Security Proofs for the RSA-PSS Signature Scheme and Its Variants – Draft 1.1

Jakob Jonsson

RSA Laboratories Europe, Stockholm, SWEDEN
jjonsson@rsasecurity.com

July 3, 2001

Abstract

We analyze the security of different versions of the adapted RSA-PSS signature scheme, including schemes with variable salt lengths and message recovery. We also examine a variant with Rabin-Williams (RW) as the underlying verification primitive. Our conclusion is that the security of RSA-PSS and RW-PSS in the random oracle model can be tightly related to the hardness of inverting the underlying RSA and RW primitives, at least if the PSS salt length is reasonably large. Our security proofs are based on already existing work by Bellare and Rogaway [3] and by Coron [10], who examined signature schemes based on the original PSS encoding method.
Keywords: digital signatures, factoring, public-key cryptography, RSA.

1 Introduction

A widely employed procedure for digitally signing a message is to encode the message into an element in the image of a one-way trapdoor function (e.g., the RSA encryption function [32]; see Section 4.1) before applying the inverse of the trapdoor function (e.g., the RSA decryption function) to the element. During the years, quite a few such encoding methods have been proposed, including the popular method used in the RSA-based signature scheme defined in PKCS #1 v1.5 [33]. While this scheme has remained unbroken, its security is based on ad hoc assumptions and cannot be expressed in terms of the hardness of inverting the RSA encryption function. This has the effect that we cannot exclude the possibility that an attacker knows how to efficiently forge signatures without being able to invert the RSA encryption function.

To address this concern, Bellare and Rogaway [3] introduced the PSS encoding method in 1996. Combined with the RSA or the Rabin [31] one-way trapdoor functions, PSS provides *provable* security. Specifically, Bellare and Rogaway were able to demonstrate that the exact security of the combinations RSA-PSS and Rabin-PSS can be expressed in terms of the hardness of inverting the underlying RSA and Rabin trapdoor functions. The security proofs take place in the random oracle model (hash functions are treated as "ideal"). Coron [9, 10] analyzed RSA-PSS further and improved the security proof for small salt lengths; PSS uses a salt (a random bit string) when encoding messages to be signed. The 1996 version of PSS is specified in Section 5.1.

A slightly modified version of PSS is suitable also in combination with the Rabin-Williams (RW) algorithm [19, 20], which is an adapted version of the Rabin algorithm [31] and based on work by Williams [39]; see Section 4.2. A rigorous security proof however has not been published for this combination.

During the process of adopting PSS into the IEEE P1363a [18] standards effort, certain adaptations to the original version of PSS were made by Bellare and Rogaway [4] and also by Burt Kaliski (the editor of IEEE P1363a) to facilitate implementation and integration into existing protocols. The adapted version is intended to be used in combination with RSA or RW. See Section 5.2 for a description of the new PSS encoding method and rationale for the modifications.

Our purpose is to examine the security of signature schemes based on the adapted PSS encoding method. Our contributions can be summarized as follows.

- We verify that the adaptations are sound. However minor a modification may appear to be, without a careful analysis of the effects of the modification, one cannot be completely sure that there are no undesired side effects. We provide that analysis here. Our security proof for RSA-PSS is based on Coron's proof in [10].
- We analyze different variants of RSA-PSS and RW-PSS, including schemes with variable PSS salt length and schemes with message recovery.
- We give a security proof for RW-PSS based on the security proof for RSA-PSS. While the Rabin-PSS security proof in [3] was completely straightforward in terms of the RSA-PSS security proof, the somewhat complex theory behind the RW primitive requires a more thorough additional analysis. On the other hand, the careful design of RW-PSS allows for tighter security bounds than those for Rabin-PSS in [3].
- We introduce an improved signature sampling method compared to the method used in [3, 10]. The method applies to both the new and the original version of PSS. While the improvement makes no difference asymptotically, it still adds a few bits of provable security, which might be of importance in particular cases. See Section 8 for details.

As in [3, 9, 10], our security proofs take place in the random oracle model, i.e., hash functions are assumed to produce random and unpredictable outputs for any given input. See the concluding discussion in Section 13 for some comments.

1.1 Related work

Over the years, plenty of signature schemes have been proposed. It is beyond the scope of this paper to give a complete historical overview; we will restrict our attention to some of the more important schemes of today together with a few interesting recent inventions.

Traditionally, asymmetric signature and encryption schemes have been based on a presumably hard number-theoretic problem. Probably the two most exploited problems are the factoring problem and the discrete logarithm problem.

RSA and RW are examples of primitives based on the factoring problem (though inverting RSA is not known to be equivalent to factoring the underlying modulus). There are several signature schemes based on the RSA algorithm; see [21] for further discussion. The ACE signature scheme [37] is a scheme with a primitive based on a hard problem similar to the RSA problem. While less efficient than RSA-PSS, ACE has the advantage of being provably secure in a *standard* model (i.e., without the random oracle assumption on the underlying hash functions). Yet, the security reduction for ACE, with or without the random oracle assumption, is not as tight as the reduction provided for RSA-PSS in this paper. Gennaro, Halevi, and Rabin [12] has introduced a very simple signature scheme with a security proof in the standard model of approximately the same strength as the one for ACE. ESIGN [11] is another signature scheme with a similar underlying hard problem. ESIGN is provably secure in the random oracle model and is significantly faster than RSA-PSS and RW-PSS. Yet, the reduction in the security proof is not tight and the underlying hard problem might be easier to solve than inverting RSA. ESIGN is specified in Draft 9 of IEEE P1363a [18]. Finally, we mention the GQ1 and GQ2 schemes [14], which are based on zero-knowledge identification schemes.

The discrete logarithm problem is to solve equations of the form $a^x = b$ in x, where a and b are elements in a finite group. IEEE Std 1363-2000 [17] includes several schemes with cryptographic primitives based on this problem. In DSA [1] the underlying group is a subgroup of the multiplicative group of integers modulo a prime p, whereas in ECDSA [2] the group is a subgroup of an elliptic curve group over a finite field. The exact security of ECDSA is analyzed in [7]. Based on work by Nyberg and Rueppel [26], variants of DSA and ECDSA have been introduced and standardized within the P1363 effort. Further variants, including the Pintsov-Vanstone scheme based on work in [30], are included in Draft 9 of IEEE P1363a [17]. The security of the Pintsov-Vanstone scheme is analyzed in [8]. Another scheme worth mentioning is the Korean KCDSA scheme [22], which is equipped with a security proof.

Among other hard problems that have been exploited for cryptographic purposes, we mention solving multivariate quadratic (MQ) equations over a field and finding small vectors in lattices of large dimensions. Examples of schemes based on the first problem are FLASH, QUARTZ, and SFLASH [27, 28, 29], while NSS [15, 16] is a new scheme based on the second problem. Given that these schemes were introduced most recently, some teething troubles might be expected. Indeed, two early versions of the NSS scheme have been broken [23, 13]. In addition, it seems unlikely that exact security proofs can be provided for any of these schemes or close variants thereof. Namely, they are not designed with such an exact security goal in mind.

2 Notation

Components of the signature schemes

E	Appended string (e.g. bc or cc), third part of y.
f	Verification primitive (e.g. RSA).
f^{-1}	Signature primitive.
g	Function (for mask generation).
H	Hash $h(M)$ of message M.
h	Function (for message hashing).
M	Message to be signed.
N	Modulus, an integer.
r	Random salt.
r^*	First part of y.
w	Output from h, second part of y.
x	Output from signature primitive.
y	Input to signature primitive.
φ, ψ	Simple padding functions.

Scheme parameters

k	Bit length of modulus.
k_E	Bit length of appended string E.
k_h	Bit length of h function output.
k_g	Bit length of g function output.
k_r	Bit length of salt.
K_{sig}	Set of salt lengths accepted by the signer.
K_{ver}	Set of salt lengths accepted by the verifier.
u	Length of padding string in input to h in RSA-PSS2000.

Security model parameters

$c(q,l)$	Collision probability among q uniformly random and independent bit strings of length l.
q_{hash}	Number of h- and g-oracle queries.
q_{sig}	Number of signing queries.
α	"Dependence number" between h and g (see Section 7).
$\lambda_0, \ldots \lambda_{\alpha-1}$	Used in the construction of g when g is defined in terms of h.

Miscellaneous

$\gcd(a,b)$	Greatest common divisor of the two integers a and b.
$\mathrm{lcm}(a,b)$	Least common multiple of the two integers a and b.
$\lfloor x \rfloor$	The real number x rounded down to the closest integer.
$\lceil x \rceil$	The real number x rounded up to the closest integer.
$X \Vert Y$	Concatenation of bit strings X and Y.
$X \oplus Y$	Bitwise exclusive-or of bit strings X and Y.
$\lvert X \rvert$	Bit length of the string X.
$s \overset{R}{\leftarrow} S$	s is chosen uniformly at random from the finite set S.
0^l	String of 0 bits of length l.
$\{0,1\}^l$	The set of bit strings of length l.
$\{0,1\}^L$	The union $\bigcup_{l \in L} \{0,1\}^l$ (L is a set).

3 Overview of the basic construction

In this section we give an overview of the basic construction on which the different proposed variants of RSA-PSS and RW-PSS are based. RSA and RW are described in Section 4; the different PSS variants are specified in Section 5.

We indicated in previous section that the signature schemes to be analyzed in this paper consist of an encoding method and a cryptographic primitive. Formally, a (randomized) encoding method will in this paper consist of an encoding operation and a verification operation. The encoding operation takes a message M, generates a random *salt* r, and outputs a string $y = \mu(M, r)$, where μ is a deterministic function producing "random-looking" outputs from its two inputs. The string y has a verifiable structure in that it is easy to verify whether $y = \mu(M, r)$ for some r once M and y are known. The verification operation hence takes a message M and a string y and checks whether $y = \mu(M, r)$ for some r; put $\mu^{-1}(M, y) = 1$ if this is the case and $\mu^{-1}(M, y) = 0$ otherwise.

In a signature scheme with message recovery, the function μ has the property that part of M can be easily recovered from $\mu(M, r)$. In this case, the verification operation is replaced with a recovery operation, which outputs the recovered message part in case $\mu^{-1}(M, y) = 1$ and "error" otherwise.

The encoding method is combined with a cryptographic primitive to form a signature scheme. Each of the cryptographic primitives employed in this article can be described as a pair (f, f^{-1}) of primitives – a verification primitive f and a signature primitive f^{-1} satisfying $f \circ f^{-1}(y) = y$ for all valid y. Schematically, the verification primitive f takes as input an element (e.g., an integer) x and outputs another element $y = f(x)$, while the signature primitive recovers x from $f(x)$. Anyone who knows the public key can compute $f(x)$, but it is assumed to be hard to compute $f^{-1}(y)$ for a random y unless the private key is known.

The overall signature and verification procedures are schematically defined as follows. To sign a message M, generate a salt r and compute $x = f^{-1}(\mu(M, r))$. To verify that x is a valid signature of M, check that $\mu^{-1}(M, f(x)) = 1$.

4 Cryptographic primitives

4.1 The RSA algorithm

The RSA algorithm is defined in terms of an RSA key pair, which consists of an RSA public key and an RSA private key. Each of these keys contains a *modulus* N, which is an integer of bit length k, where k is the security parameter. N is the product of two[1] randomly generated primes p and q. We do not give any details about how to generate N as the conclusions of this paper do not depend on the key generation method. See [35] for a method recommended by RSA Laboratories.

The public key is the pair (N, e), where $e \geq 3$ is an odd integer such that

[1]More generally, N can be defined to be the product of three or more distinct primes.

$\gcd(e,(p-1)(q-1)) = 1$. The private key is (N,d), where d satisfies

$$de \equiv 1 \pmod{\operatorname{lcm}(p-1,q-1)}.$$

The RSA verification primitive f is defined as

$$f(x) = x^e \bmod N,$$

while the RSA signature primitive f^{-1} is defined as

$$f^{-1}(y) = y^d \bmod N;$$

x and y are integers between 0 and $N-1$. It is easily seen that $f \circ f^{-1}(y) = y$ for all $y \in \mathbb{Z}_N$.

4.2 The RW algorithm

The RW algorithm is defined in terms of an RW key pair. The RW public key consists of a modulus N of bit length k and an even integer $e \geq 2$. Here, $N = pq$, where p and q are primes satisfying $p \equiv 3 \pmod 8$ and $q \equiv 7 \pmod 8$, while e is an even integer satisfying $\gcd(e,\nu) = 1$, where $\nu = \operatorname{lcm}(p-1,q-1)/2$ (note that ν is odd). The RW private key consists of N and an integer d satisfying

$$ed \equiv 1 \pmod{\nu}.$$

Again, we do not give any details about how to generate the key pair. One suggestion is to use the method described in [35] with a few minor modifications to meet the requirements above on p and q.

Let $\left(\frac{a}{N}\right)$ denote the Jacobi symbol of a with respect to N. If N is a product of two distinct primes p and q and $a \in \mathbb{Z}_N^* = \{0 < x < N : \gcd(x,N) = 1\}$, then $\left(\frac{a}{N}\right) = +1$ if either a is a quadratic residue modulo both primes or a is a quadratic nonresidue modulo both primes; a being a quadratic residue modulo an integer j means that the equation $x^2 \equiv a \pmod j$ has an integer solution x. The conditions on p and q make 2 a quadratic residue modulo q but not modulo p, while -1 is a quadratic nonresidue modulo both primes. In particular, if $a \in \mathbb{Z}_N^*$ we obtain

$$\left(\frac{a}{N}\right) = \left(\frac{-a}{N}\right) = -\left(\frac{2a}{N}\right) = -\left(\frac{-2a}{N}\right).$$

Moreover, exactly one of the four numbers $a, -a, 2a, -2a$ is a quadratic residue modulo N.

Any quadratic residue x modulo N satisfies $x^\nu \equiv x \pmod N$. Note that $x^{ed} \equiv x \pmod N$ if and only if x is a quadratic residue (otherwise $x^{ed} \equiv -x \pmod N$).

The signature primitive f^{-1} is defined as follows. The element to be decrypted must be an integer y such that $y \equiv 12 \pmod{16}$. Let z be the unique quadratic residue modulo N among the four integers $y, N-y, y/2 \bmod N, (N-y/2) \bmod N$. We define

$$f^{-1}(y) = \min\{z^d \bmod N, N - \left(z^d \bmod N\right)\}.$$

Note that the Jacobi symbol $\left(\frac{f^{-1}(y)}{N}\right)$ is always $+1$ (but $f^{-1}(y)$ is not necessarily a quadratic residue modulo N).

The verification primitive f is defined as follows. Let x be the integer to be verified. Put

$$w = x^e \bmod N.$$

$f(x)$ is defined to be the unique element y in the set

$$\{w, N - w, 2w \bmod N, (N - 2w) \bmod N\}$$

satisfying $y \equiv 12 \pmod{16}$. If the set contains no such y, then $f(x)$ is not defined. Note that if $x = f^{-1}(y)$ and z is defined as above, then $w = \left(\pm z^d\right)^e \bmod N = z$ (e is even), which implies that $f(f^{-1}(y)) = y$ whenever $y \in \mathbb{Z}_N^*$ and $y \equiv 12 \pmod{16}$.

The choice of 12 as the value of the last nibble l of $f(x)$ is quite arbitrary. In fact, any integer in the set $\{1, 2, 3, 4, 6, 8, 9, 10, 11, 12\}$ would be a fine choice for l, though even numbers are preferable for practical reasons. A proof of this fact is as follows. Suppose that the set

$$S = \{w, N - w, 2w \bmod N, (N - 2w) \bmod N\}$$

contains two elements that are congruent to l modulo 16. Put $v = w \mod 16$ and let $k \in \{0, 1\}$ satisfy $N \equiv 5 + 8k \pmod{16}$. Reducing the integers in S modulo 16, we obtain the four integers

$$\begin{aligned} x_1 &\equiv v \quad ; \quad x_2 \equiv 5 + 8k - v; \\ x_3 &\equiv 2v \text{ or } 2v - 5 + 8k \quad ; \quad x_4 \equiv 5 - 2v + 8k \text{ or } 10 - 2v; \end{aligned}$$

$x_3 \equiv 2v$ if and only if $x_4 \equiv 5 - 2v + 8k$. x_1 and x_2 are never congruent to each other modulo 16 (one is odd and one is even), neither are x_3 and x_4. x_1 and x_3 are congruent only if either $v \equiv 2v$ or $v \equiv 2v - 5 + 8k$, which is true only if $x_1 \in \{0, 5, 13\}$. x_1 and x_4 are congruent only if either $v \equiv 5 - 2v + 8k$ or $v \equiv 10 - 2v$, which is true only if $x_1 \in \{7, 15, 14\}$. By symmetry, the discussion for x_2 is analogous. The conclusion is that if two numbers are congruent modulo 16, then they are congruent to an element in the set $\{0, 5, 7, 13, 14, 15\}$.

In case one required that $y \equiv l$ modulo 8 rather than 16, only $l = 1, 2, 3, 4$ would be fine, which indicates that maybe $2, 4, 10$, and 12 are better choices for l modulo 16 than 6 and 8.

5 Original PSS proposal and revisions

As mentioned in the Introduction, several variants of the PSS encoding method have been proposed. The purpose of this section is to present the two most important variants, the original version in [3] and the adapted version as specified in Draft 9 of IEEE P1363a [18]. To distinguish between the two versions, we will refer to the original version in [3] as PSS96 and to the adapted version in [18] as PSS2000.

PSS96 and PSS2000 are both encoding methods as defined in Section 3, transforming a message to be signed into an encoded message via two functions h and

g. If g and h are viewed as random oracles, then the encoded message, viewed as a random variable, is uniformly distributed among the set of valid encoded messages and is independent of the original message.

5.1 Original proposal

PSS96 consists of an encoding operation PSS96-ENCODE and a verification operation PSS96-VERIFY and is parameterized by two positive integers k_r and k_h satisfying $k_r + k_h \leq k - 1$, where k is the length of the modulus. k_r is the bit length of the *salt*, a random element generated during encoding. PSS96 is based on interactions between two functions, a function

$$h : \{0,1\}^* \to \{0,1\}^{k_h}$$

and a function

$$g : \{0,1\}^{k_h} \to \{0,1\}^{k_g},$$

where $k_g = k - 1 - k_h$. The output from the function g "masks" the salt in the encoded message; for this reason g is often (e.g., in [18]) referred to as a *mask generation function*. We will not adopt this terminology in this paper. Instead, we will refer to both g and h as hash functions. However, whenever we are talking about the hash of a specific string m, we will always mean $h(m)$.

For a finite set S, let $s \overset{R}{\leftarrow} S$ mean that the element s is chosen uniformly at random from S. The PSS96 encoding and verification operations run as follows.[2]

```
PSS96-ENCODE($M$)
    – $r \overset{R}{\leftarrow} \{0,1\}^{k_r}$;
    – $w \leftarrow h(M \| r)$;
    – $r^* \leftarrow g(w) \oplus (0^{k_g - k_r} \| r)$;
    – Return $y = 0 \| r^* \| w$.
```

```
PSS96-VERIFY($M, y$)
    – Write $y = b \| r^* \| w \quad (|b| = 1, |r^*| = k_g$ and $|w| = k_h)$;
    – If $b = 1$, then return 0 and exit.
    – Write $g(w) \oplus r^* = \gamma \| r \quad (|\gamma| = k_g - k_r$ and $|r| = k_r)$;
    – If $h(M \| r) = w$ and $\gamma = 0^{k_g - k_r}$
        then return 1
        else return 0.
```

The full signature operation RSA-PSS96-SIGN takes a message M, computes $y =$ PSS96-ENCODE(M), and returns $x = f^{-1}(y)$, where f denotes the RSA verification (encryption) primitive. The verification operation RSA-PSS96-SIGN takes a message M and a signature x, computes $y = f(x)$, and verifies the result by applying PSS96-VERIFY to (M, y).

[2]A few editorial modifications have been made to facilitate comparisons with the PSS2000 operations introduced in Section 5.2.

5.2 New proposal

The encoding operation PSS2000 is based on PSS96. It is included in the drafts of IEEE P1363a [18] and PKCS #1 v2.1 [34] and was submitted to the NESSIE [35] and the Japan CRYPTREC [36] projects. The new features of PSS2000 compared to PSS96 are as follows:

- Instead of computing the hash of $M\|r$, one computes the hash of M alone and, in the subsequent step, computes the hash of $0^u\|h(M)\|r$, where u is an integer ($u = 64$ in current specifications). This modification is mathematically equivalent to replacing M with $0^u\|h(M)$ as input to PSS96-ENCODE. The string 0^u may seem to make little sense; its rationale is for compatibility with the corresponding encoding operation giving message recovery; see Section 11.

- The encoded message in PSS2000 has either nine or seventeen fixed bits, whereas only the first bit is fixed in the original scheme. In PSS2000, the first bit is 0 and the last eight bits have hexadecimal value *bc* or *cc*; in case the value is *cc*, the next to last eight bits form a hash identifier. The rationale for the extra fixed bits is for compatibility with the Rabin-Williams IFSP-RW signature primitive in IEEE Std 1363-2000 [17] and the corresponding primitive in the draft ISO/IEC 9796-2 [20]; see Section 4.2. This modification is equivalent to replacing the output y with $y\|bc$ and redefining k_g as $k - 1 - k_h - 8$ (which means that g is modified as well, returning strings of length $k - 1 - k_h - 8$ rather than $k - 1 - k_h$).

- There is a minor revision of the padding in r^* preceding the seed r; r^* is defined as $g(w) \oplus (0^{k_g-k_r-1}\|1\|r)$. The 1 delimiter is of importance in the more general RSA-PSS-R scheme giving message recovery; see Section 11. Note that the sender and the receiver do not need to agree on the length of the salt in advance; the length is easily derived from $r^* \oplus g(w)$. This is no loss of security; our security proof will show that it is hard to forge a signature with a salt length that is not supported by the signer. However, in practice it is typically assumed that the receiver does know the length of the salt (if only for compliance with RSA-PSS-R).

We mention that this paper does not consider the role of the hash identifier, neither does it deal with the issue of hash substitution attacks in cases where there is no hash identifier included in the signature.

Let PSS2000 denote the modified version of PSS96 according to the above adaptations; PSS2000 consists of an encoding operation PSS2000-ENCODE and a verification operation PSS2000-VERIFY. We will find it convenient to consider the hash $H = h(M)$ of M rather than M itself as input to the operations. Namely, we will make this decomposition in the security model. Let E be a fixed bit string of length k_E, where k_E is an integer such that $k_r + k_h + k_E \leq k - 1$. In practice, E is either the string $bc = 10111100$ or the string $H_{\mathrm{ID}}\|cc$, where H_{ID} is a hash identifier octet; thus $k_E = 8$ or 16. Let notation be the same as for the RSA-PSS96 scheme, except that k_g is defined to be $k - 1 - k_h - k_E$.

```
PSS2000-ENCODE($H$)
    – $r \overset{R}{\leftarrow} \{0,1\}^{k_r}$;
    – $w \leftarrow h(0^u \| H \| r)$;
    – $r^* \leftarrow g(w) \oplus (0^{k_g - k_r - 1} \| 1 \| r)$;
    – Return $y = 0 \| r^* \| w \| E$.
```

```
PSS2000-VERIFY($H, y$)
    – Write $y = b \| r^* \| w \| E'$   ($|b| = 1$, $|r^*| = k_g$, $|w| = k_h$, and $|E'| = k_E$);
    – If $b = 1$ or $E \neq E'$, then return 0 and exit.
    – Write $g(w) \oplus r^* = \gamma \| r$   ($|\gamma| = k_g - k_r$ and $|r| = k_r$);
    – If $h(0^u \| H \| r) = w$ and $\gamma = 0^{k_g - k_r - 1} \| 1$
        then return 1
        else return 0.
```

The full signature operation RSA-PSS2000-SIGN takes a message M, computes $H = h(M)$ and $y =$ PSS2000-ENCODE(H), and returns $x = f^{-1}(y)$.

6 A generalized scheme

Our goal is to analyze the security of RSA-PSS2000 from different aspects. To make the discussion as general as possible, we introduce an encoding operation GENPSS-ENCODE including PSS2000-ENCODE as a special case. Instead of having a fixed salt length, we allow salt lengths to be picked from a nonempty set K_{sig} of positive integers. Similarly, the verification operation GENPSS-VERIFY accepts salt lengths from a nonempty set K_{ver} (which might be equal to K_{sig}). In this manner we may study the following two cases:

- The forger tries to construct a signature with a salt length not supported by the signer. This corresponds to the case where $K_{\text{ver}} \setminus K_{\text{sig}}$ is nonempty.

- The forger tries to construct a signature with a specific salt length among several salt lengths supported by the signer. This corresponds to the case where $K_{\text{sig}} \setminus K_{\text{ver}}$ is nonempty.

Put $K = K_{\text{sig}} \cup K_{\text{ver}}$.

PSS2000-ENCODE contains two simple padding operations, the operation transforming (H, r) into $0^u \| H \| r$ and the operation transforming r into $0^{k_g - k_r - 1} \| 1 \| r$. To allow for minor adjustments of these operations and to simplify notation, we introduce two injective functions

$$\varphi : \{0,1\}^{k_h} \times \{0,1\}^K \to \{0,1\}^*$$

and

$$\psi : \{0,1\}^K \to \{0,1\}^{k_g}.$$

φ takes inputs of the form (H, r) (a hash value and a salt), and ψ takes inputs of the form r (a salt). We assume that ψ, φ are straightforward to invert, that is, r is easy to extract unambiguously from $\psi(r)$, and similarly for φ.

6.1 Definition of RSA-GenPSS

Let notation be the same as for PSS2000-Encode. The input parameters to GenPSS-Encode are a string H of length k_h and the desired salt length k_0 chosen from the set K_{sig}.

```
GenPSS-Encode($H, k_0$)
    – $r \overset{R}{\leftarrow} \{0,1\}^{k_0}$;
    – $w \leftarrow h(\varphi(H, r))$;
    – $r^* \leftarrow g(w) \oplus \psi(r)$;
    – Return $y = 0 \| r^* \| w \| E$.
```

The input parameters to GenPSS-Verify are a string H of length k_h, a string y of length $k-1$, and a salt length k_0 chosen from the set K_{ver}.

```
GenPSS-Verify($H, y, k_0$)
    – Write $y = b \| r^* \| w \| E'$  ($|b| = 1$, $|r^*| = k_g$, $|w| = k_h$, and $|E'| = k_E$);
    – If $b = 1$ or $E \neq E'$, then return 0 and exit;
    – If possible, write $g(w) \oplus r^* = \psi(r)$ with $|r| = k_0$
          else return 0 and exit;
    – If $h(\varphi(H, r)) = w$
          then return 1
          else return 0.
```

Our object is to define two signature schemes based on the GenPSS encoding method, one ordinary scheme RSA-GenPSS and one "reduced" scheme RSA-GenPSS-Reduced. The ordinary scheme is a straightforward generalization of RSA-PSS2000, whereas the reduced scheme takes the hash value of the message to be signed as input rather than the message itself. The reason for introducing the reduced scheme is that it enables us to give a security proof for RSA-GenPSS in two steps. In the first step we express the security of RSA-GenPSS in terms of RSA-GenPSS-Reduced. In the second step, we translate the security proofs in [3, 10] for RSA-PSS96 into a proof for RSA-GenPSS-Reduced. Combining the two steps, we obtain a proof for the full scheme RSA-GenPSS.

Formally, we define RSA-GenPSS-Reduced($K_{\text{sig}}, K_{\text{ver}}$) as follows. The input parameters to the signature operation RSA-GenPSS-Reduced-Sign are a string H of length k_h and a salt length k_0.

```
RSA-GenPSS-Reduced-Sign($H$,$k_0$)
    – $y \leftarrow$ GenPSS-Encode($H, k_0$);
    – $x \leftarrow f^{-1}(y)$;
    – Return $x$.
```

The input parameters to the verification operation RSA-GenPSS-Reduced-Verify are a string H of length k_h, a signature x, and a salt length k_0.

```
RSA-GenPSS-Reduced-Verify($H$,$x$,$k_0$)
    – $y \leftarrow f(x)$;
    – $b \leftarrow$ GenPSS-Verify($H, y, k_0$);
    – Return $b$.
```

The main signature and verification operations RSA-GenPSS-Sign and RSA-GenPSS-Verify take a string M of arbitrary length as input rather than a string H of length k_h.

```
RSA-GenPSS-Sign($M$,$k_0$)
  – $H \leftarrow h(M)$;
  – $x \leftarrow$ RSA-GenPSS-Reduced-Sign($H, k_0$);
  – Return $x$.
```

```
RSA-GenPSS-Verify($M$,$x$,$k_0$)
  – $H \leftarrow h(M)$;
  – $b \leftarrow$ RSA-GenPSS-Reduced-Verify($H, x, k_0$);
  – Return $b$.
```

6.2 Security models

We want to define security models for RSA-GenPSS and RSA-GenPSS-Reduced. In each of these security models, a *forger* knows the public key and is given access to a signing oracle and two hash oracles simulating g and h. The model has input parameters q_{sig}, q_{hash}, and t. The parameter q_{sig} is the upper bound on the number of queries to the signing oracle, while q_{hash} is the upper bound on the total number of queries to the hash oracles. Finally, t is an upper bound on the complexity (the sum of the time and the description size) of the forger. Note that the model does not consider storage. Yet, we will give storage estimates whenever appropriate.

The goal for the forger is to construct a pair (M, x) such that x is a valid signature of M. Of course, x must not be the response to a signing query with input M; however, it may well be the response to a signing query with a different input, and there might be other signing queries with input M (at least if nonzero salt lengths are allowed). The signature scheme is $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$-secure if there is no forger with complexity t and success probability ϵ making at most q_{sig} signing queries and q_{hash} hash queries.

First, we define a security model for RSA-GenPSS-Reduced. The forger knows the public key and is given access to a signing oracle and two independent random oracles simulating g and h respectively. A random oracle responds to queries $Q_1, Q_2, \ldots$ with responses $R_1, R_2, \ldots$ such that each R_i is uniformly chosen at random from a fixed set and independent from other responses, except that identical queries result in identical responses. To compute $h(x)$, the forger sends x to the h-oracle; the oracle requires that $x = \varphi(H, r)$, where $|H| = k_h$ and $|r| \in K_{\text{sig}} \cup K_{\text{ver}}$. To compute $g(w)$, the forger sends w to the g-oracle; in this case the oracle requires that $|w| = k_h$).

Second, we define a security model for RSA-GenPSS. This model is similar to the model for RSA-GenPSS-Reduced; again we have a signing oracle and two random oracles for g and h. The domain for the g-oracle is the same as before, while the h-oracle accepts inputs of any length. Yet, this time there might be dependencies between the oracles. Namely, in current draft specifications of

RSA-PSS2000, $g(w)$ is defined to be the $k_g = k - 1 - k_h - k_E$ leading bits[3] of the string

$$h(\lambda_0(w)) \| h(\lambda_1(w)) \| h(\lambda_2(w)) \| \ldots . \tag{1}$$

Here, $\lambda_i(w) = w \| (i)_{32}$, where $(i)_{32}$ is the 32-bit low-endian representation of the integer i. In particular, there are interactions between h and g. Such interactions do not occur in the reduced scheme. Namely, in RSA-PSS2000 the length of $\varphi(H, r)$ is equal to $u + k_h + k_r$. Since $u = 64$, it is never the case that $|\varphi(H, r)|$ is equal to $|\lambda_1(w)| = k_h + 32$.

We will assume that if a forger in the RSA-GenPSS scheme outputs a signature (M, x), then the forger has sent all h- and g-oracle queries necessary to verify x. Readers who want to include the case where this is not necessarily true may replace q_{hash} with $q_{\text{hash}} + 3$ in all security bounds.[4]

6.3 Storage considerations

One important component of our security reductions is a random oracle simulator, i.e., an algorithm simulating the random oracles. There is nothing that prevents the forger from sending the same oracle query more than once. In particular, since responses to queries that are identical must be identical as well, the simulator must store the entire history of queries and responses. This might be a problem, because inputs to h in RSA-GenPSS can be arbitrarily long. In particular, the storage requirements for the simulator might be unattainable, whereas the forger might have certain means for reconstructing earlier queries without having to store them.

There are means to address this concern. For example, one may use a deterministic function chosen at random from a large set $\mathcal{H}$ as an instantiation of h for long queries and use a random oracle instantiation of h only for short queries. Given an appropriate assumption on the hardness of finding collisions in a function randomly chosen from $\mathcal{H}$, we obtain a security reduction with reasonable bounds on the storage requirements for the simulator. However, the approach would require a few quite technical definitions. Also, the probability that the forger finds a collision would now depend not only on the number of queries but also on the running time and the storage capabilities of the forger. This would overshadow the clearness of an already quite complex security reduction.

What we may assume instead is that the running time for the forger to have all her queries processed is linear in the total *size* in bits of the queries rather than in the total *number* of queries. In this manner, the storage requirements of the simulator are bounded from above by the running time of the forger. (Authors tend to adopt the convention that the running time to process queries is actually 0, but this convention is inappropriate for our purposes and also unrealistic.)

[3]More precisely, to facilitate byte-oriented implementation, $g(w)$ is defined as the leading k_g bits of the string obtained by first truncating the leading $8 \cdot \lceil (k_g + 1)/8 \rceil - k_g$ bits.

[4]Typically, one assumes that q_{hash} is a very large integer, say between 2^{30} and 2^{60}, so these extra queries will have no practical effect on the security bounds.

7 Analysis of the hash construction

In RSA-PSS2000, the message M is hashed in two steps to give an output w; in the first step M alone is hashed and in the second step $h(M)$ is hashed together with the salt and some padding. In RSA-PSS96 there was only one step. There are reasons to analyze the effect this difference may have on the security, which is roughly equivalent to studying the corresponding difference between RSA-GenPSS and RSA-GenPSS-Reduced.

Let notations be as introduced in the previous section. Let α denote the number of strings y such that $h(y)$ is not independent from $g(w)$ for a given w (in the random oracle model); we assume that α is the same for all w. To avoid unnecessary complications, we assume that if $h(y)$ is not independent from $g(w)$, then the *entirety* of $h(y)$ is straightforward to extract from $g(w)$ (e.g., $h(y)$ is a substring of $g(w)$). If g is defined as in (1), then $\alpha = \lceil (k-1-k_h-k_e)/k_h \rceil$ (if g and h were independent, then α would be 0). As before, let $\lambda_0(w), \ldots, \lambda_{\alpha-1}(w)$ denote the strings such that $h(\lambda_i(w))$ is extractable from $g(w)$. We assume that w is straightforward to extract from $\lambda_i(w)$. This is true in RSA-PSS2000; $\lambda_i(w) = w \| (i)_{32}$.

Lemma 7.1 *Given a forger $\mathcal{F}$ in the security model for* RSA-GenPSS *with at most q_{sig} signing queries and q_{hash} h- and g-oracle queries, we can define a forger $\mathcal{F}_{\text{red}}$ in the security model for* RSA-GenPSS-Reduced *with at most q_{sig} signing queries and $q_{\text{hash}} + q_{\text{sig}}$ h- and g-oracle queries. $\mathcal{F}_{\text{red}}$ is successful if $\mathcal{F}$ is successful, unless there are h-collisions among the values obtained via queries to $\mathcal{F}$'s h-oracle (including those made by $\mathcal{F}$'s signing oracle). The number of such values is at most*

$$q_{\text{tot}} = \max\{\alpha, 1\} \cdot q_{\text{hash}} + (\alpha + 2) \cdot q_{\text{sig}}. \tag{2}$$

Remarks

1. In case q_{hash} is much longer than q_{sig} (which is a natural assumption), the number of h-values is approximately $\max\{\alpha, 1\} \cdot q_{\text{hash}}$.
2. The running time for $\mathcal{F}_{\text{red}}$ is (negligibly larger than) the running time for $\mathcal{F}$ plus the time needed to process q_{sig} hash queries.

Proof Let $\mathcal{F}$ be a forger trying to construct a signature in the security model for RSA-GenPSS. We want to define a forger $\mathcal{F}_{\text{red}}$ in the security model for RSA-GenPSS-Reduced in terms of the forger $\mathcal{F}$. For this purpose, $\mathcal{F}_{\text{red}}$ will have to simulate $\mathcal{F}$'s oracles.

The g-oracle is easy to simulate, because the g-oracles in RSA-GenPSS and RSA-GenPSS-Reduced are identical. The signing oracle is also easy to simulate. When $\mathcal{F}$ sends a query Q to her signing oracle, $\mathcal{F}_{\text{red}}$ simulates $\mathcal{F}$'s h-oracle in the manner described below on the input Q, sends the resulting string H to her own signing oracle, and obtains a signature x.

The h-oracle simulation is a little bit more intricate to describe. Suppose $\mathcal{F}$ sends an h-oracle query Q. There are three possible cases:

1. $Q = \lambda_i(w)$, where $|w| = k_h$ and $i \in \{0, \ldots, \alpha - 1\}$. This means that w is a typical g-oracle query in the reduced model. Hence $\mathcal{F}_{\mathrm{red}}$ sends w to her g-oracle and extracts $H = h(Q)$ from the response.
2. $Q = \varphi(H', r)$, where $|H'| = k_h$ and $|r| \in K_{\mathrm{sig}} \cup K_{\mathrm{ver}}$. This means that Q is a typical h-oracle query in the reduced model. In this case $\mathcal{F}_{\mathrm{red}}$ sends Q to her h-oracle and receives $H = h(Q)$.
3. Q cannot be transformed into a g- or h-oracle query in the reduced model (i.e., cases 1 and 2 do not apply). In this case $\mathcal{F}_{\mathrm{red}}$ generates a random string H, unless Q is an old query, in which which case $H = h(Q)$ is already defined.

$\mathcal{F}_{\mathrm{red}}$ returns the string H to $\mathcal{F}$. Recall that cases 1 and 2 are assumed to have different length queries; hence the three cases are exclusive.

The simulation of the signing oracle may require an extra q_{sig} number of h-oracle queries in RSA-GenPSS, which corresponds to at most q_{sig} additional h- or g-oracle queries in the reduced scheme. In the big scheme, the number of queries from $\mathcal{F}$ to the h-oracle (directly or via the g-oracle) is at most $\max\{\alpha, 1\} \cdot q_{\mathrm{hash}}$. In addition, there are at most $(\alpha + 2) \cdot q_{\mathrm{sig}}$ queries from the signing oracle to the h-oracle. This gives the formula (2).

In the end, $\mathcal{F}$ terminates and outputs (M, x). $\mathcal{F}_{\mathrm{red}}$ outputs (H, x), where $H = h(M)$; by assumption, she knows $h(M)$ (see the discussion at the end of previous section). (H, x) will be a valid signature in the reduced scheme if and only if (M, x) is a valid signature in the full scheme. $\mathcal{F}_{\mathrm{red}}$ will fail if and only if she has asked for the signature of H or if $\mathcal{F}$ fails. The former is possible if and only if $\mathcal{F}$ has asked for the signature of a message M' such that $h(M') = h(M)$. This is possible only if there is an h-collision. □

8 Analysis of the fixed string E (bc)

Recall that the output from PSS2000 always ends with a fixed string. Having parts of the encoded message fixed is problematic from a theoretical point of view. Namely, certain components of the security proofs involve generation of random integers x such that the first bit of $f(x)$ is 0 and $f(x)$ ends with the bit string E; f is the RSA verification primitive. The longer E is, the more applications of f are needed to find an acceptable x by trial and error. The purpose of this section is to address this concern and to estimate the number of applications of f needed to produce a certain number of strings with the appropriate format.

Let Y be a subset of $\mathbb{Z}_N = \{0, 1, \ldots, N - 1\}$. For example, Y might be the set of all integers with a k-bit representation starting with a zero bit and ending with a certain string E; k is the bit length of the modulus N. Let q be a positive integer. We want to establish a bound on the time needed to generate q uniformly random and independent elements from Y via f such that the probability of failure is negligible.

The method used in [3] and [9] is to abort the simulator whenever there are K

consecutive failures (that is, elements x such that $f(x) \notin Y$) for some fixed K. However, this turns out to be far from the best possible approach. Instead of aborting as soon as we have K consecutive failed attempts, we could continue and instead abort when the *total* number of attempts during the *complete* algorithm exceeds a certain much larger number T. An estimate of the probability of at most $q-1$ successes among T attempts can be obtained via normal approximation. However, we will make a simple discrete approach giving, basically, an equivalent bound.

Lemma 8.1 *Put*

$$T = \left\lfloor \frac{N}{|Y|} \cdot \left(\sqrt{k_0 \cdot \ln 2} + \sqrt{q}\right)^2 \right\rfloor . \tag{3}$$

Generate random independent integers $z_1, \ldots, z_T \in \mathbb{Z}_N$. The probability that fewer than q of the integers $f(z_1), \ldots, f(z_T)$ belong to Y is less than 2^{-k_0}.

Remark For typical choices of k_0 and q (e.g., $k_0 = 160$ and $q = 2^{60}$), the $\sqrt{k_0 \cdot \ln 2}$ term is small compared to $\sqrt{q}$, which means that $\frac{|Y|}{N} \cdot T$ is approximately q. In the previous approach described above, we would need, at the very least,

$$\left(1 - \frac{|Y|}{N}\right)^{T/q} \leq 2^{-k_0} \Longrightarrow \frac{|Y|}{N} \cdot T \approx q \cdot k_0 \cdot \ln 2;$$

this is approximately $k_0 \cdot \ln 2$ times as much as the new bound. In practice, k_0 is the output length k_h of the hash function h, which means that when T is the dominant term in an estimate of the complexity of a reduction, the new approach is more than 100 times faster than the previous method.

Proof Let p be an integer strictly between 0 and 1; p will be determined later. Put $\theta = |Y|/N$. The probability that at most $q-1$ among T elements $f(z_1), \ldots, f(z_T)$ belong to Y is

$$\begin{aligned}
\sum_{i=0}^{q-1} \binom{T}{i} \theta^i (1-\theta)^{T-i} &< \sum_{i=0}^{T} \binom{T}{i} \theta^i (1-\theta)^{T-i} p^{i-(q-1)} \\
&= (1-\theta+\theta p)^T \cdot p^{-(q-1)} \\
&= \exp\left(T \ln(1-\theta(1-p)) - (q-1)\ln p\right) \\
&< \exp\left(-T\theta(1-p) + (q-1)(1/p-1)\right).
\end{aligned}$$

The last expression is at most 2^{-k_0} if and only if

$$-T\theta(1-p) + (q-1)(1/p-1) \leq -k_0 \cdot \ln 2 \Longleftrightarrow T \geq \frac{1}{\theta}\left(\frac{k_0 \cdot \ln 2}{1-p} + \frac{q-1}{p}\right).$$

This expression is minimized for $p = \frac{\sqrt{q-1}}{\sqrt{k_0 \cdot \ln 2} + \sqrt{q-1}}$, for which the right-hand side becomes

$$\frac{1}{\theta}\left(\sqrt{k_0 \cdot \ln 2} \cdot (\sqrt{k_0 \cdot \ln 2} + \sqrt{q-1}) + \sqrt{q-1} \cdot (\sqrt{k_0 \cdot \ln 2} + \sqrt{q-1})\right).$$

It is easily seen that T as defined in (3) is at least equal to this quantity (namely, the expression within the floor brackets in (3) exceeds this quantity with at least 1). $\square$

9 Main result

We are now in a position to give a security proof for RSA-GenPSS. The proof is basically the same as Coron's security proof [10] for RSA-PSS96 with some minor adjustments. The goal is to define an inverter of the RSA primitive f in terms of a forger of RSA-GenPSS. Specifically, a $(t(k), \epsilon(k))$-inverter of f is an algorithm with complexity $t(k)$ that is able to compute $f^{-1}(x)$ with probability $\epsilon(k)$, where x is randomly chosen from $\mathbb{Z}_N$ and N is a k-bit modulus generated according to the key generation procedure. We say that f is $(t(k), \epsilon(k))$-hard if there is no $(t(k), \epsilon(k))$-inverter.

Recall that RSA-GenPSS(K_{sig}, K_{ver}) denotes the scheme where K_{sig} is the set of salt lengths that the signing oracle accepts, while K_{ver} is the set of salt lengths that an intended verifier will accept. Thus the forger will have to find a signature with a salt length from the set K_{ver}. Furthermore, recall that k_E denotes the length of the appended bit string E. Put

$$k_r = \min\{k_0 : k_0 \in K_{\text{sig}} \cap K_{\text{ver}}\}.$$

Note that k_r is not defined when the sets K_{sig} and K_{ver} are disjoint; this case will be treated separately.

Let $c(q, l)$ be the probability of a collision among q independent, uniformly chosen bit strings of length l. It is easily seen that

$$c(q, l) \leq \binom{q}{2} 2^{-l}.$$

Let q_{tot} be defined as in (2) in Lemma 7.1. Finally, let $T_f = T_f(k)$ be the time needed to evaluate f with a modulus of length k.

Theorem 9.1 *Suppose that inverting RSA is (t', ϵ')-hard. Then for any q_{sig} and q_{hash} the signature scheme* RSA-GenPSS(K_{sig}, K_{ver}) *is $(t, q_{\text{sig}}, q_{\text{hash}}, \epsilon)$-secure, where*

$$t(k) = t'(k) - \frac{2^{k_E+1}}{1 - q_{\text{tot}} 2^{-k_h}} \cdot \left(\sqrt{k_h \ln 2} + \sqrt{q_{\text{hash}}(k) + 2q_{\text{sig}}(k)} \right)^2 \cdot O(T_f(k))$$

and

$$\epsilon(k) = \frac{1}{\pi(p, \gamma, q_{\text{sig}})} \cdot \epsilon'(k) + c(q_{\text{tot}}, k_h) + 2^{-k_h}. \tag{4}$$

Here, p is any number between 0 and 1,

$$\gamma(q_{\text{hash}} + q_{\text{sig}}, k_r) = \min\{(q_{\text{hash}} + q_{\text{sig}}) 2^{-k_r}, 1\},$$

and

$$\pi(p, \gamma, q_{\text{sig}}) = (1 - p) \left(\frac{p}{\gamma \cdot (1 - p) + p} \right)^{q_{\text{sig}}}.$$

The optimal value for p is

$$p_{\max} = \frac{2q_{\text{sig}}}{q_{\text{sig}} + 1 + \sqrt{(q_{\text{sig}} - 1)^2 + 4q_{\text{sig}}/\gamma}}. \tag{5}$$

If $K_{\text{sig}} \cap K_{\text{ver}} = \phi$, then

$$\epsilon(k) = \epsilon'(k) + c(q_{\text{tot}}, k_h) + 2^{-k_h}.$$

The proof is given in Appendix A. As mentioned in Section 6.1, the proof is divided into two steps with a reduction to RSA-GENPSS-REDUCED in the first step. For large salt values (sufficiently large to make $q_{\mathrm{sig}}\gamma$ significantly smaller than 1), the intuition of the proof is basically the same as in [3] (the factor π in (4) will be very close to 1 in this case). See Section 10 for some estimates of π and $p_{\max}$.

For small salt values, the intuition coincides with that in [9]: Whenever the inverter is to compute $w = h(Q)$ for some $Q = \varphi(H, r)$, he has to choose between being able to invert a value of the form $y = 0\|r^*\|w\|E$ and being able to invert y multiplied with η (the challenge integer for the inverter). In the first case, he will be able to answer a signing query H with salt r. In the second case, he will be able to determine the inverse of η from a forgery of a signature of the message H with the salt r. With a certain probability depending on how he chooses between the two possibilities above, the inverter will fail with either of these tasks. In particular, for small salt values, the factor π in (4) will be significantly smaller than 1. See Section 10 for further discussion.

A few additional remarks:

- For typical values on the parameters, the complicated expression in the boundary of t can be approximated: With 2^{k_h} much larger than q_{tot} and q_{hash} much larger than k_h, we obtain

$$\frac{2^{k_E+1}}{1 - q_{\mathrm{tot}}2^{-k_h}} \cdot \left(\sqrt{k_h \ln 2} + \sqrt{q_{\mathrm{hash}} + 2q_{\mathrm{sig}}}\right)^2 \approx 2^{k_E+1} \cdot (q_{\mathrm{hash}} + 2q_{\mathrm{sig}}) .$$

  The right-hand side is roughly the average number of attempts needed to generate $q_{\mathrm{hash}}+2q_{\mathrm{sig}}$ elements starting with 0 and ending with E according to the method in Lemma 8.1.

- As will be clear from the proof, for each signing or hash oracle query (which takes place in the reduced model), the inverter $\mathcal{I}$ will have to store up to $4k$ bits (k is the size of the modulus). In addition, the inverter will have to store all queries – together with corresponding responses – that do not fit into the reduced model. If no such query has a bit length exceeding $4k - k_h$, then

$$\mathsf{space}(\mathcal{I}) \leq \mathsf{space}(\mathcal{F}) + (q_{\mathrm{hash}} + 2q_{\mathrm{sig}}) \cdot 4k \text{ bits},$$

  where $\mathcal{F}$ is the forger (we ignore minor storage requirements for local variables and for table setup). If however there are longer such queries (or, rather, if the average length of these queries exceeds $4k - k_h$), then this inequality no longer applies; we refer to Section 6.3 for discussion.

10 Discussion

This section contains a very heuristic interpretation of the results in the preceding section. The object is to give a rough estimate of the "gap" between the forger and the inverter in Theorem 9.1.

A reasonable definition of the "bit complexity" of an algorithm $\mathcal{A}$ with running time t and success probability ϵ is

$$C(\mathcal{A}) = \log_2 t - \log_2 \epsilon;$$

we assume that one application of the trapdoor function f consumes one time unit. Let us assume that the running time for one h-oracle query is (at least) 2^{-l} and that $\log q_{\text{hash}} - l$ is considerably larger than $\log q_{\text{sig}}$. Also, assume that the forger in Theorem 9.1 actually sends as many as q_{hash} queries to her hash oracles and that the additional running time for the forger is negligible in terms of the total running time. This means that the running time for the forger is approximately the time $q_{\text{hash}} \cdot 2^{-l}$ needed to make q_{hash} hash queries. Hence the bit complexity of the forger satisfies

$$C(\mathcal{F}) \approx \log q_{\text{hash}} - l - \log \epsilon.$$

The second term in the time bound in Theorem 9.1 is approximately $2^{k_E+1} \cdot q_{\text{hash}}$, which is significantly larger than the running time for the forger. The success probability of the inverter is approximately $\pi(p, \gamma, q_{\text{sig}}) \cdot \epsilon$. Hence the bit complexity of the inverter $\mathcal{I}$ satisfies

$$C(\mathcal{I}) \approx k_E + 1 + \log q_{\text{hash}} - \log \pi - \log \epsilon$$

Hence the difference in bit complexity between the inverter and the forger satisfies

$$C(\mathcal{I}) - C(\mathcal{F}) \approx k_E + 1 + l - \log \pi.$$

(Note that we have ignored storage in this model, which might be inappropriate in many cases.) In practice, $k_E + 1 + l$ is somewhere between 15 and 20 when the hash function is SHA-1 [24] or SHA-256 [25], the modulus length is 1024, and $k_E = 8$; see [38] for performance figures.

It remains to analyze $\log \pi$; a similar discussion can be found in [9]. Put

$$\mu := q_{\text{sig}}\gamma$$

It turns out that the size of μ is crucial for the behavior of p and π; the expression under the square root in (5) can be rewritten as

$$\frac{1}{\gamma^2}\left((\mu - \gamma)^2 + 4\mu\right) = \frac{1}{\gamma^2}\Big((\mu + 2 - \gamma)^2 - 4(1 - \gamma)\Big). \qquad (6)$$

For example, when μ is "large" (much larger than 1), the term $4(1 - \gamma)$ in the right-hand side of (6) is "negligible" compared to $(\mu + 2 - \gamma)^2$. This gives the approximations

$$p \approx \mu/(\mu + 1)$$

and

$$\pi \approx \left(\frac{1}{\mu + 1}\right)\left(\frac{q_{\text{sig}}}{q_{\text{sig}} + 1}\right)^{q_{\text{sig}}} \approx \frac{e^{-1}}{\mu}.$$

The security reduction is roughly equivalent to the reduction in Coron's Full Domain Hash Theorem. We obtain that the gap between the forger and the inverter is approximately

$$k_E + 1 + l + \log e + \log \mu$$

bits. Clearly, if q_{sig} doubles, then $\log \mu$ increases by one. Since the running time for the forger does not necessarily increase more than negligibly when q_{sig} increases, this means that we may lose one bit of tightness in the security proof each time q_{sig} doubles. In particular, if the salt length is short, there might be reasons to put a limit on the maximal number of applications of the signature procedure.

When μ is "small", 4μ is at least $4/\mu$ times larger than $(\mu - \gamma)^2$ in the left-hand side of (6), which gives the approximations

$$p \approx \sqrt{\mu}$$

and

$$\pi \approx (1 - \sqrt{\mu}) \left(\frac{q_{\text{sig}}}{q_{\text{sig}} - \mu + \sqrt{\mu}} \right)^{q_{\text{sig}}} \approx (1 - \sqrt{\mu}) \cdot e^{-\sqrt{\mu}}.$$

In this case the reduction is roughly equivalent to the reduction in the Bellare-Rogaway PSS theorem. When μ is small, note on the one hand that adding an extra bit to the salt does not add more than a negligible amount of security (with respect to this reduction) and on the other hand that extra signing queries will not be particularly gainful for the forger. With a small μ, $\log \pi \approx 0$, so the gap between the forger and the inverter is approximately $k_E + 1 + l$ bits.

As noted, for small salt values the security reduction is not very efficient (though far from disastrously weak). There exist heuristic arguments giving evidence that the crucial parameter π in (4) cannot be significantly improved. However, one must not exclude the possibility that certain properties of the RSA function may allow for improvements in the proof.

11 RSA-PSS with message recovery

We may extend the RSA-PSS scheme to include message recovery. We define the signature scheme RSA-GENPSS-R with input a recoverable message part M_{R} and a nonrecoverable message part M_{NR} as follows. The encoding operation GENPSS-R-ENCODE takes as input a string H of length k_h, the recoverable message part M_{R}, and the desired salt length k_0 chosen from a set K_{sig}. We require that $|M_{\text{R}}| \leq k_{mr} - k_0$, where k_{mr} is a fixed integer smaller than k_g.

```
GENPSS-R-ENCODE($M_{\text{R}}, H, k_0$)
    – $r \overset{R}{\leftarrow} \{0,1\}^{k_0}$;
    – $w \leftarrow h(\varphi(M_{\text{R}}, H, r))$;
    – $r^* \leftarrow g(w) \oplus \psi(M_{\text{R}} \| r)$;
    – Return $y = 0 \| r^* \| w \| E$.
```

Here,

$$\varphi : \{0,1\}^{\leq k_{mr} - k_0} \times \{0,1\}^{k_h} \times \{0,1\}^{K_{\text{sig}} \cup K_{\text{ver}}} \to \{0,1\}^*$$

is a function such that any triple (M_{R}, H, r) is straightforward to extract from $\varphi(M_{\text{R}}, H, r)$. In practice,

$$\varphi(M_{\text{R}}, H, r) = (|M_{\text{R}}|)_{64} \| M_{\text{R}} \| H \| r,$$

where $(i)_{64}$ is the 64-bit low-endian representation of the integer i. ψ is defined as before, except that we allow any input bit length up to k_{mr}. As before, x should be straightforward to extract from $\psi(x)$. In practice,

$$\psi(x) = 0^{k_g - |x| - 1} \| 1 \| x$$

(which implies that $k_{mr} = k_g - 1$ in this case).

Note that while each of the strings M_R and r is extractable from $\varphi(M_\mathrm{R}, H, r)$, only the concatenation of the strings can be determined from $\psi(M_\mathrm{R} \| r)$. Yet, the salt length is an input to the verification operation, which means that the separate strings can be deduced from the concatenation.

The verification procedure recovers M_R from the encoded message as follows; k_0 is an integer from the set K_ver.

```
GenPSS-R-Recover($H, y, k_0$)
  – Write $y = b \| r^* \| w \| E$   ( $|b| = 1$, $|r^*| = k_g$ and $|w| = k_h$);
  – If $b = 1$ or $E \neq E'$, then return "ERROR" and exit;
  – If possible, write $g(w) \oplus r^* = \psi(M_\mathrm{R}, r)$ with $|M_\mathrm{R}| \leq k_{mr} - k_0$
    and $|r| = k_0$
        else return "ERROR" and exit;
  – If $h(\varphi(M_\mathrm{R}, H, r)) = w$
        then return the string $M_\mathrm{R}$
        else return "ERROR".
```

Define the reduced scheme RSA-GenPSS-R-Reduced($K_\mathrm{sig}, K_\mathrm{ver}$) as follows. The signature operation takes as input $H \in \{0,1\}^{k_h}$, $M_\mathrm{R} \in \{0,1\}^{\leq n}$, and a salt length $k_0 \in K_\mathrm{sig}$.

```
RSA-GenPSS-R-Reduced-Sign($M_\mathrm{R}, H, k_0$)
  – $y \leftarrow$ GenPSS-R-Encode($M_\mathrm{R}, H, k_0$);
  – $x \leftarrow f^{-1}(y)$;
  – Return $x$.
```

The recovery operation takes $H \in \{0,1\}^{k_h}$, a signature x, and a salt length $k_0 \in K_\mathrm{ver}$ and proceeds as follows.

```
RSA-GenPSS-R-Reduced-Recover($H$,$x$,$k_0$)
  – $y \leftarrow f(x)$;
  – $M_\mathrm{R} \leftarrow$ GenPSS-R-Recover($H, y, k_0$);
  – If no error occurred
        then return $M_\mathrm{R}$
        else return "ERROR".
```

The main signature and recovery operations RSA-GenPSS-R-Sign and RSA-GenPSS-R-Recover take a string M_NR of arbitrary length as input rather than a string H of length k_h.

```
RSA-GenPSS-R-Sign($M_\mathrm{R}, M_\mathrm{NR}, k_0$)
  – $H \leftarrow h(M_\mathrm{NR})$;
  – $x \leftarrow$ RSA-GenPSS-R-Reduced-Sign($M_\mathrm{R}, H, k_0$);
  – Return $x$.
```

```
RSA-GenPSS-R-Recover($M_{\mathrm{NR}}$,$x$,$k_0$)
    – $H \leftarrow h(M_{\mathrm{NR}})$;
    – $M_{\mathrm{R}} \leftarrow$ RSA-GenPSS-R-Reduced-Recover($H, x, k_0$);
    – If no error occurred
          then return $M_{\mathrm{R}}$
          else return “ERROR”.
```

11.1 Security models for RSA-PSS with message recovery

The security models for RSA-GenPSS-R and RSA-GenPSS-R-Reduced are similar to those for RSA-GenPSS and RSA-GenPSS-Reduced with only a few minor adjustments:

- The goal for the forger in the model for RSA-GenPSS-R is to find a pair (M_{NR}, x) such that x is a valid signature of $(M_{\mathrm{R}}, M_{\mathrm{NR}})$ for some M_{R}.
- The goal for the forger in the model for RSA-GenPSS-R-Reduced is to find a pair (M_{R}, H, x) such that x is a valid signature of (M_{R}, H) for some M_{R}.
- Acceptable h-oracle queries in the reduced scheme are strings x such that $x = \varphi(M_{\mathrm{R}}, H, r)$ for some $|H| = k_h$, $|r| \in K_{\mathrm{sig}} \cup K_{\mathrm{ver}}$, and $|M_{\mathrm{R}}| + |r| \leq k_{mr}$.

As before, we assume that the g-oracle and the h-oracle in the reduced model are independent.

It is straightforward to check that Lemma 7.1 is true with GenPSS replaced with GenPSS-R. It turns out that the security bounds for RSA-GenPSS-R become exactly the same as for RSA-GenPSS:

Theorem 11.1 *Let notation be as in Theorem 9.1. Suppose that inverting RSA is (t', ϵ')-hard. Then for any q_{sig} and q_{hash} the signature scheme* RSA-GenPSS-R(K_{sig}, K_{ver}) *is $(t, q_{\mathrm{sig}}, q_{\mathrm{hash}}, \epsilon)$-secure, where*

$$t(k) = t'(k) - \frac{2^{k_E+1}}{1 - q_{\mathrm{tot}} 2^{-k_h}} \cdot \left(\sqrt{k_h \ln 2} + \sqrt{q_{\mathrm{hash}}(k) + 2q_{\mathrm{sig}}(k)} \right)^2 \cdot O(T_f(k))$$

and

$$\epsilon(k) = \frac{1}{\pi(p, \gamma, q_{\mathrm{sig}})} \cdot \epsilon'(k) + c(q_{\mathrm{tot}}, k_h) + 2^{-k_h}. \tag{7}$$

If $K_{\mathrm{sig}} \cap K_{\mathrm{ver}} = \phi$, then

$$\epsilon(k) = \epsilon'(k) + c(q_{\mathrm{tot}}, k_h) + 2^{-k_h}.$$

Proof. The proof is almost word by word the same as the proof of Theorem 9.1; see the Appendix for notation. The oracle query algorithms are exactly the same, except that H, H_i, and H_j should be replaced with $(M, H), (M_i, H_i)$, and (M_j, H_j) whenever appropriate. For example, we need a set $R(M, H, k_0)$ for each (M, H) and k_0 rather than a set $R(H, k_0)$ for each H and k_0. Also, $\psi(r)$ must of course be replaced with $\psi(M \| r)$ in steps H7 and S8. $\square$

Remark It is important that each of the strings M_{R}, H, and r can be extracted in a unique way from $\varphi(M_{\mathrm{R}}, H, r)$. For example, $\varphi(M_{\mathrm{R}}, H, r) = H\|M_{\mathrm{R}}\|r$ would be a bad choice, because the signature of $(M_{\mathrm{R}}, M_{\mathrm{NR}})$ with salt r would then be a signature of $(M'_{\mathrm{R}}, M_{\mathrm{NR}})$ with salt r' as soon as $M_{\mathrm{R}}\|r = M'_{\mathrm{R}}\|r'$.

12 A Rabin-Williams-based signature scheme

Let the RW signature and verification primitives be defined as in Section 4.2. Let GENPSS be defined as before with the additional condition that E must end with the bit string 1100 (12 written in base 2). Define the schemes RW-GENPSS-REDUCED and RW-GENPSS in the same manner as RSA-GENPSS-REDUCED and RSA-GENPSS but with the RSA primitives replaced with the RW primitives.

Our goal is to prove a theorem similar to Theorem 9.1. In fact, we want to relate the hardness of forging RW-GENPSS signatures with the hardness of factoring the modulus. A k-bit RW modulus factoring algorithm takes as input a k-bit RW modulus (generated with the selected RW key generation algorithm) and tries to find the factors of the modulus. RW modulus factoring is $(t(k), \epsilon(k))$-hard if there is no k-bit RW factoring algorithm with complexity at most $t(k)$ and success probability at least $\epsilon(k)$.

Let $T_{j+f}(k)$ be the time needed to check whether a random integer x satisfies $\left(\frac{x}{N}\right) = +1$ plus the time needed to compute $f(x)$.

Theorem 12.1 *Let notation be as in Theorem 9.1. Suppose that RW modulus factoring is (t', ϵ')-hard. Then for any $q_{\mathrm{sig}}, q_{\mathrm{hash}}$ the signature scheme* RW-GENPSS$(K_{\mathrm{sig}},\ K_{\mathrm{ver}})$ *is $(t, q_{\mathrm{sig}}, q_{\mathrm{hash}}, \epsilon)$-secure, where*

$$t(k) = t'(k) - \frac{2^{k_E - 1}}{1 - q_{\mathrm{tot}} 2^{-k_h}} \cdot \left(\sqrt{k_h \ln 2} + \sqrt{q_{\mathrm{hash}}(k) + 2q_{\mathrm{sig}}(k)}\right)^2 \cdot O(T_{j+f}(k))$$

(note the slight improvement over Theorem 9.1 by a factor of 4) and

$$\epsilon(k) = \frac{1}{\pi(p, \gamma, q_{\mathrm{sig}})} \cdot \epsilon'(k) + c(q_{\mathrm{tot}}, k_h) + 2^{-k_h}.$$

If $K_{\mathrm{sig}} \cap K_{\mathrm{ver}} = \phi$, then

$$\epsilon(k) = \epsilon'(k) + c(q_{\mathrm{tot}}, k_h) + 2^{-k_h}.$$

The proof is given in Appendix B.

Remarks

- Note that the security bounds are the same for all public exponents e, except that the factor $T_{j+f}(k)$ increases when e increases. In particular, from the point of view of this theorem, very little is gained from choosing a public exponent larger than 2. There is a similar conclusion for RSA-GENPSS. However, the underlying RSA problem is different for different public exponents and it is not known whether those problems are computationally equivalent and how they relate to the factoring problem; see [5, 6] for discussion.

- The RW-GenPSS-R signature scheme with message recovery is defined in the obvious way. One easily checks that the security bounds in Theorem 12.1 hold for this scheme as well.

13 Concluding discussion

It is important to note that the security discussions in this paper take place in the random oracle model. In particular, the security proofs cannot be translated into a security proof for a particular scheme with a fixed hash function, e.g. "RSA-PSS with SHA-1." Namely, the bound on the success probability of the forger is the average over "all" hash functions; there might be a certain class of hash functions for which the forger "defeats" the inverter with large probability. It might even be the case that no function with a reasonable description size and running time is a secure instantiation of the hash function; the fraction of such functions is negligible for certain parameter choices.

Thus far, all attempts to find security proofs for RSA-PSS and RW-PSS in a "standard" model, as opposed to the random oracle model, have been in vain. This might be due to the current method of translating a chosen message attack into an RSA inversion algorithm. As noted, this translation requires that the inverter simulate the signature operation. With a fixed hash function, the only thing the inverter can do is to generate a random salt, produce an encoded message via PSS and try to invert the result under RSA. There is no *canonical* way for the inverter to do this unless he actually knows how to compute the inverse of arbitrary integers (which is what we want to prove!). With random hash outputs, the inverter may produce a valid signature by applying the RSA verification primitive to a value x and then define the hash function on relevant strings such that the encoded message conforms with the RSA verification of x.

As mentioned in Section 1.1, there exist a few schemes [37, 12] with provable security in a standard model as well as in a random oracle model. Yet, those schemes are less efficient than RSA-PSS. Also, the security reductions for those schemes are no stronger than our reduction for RSA-PSS with zero salt length.

Acknowledgment

I thank Burt Kaliski for many valuable comments and suggestions.

References

[1] ANSI X9.30:1-1997, Public Key Cryptography for the Financial Services Industry: Part 1: The Digital Signature Algorithm (DSA) (revision of X9.30:1-1995).

[2] ANSI X9.62-1998, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).

[3] M. Bellare and P. Rogaway. The Exact Security of Digital Signatures – How to Sign with RSA and Rabin. In *Advances in Cryptology – Eurocrypt '96*, pp. 399–416. Springer Verlag, 1996.

[4] M. Bellare and P. Rogaway. *PSS: Provably Secure Encoding Method for Digital Signatures*. Submission to IEEE P1363 working group, August 1998. Available from
http://grouper.ieee.org/groups/1363/P1363a/submissions.html.

[5] D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2), 1999.

[6] D. Boneh and R. Venkatesan. Breaking RSA may not be equivalent to factoring. In *Advances in Cryptology - Eurocrypt '98*, pp. 59–71. Springer Verlag, 1998.

[7] D.R.L. Brown. *The Exact Security of ECDSA*. Technical Report 34:2000, Centre for Applied Cryptographic Research, University of Waterloo, 2000. Available from
http://cacr.math.uwaterloo.ca/techreports/2000/tech_reports2000.html.

[8] D.R.L. Brown and D.B. Johnson. *Formal Security Proofs for a Signature Scheme with Partial Message Recovery*. Technical Report 24:2000, Centre for Applied Cryptographic Research, University of Waterloo, 2000. Available from
http://cacr.math.uwaterloo.ca/techreports/2000/tech_reports2000.html.

[9] J.-S. Coron. On the Exact Security of Full Domain Hashing. In *Advances in Cryptology – Crypto 2000*, pp. 229–235. Springer Verlag, 2000.

[10] J.-S. Coron. *Security of PSS for Short Random Size*. Manuscript, July 2000.

[11] E. Fujisaki, T. Kobayashi, and H. Morita. *ESIGN: Efficient Digital Signature Scheme*. Submission to the NESSIE project, September 2000. Available from
www.cosic.esat.kuleuven.ac.be/nessie/workshop/submissions.html.

[12] R. Gennaro, S. Halevi, and T. Rabin. Secure Hash-and-Sign Signatures without the Random Oracle. In *Advances in Cryptology – Eurocrypt '99*, pp. 123–139. Springer-Verlag, 1999.

[13] C. Gentry, J. Jonsson, J. Stern and M. Szydlo. Cryptanalysis of the NTRU Signature Scheme (NSS) from Eurocrypt 2001. Presentation at the Eurocrypt 2001 Rump Session.

[14] L. C. Guillou and J.-J. Quisquater. *Technical Report on Dynamic Authentication Comparison of RSA, GQ1, and GQ2*. Manuscript, 2001.

[15] J. Hoffstein, J. Pipher, J. H. Silverman. *NSS: The NTRU Signature Scheme*. Preprint, November 2000. Available at
www.ntru.com/technology/tech.technical.htm

[16] J. Hoffstein, J. Pipher, J. H. Silverman. *Enhanced Encoding and Verification Methods for the NTRU Signature Scheme.* NTRU Cryptosystems Technical Report #017. Available at www.ntru.com/technology/tech.technical.htm

[17] *IEEE Std 1363-2000: Standard Specifications for Public Key Cryptography.* IEEE, August 2000.

[18] IEEE P1363 working group. *IEEE P1363a D9: Standard Specifications for Public Key Cryptography: Additional Techniques.* April 2, 2001. Available from grouper.ieee.org/groups/1363/P1363a/private/.

[19] *ISO/IEC 9796:1991 Information Technology – Security techniques – Digital signature scheme giving message recovery.*

[20] *Revision of ISO/IEC CD 9796-2 Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms.* Working draft, May 29, 2001.

[21] B. S. Kaliski Jr. RSA Digital Signatures. *Dr. Dobb's Journal*, May 2001 – Communications & Networking.

[22] KCDSA Task Force Team. *KCDSA : The Korean Certificate-based Digital Signature Algorithm.* Contribution to IEEE P1363a, August 1998. Available at http://grouper.ieee.org/groups/1363/P1363a/PSSigs.html#KCDSA.

[23] I. Mironov. A Note on Cryptanalysis of the Preliminary Version of the NTRU Signature Scheme. Available from eprint.iacr.org/2001/005/.

[24] National Institute for Standards and Technology (NIST). *FIPS 180-1: Secure Hash Standard.* April 1994.

[25] National Institute for Standards and Technology (NIST). *Draft FIPS 180-2: Secure Hash Standard.* May 30, 2001. Available from www.nist.gov/sha/.

[26] K. Nyberg and R. Rueppel. A New Signature Scheme based on the DSA Giving Message Recovery. *First ACM Conference on Computer and Communcations Security*, pp. 58-61. ACM Press, 1993.

[27] J. Patarin, N. Courtois, L. Goubin. *FLASH, a Fast Asymmetric Signature Scheme for Low-Cost Smartcards.* Submission to the NESSIE project, September 2000. Available from www.cosic.esat.kuleuven.ac.be/nessie/workshop/submissions.html.

[28] J. Patarin, N. Courtois, L. Goubin. *QUARTZ, an Asymmetric Signature Scheme for Short Signatures on PC.* Submission to the NESSIE project, September 2000. Available from www.cosic.esat.kuleuven.ac.be/nessie/workshop/submissions.html.

[29] J. Patarin, N. Courtois, L. Goubin. *SFLASH, a Fast Asymmetric Signature Scheme for Low-Cost Smartcards.* Submission to the NESSIE project, September 2000. Available from www.cosic.esat.kuleuven.ac.be/nessie/workshop/submissions.html.

[30] L. Pintsov and S. Vanstone. *Postal Revenue Collection in the Digital Age.* Technical Report 26:2000, Centre for Applied Cryptographic Research, University of Waterloo, 2000. Available from http://cacr.math.uwaterloo.ca/techreports/2000/tech_reports2000.html.

[31] M. O. Rabin. *Digitalized Signatures and Public-Key Functions as Intractable as Factorization.* MIT Laboratory for Computer Science Technical Report 212 (MIT/LCS/TR-212), 1979.

[32] R. Rivest, A. Shamir and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2), pp. 120-126, February 1978.

[33] RSA Laboratories. *PKCS #1 v1.5: RSA Encryption Standard.* November 1, 1993. Available from www.rsalabs.com/pkcs/pkcs-1/.

[34] RSA Laboratories. *PKCS #1 v2.1 d2: RSA Encryption Standard.* Draft 2. January 5, 2001. Available from www.rsalabs.com/pkcs/pkcs-1/.

[35] RSA Laboratories. *RSA-PSS Signature Scheme with Appendix.* Submission to the NESSIE project, September 2000. Available from www.rsalabs.com/submissions/.

[36] RSA Laboratories. *RSA-PSS Signature Scheme with Appendix.* Submission to the IPA CRYPTREC project in Japan, September 2000.

[37] T. Schweinberger and V. Shoup. *ACE: The Advanced Cryptographic Engine.* Manuscript, August 2000. Available from http://shoup.net/papers/.

[38] Wei Dai. *Crypto++ 4.0 Benchmarks.* Available at http://www.eskimo.com/~weidai/benchmarks.html.

[39] H. C. Williams. A Modification on the RSA Public-Key Encryption Procedure. *IEEE Transactions of Information Theory*, 26, pp. 726–729, 1980.

A Proof of Theorem 9.1

Let $\mathcal{F}$ be a forger of RSA-GenPSS with a success probability ϵ. By Lemma 7.1, there is a forger $\mathcal{F}_{\text{red}}$ of RSA-GenPSS-Reduced defined in terms of $\mathcal{F}$ with at most q_{sig} signing queries and $q_{\text{sig}} + q_{\text{hash}}$ h- and g-oracle queries. $\mathcal{F}_{\text{red}}$ will be successful if $\mathcal{F}$ is successful, unless there are h-collisions. The probability of an h-collision is at most $c(q_{\text{tot}}, k_h)$. The probability of an h-collision in the reduced scheme is $c(q_{\text{red}}, k_h)$, where $q_{\text{red}} \leq q_{\text{hash}} + 2q_{\text{sig}}$ is the actual number of h-oracle queries in the reduced model made by $\mathcal{F}_{\text{red}}$ or her signing oracle. Hence with probability at least

$$\epsilon'' = \frac{\epsilon - c(q_{\text{tot}}, k_h)}{1 - c(q_{\text{red}}, k_h)} \qquad (8)$$

$\mathcal{F}_{\text{red}}$ will be successful *conditioned* that there are no h-collisions in the reduced scheme.[5] Consider a model where the h-oracle is modified such that it never

[5]Recall that g and h are independent in the reduced scheme; hence we may ignore h-collisions induced by g-oracle queries.

responds with the same string to different queries. More precisely, each response is chosen uniformly at random from the set of strings of length k_h that have not been responses to earlier h-oracle queries.

We want to define an inverter I in terms of the forger $\mathcal{F}_{\text{red}}$. The inverter maintains a counter i initially set to 0 and a set $R(H, k_0)$ initialized to $\{\}$ for each H such that $|H| = k_h$ and $k_0 \in K_{\text{sig}}$ (of course, during the algorithm the inverter will only store nonempty sets $R(H, k_0)$). The goal for the inverter is to compute the eth root modulo N of an integer η chosen uniformly at random from $\mathbb{Z}_N^*$.

For each signing query and h-query, we store values H_i (the message hash), r_i (the salt), $w_i = h(\varphi(H_i, r_i))$, $g(w_i)$, and x_i (the RSA signature primitive output). For h-queries, we also store a bit b_i and add r_i to the set $R(H_i, |r_i|)$. This requires up to

$$(|H_i| + |r_i| + |w_i| + |g(w_i)| + |x_i|) + (1 + |H_i| + |r_i|) \leq k + 3k_h + 3k_g + 1 < 4k$$

bits of memory. For each g-query, we store values $w_i, g(w_i)$, which occupy less than $4k$ bits of memory.

Answering h-oracle queries
Input: A string Q

H1 Increment i. If Q is valid, Q is of the form $\varphi(H_i, r_i)$, where $|H_i| = k_h$ and $|r_i| = k_0 \in K_{\text{sig}} \cup K_{\text{ver}}$. Otherwise, return "error."

H2 If $(H_i, r_i) = (H_j, r_j)$ for some $j < i$, then put $w_i = w_j$ and go to step 8.

H3 Set $R(H_i, k_0) \leftarrow R(H_i, k_0) \cup \{r_i\}$.

H4 Let $b_i = 0$ with probability p_{k_0} and $b_i = 1$ with probability $1 - p_{k_0}$; p_{k_0} will be determined later.

H5 Repeat $x_i \overset{R}{\leftarrow} \mathbb{Z}_N$; $y_i \leftarrow \eta^{b_i} \cdot f(x_i)$ until y_i can be written as $0 \| r_i^* \| w_i \| E$ ($|r_i^*| = k_g$ and $|w_i| = k_h$) and there is no $j < i$ such that $w_j = w_i$.

H6 Define $h(\varphi(H_i, r_i)) = w_i$.

H7 Put $g(w_i) = r_i^* \oplus \psi(r_i)$.

H8 Return w_i.

Answering g-oracle queries
Input: A string Q

G1 Increment i and put $w_i = Q$. If $|w_i| \neq k_h$, then return "error."

G2 If $w_i = w_j$ for some $j < i$, then return $g(w_j)$, otherwise return $g(w_i) \overset{R}{\leftarrow} \{0,1\}^{k_g}$.

Answering signing queries
Input: A string Q and an integer k_0

S1 Increment i and put $H_i = Q$. If $|H_i| \neq k_h$, then return "error." If $k_0 \notin K_{\text{sig}}$, then return "error."

S2 Let $c_i = 0$ with probability $\kappa(|R(H_i, k_0)| \cdot 2^{-l})$ and $c_i = 1$ otherwise; κ will be determined later.

S3 If $c_i = 0$, pick $r_i \overset{R}{\leftarrow} R(H_i, k_0)$ and proceed to step 4. If $c_i = 1$, pick $r_i \overset{R}{\leftarrow} \{0,1\}^{k_0} \setminus R(H_i, k_0)$ and jump to step 5.

S4 There is a $j < i$ such that $(H_j, r_j) = (H_i, r_i)$ and the jth query was an h-oracle query. If $b_j = 0$, then put $x_i = x_j$ and go to step 9, else abort.

S5 If $(H_j, r_j) = (H_i, r_i)$ for some $j < i$, then the jth query was an identical signing query; put $x_i = x_j$ and go to step 9. Otherwise, proceed to next step.

S6 Repeat $x_i \overset{R}{\leftarrow} \mathbb{Z}_N$; $y_i \leftarrow f(x_i)$ until y_i can be written as $0\|r_i^*\|w_i\|E$ ($|r_i^*| = k_g$ and $|w_i| = k_h$) and there is no $j < i$ such that $w_j = w_i$.

S7 Define $h(\varphi(H_i, r_i)) = w_i$.

S8 Put $g(w_i) = r_i^* \oplus \psi(r_i)$.

S9 Return x_i.

Analysis The rationale for the coin flip of b_i in step H4 (which is a clever trick due to Coron [9]) is as follows. With some probability, the inverter will be forced to sign the message H_i with salt r_i, and in this case he must know the inverse of the string y_i generated in step H5, which he will if $b_i = 0$. With some other probability, the forger will output the inverse z_i of y_i generated in step H5 as the forged signature, and in this case the inverter will be able to determine the inverse of η if $b_i = 1$;

$$z_i = f^{-1}(y_i) = x_i f^{-1}(\eta) \bmod N.$$

Since the probability that the inverter aborts might be significantly larger than ϵ'' as defined in (8) for small salt lengths, we have to express ϵ' as a product of ϵ'' and a parameter rather than ϵ'' minus a parameter. This has the following implication: Under the condition that the inverter does not abort in step S4, all salts generated in the signing query algorithm must be *independent and uniformly distributed*. However, this is not achieved if the salt is uniformly generated in step S2; the probability that we do not abort in case $r_i \in R(H_i, k_0)$ is p_{k_0}, while we never abort in case $r_i \notin R(H_i, k_0)$, because step S4 is not used.

For this reason, the probability in step S2 that a certain element from $R(H_i, k_0)$ is chosen must be $1/p_{k_0}$ times larger than the probability that a certain element outside $R(H_i, k_0)$ is chosen. This means that κ has the property that

$$\frac{\kappa(\sigma)}{\sigma 2^{k_0}} = \frac{1 - \kappa(\sigma)}{p_{k_0}(1-\sigma)2^{k_0}} \iff \kappa(\sigma) = \frac{\sigma}{p_{k_0}(1-\sigma) + \sigma}.$$

With $\sigma = |R(H_i, k_0)|2^{-k_0}$, the probability that the inverter will not abort in step S4 is

$$\kappa(\sigma)p_{k_0} + (1 - \kappa(\sigma)) = \frac{p_{k_0}}{(1 - p_{k_0})\sigma + p_{k_0}}.$$

When $k_0 \notin K_{\text{ver}}$, we can put $p_{k_0} = 1$, because the forger is not allowed to output a signature with salt length k_0. If $k_0 \in K_{\text{ver}}$, then

$$\sigma \leq \min\{(q_{\text{hash}} + q_{\text{sig}})2^{-k_0}, 1\} \leq \gamma.$$

For $k_0 \in K_{\text{ver}}$, define $p_{k_0} = p_{k_r} = p$ for some $p \in (0, 1)$; the optimal value for p is given in (5). Note that

$$\frac{p_{k_0}}{(1 - p_{k_0})\sigma + p_{k_0}} \geq \frac{p}{(1 - p)\gamma + p}.$$

for all $k_0 \in K_{\text{sig}} \cup K_{\text{ver}}$.

In the end, $\mathcal{F}_{\text{red}}$ returns a forgery (H, x). $\mathcal{F}_{\text{red}}$ is defined in terms of $\mathcal{F}$, which means that $\mathcal{F}$ returns a forgery (M, x), where $H = h(M)$. If the forgery is valid, then we can write $y = f(x)$, $y = 0\|r^*\|w\|E$, and $r = \psi^{-1}(g(w) \oplus r^*)$; $|w| = k_h$. By assumption, $h(\varphi(H, r))$ is known, which means that $(H, r) = (H_i, r_i)$ for some i. If (H_i, r_i) was part of a signing query of some message M', then (H, x) is not a valid forgery for $\mathcal{F}_{\text{red}}$. If $\varphi(H_i, r_i)$ was an h-oracle query, then $b_i = 1$ with probability $1-p$, in which case the inverter will be able to compute $f^{-1}(\eta)$. Hence the probability of success for the inverter *conditioned* that there are no h-collisions in the reduced model is at least

$$(1 - p)\left(\frac{p}{(1 - p)\gamma + p}\right)^{q_{\text{sig}}} \cdot \epsilon'' = \pi(p, \gamma, q_{\text{sig}}) \cdot \epsilon''.$$

The effect of abortion in S4 has been considered. Yet, there is still a case where the inverter will have to abort – the loops in steps H5 and S6 cannot go on forever. Let the set Y in Lemma 8.1 be the set of strings starting with a 0, ending with the string E, and having a string w preceding E that is not equal to w_j for any $j < i$ (this means that Y becomes smaller during the algorithm, but this does not have any impact on the conclusions in Lemma 8.1). The probability that a random integer from $\mathbb{Z}_N$ belongs to Y is at least $2^{-k_E-1} \cdot \left(1 - q_{\text{tot}}2^{-k_h}\right)$. Let us allow the inverter to repeat the procedure in steps H5 and S6

$$\frac{2^{k_E+1}}{1 - q_{\text{tot}}2^{-k_h}} \cdot \left(\sqrt{k_h \ln 2} + \sqrt{q_{\text{hash}} + 2q_{\text{sig}}}\right)^2$$

times during the entire algorithm. By Lemma 8.1, the probability that the inverter will have to abort is at most 2^{-k_h} (recall that the forger in the reduced scheme is allowed to make $(q_{\text{hash}} + q_{\text{sig}})$ h-queries).

To summarize, the *unconditional* probability that the inverter will be successful is at least

$$\begin{aligned} & (1 - c(q_{\text{red}}, k_h)) \cdot \pi(p, \gamma, q_{\text{sig}}) \cdot \left(\epsilon'' - 2^{-k_h}\right) \\ \geq\ & \pi(p, \gamma, q_{\text{sig}}) \cdot \left(\epsilon - c(q_{\text{tot}}, k_h) - 2^{-k_h}\right) \end{aligned}$$

(2^{-k_h} can be placed within the parentheses, because the probability of a failure in step H5 or step S6 is at most 2^{k_h} regardless of what the b_i values are). This expression attains its maximum for p as defined in (5). See Section 10 some approximations of p.

Finally, note that if $K_{\text{sig}} \cap K_{\text{ver}} = \phi$, then we can define $p_{k_r} = 1$ if $k_r \in K_{\text{sig}}$ and $p_{k_r} = 0$ if $k_r \in K_{\text{ver}}$, which gives $\pi(p, \gamma, q_{\text{sig}}) = 1$. □

B Proof of Theorem 12.1

Put $\eta = 2^e$. Define the algorithms for answering oracle queries in exactly the same manner as in the proof of Theorem 9.1 in the preceding section, with the following exceptions:

- In steps H5 and S6, we require that $\left(\frac{x_i}{N}\right) = +1$ and $x_i < N/2$. If we generate an x_i such that $\left(\frac{x_i}{N}\right) = -1$, replace x_i with $\min\{2x_i \bmod N, N - 2x_i \bmod N\}$ (the new x_i value will be uniformly distributed among all $x < N/2$ satisfying $\left(\frac{x}{N}\right) = +1$). An alternative method is to pick a random $\hat{x}$ and define $x_i = \min\{\hat{x}^2 \bmod N, N - \hat{x}^2 \bmod N\}$. This method is of course not relevant in case the running time for squaring modulo N exceeds the running time for computing a Jacobi symbol modulo N.
- In case $b_i = 1$ in step H5, define $y_i = f(2x_i \bmod N)$. Note that $y_i = f(\omega_i)$ for some ω_i satisfying
$$\omega_i^e \equiv (2x_i)^e \pmod N$$
and $\left(\frac{\omega_i}{N}\right) = \left(\frac{x_i}{N}\right) = +1$.

The reason why we want $\left(\frac{x_i}{N}\right) = +1$ is two-fold:

1. We obtain that
$$f^{-1}(f(x_i)) = \min\{x_i, N - x_i\}$$
unless $b_i = 1$ in step H5. In particular, we will be able to answer signing queries in step S4 as soon as $b_j = 0$.
2. Suppose the forger outputs (M, x) with $x = f^{-1}(y_i)$ for some i such that $b_i = 1$. Then we know that
$$x^e \equiv (2x_i)^e \pmod N.$$
Since $\left(\frac{x}{N}\right) = +1$ and $\left(\frac{2x_i}{N}\right) = \left(\frac{2}{N}\right) = -1$, it is not the case that $x \equiv \pm 2x_i \pmod N$. Hence $\gcd(x - 2x_i, N)$ is equal to one of the factors of N.

The proof becomes exactly the same as the proof for Theorem 9.1, except that we find a factor of N rather than the inverse of an integer η.

There is a slight improvement in the time bound by a factor of 4 compared to Theorem 9.1. For each generated x_i in steps H5 and S6, the probability that x_i is fine is four times higher than in the RSA case. Namely, for each x_i, there are four mutually exclusive possibilities $w, N - w, 2w \bmod N, (N - 2w) \bmod N$ for y_i, and each of the possibilities is fine with probability at least $2^{-k_E - 1}$. □