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

May 9, 2000

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, quite 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 [GMR88]. This is the strongest type of security for a digital signature scheme that one can expect, and a signature scheme that is secure in this sense can be safely deployed in the widest possible range of applications. To prove that our new schemes are secure, we need to make the *strong* RSA assumption, recently introduced by [BP97]. We also need a collision-resistant hash function—actually, as we shall see, a universal one-way hash function [NY89] is sufficient.

Our new schemes are interesting in that they are state-free, unlike other provably secure schemes [GMR88, DN94, CD96]; our schemes are more efficient than these schemes as well. From a practical point of view, this state-freeness property is at least, if not more, important than efficiency. We achieve this at the expense using of a potentially stronger assumption than is made in [GMR88, DN94, CD96].

We stress that in discussing proofs of security, we *are not* making use of the "random oracle" model of computation, but rather, we are working in the "real world" of computation.

The random oracle model is a formal model in which a cryptographic hash function is treated *as if* it were a black box containing a random function. We stress that with respect to proofs of security, a proof in the random oracle model should not be viewed as just a stronger intractability

*To appear, *ACM Transactions on Information and System Security (ACM TISSEC)*; this is a major modification of an extended abstract in *Proc. 6th ACM Conf. on Computer and Communications Security*, 1999; a preliminary version was published as IBM Research Report RZ 3083 (December 1998).

assumption. It is at best a heuristic device that gives strong evidence that a scheme cannot be broken—however, it is entirely possible that a scheme can be secure in the random oracle model, and yet be broken without violating any particular intractability assumption, and without exhibiting any particular weakness in the cryptographic hash function (see [CGH98]). The random oracle model was first introduced in an informal way in [FS87], and was later formalized and further developed and applied in [BR93]. Subsequently, it has been used to analyze numerous cryptographic schemes (see, e.g., [PS96]).

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.

Independently, Gennaro, Halevi, and Rabin [GHR99] 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 [GHR99] 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 [CD96], which itself is an adaptation of [DN94], additionally using ideas from [GQ88] and [GMR88]. In §2, we discuss the RSA and strong RSA assumptions in somewhat greater detail. In §3, we present and analyze our basic scheme. In §4, borrowing ideas from the proof of security in §3, we show that the problem defining the strong RSA assumption is random self reducible. In §5, we present and analyze a variation based on trapdoor hashing. In §6, 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. In §7, we make some observations and recommendations for implementing the key generation algorithm. In §8, we briefly discuss an implementation of our new scheme and its performance.

2 The RSA and Strong RSA Assumptions

In this section, we review the RSA and strong RSA assumptions in somewhat more detail.

The RSA problem is the following. Given a randomly generated RSA modulus n, an exponent r, and a random $z \in \mathbf{Z}_n^*$, find $y \in \mathbf{Z}_n^*$ such that $y^r = z$. The exponent r is drawn from a particular distribution—particular distributions give rise to particular versions of the RSA problem. The *RSA assumption* is the assumption that this problem is hard to solve.

The *flexible* RSA problem is the following. Given an RSA modulus n and a random $z \in \mathbf{Z}_n^*$, find $r > 1$ and $y \in \mathbf{Z}_n^*$ such that $y^r = z$. The choice of r may be restricted in some fashion—particular restrictions give rise to particular versions of the flexible RSA problem. The *strong RSA assumption* is the assumption that this problem is hard to solve. 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 was introduced in [BP97], and has subsequently been used in the analysis of several cryptographic schemes (see, e.g., [FO99, GHR99]). This 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.

One of the nice features about the RSA problem is that it is random self reducible. That is, having fixed n and r, then the problem of computing $y = z^{1/r}$ for an *arbitrary* $z \in \mathbf{Z}_n^*$ can be reduced to the problem of computing $\tilde{y} = \tilde{z}^{1/r}$ for *random* $\tilde{z} \in \mathbf{Z}_n^*$. This means that given an efficient algorithm to solve the latter problem, one can efficiently solve the former problem. This is a well-known and quite trivial reduction: given z, choose $s \in \mathbf{Z}_n^*$ at random, and set $\tilde{z} = s^r z$. Then we have $y = \tilde{y}/s$.

The existence of such a random self reduction adds credibility to the RSA assumption, since if there is an algorithm that solves the RSA problem for a given n and for a non-negligible fraction of choices of z, then there is another algorithm that solves the RSA problem for the same n for all choices of z.

There is also a random self reduction for the flexible RSA problem, at least in the particular version that we need for proving the security of our signature scheme. Just as for the RSA problem, this random self reduction adds credibility to the strong RSA assumption. This random self reduction appears to be new (or at least, not very well known), and can be derived from our proof of security of our signature scheme, and we present it in §4.

3 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

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

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 [LL94], 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 §6, one can obtain a scheme that is quite competitive with standard RSA in terms of efficiency (see §8).

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.*

Before proving this theorem, for convenience, we state the following well-known, but useful lemma (see, e.g., [GQ88]).

Lemma 1 *Given $x, y \in \mathbf{Z}_n^*$, along with $a, b \in \mathbf{Z}$, such that $x^a = y^b$ and $\gcd(a, b) = 1$, one can efficiently compute $\tilde{x} \in \mathbf{Z}_n^*$ such that $\tilde{x}^a = y$.*

To prove this lemma, we use the extended Euclidean algorithm to compute integers b' and k such that $bb' = 1 + ak$. A simple calculation then shows that $\tilde{x} = x^{b'} y^{-k}$ does the job. That completes the proof of Lemma 1.

Now we turn to the proof of Theorem 1. 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 [CD96]. 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 efficiently solve the RSA problem. 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 by applying the algorithm of Lemma 1.

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 $1 \leq i \leq t$, 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 efficiently solve the flexible RSA problem. 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 from 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 $d = \gcd(e, m)$. The fact that $\gcd(d, 2p'q') = 1$ implies that $y^{e/d} = z^{m/d}$, and so we can use the algorithm of Lemma 1 to compute an (e/d)-th root of z, which is nontrivial provided $e \nmid m$. So it suffices to show that $e \nmid m$ with non-negligible probability. Let r be a prime dividing e. Now, $r \nmid 2e' \prod_i e_i$ by construction. So it suffices to show that $r \nmid (a + H(x'))$ 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 $r \nmid p'q'$, it follows that $a + H(x') \equiv 0 \pmod{r}$ 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, $r \nmid (a + H(x'))$.

Using a universal one-way hash function

The notion of a universal one-way family of hash functions was introduced by Naor and Yung [NY89]. 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 $y \neq x$ 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 [BR97] 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 $1 \le j \le t$, $e = e_j$ and $(k, x') = (k_j, x'_j)$.

Type II For some $1 \le j \le t$, $e = e_j$ 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 $1 \le i \le t$, 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.

4 A Random Self Reduction for the Flexible RSA Problem

In this section, we show that the flexible RSA problem, at least a version of it that suffices for our purposes here, is random self reducible.

We assume that $n = pq$, where p and q are primes of the form $p = 2p' + 1$, $q = 2q' + 1$, where p' and q' are also prime. Also, the exponent r is restricted to be an odd number that is smaller than both p' and q'.

Now, we first define a problem related to the flexible RSA problem: the flexible QR-RSA problem. This is the same as the flexible RSA problem, except that z is chosen as a random element of QR_n. We note the following:

- using Lemma 1, the flexible RSA problem can be reduced to the flexible QR-RSA problem;
- the signature scheme in §3, as well as all of the variants presented here, can in fact be proven secure under the weaker assumption that the flexible QR-RSA problem is hard.

We now argue that the flexible QR-RSA problem is random self reducible. The argument here is a variant of the argument presented in the proof of Theorem 1 for the Type III forger. Suppose we have an algorithm A that for random $\tilde{z} \in \mathrm{QR}_n$ computes r and $\tilde{z}^{1/r}$, with $r > 1$ odd. Now, we are given an arbitrary quadratic residue z, and want to find a corresponding r and $z^{1/r}$. We can assume that z has order $p'q'$ modulo n, since otherwise either $z = 1$ and the problem is trivial, or $\gcd(z-1, n)$ splits n, and the problem can be efficiently solved using well-known techniques. Now we choose $k \in \{1, \ldots, n^2\}$ at random, and set $\tilde{z} = z^k$. Write $k = p'q'X + Y$. Now, the distribution of Y is statistically indistinguishable from the uniform distribution on $\{0, \ldots, p'q' - 1\}$. Further, the conditional distribution of X, given Y, is also statistically indistinguishable from the uniform distribution on $\{0, \ldots, \lfloor n^2/(p'q') \rfloor\}$. Thus, the value $\tilde{z}$ is (effectively) a random quadratic residue. Now we apply algorithm A to $\tilde{z}$, obtaining an r and $\tilde{y} = \tilde{z}^{1/r}$ (with some non-negligible probability). Let $d = \gcd(r, k)$. We want $d \neq r$, so that we can apply Lemma 1. But this will happen with non-negligible probability, independent of the adversary's computation: if s is a prime dividing r, then because of the above observation about the conditional distribution of X given Y, the probability that $Xp'q' + Y \equiv 0 \pmod{s}$ is roughly $1/s$. So we have $\tilde{y}^{r'} = z^{k'}$, where $r' = r/d > 1$ and $k' = k/d$. Since $\gcd(r', k') = 1$, we can apply the algorithm of Lemma 1 to compute $z^{1/r'}$, thus solving the flexible QR-RSA problem for the given value of z.

5 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 [KR] 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.

6 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 [Rab80] 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.* [DLP93], 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 [Ble96, Chapter 3], which are correct for primes up to $10^{16} > 2^{53}$. For example, one of Bleichenbacher's results, as reported in [Mau95], 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.* [BLS75], provides an effective proof of primality.

Lemma 2 *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 $0 \le y < 2P$, then $y^2 - 4x$ is neither 0 nor a perfect square.

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

Remarks. Note that this lemma differs from the one in [BLS75] 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 2 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 2 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 2. 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 [Mau95].

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 $1 \le m < e/8P^3$. 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 $2P^2 \cdot 2Pm_2 = 4P^3m_3 < e$, 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 $x = m_1m_2$. 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 2.

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

Lemma 3 *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 [BS96]. First, by Theorem 8.8.1 in [BS96], 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 [BS96], 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.

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.

We sketch in somewhat greater detail how the above idea can be implemented. Again, assume $l = 160$, and that we are using the above algorithm to generate $e = 2PR + 1$. We suggest two different methods for generating P and R.

Method 1. In this method, let us assume that we take a block cipher that works on 128-bit blocks and uses a 256-bit key. The block ciphers conforming to the forthcoming Advanced Encryption Standard satisfy these requirements. To the signer's public key, we add two randomly chosen 128-bit strings, v_P and v_R. To generate a prime $e = 2PR + 1$ as used in a signature, the signing algorithm proceeds as follows. To generate P, it chooses a 256-bit key K_P at random, applies the block cipher using this key to v_P, and from the output of the block cipher constructs a candidate P in a canonical way. This is repeated until we get a prime P. Similarly, the signing algorithm generates keys K_R until the output of the block cipher under key K_R yields a bit string that can be converted into a number R in a canonical way such that $e = 2PR + 1$ is prime. Instead of placing the prime e in the signature, the signer places the keys K_P and K_R in the signature, from which the verifier can easily derive and validate e.

To analyze the security of this scheme, we use the ideal cipher model, rather than the random oracle model. In the ideal cipher model, we assume that each block cipher key K indexes a random permutation which an adversary can evaluate as a "black box" in either the forward or reverse direction. See [KR96] for a more detailed description of of the ideal cipher model. Using a standard argument, given a random exponent $\hat{e}$ of the form $\hat{e} = 2\hat{P}\hat{R} + 1$, a simulator can easily "plant" the the values $\hat{P}$ and $\hat{R}$ in these black boxes. The exact location of these planted values is independent of the adversary's view, and so with non-negligible probability, the adversary's forgery will yield a $\hat{e}$-th root of a given number modulo n, thus solving a given instance of the standard RSA problem.

To show that the scheme is still secure in the "real world," one must make a pseudo-randomness assumption about the block cipher. This will ensure that the probability that the signer uses the same prime twice is still acceptably small.

Method 2. One disadvantage of the first method is that it requires an additional intractability assumption—namely, an appropriate pseudo-randomness assumption for the block cipher. This second method is slightly less efficient, but does not require this additional intractability assumption. Of course, the signer still needs a source of random, or pseudo-random bits, but that is a separate problem. The technique we propose is essentially the same as that proposed in [Sho00b]. In the public key, we place two keys, K_P and K_R. These keys are randomly chosen keys that index a family PIH of pair-wise independent hash functions mapping 384-bit strings to 128-bit strings. We also assume that we have two cryptographic ("magic") hash functions, MH_P and MH_R, which map 384-bit strings to 128-bit strings. These cryptographic hashes could be implemented using a standard hash function like MD5 or SHA-1, with random initial vectors IV_P and IV_R that are also stored in the public key. To generate a candidate for P, we select a random 384-bit string v_P, and compute the 128-bit string $PIH_{K_P}(v_P) \oplus MH_P(v_P)$, deriving a candidate for P in a canonical way. Likewise, to generate a candidate for R, we select a random 384-bit string v_R, and compute the 128-bit string $PIH_{K_R}(v_R) \oplus MH_R(v_R)$, deriving a candidate for R in a canonical way. When the signer finds P and R as above such that P and $e = 2PR + 1$ are prime, the signer places the values v_P and v_R in the signature, from which the verifier can easily derive and validate e.

Modeling MH_P and MH_R as random oracles, it is straightforward for a simulator to "plant" an RSA exponent as above, and thus use a forging algorithm to solve the standard RSA problem.

To analyze the security in the "real world," we make use of the Leftover Hash Lemma (a.k.a., the Smoothing Entropy Theorem). This lemma (see Chapter 8 of [Lub96] or also [IZ89]), together with our particular choice of parameters, implies that the values $PIH_{K_P}(v_P) \oplus MH_P(v_P)$ and $PIH_{K_R}(v_R) \oplus MH_R(v_R)$ have a distribution whose statistical distance from the random distribution on 128-bits is at most 2^{-128}. In deriving this, we need to make no additional intractability assumptions, and it follows from this that the probability that the signer ever uses the same prime twice is acceptably small.

7 Remarks on Key Generation

In the key generation step, we have to generate "strong primes" of the form $p = 2p' + 1$, where p' is also prime. The number p' is also known as a Sophie Germain prime. Very little has been actually proven about the density of such primes. In particular, it has not even been proven if there infinitely many such primes. Nevertheless, it is conjectured (see [BH62, BH65]) that the number of Sophie Germain primes not greater than x, which we denote $S(x)$, satisfies

$$S(x) \sim C \int_2^x dt/(\log t)^2, \tag{2}$$

where C is the constant defined by

$$C = \prod_r \left\{ \left(1 - \frac{1}{r}\right)^{-2} \left(1 - \frac{\omega(r)}{r}\right) \right\},$$

where the product is over all primes r, and $\omega(r)$ is the number of solutions to the congruence $x(2x+1) \equiv 0 \pmod{r}$; that is, $\omega(2) = 1$ and $\omega(r) = 2$ for all primes $r > 2$. Here, log denotes the natural logarithm. From [RS62], one can easily obtain the estimate

$$C \approx 1.32.$$

This conjecture is strongly supported by numerical evidence. Note that

$$\int_2^x dt/(\log t)^2 \sim x/(\log x)^2.$$

In order to generate such primes, we recommend the following procedure.

1. Generate a random, odd number p' of desired length, say k bits.
2. Test if either p' or $2p' + 1$ are divisible by any primes up to some bound B. If so, go back to step 1.
3. Test if 2 is a Miller-Rabin witness to the compositeness of p'. If so, go back to step 1.
4. Set $p = 2p' + 1$, and test if $2^{p'} \equiv \pm 1 \pmod{p}$. If not, go back to step 1.
5. Apply the Miller-Rabin test to p' some number t times using randomly selected bases to ensure an error probability of ϵ. A reasonable choice of ϵ is $\epsilon = 2^{-80}$.

We do not claim that this procedure is new. Algorithms very similar to this have sometimes been employed in practice. However, we are not aware of any formal analysis of this procedure in the literature.

The most important thing about this procedure is the "parallel trial division" done in step 2. This reduces the number of Miller-Rabin tests that need to performed by a significant amount compared to the number that would need to be performed if we naively performed trial division on p' alone. In practice, this parallel trial division can easily yield a factor of 10 speed-up over the naive method.

To measure the effectiveness of the parallel trial division in step 2, define $g(B)$ to be the probability that a random k-bit number (including even numbers) passes the trial division step, i.e., is not divisible by any primes up to B. For our purposes, we can assume that $B \leq k^{O(1)}$. Then Brun's pure sieve[1] (see equation (2.16) in Chapter 2 of [HR74]) implies that

$$g(B) = \prod_{r \leq B} \left(1 - \frac{\omega(r)}{r}\right) (1 + O(\exp(-\sqrt{k \log 2}))) + O(2^{-k/2}),$$

where the product is over primes r up to B. This implies that as B and k tend to infinity with $B \leq k^{O(1)}$,

$$g(B) \sim \frac{D}{(\log B)^2},$$

where D is a constant, $D \approx 0.416$ (see [RS62]).

Note that in step 4, we do not need to iterate the Miller-Rabin test. This is because if p' is prime, and the test in step 4 passes, then either 2 or -2 has multiplicative order $p-1$ modulo p, which implies that p is prime.

We have to consider the choice of number t of iterations of the Miller-Rabin test in step 5 of the above algorithm needed to get an overall error probability of ϵ for a given value of ϵ.

Consider the following: let p' be a randomly chosen, odd k-bit number, and let C_k be the event that p' is composite, P_k the event that p' is prime, and let $X_{k,t}$ be the event that p' passes t iterations of the Miller-Rabin test (with independent, randomly chosen bases). Also, let Y_k be the event that $2p'+1$ is not divisible by any primes up to B, and that $2^{p'} \equiv \pm 1 \pmod{2p'+1}$, and let S_k be the probability that p' is a Sophie Germain prime. Results in [DLP93] give explicit upper bounds for the probability $p_{k,t} = \Pr[C_k | X_{k,t}]$. Unfortunately, $p_{k,t}$ is not the relevant bound on the error probability. The relevant probability is $\Pr[C_k | X_{k,t} \wedge Y_k]$.

We have

$$\begin{aligned}
\Pr[C_k | X_{k,t} \wedge Y_k] &= \frac{\Pr[C_k \wedge X_{k,t} \wedge Y_k]}{\Pr[X_{k,t} \wedge Y_k]} \\
&\leq \frac{\Pr[C_k \wedge X_{k,t}]}{\Pr[X_{k,t} \wedge Y_k]} \\
&= p_{k,t} \frac{\Pr[X_{k,t}]}{\Pr[X_{k,t} \wedge Y_k]} \\
&\leq p_{k,t} \frac{\Pr[X_{k,t}]}{\Pr[S_k]}. \qquad (3)
\end{aligned}$$

Also, we have

$$\Pr[X_{k,t}] = \Pr[X_{k,t} \wedge C_k] + \Pr[X_{k,t} \wedge P_k] \leq \Pr[C_k | X_{k,t}] \Pr[X_{k,t}] + \Pr[P_k],$$

[1] Certainly stronger sieves are also applicable, but this is the simplest sieve which gives the result we need.

from which it follows that

$$\Pr[X_{k,t}] \le \frac{\Pr[P_k]}{1 - p_{k,t}}. \tag{4}$$

From (3) and (4), we have

$$\Pr[C_k | X_{k,t} \wedge Y_k] \le \frac{p_{k,t}}{1 - p_{k,t}} \cdot \frac{\Pr[P_k]}{\Pr[S_k]}. \tag{5}$$

Adapting the arguments and estimates used in the proof of Proposition 2 in [DLP93], one can show that for $k \ge 25$,

$$\Pr[P_k] \le 4/k. \tag{6}$$

Unfortunately, there are no known lower bounds on the density of Sophie Germain primes. However, in light of conjecture (2), it seems reasonable to conjecture that

$$S(x) - S(x/2) \ge \frac{1}{2} C \int_{x/2}^{x} dt/(\log t)^2 \tag{7}$$

for $x \ge 2^k$ and for all k of practical interest (e.g., $k \ge 500$). Such a conjecture, if true, would imply that

$$\Pr[S_k] \ge 2/k^2, \tag{8}$$

and hence

$$\Pr[C_k | X_{k,t} \wedge Y_k] \le \frac{p_{k,t}}{1 - p_{k,t}} \cdot 2k. \tag{9}$$

If one is willing to believe the above conjecture about the density of Sophie Germain primes, then we can choose t in the above algorithm so that

$$\frac{p_{k,t}}{1 - p_{k,t}} \cdot 2k \le \epsilon. \tag{10}$$

If one is not willing to believe this conjecture, then we can proceed as follows. In the above algorithm, we initialize a counter j to 1, and every time we execute the loop, we increment j. In step 5, we choose the iteration count t so that

$$\frac{p_{k,t}}{1 - p_{k,t}} \cdot \frac{4}{k} \le \frac{1}{5j^{1.25}} \epsilon. \tag{11}$$

To analyze the overall error probability of this algorithm, note that the above algorithm is logically equivalent to, though much more efficient than, the algorithm obtained by moving step 5 up to the top of the loop. In this transformed algorithm, for the jth loop iteration, consider the probability that p' is composite *and* passes the iterated Miller-Rabin test. The probability we are interested in, then, is $\Pr[C_k \wedge X_{k,t}]$, for t chosen as in (11). Using (4) and (6), along with (11), one can easily calculate that

$$\Pr[C_k \wedge X_{k,t}] \le \frac{p_{k,t}}{1 - p_{k,t}} \cdot \frac{4}{k} \le \frac{1}{5j^{1.25}} \epsilon. \tag{12}$$

Summing over all loop iterations, the probability that we *ever* make such an error is at most

$$\epsilon \cdot \frac{1}{5} \sum_{j=1}^{\infty} \frac{1}{j^{1.25}} \le \epsilon. \tag{13}$$

This second approach does not rely on any conjectures for its correctness, but it is slightly less efficient. We emphasize, however, that the increase in running time will be, in practice, quite negligible, and so we would recommend its use.

8 Implementation and Performance

The second author, together with Thomas Schweinberger and Mehdi Nassehi at IBM Research in Zurich, have designed and implemented a fully specified version of this signature scheme. The specification, along with a detailed, concrete security analysis will be available in a forthcoming paper.

All of the implementation ideas sketched in §3, §6, and §7 have been fully incorporated into the implementation. Moreover, the scheme makes use of the new universal one-way hash construction of [Sho00a], using SHA-1 as the underlying hash function. The assumption that one must make about SHA-1 is that the underlying compression function (mapping 672 bits to 160 bits) is second pre-image collision resistant. This means that given a random input, it is infeasible to find a different input that yields the same output. This is both qualitatively and quantitatively a much weaker assumption than the assumption that SHA-1 is collision resistant.

We report some preliminary measurements on the performance of the scheme. For short messages, the time to create or verify a signature is dominated by the public-key operations. For long messages, the time to hash the message must also be taken into account. However, the hash algorithm we employ, as described above, has essentially the same speed as that of SHA-1, so we do not address this any further here.

Our implementation is in the `C` programming language, and we used the publicly available GMP package (version 2.0.2) for long integer arithmetic, although we implemented our own sliding-window modular exponentiation routine as such a routine is not provided by GMP.

Our timing experiments were conducted on a PowerPC 604, model 43P-140, running AIX. As a base line, we first report the times to perform some basic arithmetic operations. The times to perform 512-bit modular multiplication, squaring, and exponentiation (with a 512-bit exponent) were

$$34\mu s,\ 32\mu s,\ \text{and } 18ms,$$

respectively. The corresponding times for 1024-bit numbers (and 1024-bit exponent) are

$$103\mu s,\ 100\mu s,\ \text{and } 118ms,$$

respectively.

In our implementation of the signature scheme, we break the signing algorithm up into two phases: the *key set-up phase*, and the *main signing phase*. The key set-up phase needs to be run just once for a given signing key, and it builds some tables that expedite the exponentiations performed in the main signing phase.

For a 1024-bit RSA modulus, we measured the time for the key set-up phase at $31ms$, and the time for the main signing phase at $50ms$. The time to verify a signature is $57ms$.

For the main signing phase, roughly one third of the time is spent generating a 161-bit prime (this is an average value which can vary substantially), and roughly one third of the time is spent in each of the two exponentiation operations.

These running times indicate that our signing algorithm is about 1.4 times slower than standard RSA, if we do not take into account the key set-up time, which may be appropriate in an environment where the same key will be used to sign many messages; if we take this key set-up time into account, then our signing algorithm is about 2.3 times slower than standard RSA.

It is a bit more difficult to compare the signature verification time of our scheme to that of standard RSA, since the size of the public exponent in a standard RSA signature may vary quite a bit. Certainly, if a very small exponent is used in standard RSA, this will be substantially faster than our scheme; however, if a full-length exponent is used, then our scheme is faster.

Of course, the running time of the signing and verification algorithms is not the only performance measurement. Other important characteristics include the size of a signature, as well as the size of a public key. Here, we must admit, the size of our signatures and public keys are several times the size of those in a standard RSA signature.

Another, though perhaps much less important, characteristic is the running time of the key generation algorithm. In our implementation, the average time to generate a public key/private key pair is $26s$. Because we have to generate Sophie Germain primes, this is unfortunately quite costly, but since this only affects the running time of the key generation algorithm, it is probably acceptable in most typical applications.

Acknowledgments

Thanks to Dan Boneh for pointing out the random self reducibility property discussed in §4.

References

[BH62] P. Bateman and R. Horn. A heuristic asymptotic formula concerning the distribution of prime numbers. *Math. Comp.*, 16:363–367, 1962.

[BH65] P. Bateman and R. Horn. Primes represented by irreducible polynomials in one variable. *Proc. Sympos. Pure Math.*, 8:119–135, 1965.

[Ble96] D. Bleichenbacher. *Efficiency and security of cryptosystems based on number theory.* PhD thesis, Swiss Federal Institute of Technology Zurich, 1996.

[BLS75] J. Brillhart, D. Lehmer, and J. Selfridge. New primality criteria and factorizations of $2^m \pm 1$. *Math. Comp.*, 29:620–647, 1975.

[BP97] 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.

[BR93] 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.

[BR97] M. Bellare and P. Rogaway. Collision-resistant hashing: towards making UOWHFs practical. In *Advances in Cryptology–Crypto '97*, 1997.

[BS96] E. Bach and J. Shallit. *Algorithmic Number Theory*, volume 1. MIT Press, 1996.

[CD96] R. Cramer and I. Damgard. New generation of secure and practical RSA-based signatures. In *Advances in Cryptology–Crypto '96*, pages 173–185, 1996.

[CGH98] R. Canetti, O. Goldreich, and S. Halevi. The random oracle model, revisted. In *30th Annual ACM Symposium on Theory of Computing*, 1998.

[DLP93] I. Damgard, P. Landrock, and C. Pomerance. Average case error estimates for the strong probable prime test. *Math. Comp.*, 61:177–194, 1993.

[DN94] C. Dwork and M. Naor. An efficient existentially unforegeable signature scheme and its applications. In *Advances in Cryptology–Crypto '94*, pages 218–238, 1994.

[FO99] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Advances in Cryptology–Crypto '97*, 1999.

[FS87] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology–Crypto '86, Springer LNCS 263*, pages 186–194, 1987.

[GHR99] 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.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17:281–308, 1988.

[GQ88] 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.

[HR74] H. Halberstam and H. Richert. *Sieve Methods*. Academic Press, 1974.

[IZ89] R. Impagliazzo and D. Zuckermann. How to recycle random bits. In *30th Annual Symposium on Foundations of Computer Science*, pages 248–253, 1989.

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

[KR96] J. Kilian and P. Rogaway. How to protect DES against exhaustive key search. In *Advances in Cryptology–Crypto '96*, pages 252–267, 1996.

[LL94] C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. In *Advances in Cryptology–Crypto '94*, pages 95–107, 1994.

[Lub96] M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton University Press, 1996.

[Mau95] U. Maurer. Fast generation of prime numbers and secure public-key cryptographic parameters. *J. Cryptology*, 8:123–155, 1995.

[NY89] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *21st Annual ACM Symposium on Theory of Computing*, 1989.

[PS96] D. Pointcheval and J. Stern. Provably secure blind signature schemes. In *Advances in Cryptology–Asiacrypt '96*, pages 252–265, 1996.

[Rab80] M. O. Rabin. Probabilistic algorithms for testing primality. *J. of Number Theory*, 12:128–138, 1980.

[RS62] J. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Ill. J. Math.*, 6:64–94, 1962.

[Sho00a] V. Shoup. A composition theorem for universal one-way hash functions. In *Advances in Cryptology–Eurocrypt 2000*, 2000.

[Sho00b] V. Shoup. Using hash functions as a hedge against chosen ciphertext attack. In *Advances in Cryptology–Eurocrypt 2000*, 2000.