Signature Schemes Based on the Strong RSA Assumption*

Ronald Cramer
Institute for Theoretical Computer Science, ETH Zurich, 8092 Zurich, Switzerland
`cramer@inf.ethz.ch`

Victor Shoup
IBM Zurich Research Laboratory, Säumerstr. 4, 8803 Rüschlikon, Switzerland
`sho@zurich.ibm.com`

July 27, 1999

Abstract

We describe and analyze a new digital signature scheme. The new scheme is quite efficient, does not require the the signer to maintain any state, and can be proven secure against adaptive chosen message attack under a reasonable intractability assumption, the so-called strong RSA assumption. Moreover, a hash function can be incorporated into the scheme in such a way that it is also secure in the random oracle model under the standard RSA assumption.

1 Introduction

We describe new, efficient digital signature schemes whose security is based on the strong RSA assumption.

By security, we mean security against an adaptive chosen message attack, as defined in [11]. To prove that our new schemes are secure, we need to make the *strong* RSA assumption, recently introduced by [2]. We also need a collision-resistant hash function—actually, as we shall see, a universal one-way hash function [16] is sufficient.

*This is a revision of IBM Research Report RZ 3083 (December 1998).

Our new schemes are interesting in that they are state-free, unlike other provably secure schemes [11, 9, 7]. Of course, we achieve this at the expense using of a potentially stronger assumption than is made in [11, 9, 7].

We stress that in discussing proofs of security, we *are not* making use of the "random oracle" model of computation [3], but rather, we are working in the "real world" of computation. Indeed, the standard "hash and invert" RSA signature is provably secure in the random oracle model under the standard RSA assumption, but in the "real world," its security is not well understood.

We also make the further observation that—at almost no cost—another hash function can be incorporated into our new schemes in such a way that they are also secure in the random oracle model under the standard RSA assumption. In this sense, our schemes can be made to be at least as secure as a standard RSA signature.

The strong RSA assumption is the assumption that the following problem is hard to solve. Given a randomly chosen RSA modulus n and a random $z \in \mathbf{Z}_n^*$, find $r > 1$ and $y \in \mathbf{Z}_n^*$ such that $y^r = z$. Note that this differs from the ordinary RSA assumption, in that for the RSA assumption, the exponent r is chosen independently of z, whereas for the strong RSA assumption, r may be chosen in a way that depends on z. The strong RSA assumption is a potentially stronger assumption than the RSA assumption, but at the present time, the only known method for breaking either assumption is to solve the integer factorization problem.

Independently, Gennaro, Halevi, and Rabin [10] have also recently discovered efficient, state-free signature schemes based on the strong RSA assumption. Our schemes are actually quite different from theirs, and we think that all of these different schemes are of interest from both a theoretical and practical perspective, because they are the only truly practical and state-free schemes available that admit a proof of security under a natural intractability assumption. Moreover, our scheme is potentially more efficient for the following reason. The paper [10] contains several signature schemes, but the only fully proved scheme requires a "trapdoor" or "chameleon" collision-resistant hash function with the following very special property: its output is a prime number. Implementing such a hash function is both awkward and potentially computationally expensive. Indeed, depending on the security parameters and implementation details, evaluating this hash function can dominate the running time of the signing algorithm. Our scheme sidesteps this problem altogether. While the signing algorithm still has to generate a prime number, it has a great deal of flexibility in how this is done, yielding a much more efficient algorithm. Basically, our signing algorithm just needs to generate *any* prime number of appropriate length (e.g., 161 bits) subject only to the requirement that the probability of generating the same prime twice is small.

Our new schemes can be seen as variations of the scheme of Cramer and Damgard [7], which itself can be seen as an adaptation of the identification scheme of Guillou and Quisquater [12]. In §2, we present and analyze our basic scheme. In §3, we present and analyze a variation based on trapdoor hashing. In §4, we sketch an algorithm for fast prime generation, as required by the signing algorithm, and discuss how the intractability assumption can be weakened by using hash functions.

2 The Basic Scheme

In this section we describe the basic scheme, and give a proof of its security.

The scheme is parameterized by two security parameters, l and l', where $l+1 < l'$. Reasonable choices might be $l = 160$ and $l' = 512$. The scheme makes use of a collision-resistant hash function H whose output can be interpreted as a positive integer less than 2^l. A reasonable choice for H might be SHA-1.

For a positive integer n, we let QR_n denote the subgroup of $\mathbf{Z}_n^*$ of squares (i.e., the quadratic residues modulo n).

Key Generation Two random l'-bit primes p and q are chosen, where $p = 2p' + 1$ and $q = 2q' + 1$, with both p' and q' prime. Let $n = pq$. Also chosen are:

- random $h, x \in \mathrm{QR}_n$;
- a random $(l+1)$-bit prime e'.

The public key is

$$(n, h, x, e').$$

The private key is

$$(p, q).$$

Signature Generation To sign a message m (an arbitrary bit string), a random $(l+1)$ bit prime $e \neq e'$ is chosen, and a random $y' \in \mathrm{QR}_n$ is chosen. The equation

$$y^e = xh^{H(x')}$$

is solved for y, where x' satisfies the equation

$$(y')^{e'} = x' h^{H(m)}.$$

Note that y can be calculated using the factorization of n in the private key. The signature is

$$(e, y, y').$$

Signature Verification To verify a putative signature (e, y, y') on a message m, it is first checked that e is an odd $(l+1)$-bit number different from e'. Second, $x' = (y')^{e'} h^{-H(m)}$ is computed. Third, it is checked that $x = y^e h^{-H(x')}$.

Implementation Notes

We remark that the signature verification algorithm *does not* need to verify that e is prime.

To speed both verification and signing, the public key might contain h^{-1} instead of h.

In generating a signature, the only full-length exponentiation that needs to be performed is in the computation of y. The cost of this can be significantly reduced, as follows. First, we can arrange that $x = h^a$ for a random number $a \bmod p'q'$, where a is stored in the secret key. This is acceptable, because h is with overwhelming probability a generator of QR_n, and thus the distribution of the public key does not change significantly. Now, if d is the inverse of $e \bmod p'q'$, then $y = h^b$, where $b = da + dH(x') \bmod p'q'$. So the computation of y involves exponentiation with the *fixed* base h to the power b. Using pre-computation techniques [14], we can substantially reduce the number of modular multiplications using a table of pre-computed numbers.

Of course, in all of the above, one utilizes the Chinese Remainder Theorem as well to speed the exponentiations.

We also note that the primes generated by the signer do not have to be random primes. The only requirement is that the probability of generating the same prime twice is negligible.

Using these implementation ideas, together with a fast prime generator like the one described in §4, it appears that the time for signing is no more than 1.5 times the time for signing with the standard RSA algorithm. Verifying a signature will take less time than signing, but still substantially more time than that required to verify a low-exponent RSA signature.

Proof of Security

Now we proceed to prove the security of the above scheme.

Theorem 1 *The above signature scheme is secure against adaptive chosen message attack, under the strong RSA assumption and the assumption that H is collision-resistant.*

To prove this theorem, let us consider a forging algorithm that makes t signing queries and then produces a forgery. For $1 \le i \le t$, let m_i be the ith message signed, let (e_i, y_i, y'_i) be the ith signature, and let x'_i be defined as $x'_i = (y'_i)^{e'} h^{-H(m_i)}$. Let (e, y, y') be the forgery on message m (so $m \neq m_i$ for all $1 \le i \le t$). Also, let $x' = (y')^{e'} h^{-H(m)}$.

We distinguish between three types of forgeries:

Type I For some $1 \le j \le t$, $e = e_j$ and $x' = x'_j$.

Type II For some $1 \le j \le t$, $e = e_j$ and $x' \neq x'_j$.

Type III For all $1 \le i \le t$, $e \neq e_i$.

We assume that no two e_i are equal, and so a forgery has a unique type. We also assume that no e_i is equal to e'.

If there is a forger that succeeds with non-negligible probability, then there exists either Type I forger, a Type II forger, or a Type III forger, one of which succeeds with non-negligible probability. We show that any of these forgers can be turned into an algorithm breaking the strong RSA assumption. In fact, a forger of Types I or II can be used to break the RSA assumption, and the proof of this is quite similar to proofs in [7]. We only need the strong RSA assumption in case the forger is Type III.

Type I Forger

Suppose we have a Type I forger that succeeds with non-negligible probability. We want to show how to use this forger to break the RSA assumption. That is, we are given n, a random $z \in \mathbf{Z}_n^*$, and a random $(l+1)$-bit prime r, and we want to compute $z^{1/r}$.

We describe a simulator that interacts with the forger. We choose random $(l+1)$-bit primes $e_1, \ldots, e_t$, and we create a public key as follows. We set

$$h = z^{2 \prod_i e_i}.$$

We next choose $w \in \mathbf{Z}_n^*$ at random and set

$$x = w^{2\prod_i e_i}.$$

Finally, we set $e' = r$.

Now, to sign message m_i, the simulator chooses $y_i' \in \mathrm{QR}_n$ at random, and computes $x_i' = (y_i')^{e'} h^{-H(m_i)}$. Next, the simulator solves the equation $y_i^{e_i} = x h^{H(x_i')}$ for y_i, which is can easily do, since it knows the e_ith roots of x and h.

It is easy to see that the simulator perfectly simulates the forger's view.

Now suppose the forger creates a Type I forgery (e, y, y') on a message m. So for some $1 \le j \le t$, $e = e_j$ and $x' = x_j'$. This yields two equations

$$\begin{aligned} (y')^{e'} &= x' h^{H(m)}; \\ (y_j')^{e'} &= x' h^{H(m_j)}. \end{aligned}$$

Since we are assuming H is collision-resistant, we may assume that $H(m) \neq H(m_j)$. Thus, dividing these two equations, we can calculate $v \in \mathbf{Z}_n^*$ and an integer $a \not\equiv 0 \pmod{e'}$ such that

$$v^{e'} = h^a = z^{2a\prod_i e_i}.$$

Moreover, since $\gcd(2a\prod_i e_i, e') = 1$ and $e' = r$, we can easily compute an rth root of z. This is done via a standard procedure, which we will need to use several times, and we recall it for completeness. If we have $v^r = z^b$, where $\gcd(r, b) = 1$, then we compute b' such that $bb' = 1 + rk$. It follows that $(v^{b'} z^{-k})^r = z$.

Type II Forger

As in the Type I case, we are given n, $z \in \mathbf{Z}_n^*$ and r, and we want to find an rth root of z.

We may assume that the value j in the definition of a Type II forgery is fixed. If not, we can guess it.

Again, we describe a simulator. We create a public key as follows. For $1 \le i \le t$, with $i \neq j$, we choose e_i to be a random $(l+1)$-bit prime. We set $e_j = r$. We also select e' to be a random $(l+1)$-bit prime. We set

$$h = z^{2e'\prod_{i\neq j} e_i}.$$

We choose $w \in \mathbf{Z}_n^*$ at random, and set

$$y_j = w^{2\prod_{i \neq j} e_i}.$$

We choose $u \in \mathbf{Z}_n^*$ at random, and set

$$x'_j = u^{2e'}.$$

We compute

$$x = y_j^{e_j} h^{-H(x'_j)}.$$

Next, we describe how to sign message m_i. First, suppose $i \neq j$. We choose $y'_i \in \mathrm{QR}_n$ at random, and compute as $x'_i = (y'_i)^{e'} h^{-H(m_i)}$. Then, since we know the e_ith roots of x and h, we can easily compute the corresponding value y_i.

Second, suppose $i = j$. Since we know the e'th roots of h and x'_j, we can compute the correct value y'_j. The correct value of y_j has already been determined.

That completes the description of the simulator. It is easy to see that the simulator perfectly simulates the forger's view.

Now suppose the forger creates a Type II forgery (e, y, y') on a message m, where $e = e_j$ and $x' \neq x'_j$. Then we have

$$\begin{aligned} y^e &= x h^{H(x')}; \\ y_j^e &= x h^{H(x'_j)}. \end{aligned}$$

Then by an argument similar to that in the Type I case, we can divide these two equations, and calculate an rth root of z.

Type III Forger

Given a Type III forger, we show how to break the strong RSA assumption. That is, given n and $z \in \mathbf{Z}_n^*$, compute $r > 1$ and an rth root of z.

The simulator runs as follows. We choose random $(l+1)$-bit primes $e', e_1, \ldots, e_t$. We set

$$h = z^{2e' \prod_i e_i}.$$

Now we choose a random $a \in \{1, \ldots, n^2\}$, and set $x = h^a$.

Now, by construction, QR_n is a cyclic group of order $p'q'$. We can assume that h generates QR_n, since this happens with overwhelming probability.

Now let $a = bp'q' + c$, where $0 \le c < p'q'$. Because a was chosen at random from a suitably large interval, the distribution of c is statistically indistinguishable the uniform distribution on $\{0, \ldots, p'q'-1\}$. Moreover, the conditional distribution of b given c is statistically indistinguishable from the uniform distribution on $\{0, \ldots, \lfloor n^2/p'q' \rfloor\}$. That is, c and b are essentially independent.

Because the distribution of c is essentially uniform, x is essentially distributed like a random element of QR_n. Since we know all the relevant roots of x and h, we can easily sign all messages.

Now suppose the forger creates a Type III forgery, (e, y, y'). Then we have

$$y^e = xh^{H(x')} = z^m,$$

where

$$m = 2e' \prod_i e_i \cdot (a + H(x')).$$

Let $d = \gcd(e, m)$. Now, using the same procedure as was used in the Type I and Type II cases, we can compute an $(e/\gcd(e, m))$-th root of z, which is nontrivial provided $e \not| m$. So it suffices to show that $e \not| m$ with non-negligible probability. Let r be a prime dividing e. Now, $r \not| 2e' \prod_i e_i$ by construction. So it suffices to show that $r \not| (a + H(x'))$ with non-negligible probability. Let $a = bp'q' + c$ as above. Now, r may depend on c, but we observed above that c and b are essentially independent. And since by construction $r \not| p'q'$, it follows that $r \mid (a + H(x'))$ with probability very close to $1/r$, as we are evaluating a linear polynomial in b at a random point. Thus, with non-negligible probability, $r \not| (a + H(x'))$.

Using a universal one-way hash function

The notion of a universal one-way family of hash functions was introduced by Naor and Yung [16]. A family H of hash functions indexed by a key k is universal one-way if the following property holds: if an adversary chooses a message x, and then a random key k is chosen, it should be hard for the adversary to find $y \ne x$ such that $H_k(x) = H_k(y)$.

The universal one-way property is a much weaker property than full collision resistance, so it is desirable to rely on this weaker property from a security point of view. See [4] for further discussion. Note that the length of the key k may grow with the message length; for some constructions, the growth rate is logarithmic.

We can modify our basic signature scheme to use a universal one-way hash as follows. We add a random hash key k' to the public key. A signature is of the form (e, y, y', k), where k is a random hash key, and we have:

$$y^e = xh^{H_{k'}(k,x')} \text{ and } (y')^{e'} = x'h^{H_k(m)}. \tag{1}$$

Theorem 2 *The above signature scheme is secure against adaptive chosen message attack, under the strong RSA assumption and the assumption that H is a universal one-way family of hash functions.*

The proof is a simple modification of the proof of Theorem 1. We sketch the differences.

In classifying types of forgeries, we define Type I and Type II forgeries as follows.

Type I For some $1 \leq j \leq t$, $e = e_j$ and $(k, x') = (k_j, x'_j)$.

Type II For some $1 \leq j \leq t$, $e = e_j$ and $(k, x') \neq (k_j, x'_j)$.

Here, k_j represents the hash key used in signing the jth message, and k the hash key appearing in the forged signature. Type III forgeries are the same as before.

In the case of a Type I forgery, the proof is identical to the proof of Theorem 1, except that we have to observe that the universal one-way property and the fact that $k = k_j$ implies that $H_k(m) = H_{k_j}(m_j)$ with negligible probability.

In the case of a Type II forgery, the proof is the same as before, except that we have to argue that $H_{k'}(k, x') = H_{k'}(k_j, x'_j)$ with negligible probability. Suppose, to the contrary, that the adversary succeeds in finding (k, x') such that $H_{k'}(k, x') = H_{k'}(k_j, x'_j)$. Then we can break the universal one-way property of H using a different simulator, as follows. This new simulator generates a public key/private key pair for the signature scheme, but without choosing the hash key k'. Next, the simulator guesses the value j defining the Type II forgery, and generates $x'_j \in \mathrm{QR}_n$ at random, along with a random hash key k_j. Note that the correct length of k_j may depend on the length of m_j, which is at this point in time unknown to the simulator, so the simulator will also have to make a guess here as well. Now, a hash key k' is chosen, and the simulator is going to use the adversary to find a collision $H_{k'}(k, x') = H_{k'}(k_j, x'_j)$. The simulator completes the public key by adding k' to it. Now the adversary is run against this public key. Since the simulator knows the private key of the signature scheme, it can easily generate signatures for message i, for $1 \leq i \leq t$, with $i \neq j$. For $i = j$, the simulator generates a signature on a message m_j so that the resulting (k_j, x'_j) is equal to the previously chosen value of (k_j, x'_j). But this the simulator can easily do since it has the factorization of n available to it: it generates a prime e_j, and then solves the equations (1) for y_j and y'_j, using the given values of k_j, x'_j, e_j, along with k'. That completes the description of the simulator. If the adversary succeeds in finding a collision $H_{k'}(k, x') = H_{k'}(k_j, x'_j)$, then this breaks the universal one-way assumption on H.

3 Trapdoor Hash Scheme

Consider the basic signature scheme presented above, and consider a signature (e, y, y') on a message m. Let $x' = (y')^{e'} h^{-H(m)}$. Then we have $y^e = x h^{H(x')}$.

One can view the value $H(x')$ as a kind of "trapdoor hash," also called a "chameleon hash" (see [13] for detailed discussion, references, and further applications). One can also base a trapdoor hash on the assumed hardness of the Discrete Logarithm problem in a standard way, as follows. Let g_1, g_2 be two random generators for a group G of order s, where s is an $(l+1)$-bit prime. To hash a message m, we compute the hash value $\alpha = H(g_1^t g_2^{H(m)})$, where H is an ordinary, collision-resistant hash function, and t is chosen at random mod s. In addition to the hash value α, we also output the side information t. The trapdoor in this scheme is the g_1 logarithm of g_2. A simulator that knows the trapdoor can construct a hash value α without knowing m, and then later, given m, can construct and the appropriate side information t.

We now describe a signature scheme based on this.

Key Generation Two random l'-bit primes p and q are chosen, where $p = 2p' + 1$ and $q = 2q' + 1$, with both p' and q' prime. Let $n = pq$. Also chosen are:

- random $h, x \in \mathrm{QR}_n$;
- a group G of order s, where s is an $(l+1)$-bit prime, and two random generators g_1, g_2 of G.

The public key is

$$(n, h, x, g_1, g_2),$$

along with an appropriate description of G (including s). The private key is

$$(p, q).$$

Signature Generation To sign a message m (an arbitrary bit string), a random $(l+1)$ bit prime e is chosen, and a random $t \in \mathbf{Z}_s$ is chosen. The equation

$$y^e = xh^{H(g_1^t g_2^{H(m)})}$$

is solved for y. The signature is

$$(e, y, t).$$

Signature Verification To verify a putative signature (e, y, t) on a message m, it is first checked that e is an odd $(l+1)$-bit number. Second, it is checked that

$$x = y^e h^{-H(g_1^t g_2^{H(m)})}.$$

Theorem 3 *The above signature scheme is secure against adaptive chosen message attack, under the strong RSA assumption and the assumption that H is collision-resistant, and the assumption that the Discrete Logarithm problem for the group G is hard.*

The proof of this theorem is very similar to the proof of Theorem 1. We leave the details to the reader.

In a variation on this scheme, we give the signing algorithm the trapdoor to the hash. The advantage of doing this can be appreciated if one makes a distinction between the "off line" and "on line" cost of signing. If the signer has the trap door, then in fact the "on line" cost is essentially a single multiplication mod s—all of the other work in creating the signature can be done before the message m is actually received.

4 Remarks on Prime Generation

In our signature scheme, the signer must generate a random $(l+1)$-bit prime with each signature. As we remarked already, these primes need not be chosen from the uniform distribution $(l+1)$-bit primes. The only requirement is that the probability of generating two equal primes should be negligible. Thus, we have quite a bit of flexibility in how we generate these primes. This is perhaps important, because if one is not careful, the cost of prime generation can easily be the dominant cost of signing. This is especially so if one wants a completely rigorous algorithm with a sufficiently small error probability.

For example, suppose one uses the Miller-Rabin test [17] to test for primality. Suppose $l = 160$. Further, suppose we want an error rate of

2^{-96}, which will allow us to make 2^{32} signatures with an overall error rate of 2^{-64}. Now suppose we choose random 161-bit numbers until we have found a number that passes a number of trial divisions and a single Miller-Rabin test. Along the way, we will make a number of Miller-Rabin tests that reject some composite numbers that pass the trial division test. On average, we need to do 8-10 such Miller-Rabin tests, depending on how much trial division one does. Once we have found a number that passes a single Miller-Rabin test, we have to perform a number of additional Miller-Rabin tests to reduce the error probability sufficiently. Using results of Damgard *et al.* [8], roughly 20 additional tests suffice (although it is not clear how tight this bound is). Performing these tests can be quite costly. Empirical tests suggest that the time to generate primes via this technique will dominate the running time of the signature algorithm when $l' \approx 512$.

A Fast Prime Generation Algorithm

Here we sketch a very efficient algorithm for generating primes as required by the signing algorithm. Again, assume $l = 160$. So we need to generate a 161-bit prime. To do this, we first generate a random prime P in the range $(2^{52}, 2^{53})$. Because P is small enough, the primality of P can be quickly verified using one of a number of procedures described in Bleichenbacher's thesis [5, Chapter 3], which are correct for primes up to $10^{16} > 2^{53}$. For example, one of Bleichenbacher's results, as reported in [15], states that the Miller-Rabin test for the bases $2, 3, 5, 7, 11, 13$ and 23 is a correct primality test for numbers in this range. Next, we repeatedly choose integers R in the interval $((2^{160} - 1)/2P, (2^{161} - 1)/2P)$ until $e = 2PR + 1$ is prime.

The following lemma, which is a variant of a result of Brillhart *et al.* [6], provides an effective proof of primality.

Lemma 1 *Let e, P, and R be as above. Then e is prime if and only if the following conditions hold.*

(i) *There exists an integer a such that*

- $a^{e-1} \equiv 1 \bmod e$, *and*
- $\gcd(a^{2R} - 1, e) = 1$.

(ii) *If $R = 2Px + y$, where x and y are integers with $0 \leq y < 2P$, then $y^2 - 4x$ is neither 0 nor a perfect square.*

(iii) $R \not\equiv m \bmod 2Pm + 1$ *for all integers m with $1 \leq m < e/4P^3$.*

Remarks. Note that this lemma differs from the one in [6] only in condition *(iii)*. This condition is equivalent to the condition that $2Pm + 1 \nmid 2PR + 1$ for the stated values of m, and is formulated as it to allow for more efficient calculation. The reason we need this particular condition is that P is just slightly too small for the lemma to be valid otherwise, and until Bleichenbacher's results are improved, we cannot increase the size of P enough to drop this condition. The bound $e/4P^3$ on m in this condition is less than 8 in the worst case, and empirically appears to be less than 2 on average. The time spent evaluating this condition is insignificant compared to the overall prime generation time.

To apply this lemma, one first does some trial division, and if e passes this test, one applies the Miller-Rabin test with base $a = 2$. If this test passes, one applies the test of Lemma 1 with the same base a. The quantity $a^{e-1} \bmod e$ in condition *(i)* can be obtained for free as a by-product of the Miller-Rabin test. If the primality of e is still undetermined, we repeat the Miller-Rabin test and the test of Lemma 1 with a random base a, until the primality of e is determined. The expected number of such repetitions is bounded by a constant, but in practice, the primality of e is almost always determined by the base $a = 2$.

Proof of Lemma 1. We first show that if the three conditions hold, then e is prime. The other implication is left to the reader. Our proof follows the arguments in [15].

Condition *(i)* in the lemma implies that every prime divisor of e is of the form $2Pm + 1$ for some positive integer m.

Given the relative sizes of P and R, e can have at most three such prime divisors. If we have $e = \prod_{i=1}^{3}(2Pm_i + 1)$, then we must have $8P^3 m_1 m_2 m_3 < e$, which implies that e is divisible by a number $2Pm + 1$ where $1 \leq m < e/8P^3$. The condition that e is divisible by $2Pm + 1$ is easily seen to be equivalent to the condition that $R \equiv m \bmod 2Pm + 1$. Condition *(iii)* in the lemma rules out this possibility, so we can assume that if e is composite, we must have $e = (2Pm_1 + 1)(2Pm_2 + 1)$, or equivalently, $R = 2Pm_1m_2 + (m_1 + m_2)$.

We claim that $m_1 + m_2 < 2P$. To see this, suppose $m_1 + m_2 \geq 2P$. Then one of m_1 or m_2 must be at least P, say $m_1 \geq P$. Then we must have $2P^2 \cdot 2Pm_2 = 4P^3 m_3 < e$, and again, condition *(iii)* in the lemma rules out this possibility.

So we may assume that $m_1 + m_2 < 2P$, from which it follows that $y = m_1 + m_2$, and hence $x = m_1 m_2$. This implies that m_1 satisfies the equation $m_1^2 - ym_1 + x = 0$. But this is impossible, because condition *(ii)* in the lemma rules out the existence of an integer solution to this equation. So we conclude that e must be prime.

That completes the proof of Lemma 1.

The following lemma analyzes the running time and collision probability of the prime generation algorithm.

Lemma 2 *With this procedure, the expected number of trials until P is prime is at most 64. Assuming the Generalized Riemann Hypothesis, the following two assertions hold. For any fixed P, the expected number of trials until $e = 2PR + 1$ is prime is at most 128. For any fixed $\hat{e}$, the probability that a random $e = 2PR + 1$ is equal to $\hat{e}$ is at most 2^{-144}.*

To prove this we use some explicit estimates from [1]. First, by Theorem 8.8.1 in [1], we have the number of primes P in the given range is more than 2^{46}. The first claim in the lemma follows trivially.

For the second and third claims, we use Theorem 8.8.18 in [1], which gives a very sharp estimate on the number of primes in an arithmetic progression, assuming the Generalized Riemann Hypothesis. Using a simple calculation, this theorem implies that for any P, there are more than 2^{100} primes of the form $2PR+1$ in the range $(2^{160}, 2^{161})$. The second claim in the lemma now follows easily.

For the third claim, the number of primes dividing $\hat{e}-1$ that are greater than 2^{52} is at most 3. Therefore, $P \mid \hat{e}-1$ with probability at most $3 \cdot 2^{-46} \leq 2^{-44}$. Moreover, for any P, the probability that $2PR + 1 = \hat{e}$ is at most 2^{-100}. The third claim follows immediately.

We do not claim that this is the best way to generate 161-bit primes, or even particularly original, but it seems like a reasonable one, and it is certainly much more efficient than an iterated Miller-Rabin test. Some preliminary empirical tests indicate that the average running time of this prime generation technique will be less than one third the total time needed to sign a message, assuming $l' \approx 512$. Note, however, that the variance of running time of the prime generation step is quite high.

Since the technique suggested here provides a certificate of primality that is small and easily verified, one could augment the signature scheme by adding this certificate to the signature and having the verifier check it. This can only improve security, allowing us to weaken the strong RSA assumption so that the adversary's exponent has to be a certified prime of the proper form, and not just an arbitrary integer.

Using a Hash Function

We can weaken the intractability assumption even further. Suppose that in the above algorithm for generating a prime, we require that the random numbers P and R are outputs of a cryptographic hash function. The signing algorithm can feed random bits into such a hash function, and if the hash function is nearly uniform, the same properties that are proved above will still hold.

The hash function inputs that yield P and R (respectively) are included as part of the signature, and the verifier checks that these values are correct. By doing this, we greatly constrain the adversary's attack strategy, allowing us to weaken the strong RSA assumption so that the adversary's exponent is a certified prime of this very special form. This intuitively seems like a much harder problem, and indeed, this intuition is somewhat justified by the fact the resulting signature scheme is secure *in the random oracle model* under the *ordinary* RSA assumption. The details of this are fairly straightforward, and we leave them to the reader.

References

[1] E. Bach and J. Shallit. *Algorithmic Number Theory*, volume 1. MIT Press, 1996.

[2] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology–Eurocrypt '97*, pages 480–494, 1997.

[3] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[4] M. Bellare and P. Rogaway. Collision-resistant hashing: towards making UOWHFs practical. In *Advances in Cryptology–Crypto '97*, 1997.

[5] D. Bleichenbacher. *Efficiency and security of cryptosystems based on number theory.* PhD thesis, Swiss Federal Institute of Technology Zurich, 1996.

[6] J. Brillhart, D. Lehmer, and J. Selfridge. New primality criteria and factorizations of $2^m \pm 1$. *Math. Comp.*, 29:620–647, 1975.

[7] R. Cramer and I. Damgard. New generation of secure and practical RSA-based signatures. In *Advances in Cryptology–Crypto '96*, pages 173–185, 1996.

[8] I. Damgard, P. Landrock, and C. Pomerance. Average case error estimates for the strong probable prime test. *Math. Comp.*, 61:177–194, 1993.

[9] C. Dwork and M. Naor. An efficient existentially unforegeable signature scheme and its applications. In *Advances in Cryptology–Crypto '94*, pages 218–238, 1994.

[10] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *Advances in Cryptology–Eurocrypt '99*, pages 123–139, 1999.

[11] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17:281–308, 1988.

[12] L. Guillou and J. Quisquater. A practical zero-knowledge protocol fitted to security microprocesors minimizing both transmission and memory. In *Advances in Cryptology–Eurocrypt '88, Springer LNCS 330*, pages 123–128, 1988.

[13] H. Krawczyk and T. Rabin. Chameleon hashing and signatures. Preprint, *Theory of Cryptography Library*, March 1998.

[14] C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In *Advances in Cryptology–Crypto '94*, pages 95–107, 1994.

[15] U. Maurer. Fast generation of prime numbers and secure public-key cryptographic parameters. *J. Cryptology*, 8:123–155, 1995.

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

[17] M. O. Rabin. Probabilistic algorithms for testing primality. *J. of Number Theory*, 12:128–138, 1980.