VSH, an Efficient and Provable Collision Resistant Hash Function

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

[1] Department of Computing, Macquarie University, NSW 2109, Australia
[2] Lucent Technologies, Bell Laboratories, Room 2T-504
600 Mountain Avenue, P.O.Box 636, Murray Hill, NJ 07974-0636, USA
and Technische Universiteit Eindhoven
P.O.Box 513, 5600 MB Eindhoven, The Netherlands

Version 3.57

Abstract. We introduce VSH, *very smooth hash*, a new S-bit hash function that is provably collision-resistant assuming the hardness of finding nontrivial modular square roots of very smooth numbers modulo an S-bit composite integer n. By very smooth, we mean that the smoothness bound is some fixed polynomial function of S. We argue that finding collisions for VSH has the same asymptotic complexity as factoring using the Number Field Sieve factoring algorithm, i.e., subexponential in S.
VSH is theoretically pleasing because it requires only $O(\frac{1}{S})$ multiplications modulo the S-bit composite n per message-bit (as opposed to $\Omega(\frac{1}{\log S})$ multiplications for previous provably secure hashes). It is also practical. A preliminary implementation on a 1GHz Pentium III processor that achieves collision resistance at least equivalent to the difficulty of factoring a 1024-bit RSA modulus, runs at 1.1 MegaByte per second, with a moderate slowdown to 0.7MB/s for 2048-bit RSA security.
VSH can be used to build a fast, provably secure randomised trapdoor hash function, which can be applied to speed up provably secure signature schemes (such as Cramer-Shoup) and designated-verifier signatures.

Keywords: hashing, provable reducibility, integer factoring

1 Introduction

Current collision-resistant hash algorithms that have provable security reductions are too inefficient to be used in practice. One example [16, 19] that is provably reducible from 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 prespecified base value. A collision $x^m \equiv x^{m'} \bmod n$ reveals a multiple $m - m'$ of the order of x (which in itself divides $\phi(n)$). 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 multiplications modulo n per message-bit, it is quite inefficient. Improved provable algorithms exist [6] which require $\Omega(1/\log\log n)$ multiplies modulo n per message-bit, but beyond that it seems that so far all attempts to gain efficiency came at the cost of losing provability (see also [1]). We propose a hash algorithm that, as far as we are aware, improves upon the efficiency of previous provable hash algorithms. Our algorithm requires $O(\frac{\log\log n}{\log n})$ multiplications modulo n per message-bit, which is reduced to $O(\frac{1}{\log n})$ by a simple modification. It uses the same type of arithmetic as RSA, obviating the need for completely separate hash function code such as SHA-1. Our algorithm may therefore be useful in embedded environments where code space is limited.

We refer to our new hash as VSH, for *very smooth hash*, because finding a collision (i.e., strong collision resistance) is provably as difficult as finding a nontrivial modular square root of a very smooth number modulo n. Here *very smooth* means that the smoothness bound is some fixed polynomial function of $\log n$. Based on its connection to integer factorisation, we argue that it is reasonable to believe that it is hard to find a nontrivial modular square root of a very smooth number. We use NMSRVS to refer to our new hardness assumption.

Given this connection a natural question is if collisions can be created by a party that knows the factorisation of the VSH-modulus. That is indeed the case (cf. trapdoor hashes in [19]). Therefore, for wide-spread application of VSH with a single modulus one would have to rely on a trusted party that generates the modulus (and that can create collisions at will), or one would have to rely on the method from [2] to generate a modulus with knowledge of its factorisation shared among a group of authorities. Since the modulus generation is a one time computation the latter alternative looks more reasonable than the former. For application of VSH where each user would have a 'personalized' VSH-modulus, the repudiation concerns are not different from those concerning regular RSA.

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 $k < \log_2 n$ bits (compared to the fastest construction in [19], which requires about k modular multiplications). Randomised trapdoor hash functions are used in signature schemes to achieve provable security against adaptive chosen message attack [19], and in designated-verifier signature schemes to achieve privacy [10, 20]. For example, our function can replace the trapdoor function used in the Cramer-Shoup signature scheme [5], maintaining its provable security while speeding up verification time by about 50%.

We also present a variant of VSH using a prime modulus p (with no trapdoor), which has about the same efficiency as the composite modulus version, and is provably collision-resistant assuming the hardness of finding discrete logarithms of very smooth numbers modulo p.

Related Work. We mention several hash functions with collision resistance provably related to factoring which have been proposed in the literature. They all have lower efficiency than VSH. The $x^m \bmod n$ function mentioned above appeared in [16, 19]. 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 in [8]—this function requires 1 squaring per bit processed. In [6] the construction is generalised to use families of $r \geq 2$ claw free permutations, such that $\log_2(r)$ bits can be processed per permutation evaluation. Two factoring based constructions are presented, which require 2 modular multiplications per permutation evaluation. In the first construction the modulus n has $1+\log_2(r)$ prime factors and thus becomes impractical already for small $\log_2(r)$. The second one uses a regular RSA modulus, but requires publishing r random quadratic residues modulo n. This becomes prohibitive too for relatively small $\log_2(r)$; as a result the construction requires $\Omega(1/\log S)$ multiplications modulo an S-bit RSA modulus n per message bit while consuming polynomial space ($r = O(poly(S))$). The constructions in [1] are more efficient but are only provably collision-resistant assuming an underlying hash function is modeled as a random oracle (while we make no such assumption).

Section 2 introduces security definitions and assumptions related to the NM-SRVS problem. The VSH algorithm and some of its variations are presented in Section 3. Section 4 describes a VSH-based randomised trapdoor hash function and shows how to speed up the provably secure Cramer-Shoup signature scheme. Section 5 concludes the paper with some implementation results.

2 Security Definitions

Notation. In this paper the following values and notation is used. Let $c > 0$ be a fixed constant and let n be a hard to factor S-bit composite for an integer $S > 0$. The ring of integers modulo n is denoted $\mathbf{Z}_n$, and its elements are represented by $\{0, 1, \ldots, n-1\}$ or $\{-n+1, -n+2, \ldots, 0\}$. It will be clear from the context which representation is being used. The ith prime is denoted p_i: $p_1 = 2$, $p_2 = 3$, $\ldots$, and $p_0 = -1$. An integer is p_k-smooth if all its prime factors are $\leq p_k$.

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

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

Trivial modular square roots have no relation to the modulus n. Such identities are easy to create, and therefore they are not allowed in the security reduction. A sufficient condition for a very smooth integer b representing a quadratic residue not to have a trivial modular square root is having some prime p such that p divides b but p^2 does not. Another sufficient condition is that b is negative. Our new hardness assumption is that it is difficult to find a nontrivial modular square root of a very smooth quadratic residue modulo n. Before formulating our assumption, we give some relevant background on integer factorisation.

Background on general purpose integer factorisation. General purpose factoring algorithms are used for the security evaluation of RSA, since they do not take advantage of properties of the factors. They all work by constructing nontrivial congruent squares modulo n since such squares can be used to factor n:

if $x, y \in \mathbf{Z}$ are such that $x^2 \equiv y^2 \bmod n$ and $x \not\equiv \pm y \bmod n$, then $\gcd(x \pm y, n)$ are proper factors of n. To construct such x, y a common strategy uses so-called *relations*. An example of a relation would be an identity of the form

$$v^2 \equiv \prod_{0 \le i \le u} p_i^{e_i(v)} \bmod n,$$

where u is some fixed integer, $v \in \mathbf{Z}$, and $(e_i(v))_{i=0}^u$ is a $(u+1)$-dimensional integer vector. Given $u + 1 + t$ relations, at least t linearly independent dependencies modulo 2 among the $u + 1 + t$ vectors $(e_i(v))_{i=0}^u$ can be found using linear algebra. Each such dependency corresponds to a product of v^2-values that equals a product modulo n of p_i's with all even exponents, and thus a solution to $x^2 \equiv y^2 \bmod n$. If $x \not\equiv \pm y \bmod n$, then it leads to a proper factor of n. A relation with all even exponents $e_i(v)$ leads to a pair x, y right away, which has, in our experience with practical factoring algorithms, never happened unless n is very small. It may safely be assumed that for each relation found at least one of the $e_i(v)$'s is odd—actually most that are non-zero will be equal to 1.

For any u, relations are easily computed if n's factorisation is known, since square roots modulo primes can be computed efficiently and the results can be assembled via the Chinese Remainder Theorem. If the factorisation is unknown, however, relations in practical factoring algorithms are found by a deterministic process that depends on the factoring algorithm used. It is sufficiently unpredictable that the resulting x, y may be assumed to be random solutions to $x^2 \equiv y^2 \bmod n$, implying that the condition $x \not\equiv \pm y \bmod n$ holds for at least half of the dependencies. Despite the lack of a rigorous proof, this heuristic argument has not failed us yet. A few dependencies usually suffice to factor n.

The expected relation collection runtime is proportional to the product of u (approximately the number of relations one needs) and the inverse of the smoothness probability of the numbers that one hopes to be p_u-smooth, since this probability is indicative for the efficiency of the collection process. For the fastest factoring algorithms published so far, the Number Field Sieve (NFS, cf. [12, 4, 3]), the overall expected runtime (including the linear algebra) is minimized—based on loose heuristic grounds—when, asymptotically for $n \to \infty$, u behaves as $L[n, 0.96...]$. Here

$$L[n, \alpha] = e^{(\alpha + o(1))(\log n)^{1/3}(\log \log n)^{2/3}}$$

for constant $\alpha > 0$ and $n \to \infty$, and where the logarithms are natural. The overall NFS expected runtime asymptotically, and heuristically, behaves as the square of the optimal u, i.e., $L[n, 1.923...]$.

With the current state of the art of integer factorisation, one cannot expect that, for any value of u, a relation can be found faster than $L[n, 1.923...]/u$ on average, asymptotically for $n \to \infty$. For u-values much smaller than the optimum, the actual time to find a relation will be considerably larger (cf. remark below and [13]). For $u \approx (\log n)^c$, it is conservatively estimated that finding a relation requires runtime at least

$$\frac{L[n, 1.923...]}{(\log n)^c} = L[n, 1.923...],$$

asymptotically for $n \to \infty$, because the denominator gets absorbed in the numerator's $o(1)$. This observation that finding relations for very small u (i.e., u's that are bounded by a polynomial function of $\log n$) can be expected to be asymptotically as hard as factoring n, is the basis for our new hardness assumption.

Before formulating it, we present two ways to use the hardness estimate $L[n, 1.923...]/u$ for small u in practice. One way is to use the asymptotics and assume that finding a relation is as hard as factoring n. A more conservative approach incorporates the division by u in the estimate. In theory this is a futile exercise because, as argued, a polynomially bounded u disappears in the $o(1)$ for $n \to \infty$. In practice, however, n does not go to infinity but actual values have to be dealt with. If n' is a hard to factor integer for which $\log n$ and $\log n'$ are relatively close, then it is widely accepted that the ratio of the NFS-factoring runtimes for n and n' approximates $L[n, 1.923...]/L[n', 1.923...]$ where the $o(1)$'s are dropped. To assess the hardness estimate $L[n, 1.923...]/u$ for very small u, one therefore finds the least integer S' for which, after dropping the $o(1)$'s,

$$L[2^{S'}, 1.923...] \geq \frac{L[n, 1.923...]}{u}, \tag{1}$$

and assumes that finding a relation for this n and u may be expected to be (at least) as hard as NFS-factoring a hard to factor S'-bit integer. Note that S' will be less than S, the length of n. Examples of matching S, S', u values are given in Section 5.

This factoring background provides the proper context for our new problem and its hardness assumption.

Definition 3. *(NMSRVS: Nontrivial Modular Square Root of Very Smooth numbers) Let n be the product of two unknown primes of approximately the same size and let $k \leq (\log n)^c$. The NMSRVS problem is the following: Given n, find $x \in Z_n^*$ such that $x^2 \equiv \prod_{i=0}^{k} p_i^{e_i} \bmod n$ 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 $\log n$) 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 the factors of n are chosen randomly, as required. According to proper security definitions [17], 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 useless in practice since it does not tell us for what size moduli the NMSRVS problem would be sufficiently hard. This is similar to the situation in integer factorisation where the hardness assumption does not suffice to select secure modulus sizes. We therefore make an additional, stronger assumption that links the hardness of the NMSRVS problem to the current state of the art in factoring. It is based on the conservative estimate for the difficulty of finding a relation for very small u given above.

Computational NMSRVS Assumption. The computational NMSRVS assumption is that solving the NMSRVS problem is as hard as factoring a hard to factor S'-bit modulus, where S' is the least positive integer for which equation (1) holds (where, as in (1), the $o(1)$'s in the $L[...]$'s are dropped).

Remark. For existing factoring algorithms, the relation collection runtime increases sharply for smoothness bounds that are too low, almost disastrously so if the bound is taken as absurdly low as in the NMSRVS problem (cf. [13]). Therefore, the Computational NMSRVS Assumption is certainly overly conservative. Just assuming—as suggested above—that solving the NMSRVS problem is as hard as factoring n may be more accurate. Nevertheless, the runtime estimates for our new hash function will be based on the overly conservative Computational NMSRVS Assumption.

Although our analysis is based on the *average* runtime to find a relation using the NFS, it is very conservative (i.e., leads to a large n) compared to a more direct analysis involving the relevant smoothness probability of squares modulo n. The latter would lead to a hardness estimate more similar to the runtime of the Quadratic Sieve integer factorisation algorithm, and thereby to much smaller 'secure' modulus sizes. Thus, we feel more comfortable using our NFS-based approach.

3 Very Smooth Hash Algorithm

The basic version of VSH follows below. More efficient variants of VSH are discussed later in 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:

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

Compression function H**.** VSH applies the compression function $H(x, m)$: $\mathbf{Z}_n^* \times \{0,1\}^k \to \mathbf{Z}_n^*$ with $H(x, m) = x^2 \prod_{i=1}^{k} p_i^{m_i} \bmod n$, and applies a variant of the Merkle-Damgård transformation [14, 7] to extend H to arbitrarily long inputs. Other ways to implement this idea to improve efficiency and to deal with the case where the length is not known ahead of time are treated in Section 4.

1024-bit n. For 1024-bit n, the value for k would be 131. The requirement $\ell < 2^{k-2}$ is therefore not a problem in any real application, and most of the bits ℓ_i will be zero. The Computational NMSRVS Assumption with $S = 1024$ and $k = u = 131$ leads to $S' = 840$. The security level obtained by VSH using 1024-bit n is therefore at least the security level obtained by 840-bit RSA.

Efficiency. Because $\prod_{0<i\leq K} p_i$ is asymptotically proportional to $e^{K \log K}$, the k used in the basic version of VSH is proportional to $\frac{\log n}{\log\log n}$. It follows that the product $\prod_{i=1}^{k} p_i^{m_{j \cdot k+i}}$ can be computed in time $O((\log n)^2)$ using straightforward multiplication without modular reduction. Therefore the cost of each iteration is less than the cost of 3 modular multiplications. Since k bits are processed per iteration, the basic version of VSH requires $O(\frac{\log\log n}{\log n})$ modular multiplications per message-bit, with a small constant in the big-Oh.

Creating collisions. With $e_i = \sum_{j=0}^{\mathcal{L}-1} m_{j \cdot k+i} 2^{\mathcal{L}-j-1}$ for $1 \leq i \leq k$, the value calculated by the VSH algorithm is the same as the multi-exponentiation $\prod_{i=1}^{k} p_i^{e_i} \bmod n$, except for powers of the initial factor x_0. Given $\phi(n)$ and assuming large enough $\mathcal{L}$, collisions can be generated by replacing e_i by $e_i + t_i\phi(n)$ for any set of i's with $1 \leq i \leq k$ and positive integers t_i (see also VSH-DL below). Thus, parties that know n's factorisation can create collisions at will. But collisions of this sort immediately reveal $\phi(n)$ and thereby 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 PKIs to allow one to choose one's own hash function, rather than using a 'fixed target' for all. In this setting, one 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 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.

Short message inversion. The VSH algorithm described above allows easy inversion of messages of length $\ell \leq k$: if the resulting hash is divided modulo n by each of the k possibilities for x_0^2, one of the k resulting 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 m. We emphasize that this type of invertibility may be undesirable for some applications, but that other applications require just collision resistance (cf. below). See [15] for a related application.

A 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. Note that for all hashes the hash of an extremely short message can always be 'inverted' by trial and error.
Multiplicative properties. It is easy to find messages for which the hashes $\hbar$ and $\hbar'$ satisfy possibly undesirable multiplicative properties such as $\hbar = 2\hbar'$. Our solution to the invertibility problem addresses this issue as well.
Having stressed upfront in the last three remarks the disadvantages of VSH, we 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 [18], (strong) collision resistance also implies second preimage resistance.

Theorem 1. *Finding a collision in VSH is as hard as solving the NMSRVS problem (i.e., VSH is collision-resistant under the assumptions from Section 2).*

Proof. We show that different colliding messages m and m' lead to a solution of the NMSRVS problem. Let $x'_{\dots}$ denote the $x_{\dots}$ values in the VSH algorithm applied to m' and let $\ell, \mathcal{L}$ and $\ell', \mathcal{L}'$ be the bit- and blocklengths of m and m', respectively. Since m and m' collide, $m \neq m'$ and $x_{\mathcal{L}} = x'_{\mathcal{L}'}$. Working backwards from the last blocks of m and m', let j be the smallest positive integer (if it exists) such that $x_{\mathcal{L}-j+1} = x'_{\mathcal{L}'-j+1}$ but either $x_{\mathcal{L}-j} \neq x'_{\mathcal{L}'-j}$ or $m_{(\mathcal{L}-j)\cdot k+i} \neq m'_{(\mathcal{L}'-j)\cdot k+i}$ for some $i \in \{1, \dots, k\}$. If $\ell = \ell'$, such j exists because $m \neq m'$, and if $\ell \neq \ell'$ but $\mathcal{L} = \mathcal{L}'$, it exists because $x_0 \neq x'_0$. The only case where j may not exist is $\mathcal{L} \neq \mathcal{L}'$, and in this case (assuming, without loss of generality, that $\mathcal{L} < \mathcal{L}'$) if j does not exist we must have $x_0 = x'_{\mathcal{L}'-\mathcal{L}}$.

If j as above exists, let $a = \mathcal{L} - j$ and $b = \mathcal{L}' - j$. If $a > 0$ and $b > 0$ then

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

Let $\Delta = \{i : m_{a\cdot k+i} \neq m'_{b\cdot k+i}, 1 \leq i \leq k\}$ and $\Delta_{10} = \{i \in \{1, \dots, k\} : m_{a\cdot k+i} = 1 \text{ and } m'_{b\cdot k+i} = 0\}$. Because all factors in Equation (2) are invertible modulo n, it is equivalent to

$$\left[(x_a/x'_b) \times \prod_{i\in\Delta_{10}} p_i\right]^2 \equiv \prod_{i\in\Delta} p_i \bmod n \ . \tag{3}$$

If $\Delta \neq \emptyset$, Equation (3) solves the NMSRVS problem. If $\Delta = \emptyset$, then $(x_a)^2 \equiv (x'_b)^2 \bmod n$. With $x_a \not\equiv \pm x'_b \bmod n$ the NMSRVS problem can be solved by factoring n. If $x_a \equiv \pm x'_b \bmod n$ then $x_a \equiv -x'_b \bmod n$, since $\Delta = \emptyset$ implies (by definition of a and b) that $x_a \neq x'_b$. But $x_a \equiv -x'_b \bmod n$ leads to $(x_{a-1}/x'_{b-1})^2$ being congruent to -1 times a very smooth number and thus solves the NMSRVS problem. This completes the case of $a > 0$ and $b > 0$.

If both a and b are 0, then a similar argument holds, the only difference being that $x_0 \equiv -x'_0 \bmod n$ cannot hold because $x_0, x'_0 < n/2$ due to $\ell < 2^{k-2}$.

If exactly one of a and b is zero, we may assume without loss of generality that $a = 0$ and $b > 0$. Then a similar argument to the above holds: the only difference is that $x_0 \equiv -x'_b \bmod n$ implies $p_{k+1} \times \prod_{i=1}^{k-2} p_i^{\ell_i} \equiv -(x'_{b-1})^2 \times \prod_{i=1}^{k} p_i^{m'_{(b-1)\cdot k+i}} \bmod n$, and hence leads to $(x'_{b-1})^2$ being congruent to -1 times a very smooth number and thus solves the NMSRVS problem. This completes the case where j defined above exists.

Finally, if j does not exist, then $x_0 = x'_b$, where $b = \mathcal{L}' - \mathcal{L} > 0$. This implies that $p_{k+1} \times \prod_{i=1}^{k-2} p_i^{\ell_i} \equiv (x'_{b-1})^2 \times \prod_{i=1}^{k} p_i^{m'_{(b-1)\cdot k+i}} \bmod n$. Because of the odd number of occurrences of p_{k+1} in this congruence, applying a transformation similar to the one above solves the NMSRVS problem. $\square$

Remark. A more refined version of the argument in the last paragraph of the proof allows a modification of VSH that omits p_{k+1}.

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_1, r_2, ... of $K > 1$ bits and uses the compression function $x_{j+1} := x_j^2 \times 2^{r_{j+1}} \bmod n$. Because $K > 1$ collisions can simply be created. For example, for any e with $0 < e < 2^{K-1}$ the message blocks $r_1 = e$ and $r_2 = 2e$ collide with $r'_1 = 2e$ and $r'_2 = 0$. The colliding hashes are $(x_0^2 2^e)^2 2^{2e}$ and $(x_0^2 2^{2e})^2 2^0$ respectively, for some x_0, but the collision does not lead to a solution of the NMSRVS problem or a possibility to factor n. Such trivial relations are useless, and thus the security of this hash algorithm is not reducible to any hard problem. The fix is to replace $x_{j+1} := x_j^2 \times 2^{r_{j+1}}$ by the costlier variant $x_{j+1} := x_j^{2^K} \times 2^{r_{j+1}}$, but that is the same as the function $x^m \bmod n$ from [16, 19].

3.3 Combining VSH and RSA

Since the output length of VSH is the length of a secure RSA modulus (thus 1024–2048 bits), VSH seems quite suitable in practice for constructing 'hash-then-sign' RSA signatures for arbitrarily long messages. However, such a signature scheme must be designed carefully to ensure its security. To illustrate a naive insecure scheme, let (n, e) be the signer's public RSA key, where the modulus n is used for both signing and hashing. The signing function $\sigma : \{0,1\}^* \to \mathbf{Z}_n$ is $\sigma(m) = VSH_n(m)^{1/e} \bmod n$, where $VSH_n : \{0,1\}^* \to \mathbf{Z}_n$ is VSH with modulus n. For a k-bit message $m = (m_1, \ldots, m_k) \in \{0,1\}^k$, the signature is thus $\sigma(m) = (x_0^2 \prod_{i=1}^{k} p_i^{m_i})^{1/e} \bmod n$, where x_0 is the same for all k-bit messages.

This scheme is insecure under the following chosen message attack. After obtaining signatures on three k-bit messages: $s_0 = \sigma((0,0,0,\ldots,0)) = (x_0^2)^{1/e} \bmod n$, $s_1 = \sigma((1,0,0,\ldots,0)) = (x_0^2 \cdot p_1)^{1/e} \bmod n$, and $s_2 = \sigma((0,1,0,\ldots,0)) = (x_0^2 \cdot p_2)^{1/e} \bmod n$, the attacker easily computes the signature $\frac{s_1 s_2}{s_0} \bmod n$ on the new k-bit forgery message $(1, 1, 0, \ldots, 0)$. It is easy to see that the $k+1$ signatures on $k+1$ properly chosen messages suffice to sign any k-bit message.

To avoid such attacks, we suggest a more theoretically sound design approach for using VSH with 'hash-then-sign' RSA signatures that does not rely on any property of VSH beyond the collision resistance which it was designed to achieve:
Step 1. Let $\bar{n}$ be an $(S+1)$-bit RSA modulus, with $\bar{n}$ and the S-bit VSH modulus n chosen independently at random. So, $\bar{n} > 2^S$. Specify a one-to-one one-way encoding function $f : \{0,1\}^S \to \{0,1\}^S$, and define the short-message (S-bit) RSA signature scheme with signing function $\sigma_{\bar{n}}(m) = (f(m))^{1/e} \bmod \bar{n}$. The function f is chosen such that the short-message scheme with signing function $\sigma_{\bar{n}}$ is existentially unforgeable under chosen message attack. In the standard model no provable techniques are known to find f, but since f is one-to-one, there are no collision resistance issues to consider when designing f.
Step 2. With $(\bar{n}, n, e)$ as the signer's public key, the signature scheme for signing arbitrary length messages is now constructed with signing function $\sigma_{\bar{n},n}(m) = \sigma_{\bar{n}}(VSH_n(m))$. It is easy to prove that the scheme with signing function $\sigma_{\bar{n},n}$ is existentially unforgeable under chosen message attack, assuming that the scheme with signing function $\sigma_{\bar{n}}$ is and that VSH_n is collision-resistant. We emphasize that the proof no longer holds if $\bar{n} = n$: in order to make the proof work in that case, one needs the stronger assumption that VSH_n is collision-resistant even given access to a signing oracle σ_n. However, it is worth remarking that if the function f is modeled as a random oracle, then the proof of security works (under the RSA and NMSRVS assumptions) even with a shared modulus ($\bar{n} = n$).

3.4 Variants of VSH

Cubing instead of squaring. Let $H' : \mathbf{Z}_n^* \times \{0,1\}^k \to \mathbf{Z}_n^*$ with $H'(x, m) = x^3 \prod_{i=1}^{k} p_i^{m_i} \bmod n$ be a compression function that replaces the squaring in H by a cubing. If $\gcd(3, \phi(n)) = 1$ then thanks to the injectivity of the RSA cubing map modulo n, the function H' is collision-resistant, assuming the difficulty of computing a modular cube root of a very smooth cube-free integer of the form $\prod_{i=1}^{k} p_i^{e_i} \neq 1$, where $e_i \in \{0, 1, 2\}$ for all i. This problem is related to RSA inversion, and is also conjectured to be hard. Although H' requires about 4 modular multiplications per k message bits (compared to 3 for H), it has the interesting property that H' itself is collision-resistant, while this is not quite the case for H (because $x^2 \prod_i {p_i}^{m_i} \equiv (-x)^2 \prod_i p_i^{m_i} \bmod n$).
Increasing the number of small primes. A speed-up is obtained by allowing the use of larger k than the largest one for which $\prod_{i=1}^{k} p_i < n$. This does not affect the proof of security and reduction to the NMSRVS problem, as long as k is still polynomially bounded in $\log n$. The Computational NMSRVS Assumption implies that a larger modulus 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 resulting smaller $\mathcal{L}$ may outweigh these disadvantages.
Precomputing products of primes. An implementation speed-up may be obtained by precomputing products of primes. Let $b > 1$ be a small integer,

and assume that $k = \bar{k}b$ for some integer $\bar{k}$. For $i = 1, 2, \ldots, \bar{k}$ compute the 2^b products over all subsets of the set of b primes $\{p_{(i-1)b+1}, p_{(i-1)b+2}, \ldots, p_{ib}\}$, resulting per i in 2^b moderately sized values $v_{i,t}$ for $0 \leq t < 2^b$. The k message-bits per iteration of VSH are now split into $\bar{k}$ chunks $m[0], m[1], \ldots, m[\bar{k}-1]$ of b bits each, interpreted as non-negative integers $< 2^b$. The usual product is then calculated as $\prod_{i=1}^{\bar{k}} v_{i,m[i-1]}$. This has no effect on the number of iterations or the modulus size to be used to achieve a certain level of security.

Fast VSH. Redefining the above $v_{i,t}$ as $p_{(i-1)2^b+t+1}$ and using $i = 1, 2, \ldots, k$ instead of $i = 1, 2, \ldots, \bar{k}$, the block length increases from k to bk, the number of iterations is reduced from $\lceil \frac{\ell}{k} \rceil$ to $\lceil \frac{\ell}{bk} \rceil$, and the calculation in Step 4 of the VSH algorithm becomes

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

where $m[r]$ is the rth b-bit chunk of the message, with $0 \leq m[r] < 2^b$. Because the number of small primes increases from k to $k2^b$, a larger modulus would, conservatively, have to be used to maintain the same level of security. But this change does not affect the proof of security and, as shown in the analysis below and the runtime examples in the final section, it is clearly advantageous.

Analysis of Fast VSH. Since $p_{(i-1)2^b+m[jbk+i-1]+1} \leq p_{i2^b}$, each intermediate product in the compression function for Fast VSH will be less than n if $\prod_{i=1}^{k} p_{i2^b} < n$. If k is maximal such that $\prod_{i=1}^{(k+1)2^b} p_i \leq (2n)^{2^b}$, then

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

so that $\prod_{i=0}^{k} p_{i2^b+1} \leq 2n$. With $p_{i2^b} < p_{i2^b+1}$ it follows that $\prod_{i=1}^{k} p_{i2^b} < n$. Thus, for $(k+1)2^b$ proportional to $\frac{2^b \log(2n)}{\log(2^b \log(2n))}$ and k to $\frac{\log(2n)}{\log(2^b \log(2n))} - 1$, the cost of Fast VSH is $O(\frac{1}{bk})$ modular multiplications per message-bit, with bk proportional to $\frac{b \log(2n)}{\log(2^b \log(2n))} - b$. Selecting 2^b as any fixed positive power of $\log(2n)$, it follows that bk is proportional to $\log n$ and thus that Fast VSH requires $O(\frac{1}{\log n})$ modular multiplications per message bit. It also follows that the number of small primes $k2^b$ is polynomially bounded in $\log n$ so that, with S' the overly conservative RSA security level obtained according to the Computational NMSRVS Assumption, Fast VSH requires $O(\frac{1}{S'})$ modular multiplications per message bit.

Zero chunks in Fast VSH. In Fast VSH a prime is always multiplied in, even for a b-bit chunk of zero bits. In the original VSH, a chunk of zero bits does not result in any primes multiplied in. A negligible speed-up and tiny saving in the number of primes can be obtained in Fast VSH if for a particular b-bit pattern (such as all zeros) no prime is multiplied in.

Fast VSH with increased block length. Fast VSH can be used in a straightforward fashion with a larger block length than suggested by the above analysis. 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).

Generating collisions. For all variants 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 it would give out a break to the user's hash function (i.e. it would be similar to someone giving out the factorisation of their RSA modulus).

VSH-DL, a discrete logarithm variant. We present a discrete logarithm (DL) variant of VSH that has no trapdoor. Its security depends on the following problem and its hardness assumption.

Definition 4. *(NDLVS: Nontrivial Discrete-Log of Very Smooth numbers) Let* p, q *be primes with* $p = 2q + 1$ *and let* $k \leq (\log p)^c$*. The NDLVS problem is the following: given* p*, find integers* $e_1, e_2, \ldots, e_k$ *such that* $2^{e_1} \equiv \prod_{i=2}^{k} p_i^{e_i} \bmod p$ *with* $|e_i| < q$ *for* $i = 1, 2, \ldots, k$*, and at least one of* $e_1, e_2, \ldots, e_k$ *is non-zero.*

NDLVS Assumption. The NDLVS assumption is that there is no probabilistic polynomial (in $\log p$) time algorithm which solves the NDLVS problem with non-negligible probability (the probability is taken over the random choice of the prime p and the random coins of the algorithm).

A solution to an NDLVS instance produces the base 2 DL modulo p of a very smooth number (the requirements on the exponents e_i avoids trivial solutions in which all exponents are zero modulo q). Given k random NDLVS solutions, the base 2 DL of nearly all primes $p_1, \ldots, p_k$ can be solved with high probability by linear algebra modulo q. The (hard) 'precomputation' stage of all known index calculus DL algorithms [4] works in a similar way, but solves the DL for subexponentially many small primes. Although computing the DLs of a polynomial number of small primes is an impressive feat, it does not help to solve arbitrary DL problems faster than via the usual subexponential-time detour, at least not as far as we know. Nevertheless, there is a strong connection between the hardness of NDLVS and the hardness of computing DLs modulo p. This connection is analogous to, but seems to be somewhat weaker than, the connection between NMSRVS and factorisation. As was the case for NMSRVS, moduli for which NDLVS is not difficult are easily constructed. For the same reasons as for NMSRVS, they are not worthy of further consideration.

Let p be an S-bit prime of the form $2q+1$ for prime q, let k be a fixed integer length (number of small primes, typically $k \approx S$), and let $\mathcal{L} \leq S-2$. We define a VSH-DL compression function $H_{DL} : \{0,1\}^{\mathcal{L}k} \to \{0,1\}^S$, where m is an $\mathcal{L}k$-bit message consisting of bits $m_1, \ldots m_{\mathcal{L}k}$:

– Set $x_0 = 1$. For $j = 0, 1, \ldots, \mathcal{L}-1$, compute $x_{j+1} = x_j^2 \times \prod_{i=1}^{k} p_i^{m_{j \cdot k+i}} \bmod p$.
– Return $H_{DL}(m) = x_{\mathcal{L}}$ interpreted as a value in $\{0,1\}^S$.

If $e_i = \sum_{j=0}^{\mathcal{L}-1} m_{j \cdot k+i} 2^{\mathcal{L}-j-1}$ for $1 \leq i \leq k$, then $H_{DL}(m) = \prod_{i=1}^{k} p_i^{e_i} \bmod p$. A collision $m, m' \in \{0,1\}^{\mathcal{L}k}$ with $m \neq m'$ therefore implies that $\prod_{i=1}^{k} p_i^{e_i} \equiv$

$\prod_{i=1}^{k} p_i^{e'_i} \bmod p$, where $e'_i = \sum_{j=0}^{\mathcal{L}-1} m'_{j\cdot k+i} 2^{\mathcal{L}-j-1}$ and m' consists of the bits $m'_1, \ldots m'_{\mathcal{L}k}$. Rearranging this congruence, a solution $2^{e_1 - e'_1} \equiv \prod_{i=2}^{k} p_i^{e'_i - e_i} \bmod p$ to the NDLVS problem follows, because $|e'_i - e_i| < 2^{\mathcal{L}} \leq 2^{S-2} \leq q$ for all i and $e'_i - e_i \neq 0$ for some i since $m \neq m'$. Hence the compression function H_{DL} is collision-resistant under the NDLVS assumption.

The compression function H_{DL} uses the same iteration as the basic VSH algorithm. Hence, for the same modulus length S and number of primes k it has the same throughput efficiency of about $\frac{3}{k}$ modular multiplications per message-bit. By applying the Merkle-Damgård transformation [14, 7], H_{DL} can be used to hash messages of arbitrary length in blocks of $\mathcal{L}k - S$ message bits per evaluation of H_{DL}. This leads to a reduction in throughput by a factor of $\frac{\mathcal{L}k-S}{\mathcal{L}k}$ (since only $\mathcal{L}k - S$ of the $\mathcal{L}k$ bits processed in each H_{DL} evaluation are new message bits) relative to factoring based VSH. However, for long messages, this throughput reduction factor can be made close to 1 by choosing a sufficiently large block length $\mathcal{L}k$; indeed, the construction allows block lengths up to $\mathcal{L}k = k(S-2)$, and for this choice the throughput reduction factor is $1 - \frac{S}{k(S-2)} \approx 1 - \frac{1}{k} \approx 1$.

Reducing the hash length. A possible drawback of VSH is its relatively large length. We are currently investigating the possibility to reduce the hash length by combining VSH-DL with elliptic curve, XTR, or torus-based techniques [9, 11, 21].

4 VSH Randomised Trapdoor Hash and Applications

Let $\mathcal{M}, \mathcal{R}, \mathcal{H}$ be a message, randomiser, and hash space, respectively. A *randomised trapdoor hash function* [19] $F_{pk} : \mathcal{M} \times \mathcal{R} \to \mathcal{H}$ is a collision-resistant function that can be efficiently evaluated using a public key pk, but for which certain randomly behaving collisions can be found given a secret trapdoor key sk:

Collision Resistance in Message Input: Given pk, it is hard to find $m, m' \in \mathcal{M}$ and $r, r' \in \mathcal{R}$ for which $m \neq m'$ and $F_{pk}(m,r) = F_{pk}(m',r')$.

Random Trapdoor Collisions: There exists an efficient algorithm that given trapdoor (sk, pk), $m, m' \in \mathcal{M}$ with $m \neq m'$, and $r \in \mathcal{R}$, finds a randomiser $r' \in \mathcal{R}$ such that $F_{pk}(m,r) = F_{pk}(m',r')$. Furthermore, 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 [19], and constructing designated-verifier proofs/signature schemes [10, 20]. The factorisation trapdoor of VSH suggests that it can be used to build such a function. Here we describe a provably secure randomised trapdoor hash family which preserves the efficiency of VSH.

Key Generation: Choose two $S/2$-bit random primes p, q with $p \equiv q \equiv 3 \bmod 4$ and S-bit product n. The public key is n with trapdoor key $sk = (p,q)$. Let k be as in the basic VSH algorithm, $\mathcal{M} = \cup_{\ell=0}^{2^k-1} \{0,1\}^{\ell}$, and $\mathcal{R} = \mathbf{Z}_n^*$.

Hash Function: Let $m \in \mathcal{M}$ of length $\ell < 2^k$ and $r \in \mathcal{R}$. Calculate the basic VSH of m with x_0 replaced by r to compute $x_{\mathcal{L}}$, let $x_{\mathcal{L}+1} = H(x_{\mathcal{L}}, \ell)$ where H is the VSH compression function, and output $F_n(m,r) = x_{\mathcal{L}+1}^2 \bmod n$.

Theorem 2. *The above construction satisfies the security requirements for randomised trapdoor hash functions, under the NMSRVS assumption.*

Proof. For (padded) $m \in \mathcal{M}$ and $0 \le j < \mathcal{L}$ let $m[j]_i$ denote the ith bit of m's jth k-bit block $m[j] = (m_{j\cdot k+i})_{i=1}^k$ and $\ell = \sum_{i=1}^k \ell_i 2^{i-1}$ with $\ell_i \in \{0,1\}$.
Collision Resistance in Message Input. Let $m, m' \in \mathcal{M}$ of lengths ℓ, ℓ', respectively, and $r, r' \in \mathcal{R}$, such that $m \neq m'$ and $F_n(m,r) = F_n(m',r')$. Denoting the intermediate values while computing $F_n(m,r)$ and $F_n(m',r')$ by $x_{\dots}$ and $x'_{\dots}$, respectively, it follows that $(x_{\mathcal{L}+1})^2 \equiv (x'_{\mathcal{L}'+1})^2 \bmod n$, where $\mathcal{L} = \lceil \frac{\ell}{k} \rceil$ and $\mathcal{L}' = \lceil \frac{\ell'}{k} \rceil$. Assuming that $x_{\mathcal{L}+1} \equiv \pm x'_{\mathcal{L}'+1} \bmod n$—otherwise n can be factored and the NMSRVS problem solved—it follows that $(x_{\mathcal{L}}/x'_{\mathcal{L}'})^2 \equiv \pm \prod_{i=1}^k p_i^{\ell'_{i-1}-\ell_{i-1}} \bmod n$. If $\ell \neq \ell'$, then $|\ell'_i - \ell_i| = 1$ for at least one i, thereby solving the NMSRVS problem. Otherwise, if $\ell = \ell'$, let $t \le \mathcal{L}+1$ be the largest index such that $(x_t, m[t]) \neq (x'_t, m'[t])$ but $(x_j, m[j]) = (x'_j, m'[j])$ for $j > t$ (where we define $m[\mathcal{L}] = m'[\mathcal{L}] = \ell$ and $m[j] = m'[j] = 0$ for $j > \mathcal{L}$). So $(x_t/x'_t)^2 \equiv \prod_{i=1}^k p_i^{m'[t]_i - m[t]_i} \bmod n$. If $m[t] \neq m'[t]$, then $|m'[t]_i - m[t]_i| = 1$ for at least one i, thereby solving the NMSRVS problem. Otherwise, if $m[t] = m'[t]$, then $x_t \neq x'_t$. Combined with $(x_t)^2 \equiv (x'_t)^2 \bmod n$ it follows that n can be factored and the NMSRVS problem solved if $x_t \neq -x'_t \bmod n$, or that $x_t \equiv -x'_t \bmod n$. But with $t \ge 1$ (because $m \neq m'$ and $(x_j, m[j]) = (x'_j, m'[j])$ for $j \ge t$), $x_t \equiv -x'_t \bmod n$ implies that $(x_{t-1}/x'_{t-1})^2 \equiv -\prod_{i=1}^k p_i^{m'[t-1]_i - m[t-1]_i} \bmod n$, which solves the NMSRVS problem due to the -1 factor on the right hand side.
Random Trapdoor Collisions: Let $m, m' \in \mathcal{M}$ with $m \neq m'$ and $r \in \mathcal{R}$. Because $F_n(m,r) = (r^{2^{\mathcal{L}+1}} \prod_{i=1}^k p_i^{2e_i + \ell_{i-1}})^2 \bmod n$ where $e_i = \sum_{j=0}^{\mathcal{L}-1} m[j]_i 2^{\mathcal{L}-j-1}$, finding $r' \in \mathcal{R}$ with $F_n(m,r) = F_n(m',r')$ amounts to finding r' such that

$$(r')^{2^{\mathcal{L}'+2}} \equiv r^{2^{\mathcal{L}+2}} \cdot (\prod_{i=1}^k p_i^{2e_i + \ell_{i-1} - 2e'_i - \ell'_{i-1}})^2 \bmod n,$$

i.e., finding an $(\mathcal{L}'+2)$nd square root modulo n of the right hand side g of the equation for $(r')^{2^{\mathcal{L}'+2}}$. Given the trapdoor key (p,q) this is achieved as follows.

Let $QR_n = \{y \in \mathbf{Z}_n^* : (\frac{y}{p}) = (\frac{y}{q}) = 1\}$ denote the subgroup of quadratic residues of $\mathbf{Z}_n^*$. The choice $p \equiv q \equiv 3 \bmod 4$ implies that -1 is a quadratic non-residue in $\mathbf{Z}_p^*$ and $\mathbf{Z}_q^*$, so for each element of QR_n exactly one of its 4 square roots in $\mathbf{Z}_n^*$ belongs to QR_n. Hence the squaring map on QR_n permutes QR_n and given (p,q) it can be efficiently inverted by computing the proper square roots modulo p and q and combining them by Chinese remaindering. Since $g \in QR_n$, its $(\mathcal{L}'+1)$st square root $d \in QR_n$ can thus be computed, and r' is then chosen uniformly at random among the 4 square roots in $\mathbf{Z}_n^*$ of d.

If r is uniformly distributed in $\mathbf{Z}_n^*$, then (since each element of QR_n has 4 square roots in $\mathbf{Z}_n^*$) the value $r^2 \bmod n$ is uniformly distributed in QR_n. The squaring map on QR_n permutes QR_n, so that g and d are also uniformly distributed in QR_n. It follows that r' is uniformly distributed in $\mathbf{Z}_n^*$. $\square$

Length at end. A collision-resistant variant of VSH where the message length is only needed at the end of the hashing process (rather than the beginning) is obtained by fixing the random input $r = 1$. This variant may be better suited for some applications than the versions presented in the previous section.

Efficiency. The function F_n is as efficient as VSH. For short fixed-length messages with $\ell \leq k$ (i.e., 1 block), the message length can be omitted, so that $F_n(m, r) = (r^2 \prod_{i=1}^{k} p_i^{m_i})^2 \bmod n$. Evaluation requires only about 4 compared to at least k modular multiplications required by the trapdoor functions in [19]. On the other hand, the trapdoor collision-finding algorithm for F_n is not very fast, requiring a square root modulo n per message block. This is not a 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. However, it reduces the efficiency of the reduction and thus requires slightly increased security parameters.

'Inversion' trapdoor property. It follows from the proof of Theorem 2 that F_n also satisfies the 'inversion' trapdoor property [19]. This is stronger than the trapdoor collision property, and can be used to upgrade a signature scheme's resistance against random message attacks to chosen message attacks: Given the trapdoor key, a random element $d \in QR_n$ in the range of F_n and an $m \in \mathcal{M}$, it is easy to find a randomiser $r \in \mathbf{Z}_n^*$ such that $F_n(m, r) = d$ and r is uniformly distributed in $\mathbf{Z}_n^*$ when d is uniformly distributed in QR_n.

Why Merkle-Damgård works. The function F_n applies a variant of the Merkle-Damgård transformation [14, 7] to hash arbitrary length messages using the VSH compression function $H : \mathbf{Z}_n^* \times \{0,1\}^k \to \mathbf{Z}_n^*$. The proof in [7] shows that a sufficient condition for the Merkle-Damgård function F_n to be collision-resistant is that the compression function H is collision-resistant, i.e. it is hard to find any $(x, m) \neq (x', m')$ with $H(x, m) = H(x', m')$. However, our compression function $H(x, m) = x^2 \prod_{i=1}^{k} p_i^{m_i} \bmod n$ is not strictly collision-resistant ($H(-x \bmod n, m) = H(x, m)$), and yet we proved that H is still sufficiently strong to make F_n collision-resistant. Therefore, one may ask whether we can strengthen the result in [7] to state explicitly the security properties of a compression function (which are weaker than full collision-resistance) that our compression function satisfies and that are still sufficient in general to make the Merkle-Damgård function F_n 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 $(x, m), (x', m') \in \mathbf{Z}_n^* \times \{0,1\}^k$ with $m \neq m'$ such that $H(x, m) = H(x', m')$.
(2) Preimage Resistance for a collision in first input: It is hard to find $(x, m) \neq (x', m') \in \mathbf{Z}_n^* \times \{0,1\}^k$ and $m^* \in \{0,1\}^k$ such that $H(y, m^*) = H(y', m^*)$, where $y = H(x, m)$, $y' = H(x', m')$ and $y \neq y'$.

Our VSH compression function satisfies these properties, assuming the NMSRVS Assumption.

Application. 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). The CS scheme makes use of an RSA-based randomised trapdoor

hash function to achieve security against adaptive message attacks. Using F_n instead cuts the signing and verification costs by about a double exponentiation each, while preserving the proven security. The modified CS scheme is as follows:
Key Generation: Choose two safe random $\approx S/2$-bit primes $\bar{p}, \bar{q}$ and two random $\approx S/2$ bit primes p, q with $p \equiv q \equiv 3 \bmod 4$ that result in S-bit moduli $\bar{n} = \bar{p}\bar{q}$ and $n = pq$, and choose $x, z \in QR_{\bar{n}}$ at random. Let $h : \{0,1\}^S \to \{0,1\}^\ell$ be a collision-resistant hash function for a security parameter ℓ chosen such that an ℓ-bit hash and S-bit RSA offer comparable security (typically $\ell = 160$ when $S = 1024$). The public key is $(x, z, n, \bar{n}, h)$ and the secret key is $(\bar{p}, \bar{q})$.
Signing: To sign $m \in \{0,1\}^*$, choose a random $(\ell+1)$-bit prime e and a random $r \in \mathbf{Z}_n^*$ and compute $y = (x \cdot z^{h(F_n(m,r))})^{1/e} \bmod \bar{n}$. The signature is (e, y, r).
Verifying: To verify message/signature pair $(m, (e, y, r))$, check that e is an odd $(\ell+1)$-bit integer and that $y^e z^{-h(F_n(m,r))} \equiv x \bmod \bar{n}$.

The cost of verification in the original CS scheme is about two double exponentiations with ℓ-bit exponents. The modified scheme requires approximately one such double exponentiation, so a saving in verification time of about 50% can be expected. The relative saving in signing time is smaller. However, the length of the public key is larger than in the original scheme by typically 25%.

Because VSH's output length S is typically much larger than ℓ, VSH cannot be used for the ℓ-bit collision-resistant hash function h above. To avoid the need for an ad-hoc ℓ-bit hash function, h may be dropped and e chosen as an $(S+1)$-bit prime, making the scheme much less efficient. The variant below eliminates the need for h and maintains almost the computational efficiency of the scheme above, but has a larger public key and requires some precomputation.
Key Generation: Let $\bar{p}, \bar{q}, p, q, \bar{n}, n$ be as above, let $s = \lceil \frac{S}{\ell} \rceil$ and choose $x, z_1, \ldots, z_s \in QR_{\bar{n}}$ at random. The public key is $(x, z_1, \ldots, z_s, n, \bar{n})$ and the secret key is $(\bar{p}, \bar{q})$.
Signing: To sign $m \in \{0,1\}^*$, choose a random $(\ell+1)$-bit prime e and a random $r \in \mathbf{Z}_n^*$, and compute $F_n(m,r)$. Interpret $F_n(m,r)$ as a value in $\{0,1\}^{s \cdot \ell}$ (possibly after padding) consisting of s consecutive ℓ-bit blocks $F_{n,1}(m,r), \ldots, F_{n,s}(m,r)$ and compute $y = (x \cdot \prod_{u=1}^{s} z_u^{F_{n,u}(m,r)})^{1/e} \bmod \bar{n}$. The signature is (e, y, r).
Verifying: To verify message/signature pair $(m, (e, y, r))$, check that e is an odd $(\ell+1)$-bit integer and that $y^e \prod_{u=1}^{s} z_u^{-F_{n,u}(m,r)} \equiv x \bmod \bar{n}$.

For typical parameter values such as $S = 1024$, $\ell = 171$, $s = 6$, the $2^s = 64$ subset products modulo $\bar{n}$ of the z_u's may be precomputed. Using multi-exponentiation, that would make the above scheme about as efficient as the previous variant. It can be proved (cf. Appendix A) that the above CS signature variant is secure assuming the strong-RSA and NMSRVS assumptions. Thus we have obtained an efficient signature scheme proven secure without ad-hoc assumptions. This is unlike the original CS scheme, which relied on a collision-resistance or universal one-wayness assumption regarding a 160-bit hash function—as far as we are aware, the only practical provably secure design for such a function is an inefficient discrete log based construction using an elliptic curve defined over a 160-bit order finite field. A disadvantage of our variant is that its public key is somewhat long: typically 9 kbits, which is about 3 times more than in the original CS scheme.

5 Efficiency of VSH in Practice

Let the cost of a multiplication modulo n be $O((\log n)^{1+\epsilon})$ operations, where $\epsilon = 1$ if ordinary multiplication is used, and where $\epsilon > 0$ can be made arbitrarily small if fast multiplication methods are used. Asymptotically the cost of the basic VSH algorithm is $O(\frac{(\log n)^{1+\epsilon}}{k}) = O((\log n)^\epsilon \log\log n)$ operations per message-bit. Given n's factorisation one can do better for long messages by reducing the k exponents of the p_i's modulo $\phi(n)$. Asymptotically, Fast VSH costs $O((\log n)^\epsilon)$ operations per message-bit. It is faster in practice too, cf. below.

If the length of the message is not known ahead of time (streaming data), the proper power of x_0 can be 'pasted on' at the end of the computation at the cost of modular 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.

The table below lists VSH runtimes obtained using a straightforward gmp-based implementation on a 1GHz Pentium III. The two security levels conservatively correspond to 1024-bit and 2048-bit RSA (based on the Computational NMSRVS Assumption, where an S-bit VSH modulus leads to a lower RSA security level S' depending on the number of small primes used). In the 2nd and 6th rows basic VSH is used with more small primes, in the 3rd and 7th rows extended with precomputed prime products and message processing $b = 8$ bits at a time. Fast VSH also processed $b = 8$ message-bits at a time. With $S' = 1024$ and $S = 1516$ (i.e., security at least as good as 1024-bit RSA, at the cost of a 1516-bit VSH modulus) Fast VSH is about 26 times slower than Wei Dai's SHA-1 benchmark [22]. Better throughput will be obtained under the more aggressive assumption that VSH with an S-bit modulus achieves S-bit RSA security.

S'	Method	# small primes	S	b	# products	Megabyte/second
1024	Basic VSH	152	1234	1	n/a	0.355
		1024	1318	1	n/a	0.419
				8	$128 * 256$	0.486
	Fast VSH	$2^{16} = 65536$	1516	8	n/a	1.135
2048	Basic VSH	272	2398	1	n/a	0.216
		1024	2486	1	n/a	0.270
				8	$128 * 256$	0.303
	Fast VSH	$2^{18} = 262144$	2874	8	n/a	0.705

Acknowledgements: We gratefully acknowledge inspiring discussions with Igor Shparlinski and Eran Tromer, 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*, page 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)*, page 161–185, 2000.
6. I. Damgård. Collision-free hash functions and public key signature schemes. In EUROCRYPT 87, volume 304 of *LNCS*, page 203–216, Berlin, 1987, Springer-Verlag.
7. I. Damgård. A design principle for hash functions. In CRYPTO 89, volume 435 of *LNCS*, page 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 J. on Comp.*, 17(2):281–308, 1988.
9. S. Hankerson, A. Menezes, S. Vanstone. *Guide to Elliptic Curve Cryptography*, New York, 2004, Springer-Verlag.
10. M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In EUROCRYPT 96, volume 1070 of *LNCS*, page 143–154, Berlin, 1996, Springer-Verlag.
11. A.K. Lenstra and E.R. Verheul. The XTR public key system. In CRYPTO 2000, volume 1880 of *LNCS*, page 1–19, Berlin, 2000, Springer-Verlag.
12. A.K. Lenstra and H.W. Lenstra Jr. *The Development of the Number Field Sieve*, Berlin, 1993, Springer-Verlag.
13. A.K. Lenstra, E. Tromer, A. Shamir, W. Kortsmit, B. Dodson, J. Hughes, and P. Leyland, Factoring estimates for a 1024-bit RSA modulus. In Chi Sung Laih, editor, ASIACRYPT 2003, volume 2894 of *LNCS*, page 55–74, Berlin, 2003, Springer-Verlag.
14. R. Merkle. One way hash functions and DES. In CRYPTO 89, volume 435 of *LNCS*, page 428–446, Berlin, 1989, Springer-Verlag.
15. D. Naccache and J. Stern A new public-key cryptosystem. In Walter Fumy, editor, EUROCRYPT 97, volume 1233 of *LNCS*, page 27–36, Berlin, 1997, Springer-Verlag.
16. D. Pointcheval. The composite discrete logarithm and secure authentication. In PKC 2000, volume 1751 of *LNCS*, page 113–128, Berlin, 2000, Springer-Verlag.
17. R.L. Rivest and R.D. Silverman. Are 'strong' primes needed for RSA. Report 2001/007, Cryptology ePrint Archive, 2001. `eprint.iacr.org/2001/007/`.
18. 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 *LNCS*, page 371–388, Berlin, 2004, Springer-Verlag.
19. A. Shamir and Y. Tauman. Improved online/offline signature schemes. In CRYPTO 2001, volume 2139 of *LNCS*, page 355–367, Berlin, 2001, Springer-Verlag.
20. 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*, page 86–100, Berlin, 2004, Springer-Verlag.
21. K. Rubin and A. Silverberg. Torus-based cryptography. In CRYPTO 2003, volume 2729 of *LNCS*, page 349–365, Berlin, 2003, Springer-Verlag.
22. Wei Dai. *Crypto++ 5.2.1 Benchmarks*. `www.eskimo.com/~weidai/benchmarks.html`.

A Security Proof for CS Signature Variant in Section 4

We give the security proof for the second CS variant signature scheme in Section 4 (the proof of the first CS variant scheme is essentially a special case with $s = 1$ and the collision-resistant function $F_n(\cdot,\cdot)$ replaced by the collision-resistant function $h(F_n(\cdot,\cdot))$).

Let A be an adaptive chosen message attacker, let $(m_i,(e_i,y_i,r_i))$ for $1 \leq i \leq t$ be the ith sign query/answer pair of A, and let $(m,(e,y,r))$ be the output forgery message/signature pair of A. As in [5], there are three possible types of attacks according to the output forgery. For each attacker type, we outline how to use it to contradict either the collision-resistance of F_n (and hence the NMSRVS assumption) or the strong-RSA assumption.

Type I. For some $1 \leq j \leq t$, $e = e_j$ and $F_n(m,r) = F_n(m_j,r_j)$. Given VSH modulus n, we construct a collision finder for F_n as follows. Generate $(x,z_1,\ldots,z_s,\bar{n})$ and $(\bar{p},\bar{q})$ as in key generation and run A on input $(x,z_1,\ldots,z_s,n,\bar{n})$. Using the known $(\bar{p},\bar{q})$, we efficiently answer (and store) A's sign queries. When A outputs its forgery (with $m \neq m_j$), we output the collision $(m,r),(m_j,r_j)$ for F_n.

Type II. For some $1 \leq j \leq t$, $e = e_j$ and $F_n(m,r) \neq F_n(m_j,r_j)$; in particular, with $F_{n,u}(m,r)$ for $1 \leq u \leq s$ denoting the uth consecutive ℓ-bit block of $F_n(m,r)$ (as in Section 4), let $F_{n,u^*}(m,r) \neq F_{n,u^*}(m_j,r_j)$ for some $u^* \in \{1,2,\ldots,s\}$. Given a strong-RSA instance $(\bar{n},\bar{z})$ with $\bar{z} \in_R \mathbf{Z}_{\bar{n}}^*$, we compute $e > 1$ and the eth root of $\bar{z} \bmod \bar{n}$ as follows. We randomly guess the values of $j \in \{1,\ldots,t\}$ and $u^* \in \{1,\ldots,s\}$ (our guess is right with non-negligible probability $1/(ts)$). Then we generate a random VSH modulus n (keeping its factors p,q for later use), t random $(\ell+1)$-bit primes $e_1,\ldots,e_t$, a total of s random elements $\{\bar{z}_u\}_{u=\{1,\ldots,s\}\setminus\{u^*\}}$ and v from $\mathbf{Z}_{\bar{n}}^*$, and a random $\bar{r} \in \mathbf{Z}_n^*$, and we compute $(x,z_1,\ldots,z_s)$, where $z_u = \bar{z}_u^{2\prod_i e_i} \bmod \bar{n}$ for $u \neq u^*$, $z_{u^*} = \bar{z}^{2\prod_{i\neq j} e_i} \bmod \bar{n}$, and $x = v^{2\prod_i e_i} \cdot (\prod_{u=1}^s z_u^{F_{n,u}(\bar{m},\bar{r})})^{-1} \bmod \bar{n}$, where $\bar{m}$ is an arbitrary message.
Given this set-up, we run A on input $(x,z_1,\ldots,z_s,n,\bar{n})$. For $i \neq j$, we answer the ith sign query of A with (e_i,y_i,r_i), where r_i is chosen uniformly from $\mathbf{Z}_n^*$ and $y_i = (x \cdot \prod_{u=1}^s z_u^{F_{n,u}(m_i,r_i)})^{1/e_i} \bmod \bar{n}$ is easy to compute since we know the e_ith roots of $x,z_1,\ldots,z_s \bmod \bar{n}$. For answering the jth sign query m_j, we first use the 'random trapdoor collision' algorithm for F_n (which is efficient since we know p,q) to compute $r_j \in \mathbf{Z}_n^*$ such that $F_n(m_j,r_j) = F_n(\bar{m},\bar{r})$ (note r_j is uniform since $\bar{r}$ is uniform), and we answer (e_j,y_j,r_j), where $y_j = (x \cdot \prod_{u=1}^s z_u^{F_{n,u}(\bar{m},\bar{r})})^{1/e_j} \bmod \bar{n} = v^{2\prod_{i\neq j} e_i} \bmod \bar{n}$ is easy to compute. Finally, if A's output forgery $(m,(e,y,r))$ is valid, then, defining $\Delta = F_{n,u^*}(m_j,r_j) - F_{n,u^*}(m,r) \neq 0$, we have

$$\frac{y_j}{y} \cdot \prod_{u\neq u^*} (z_u^{1/e_j})^{F_{n,u}(m,r)-F_{n,u}(m_j,r_j)} \equiv (z_{u^*}^{1/e_j})^{\Delta} \bmod \bar{n}.$$

Using the known e_jth roots of z_u for $u \neq u^*$, we efficiently compute the left hand side of the above congruence obtaining the result α. From the right hand side we see that $\alpha \equiv (\bar{z}^{1/e_j})^{2\Delta \prod_{i \neq j} e_i} \bmod \bar{n}$. Note we also know $\beta = (\bar{z}^{1/e_j})^{e_j} \bmod \bar{n} = \bar{z}$. Using the Euclidean algorithm we compute a and b such that $d = \gcd(2\Delta \prod_{i \neq j} e_i, e_j) = a \cdot 2\Delta \prod_{i \neq j} e_i + b \cdot e_j$ and hence we can efficiently compute $\gamma \equiv (\bar{z}^{1/e_j})^d \equiv \alpha^a \beta^b \bmod \bar{n}$. Since the e_i's are primes greater than 2^ℓ (and are distinct with overwhelming probability when t is small compared to $2^{\ell/2}$) while $0 < |\Delta| < 2^\ell$, it follows that $d = 1$ so the value γ is the $e = e_j$th root of $\bar{z} \bmod \bar{n}$ and hence a solution to our strong-RSA instance $(\bar{n}, \bar{z})$.

Type III. For all $1 \leq i \leq t$, $e \neq e_i$. Given strong-RSA instance $(\bar{n}, \bar{z})$ with $\bar{z} \in_R \mathbf{Z}^*_{\bar{n}}$, we compute $\delta > 1$ and the δth root of $\bar{z} \bmod \bar{n}$ as follows. We generate a VSH modulus n at random, t random $(\ell+1)$-bit primes $e_1, \ldots, e_t$, a total of s integers a and $a_2, a_3, \ldots, a_s$ chosen uniformly at random from $\mathbf{Z}_B$ with $B \geq \bar{n}^2$, and we compute $(x, z_1, \ldots, z_s)$, where $z_1 = \bar{z}^{2\prod_i e_i} \bmod \bar{n}$, $z_u = z_1^{a_u} \bmod \bar{n}$ for $u = 2, \ldots, s$ and $x = z_1^a \bmod \bar{n}$.
We then run A on input $(x, z_1, \ldots, z_s, n, \bar{n})$. Here we use the fact that z_1 is a generator of $QR_{\bar{n}}$ with overwhelming probability, because $\bar{n}$ is a product of safe primes, and that $z_1, z_2, \ldots, z_s, x$ are statistically indistinguishable from independent uniformly random elements in $QR_{\bar{n}}$ because $B/|QR_{\bar{n}}| \geq \bar{n}$. For $i = 1, \ldots, t$, we answer the ith sign query of A with (e_i, y_i, r_i), where r_i is chosen uniformly from $\mathbf{Z}^*_n$ and $y_i = (x \cdot \prod_{u=1}^{s} z_u^{F_{n,u}(m_i, r_i)})^{1/e_i} \bmod \bar{n}$ is easy to compute since we know the e_ith roots of $x, z_1, \ldots, z_s \bmod \bar{n}$. If A outputs a valid forgery $(m, (e, y, r))$ we have $y \equiv (\bar{z}^{1/e})^{2(c+a)\prod_i e_i} \bmod \bar{n}$, where $c = F_{n,1}(m, r) + \sum_{u \geq 2} F_{n,u}(m, r) \cdot a_u$. Similary to the Type II case, we can efficiently compute $\gamma \equiv \bar{z}^{1/(e/d)} \bmod \bar{n}$, where $d = \gcd(e, 2(c+a)\prod_i e_i)$. To ensure that $\delta = e/d > 1$, it suffices (since e is odd and $e \neq e_i$ for all i) that $(c + a) \not\equiv 0 \bmod \rho$ for some prime divisor ρ of e. But, dividing a by the order $ord(z_1)$ of z_1 in $\mathbf{Z}^*_{\bar{n}}$, we have $a = \lfloor \frac{a}{ord(z_1)} \rfloor \cdot ord(z_1) + (a \bmod ord(z_1))$. Since $B \geq \bar{n}^2$ and $ord(z_1) < \bar{n}$ the distribution of the quotient $\lfloor \frac{a}{ord(z_1)} \rfloor$ is statistically indistinguishable from uniform on $\mathbf{Z}_{\lfloor \frac{B}{ord(z_1)} \rfloor}$ and essentially independent of the attacker's view (which is only a function of the remainder $a \bmod ord(z_1)$). It follows (using also the fact that $\gcd(ord(z_1), \rho) = 1$ since $\bar{n}$ is safe and $\ell < S/2$) that $(c + a) \not\equiv 0 \bmod \rho$ occurs with non-negligible probability very close to $1 - 1/\rho > 1/2$, so that with non-negligible probability $\delta > 1$ and γ is a δth root of $\bar{z} \bmod \bar{n}$, solving the strong-RSA instance $(\bar{n}, \bar{z})$.

This completes the proof of security. □