Design and Analysis of a Non-deterministic Digital Signature Protocol

†Odule, Tola John[1] and Olatubosun Abiodun Kaka[2]

[1]*Department of Mathematical Sciences, Olabisi Onabanjo University*
P. M. B. 2002, Ago-Iwoye, Ogun State, Nigeria

[2]*Department of Computer Science, Tai Solarin College of Education*
P. M. B. 2128, Ijebu-Ode, Ogun State, Nigeria

tee_johnny@yahoo.com, boskaykk@yahoo.com

Abstract

This study describes a modular arithmetic-based signing scheme called NDSP 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.

Keywords: *Full domain hash, trapdoor permutation, ideal hash, probabilistic signature, provable security*

1. Introduction

A widely employed paradigm to sign a document M is to first compute some "hash" $y = Hash(M)$ and then set the signature to $x = f^{-1}(y) = y^d \bmod N$ [12]. To verify that x is a signature of M, compute $f(x) = x^e \bmod N$ and check this equals $Hash(M)$. In particular, this is the basis for several existing standards. A necessary requirement on $Hash$ in such scheme is that it be collision-intractable and produce a k-bit output in $\mathrm{Z}_{\mathrm{N}}^{*}$ Accordingly, $Hash$ is most often implemented via a cryptographic hash function like $\mathrm{H} = \mathrm{MD5}$ (which yields a 128 bit output and is assumed to be collision-intractable) and some padding. A concrete example of such a scheme is [12, 13], where the hash is

$$Hash_{PKCS}(M) = 0x\,00\,01\,\mathrm{FF\,FF}\cdots\mathrm{FF\,FF\,FF}\,00 \,\|\, H(M)$$

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

We hereby emphasize that the security of such a scheme depends very much on how exactly one implements $Hash$ [16]. In particular, it is important to recognize that the security of a signature scheme like $Sign_{PKCS}(M) = f^{-1}(Hash_{PKCS}(M))$ can't be justified given

† Correspondence author

(only) that RSA is trapdoor one-way, even under the assumption that hash function H is ideal.

1.1 Full Domain Hash Scheme

Earlier work like [4] suggested hashing M onto the full domain Z_N^* of the RSA function before decrypting. That is, $Hash_{FDH} : \{0,1\}^* \to Z_N^*$ is understood to hash strings "uniformly" into Z_N^*, and the signature of M is $Sign_{FDH}(M) = f^{-1}(Hash_{FDH}(M))$. (Candidates for suitable functions $Hash_{FDH}$ can easily be constructed out of $MD5$ or similar hash functions, as described in [3].) This is referred to as the Full-Domain-Hash scheme (FDH). Assuming $Hash_{FDH}$ is ideal such that 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.

The security of RSA as a trapdoor permutation is hereby quantified. We say it is $(t', \in')$-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 $\in(k)$. Values of t for which it is safe to assume RSA is $(t', \in')$-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 $(t, q_{sig}, q_{hash}, \in)$-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 $q_{sig}(k)$ 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 $\in(k)$.

The reduction of [4] used to prove the security of the FDH signature scheme is analyzed in Theorem 3.1. It says that if RSA is $(t', \in')$-secure and q_{sig}, q_{hash} are given then the FDH signature scheme is $(t, q_{sig}, q_{hash}, \in)$-secure for $t = t' - \text{poly}(t, q_{sig}, q_{hash}, k)$ and $\in = (q_{sig} + q_{hash}) \cdot \in'$ Here poly is some small polynomial explicitly specified in the Theorem.

Our new scheme, referred to as, a *nondeterministic digital signature protocol*, 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 $y = Hash_{PSS}(M, r) \in Z_N^*$. As usual, the signature is $x = f^{-1}(y) = y^d \bmod N$. Verification is no longer straightforward, since one cannot just "re-compute" this probabilistic hash, but still takes only one RSA encryption and some hashing.

1.2 Related Work

It has been shown in [13, 14] security of [9] cannot be justified based on the assumption that RSA is trapdoor one-way. Other standards, such as [1], are similar to [13], and the same statement applies. The scheme we discuss in the remainder of this section does not use the hash-then-decrypt paradigm.

Signature schemes whose security can be provably based on the RSA assumption include [8, 2, 10, 15, 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 [8, 2, 10, 15] 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 [7]. 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, known as random oracles, is developed in [4] and continued in [3]. Further work on signing in the random oracle model includes Pointcheval and Stern [11].

2. Notations and Definitions

Definitions and notational conventions that will be used in the context of our discussion in the remainder of this article are hereby provided.

2.1 Signature Schemes

A digital signature scheme $\Pi = (Gen, Sign, Verify)$ 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 $x = Sign_{sk}(M)$. The verifying algorithm takes a message M, a candidate signature x', and the public key pk, and it returns a bit $Verify_{pk}(M, x')$, with 1 signifying “accept” and 0 signifying “reject.” We demand that if x was produced via $x \leftarrow Sign_{sk}(M)$ then $Verify_{pk}(M, x) = 1$.

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 $|z|$.

2.2 Security of Signature Schemes

Definitions for the security of signatures in the asymptotic setting were provided by [8], and enhanced to take into account the presence of an ideal hash function in [4]. Here we provide an exact version of these definitions.

A forger takes as input a public key pk, where $(pk, sk) \xleftarrow{R} Gen(1^k)$, 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 modelled 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_{sig}, q_{hash})-forger if its running time plus description size is bounded by $t(k)$; it makes at most $q_{sig}(k)$ queries of its signing oracle; and it makes a total of at most $q_{hash}(k)$ 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 $(t, q_{sig}, q_{hash}, \in)$-break the signature scheme if, for every k, the probability that F outputs a valid forgery is at least $t'(k)$. Finally we say that the signature scheme $(Gen, Sign, Verify)$ is $(t, q_{sig}, q_{hash}, \in)$-secure if there is no forger who $(t, q_{sig}, q_{hash}, \in)$-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 Approach to Reductions

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

3. The Full-Domain-Hash Scheme – FDH

The scheme. Signature scheme $\text{FDH} = (GenFDH, SignFDH, VerifyFDH)$ is defined as follows [4]. The key generation algorithm, on input 1^k, runs $RSA(1^k)$ 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 $H_{FDH} : \{0,1\}^* \to Z_N^*$. (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:

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

The following theorem summarizes the exact security of the FDH scheme as provided by the reduction of [4]. 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 $\in'$ could be much smaller than $\in$.

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

$$t(k) = t'(k) - [q_{hash}(k) + q_{sig}(k) + 1] \cdot \Theta(k^3) \text{ and}$$
$$\in(k) = [q_{sig}(k) + q_{hash}(k) + 1] \cdot \in'(k).$$

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

Inverting algorithm I is given as input (N, e, y) where N, e, d were obtained by running the generator $RSA(1^k)$, 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 without loss of generality (wlog).) Let $q = q_{sig} + q_{hash}$. 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, with an output of some (attempted forgery) (M, x). Let the inverting algorithm I output x. wlog (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 $q_{hash} + q_{sig} + 1$, which is the number we used in the time bound above.

4. A Non-deterministic Digital Signature Protocol (NDSP)

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

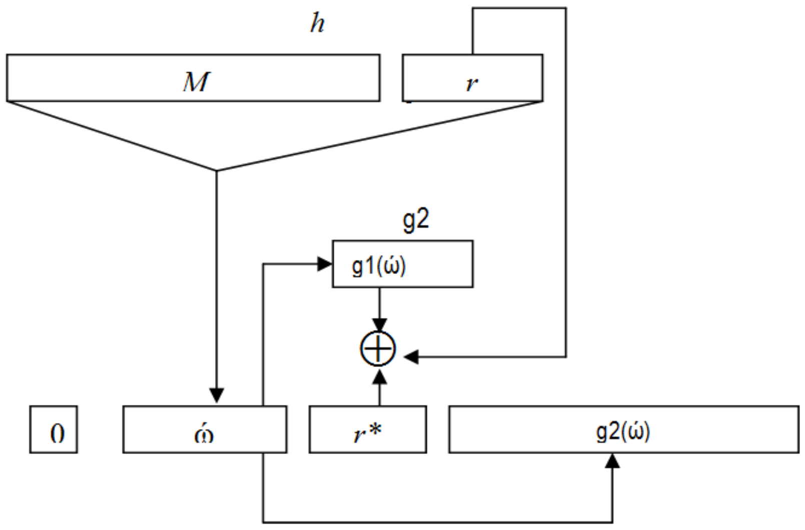

Figure 1. Non-deterministic Digital Signature Protocol: Components of image $y = 0 \| \omega \| r^* \| g2(\omega)$ are darkened. The signature of M is $y^d \bmod N$.

4.1 Description of the NDSP

Signature scheme $NDSP[k_0, k_1] = (GenNDSP, SignNDSP, VerifyNDSP)$ is parameterized by k_0 and k_1, which are numbers between 1 and k satisfying $k_0 + k_1 \le k - 1$. To be concrete, the reader may like to imagine $k = 1024$, $k_0 = k_1 = 128$.

The key generation algorithm $GenNDSP$ is identical to $GenFDH$: on input 1^k, run $RSA(1^k)$ 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 $g:\{0,1\}^{k_1} \rightarrow \{0,1\}^{k-k_1-1}$. (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 $\omega \in \{0,1\}^{k_1}$ returns the first k_0 bits of $g(\omega)$, and let g_2 be the function which on input $\omega \in \{0,1\}^{k_1}$ returns the remaining $k-k_0-k_1-1$ bits of $g(\omega)$. We now describe how to sign and verify, schematically depicted in Figure 1.

```
$SignNDSP(M)$
    $r \xleftarrow{R} \leftarrow \{0,1\}^{k_0}$ ; $\omega \leftarrow h(M \| r)$; $r^* \leftarrow g_1(\omega) \oplus r$
    $y \leftarrow 0 \| \omega \| r^* \| g_2(\omega)$
    **return** $y^d \bmod N$
$VerifyNDSP(M, x)$
    $y \leftarrow x^e \bmod N$
    $Break\ up\ y\ as\ b \| \omega \| r^* \| y$ (that is let b be the first bit of y, ω the next $k_1$ bits
    $r^*$ the next $k_0$ bits and $y$ the remaining bits.)
    $r \leftarrow r^* \oplus g_1(\omega)$
    $if$ $(h(M \| r) = \omega$ and $g_2(\omega) = y$ 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 ω. Then the generator g is applied to ω yielding a k_0 bit string $r^* = g_1(\omega)$ and a $k-k_0-k_1-1$ bit string $g_2(\omega)$. The first is used to "mask" the k_0-bit seed r, resulting in the masked seed r^*. Now $\omega \| r^*$ is pre-pended with a 0 bit and appended with $g_2(\omega)$ 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^*.)

We remark 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^*, ω, r. These are used to check that y was correctly constructed, and the verifier only accepts if all the checks succeed.

Note that 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 Nondeterministic Signature Protocol

The following theorem proves the security of our scheme based on the security of RSA, but with a relation between the two securities that is much tighter than was given for the FDH scheme. The key difference is that $\in(k)$ is within an additive, rather than multiplicative, factor of $\in'(k)$, 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* RSA *is* $(t',\in')$*-secure. Then for any* q_{sig}, q_{hash} *the signature scheme* $NDSP[k_0, k_1]$ *is* $(t, q_{sig}, q_{hash}, \in)$*-secure, where*

$t(k) = t'(k) - [q_{sig}(k) + q_{hash}(k) + 1] \cdot k_0 \cdot \Theta(k^3)$ *and*

$\in(k) = \in'(k) + [3(q_{sig}(k) + q_{hash}(k))^2] \cdot (2^{-k_0} + 2^{-k_1})$.

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

Proof Sketch: Let F be a forger which $(t, q_{sig}, q_{hash}, \in)$-breaks the $\mathrm{NDS}P$. We present an inverter I which $(t',\in')$-breaks the trapdoor permutation family 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 $RSA(1^k)$. (But d is not provided to I !) We let $f: Z_N^* \to Z_N^*$ 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 $Q_1,\ldots,Q_{qsig+qhash}$ 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 $0 \| \omega \| r^* \| \gamma$, and then defining $h(M \| r) = \omega$ and $g(\omega) = r^* \oplus r \| \gamma$ 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 $r_i \xleftarrow{R} \{0,1\}^{k_0}$. (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 $x_i \xleftarrow{R} Z_N^*$; $y \leftarrow f(x_i)$ until the first bit of y_i is 0.

(5) Break up y_i to write it as $0 \| \omega_i \| r_i^* \| \gamma_i$. (That is, let ω_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 $h(M_i \| r_i) = \omega_i$.

(7) If $\exists\, j : j < i : \omega_j = \omega_i$ then abort.

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

(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 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 $f^{-1}(\eta x_i^e) = \eta^d$, 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 $0 \| \omega \| r^* \| \gamma$, and, as before, defining $h(M \| r) = \omega$ and $g(\omega) = r^* \oplus r \| \gamma$. 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 $(\omega_i, r_i^*, \gamma_i) = (\omega_j, r_j^*, \gamma_j)$ and return ω_i (which is $h(M_j \| r_j)$); Else go on to next step.

(3) Repeat $x_i \xleftarrow{R} Z_N^*$; $y \leftarrow f(x_i)$; $z_i \leftarrow f(x_i)$; $y_i \leftarrow \eta z_i \bmod N$ until the first bit of y_i is 0.

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

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

(6) If $\exists\, j : j < i : \omega_j = \omega_i$ then abort.

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

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

Answering g -oracle queries. Last, suppose h 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 $\omega_i = Q_i$.

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

Analysis. Let 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 $\Pr[\neg\text{Distinct}] \le p$ where $p = 2(q_{sig} + q_{hash})^2 \cdot (2^{-k_0} + 2^{-k_1})$. So with probability $\in -p$, Distinct holds and F halts and outputs 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 $0 \| \omega_i \| r^* \| \gamma_i$.

Let $r = r^* \oplus g_1(\omega)$ .We now claim that with probability at least $\in -p - 2^{-k_1}$, there is an i such that: $(M, r, \omega, r^*, \gamma) = (M_i, r_i, \omega_i, r_i^*, \gamma_i)$; h -oracle query $Q_i = M_i \| r_i$ 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.

Next, we justify the claim. If $M \| r \neq M_i \| r_i$ for all i then the probability that $h(M \| r) = \omega$ 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 $M \| r$ 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, thus completing our proof sketch.

We stress how this proof differs from that of Theorem 3.1. There, we had to "guess" the value of $i \in \{1, \ldots, q_{sig} + q_{hash}\}$ 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. Conclusion

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 $(t', \in')$ -secure then, given q_{sig} q_{hash} scheme PSS is $(t, q_{sig}, q_{hash}, \in)$ -secure for $t = t' - \text{poly}(t, q_{sig}, q_{hash}, k)$ and $\in = \in' - o(1)$. 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. In line with the RSA de facto industry standard, 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!

Acknowledgements

We thank M. Bellare and P. Rogaway for allowing us to make use of some of the materials from their research notes.

References

[1] S. Santesson, "X.509 Certificate Extension for S/MIME Capabilities", Draft IETF-SMIME-CERTCAPA 05.txt, **(2005)**.

[2] M. Bellare and S. Micali, "How to sign given any trapdoor permutation", Journal of ACM, vol. 39, no. 1, **(1992)**, pp. 214-233.

[3] T. J. Odule, "Incremental Cryptography and Security of Public Hash Functions", Journal of Nigerian Association of Mathematical Physics, vol. 11, **(2007)**, pp. 467-471.

[4] T. J. Odule, "Design of Provably-secure Cryptographic Hash Functions", Journal of Nigerian Association of Mathematical Physics, vol. 13, **(2008)**, pp. 273-280.

[6] T. J. Odule, "Adaptive Update Algorithm for Collision-free Hash-and-sign Signature Schemes", International Journal of Physical Sciences, vol. 3, no. 1, **(2008)**, pp. 81-87.

[7] 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)**.

[8] H. Gilbert and H. Handschuh, "Security Analysis of SHA-256 and sisters", in Selected Areas in Cryptography. M. Matsui and R. Zuccherato (eds.), vol. 3006, Lecture Notes in Computer Science, Springer Verlag, **(2004)**, pp.175-193.

[9] B. Waters, "Efficient Identity-based Encryption without Random Oracles", in EUROCRYPT, vol. 3494, Lecture Notes in Computer Science, **(2005)**, pp. 114-127.

[10] 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)**.

[11] D. Pointcheval, J. Stern and U. Maurer (eds.), "Security proofs for signatures", Advances in Cryptology Eurocrypt 96 Proceedings, Lecture Notes in Computer Science, vol. 1070, Springer-Verlag, **(1986)**.

[12] R. Rivest, A. Shamir and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems", Communications of the ACM (CACM), vol. 21, **(1978)**.
[13] RSA Data Security, Inc., "PKCS #1: RSA Encryption Standard", (Version 1.4), **(1991)**.
[14] RSA Data Security, Inc., "PKCS #7: Cryptographic Message Syntax Standard", (version 1.4), **(1991)**.
[15] J. Rompel, "One-Way Functions are Necessary and Sufficient for Secure Signatures", Proceedings of the 22nd Annual Symposium on Theory of Computing, ACM, **(1990)**.
[16] T. J. Odule, "A modular Arithmetic-based encryption scheme", African Journal of Pure & applied sciences, vol. 2, no. 1, **(2008)**.

Appendix: 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 $H = \mathrm{MD5}$ or $H = \mathrm{SHA-1}$. 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 $H(const \cdot \| \langle 0 \rangle \cdot x) \| H(const \cdot \| \langle 1 \rangle \cdot x) \| H(const \cdot \| \langle 2 \rangle \cdot x) \| \cdots$.

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