Submission to IEEE P1363

PSS: Provably Secure Encoding Method for Digital Signatures

Mihir Bellare* Phillip Rogaway†

August 1998

Abstract

We describe two encoding methods: EMSA-PSS, for signing with appendix, and EMSR-PSS, for signing with message recovery. These encodings are appropriate for signatures based on the RSA or Rabin/Williams primitive. The methods are as simple and efficient as the methods in the current P1363 draft (based on X9.31 and ISO 9796), but they have better demonstrated security. In particular, treating the underlying hash function as ideal, EMSA-PSS and EMSR-PSS give rise to provably-secure schemes: the ability to forge implies the ability to invert the underlying trapdoor permutation. In fact, when the underlying primitive is RSA, the schemes are not only provably secure, but are so in a *tight* way: the ability to forge with a certain amount of computational resources implies the ability to invert RSA (on the same size modulus) with essentially the *same* computational resources. Additional benefits are described in the body of this paper.

The methods described in this contribution are from our *Eurocrypt '96* paper, *The exact security of digital signatures— How to sign with RSA and Rabin* [3].

*Department of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093. E-Mail: mihir@cs.ucsd.edu. URL: http://www-cse.ucsd.edu/users/mihir. Supported by NSF CAREER Award CCR-9624439 and a 1996 Packard Foundation Fellowship in Science and Engineering.

† Department of Computer Science, University of California at Davis, Davis, California 95616. E-mail: rogaway@cs.ucdavis.edu. URL: http://www.cs.ucdavis.edu/~rogaway. Supported by NSF CAREER Award CCR-9624560, and RSA Data Security – MICRO Grant 97-150.

Contents

1 Description of the Encoding Methods

1.1 Overview

To compute the signature of a message M with a trapdoor bijection such as the RSA primitive, prevailing practice is to first *encode* the message M into a *message representative*, f, and then apply the trapdoor-requiring direction of the function to arrive at the signature. For example, to sign M with RSA one maps M to a number f, where $f < n$, with n the RSA modulus, and then the signature of M is $s = f^d \bmod n$, where d is the RSA decryption exponent.

This contribution describes a way to do the message encoding step of the process outlined above, that is, the $M \mapsto f$ mapping. The encoding method we describe is from our Eurocrypt '96 paper, *The exact security of digital signatures — How to sign with RSA and Rabin* [3]. Our encoding method is simple and efficient, yet has proven security properties.

Actually we give two message encoding methods. The first, EMSA-PSS, does not embed the message M into the message representative f. This encoding method is used for signature schemes with appendix: the message M needs to be transmitted along with its signature s. The second encoding method, EMSR-PSS, embeds as much of the message M into the message representative f as will fit. The remaining portion of the message, if any, will have to be transmitted along with s. This is for signature schemes permitting message recovery: the message M is recovered from the signature s and the remaining portion of the message, if any, which couldn't be packaged into the signature.

Here we document the methods. For the materiel providing and supporting the technical security claims, see [3].

1.2 Encoding Method for Signatures with Appendix: EMSA-PSS

EMSA-PSS is an encoding method for signatures with appendix based on a hash function. It is intended for use with IFSA (integer factorization signature schemes with appendix). The encoding method is parameterized by a mask generation function, G. Recall that a mask generation function returns as many octets as requested. For $b \geq 0$ a number, we indicate by $G(x)[b]$ that we request b octets to be generated by applying the mask generation function G to x.

The message encoding process includes the selection of a random seed *seed* of *seedLen* octets, and the application of a mask generation function G, producing a string w of *wLen* octets. The values *seedLen* and *wLen* could either be fixed constants, parameters associated to the underlying key, or inputs to the encoding and decoding operations. Below, we take them as constants: *seedLen* = *wLen* = 20 octets. We discuss in Section 1.2.3 how to adapt the method to allow *seedLen* and/or *wLen* to be inputs to the encoding and decoding operations. We also discuss in Section 1.2.3 changing slightly the arguments to the mask generation function G.

1.2.1 Encoding Operation

Input:
— the octet string M, the message
— the number l, which is the maximum bit length of the output.

Output: a message representative, which is an integer f of bit length at most l; or "error"

The message representative f is computed by the following or an equivalent sequence of steps. (Also see Figure 1.)

1. If the length of M in octets, $mLen$, is greater than the length limitation (eg., $mLen \geq 2^{61}-21$), or $l < 320$, then return "error". (More generally, the shortest possible message representative is $wLen + seedLen$ octets.)
2. Let $oLen = \lceil l/8 \rceil$ denote the maximal length of the message representative in octets. (Note $oLen \geq 40$ octets by Step 1.)
3. Generate a fresh, random octet string, $seed$, having 20 octets.
4. Let $w = G(seed \parallel M)[20]$. (Recall that this means to use the mask generation function to make 20 octets.)
5. Let $expandedW = G(w)[oLen - 20]$. (The length of $expandedW$ is at least 20 octets.)
6. Let $seedMask = expandedW[1..20]$ be the first 20 octets of $expandedW$.
7. Let $remainMask = expandedW[21..oLen-20]$ be the remaining $oLen-40$ octets of $expandedW$.
8. Let $maskedSeed = seed \oplus seedMask$.
9. Let $T = w \parallel maskedSeed \parallel remainMask$.
10. Convert the octet string T to the binary number f which it represents using BS2IP.
11. Return f.

1.2.2 Decoding Operation

Input:
— the octet string M, the message
— the number l, which is the maximum bit length of the output
— the message representative, which is an integer $f \geq 0$

Output: "valid" if f is a correct message representative for M; "invalid" otherwise

The validity indicator shall be computed by the following or an equivalent sequence of steps.

1. If the length of M in octets, $mLen$, is greater than the length limitation (eg., $mLen \geq 2^{61}-21$), or $l < 320$, then return "invalid".
2. Let $oLen = \lceil l/8 \rceil$ denote the maximal length of the message representative in octets.
3. Convert f into the octet string T of length $oLen$ which represents it using I2OSP.
4. Let $w = T[1..20]$ be the first 20 octets of T.
5. Let $maskedSeed = T[21..40]$ be the next 20 octets of T.
6. Let $remainMask = T[41..oLen]$ be the remaining $oLen$ octets of T.
7. Let $expandedW = G(w)[oLen - 20]$.
8. Let $seedMask = expandedW[1..20]$.
9. Let $remainMask2 = expandedW[21..oLen - 20]$.
10. Let $seed = maskedSeed \oplus seedMask$.
11. If $G(seed \parallel M)[20] \neq w$, or $remainMask2 \neq remainMask$, then return "invalid". Otherwise return "valid".

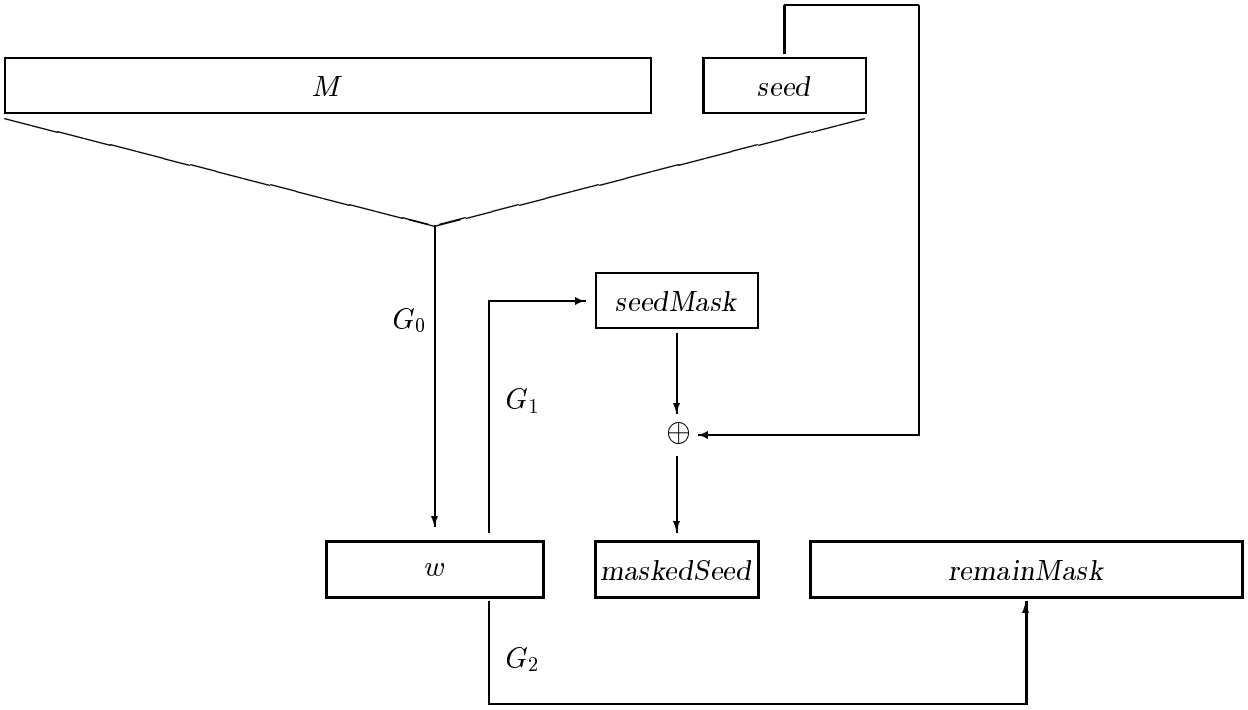

Figure 1: EMSA-PSS. Computing the message representative, the pieces of which (prior to integer conversion) are darkened. Here $G_0(M, seed) = G(seed \parallel M)[20]$, $G_1(x)$ denotes the first 20 octets of $G(x)[l-20]$, and $G_2(x)$ denotes the remaining octets. Different ways to instantiate G_0, G_1 and G_2 are discussed in the body of the paper.

1.2.3 Notes

Remark 1.1 Consider the method above except make *seedLen* an input to the encoding and decoding process. In particular, suppose the forger may select a chosen value of *seedLen* having seen messages signed with respect to some other value of *seedLen*. Then the resulting mechanism is not secure. Let us illustrate.

Above we concatenated *seed* and M and then applied G in order to get w. Let the adversary have the signature s of some message M, a signature which uses a seed *seed* of length *seedLen*. Let $seed = seed1 \parallel seed2$, where *seed2* is not the empty string. Then the adversary immediately knows a signature for the message $seed2 \parallel M$ with respect to seed *seed1*, since s itself is such a signature. Thus the signature of a new message has been created.

As can be seen from this example, careless combining of M and *seed* prior to hashing allows one to forge messages with one value of *seedLen* using the signature associated to another set of encoding parameters. This was not ruled out by the proof of [3] because the values of *seedLen* and *wLen* were fixed.

If *seedLen* is to be regarded as an input, the mask generation function could be treated as a function of multiple arguments, each of them meaningfully differentiated. In particular, a suitable instantiation of the mask generation function would encode the length of each of its arguments.

The desired output length could also be encoded. For example, to compute $G(x_1, x_2, \ldots, x_i)[b]$ one could first set

$$x = x_1 \ \| \ \mathrm{I2OSP}_8(|x_1|) \ \| \ x_2 \ \| \ \mathrm{I2OSP}_8(|x_2|) \ \| \ \cdots \ \| \ x_i \ \| \ \mathrm{I2OSP}_8(|x_i|) \ \| \ \mathrm{I2OSP}_8(b)$$

and then use x to generate a string of b octets using the method currently described as the mask generation function MGF1 of [6]. Here $|x_i|$ denotes the length of x_i, and $\mathrm{I2OSP}_8(i)$ is the 8-octet string which encodes the number i.

Remark 1.2 The proof of security assumes two (unrelated) random oracles, g and h. Above we used a single mask generation function, G, to instantiate both of these oracles. To keep separate what are conceptually (and in proofs) independent random oracles one could encode a unique identifier inside the scope of the mask generation function. For this particular protocol, this turns out to be unnecessary, at least insofar as interactions among these two oracles. That is, provable security remains (in the random-oracle model) even had one used the same random oracle for what are, in [3], two distinct random oracles. Nonetheless, we believe that separating the different uses of the mask generation function is a generally good practice. It makes security more transparent and, when all algorithms do this, it avoids interaction among hash functions of *different* protocols.

Remark 1.3

Combining the last two comments, the encoding function for EMSA-PSS could be as follows. Note the change in arguments to G at lines 4 and 5 for the encoding procedure, and the analogous change in the decoding procedure.

EMSA-PSS Encoding (with multiple-argument mask generation function)

1. If the length of M, $mLen$, is greater than the length limitation (eg., $mLen \geq 2^{60}$), or $l < 8(wLen + seedLen)$, then return "error".
2. Let $oLen = \lceil l/8 \rceil$ denote the maximal length of the message representative in octets.
3. Generate a fresh, random octet string, $seed$, having $seedLen$ octets.
4. Let $w = G(\text{"p1363-emsa-pss-make-w"}, seed, M)[wLen]$.
5. Let $expandedW = G(\text{"p1363-emsa-pss-expand-w"}, w)[oLen - wLen]$.
6. Let $seedMask = expandedW[1..seedLen]$ be the first $seedLen$ octets of $expandedW$.
7. Let $remainMask = expandedW[seedLen + 1..oLen - wLen]$ be the remaining $oLen - wLen - seedLen$ octets of $expandedW$.
8. Let $maskedSeed = seed \oplus seedMask$.
9. Let $T = w \ \| \ maskedSeed \ \| \ remainMask$.
10. Convert the octet string T to the binary number f which it represents using BS2IP.
11. Return f.

Naturally the decoding function would be similarly modified:

EMSA-PSS Decoding (with multiple-argument mask generation function)

1. If the length of M, $mLen$, is greater than the length limitation (eg., $mLen \geq 2^{60}$), or $l < 8(wLen + seedLen)$, then return "invalid".

2. Let $oLen = \lceil l/8 \rceil$ denote the maximal length of the message representative in octets.
3. Convert f into the octet string T of length $oLen$ which represents it using I2OSP.
4. Let $w = T[1..wLen]$ be the first $wLen$ octets of T.
5. Let $maskedSeed = T[wLen + 1..wLen + seedLen]$ be the next $seedLen$ octets of T.
6. Let $remainMask[wLen + seedLen + 1..oLen]$ be the remaining octets of T.
7. Let $expandedW = G(\text{“p1363-emsa-pss-expand-w”}, w)[oLen - wLen]$.
8. Let $seedMask = expandedW[1..seedLen]$.
9. Let $remainMask2 = expandedW[seedLen + 1..oLen - wLen]$.
10. Let $seed = maskedSeed \oplus seedMask$.
11. If $G(\text{“p1363-emsa-pss-make-w”}, seed, M)[wLen] \neq w$, or $remainMask2 \neq remainMask$, then return “invalid”. Otherwise return “valid”.

Remark 1.4 If $wLen$ is allowed to vary, one might set a minimal acceptable value. Values less than 8 octets, say, are clearly insecure. The proof of security proof a suggests a choice such as 16 or 20 octets.

Remark 1.5 The security analysis suggests a like value for $seedLen \approx wLen$, say 16 or 20 octets. But smaller values are not overly problematic. Indeed when $seedLen = 0$ the scheme effectively “collapses” to something akin to the “full domain hash (FDH)” encoding method [3, 1]. Like FDH, the $seedLen = 0$ scheme remains provably secure (in the random oracle model), albeit without the tight quantitative security.

Remark 1.6 The authors of this note do not have an opinion about whether $wLen$ and $seedLen$ should be constants, parameters tied to the modulus, or inputs. On the other hand, we do believe that it generally good practice to modify the mask generation function G to allow multiple differentiated arguments, one of which is used to effectively separate oracle instances. This issue, however, is orthogonal to the PSS proposal.

1.3 Encoding Method for Signatures with Message Recovery: EMSR-PSS

Often signatures schemes with appendix are able to handle messages of arbitrary (or essentially arbitrary) length, while signatures with message recovery can not be used to sign messages with lengths which exceed a certain portion of the length of the signature itself. This sharp cutoff in the maximal length of messages which can be signed seems rather arbitrary and restrictive. Of course a 1025-bit message can not possibly be encoded into 1024-bit signature, but there is no *a priori* reason to believe that it can not be encoded into a 1024-bit signature plus one extra bit. Thus, in contrast to other schemes for signatures with message recovery, we permit messages of arbitrary length (or essentially arbitrary length) to be signed. We shove into the signature “as much as will fit.” Any excess, the “overhanging portion of the message,” will have to be transmitted along with the signature. Of course this overhanging portion of the message will often be empty.

The method follows. (Again, the choice of a multiple-argument mask generation function and the style of its use are issues orthogonal to the substance of this proposal.)

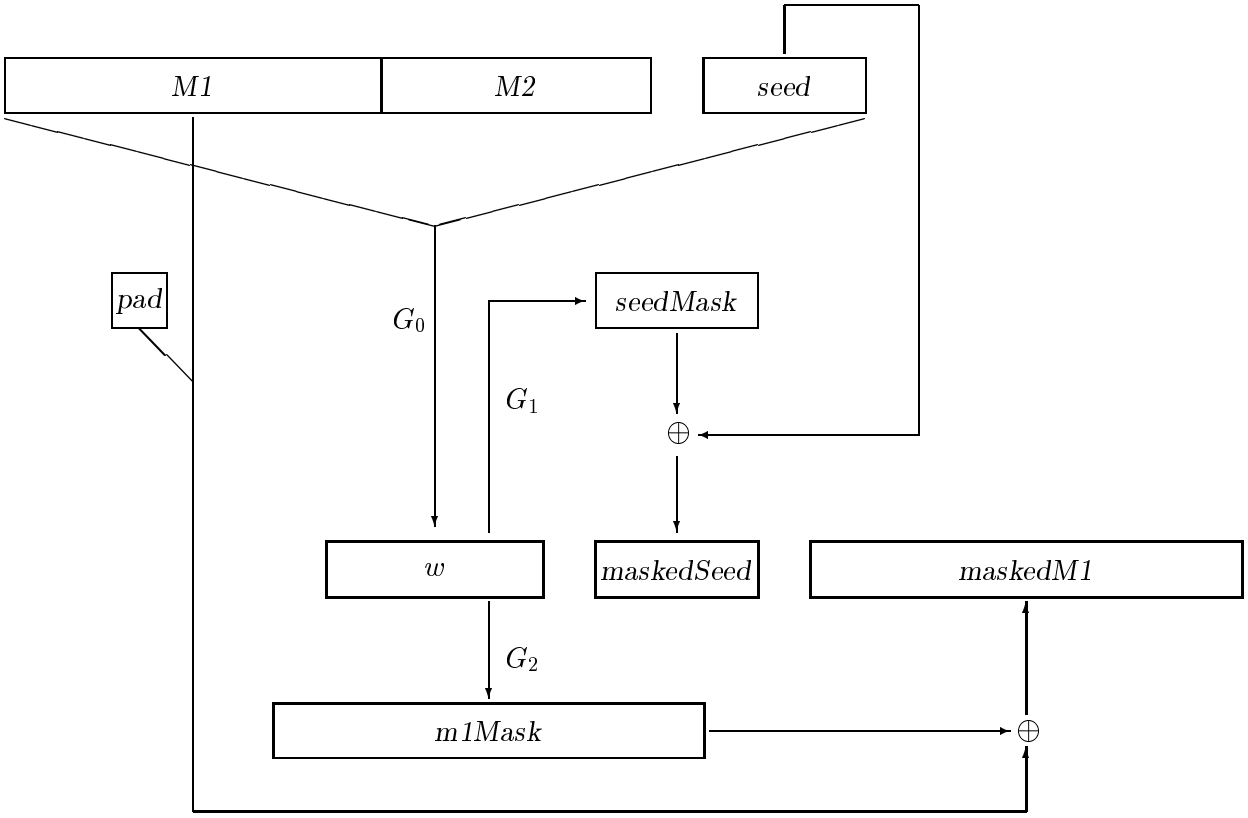

Figure 2: EMSR-PSS. Computing the message representative, the pieces of which (prior to integer conversion) are darkened.

1.3.1 Encoding Operation

Input:
— the octet string M, the message
— the number l, which is the maximum bit length of the output.

Output: Assuming the output is not "error", the method returns:
— a message representative, which is an integer f
— an octet string, $M2$, which is the "overhanging piece of the message".

The message representative, f, and the overhanging piece of the message, $M2$, are computed by the following or an equivalent sequence of steps. (Also see Figure 2.)

1. If the length of M in octets, $mLen$, in greater than the length limitation (eg, $mLen \geq 2^{60}$), or $l < 8(wLen + seedLen) + 8$, then return "invalid".
2. Let $oLen = \lceil l/8 \rceil$ denote the maximal length of the message representative in octets.
3. Let $m1Len = \min\{mLen,\ oLen - wLen - seedLen - 1\}$.
4. Let $M1 = M[1..m1Len]$ be the first $m1Len$ octets of M

5. Let $M2 = M[m1Len + 1..mLen]$. be the remaining octets of M.
6. Let $padLen = oLen - wLen - seedLen - m1Len$. (Note $padLen \geq 1$.)
7. Let $pad = 00 \cdots 00\ 01$ be a sequence of $padLen - 1$ zero-bytes followed by a single byte encoding the number 1.
8. Let $paddedM1 = pad \parallel M1$.
9. Generate a fresh, random octet string, *seed*, having *seedLen* octets.
10. Let $w = G(\text{“p1363-emsr-pss-make-w”}, seed, M)[wLen]$.
11. Let $expandedW = G(\text{“p1363-emsr-pss-expand-w”}, w)[oLen - wLen]$.
12. Let $seedMask = expandedW[1..seedLen]$ be the first *seedLen* octets of *expandedW*.
13. Let $m1Mask = expandedW[seedLen+1..oLen-wLen]$ be the remaining octets of *expandedW*.
14. Let $maskedSeed = seed \oplus seedMask$.
15. Let $maskedM1 = paddedM1 \oplus m1Mask$.
16. Let $T = w \parallel maskedSeed \parallel maskedM1$.
17. Convert the octet string T to the binary number f which it represents using BS2IP.
18. Return f and *M2*

1.3.2 Decoding Operation

Input:
— the message representative, which is an integer $f \geq 0$
— the octet string *M2*, the overhanging piece of the message (may be empty).
— the number l, which is the maximum bit length of the output

Output: the message M or “invalid”

The message (or invalidity indicator) shall be determined by the following or an equivalent sequence of steps:

1. If the length of *M2* in octets, *m2Len*, in greater than the length limitation, or $l < 8(wLen + seedLen) + 8$, then return “invalid”.
2. Let $oLen = \lceil l/8 \rceil$ denote the maximal length of the message representative in octets.
3. Convert f into the octet string T of length *oLen* which it represents using I2OSP.
4. Let $w = T[1..wLen]$ be the first *wLen* octets of T.
5. Let $maskedSeed = T[wLen + 1..wLen + seedLen]$ be the next *seedLen* octets of T.
6. Let $maskedM1 = T[wLen + seedLen + 1..oLen]$ be the remaining octets of T.
7. Let $expandedW = G(\text{“p1363-emsr-pss-expand-w”}, w)[oLen - wLen]$.
8. Let $seedMask = expandedW[1..seedLen]$ be the first *seedLen* octets of *expandedW*.
9. Let $m1Mask = expandedW[seedLen+1..oLen-wLen]$ be the remaining octets of *expandedW*.
10. Let $seed = maskedSeed \oplus seedMask$.
11. Let $paddedM1 = maskedM1 \oplus m1Mask$
12. Let i be the smallest positive number such that the i-th octet of *paddedM1*, $paddedM1[i]$, is the octet 01. If there is no such 01-octet in *paddedM1*, return “invalid”.
13. Let $pad = paddedM1[1..i]$.
14. Let $M1 = paddedM1[i + 1..oLen - wLen]$.

15. Let $M = M1 \parallel M2$.
16. If $G(\text{“p1363-emsr-pss-make-w”}, seed, M)[wLen] \neq w$, or
    $pad[j] \neq 00$ for any number j between with $1 \leq j \leq i-1$, or
    ($i > 1$ and $M2$ is the not the empty string)
    then return “invalid”. Otherwise, return the recovered message, M.

1.3.3 Notes

Remark 1.7 All of the remarks in Section 1.2.3 apply here, too.

Remark 1.8 By virtue of Remarks 1.4, 1.5 and 1.7, one might wish to allow greater flexibility in the choice of *seedLen* than the choice of *wLen*. Shorter values of *seedLen* allow longer messages to be packed into the signature.

2 Attributes and Advantages of the Encoding Methods

2.1 Enumeration of Attributes and Advantages

We enumerate what we see as the key attributes and advantages of PSS (meaning EMSA-PSS and EMSR-PSS both) when compared with the techniques EMSA2 and EMSR1 of [6]. (Because of the familiarity of the readership with [6], we dispense with a review of its signature encoding techniques.)

1. **Provable security** (in the random oracle model). Assurance for PSS stems not from an inability to find attacks, but from proofs. By virtue of these proofs, an ability to forge using *any* attack which does not exploit some structural characteristic of the MGF implies the ability to break the underlying primitive in a very strong sense: that is, invert RSA or the Rabin/Williams primitive on random instances. This is a qualitatively higher level of guarantee than the more *ad hoc* techniques of EMSA2 or EMSR1.

2. **Tight provable security with RSA** (in the random-oracle model). When RSA is the underlying primitive, something even more is known: that the ability to forge with resources $\mathcal{R}$ in an attack which does not exploit some structural characteristic of the MGF implies the ability to invert RSA on random strings using computational resources *only slightly greater than* $\mathcal{R}$. This is called *tight* provable security and it is, apart from reliance on the RO model, about the best one can hope for in proofs of cryptographic security.

3. **Weakened assumptions on collision-intractability**. As first pointed out to us by Matt Robshaw, the manner in which we use the mask generation function to map the seed *seed* and message M to a string w does not necessitate collision intractability in its customary, strong form. Call the map which takes *seed* and M to w by $w = G_0(seed, M)$. Given $G_0(seed_i, M_i)$ for random $seed_i$ and adversarially chosen M_i, the adversary should be unable to come up with a new $(seed, M)$ such that $G_0(seed, M) = G_0(seed_i, M_i)$ for some i. By virtue of the $seed_i$ values, which the adversary can not control, this is an apparently weaker requirement than asking for G_0 to be collision intractable. Intuitively, the random seed works to give the adversary very limited ability to manipulate what message representatives will be used.

4. **Robustness against randomness failures**. Although PSS encoding makes use of a random string *seed*, a failure in this string being random is not catastrophic. What happens in that case (say the random number generator is "stuck" at the constant 0) is that one loses Advantages 2 and 3 that we have just described. Provable-security remains, it is just that the underlying reduction is not as good. (Note that the [6] encoding schemes do not give any form of provable security, tight or otherwise.)

5. **No length restrictions for message recovery** (EMSR-PSS). Method EMSR-PSS works on messages of arbitrary length, packing into the signature as many octets of the message as fit. Unlike EMSR1 and ISO 9796 there is no "discontinuity" in the lengths of messages for which the method is applicable.

6. **Greater bandwidth efficiency** (EMSR-PSS). In a signature scheme with message recovery, the *bandwidth efficiency* measures the fraction of the signature that is occupied by (an encoding of) message bits; see [9]. ISO 9796 achieves bandwidth efficiency of about 1/2; roughly $oLen/2$ octets of message can be placed in a $oLen$ octet signature. EMSR-PSS allows one to pack $oLen - wLen - seedLen - 1$ octets of information in an $oLen$ octet signature. For recommended values of $oLen, wLen, seedLen$, this is greater bandwidth efficiency.

7. **Unified treatment for signatures with appendix and message recovery**. Methods EMSA-PSS and EMSR-PSS are nearly identical. They share they same security proof. With a small change in EMSA-PSS, the methods could be made to have a common implementation or specification.

2.2 Comment

We mention in passing that we view differently the level of assurance provided by EMSA2 (which is similar to X9.31) and EMSR1 (which is based on ISO 9796-1). Though neither enjoys any sort of provable-security guarantee when used in its intended way, it seems to us more likely that EMSA2 will fall to a concrete attack. This would mean that the "increment" that PSS provides to EMSA2 is effectively greater. Let us explain.

In EMSA2 / ISO 9796 there is again defined a *redundancy function* R which maps a string M to its message representative f. Though there are a variety of details, f is basically the octets of M at odd octet positions, and the "shadow" of these octets at even octet positions. The shadow of an octet is what one gets by looking up that octet in a table which is specified in the standard.

The security of EMSA2 depends on the following assumption: that no adversary, given the modulus n, can find distinct strings $x_1, \ldots, x_a$ and $y_1, \ldots, y_b$ such that $\prod_i R(x_i) \bmod n = \prod_i R(y_j) \bmod n$. Given the non-cryptographic nature of R, such an assumption seems quite strong. It even seems plausible that one could find distinct $x_1, \ldots, x_a, y_1, \ldots, y_b$ such that $\prod_i R(x_i) = \prod_i R(y_j)$, that is, the equality holds in the integers. This would amount to a way to get a forgery that works for *any* modulus n. The stated assumption would seem to be unrelated to factoring.

3 Security Assessment of the Encoding Methods

To avoid a lengthy (and increasingly well-known) discussion and set of definitions and proofs, we incorporate by reference the contents of Sections 1–5 of [3].[1] No changes have been made to the methods described in [3] beyond the trivial one to allow arbitrary-length messages be signed with EMSR-PSS. (For ease of explication, this was only mentioned in [3].)

By virtue of the referenced results, we view PSS under RSA as effectively "leapfrogging" one generation of assurance for a cryptographic standard. In the past, methods defined within standards have rarely had any sort of provable-security properties. Now we are just beginning to see standards which use schemes about which one can make provable-security claims. But the methods of this proposal go further, by actually providing a scheme which, when instantiated with RSA, admit a tight reduction.

(The value of tightness in a reduction in made explicit by an example of [3] (Section 1.4) which indicates that, under specified assumptions, one would need a 3447-bit modulus under the "FDH" encoding method (Section 3 of [3]) to get the same guaranteed security achieved by a 1024-bit modulus under PSS. Nonetheless, the value of tightness in a reduction should neither be understated nor overstated. While desirable, *any* sort of reduction is often well ahead of prevailing practice, and, in many cases, tight reductions are not known or are known only by virtue of impractical techniques.)

The comment we made in Section 2 about what happens if the seed *seed* is *not* selected at random (Comment 4 of that section) could use some additional clarification. Suppose, instead of *seed* being chosen at random, it is actually selected by the *adversary*. (This is as bad a failure of randomness as one could ever possibly imagine.) Even then, PSS retains provable security. The proof is along the exact same line as the proof of FDH security in Section 3 of [3].

One can conclude that the randomness in PSS is less essential than the acronym for PSS ("probabilistic signature scheme") might suggest. Again, the randomness is used for tight security, and to obtain a heuristic benefit of apparently weakened cryptographic assumptions. The probabilism is less crucial than in contexts such as OAEP or PKCS #1.

4 Limitations

Recently some concerns have been raised about the use of the random-oracle paradigm of [1]. The best expression of these concerns is in [5], where the authors demonstrate that it is possible to devise cryptographic protocols which are provably secure in the random-oracle model but for which no complexity assumption properly instantiates the random-oracle-modeled hash function. However, these examples are contrived to make the paradigm fail by having the schemes explicitly depend on the instantiating functions, so that these concerns do not appear to apply to the schemes presented here, or, in fact, to any of the concrete practical schemes that have been proven secure in the random-oracle model. A protocol with a proof of security in the random-oracle model does have higher assurance than one that lacks any proof in the provable-security tradition, and the competing methods fall into the latter category. (A protocol with a proof of security under a

[1] Readers of both this document and [3] will note that what is here called EMSA-PSS is in [3] known as PSS; and what is here called EMSR-PSS is in [3] called PSS-R. The change in names was just to be a bit more consistent with P1363 conventions.

standard complexity theoretic assumption would provide even higher assurance, but all protocols known with such proofs are still impractical.)

The tight security guarantee has been proven only for RSA-based signing. We do not know if one also obtains a tight bound for RW-based signing, though we expect that one does not. For a non-tight bound for RW, one can fall back on the FDH-style analysis from [3].

We comment that tight-proven-security Rabin signatures are in fact easy to obtain, but by changing not just the encoding scheme, but the primitive. One Rabin-based signature scheme is described in [3]. We did not introduce this method in this writeup because it involves specifying not just the PSS encoding method, but also a new primitive: a different (and simpler) version of the Rabin scheme. The authors will be happy to formally submit this Rabin primitive if there appears to be interest.

Finally we note that while we have provided a "heuristic" argument to suggest that the hash functions of this proposal are being used in a rather weak way, where even collisions in the hash function might not give rise to any insecurity in the signature scheme. To achieve this heuristic benefit the hash function $h(r, M)$ would have to be instantiated with care; otherwise, collisions in h may well lead to forgeries. And we emphasize that this benefit has only been argued heuristically, and we do not know how to meaningfully formalize it.

5 Intellectual Property Statement

The University of California has filed a patent application (US only) on the techniques used in PSS [2]. The University of California will license any resulting patent in a reasonable and non-discriminatory fashion. A letter to this effect will be provided.

References

[1] M. Bellare and P. Rogaway, Random oracles are practical: a paradigm for designing efficient protocols. Available via URL of either author. Preliminary version in *Proceedings of the First Annual Conference on Computer and Communications Security*, ACM, 1993.

[2] M. Bellare and M. Rogaway, Probabilistic signature scheme. US Patent Application.

[3] M. Bellare and P. Rogaway, The exact security of digital signatures— How to sign with RSA and Rabin. Available via URL of either author. Preliminary version in *Advances in Cryptology – Eurocrypt* 96 *Proceedings*, Lecture Notes in Computer Science Vol. 1070, U. Maurer ed., Springer-Verlag, 1996.

[4] M. Bellare and P. Rogaway, Optimal asymmetric encryption — How to encrypt with RSA. Available via URL of either author. Preliminary version in *Advances in Cryptology – Eurocrypt* 94 *Proceedings*, Lecture Notes in Computer Science Vol. 950, A. De Santis ed., Springer-Verlag, 1994.

[5] R. Canetti, O. Goldreich and S. Halevi, The random oracle methodology, revisited. *Proceedings of the* 30th *Annual Symposium on the Theory of Computing*, ACM, 1998.

[6] IEEE P1363 Committee, IEEE P1363 / D5 (Draft version 5), Standard specifications for public key cryptography. August 1998. See http://grouper.ieee.org/groups/1363/index.html/

[7] ISO/IEC 9796, Information technology security techniques – Digital signature scheme giving message recovery. International Organization for Standards, 1991.

[8] ISO/IEC 14888-3, Digital signatures with appendix - Part 3: Certificate-based mechanisms Draft. International Organization for Standards, 1998.

[9] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

[10] R. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, vol. 21 (1978).

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

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