^* \parallel \gamma$. Let $r = r^* \oplus g_1(w)$. We now claim that with probability at least $\epsilon - p - 2^{-k_1}$, there is an i such that: $(M, r, w, r^*, \gamma) = (M_i, r_i, w_i, r_i^*, \gamma_i)$; h-oracle query $Q_i = M_i \parallel r_i$ was made; and this query was new when it was made. Assuming this claim we have $y = y_i = \eta z_i \bmod N$. Now I outputs $x/x_i \bmod N$. Note $(x/x_i)^e = x^e/x_i^e = y/z_i = \eta$ so x/x_i is indeed $f^{-1}(\eta)$ as desired.

Now let us justify the claim. If $M \parallel r \neq M_i \parallel r_i$ for all i then the probability that $h(M \parallel r) = w$ is at most 2^{-k_1}. So now assume there is such an i. Since (M, x) is a valid forgery we know that M was never a signing query, so it must be that $M \parallel r$ was a h-oracle query. Furthermore, for the same reason, it must have been new.

Finally, note that the time for Step (4) in answering signing queries and Step (3) in answering h-oracle queries can't be bounded. But the expected time is two executions of the loop. So we just stop the loop after $1 + k_0$ steps. This adds at most 2^{-k_0} to the error, completing our proof sketch. ∎

We stress how this proof differs from that of Theorem 3.1. There, we had to "guess" the value of $i \in \{1, \ldots, q_{\text{sig}} + q_{\text{hash}}\}$ for which F would forge a message, and we were only successful if we guessed right. Here we are successful (except with very small probability) no matter what is the value of i for which the forgery occurs.

5 Signing with Message Recovery – PSS-R

MESSAGE RECOVERY. In a standard signature scheme the signer transmits the message M in the clear, attaching to it the signature x. In a scheme which provides message recovery, only an "enhanced signature" τ is transmitted. The goal is to save on the bandwidth for a signed message: we want the length of this enhanced signature to be smaller than $|M| + |x|$. (In particular, when M is short, we would like the length of τ to be k, the signature length.) The verifier recovers the message M from the enhanced signature and checks authenticity at the same time.

We accomplish this by "folding" part of the message into the signature in such a way that it is "recoverable" by the verifier. When the length n of M is small, we can in fact fold the entire message into the signature, so that only a k bit quantity is transmitted. In the scheme below, if the security parameter is $k = 1024$, we can fold up to 767 message bits into the signature.

DEFINITION. Formally, the key generation and signing algorithms are as before, but *Verify* is replaced by *Recover*, which takes *pk* and x and returns $\mathit{Recover}_{pk}(x) \in \{0,1\}^* \cup \{\mathsf{REJECT}\}$. The distinguished point REJECT is used to indicate that the recipient rejected the signature; a return value of $M \in \{0,1\}^*$ indicates that the verifier accepts the message M as authentic. The formulation of security is the same except for what it means for the forger to be *successful*: it should provide an x such that $\mathit{Recover}_{pk}(x) = M \in \{0,1\}^*$, where M was not a previous signing query. We demand that if x is produced via $x \leftarrow \mathit{Sign}_{sk}(M)$ then $\mathit{Recover}_{pk}(x) = M$.

A simple variant of PSS achieves message recovery. We now describe that scheme and its security.

THE SCHEME. The scheme $\mathsf{PSS\text{-}R}[k_0, k_1] = (\mathit{GenPSSR}, \mathit{SignPSSR}, \mathit{RecPSSR})$ is parameterized by k_0 and k_1, as before. The key generation algorithm is *GenPSS*, the same as before. As with PSS, the signing and verifying algorithms depend on hash functions h: $\{0,1\}^* \to \{0,1\}^{k_1}$ and g: $\{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$, and we use the same g_1 and g_2 notation. For simplicity of explication, we assume that the messages to be signed have length $n = k - k_0 - k_1 - 1$. (Suggested choices

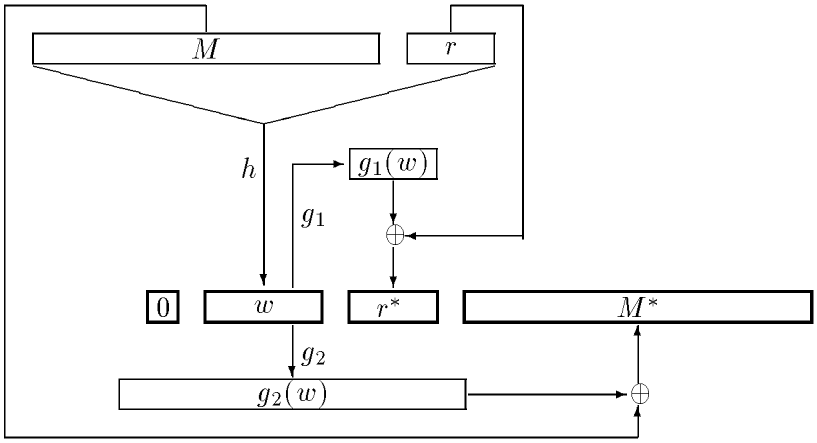

Figure 2: PSS-R: Components of image $y = 0 \parallel w \parallel r^* \parallel M^*$ are darkened.

of parameters are $k = 1024$, $k_0 = k_1 = 128$ and $n = 767$.) In this case, we produce "enhanced signatures" of only k bits from which the verifier can recover the n-bit message and simultaneously check authenticity. Signature generation and verification proceed as follows. Refer to Figure 2 for a picture.

SignPSSR(M)
$r \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$; $w \leftarrow h(M \parallel r)$; $r^* \leftarrow g_1(w) \oplus r$
$M^* \leftarrow g_2(w) \oplus M$
$y \leftarrow 0 \parallel w \parallel r^* \parallel M^*$
return $y^d \bmod N$

RecPSSR(x)
$y \leftarrow x^e \bmod N$
Break up y as $b \parallel w \parallel r^* \parallel M^*$. (That is, let b be the first bit of y, w the next k_1 bits, r^* the next k_0 bits, and M^* the remaining bits.)
$r \leftarrow r^* \oplus g_1(w)$
$M \leftarrow M^* \oplus g_2(w)$
if ($h(M \parallel r) = w$ and $b = 0$) **then return** M **else return** REJECT

The difference in *SignPSSR* with respect to *SignPSS* is that the last part of y is not $g_2(w)$. Instead, $g_2(w)$ is used to "mask" the message, and the masked message M^* is the last part of the image point y.

The above is easily adapted to handle messages of arbitrary length. A fully-specified scheme would use about $\min\{k,\ n + k_0 + k_1 + 16\}$ bits.

Security. The security of PSS-R is the same as for PSS.

Theorem 5.1 *Suppose that* $\mathcal{RSA}$ *is* (t', ϵ')*-secure. Then for any* $q_{\text{sig}}, q_{\text{hash}}$ *the signing-with-recovery scheme* PSS-R$[k_0, k_1]$ *is* $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$*-secure, where*

$$\begin{aligned} t(k) &= t'(k) - [q_{\text{sig}}(k) + q_{\text{hash}}(k) + 1] \cdot k_0 \cdot \Theta(k^3), \quad \textit{and} \\ \epsilon(k) &= \epsilon'(k) + [3(q_{\text{sig}}(k) + q_{\text{hash}}(k))^2] \cdot (2^{-k_0} + 2^{-k_1}) \,. \end{aligned}$$

The proof of this theorem is very similar to that of Theorem 4.1 and hence is omitted.

6 Rabin signatures – PRab and PRab-R

The ideas of this paper extend to Rabin signatures [18, 19], yielding a signature scheme and a signing with recovery scheme whose security can be tightly related to the hardness of factoring.

THE SCHEME. Scheme $\mathsf{PRab}[k_0, k_1] = (\mathit{GenPRab}, \mathit{SignPRab}, \mathit{VerifyPRab})$, the probabilistic Rabin scheme, depends on parameters k_0, k_1, where $k_0 + k_1 \leq k$. Algorithm *GenPRab*, on input 1^k, picks a pair of random distinct $(k/2)$-bit primes p, q and multiplies them to produce the k-bit modulus N. It outputs (pk, sk), where $pk = N$ and $sk = (N, p, q)$.

The signing and verifying algorithms of PRab use hash functions h, g, where h: $\{0,1\}^* \to \{0,1\}^{k_1}$ and g: $\{0,1\}^{k_1} \to \{0,1\}^{k-k_1}$. We let g_1 be the function which on input $w \in \{0,1\}^{k_1}$ returns the first k_0 bits of $g(w)$, and let g_2 be the function which on input $w \in \{0,1\}^{k_1}$ returns the remaining $k - k_0 - k_1$ bits of $g(w)$.

The signing procedure, *SignPRab*, is similar to the corresponding *SignPSS*, but it returns a random square root of the image y, as opposed to $y^d \bmod N$. We stress that a random root is chosen; a fixed one won't do. The verification procedure checks if the square of the signature has the correct image. Thus verification is particularly fast. Here, in full, are *SignPRab* and *VerifyPRab*:

> *SignPRab*(M)
>   **repeat**
> $r \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$; $w \leftarrow h(M \,\|\, r)$; $r^* \leftarrow g_1(w) \oplus r$
> $y \leftarrow w \,\|\, r^* \,\|\, g_2(w)$
>   **until** y is a quadratic residue $\bmod N$.
>   Let $\{x_1, x_2, x_3, x_4\}$ be the four distinct square roots of y in Z_N^*.
>   Let $x \stackrel{R}{\leftarrow} \{x_1, x_2, x_3, x_4\}$.
>   **return** x

> *VerifyPRab*(M, x)
> $y \leftarrow x^2 \bmod N$
>   Break up y as $w \,\|\, r^* \,\|\, \gamma$. (That is, let w be the first k_1 bits of y,
> r^* the next k_0 bits, and γ the remaining bits.)
> $r \leftarrow r^* \oplus g_1(w)$
>   **if** ($h(M \,\|\, r) = w$ and $g_2(w) = \gamma$) **then return** 1 **else return** 0

EXACT SECURITY OF FACTORING. This scheme is based on the hardness of factoring, so we need an exact security formulation of the hardness of factoring assumption.

A factoring algorithm takes a k-bit number and tries to factor it. It is a t-factoring algorithm if the size of its description plus its running time is at most $t(k)$ for every k. We say that A (t, ϵ)-factors if, given a number which is the product of two random distinct $(k/2)$-bit primes, A produces the correct factorization with probability at least $\epsilon(k)$. We say that factoring is (t, ϵ)-hard if there is no algorithm which (t, ϵ)-factors. A reasonable assumption would be that factoring is (t, ϵ)-hard for any t, ϵ satisfying $t(k)/\epsilon(k) = e^{k^{1/4}(\log k)^{3/4}}$.

SECURITY OF THE PRab. The following theorem says that the security of PRab is similar to that of PSS.

Theorem 6.1 *Suppose that factoring is (t', ϵ')-hard. Then for any $q_{\rm sig}, q_{\rm hash}$ the signature scheme* $\mathsf{PRab}[k_0, k_1]$ *is $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-secure, where*

$$
\begin{aligned}
t(k) &= t'(k) - [q_{\rm sig}(k) + q_{\rm hash}(k) + 1] \cdot k_0 \cdot \Theta(k^2), \quad \textit{and} \\
\epsilon(k) &= 2\epsilon'(k) + [6(q_{\rm sig}(k) + q_{\rm hash}(k))^2] \cdot (2^{-k_0} + 2^{-k_1}) .
\end{aligned}
$$

The proof of this theorem is analogous to that of Theorem 4.1. Given a forger F who $(t, q_{\rm sig}, q_{\rm hash}, \epsilon)$-breaks PRab we construct an algorithm which (t', ϵ')-factors. We begin by picking an element $\alpha \in \mathsf{Z}_N^*$ at random and setting $\eta = \alpha^2 \bmod N$. Then we proceed as in the proof of Theorem 4.1, with e set to 2 rather than to the RSA encryption exponent. We thereby recover a square root of η with probability $\epsilon(k) - \delta(k)$ where $\delta(k) = [3(q_{\rm sig}(k) + q_{\rm hash}(k))^2] \cdot (2^{-k_0} + 2^{-k_1})$. But with probability $\epsilon(k)/2 - \delta(k)$ this square root is different from α and hence we factor N. Thus we have a factor of two deterioration in the success probability. On the other hand, there is an improvement in the time complexity, since our algorithm has to raise numbers to the power two rather than to an arbitrary RSA exponent e, thereby bringing the $\Theta(k^3)$ time to $\Theta(k^2)$. Also, it is a potentially weaker assumption to say that factoring is (t', e') hard.

Recovery. As with PSS, we can add message recovery to the PRab scheme in the same way, resulting in the PRab-R signing-with-recovery scheme. Its security is the same as that of PRab.

Acknowledgments

Thanks to Tal Rabin for many helpful comments and corrections.

References

[1] D. Balenson, "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers," IETF RFC 1423, February 1993.

[2] M. Bellare and S. Micali, "How to sign given any trapdoor permutation," *JACM* Vol. 39, No. 1, 214-233, January 1992.

[3] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," *Proceedings of the First Annual Conference on Computer and Communications Security*, ACM, 1993.

[4] M. Bellare and P. Rogaway, "Optimal Asymmetric Encryption," *Advances in Cryptology – Eurocrypt 94 Proceedings*, Lecture Notes in Computer Science Vol. 950, A. De Santis ed., Springer-Verlag, 1994.

[5] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Info. Theory* IT-22, 644-654, November 1976.

[6] C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. *Advances in Cryptology – Crypto 94 Proceedings*, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994.

[7] T. El Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, Vol. 31, No. 4, July 1985.

[8] A. Fiat and A. Shamir, "How to prove yourself: practical solutions to identification and signature problems," *Advances in Cryptology – Crypto* 86 *Proceedings*, Lecture Notes in Computer Science Vol. 263, A. Odlyzko ed., Springer-Verlag, 1986.

[9] S. Goldwasser, S. Micali and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal of Computing*, 17(2):281–308, April 1988.

[10] ISO/IEC 9796, "Information Technology Security Techniques – Digital Signature Scheme Giving Message Recovery," International Organization for Standards, 1991.

[11] M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," *Proceedings of the* 21st *Annual Symposium on Theory of Computing*, ACM, 1989.

[12] A. Lenstra and H. Lenstra (eds.), "The development of the number field sieve," Lecture Notes in Mathematics, vol 1554, Springer-Verlag, 1993.

[13] D. Pointcheval and J. Stern, "Security proofs for signatures," *Advances in Cryptology – Eurocrypt* 96 *Proceedings*, Lecture Notes in Computer Science Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.

[14] R. Rivest, "The MD5 Message-Digest Algorithm," IETF RFC 1321, April 1992.

[15] R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," CACM 21 (1978).

[16] RSA Data Security, Inc., "PKCS #1: RSA Encryption Standard (Version 1.4)." June 1991.

[17] RSA Data Security, Inc., "PKCS #7: Cryptographic Message Syntax Standard (version 1.4)." June 1991.

[18] M. Rabin, "Digital signatures," in *Foundations of secure computation,* R. A. Millo et. al. eds, Academic Press, 1978.

[19] M. Rabin., "Digital signatures and public key functions as intractable as factorization," MIT Laboratory for Computer Science Report TR-212, January 1979.

[20] J. Rompel, "One-Way Functions are Necessary and Sufficient for Secure Signatures," *Proceedings of the* 22nd *Annual Symposium on Theory of Computing*, ACM, 1990.

[21] H. Williams, "A modification of the RSA public key encryption procedure," *IEEE Transactions on Information Theory*, Vol. IT-26, No. 6, November 1980.

A How to implement the hash functions h, g

In the PSS we need a concrete hash function h with output length some given number k_1. Typically we will construct h from some cryptographic hash function H such as H = MD5 or H = SHA-1. Ways to do this have been discussed before in [3, 4]. For completeness we quickly summarize some of these possibilities. The simplest is to define $h(x)$ as the appropriate-length prefix of

$$H(\texttt{const}.\langle 0\rangle.x) \,\|\, H(\texttt{const}.\langle 1\rangle.x) \,\|\, H(\texttt{const}.\langle 2\rangle.x) \,\|\, \cdots \ .$$

The constant `const` should be unique to h; to make another hash function, g, simply select a different constant.