Secure Hash-and-Sign Signatures Without the Random Oracle

Rosario Gennaro, Shai Halevi, and Tal Rabin

IBM T.J.Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, USA
{rosario,shaih,talr}@watson.ibm.com

Abstract. We present a new signature scheme which is existentially unforgeable under chosen message attacks, assuming some variant of the RSA conjecture. This scheme is not based on "signature trees", and instead it uses the so called "hash-and-sign" paradigm. It is unique in that the assumptions made on the cryptographic hash function in use are well defined and reasonable (although non-standard). In particular, we *do not* model this function as a random oracle.

We construct our proof of security in steps. First we describe and prove a construction which operates in the random oracle model. Then we show that the random oracle in this construction can be replaced by a hash function which satisfies some strong (but well defined!) computational assumptions. Finally, we demonstrate that these assumptions are reasonable, by proving that a function satisfying them exists under standard intractability assumptions.

Keywords: Digital Signatures, RSA, Hash and Sign, Random Oracle, Smooth Numbers, Chameleon Hashing.

1 Introduction

Digital signatures are a central cryptographic primitive, hence the question of their (proven) security is of interest. In [12], Goldwasser, Micali and Rivest formally defined the strongest notion of security for digital signatures, namely "existential unforgeability under an adaptive chosen message attack". Since then, there have been many attempts to devise practical schemes which are secure even in the presence of such attacks.

Goldwasser, Micali and Rivest presented a scheme in [12] which provably meets this definition (under some standard computational assumption). Their scheme is based on *signature trees*, where the messages to be signed are associated with the leaves of a binary tree, and each node in the tree is authenticated with respect to its parent. Although this scheme is feasible, it is not very practical, since a signature on a message involves many such authentication steps (one for each level of the tree). This was improved by Dwork and Naor [9] and Cramer and Damgård [7], who use "flat trees" with high degree and small depth, resulting in schemes where (for a reasonable setting of the parameters) it only takes about four basic authentication steps to sign a message. Hence in these schemes the time for generating a signature and its verification (and the size of the signatures)

J. Stern (Ed.): EUROCRYPT'99, LNCS 1592, pp. 123–139, 1999.

is about four times larger than in the RSA signature scheme, for which no such proof of security exist. Besides efficiency concerns, another drawback of these schemes is their "stateful" nature, i.e. the signer has to store some information from previously signed messages.

Another line of research concentrates on *hash-and-sign* schemes, where the message to be signed is hashed using a so called "cryptographic hash function" and the result is signed using a "standard signature scheme" such as RSA or DSA. Although hash-and-sign schemes are very efficient, they only enjoy a heuristic level of security: the only known security proofs for hash-and-sign schemes are carried out in a model where the hash function is replaced by a random oracle. It is hoped that these schemes remain secure as long as the hash function used is "complicated enough" and "does not interact badly" with the rest of the signature scheme. This "random oracle paradigm" was introduced by Bellare and Rogaway in [2], where they show how it can be used to devise signature schemes from any trapdoor permutation. They later described concrete implementations for the RSA and Rabin functions (with some security improvements) in [3]. Also, Pointcheval and Stern proved similar results with respect to ElGamal-like schemes in [15].

Security proofs in the random oracle model, however, can only be considered a heuristic. A recent result by Canetti, Goldreich and Halevi [5] demonstrates that "behaving like a random oracle" is not a property that can be realized in general, and that security proofs in the random-oracle model do not always imply the security of the actual scheme in the "real world". Although this negative result does not mean that the schemes in [2,3,15] cannot be proven secure in the standard model, to this day nobody was able to formalize precisely the requirements on the cryptographic hash functions in these schemes, or to construct functions that can provably replace the random oracle in any of them.

Our result. We present a new construction of a hash-and-sign scheme (similar to the standard hash-and-sign RSA), for which we can prove security in a standard model, *without a random oracle*. Instead, the security proof is based on a stronger version of the RSA assumption and on some specific *constructible* properties that we require from the hash function. At the same time, our scheme enjoys the same level of efficiency of typical hash-and-sign schemes. Compared to tree-based schemes this new algorithm fares better in terms of efficiency (typically 2.5 times faster), size of keys and signatures and does not require the signer to keep state (other than the secret signature key).

1.1 The New Construction

Our scheme resembles the standard RSA signature algorithm, but with a novel and interesting twist. The main difference is that instead of encoding the message in the base of the exponent and keeping the public exponent fixed, we encode the message in the exponent while keeping a fixed public base.

Set up. The public key is an RSA modulus $n = pq$ and a random element $s \in \mathbb{Z}_n^*$.

Signing. To sign a message M with respect to the public key (n, s), the signer first applies a hash function to compute the value $e = H(M)$, and then uses it as an exponent, i.e. he finds the e^{th} root of s mod n. Hence a signature on M is an integer σ such that $\sigma^{H(M)} = s \mod n$.

Assumptions and requirements. In our case, it is necessary to choose p, q as "safe" or "quasi-safe" primes (i.e., such that $(p-1)/2, (q-1)/2$ are either primes or prime powers.) In particular, this choice implies that $p-1, q-1$ do not have any small prime factors other than 2, and that finding an odd integer which is not co-prime with $\phi(n)$ is as hard as factoring n. This guarantees that extracting e^{th} roots when $e = H(M)$ is always possible (short of factoring n).

Intuitively, the reason that we can prove the security of our scheme without viewing H as a random oracle, is that in RSA the base must be random, but the exponent can be arbitrary. Indeed, it is widely believed that the RSA conjecture holds for any fixed exponent (greater than one). Moreover, if e_1, e_2 are two different exponents, then learning the e_1'th root of a random number s does not help in computing the e_2'th root of s, as long as e_2 does not divide e_1. Hence, it turns out that the property of H that is needed for this construction is that it is hard to find a sequence of messages $M_1, M_2, \ldots$ such that for some i, $H(M_i)$ divides the other $H(M_j)$'s. In the sequel, we call this property of the hash function *division intractability*.

In our scheme, an attacker who on input (n, s) can find both an exponent e and the e^{th} root of s, may have the ability to forge messages. Thus our formal security proof is based on the assumption that such a task is computationally infeasible. This stronger variant of the RSA assumption has already appeared in the literature, in a recent work of Barić and Pfitzmann for constructing fail-stop signatures without trees [1].

The proof. We present our proof in three steps:

1. First, we prove the security of the scheme in the random oracle model. This step already presents some technical difficulties. One of the main technical problems for this part is to prove that a random oracle is division-intractable. We prove this using some facts about the density of smooth numbers.
2. Next, we show that the random oracle in the proof of Step 1 can be replaced by a hash function which satisfies some (well defined) computational assumptions. We believe that this part is the main conceptual contribution of this work.

   We introduce a new computational assumption which is quite common in complexity theory, yet we are unaware of use of this type of assumptions in cryptography. Instead of assuming that there is no efficient algorithm that solves some problem, we assume that *there is no efficient reduction* between two problems. We elaborate on this issue in Subsection 5.2.
3. As we have introduced these non-standard assumptions, we need to justify that they are "reasonable". (Surely, we should explain why they are more reasonable than assuming that a hash function "behaves like a random oracle").

We do this by showing how to construct functions that satisfy these assumptions from any collision-intractable hash function [8] and Chameleon commitment scheme [4]. It follows, for example, that such functions exist if factoring is hard. As we explained above, this is in sharp contrast to the hash functions that are needed in previous hash-and-sign schemes, for which no provable construction is known.

2 Preliminaries

Before discussing our scheme, let us briefly present some notations and definitions which are used throughout the paper. In the sequel we usually denote integers by lowercase English letters, and strings by uppercase English letters. We often identify integers with their binary representation. The set of positive integers is denoted by $\mathcal{N}$.

Families of hash functions. We usually consider hash functions which map strings of arbitrary length into strings of a fixed length. In some constructions we allow these functions to be randomized. Namely, we consider functions of the type $h : \$ \times \{0,1\}^* \to \{0,1\}^k$ for some set of coins $\$$ and some integer k. We write either $h(X) = Y$ or $h(R;X) = Y$, where $R \in \$, X \in \{0,1\}^*$, and Y is the output of h on the the input X (and the coins R, if they are specified).

A family of hash function is a sequence $\mathcal{H} = \{H_k\}_{k\in\mathcal{N}}$, where each H_k is a collection of functions as above, such that each function in H_k maps arbitrary-length strings into strings of length k. The properties of such hashing families that are of interest to us, are *collision-intractability* which was defined by Damgård in [8], and *division-intractability* (which we define below). For the latter, we view the output of the hash function as the binary representation of an integer. For our scheme we use hash functions with the special property that their output is *always an odd integer*. Such a function can be easily obtained from an arbitrary hash function by setting $h'(X) = h(X)|1$ (or just setting the lowest bit of $h(X)$ to one).

Definition 1 (Collision intractability [8]). *A hashing family $\mathcal{H}$ is collision intractable if it is infeasible to find two different inputs that map to the same output. Formally, for every probabilistic polynomial time algorithm A there exists a negligible function* $\mathsf{negl}()$ *such that*

$$\Pr_{h\in H_k}[A(h) = \langle X_1, X_2\rangle \ \ s.t.\ X_1 \neq X_2 \text{ and } h(X_1) = h(X_2)] = \mathsf{negl}(k)$$

If h is randomized, we let the adversary algorithm A choose both the input and the randomness. That is, A is given a randomly chosen function h from H_k, and it needs to find two pairs $(R_1, X_1), (R_2, X_2)$ such that $X_1 \neq X_2$ but $h(R_1; X_1) = h(R_2; X_2)$.

Definition 2 (Division intractability). *A hashing family $\mathcal{H}$ is division intractable if it is infeasible to find distinct inputs $X_1, \ldots, X_n, Y$ such that $h(Y)$*

divides the product of the $h(X_i)$'s. Formally, for every probabilistic polynomial time algorithm A there exists a negligible function $\mathsf{negl}()$ *such that*

$$\Pr_{h\in H_k}\left[\begin{array}{l} A(h)=\langle X_1,\ldots,X_n,Y\rangle \\ \textit{s.t. } Y\neq X_i \textit{ for } i=1\ldots n, \\ \textit{and } h(Y) \textit{ divides the product } \prod_{i=1}^{n} h(X_i)\end{array}\right] = \mathsf{negl}(k)$$

Again, if h is randomized then we let A choose the inputs and the randomness. It is easy to see that a division intractable hashing family must also be collision intractable, but the converse does not hold.

Signature schemes. Recall that a signature scheme consists of three algorithms: a randomized key generation algorithm Gen, and (possibly randomized) signature and verification algorithms, Sig and Ver. The algorithm Gen is used to generate a pair of public and secret keys, Sig takes as input a message, the public and secret key and produces a signature, and Ver checks if a signature on a given message is valid with respect to a given public key. To be of any use, it must be the case that signatures that are generated by the Sig algorithm are accepted by the Ver algorithm. The strongest notion of security for signature schemes was defined by Goldwasser, Micali and Rivest as follows:

Definition 3 (Secure signatures [12]). *A signature scheme* $\mathcal{S}=\langle \mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver}\rangle$ *is* existentially unforgeable under an adaptive chosen message attack *if it is infeasible for a forger who only knows the public key to produce a valid (message, signature) pair, even after obtaining polynomially many signatures on messages of its choice from the signer.*

Formally, for every probabilistic polynomial time forger algorithm $\mathcal{F}$, there exists a negligible function $\mathsf{negl}()$ *such that*

$$\Pr\left[\begin{array}{l} \langle \mathsf{pk},\mathsf{sk}\rangle \leftarrow \mathsf{Gen}(1^k); \\ \textit{for } i=1\ldots n \\ \quad M_i \leftarrow F(\mathsf{pk},M_1,\sigma_1,\ldots,M_{i-1},\sigma_{i-1});\ \sigma_i \leftarrow \mathsf{Sig}(\mathsf{sk},M_i); \\ \langle M,\sigma\rangle \leftarrow F(\mathsf{pk},M_1,\sigma_1,\ldots,M_n,\sigma_n), \\ M\neq M_i \textit{ for } i=1\ldots n, \quad \textit{and } \mathsf{Ver}(\mathsf{pk},M,\sigma)=\textit{accept}\end{array}\right] = \mathsf{negl}(k)$$

3 The Construction

Key generation. The key-generation algorithm in our construction resembles that of standard RSA. First, two random primes p,q of the same length are chosen, and the RSA modulus is set to $n=p\cdot q$. In our case, we assume that p,q are chosen as "safe" or "quasi-safe" primes (i.e., that $(p-1)/2,(q-1)/2$ are either primes or prime-powers.) In particular, this choice implies that $p-1,q-1$ do not have any small prime factors other than 2, and that finding an odd integer which is not co-prime with $\phi(n)$ is as hard as factoring n. After the modulus n is set, an element $s\in\mathbb{Z}_n^*$ is chosen at random.

Finally, since we use a hash-and-sign scheme, a hash function has to be chosen from a hashing family. The properties that we need from the hashing family are

discussed in the security proof (but recall that we use a hash function whose output is always an odd integer). Below we view the hash function h as part of the public key, but it may also be a system parameter, so the same h can be used by everyone. The public key consists of n, s, h. The secret key is the primes p and q.

Signature and verification. To sign a message M, the signer first hashes M to get an odd exponent $e = h(M)$. Then, using its knowledge of p, q, the signer computes the signature σ as the e'th root of the public base s modulo n. If the hash function h is randomized, then the signature consists also of the coins R which were used for computing $e = h(R; M)$.

To verify a signature σ (resp. $\langle \sigma, R \rangle$) on message M with respect to the hash function h, RSA modulus n and public base s, one needs to compute $e = h(M)$ (resp. $e = h(R; M)$) and check that indeed $\sigma^e = s \pmod{n}$.

3.1 A Few Comments

1. Note that with overwhelming probability, the exponent $e = h(M)$ will be co-prime with $\phi(n)$. This is since finding an odd number e which is not co-prime with $\phi(n)$ is as hard as factoring n, for the class of moduli used in this scheme.

2. The output length of the hash function is relevant for the efficiency of the scheme. If we let the output of the hash function be $|n|$-bit long then signature generation will take roughly twice as long as standard RSA (since the signer must first compute $e^{-1} \bmod \phi(n)$ and then a modular exponentiation to compute σ). Also signature verification takes a full exponentiation modulo n. The efficiency can be improved by shortening the output length for h. However (as it will become clear from the proof of security), in order for h to be division intractable, its output must be sufficiently long. Our current experimental results suggest that to get equivalent security to a 1024-bit RSA, the output size of the hash should be about 512 bits. For this choice of hash output length we have that computing a signature will be less than 1.5 times slower than for a standard RSA signature.

3. When a key for our scheme is certified, it is possible for the signer to prove that the modulus n has been chosen correctly (i.e. the product of two quasi-safe primes) by using a result from [11].

4 Security in the Random-Oracle Model

As we have stated, for the security of our scheme we must use the "strong RSA conjecture" which was introduced recently by Barić and Pfitzmann. The difference between this conjecture and the standard RSA conjecture is that here the adversary is given the freedom to choose the exponent e. Stated formally:

Conjecture 4 (Strong-RSA [1]) *Given a randomly chosen RSA modulus n, and a random element $s \in \mathbb{Z}_n^*$, it is infeasible to find a pair $\langle e, r \rangle$ with $e > 1$ such that $r^e = s \pmod{n}$.*

The meaning of the "randomly chosen RSA modulus" in this conjecture depends on the way this modulus is chosen in the key generation algorithm. In our case, this is a product of two randomly chosen "safe" (or "quasi-safe") primes of the same length.

We start by analyzing the security of this construction in a model where the hash function h is replaced by a random oracle.[1]

Theorem 5. *In the random oracle model, the above signature scheme is existentially unforgeable under an adaptive chosen message attack, assuming the strong-RSA conjecture.*

Proof. Let $\mathcal{F}$ be a forger algorithm. We assume w.l.o.g. that $\mathcal{F}$ always queries the oracle about a message M before it either asks the signer to sign this message, or outputs (M, σ) as a potential forgery. Also, let v be some polynomial upper bound on the number of queries that $\mathcal{F}$ makes to the random oracle.

Using the same method as in Shamir's pseudo-random generator [17], we now show an efficient algorithm $\mathcal{A}_1$ (which we call *the attacker*), that uses $\mathcal{F}$ as a subroutine, such that if $\mathcal{F}$ has probability ϵ of forging a signature, then $\mathcal{A}_1$ has probability $\epsilon' \approx \epsilon/v$ of breaking the strong RSA conjecture.

The random-oracle attacker. The attacker $\mathcal{A}_1$ is given an RSA modulus n (chosen as in the key generation algorithm) and a random element $t \in \mathbb{Z}_n^*$, and its goal is to find e, r (with $e > 1$) such that $r^e = t \pmod{n}$.

First, $\mathcal{A}_1$ prepares the answers for the oracle queries that $\mathcal{F}$ will ask. He does so by picking at random v odd k-bit integers $e_1 \ldots e_v$ and an index $j \in \{1 \ldots v\}$. Intuitively, $\mathcal{A}_1$ bets on the chance that $\mathcal{F}$ will use its j'th oracle query to generate the forgery.[2]

Next, $\mathcal{A}_1$ prepares the answers for signature queries that $\mathcal{F}$ will ask. $\mathcal{A}_1$ computes $E = (\prod_i e_i)/e_j$ (i.e., E is the product of all the e_i's except e_j). If e_j divides E, then $\mathcal{A}_1$ outputs "failure" and halts. Otherwise, it sets $s = t^E \pmod{n}$, and initializes the forger $\mathcal{F}$, giving it the public key (n, s). The attacker then runs the forger algorithm $\mathcal{F}$, answering:
1. the i'th oracle query with the odd integer e_i. Namely, if the forger makes oracle queries $M_1 \ldots M_v$, then $\mathcal{A}_1$ answers these queries by setting $h(M_i) = e_i$.
2. signature query for message M_i for $i \neq j$ with the answer $\sigma_i = t^{E/e_i} \pmod{n}$ (recall that E/e_i is an integer for all $i \neq j$).

If $\mathcal{F}$ asks signature query for message M_j, or halts with an output other than (M_j, σ) then $\mathcal{A}_1$ outputs "failure" and halts. If $\mathcal{F}$ does output (M_j, σ) for which $\sigma^{e_j} = s \pmod{n}$, then $\mathcal{A}_1$ proceeds as follows. Using the extended Euclidean gcd algorithm, it computes $g = GCD(e_j, E)$, and also two integers a, b such that

[1] Also here, we assume that the random oracle always return an odd integer as output. Namely, the answer of the oracle on every given query is a randomly chosen odd k-bit integer.

[2] This is where we get the $1/v$ factor in the success probability. Interestingly, this factor *only shows up in the random oracle model*, so we get a tighter reduction in the standard model.

$ae_j + bE = g$. Then $\mathcal{A}_1$ sets $e = e_j/g$ and $r = t^a \cdot \sigma^b \pmod n$, and outputs (e, r) as its solution to this instance of the strong-RSA problem.

Analysis of $\mathcal{A}_1$. If $\mathcal{A}_1$ does not output "failure", then it outputs a correct solution for the strong RSA instance at hand (except with a negligible probability): First, since e_j does not divide E, then $g < e_j$, which means that $e = e_j/g > 1$. Moreover, we have

$$r^e = \left(t^a \cdot \sigma^b\right)^{e_j/g} = t^{ae_j/g} \cdot \sigma^{be_j/g} \stackrel{*}{=} t^{ae_j/g} \cdot t^{bE/g} = t^{(ae_j+bE)/g} = t \pmod n$$

Equality $(*)$ holds because: (a) $\sigma^{e_j} = s = t^E \pmod n$, which implies that also $\sigma^{be_j} = t^{bE} \pmod n$; and (b) e_j is co-prime with $\phi(n)$ (except with negligible probability), which means that so is g. Therefore, there is a single element $x \in \mathbb{Z}_n^*$ satisfying $x^g = \sigma^{be_j} = t^{bE} \pmod n$.

It is left to show, therefore, that the event in which $\mathcal{A}_1$ does not output "failure" happens with probability $\epsilon' \approx \epsilon/v$. Denote by DIV the event in which e_j divides E. Conditioned on the complement of DIV, $\mathcal{F}$ sees the same transcript when interacting with $\mathcal{A}_1$ as when it interacts with the real signer, and so it outputs a valid forgery for M_j with probability ϵ/v (since j is chosen at random between 1 and v). It follows that the probability that $\mathcal{A}_1$ does not output "failure" is $\epsilon' \geq \epsilon/v - \Pr[DIV]$. In Lemma 6 we prove that when the output length of the random oracle is k, then $\Pr[DIV] \leq 2^{-\sqrt{k}}$, which completes the proof of Theorem 5.

Lemma 6. *Let $e_1 \ldots e_v$ be random odd k-bit integers, let j be any integer $j \in \{1 \ldots v\}$, and denote $E = (\prod_i e_i)/e_j$. Then, the probability that e_j divides E is less than $2^{-\sqrt{k}}$.*

Proof. As before, we denote the above event by DIV. To prove Lemma 6, we use some facts about the density of smooth numbers. Recall that when x, y are integers, $0 < y \leq x$, we say that x is *y-smooth* if all the prime factors of x are no larger than y, and let $\Psi(x, y)$ denote the number of integers in the interval $[0, x]$ which are y-smooth. The following fact can be found in several texts on number-theory (e.g., [14]).

Proposition 7. *Fix some real number $\epsilon > 0$, let x be an integer $x \geq 10$, let y be another integer such that $\log x > \log y \geq (\log x)^\epsilon$, and denote $\mu \stackrel{\text{def}}{=} \log x / \log y$ (namely, $y = x^{1/\mu}$). Then $\Psi(x, y)/x = \mu^{-\mu(1-f(x,\mu))}$, where $f(x, \mu) \to 0$ as $\mu \to \infty$, uniformly in x.*

Below we write somewhat informally $\Psi(x, x^{1/\mu}) = \mu^{-\mu(1-o(1))}$. Substituting 2^k for x and $\sqrt{k}/2$ for μ in the expression above, we get

$$\Psi(2^k, 2^{2\sqrt{k}})/2^k = \left(\sqrt{k}/2\right)^{-\sqrt{k}(1-o(1))/2} < 2^{-\sqrt{k}\log k/16} < 2^{-2\sqrt{k}}.$$

We comment that the same bound also holds when we talk about odd k-bit integers, (this can be shown using the fact that an even k-bit integer x is smooth if and only if the $(k-1)$-bit integer $x/2$ is also smooth). If we denote by $SMOOTH$

the event in which the integer e_j is $2^{2\sqrt{k}}$-smooth, then by the bound above, $\Pr[SMOOTH] \leq 2^{-2\sqrt{k}}$.

Assume, then, that the event $SMOOTH$ does not happen. Then e_j has at least one prime factor $p > 2^{2\sqrt{n}}$. In this case, the probability that e_j divides the product of the other e_i's is bounded by the probability that at least one of these e_i's is divisible by p. But since all the other e_i's are chosen at random, then the probability that any specific e_i is divisible by p is at most $1/p < 2^{-2\sqrt{k}}$, and the probability that there exists one which is divisible by k is at most $v \cdot 2^{-2\sqrt{k}}$. As v is polynomial in k, we get $v \cdot 2^{-2\sqrt{k}} < 2^{-1.5\sqrt{k}}$. Combining the two bounds, we get $\Pr[DIV] < \Pr[SMOOTH] + \Pr[DIV \mid \neg SMOOTH] < 2^{-2\sqrt{k}} + 2^{-1.5\sqrt{k}} < 2^{-\sqrt{k}}$.

4.1 The Value of k

The above bound on $\Pr[DIV]$ is very weak. For example, to get security level of 2^{-80}, this bound suggest a value of $k \approx 6000$. Although the equations above can be optimized, they still only give a very crude bound. One reason for this is that we only bound the probability that p, the largest prime factor of e_j, divides the product of the e_i's. If e_j is rather smooth, then e_j/p is still rather large, so even if p divides one of the e_i's, the probability that e_j/p divides the product of the e_i's is still rather small. We therefore performed some experiments to get a practical estimate for the value of k. Our experiments suggest that $\Pr[DIV]$ is in fact much smaller than the bound $2^{-\sqrt{k}}$. (In fact, for the values of k which we tested, we got $\Pr[DIV] \approx 2^{-k/8}$.) See more details in Appendix A.

5 Eliminating the Random Oracle

Below we show that the random oracle in the above proof can be replaced by a randomized hash function with certain properties. Clearly, this hash function should be division-intractable, since violating division intractability immediately yields an attack on the signature scheme. However, this property alone is not sufficient: even if we assume that the hash function is division intractable, we still face problems carrying out the above security proof in the standard model. Specifically, recall that in the previous proof, the attacker $\mathcal{A}_1$ had to simulate the signer for $\mathcal{F}$, and do it without knowing the prime factorization of the modulus.

$\mathcal{A}_1$ was able to carry out this task since it could choose the outputs of the oracle (the e_i's) before seeing the inputs, and so it was able to "tailor" the public base s to these specific e_i's. In a standard model this is no longer the case: Clearly, if h is deterministic, then the forger's choice of M_i's uniquely determines the e_i's, and the attacker has no room to play with these values. But even if h is randomized this does not help the attacker due to the fact that h is also division-intractable which implies that it is one-way. Thus, if the attacker first chooses e and then sees M, it cannot find randomness R for which $e = h(R; M)$ (even if such R exists).

As a first step towards overcoming this difficulty, we note that the hardness of finding such randomness R is in some sense "unrelated" to the hardness of solving

the strong RSA problem. Namely, our intuition is that being able to find R should not help anyone solving strong RSA.[3] We formalize this intuition by replacing the strong RSA conjecture (which asserts that there is no efficient algorithm to solve strong RSA), with the "funny looking" conjecture which asserts that there is *no efficient reduction* between finding the randomness R and solving strong RSA. Technically, this is done by asserting that the strong RSA conjecture remains valid *even in a relativized world where there is an oracle that finds this randomness.*

5.1 The Hashing Family

To be able to carry the security proof in a standard world, we have to make the following assumptions on the hashing family $\mathcal{H}$ used in the scheme and its relation to the strong RSA conjecture.

We say that a hashing family $\mathcal{H}$ is *suitable* if

1. For any $h \in \mathcal{H}$, the outputs of h are always odd integers.
2. $\mathcal{H}$ is division-intractable.
3. For every $h \in \mathcal{H}$ and every two messages M_1, M_2, the distributions $h(R; M_1)$, $h(R; M_2)$, induced by the random choice of R, are statistically close.[4]
4. The strong RSA conjecture also holds in a model where there exists an oracle that on input h, M, e, returns a random $R \in \$$ subject to $h(R; M) = e$. [5]

We discuss these assumptions further in Section 5.2 below. But first let us prove that our signature scheme is secure when using a suitable hashing family $\mathcal{H}$. We stress that although one of our computational assumptions holds in a relativized world, we then prove the security of the scheme in the "real world".

Theorem 8. *If $\mathcal{H}$ is suitable, then the construction from Section 3 is existentially unforgeable under an adaptive chosen message attack.*

Proof. The proof proceeds similarly to the proof of Theorem 5, i.e. we construct an attacker $\mathcal{A}_2$ which will use the forger $\mathcal{F}$. The main difference is that the attacker A_2 operates in a relativized model, given in addition access to the oracle from Condition 4 of the suitable hash function. We show that if the forger $\mathcal{F}$ has probability ϵ of breaking the scheme (in the "real world"!) then the attacker has probability $\epsilon' \approx \epsilon$ of solving strong RSA in the relativized world. (Note that this reduction is tighter than the reduction in the random-oracle model.)

The oracle-assisted attacker. We again assume a bound of v on the number of signatures that the forger $\mathcal{F}$ asks to see before it outputs its forgery. As before, A_2 is given an RSA modulus n and a random element $t \in \mathbb{Z}_n^*$ (chosen as in the key generation algorithm), and its goal is to find e, r (with $e > 1$) such

[3] For example, if one thinks of h as SHA-1, then we have a very strong intuition that finding collisions in SHA-1 provides no help in violating the RSA conjecture.

[4] Together with the collision-intractability, this implies that $\mathcal{H}$ is a statistically hiding string-commitment scheme.

[5] Such R must exist because of Condition 3.

that $r^e = t \pmod n$. It starts by picking at random a hash function $h \in \mathcal{H}$ to be used for the forger. And v arbitrary values $e_1, ..., e_v$ in the range of the function. This can be done for example by picking v arbitrary "dummy messages" $M'_1 \dots M'_v$ and computing $e_i = h(R'_i; M'_i)$ (for random R'_i's).

Then A_2 computes $E = \prod_i e_i$, sets $s = t^E \pmod n$ and initializes $\mathcal{F}$ with the public key (n, s, h). Whenever $\mathcal{F}$ asks for a signature on a message M_i, A_2 queries its randomness-finding oracle for a randomness R_i for which $h(R_i; M_i) = e_i$, and then computes the signature by setting $\sigma_i = t^{E/e_i} \pmod n$. A_2 returns the pair $\langle R_i, \sigma_i \rangle$ to $\mathcal{F}$.

It is important to note that because of Condition 3 on $\mathcal{H}$, the distribution that $\mathcal{F}$ sees in this simulation is statistically close to the distribution it sees when interacting with the real signer. In particular, since $\mathcal{H}$ is division intractable, then $\mathcal{F}$ has only a negligible probability of finding M', R' such that $e' = h(R'; M')$ divides the product of the e_i's.

It follows that with probability $\epsilon' \geq \epsilon - \mathsf{negl}$, $\mathcal{F}$ outputs a forgery M', R', σ such that $e' = h(R'; M')$ does not divide the product of the other e_i's, and yet $\sigma^{e'} = s \pmod n$. When this happens, the attacker A_2 uses the same gcd procedure as above to find (e, r) with $e > 1$ such that $r^e = t \pmod n$.

5.2 Discussion

The proof in the previous section eliminates the random oracle, but substitutes it with a non-standard assumption: the strong RSA assumption must still be true even in a relativized world where finding randomness for h is not hard. Is this a more reasonable assumption than just assuming that h "behaves like a random oracle"? We strongly believe it is. The assumption we use has a very concrete interpretation in the real world, meaning that there is no reduction from the problem of randomness-finding for h to the problem of solving the strong RSA problem. In other words the difficulty of the two problems are somewhat "independent". Moreover we show later that suitable families of hash functions are actually constructible. On the other hand the notion of "behaving as a random oracle" has no concrete counterpart in the real world, and there are no provable constructions of "good hash functions" for previously known schemes.

It is interesting to ask if our technique of substituting the random oracle in the security proof with a relativized assumption can be used in other proofs that employ random oracles (such as [2,3,15]). Unfortunately, it does not appear to be likely. The main reason our technique seems to fail in those proofs, is that their requirement from h is that the forger cannot find a message M for which he "knows" something about $h(M)$. In our scheme instead we were able to pin down the specific combinatorial property we require from h and flesh it out as a specific assumption.

In the next section we describe a construction of a suitable family of hash functions. The main purpose of this construction is to prove that the assumptions we make can be realized. However the construction requires the signer to search for a prime exponent in a large subset and thus it might require a significant amount of time. It is however plausible to conjecture that families built from

widely used collision-resistant hash functions such as SHA-1 [10] can be suitable. The rationale is that such functions have been designed in a way that destroys any "structure" in the input-output relationship. In particular it is very unlikely (although we don't know how to prove it) that division intractability does not hold for such functions. A possible candidate would be to define h as following

$$h(R_1; R_2; R_3; R_4; M)$$
$$= 1 \,|\, SHA1(M|1|R_1) \,|\, SHA1(M|2|R_2) \,|\, SHA1(M|3|R_3) \,|\, SHA1(M|4|R_4) \,|\, 1$$

for a 642-bit exponent (this is the definition of a single h, a family could be constructed via any standard method of extending SHA-1 to a family, for example by keying the IV).

6 Implementing the Hashing Family $\mathcal{H}$

To argue that Conditions 1-4 are "reasonable" we at least need to show that they could be met. Namely, that there exists a function family $\mathcal{H}$ satisfying these conditions (under some standard assumptions). Below we show that such families exist, assuming that collision-intractable families and Chameleon commitment families exist. In particular, it follows that such families exist under the factoring conjecture (which is weaker than our "strong RSA" conjecture), or under the Discrete-log conjecture.[6]

We construct $\mathcal{H}$ in two steps: first we show how to transform any collision-intractable hashing family into a (randomized) division-intractable family, and then we show how to take any division-intractable hash function and transform it into one that satisfy Conditions 1 through 4.

6.1 From Collision-Intractable to Division-Intractable

The idea of this transformation is simply to force the output of the hash functions to be a prime number. Then, the function is division-intractable if and only if it is collision intractable. A heuristic for doing just that was suggested by Barić and Pfitzmann in [1]: If h is collision intractable with output length k, then define a randomized function $\tilde{h}$ with output length of (say) $2k$ bits, by setting for $r = 0, \ldots, 2^k - 1$, $\tilde{h}(r; X) = 2^k \cdot h(X) + r$, provided that $h(X) + r$ is an odd prime ($\tilde{h}(r; X)$ is undefined otherwise).

It is obvious that $\tilde{h}$ is still collision-intractable, and that it always outputs primes, so it is also division-intractable. However, to argue that $\tilde{h}$ is efficiently computable, we must assume that the density of primes in the interval $[2^k h(X), 2^k(h(X)+1)]$ is high enough (say, $1/poly(k)$ fraction). Hence, to use this heuristic,

[6] The way we set the definitions in this paper, Condition 4 on $\mathcal{H}$ implies the strong RSA conjecture, so formally there is no point in using any other conjecture. This technicality can be dealt with in some ways, but we chose to ignore it in this preliminary report.

one must rely on some number-theoretic conjecture about the density of primes in small intervals.

Below we show a simple technique that allows us to get rid of this extra conjecture: Just as in the above heuristic, we let the output size of $\tilde{h}$ be larger than that of h (letting $\tilde{h}$ output $3k$ bits is sufficient for our purposes), and partition the space of outputs in such a way that each output of the original h is identified with a different subset of the possible outputs of $\tilde{h}$. However, we choose the partition in a randomized manner, so we can prove that (with high probability) each one of the subsets is dense with primes.

The main tool that we use in this transformation is *universal hashing families* as defined by Carter and Wegman in [6]. Recall that a universal family of hash functions from a domain D to a range R is a collection U of such functions, such that for all $X_1 \neq X_2 \in D$ and all $Y_1, Y_2 \in R$, $\Pr_f[f(X_1) = Y_1 \text{ and } f(X_2) = Y_2] = 1/|R|^2$ (the probability is taken over the uniformly random choice of $f \in U$). Several constructions of such universal families were described in [6].

In our case, we use universal hash functions which maps $3k$ bits to k bits, with the property that given a function $f \in U$ and a k-bit string Y, it is possible to efficiently sample uniformly from the space $\{X \in \{0,1\}^{3k} \ : \ f(X) = Y\}$. For any function $f : \{0,1\}^{3k} \to \{0,1\}^k$, we associate a partition of the set of outputs ($\{0,1\}^{3k}$) into 2^k subsets according to the values assigned by f. Each output value of the original h (which is a k-bit string Y) is then associated with the subset $f^{-1}(Y)$. The modified function $\tilde{h}$, on input X, outputs a random odd prime from the set $f^{-1}(h(X))$. Again, it is clear that $\tilde{h}$ is collision-intractable if h is, and that it only outputs primes, hence it is division-intractable. On the other hand, a standard hashing lemma shows that with high probability over the random choice of f, the subset $f^{-1}(h(X)) \subset \{0,1\}^{3k}$ is dense with primes (for all X). Thus, $\tilde{h}$ is also efficiently computable.

Lemma 9. *Let U be a universal family from $\{0,1\}^{3k}$ to $\{0,1\}^k$. Then, for all but a 2^{-k} fraction of the functions $f \in U$, for every $Y \in \{0,1\}^k$ a fraction of at least $1/ck$ of the elements in $f^{-1}(Y)$ are primes, for some small constant c.*
Proof omitted.

6.2 From Division-Intractable to Suitable

Finally, we show how to take any division-intractable hashing family (that always output odd integers) and transform it into a suitable one (i.e. one that satisfies Conditions 1 through 4 from Subsection 5.1). To this end, we use *Chameleon commitment schemes*, as defined and constructed by Brassard, Chaum and Crepeau [4]. In fact we use them as Chameleon Hashing exactly as defined and required in [18].

The Chameleon Hashing is a function $ch(\cdot;\cdot)$ which on input a random string R and a message M is easily computed. Furthermore, it is associated with a value known as the "trapdoor". It satisfies the following properties:

- Without knowledge of the trapdoor there is no efficient algorithm that can find pairs M_1, R_1 and M_2, R_2 such that $ch(M_1, R_1) = ch(M_2, R_2)$.

- There is an efficient algorithm that given the trapdoor, a pair M_1, R_1 and M_2 can compute R_2 such that $ch(M_1, R_1) = ch(M_2, R_2)$.
- For any pair of messages M_1, M_2 and for randomly chosen R the distribution $ch(M_1, R_1)$ and $ch(M_2, R_2)$ are statistically close.

To transform a division intractable hash function h into one that also satisfies Conditions 3 and 4 from Subsection 5.1, we simply apply it to the hash string $c = ch(R; M)$ instead of to the message M itself. A little more formally, we have the following construction.

Let $\mathcal{H}$ be a division-intractable family, and let CH be a Chameleon hashing scheme. We construct a randomized family $\tilde{\mathcal{H}}$ in which each function is associated with a function $h \in \mathcal{H}$ and an instance $ch \in CH$. We denote this by writing $\tilde{h}_{h,ch}$. This function is defined as $\tilde{h}_{h,ch}(R; M) = h(ch(R; M))$. (if h itself is randomized, then we have $\tilde{h}_{h,ch}(R_1, R_2; M) = h(R_2; ch(R_1; M))$). It is easy to see that $\tilde{\mathcal{H}}$ enjoys the following properties

1. $\tilde{\mathcal{H}}$ always outputs odd integers if $\mathcal{H}$ does.
2. $\tilde{\mathcal{H}}$ is collision intractable, since violating division-intractability requires either finding two different messages with the same hash string, or violating the division-intractability of $\mathcal{H}$.
3. $\tilde{\mathcal{H}}$ is a statistically hiding hashing scheme (since CH is, and $\mathcal{H}$ is collision intractable).

It is left to show that $\tilde{\mathcal{H}}$ also satisfies the last condition. This is shown in the following proposition:

Proposition 10. *If the Strong RSA conjecture holds, then it also holds in a relativized world where there is a randomness-finding oracle for $\tilde{\mathcal{H}}$.*

Proof. We need to show that an efficient algorithm for solving strong RSA in a relativized world where there is a randomness-finding oracle for $\tilde{\mathcal{H}}$ can be used to solve strong RSA also in the "real world". To do that, we use the trapdoor for the chameleon hashing scheme to implement the randomness-finding oracle in the real world.

A little more precisely, if there exists an efficient reduction algorithm A that solves strong RSA in the relativized world, then we construct an efficient algorithm that solves strong RSA (without the oracle) by picking a Chameleon hashing instance *ch together with its trapdoor*. Now, we execute the algorithm $\mathcal{A}$, and whenever the forger asks a query concerning the hash, $\mathcal{A}$ turns to the randomness-finding oracle, which uses the randomness-finding algorithm of CH with the trapdoor to answer that query.

Since Chameleon hashing exists based on the factoring conjecture (which, in turn, is implied by the strong RSA conjecture) we have

Corollary 11. *Under the Strong RSA conjecture, suitable hashing families exist.*

7 Conclusions

We present a new signature scheme which has advantages in terms of both security and efficiency. In terms of efficiency, this scheme follows the "hash-and-sign" paradigm, i.e. the message is first hashed via a specific kind of hash function and then an RSA-like function is applied. Thus, in total the scheme requires a hashing operation and the only one modular exponentiation. These is no need to maintain trees and to rely on some stored information on the history of previous signatures.

The security of the scheme is based on two main assumptions. One is the "strong RSA" assumption: although this assumption has already appeared previously in the literature, it is still quite new and we think it needs to be studied carefully. The other assumption is the existence of division-intractable hash functions. We showed that such functions exist and that efficient implementations (like the one in Section 5.2) are possible based on conjectures which seem to be supported by experimental results and which we invite the research community to explore. In any case the proof of security is still based on *concrete* computational assumptions rather than on idealized models of computation (like the random oracle model).

References

1. N. Barić, and B. Pfitzmann. Collision-free accumulators and Fail-stop signature schemes without trees. In *Advances in Cryptology - Eurocrypt '97*, LNCS vol. 1233, Springer, 1997, pages 480-494.
2. M. Bellare and P. Rogaway. Random Oracles are Practical: a Paradigm for Designing Efficient Protocols. In *1st Conf. on Computer and Communications Security*, ACM, pages 62–73, 1993.
3. M. Bellare and P. Rogaway. The Exact Security of Digital Signatures: How to Sign with RSA and Rabin. In Advances in Cryptology – Eurocrypt '96, LNCS vol. 1070, Springer-Verlag, 1996, pages 399-416.
4. G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *JCSS*, 37(2):156–189, 1988.
5. R. Canetti, O. Goldreich and S. Halevi. The Random Oracle Methodology, Revisited. STOC '98, ACM, 1998, pages ???-???.
6. L. Carter and M. Wegman. Universal Hash Functions. J. of Computer and System Science 18, 1979, pp. 143-154.
7. R. Cramer and I. Damgård. New generation of secure and practical RSA-based signatures. In Advances in Cryptology – CRYPTO '96, LNCS vol. 1109, Springer-Verlag, 1996, pages 173-185.
8. I. Damgård. Collision free hash functions and public key signature schemes. In *Advances in Cryptology - Eurocrypt '87*, LNCS vol. 304, Springer, 1987, pages 203-216.
9. C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. In *J. of Cryptology*, 11(3), Summer 1998, pp. 187–208
10. National Institute for Standards and Technology. Secure Hash Standard, April 17 1995.

11. R. Gennaro, D. Micciancio, and T. Rabin. An Efficient Non-Interactive Statistical Zero-Knowledge Proof System for Quasi-Safe Prime Products. Proceedings of 1998 ACM Conference on Computers and Communication Security.
12. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.
13. National Institute of Standards and Technology. Digital Signature Standard (DSS), Technical report 169, August 30, 1991.
14. A.K. Lenstra and H.W. Lenstra, Jr. Algorithms in number theory. In Handbook of theoretical computer science, Volume A (Algorithms and Complexity), J. Van Leeuwen (editor), MIT press/Elsevier, 1990. Pages 673-715.
15. D. Pointcheval and J. Stern. Security Proofs for Signature Schemes. In *Advances in Cryptology – Proceedings of EUROCRYPT'96*, LNCS vol. 1070, Springer-Verlag, pages 387–398.
16. R. Rivest, A. Shamir and L. Adelman. A Method for Obtaining Digital Signature and Public Key Cryptosystems. *Comm. of ACM*, 21 (1978), pp. 120–126
17. A. Shamir. On the generation of cryptographically strong pseudorandom sequences. ACM Trans. on Computer Systems, 1(1), 1983, pages 38-44.
18. H.Krawczyk and T.Rabin. Chameleon Hashing and Signatures. manuscript.

A Experimental Results

Here we describe the results of some experiments which we performed to estimate the "true complexity" of the division property. We tried to measure how many random k-bit integers need to be chosen until we have a good chance of finding one that divides the product of all the others.

We carried out these experiments for bit-lengths 16 through 96 in increments of 8 (namely $k = 16, 24, \ldots, 88, 96$). For each bit length we performed 200 experiments in which we counted how many random integers of this length were chosen until one of them divides the product of the others. For each length, we took the second-smallest result (out the of the 200 experiments) as our estimate for the number of integers we need to choose to get a 1% chance of violating the division-intractability requirement.[7]

We repeated this experiment twice: in one experiment we chose random k-bit integers, and in the other we forced the least- and most-significant bits to '1'. The results are described in Figure 1. It can be seen that the number of integers seems to behave exponentially in the bit-length. Specifically for the bit-lengths $k = 16 \ldots 96$, it seems to behave more or less as $2^{k/8}$ (in fact even a little more). Forcing the low and high bits to '1' seems to increase the complexity slightly.

[7] Taking the 2nd-smallest of 200 experiments seems like a slightly better estimate than taking the smallest of 100 experiments, and our equipment didn't allow us to do more than 200 experiments for each bit-length.

Bit length	16	24	32	40	48	56	64	72	80	88	96
random	5	9	23	50	151	307	691	1067	2786	3054	8061
msb=lsb=1	8	17	39	63	160	293	710	1472	3198	4013	8124

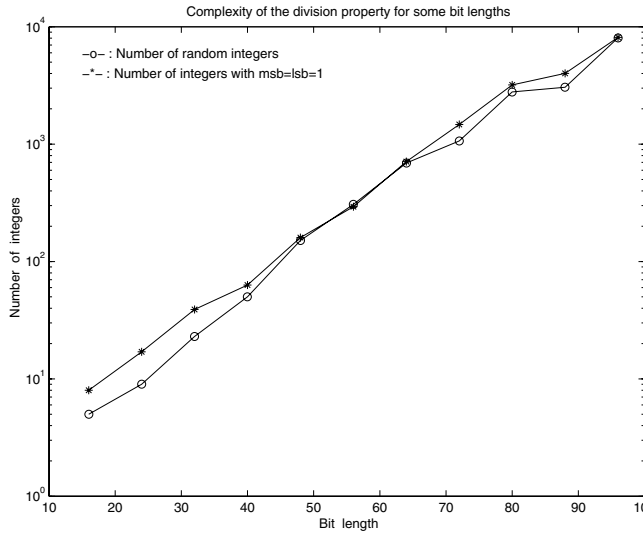

Fig. 1. Experimental results. The line –o– describes the number of random k-bit integers, and the line –⋆– describes the number of random k-bit integers with the first and last bits set to '1'.