VSH, an Efficient and Provable Collision Resistant Hash Function

Scott Contini[1], Arjen K. Lenstra[2], and Ron Steinfeld[1]

[1] Macquarie University
[2] Lucent Technologies Bell Laboratories and Technische Universiteit Eindhoven

Preliminary version 3.31

Abstract. We introduce VSH, *very smooth hash*, a new hash function for which finding collisions is provably reducible to finding nontrivial modular square roots of very smooth numbers modulo a composite integer n. By very smooth, we mean that the smoothness bound is some fixed polynomial function of the bitlength N of n.
We show that if collisions for VSH can, asymptotically, be found faster than factoring n using the Number Field Sieve factoring algorithm (NFS), then n can be factored faster than by means of the NFS. Furthermore, we show how our asymptotic argument can be turned into a practical method to select n so that VSH meets a desired security level. Our hardness assumption—and thereby the collision resistance of VSH—is thus linked to the current state of the art of integer factorisation.
The fastest variant of VSH is theoretically pleasing because it requires only a constant number of multiplications modulo n per N message bits. It is also practical. A preliminary implementation on a 1GHz Pentium III processor that achieves collision resistance at least equivalent to the difficulty of NFS factoring of a 1024-bit RSA modulus, runs at more than 1.1 MegaByte per second, with a moderate slowdown to 0.7MB/s for 2048-bit RSA security.
We also show how to constructively use the factorisation trapdoor of VSH to build a fast and provably secure randomised trapdoor hash function, which has applications in speeding up provably secure signature schemes (including the Cramer-Shoup signature scheme) and designated-verifier signatures.

1 Introduction

Current collision resistant hash algorithms that have provable security reductions are too inefficient to be used in practice. One example [12, 15] that is provably reducible to integer factorisation is of the form

$$x^m \bmod n$$

where m is the message, n a supposedly hard to factor composite, and x is some pre-specified base value. Given a collision $x^m \equiv x^{m'} \bmod n$, we learn that

$m - m'$ is a multiple of the order of x (which is in itself a divisor of $\phi(n)$). Such information can be used to factor n in polynomial time assuming certain properties of x.

Since the above algorithm requires on average 1.5 (multiprecision) multiplications modulo n per message-bit processed, it is quite inefficient. It seems that so far all attempts to gain efficiency came at the cost of losing provability (see also [1]). In this document, we propose a more efficient hash algorithm where finding any collision (i.e., strong collision resistance) is provably as difficult as finding nontrivial modular square roots of *very* smooth numbers modulo a composite n, abbreviated NMSRVS. By very smooth, we mean that the smoothness bound is some fixed polynomial function of the bitlength N of n. We refer to our new hash as VSH, for *very smooth hash*.

Although it appears that the NMSRVS assumption has not been used for security reductions before, there are two good reasons to believe that it is a difficult problem. The first is that the density of very smooth numbers $\leq n$ is $O(e^{-u/2})$ where $u = \frac{\log n}{\log \log n}$ for $u \to \infty$ and where log denotes the natural logarithm (see Theorem 1 in Chapter III, Section 5.1 of [17]). This implies that if we were to try to find a very smooth quadratic residue by random search, it would take more time (attempts) than it would to factor n using a subexponential-time factoring method such as the Number Field Sieve (NFS, cf. [10, 4, 3]). Of course random search is naive, so the real question is how difficult it is to solve the NMSRVS problem using an intelligent algorithm. This brings us to our second reason for believing it is a difficult problem. Solving NMSRVS is strongly related to factoring, as all of the best general purpose factoring algorithms work by finding nontrivial square roots of numbers that are smooth with respect to a subexponential smoothness bound (as opposed to very smooth, i.e., smooth with respect to a polynomial smoothness bound). All these algorithms have running times of the form

$$L[n, r, \alpha] = e^{(\alpha + o(1))(\log n)^r (\log \log n)^{1-r}}$$

for constants $0 < r < 1$ and $\alpha > 0$ and for $n \to \infty$. Currently, the best algorithm is NFS which, heuristically, has expected asymptotic runtime $L[n, 1/3, 1.923...]$. Although this is not provable, it has not failed us yet. We show that if one can develop an algorithm that finds collisions in VSH in some time of the form $L[n, r, \alpha]$, then the algorithm can be converted into a factoring algorithm of the same expected asymptotic running time. For instance, if we can find collisions (asymptotically) faster than factoring n with NFS, then we have a new factoring algorithm that has expected (asymptotic) runtime faster than NFS. Thus, if one is willing to believe in the difficulty of factoring, NMSRVS must be hard as well.

We also show how to interpret our asymptotic arguments in a practical setting. This interpretation takes the following form. If VSH uses an N-bit modulus n and smoothness bound B, then there is an easily computable integer $N' < N$ that depends on N and B and that is of the same order of magnitude as N, such that, if VSH-collisions can be found faster than it would take to factor N'-bit numbers using NFS, then it is also the case that the N-bit modulus n can be factored faster than using NFS. Examples of actual values are given. Typically, N and N' are close for small B. VSH gets faster by allowing a larger B but by doing so one incurs a larger gap between N—the VSH-modulus bitlength—and N'—the modulus bitlength whose security one obtains.

As mentioned above, general purpose factoring algorithms such as NFS never set the smoothness bound as absurdly low as it takes to break VSH since such very smooth 'relations', as they are called in factoring jargon, are never found. Practical factoring algorithms such as Quadratic Sieve (QS) and NFS use subexponential functions of $\log n$ for the smoothness bound to optimise their runtimes, whereas our construction uses a polynomial function of $\log n$. Using polynomial smoothness bound in QS or NFS would enormously—and disastrously—increase their runtimes. Thus, it would not be a stretch of the imagination to believe that the NMSRVS problem with an N-bit modulus is as hard as factoring an N-bit modulus, i.e., without incurring the gap between N and N' referred to above. In our implementation figures, however, we take the conservative approach and distinguish between N and N'.

As far as we are aware, our algorithm improves upon the efficiency of previous provable algorithms. If we use a smoothness bound that is linear in $\log n$, then the basic VSH algorithm processes approximately $\frac{\log n}{\log \log n}$ bits of the message for the cost of less than 3 modular multiplications. Using a variation that, among others, releases the smoothness restriction to any fixed higher degree polynomial in $\log n$, the asymptotic cost can be reduced to a small constant number of modular multiplications per $\log n$ message-bits.

Given the relationship between VSH-collision finding and integer factorisation, a natural question to ask is if a party that knows the factorisation of the modulus n used in the hash can use this knowledge to create collisions. That is indeed the case (cf. trapdoor hashes in [15]). Therefore, for wide-spread application of VSH with a single modulus one would either have to rely on a trusted party that generates the modulus (and that can create collisions at will)—we find it hard to imagine that this would be an appealing scenario—or one would have to rely on the method from [2] to generate a modulus with knowledge of its factorization shared among a group of authorities. Since the modulus generation is a one time computation the latter alternative looks reasonable. For 'personalized' application of VSH the repudiation concerns by the owner of the VSH-modulus are not different from the repudiation issues concerning the owner of a regular RSA modulus.

On the positive side, we show how to constructively use the factorisation trapdoor of VSH to build a provably secure randomised trapdoor hash function which requires only about 4 modular multiplications to evaluate on fixed-length messages of length $k < \log n$ bits (compared to the fastest construction in [15], which requires about k modular multiplications). Randomised trapdoor hash functions are used in signature schemes to achieve provable security against adaptive chosen message attack [15], and in designated-verifier signature schemes to achieve privacy [9, 16]. For example, our function can replace the trapdoor function used in the Cramer-Shoup signature scheme [5], maintaining its provable security while saving the cost of a 160-bit double-exponentiation (this is expected to speed up verification time by about 50%).

Related Work. We mention here several other hash functions with collision resistance provably related to factoring which have been proposed in the literature (although all those have lower efficiency than VSH). This subsection will be expanded substantially in a later version. The function we discussed in the introduction appeared in [12, 15]. A collision resistant hash function based on a claw free permutation pair (where claw finding is provably as hard as factoring an RSA modulus) was proposed by Goldwasser, Micali and Rivest in [8]—this function requires 1 squaring per bit processed. Damgård [6] generalised the construction to use families of $r \geq 2$ claw free permutations, such that $\log_2(r)$ bits can be processed per permutation evaluation. He also gave two factoring based constructions for such families, which require 2 modular multiplications per permutation evaluation. The first construction requires the modulus n to have $1 + \log_2(r)$ prime factors, so the modulus length becomes impractical already for small $\log_2(r)$. The second construction uses an RSA modulus with two prime factors, but requires publishing $2^{\log_2(r)}$ random quadratic residues modulo n. Again, this becomes prohibitive for relatively small values of $\log_2(r)$.

The remainder of this paper is organised as follows. In Section 2 we present the basic security definitions and assumptions concerning the hardness of the NMSRVS problem. The VSH algorithm, including some of its variations, is described and discussed in Section 3. Section 4 presents a randomised trapdoor hash function based upon VSH, and shows how it can be used to speed up the provably secure Cramer-Shoup signature scheme. Section 5 concludes with some implementation results.

2 Security Definitions

Notation. Throughout this paper, let c be a fixed positive constant and let n be a hard to factor N-bit composite for some positive integer N. Let B be a smoothness bound $< (\log n)^c$, where we say that an integer is B-smooth if all its prime factors are $\leq B$. We represent residues modulo n as least non-negative residues $\{0, 1, \ldots, n-1\}$ or largest non-positive residues $\{-n+1, -n+2, \ldots, 0\}$ modulo n. It will be clear from the context which representation is being used. By p_i we denote that ith prime: $p_1 = 2$, $p_2 = 3$, $\ldots$.

Definition 1. *We say that an integer a is a very smooth quadratic residue modulo n if the largest prime in the factorisation of a is at most $(\log n)^c$ and there exists some integer x such that $a \equiv x^2 \bmod n$. The integer x is said to be a modular square root of a.*

Definition 2. *An integer x is said to be a* trivial *modular square root of an integer a if $a = x^2$, i.e. a is a perfect square and x is just the integer square root of a.*

Trivial modular square roots have nothing to do with the modulus n. Such relations are easy to create, and therefore we do not want to allow them in our security reduction. A sufficient condition for a very smooth integer a representing a quadratic residue not to have a trivial modular square root is having some prime p such that p divides a but p^2 does not divide a. Another sufficient condition is that a is negative.

Many factoring algorithms construct nontrivial congruent squares modulo n since they can be used to factor n. More precisely, in a relation of the form $x^2 \equiv y^2 \bmod n$ where $x \not\equiv \pm y \bmod n$, we have $\gcd(x-y, n)$ (or $\gcd(x+y, n)$) as a proper factor of n. Once the factorisation of n is known, it is easy to compute nontrivial modular square roots of smooth numbers, since computing modular square roots is easy for primes and the results can be assembled via the Chinese Remainder Theorem. However, without knowing the factorisation of n, it is assumed to be a hard problem:

Definition 3. *(NMSRVS: Nontrivial Modular Square Root of Very Smooth numbers) Let n be the product of two primes of approximately the same size. The NMSRVS problem is the following: Given n, find $x \in Z_n^*$ such that $x^2 \equiv (-1)^{e_0} \prod_{i=1}^{k} p_i^{e_i} \pmod n$, k is such that $p_k \leq B$, and at least one of $e_0, \ldots, e_k$ is odd.*

NMSRVS Assumption. The NMSRVS assumption is that there is no probabilistic polynomial (in N) time algorithm which solves the NMSRVS problem with non-negligible probability (the probability is taken over the random choice of the factors of n and the random coins of the algorithm).

One can contrive moduli where NMSRVS is not difficult, such as if n is very close to a perfect square. However, such examples occur with exponentially small probability assuming p and q are chosen randomly, as required. According to proper security definitions [13], these examples do not even qualify as weak keys since the time-to-first-solution is slower than factoring, and therefore are not worthy of further consideration.

The NMSRVS Assumption is rather weak and, more importantly, mostly useless in practice since it does not tell us for what size moduli the NMVRVS problem would be sufficiently hard. This is similar to the situation in integer factorisation where the assumption that factoring is hard does not suffice to select moduli that are believed to meet a certain security requirement. For that reason, we make an additional, stronger assumption about the hardness of the NMSRVS problem that links it to the current state of the art in factoring. We formulate the following simple result without any attempt to be rigorous.

Theorem 1. *Let n of bitlength N and k be as in the NMSRVS problem, let $T(M)$ denote the expected runtime to factor (any) M-bit integer, and let N' be a positive integer such that $T(N') \approx T(N)/(k+t)$, for a small positive constant t. Then on average finding random solutions to the NMSRVS problem with n and k takes time at least $T(N')$.*

Proof. If random solutions to the NMSRVS problem at hand can on average be found faster than in time $T(N')$, then $k+t$ such solutions can be found in time $T' < T(N)$. But $k+t$ random solutions to the NMSRVS problem for n and k can be used (in time $O(k^3)$ using simple linear algebra) to construct t independent integer solutions to $v^2 \equiv w^2 \bmod n$, and thus t independent chances of at least 50 percent to factor n. Thus, one may expect to factor n in time $T' < T(N)$, contradicting the definition of $T(N)$. □

This result gives a lowerbound on the hardness of the NMSRVS problem based on the difficulty of factoring. Obviously, its practical implications will change as soon as the state of the art in factoring changes—but that is true for applications of the RSA cryptosystem as well and has so far not been a major obstacle against its wide-spread application.

Corollary 1. *If integers cannot be expected to be factored faster than* $L[n, r, \alpha]$ *for some* r *with* $0 < r < 1$ *and* $\alpha > 0$*, for* $n \to \infty$*, then finding solutions to the NMSRVS problem cannot be easier than* $L[n, r, \alpha]$ *either, for* $n \to \infty$*.*

Proof. This is a consequence of the fact that the polynomially bounded factor $1/(k + t)$ is absorbed in the $o(1)$ of the $L[\ldots]$ function.

A slight inconvenience when trying to use the above theorem and its corollary is that the runtime of integer factorisation algorithms (in particular of the current fastest one, the NFS) is notoriously hard to pinpoint. Fortunately, however, all that is needed for our application of Theorem 1 is an approximation of the runtime ratio $T(N)/T(N')$ for N-bit and N'-bit moduli. The latter is something for which a widely accepted approach exists: if N and N' are relatively close the relative runtime can be obtained by dividing the $L[...]$ functions that give the asymptotic growth rate of the NFS runtime (mentioned in the Introduction), after dropping the $o(1)$'s in the expression for L.

The resulting computational hardness assumption for NMSRVS as formulated below, allows us to effectively select parameters for VSH in such a way that if our assumption does not hold, then integers can be factored faster than using the current fastest integer factorisation algorithm (namely, NFS).

Computational NMSRVS Assumption. Finding random solutions to the NMSRVS problem with an N-bit modulus n and number of different primes k is at least as hard as factoring an N'-bit RSA modulus, where N' is the least positive integer such that

$$L[2^{N'}, 1/3, 1.923] \geq L[2^N, 1/3, 1.923]/k.$$

Note that, asymptotically, for these N and N' both N-bit and N'-bit numbers can be factored in the same time $L[2^N, 1/3, 1.923] = L[2^{N'}, 1/3, 1.923]$ for $2^N, 2^{N'} \to \infty$, since k is bounded by a polynomial function of 2^N and gets absorbed by the $o(1)$. For 'practical' and 'relative' sizes, we forget about the $o(1)$, so that the division by a polynomially bounded k becomes meaningful and leads to useful and realistic results.

3 Very Smooth Hash Algorithm

Let the notation be as in the previous section. The basic version of VSH follows below. More efficient variants of VSH are discussed at the end of this section.

VSH Algorithm. Let k (the block length) be the largest integer such that $\prod_{i=1}^{k} p_i < n$. Let m be an ℓ-bit message to be hashed, consisting of bits $m_1, \ldots, m_\ell$, and assume that $\ell < 2^{k-2}$. To compute the hash of m perform steps 1 through 5 in succession:

1. Let $\ell = \sum_{i=1}^{k} \ell_i 2^{i-1}$ with $\ell_i \in \{0, 1\}$ for $1 \le i \le k$ be the binary representation of the message length ℓ. Note that $\ell_{k-1} = \ell_k = 0$.
2. Compute $x_0 = p_{k+1} \times \prod_{i=1}^{k} p_i^{\ell_i} \bmod n$ (note that the size restriction on ℓ implies $x_0 < n/2$).
3. Let $L = \lceil \frac{\ell}{k} \rceil$ (the number of blocks). Let $m_i = 0$ for $\ell < i \le Lk$ (padding).
4. For $j = 0, 1, \ldots, L-1$ in succession compute

$$x_{j+1} := x_j^2 \times \prod_{i=1}^{k} p_i^{m_{j \cdot k+i}} \bmod n.$$

5. Return x_L.

Remarks.

1. For 1024-bit n (i.e., $N = 1024$), the smallest potentially acceptable modulus choice, k would be 131. The requirement that $\ell < 2^{k-2}$ is therefore not a problem in any real application. Actually most of the bits ℓ_i will be zero. Note that according to the Computational NMSRVS Assumption the security level obtained by VSH using $N = 1024$ and $k = 131$ may be lower than '1024-bit RSA', namely at least about '840-bit RSA' (i.e., $N' \approx 840$).
2. VSH essentially applies a variant of the well-known Merkle-Damgård transformation [11, 7] to extend the compression function $H(x, m) = x^2 \prod_i p_i^{m_i} \bmod n$ to arbitrarily long inputs. There are other ways to implement this idea to improve efficiency and to deal with the case where the length is not known ahead of time (see Section 5).
3. In terms of efficiency, k bits are processed per iteration. Since the product of the first K primes is asymptotically $e^{K \log K}$, we find that the k used in the basic version of VSH is proportional to $\frac{\log n}{\log\log n}$. Computing the product $\prod_{i=1}^{k} p_i^{m_{j \cdot k+i}}$ takes $O((\log n)^2)$ using the straightforward multiplication method and has the benefit that it requires no modular reduction. Therefore the cost of each iteration of the hash is less than the cost of 3 modular multiplications. In particular, the basic VSH algorithm needs a small constant number of modular multiplications per $\frac{\log n}{\log\log n}$ message bits.
4. For L-bit exponents e_i for $1 \le i \le k$, and with $m_{j \cdot k+i} = e_{i\,j}$ (where e_i consists of the bits $e_{i\,1}, e_{i\,2}, \ldots, e_{i\,L}$), the calculation of VSH as presented above is the same as the multi-exponentiation $\prod_{i=1}^{k} p_i^{e_i} \bmod n$, except for the initial factor x_0. With knowledge of $\phi(n)$, and assuming sufficiently large L, collisions can be generated by replacing e_i by $e_i + t_i \phi(n)$ for any set of i's with $1 \le i \le k$ and positive integers t_i. Thus, parties that know the factorisation of the modulus n can create collisions at will. But note that collisions of this sort immediately reveal $\phi(n)$ and thus n's factorisation. Creating collisions

that cannot immediately be used to factor n is a harder problem, involving discrete logarithms of very smooth numbers.
To avoid repudiation concerns if VSH would be used 'globally' with the same modulus it would be advisable to generate n using the method from [2]. On the other hand, it is conceivable—and may be desirable—to expand PKI's to allow one to choose one's own hash function, rather than using a 'fixed target' for all. In this setting, we cannot allow the owner of a VSH-modulus to claim he did not sign something by displaying a collision. Especially taking into consideration that the only easy way the user can create a collision would also reveal the factorisation of n, this would be analagous to somebody using RSA who anonymously posts the factorisation of their modulus on the web in order to fraudulently claim that he did not sign something. Thus, in such a situation the VSH-modulus should be considered compromised and the user's certificate should be revoked.

5. The basic version of VSH described above can easily be inverted for messages of length $\ell \leq k$: in that case there are only k possibilities for x_0, so if we divide the resulting hash modulo n by each of the possibililities for x_0^2, one of the k remaining values will equal $\prod_{i=1}^{k} p_i^{m_i} \bmod n$. But this value actually equals $\prod_{i=1}^{k} p_i^{m_i}$, since the product is small enough so that there is no 'wrap-around' modulo n. Thus, the factorisation of one of the resulting values reveals which bits were set. We emphasize that this type of invertibility may be undesirable for some applications, but that others require just collision resistance (cf. Subsection below).
One possible solution to the short message invertibility problem that does not affect our proof of security (cf. below) is to square the final output enough times to ensure wrap-around (no more than $\log_2 \log_2 n$ times). Other, more efficient solutions may be possible.
6. It is not hard to come up with different messages m and m' for which the hashes $h(m)$ and $h(m')$ satisfy possibly undesirable multiplicative properties such as $h(m) = 2h(m')$. Our method that solves the invertibility problem addresses this issue as well.

Having stressed upfront (in the last three remarks above) the known disadvantages of VSH, we now turn to its most attractive property, namely its provable collision resistance.

3.1 Security Proof for VSH

We prove that VSH is (strongly) collision resistant. Using proper security notions [14], (strong) collision resistance also implies second preimage resistance.

Theorem 2. *Finding any collision in VSH is as hard as solving the Nontrivial Modular Square Root of Very Smooth numbers (NMSRVS) problem (i.e., our algorithm is collision resistant under the assumptions from the previous section).*

Proof. We show that finding a collision either reveals a nontrivial modular square root of a very smooth number or else it reveals the factorisation of n (and hence we can easily solve the NMSRVS problem).

Let m and m' be two different colliding messages. For notational convenience we replace the $x_{\dots}$ values for m' by $y_{\dots}$. Assume we have a collision with $x_{a+1} \equiv y_{b+1} \bmod n$ but $x_a \not\equiv y_b \bmod n$. If both a and b are larger than 0, then we get the following relation

$$x_a^2 \times \prod_{i=1}^{k} p_i^{m_{a\cdot k+i}} \equiv y_b^2 \times \prod_{i=1}^{k} p_i^{m'_{b\cdot k+i}} \bmod n \ .$$

All quantities are by construction invertible modulo n, so

$$(x_a/y_b)^2 \equiv \prod_{i=1}^{k} p_i^{m'_{b\cdot k+i} - m_{a\cdot k+i}} \bmod n \ . \tag{1}$$

Since the kth prime is $\approx \log n$ (by the prime number theorem), the right hand side is a ratio of two very smooth numbers having modular square root x_a/y_b. Let $S = \{i : m'_{b\cdot k+i} - m_{a\cdot k+i} = 1 \text{ and } 1 \leq i \leq k\}$ and let $T = \{i : m'_{b\cdot k+i} - m_{a\cdot k+i} = -1 \text{ and } 1 \leq i \leq k\}$. Equation 1 is equivalent to

$$\left[(x_a/y_b) \times \prod_{i\in T} p_i\right]^2 \equiv \prod_{i \in S\cup T} p_i \bmod n \ . \tag{2}$$

So we have transformed the relation into a form where we have a modular square root of a very smooth number. Notice that any prime with exponent -1 or 1 in Equation 1 will have an exponent of 1 in Equation 2. Thus, as long as there is at least one message bit in the current k-bit message-blocks that differs, Equation 2 is nontrivial.

If all bits in the current message block are the same, meaning $m'_{b\cdot k+i} = m_{a\cdot k+i}$ for $1 \leq i \leq k$, then it must be the case that $x_a^2 \equiv y_b^2 \bmod n$. We will be able to factor n if $x_a \not\equiv \pm y_b \bmod n$, so we only need to consider the cases $x_a \equiv \pm y_b \bmod n$. By assumption, $x_a \not\equiv y_b \bmod n$. If $x_a \equiv -y_b \bmod n$, then we have found a nontrivial modular square root of a very smooth number in the previous iteration. It must be nontrivial because the exponent of -1 is 1. This completes the case of $a > 0$ and $b > 0$.

If both a and b are 0, then a similar argument to the above holds: The only difference is that we cannot have $x_0 \equiv -y_0 \bmod n$ because the size restriction on ℓ forces both values to be less than $\frac{n}{2}$, and clearly $x_0 \not\equiv y_0 \bmod n$ if the message lengths differ (if the lengths are the same, then it is not a real collision).

Finally, assume exactly one of a or b is 0. Without loss of generality we take $a = 0$ and $b > 0$ and we are asking whether we can have $x_0 \equiv \pm y_b \bmod n$ in the relation $x_0^2 \equiv y_b^2 \bmod n$. Substituting the equation for y_b, it can only happen if

$$x_0 / \prod_{i=1}^{k} p_i^{m'_{(b-1)k+i}} \equiv \pm y_{b-1}^2 \bmod n \ .$$

After performing a similar transformation to the one used in Equation 2, the left hand side has the prime p_{k+1} to the exponent of 1 (from x_0), meaning that y_{b-1} is a nontrivial square root of a very smooth number. □

3.2 Example: A Related Algorithm that can be Broken

To emphasize the importance of the nontrivialness, consider a hash function that works similarly to VSH, except breaks the message into blocks r_i of K bits and uses the compression function $x_{i+1} := x_i^2 \times 2^{r_i} \bmod n$. By allowing $r_i > 1$ we can create trivial collisions. For example the message blocks $r_1 = e$ and $r_2 = 2e$ collide with $r_1' = 2e$ and $r_2' = 0$. The derivable relation for this collision (similar to the formula for Equation 1) is

$$\left(\frac{x_0^2 2^{2e}}{x_0^2 2^e}\right)^2 \equiv 2^{2e-0} \bmod n \ ,$$

or, in other words,

$$(2^e)^2 \equiv 2^{2e} \bmod n \ .$$

Such trivial relations are useless and thus, the security of this hash algorithm is not reducible to factoring or any hard problem. The problem disappears again if we replace $x_{i+1} := x_i^2 \times 2^{r_i}$ by the costlier variant $x_{i+1} := x_i^{2^K} \times 2^{r_i}$, but that is the same as the function $x^m \bmod n$ from [12, 15].

3.3 Other Security Issues

Since the output length of VSH is the length of a secure RSA modulus (thus in the range of 1024 or 2048 bits), it seems quite suitable in practice for constructing 'hash-then-sign' RSA signatures for arbitrarily long messages. However, we warn the reader that such constructions must be designed carefully to ensure that the resulting signature scheme is secure. To illustrate a naive insecure construction, suppose that the signer with public key (n, e) uses the same RSA modulus n for both hashing and signing, so the signing function $S^* : \{0,1\}^* \to Z_n$ is $S^*(m) = H_n(m)^{1/e} \bmod n$, where $H_n : \{0,1\}^* \to Z_n$ is VSH with modulus n. For a k-bit message $m = (m_1, \ldots, m_k) \in \{0,1\}^k$, the corresponding signature is thus $\sigma = (x_0^2 \prod_{i=1}^k p_i^{m_i})^{1/e} \bmod n$, where x_0 has the same value for all k-bit messages.

This scheme is insecure under a chosen message attack, which proceeds as follows. The attacker obtains signature $\sigma_0 = (x_0^2)^{1/e} \bmod n$ on message $m_0 = (0, 0, 0, \ldots, 0)$ (k zero bits), a signature $\sigma_1 = (x_0^2 \cdot p_1)^{1/e} \bmod n$ on message $m_1 = (1, 0, 0, \ldots, 0)$, and a signature $\sigma_2 = (x_0^2 \cdot p_2)^{1/e} \bmod n$ on message $m_2 = (0, 1, 0, \ldots, 0)$. Then the attacker can easily compute the signature $\sigma_3 = \sigma_1 \cdot \sigma_2 / \sigma_0 \bmod n$ on the new k-bit forgery message $m_3 = (1, 1, 0, \ldots, 0)$ (more generally, it is easy to see that $k + 1$ signatures suffice to sign any k-bit message).

To avoid attacks of the type above, we would suggest the following more theoretically sound design approach for using VSH with 'hash-then-sign' RSA signatures, which does not rely on any security property of the hash function beyond the collision resistance which it was designed to achieve:

1. Let ℓ_{mod} be the desired RSA signature modulus n_s length (typ. $\ell_{mod} = 1024$). Let $\alpha = \ell_{mod} - 1$ so that $2^\alpha < n_s$. Specify a one-to-one one-way encoding

function $f : \{0,1\}^\alpha \to \{0,1\}^\alpha$, and define the short-message (α-bits) RSA signature scheme with signing function $S_{n_s}(m) = (f(m))^{1/e} \bmod n_s$. The function f is chosen such that the short-message scheme S_{n_s} is existentially unforgeable under chosen message attack. Note that no provable techniques are currently known for finding such a function f (in the standard model), but since f is one-to-one, there are no collision resistance issues to consider when designing f.

2. The signature scheme for signing arbitrary length messages is now constructed with signing function $S^*_{n_s,n_h}(m) = S_{n_s}(H_{n_h}(m))$, where H_{n_h} is VSH with a separate RSA modulus n_h (chosen randomly and independently of the signing modulus n_s) of length α bits. The public key of the signer is (n_s, n_h, e). It is now easy to prove that the scheme S^* is existentially unforgeable under chosen message attack, assuming that S is and that VSH H_{n_h} is collision resistant. We emphasize that the proof of this latter statement no longer holds if one uses the same modulus for both hashing and signing (in order to make the proof work for $n_s = n_h = n$ we would need the stronger assumption that H_n is collision resistant even given access to a signing oracle S_n for scheme S). However, it is worth remarking that if we model the function f as a random oracle, then the proof of security works (under the RSA assumption and NMSRVS assumption) even with a shared modulus ($n_s = n_h$).

We remarked above that the function VSH processes long inputs by applying a variant of the Merkle-Damgård transformation to the compression function $H_c(x,m) : Z_n^* \times \{0,1\}^{k+1} \to Z_n^*$, where $H_c(x,m) = x^2 \prod_{i=1}^{k+1} p_i^{m_i} \bmod n$. This transformation may be stated in general as follows. Let $H_c(x,m) : Z_n^* \times \{0,1\}^{k+1} \to Z_n^*$ be the given compression function. Let $\mathscr{B} < n$ be a message length bound. To hash a message M of length $\ell < \mathscr{B}$, pad M with zero bits to make its length the nearest multiple of k bits, and split the padded M into k-bit blocks $M_0, M_1, \ldots, M_{L-1}$. Define $c_1 = H_c(\ell, 1||M_0)$ (where $||$ denotes concatanation), and for $i = 1, \ldots, L-1$, compute $c_{i+1} = H_c(c_i, 0||M_i)$. The final hash value is $H(M) = c_L$. Note that the bit prepended to the message blocks (1 for first block and 0 for subsequent blocks) is there to prevent collisions for messages of different block length. In our VSH function this corresponds to the $k+1$st prime used only in hashing the first block. Except for cosmetic differences this is the same as the transformation in [7].

The proof in [7] shows that a sufficient condition for the resulting Merkle-Damgård function H to be collision resistant is that the compression function H_c is collision resistant, i.e. it is hard to find any $(x,m) \neq (x',m')$ with $H_c(x,m) = H_c(x',m')$. Our compression function $H_c(x,m) = x^2 \prod_{i=1}^{k+1} p_i^{m_i} \bmod n$ is not strictly collision resistant ($H_c(-x \bmod n, m) = H_c(x,m)$) but as we proved it is still sufficiently strong to make H collision resistant. One may ask whether we can generalize the result in [7] to state the conditions on a compression function (which are weaker than full collision-resistance) that our compression satisfies and that are still sufficient to make the Merkle-Damgård function H collision resistant. Indeed, these conditions can be readily generalised from our proof of Theorem 2, so we only state them here:

(1) Collision Resistance in Second input: It is hard to find $(x, m), (x', m') \in Z_n^* \times \{0,1\}^{k+1}$ with $m \neq m'$ such that $H_c(x, m) = H_c(x', m')$.
(2) Preimage Resistance for a collision in first input: It is hard to find $(x, m) \neq (x', m') \in Z_n^* \times \{0,1\}^{k+1}$ and $m \in \{0,1\}^{k+1}$ such that $H_c(y, m) = H_c(y', m)$, where $y = H_c(x, m)$, $y' = H_c(x', m')$ and $y \neq y'$.
(3) Small Collision Resistance in first input: It is hard to find $x, x' \in Z_{\mathscr{B}}$ (note both x and x' are smaller than $\mathscr{B} < n$) and $m \in \{0,1\}^{k+1}$ with $x \neq x'$ such that $H_c(x, m) = H_c(x', m)$.

Our VSH compression function satisfies all these properties (with $\mathscr{B} < n/2$), assuming the Computational NMSRVS assumption.

3.4 Variants of VSH

We briefly mention some variants of VSH.

Variation I: Cubing instead of squaring. The first is to change the squaring operation in the compression function to a cubing, i.e., a compression function of the form $H : Z_n \times \{0,1\}^k \to Z_n$ where $H(x, m) = x^3 \prod_{i=1}^{k} p_i^{m_i} \bmod n$. If $\gcd(3, \phi(n)) = 1$ then thanks to the injectivity of the RSA cubing map modulo n, this compression function is collision resistant, assuming the difficulty of computing a modular cube root of a very smooth cube-free integer of the form $\prod_{i=1}^{k} p_i^{e_i}$, where $e_i \in \{0,1,2\}$ for all i and at least one e_i is not zero. This problem is related to RSA inversion, and is also conjectured to be hard. Although this function requires about 4 modular multiplications per k bits processed (compared to 3 for the squaring version), it has the interesting property that the compression function itself is collision resistant, while this is not quite the case for the squaring compression function (because $x^2 \prod_i {p_i}^{m_i} \equiv (-x)^2 \prod_i p_i^{m_i} \pmod{n}$).

Variation II: Increasing the smoothness bound. A speed-up is obtained by allowing the use of larger values of k than the largest k for which $\prod_{i=1}^{k} p_i < n$. It can easily be seen that allowing larger k does not affect the proof of security and reduction to the NMSRVS problem, as long as the smoothness bound is still polynomially bounded in $\log n$. As a consequence of the Computational NMSRVS Assumption, a larger k implies that a larger N has to be used to maintain the same level of security. Furthermore, the intermediate products in step 4 of the VSH algorithm may get larger than n and may thus have to be reduced modulo n every so often. Nevertheless, the fact that L gets smaller because more bits are processed per iteration so that fewer multiplications and squarings modulo n have to be performed, outweighs the disadvantages. A detailed analysis will appear in a later version of this paper.

Variation III: Bytewise message processing using precomputed prime-products. An implementation speed-up may be obtained by processing the bits of the message b bits at a time, for some $b > 1$, instead of one bit at a time as in the original description. For instance, the choice $b = 8$ leads to bytewise processing of m. For ease of description, suppose that k is a multiple of b, in particular that $k = Sb$ for an integer S. For $1 \leq s \leq S$ let

$$P_s = \{\prod_{t=1}^{b} p_{(s-1)b+t}^{e_t} : e_t \in \{0,1\} \text{ for } 1 \leq t \leq b\}$$

be the set consisting of all 2^b products over the sth b-tuple of primes. Then each of the 2^b elements of P_s can be indexed by a b-bit value v, namely the element whose exponents e_t correspond to the bits of v, i.e.,

$$P_s[v] = \prod_{t=1}^{b} p_{(s-1)b+t}^{e_t} \text{ if and only if } v = \sum_{t=1}^{b} e_t 2^{t-1}.$$

The sets P_s for $1 \leq s \leq S$ can be precomputed. As a result the calculation in step 4 of the VSH algorithm can be replaced by the possibly slightly faster but equivalent computation

$$x_{j+1} := x_j^2 \times \prod_{s=1}^{S} P_s[m[j \cdot S + s]] \bmod n,$$

where $m[u]$ now refers to the uth b-bit chunk of m. This change has no effect on the number of iterations or the value of N to be used to reach a certain security level.

Variation IV: Bytewise message processing using primes. A no longer equivalent but faster variant follows from the one above by redefining $P_s[v]$ as $p_{(s-1)2^b+v}$. As a result L in Step 3 becomes $L = \lceil \frac{\ell}{bk} \rceil$, and the calculation in Step 4 becomes:

$$x_{j+1} := x_j^2 \times \prod_{i=1}^{k} p_{(i-1)2^b+m[jbk+i]+1} \bmod n,$$

where the ith b-bit chunk $m[i]$ of the message is interpreted as b-bit integer. As a consequence of this change the block length increases from k to bk, and the number of small primes goes from k to $k2^b$. Thus, a larger N has to be used to maintain the same level of security. Overall, however, this change is clearly advantageous, as shown in the analysis below and the runtime examples in the next section.

Analysis of Variation IV. Let k be maximal such that

$$\prod_{i=1}^{(k+1)2^b} p_i \leq (2n)^{2^b},$$

i.e., $(k+1)2^b$ is proportional to $\frac{2^b \log(2n)}{\log(2^b \log(2n))}$ and k to $\frac{\log(2n)}{\log(2^b \log(2n))} - 1$. With

$$\prod_{i=1}^{(k+1)2^b} p_i = \prod_{j=1}^{2^b} \prod_{i=0}^{k} p_{i2^b+j}$$

it follows that

$$\prod_{i=0}^{k} p_{i2^b+1} \leq (2n).$$

Because $p_{i2^b} < p_{i2^b+1}$ we find that

$$\prod_{i=1}^{k} p_{i2^b} < n.$$

Therefore, each intermediate product in the compression function will be $< n$, since p_{i2^b} is the largest factor that can be used for a b-bit chunk $m[jbk+i]$ of the message. The cost of Variation IV is therefore a constant number of modular multiplications per bk message bits, where bk is proportional to $\frac{b\log(2n)}{\log(2^b\log(2n))} - b$. By selecting 2^b as any fixed positive power of $\log n$, it follows that bk is proportional to $\log n$: with $2^b = (\log(2n))^d$ for some $d > 0$, we find that bk is proportional to

$$\frac{d}{d+1}\log(2n) - d\log\log(2n).$$

For this choice, the number of small primes $k2^b$ and the smoothness bound p_{k2^b} are also both polynomially bounded in $\log n$. Denoting by N' the security level one achieves using $k2^b$ small primes and n, it then follows from the Computational NMSRVS Assumption that the number of message-bits processed per iteration is linear in N' as well.

Remarks.

1. For all variations given above knowledge of $\phi(n)$ can be used to generate collisions, though displaying such a collision is not in the user's interest since he would be giving out a break to his hash function (i.e. it would be similar to someone giving out the factorisation of their RSA modulus).
2. Variations IV and II can be combined in the usual straightforward fashion. If, for instance, the number of small primes is taken almost w times larger, for some integer $w > 1$, the small prime product can be split into w factors each less than n. Per iteration this results in a single modular squaring, $w-1$ modular multiplications plus the time to build the w products. The best value for w is best determined experimentally, and will depend on various processor characteristics (such as cache size to hold a potentially rather large table of primes).
3. Note that in Variation IV, a prime is multiplied in when we have a $b-$bit chunk of zero bits. In the original VSH, a chunk of zero bits has no prime multiplied in. Obviously, a negligible speed-up and small saving in the number of primes can be obtained if for a particular b-bit pattern (such as all zeros) no prime is multiplied in.

4 A Randomised Trapdoor Hash Function Based on VSH and its Application to the Cramer-Shoup Signature Scheme

A *randomised trapdoor hash function (family)* [15] $F_{pk}(\cdot;\cdot)$ accepts, in addition to a message $M \in \mathcal{M}$, a random input $r \in \mathcal{R}$ (for some message/randomiser spaces $\mathcal{M}$ and $\mathcal{R}$). The function F_{pk} is collision resistant and can be efficiently evaluated using a public key pk, but an associated secret trapdoor key sk can be used to find collisions for F_{pk} satisfying a certain randomness property. More precisely, the security requirements for these functions are:

(1) Collision Resistance (in Message Input): Given pk, it is hard to find two message/randomiser pairs $(M, r), (M', r') \in \mathcal{M} \times \mathcal{R}$ with $M \neq M'$ such that $F_{pk}(M, r) = F_{pk}(M', r')$.
(2) Random Trapdoor Collisions: There exists an efficient collision-finding algorithm that given trapdoor (sk, pk), a message/randomiser pair $(M, r) \in \mathcal{M} \times \mathcal{R}$ and another message $M' \in \mathcal{M}$, outputs a randomiser $r' \in \mathcal{R}$ such that
   - $F_{pk}(M, r) = F_{pk}(M', r')$.
   - If r is chosen uniformly from $\mathcal{R}$ then r' is uniformly distributed in $\mathcal{R}$.

Randomised trapdoor hash functions have applications in provably strengthening the security of signature schemes [15], and constructing designated-verifier proofs/signature schemes [9, 16]. The built-in factorisation trapdoor of VSH suggests that it can be used to build such a function. However, the method described above for finding VSH collisions by working modulo $\phi(n)$ in one of the p_i exponents leads to an inefficient function evaluation requiring a long exponentiation. Here we describe a provably secure trapdoor hash family which preserves the efficiency of VSH.

- *Key Generation:* Choose two random primes p and q such that $p \equiv q \equiv 3 \pmod 4$. The public key is $n = pq$. The trapdoor key is $sk = (p, q)$.
- *Hash Function:* Given public key n, a message $M \in \{0,1\}^*$ of bit-length $\ell < 2^k$ and a randomiser $r \in Z_n^*$, evaluate $F_n(M; r)$ as follows. Pad M with zero bits to a multiple of k bits (where k is chosen as in VSH). Let $M_i = (M_{i,1}, \ldots, M_{i,k}) \in \{0,1\}^k$ denote the ith k-bit block of the padded M for $i = 0, \ldots, L-1$, where $L = \lceil \ell/k \rceil$, and let $f_n : Z_n^* \times \{0,1\}^k \to Z_n^*$ denote the VSH compression function $f(x, m) = x^2 \prod_{i=1}^{k} p_i^{m_i} \bmod n$. Define initial value $x_0 = r$. For $i = 0, \ldots, L-1$, compute $x_{i+1} = f_n(x_i, M_i)$. Then compute $x_{L+1} = f(x_L, \ell)$ (where ℓ is represented in k-bit binary), and finally $F_n(M; r) = x_{L+1}^2 \bmod n$.

Note that $F_n(M; r) = (r^{2^{L+1}} \prod_{i=1}^{k} p_i^{\hat{M}_i})^2 \bmod n$, where $\hat{M}_i = \sum_{j=0}^{L-1} M_{j,i} \cdot 2^{L-j} + \ell_{i-1}$ for $i = 1, \ldots, k$.

Theorem 3. *The above construction satisfies the security requirements for randomised trapdoor hash functions, under the NMSRVS assumption.*

Proof. (1) *Collision Resistance (in Message Input):* Suppose that given n an attacker finds a collision pair $(M, r), (M', r') \in \{0,1\}^* \times Z_n^*$ with $M \neq M'$ such that $F_n(M; r) = F_n(M'; r')$. Let $\ell, \ell' < 2^k$ denote the bit-length of M (and M'), and x_i, x'_i denote the intermediate values in hashing M (and M'). We can assume that $x'_{L+1} \equiv \pm x_{L+1}$ since otherwise we can factor n by computing $\gcd(x'_{L+1} - x_{L+1}, n)$ and break NMSRVS. So we have $(x_L/x'_L)^2 \equiv \pm \prod_{j=1}^k p_j^{\ell'_{j-1} - \ell_{j-1}} \pmod{n}$. If $\ell \neq \ell'$, one of the prime exponents on the right-hand side is equal to 1 and we have broken NMSRVS (after multiplying both sides by squares of primes with negative exponents). Otherwise $\ell = \ell'$, so $L' = L$, but since $M' \neq M$ there must exist block index $i \in \{0, \dots, L+1\}$ such that $(x_i, M_i) \neq (x'_i, M'_i)$ but $(x_j, M_j) = (x'_j, M'_j)$ for $j \geq i+1$ (where we define $M_{L+1} = M_{L+2} = M'_{L+1} = M'_{L+2} = 0^k$, $M_L = \ell$, $M'_L = \ell'$). So $(x_i/x'_i)^2 \equiv \prod_{j=1}^k p_j^{M'_{i,j} - M_{i,j}} \pmod{n}$. If $M_i \neq M'_i$ then we have broken NMSRVS (one of the prime exponents is ± 1). Otherwise, we must have $i \geq 1$ (since $M \neq M'$) and $(x_i)^2 \equiv (x'_i)^2 \pmod{n}$ so we can assume $x_i \equiv -x'_i \pmod{n}$ (else we factor n) and hence $(x_{i-1}/x'_{i-1})^2 \equiv -\prod_{j=1}^k p_j^{M'_{i-1,j} - M_{i-1,j}} \pmod{n}$ so we have broken NMSRVS thanks to the -1 factor on the RHS.

(2) *Random Trapdoor Collisions:* Let $QR_n = \{y \in Z_n^* : (\frac{y}{p}) = (\frac{y}{q}) = 1\}$ denote the subgroup of quadratic residues of Z_n^*. The proof is based on the fact that, thanks to the choice $p \equiv q \equiv 3 \pmod{4}$, -1 is a quadratic non-residue in both Z_p^* and Z_q^*, so for each $y \in QR_n$, exactly one of its 4 square roots in Z_n^* is in QR_n. Hence the squaring map $g : QR_n \to QR_n$ defined by $g(x) = x^2 \bmod n$ is a permutation on QR_n, and g^{-1} can be efficiently computed given p and q by performing square roots in Z_p^* and Z_q^* and combining the results by Chinese remaindering. Note that $F_n(M; r) = (r^{2^{L+1}} \prod_{i=1}^k p_i^{\hat{M}_i})^2 \bmod n$, where $\hat{M}_i = \sum_{j=0}^{L-1} M_{j,i} \cdot 2^{L-j} + \ell_{i-1}$ for $i = 1, \dots, k$. So given trapdoor (p, q), a message/randomiser pair $(M, r) \in \{0,1\}^* \times Z_n^*$ and another message $M' \in \{0,1\}^*$, a randomiser $r' \in Z_n^*$ satisfying $F_n(M; r) = F_n(M'; r')$, i.e.

$$(r')^{2^{L'+2}} \equiv r^{2^{L+2}} \cdot (\prod_{i=1}^k p_i^{\hat{M}_i - \hat{M}'_i})^2 \equiv h \pmod{n},$$

(where $\hat{M}'_i = \sum_{j=0}^{L'-1} M'_{j,i} \cdot 2^{L'-j} + \ell'_{i-1}$ for $i = 1, \dots, k$) is found by computing $s = g^{-(L'+1)}(h) \in QR_n$, and then choosing r' uniformly at random among the 4 square roots of s in Z_n^*. Note that if r is uniformly distributed in Z_n^*, then (since each element of QR_n has 4 square roots in Z_n^*) the value $r^2 \bmod n$ is uniformly distributed in QR_n. Since the squaring map g is a permutation on QR_n, it follows that h and s are also uniformly distributed in QR_n and hence r' is uniformly distributed in Z_n^*, as required. □

Remarks.

– Our function has the same evaluation efficiency as VSH. In particular, for short fixed-length messages $M = (m_1, \dots, m_k) \in \{0,1\}^k$ (with $k < \log n$, i.e.

1 block), it is not even necessary to append the message length, so the function becomes simply $F_n(M; r) = (r^2 \prod_{i=1}^{k} p_i^{m_i})^2 \bmod n$ and requires only about 4 modular multiplications to evaluate, compared to the trapdoor functions in [15] which require at least k modular multiplications. On the other hand, our trapdoor collision-finding algorithm is not very fast, requiring one square root modulo n per message block. Fortunately, this is not major issue because in many applications of randomised hash functions, the collision-finding algorithm is only used in the *security proof* of a signature scheme rather than in the scheme itself.

– It is easy to see from the proof of Theorem 3 that our randomised hash function also satisfies the 'inversion' trapdoor property [15], which is stronger than the trapdoor collision property, and can be used to upgrade a signature scheme's security from random message attacks to chosen message attacks. Namely, given the trapdoor key, a random element $c \in QR_n$ in the range of F_n and a message $M \in \{0,1\}^*$, it is easy to find a randomiser $r \in Z_n^*$ such that $F_n(M, r) = c$, and r is uniformly distributed in Z_n^* when c is uniformly distributed in QR_n.
– Fixing the random input $r = 1$, we obtain a collision resistant variant of VSH where the length of the hashed message is only needed at the end of the hashing process, rather than the beginning. This variant may be better suited for some applications than the version of VSH presented in the previous sections.

As an example application, we mention the Cramer-Shoup (CS) signature scheme [5], which to our knowledge is the most efficient factoring-based signature scheme provably secure in the standard model (under the strong-RSA assumption). This signature scheme makes use of an RSA-based randomised trapdoor hash function to achieve security against adaptive message attacks. Using our randomised trapdoor hash function instead, we can preserve the proven security, while cutting the signing/verification cost by about 1 double-exponentiation with 160-bit exponents. The modified Cramer-Shoup scheme (typ. $\ell = 160$ and $\ell' = 512$) is as follows:

– *Key Generation*: Choose two safe random ℓ' bit primes p,q and another pair of random ℓ' bit primes p_v,q_v with $p_v \equiv q_v \equiv 3 \pmod 4$. Compute $n = pq$ and $n_v = p_v q_v$. Choose two random quadratic residues x,h in Z_n^*. The public key is $pk = (x, h, n_v, n, H)$, where $H : \{0,1\}^{2\ell'} \rightarrow \{0,1\}^{\ell}$ is a collision resistant hash function. The secret key is $sk = (p, q)$.
– *Signing*: To sign message $m \in \{0,1\}^*$, choose a random $\ell + 1$ bit prime e and a random $r \in Z_{n_v}^*$ and compute $y = (x \cdot h^{H(F_{n_v}(m;r))})^{1/e} \bmod n$, where $F_{n_v}(\cdot;\cdot)$ is the VSH randomised trapdoor function with modulus n_v. The signature is (e, y, r).
– *Verifying*: To verify message/sig. pair $(m, (e, y, r))$, check that e is an odd $\ell + 1$ bit integer and that $y^e h^{-H(F_{n_v}(m;r))} \equiv x \pmod n$.

The cost of verification in the original CS scheme is about two double exponentiations with $\ell = 160$ bit exponents, while in our modified scheme above it is approximately only one such double exponentiation, so we expect a saving in verification time of about 50% (the relative saving in signing time would be smaller). However, the length of the public key is larger than in the original scheme by typically 25%.

Note that in the above modified CS scheme we still need a separate collision resistant hash function H with $\ell = 160$ bit output length. Unfortunately, VSH cannot be used to implement H because of VSH's large $2\ell' = 1024$ bit output hash length. If we wish to avoid the need for ad-hoc 160 bit hash functions, one option is to drop H completely, and increase the length of the random prime e to 1025 bits, but this will entail a significant loss of efficiency. Fortunately, we now present another variant of the CS scheme which eliminates H but maintains almost the same computational efficiency of the variant above (at the expense of an increase in the length of the public key and some precomputation):

- *Key Generation*: Choose two safe random ℓ' bit primes p,q and another pair of random ℓ' bit primes p_v,q_v with $p_v \equiv q_v \equiv 3 \pmod 4$. Compute $n = pq$ and $n_v = p_v q_v$. Choose $s+1$ (where $s = \lceil 2\ell'/\ell \rceil = 6$ typically) random quadratic residues x,$h_1, \ldots, h_s$ in Z_n^*. The public key is $pk = (x, h_1, \ldots, h_s, n_v, n)$. The secret key is $sk = (p, q)$.
- *Signing*: To sign message $m \in \{0,1\}^*$, choose a random $\ell+1$ bit prime e and a random $r \in Z_{n_v}^*$ and compute $y = (x \cdot \prod_{i=1}^{s} h_i^{[F_{n_v}(m;r)]_i})^{1/e} \bmod n$, where $F_{n_v}(\cdot;\cdot)$ is the VSH randomised trapdoor function with modulus n_v and, for $i = 1, \ldots, s$, $[F_{n_v}(m;r)]_i \in \{0,1\}^\ell$ is the ith ℓ bit block of $F_{n_v}(m;r)$ when the latter is interpreted as an $s \cdot \ell \geq 2\ell'$ bit string. The signature is (e, y, r).
- *Verifying*: To verify message/sig. pair $(m, (e, y, r))$, check that e is an odd $\ell+1$ bit integer and that $y^e \prod_{i=1}^{s} h_i^{-[F_{n_v}(m;r)]_i} \equiv x \pmod n$.

For typical parameter values ($\ell = 171$ bit, $s = 6$, $\ell' = 512$ bit), and assuming one precomputes the 64 subset products of the h_i's modulo n, one can use the well-known multi-exponentiation technique to perform the signing/verifiying of the above scheme at about the same cost as in the previous CS variant. We can prove that the above CS signature variant is secure assuming only the strong-RSA and NMSRVS assumptions (the proof will be added to a later version of the paper). Thus we obtain an efficient signature scheme proven secure without any ad-hoc assumptions (unlike the original CS scheme, which relied on a collision-resistance or universal one-wayness assumption regarding a 160-bit hash function H – as far as we are aware, the only practical provably secure design for such a hash function H is an inefficient discrete log based construction using an elliptic curve defined over a 160-bit order finite field). A disadvantage of the above variant is that its public key is somewhat long (typically 9 kbits, which is about 3 times longer than in the original CS scheme).

5 Efficiency of VSH in Practice

Asymptotically, the runtime of the basic VSH algorithm is $O(\ell/k \times (\log n)^2) = O(\ell \log n \log\log n)$ for an ℓ bit message. Note that we have treated a modular multiplication as an $O((\log n)^2)$ operation, which is how it is usually implemented in practice, but better multiplication algorithms exist. The owner of the modulus $n = pq$ can do better for long messages, assuming he knows the factorisation. This is based on Remark 4 in Section 3 since each of the k exponents of bitlength L can be reduced modulo $\phi(n)$. We do not elaborate. As noted at the end of Section 3, Variation IV can be made to run asymptotically faster than the basic VSH algorithm, resulting in overall runtime $O(\ell \log n)$ when traditional arithmetic is used. As shown below, this is not just an asymptotic speedup.

If the length of the message is not known ahead of time (streaming data), the value x_0 with its proper power can be 'pasted on' at the end of the computation at the cost of single moduluar exponentiation. Or better yet, using the variant described in Section 4 with $r = 1$, the length can be pasted on at the end for the cost of less than three modular multiplies.

Using a straightforward gmp-implementation of the basic VSH algorithm on a 1GHz Pentium III, we achieved 0.355 Megabyte per second (MB/s) with $N = 1234$ and $k = 152$, corresponding to $N' = 1024$, i.e., at least 1024-bit RSA security. With $k = 1024$ (and thus a larger $N = 1318$ to maintain the $N' = 1024$ security level) we got 0.419 MB/s if we process the message bitwise (i.e., Variation II above) and 0.486 MB/s if we precompute $1024/8 = 128$ sets of 256 small prime products and do bytewise message processing (i.e., Variations II and III combined). Using Variations IV and II combined with $2^{16} = 65536$ small primes and again bytewise processing, we achieved 1.135 MB/s (where we had to use $N = 1516$ to maintain $N' = 1024$), which is approximately 26 times slower than Wei Dai's SHA-1 benchmark [18]. This last implementation processes $65536/256 = 256$ bytes per iteration, for a total of $2^{12} = 4096$ iterations per Megabyte of input. The basic VSH algorithm processes 152 bits (i.e., 19 bytes) per iteration, for a total of 55189 iterations per Megabyte of input.

For 2048-bit RSA security (i.e., $N' = 2048$) we got the following figures: 0.216 MB/s for the basic variant with $k = 272$ and $N = 2398$, 0.270 MB/s for Variation II with $k = 1024$ and $N = 2486$, 0.303 MB/s for Variations II and III (bytewise) combined, and 0.705 MB/s for Variations IV and II (bytewise) combined with $2^{18} = 262144$ small primes and $N = 2874$.

Acknowledgements: We gratefully acknowledge inspiring discussions with Igor Shparlinski, and we thank Yvo Desmedt, Josef Pieprzyk, and Benne de Weger for their very helpful comments.

References

1. M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: incrementality at reduced cost. In EUROCRYPT 97, volume 1233 of *LNCS*, page 163–192, Berlin, 1997, Springer-Verlag.
2. D. Boneh and M. Franklin. Efficient generation of shared RSA keys. In CRYPTO 97, volume 1294 of *LNCS*, page 425–439, Berlin, 1997, Springer-Verlag.
3. D. Coppersmith. Modifications to the number field sieve. In volume 6 of *J. Cryptology*, pages 169–180, 1993.

4. R. Crandall and C. Pomerance. *Prime numbers: a Computational Perspective*, New York, 2001, Springer-Verlag.
5. R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In volume 3 of *ACM Transactions on Information and System Security (ACM TISSEC)*, pages 161–185, 2000.
6. I. Damgård. Collision-free hash functions and public key signature schemes. In EUROCRYPT 87, volume 304 of *LNCS*, pages 203–216, Berlin, 1987, Springer-Verlag.
7. I. Damgård. A design principle for hash functions. In CRYPTO 89, volume 435 of *LNCS*, pages 416–427, Berlin, 1989, Springer-Verlag.
8. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptively chosen message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
9. M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In Eurocrypt '96, volume 1070 of *LNCS*, pages, 143–154, Berlin, 1996, Springer-Verlag.
10. A.K. Lenstra and H.W. Lenstra Jr. *The Development of the Number Field Sieve*, Berlin, 1993, Springer-Verlag.
11. R. Merkle. One way hash functions and DES. In CRYPTO 89, volume 435 of *LNCS*, pages 428–446, Berlin, 1989, Springer-Verlag.
12. D. Pointcheval. The composite discrete logarithm and secure authentication. In PKC 2000, volume 1751 of *LNCS*, pages 113–128, Berlin, 2000, Springer-Verlag.
13. R.L. Rivest and R.D. Silverman. Are 'strong' primes needed for RSA. Report 2001/007, Cryptology ePrint Arhive, 2001. Available at `http://eprint.iacr.org/2001/007/`.
14. P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In B. Roy and W. Meier, editors, FSE 2004, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388, 2004. Springer-Verlag.
15. A. Shamir and Y. Tauman. Improved online/offline signature schemes. In CRYPTO 2001, volume 2139 of *LNCS*, pages 355–367, Berlin, 2001, Springer-Verlag.
16. R. Steinfeld, H. Wang, and J. Pieprzyk. Efficient extension of standard Schnorr/RSA signatures into universal designated-verifier signatures. In PKC 2004, volume 2947 of *LNCS*, pages 86–100, Berlin, 2004, Springer-Verlag.
17. G. Tenenbaum. *Introduction to analytic and probabilistic number theory*, Cambridge Univ. Press, 1995.
18. Wei Dai. *Crypto++ 5.2.1 Benchmarks*.
Available at `http://www.eskimo.com/~weidai/benchmarks.html`