Faster and Smoother – VSH Revisited

Juraj Šarinay*

EPFL IC LACAL, Station 14, CH-1015 Lausanne, Switzerland
juraj.sarinay@a3.epfl.ch

Abstract. We reconsider the provably collision resistant Very Smooth Hash and propose a small change in the design aiming to improve both performance and security. While the original proofs of security based on hardness of factoring or discrete logarithms are preserved, we can base the security on the k-sum problem studied by Wagner and more recently by Minder & Sinclair. The new approach allows to output shorter digests and brings the speed of Fast VSH closer to the range of "classical" hash functions. The modified VSH is likely to remain secure even if factoring and discrete logarithms are easy, while this would have a devastating effect on the original versions. This observation leads us to propose a variant that operates modulo a power of two to increase the speed even more. A function that offers an equivalent of 128-bit collision resistance runs at 68.5 MB/s on a 2.4 GHz Intel Core 2 CPU, more than a third of the speed of SHA-256.

Keywords: hash functions, generalized birthday problem, knapsacks, provable security

1 Introduction

A hash function is a mapping that on input a string of arbitrary length outputs a fixed-length digest. Such functions are among the most basic building blocks in cryptology. In the recent years there have been several attempts to design provably secure hash functions. Several of them follow the same general idea and hash by computing sums in finite Abelian groups.

Very Smooth Hash The function designed by Contini et al. [7] is a provably collision resistant hash function based on arithmetic in multiplicative groups modulo an integer. If the modulus is of a proper form, collision resistance of the function can be proved under an assumption heuristically linked to well-known problems, such as integer factoring and discrete logarithm. The function is relatively practical, but still considerably less convenient than the established hash functions. For example, in order to reach collision resistance that corresponds to 1024-bit RSA security, one needs to compute a digest that is 1516 bits long. The function was reported to be about 25 times slower than SHA-1. It is our goal to improve the multiplicative VSH in terms of both efficiency and security.

* Supported by a grant of the Swiss National Science Foundation, 200021-116712.

The generalized VSH-DL variants proposed in [16] are not targeted in this paper. While the functions output shorter digests, their performance is much worse than in case of the multiplicative versions.

Knapsack Based Functions From a high level point of view, VSH can be seen as a multiplicative knapsack. Compression function families based on 0-1 knapsacks in groups were introduced by Impagliazzo & Naor. Such functions are parametrized by k randomly selected elements of a finite group G denoted by $a_1, \ldots, a_k$. The function maps a k-bit string $b_1 \ldots b_k$ to $\sum_{i \in T} a_i$ where $T \subseteq \{1, \ldots k\}$ is the set of indices i such that $b_i = 1$. If $k > \lg |G|$,[1] the functions compress. Under certain assumptions it was shown that the families were universal and one way. Examples were given in groups $(\mathbb{Z}_n, +)$ and $(\mathbb{Z}_n^*, \times)$ and the security of some variants related to hardness of discrete logarithm and factoring [13, 14].

A similar additive function was proposed by Damgård in [9]. It is again a 0-1 knapsack, where a_i are positive integers and the group operation is addition. In a concrete example, k equaled 256 and the a_i were random integers with bit length under 120. The function was broken by Camion & Patarin using techniques that turn out to be central to our paper [6]. Joux et al. successfully applied lattice reduction to the class functions [15].

Following Ajtai's discovery of the one-way function related to worst case assumptions on lattices [1], Goldreich et al. showed that the function is actually collision resistant [12]. It is a 0-1 knapsack on k random elements of the additive group $\mathbb{Z}_q^n$ for prime $q = O(n^c)$ and $n \lg q < k \leq \frac{q}{2n^4}$.

Micciancio generalized the concept to certain rings in [20, 21]. Parametrize the function by k elements of a ring R. For $T \subseteq R$, define a compression function $T^k \to R$ as $\sum_{i=1}^{n} x_i \cdot a_i$. For appropriate choices of R, the family is preimage and collision resistant. Properties of the family are connected to worst-case assumption on special classes of lattices. For comprehensive treatment of the family, see also [24, 18]. A practical hash function SWIFFT following the principles was proposed in [17, 19] and its modification SWIFFTX was submitted as a SHA-3 candidate [2]. The generalized knapsacks considerably lower the key length and the number of operations needed to hash a bit. If $b = \lg |T|$, then kb bits can be hashed in k ring multiplications and $k-1$ ring additions and only k ring elements are needed to specify a particular function.

Incremental Hashing Bellare and Micciancio proposed a family of incremental hash functions in [5]. The functions are designed to map strings of arbitrary length to group elements. Fix an integer b, assume that the message is composed of kb bits. Denote the i-th b-bit chunk of the input by x_i. Define a function $f : \mathbb{N} \times \{0,1\}^b \to G$ and map x_i to $f(i, x_i)$.[2] Obtain k group elements, output

[1] lg stands for base 2 logarithm.

[2] The f is considered a random oracle in proofs and instantiated by a real-world hash function such as SHA.

their product in G. The authors define a *balance problem* in groups and use it as a security assumption in proofs. The groups considered include $(\mathbb{Z}_n, +)$, $(\mathbb{Z}_n^*, \times)$, $(\mathbb{Z}_q^n, +)$, $(\mathbb{Z}_2^n, +)$. The functions defined in these groups were named AdHASH, MuHASH, LtHASH and XHASH, respectively. In case of multiplicative groups, security follows from hardness of discrete logarithm. The authors observe that the construction appears more secure than what the proof suggests, there seems to be no attack even if discrete logarithms were easy to compute.

The (Extended) k-tree Algorithm All the above functions map an input string of length kb to k group elements that are then added in G. This can be seen as if we had k lists of 2^b group elements each and selected a single element from every list. This view is valid also for 0-1 knapsacks where $b = 1$, simply think of the lists as containing the identity element of G in addition to a_i. The problem of inverting such a function was considered by Wagner, it is known as the generalized birthday problem [28, 27]. In many groups it can be solved by the *tree algorithm* in time $O\left(k2^{\frac{S}{1+\lg k}}\right)$ where $S = \lg|G|$, provided all the lists contain enough elements. The idea is a generalization of the attack of Camion and Patarin on Damgård's function. Minder and Sinclair extended the algorithm to cases where the length of the lists is restricted [22].

Recent Provably Secure Hash Functions The fastest known attack on SWIFFT applies the extended tree algorithm while ignoring the ring structure and the relation to lattices [19].

Finiasz et al. proposed a code based function FSB that is essentially a knapsack in the group $(\mathbb{Z}_2^n, +)$ [3]. Joux et al. broke the function applying Wagner's algorithm [8]. Later variants of FSB [4, 10, 11] consider Wagner's attack in security analysis. This paper is in part inspired by the known applications of the tree algorithm to hash functions.

Main Contributions We propose two new variants of VSH and interpret them as knapsacks or k-sums. This brings the function closer to SWIFFT or FSB and allows us to adapt the cryptanalytic methods previously used on the two compression functions. We quantify security of the new VSH variants applying the known results on complexity of the extended k-tree algorithm. This suggests that our minor modification may improve security by many orders of magnitude compared to the original VSH variants. We point out evidence that the functions remain secure even if factoring and discrete logarithms are easy. Practicality of the new functions is demonstrated on an implementation.

2 Very Smooth Hash Algorithm

We recall two variants of VSH from [7] that are the starting points for our improvements. Denote the i-th prime number by p_i. In addition let $p_0 = -1$.

Fast VSH Let n be an S-bit integer, fix a small integer $b > 0$. Let k be the maximal integer such that $\prod_{i=1}^{k} p_{i2^b} < n$. Have an l-bit message m, denote the r-th b-bit chunk of m by $m[r]$ with $0 \leq m[r] < 2^b$. The Fast VSH algorithm proceeds as follows:

1. Let $x_0 = 1$.
2. Let $\mathcal{L} = \lceil \frac{l}{bk} \rceil$. Pad the message with zero bits up to an integral multiple of bk.
3. Append a bk-bit binary representation of l to the message, denote the new chunks $m[\mathcal{L}k + 1]$ to $m[(\mathcal{L} + 1)k]$.
4. For $j = 0, 1, \ldots, \mathcal{L}$ in succession compute

$$x_{j+1} = x_j^2 \times \prod_{i=1}^{k} p_{(i-1)2^b + m[jk+i]+1} \mod n \ .$$

5. Return $x_{\mathcal{L}+1}$.

Step 4 maps the S bits of x_j and kb fresh message bits to a new S-bit value x_{j+1}. The x_j can be viewed as a chaining variable, Fast VSH employs a variant of the Merkle-Damgård transform [9]. This special chaining mode is made use of in the original security proof that links collision resistance to the hardness of the following problem:

Definition 1 (VSSR: Very Smooth number nontrivial modular Square Root). *Let n be the product of two primes of approximately the same size and let $k' \leq (\log n)^c$. Given n, find x such that $x^2 \equiv \prod_{i=0}^{k'} p_i^{e_i} \mod n$ and at least one of the $e_0, e_1, \ldots, e_{k'}$ is odd.*

The (Fast) VSH security proof establishes that any collision in Fast VSH leads to a solution to VSSR with $k' = k2^b$.

Fast VSH-DL If the modulus is replaced by an S-bit prime number $p = 2q + 1$ for prime q, one obtains the Discrete Log variant of Fast VSH. This function maps $\mathcal{L}bk$ bits to S bits for fixed $\mathcal{L} \leq S - 2$. It uses the same iteration as Fast VSH, but because the length is fixed, no length needs to be appended. To hash $\mathcal{L}bk$ bits proceed as follows:

1. Let $x_0 = 1$.
2. For $j = 0, 1, \ldots, \mathcal{L}$ in succession compute

$$x_{j+1} = x_j^2 \times \prod_{i=1}^{k} p_{(i-1)2^b + m[jk+i]+1} \mod p \ .$$

3. Return $x_{\mathcal{L}+1}$.

The compression function can be extended to process inputs of arbitrary length by applying the Merkle-Damgård transform. The function is proved collision resistant assuming the following problem is hard:

Definition 2 (VSDL: Very Smooth number Discrete Log). *Let p, q be prime numbers with $p = 2q + 1$ and let $k' \leq (\log p)^c$. Given p, find integers $e_1, e_2, \ldots, e_{k'}$ such that $2^{e_1} \equiv \prod_{i=2}^{k'} p_i^{e_i} \mod p$ 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.*

2.1 Security

Security of Fast VSH is related to the hardness of factoring the modulus n. The designers base the assessment of collision resistance of Fast VSH on the following assumption:

Computational VSSR Assumption Solving VSSR is as hard as factoring a hard to factor S'-bit modulus, where S' is the least positive integer such that

$$L'[2^{S'}] \geq \frac{L'[n]}{k2^b} \ , \tag{1}$$

where the function

$$L'[n] = e^{1.923(\log n)^{1/3}(\log\log n)^{2/3}} \tag{2}$$

approximates the running time of Number Field Sieve factoring the integer n.

Heuristically, given $k2^b$ solutions to VSSR, one can use linear algebra to find $x \not\equiv \pm y \mod n$ such that $x^2 \equiv y^2 \mod n$ and obtain factorization of the modulus. If NFS is assumed to be the fastest method for factoring integers of the form of n, the VSSR problem is no easier than a fraction $\frac{1}{k2^b}$ of the cost of NFS. The above assumption can directly be used to derive a provable lower bound on collision resistance for a particular Fast VSH variant.

In contrast, no computational VSDL assumption was made in the proposal. Therefore no computational lower bound on collision resistance of Fast VSH-DL can immediately be derived.

Generating Collisions Collisions in Fast VSH are trivial to create if the factorization of the modulus n is known. This allows to compute $\varphi(n) = (p-1)(q-1)$. The hash function computes a modular multi-exponentiation $\prod_{i=1}^{k2^b} p_i^{e_i} \mod n$. The product does not change if an integral multiple of $\varphi(n)$ is added to any of the exponents. Creating messages that lead to appropriate differences in e_i is rather easy. Note that the collisions created in this way are quite long, measuring some Skb bits. Creating short collisions appears to be a much harder problem, even if p and q are known.

Similarly, the ability to compute discrete logarithms modulo p leads to an immediate straightforward way to create collisions in Fast VSH-DL.

3 A Variant Without Modular Squaring

We now proceed to modify Fast VSH and Fast VSH-DL such that the collision attacks from the previous section are no longer possible. The changes we make preserve the original security proofs.

Note that if $kb > S$, the operation

$$\prod_{i=1}^{k} p_{(i-1)2^b+m[jk+i]+1} \mod n$$

in Step 4 of Fast VSH compresses its input. It can therefore be extended to a hash function in the usual ways, such as the plain Merkle-Damgård mode. Our new Fast VSH variant will impose the condition $kb > S$ and build a compression function only.

Faster VSH Have an S-bit modulus n, let k, b be integers such that $kb > S$ and $k2^b < \log(n)^c$. Define a compression function H from kb bits to S bits that outputs

$$H(m) = \prod_{i=1}^{k} p_{(i-1)2^b+m[i]+1} \mod n \ .$$

The function computes a modular product of precisely k out of the primes $p_1, \ldots, p_{k2^b}$. If $n = pq$, we can prove collision resistance of the compression function based on the VSSR problem:

Theorem 1. *A collision in Faster VSH solves VSSR for $k' = k2^b$.*

Proof. Suppose there are two messages $m \neq m'$ such that $H(m) = H(m')$ and

$$H(m) = \prod_{i=1}^{k} s_i \mod n \tag{3}$$

$$H(m') = \prod_{i=1}^{k} t_i \mod n \ . \tag{4}$$

Denote the hash value $H(m)$ by x. From (3) and (4) it follows that

$$x^2 \equiv \prod_{i=1}^{k} t_i \prod_{i=1}^{k} s_i \mod n \ .$$

Because $m \neq m'$, at least one of the exponents on the right hand side equals one, the above expression solves VSSR. □

If n is replaced by the prime $p = 2q + 1$ for prime q, we are in the Fast VSH-DL setting. In a direct analogy of the original proof we can link the security of modified Fast VSH-DL to VSDL.

Theorem 2. *A collision in Faster VSH-DL solves VSDL for* $k' = k2^b$.

The modular squaring is no more crucial for the security proof. Our modified function is secure under the original assumptions, the proofs can be easily adapted to the new setting.

Note however, that our modification limits all the exponents in the prime products to one. The factoring attack on Fast VSH and the discrete log attack on Fast VSH-DL do not extend to the modified variants.

Performance The change proposed in this section slows the functions down. Only $bk - S$ fresh bits are processed per iteration due to the use of the ordinary Merkle-Damgård mode. In contrast, the original Fast VSH processed bk bits per iteration. The performance of the modified variant is approximately a fraction $1 - \frac{S}{bk}$ of the original. The greater the compression ratio of the new function, the less significant the slowdown.

The proposed modification to Fast VSH does not appear to be beneficial at all. The mere fact that the original collision-finding attacks do not work is not worth the performance loss. The security level implied by security proofs is the same after all.

As we will show in the next section, the modification allows a radical change in security assessment and in the end permits faster and/or more secure hashing.

4 The *k*-sum Problem and the Tree Algorithm

The following equivalent view of the two new VSH variants will be useful in security analysis. Split the primes into k lists $L_1, \ldots, L_k$, such that L_i contains the 2^b primes $p_{(i-1)2^b+1}, \ldots, p_{i2^b}$.

Let $f_i : \{0,1\}^b \to L_i$ be a function that interprets the bit string $m[i]$ as an integer and returns the element on position $m[i]$ from the list L_i. The new compression functions can be rewritten as follows:

$$H(m) = f_1(m[1]) * f_2(m[2]) * \cdots * f_k(m[k]) \tag{5}$$

where $m[i]$ is the i-th b-bit chunk of the input m. In the above description, $*$ stands for modular multiplication. For the rest of this section, let $*$ simply denote a group operation in a finite Abelian group G such that $\lg |G| \approx S$.

To measure the security of compression functions of the above type consider the following problem:

Definition 3 (The *k*-sum problem). *Given a group* G*, an element* $y \in G$ *and disjoint lists of group elements* $L_1, \ldots, L_k$ *find* $g_i \in L_i$ *such that*

$$g_1 * g_2 * \ldots * g_k = y \ .$$

Pre-image resistance of our two new compression functions is easily seen to be equivalent to an instance of the above problem. We show that collisions also naturally correspond to an instance of a k-sum problem.

Given the k lists L_i, form new lists L'_i containing all the elements gh^{-1} for $g, h \in L_i$. Size of L'_i is approximately 2^{2b}. A collision in H corresponds to a solution to the k-sum problem with the lists L'_i and target value 1. Note that a solution to the new k-sum problem leads in turn to a pair of colliding messages. A solution is necessarily of the form $g_1 h_1^{-1} * g_2 h_2^{-1} * \ldots * g_k h_k^{-1}$. If $m[i] \in \{0,1\}^b$ is the unique value such that $f_i(m[i]) = g_i$ and $m'[i] \in \{0,1\}^b$ is such that $f_i(m'[i]) = h_i$, then the concatenation of $m[i]$ collides with the concatenation of $m'[i]$. This leads to a collision if $m \neq m'$, or equivalently if the solution to the k-list problem does not select 1 in all of the lists.[3] In general, collision search for H is as hard as pre-image search for a function (over the same group) that compresses twice as much as H.

Known Lower Bounds For certain groups, hardness of the k-sum problem follows from more "usual" assumptions. Wagner shows that the problem in a cyclic group is no easier than discrete logarithms. Impagliazzo & Naor prove that a 0-1 knapsack in any group G is hard whenever there is a homomorphism onto G that is hard to invert [14]. Their theorem easily extends to k-sums.

The connection of k-sums to homomorphisms was also pointed out in [5]. The discrete logarithm is a special case. Squaring in the multiplicative group of quadratic residues modulo $n = pq$ is another example of an onto homomorphism. Solving the k-sum problem in the group is therefore at least as hard as factoring n.

Wagner's Tree Algorithm The tree algorithm by Wagner solves the k-sum problem in about $k2^{\frac{S}{1+\lg k}}$ group operations, provided all the lists contain at least $2^{\frac{S}{1+\lg k}}$ elements. To simplify the exposition, we deliberately neglect other operations the algorithm performs, such as sorting and comparisons. The algorithm needs a sequence of groups K_j normal in G for $j = 0, \ldots 1 + \lg k$, such that $K_0 \subseteq K_1 \subseteq K_2 \subseteq \ldots \subseteq K_{1+\lg k}$ and $|K_j| \approx |G|^{\frac{j}{1+\lg k}}$. In addition, the algorithm makes use of homomorphisms $\rho_j : G \to G/K_j$ with $K_j = \operatorname{Ker} \rho_j$.

Without loss of generality one can assume that the target value y is the identity of G. For a different y, simply multiply all elements in one of the lists by y^{-1} and solve for 1. The tree algorithm successively merges pairs of the lists. Only "useful" entries that fall in the subgroups K_j are kept. For details of the algorithm, the reader is referred to [27].

It is not necessary that the K_j form a sequence of normal subgroups of G. A "weaker" chain of subsets may be sufficient, as demonstrated in Wagner's paper for groups $(\mathbb{Z}_M, +)$ where a chain of intervals in used. We will further assume that the structure of G allows the tree algorithm to run with k lists for any k. This is a strong assumption, the group structure will often limit the applicability and/or exact performance of the algorithm. Our goal is to estimate

[3] We might remove the element 1 from all the lists and limit ourselves to colliding messages that differ in all b-bit blocks.

the cost of the tree algorithm from below. The assumption well fits the purpose and greatly simplifies further analysis.

Extended Tree Algorithm Wagner's analysis assumes the lists are long enough. If it is not the case, one can combine the short lists into (fewer) longer lists and invoke the ordinary tree algorithm once the lists are long enough. The extended tree algorithm by Minder and Sinclair[4] [22] builds lists of maximal length 2^u where

$$u = \frac{S - b2^p}{\lg k - p} \tag{6}$$

such that p is the least integer satisfying

$$S \leq (\lg k - p + 1)b2^p \ . \tag{7}$$

We will use the two above expressions to measure the security of Fast VSH variants. To simplify the exposition, we will measure the complexity of a particular instance of the k-sum problem exclusively by the maximum expected length of the lists produced. The workload will usually be slightly higher, but a lower bound on the cost is sufficient for our purposes.

What If Factoring and DL Are Easy? Interestingly, the k-sum problems over multiplicative groups appear to remain hard even if factoring and discrete logarithms are easy. The multiplicative group $\mathbb{Z}_n^*$ for $n = pq$ is isomorphic to $(\mathbb{Z}_{p-1}, +) \times (\mathbb{Z}_{q-1}, +)$ and the multiplicative group $\mathbb{Z}_p^*$ is isomorphic to $(\mathbb{Z}_{p-1}, +)$. Suppose we can factor and compute discrete logarithms efficiently. This allows us to compute the isomorphisms from the multiplicative to the additive group. While a group isomorphism transposes a k-sum problem instance to a setting where we may be able to compute faster, the lengths of the lists are preserved. Even in $(\mathbb{Z}_m, +)$ no faster approach than the (extended) tree algorithm is known.

For all the multiplicative functions we consider, the cost of factoring and/or discrete logarithms is negligible compared to the cost of the tree algorithm. We may therefore simply consider the steps to come for free in our analysis, even if a run of the tree algorithm is preceded by DL computations.

4.1 Security of Faster VSH

Table 1 captures the collision resistance level implied by the computational VSSR assumption ("factoring hardness") and the new security estimates based on the extended k-tree algorithm for two variants of Faster VSH. The parameters originate from [7]. The modulus is assumed to be a product of two large primes. The factoring hardness measure is the complexity of factoring a hard to factor integer of said bit length. Columns 5 and 6 are computed using (6) and (7).

[4] The algorithm was described for XOR as the group operation, but can be generalized to other groups.

Table 1. Security estimates for variants of Faster VSH

k	S	b	factoring coll	coll	pre
256	1516	8	1024 bits	$2^{334.7}$	$2^{502.0}$
1024	2874	8	2048 bits	$2^{472.4}$	$2^{616.7}$

Note that the two security measures are in somewhat incompatible units. The exponent in column 5 means "bit security", the number in column 4 is "RSA security".

Hardness of factoring a 1024-bit RSA modulus is considered equivalent to roughly "80-bit" security. If Faster VSH with $S = 1516$ is assessed as a k-sum problem, it provides collision resistance of at least 335 bits. For these parameters, the removal of squarings and the use of Merkle-Damgård mode slows down the original Fast VSH by approximately 74%. This is only a moderate slowdown given the great increase in security.

The slowdown is only approx. 35% for the other variant with $S = 2874$, while the gap between the hardness of factoring an 2048-bit modulus and 472-bit security is even greater.

Collision resistance of 335 or 472 bits is more than enough for many years to come. We can therefore aim for lower security levels and tweak the parameters, in particular the digest length, to gain performance. Precise parameters for practical instances of the functions as well as speed measurements are given in Section 6.

5 A Variant Without Modular Reduction

The lower bound on the complexity of the extended tree algorithm only depends on the number of lists k, the length of the lists 2^b and the output length S. While the structure of the particular group does affect the practicality of the algorithm to an extent, it almost certainly makes the job harder than what our estimates suggest.

Because we choose not to rely on the group structure, we propose to replace the modulus in Faster VSH by a power of two. No costly modular reductions are needed in the computation of the compression function, because reducing modulo a power of two can be done for free. This will lead to a considerable speed-up while maintaining the k-list security. Note that the prime $2 = p_1$ cannot be used in this setting, because it does not belong to the multiplicative group of integers modulo 2^S. The lists are to be filled with small primes starting with $p_2 = 3$. To hash bk input bits, compute the following product:

$$H(m) = \prod_{i=1}^{k} p_{(i-1)2^b+m[i]+2} \mod 2^S \ .$$

Any hash value will be an odd number, i.e. the least significant bit is always 1. Therefore only $S-1$ leftmost bits of the S-bit modular product should be output. Call the function *Smoother VSH*.[5]

Technically, the provable connection between the security of the function and the VSSR assumption (or a variant of VSDL) is preserved. This provides little confidence in security, because factoring the number 2^S is trivial and so are discrete logarithms in (the large cyclic subgroup of) $\mathbb{Z}_{2^S}^*$.

The security assessment based on the extended tree algorithm remains valid. Recall that it does not rely on the modulus, but only on the values k, b and S.

6 Experimental Results

We propose seven parameter sets with varying security, speed, and memory requirements. All the parameter choices were tailored to meet one of the three collision resistance levels 2^{128}, 2^{192} and 2^{256}. The length of the modulus is always an integral multiple of word size (64 bits in our case). The value b has a significant impact on performance. Small b results in many multiplications, but keeps the memory requirements down. With larger b one saves on multiplications, but needs much more memory. The ideal value is best determined empirically. The value $b = 8$ used in all our variants is the most convenient choice also from an implementation point of view.

The pre-image and collision resistance are measured as list lengths in the extended tree algorithm. For comparison, in the case of Faster VSH modulo a product of two primes, we include "factoring" security levels implied by the Computational VSSR assumption. The one bit difference in output length caused by the even modulus can be neglected, we consider the two variants equivalent for the purposes of the tree algorithm.

The modified functions are very simple and quite efficient in software. Our C implementation uses GNU MP 5.0.1 to perform arithmetic on large integers. The Faster VSH code uses Montgomery arithmetic to improve performance [23].

The speed was measured on a 2.40 GHz Intel Core 2 CPU running a 64-bit system. Table 2 displays speed measurements for the variants running in the Merkle-Damgård mode. The table also captures the total size of the lists of small prime numbers, represented by three bytes each. All the primes used in the proposed variants are at most 21 bits long, therefore the product of any three primes fits in 64 bits. Our implementation processes them in triples to save on multiplications.

The values in columns 3 and 4 were computed using (6) and (7), column 5 follows from (1) and (2).

Use of the modulus 2^S makes Smoother VSH approximately twice as fast as in the case of a random S-bit modulus M. For comparison, the speed of SHA-256 on the same platform is approximately 160 MB/s.[6] Our fastest variant with 128-bit collision resistance runs at more than third of that speed.

[5] Now we are running Very Smooth Hash modulo a smooth number.

[6] OpenSSL benchmark.

Table 2. Proposed parameters for Faster VSH and Smoother VSH

k	S	coll	pre	factoring coll	memory	Smoother VSH speed	Faster VSH speed
128	640	2^{128}	2^{192}	375	96 kB	45.5 MB/s	24.9 MB/s
256	768	2^{128}	2^{170}	452	192 kB	63.4 MB/s	33.2 MB/s
512	896	2^{128}	2^{160}	528	384 kB	68.5 MB/s	35.8 MB/s
192	960	2^{192}	2^{288}	603	144 kB	34.0 MB/s	18.1 MB/s
384	1152	2^{192}	2^{256}	727	288 kB	48.0 MB/s	24.3 MB/s
256	1280	2^{256}	2^{384}	839	192 kB	27.9 MB/s	14.6 MB/s
512	1536	2^{256}	2^{341}	1013	384 kB	39.8 MB/s	19.8 MB/s

Note that none of the seven variants of Faster VSH would be considered sufficiently secure based on the original computational VSSR assumption, possibly with the exception of the last one that is as secure as a 1013-bit RSA modulus.

7 Choice of the List Elements

The analysis in Section 4 assumes the k lists L_i contain random elements of the group G. This is not at all the case for our functions. We deliberately ignore this and expect the tree algorithm to behave as if the elements were random. Similar reasoning was used in cryptanalysis of SWIFFT and FSB, where the lists are far from random [19, 10, 4].

Small prime numbers used in VSH have been known to have two positive effects. Because a k-list function is defined by $k2^b$ group elements, if full S bits were used for every single entry, the memory requirements would soon become prohibitive. For the very same reason the function should not be implemented in its equivalent additive representation.

Another advantage of the use of small primes is speed. Multiplication of the S-bit modulus by a small (say 21-bit) prime is much easier an operation than a full $S \times S$ bit multiplication. The small prime numbers are absolutely necessary for VSH to remain practical.

7.1 Minimal Distance of Colliding Inputs

The list elements used in VSH are independent in a rather strong sense. Suppose there is a pair of colliding inputs $m \neq m'$ under Faster VSH such that

$$H(m) = \prod_{i=1}^{k} s_i \mod n$$

$$H(m') = \prod_{i=1}^{k} t_i \mod n$$

and the two inputs differ in precisely $l \leq k$ of the b-bit chunks that select a particular element from a list. Given $H(m) = H(m')$, there is a congruence of two products of primes:

$$\prod_{i=1}^{l} s'_i = \prod_{i=1}^{l} t'_i \mod n \ .$$

At least one of the products must exceed n. Therefore if d is the maximal bit length of any prime in our lists, then

$$l \geq \frac{S}{d} \ .$$

More precisely, if the first $k2^b$ primes are used to fill the lists, the largest prime is approximately $k2^b \ln(k2^b)$. Its bit length is then

$$d \approx b + \lg(bk \ln 2 + k \ln k) \ .$$

Any pair of colliding messages must differ in at least

$$l \gtrsim \frac{S}{b + \lg(bk \ln 2 + k \ln k)}$$

of the b-bit input chunks.

As an example, if $k = 128$ and $S = 640$, then $l \approx 35$. For the variant with $k = 512$ and $S = 1536$, the minimal number of different 8-bit chunks for colliding messages is $l \approx 75$.

Although we do not have a more direct link to collision resistance, we believe this property may support confidence in the functions. Small changes in input *never* lead to a collision. This property was in some form present in the original VSH as well, but (to our knowledge) it was never explicitly described. It is however considerably easier to quantify in our new setting.

A similar property was derived by Zémor & Tillich for hash functions based on (Cayley) graphs [25, 26].

8 On Provable Security

The main advantage of Fast VSH was the (heuristic) connection to well-known hard algorithmic problems. The modifications we propose in Sections 3 and 5 drop this feature, security is no longer supported by hardness of factoring or discrete logarithms.

Although our first modification to VSH preserved the proof, we ignored the security level implied by the VSSR assumption and measured it in a different way. This new viewpoint renders the proof and the VSSR assumption rather useless and provides some justification for dropping the VSSR and VSDL altogether with the smooth modulus in Section 5.

Note that in case of VSDL and the corresponding variant, there never has been a hardness estimate leading to meaningful bounds on collision resistance. All that was known is that discrete logs imply collisions. Our modification unifies the DL and factoring variants and allows the two to be proved secure based on a single new assumption.

The modified VSH shares several features with the functions SWIFFT and FSB. The two are provably secure compression functions also building on the hardness of the k-sum problem in commutative groups. The main feature shared is the method for measuring security. All the three function families build on their own hardness assumptions, originating from number theory (VSH), lattice theory (SWIFFT) or coding theory (FSB). If the k-sum problem were better understood, the various assumptions could be replaced by a single universal k-sum assumption. The functions would then all share a single security proof.

9 Conclusions

We proposed modifications to Very Smooth Hash that allow a radical change in security assessment. There was no need to apply the extended tree algorithm to the original multiplicative variants of VSH before, because the known attacks were considerably faster.

Our new variants of Faster VSH are designed to prevent factoring and DL attacks. The tree algorithm becomes a useful tool to measure security. We have shown that (independent of factoring or discrete logarithms) there is a deep combinatorial problem behind Faster VSH, the same problem that supports several other k-list hash functions.

We designed a new multiplicative compression function Smoother VSH that relies exclusively on the hardness of the k-sum problem. The function has still much longer output compared to "classical" hash functions of comparable security, but its speed has become reasonable. There is room for improvement in terms of implementation.

The extreme simplicity and clear structure of the new functions can be considered advantages. On the other hand, more insight in the complexity of multiplicative k-sum problems is desirable.

Acknowledgements The author would like to thank Arjen Lenstra, Ron Steinfeld, Scott Contini, Dimitar Jetchev and the anonymous reviewers for useful comments on the text.

References

1. Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108, 1996.
2. Yuriy Arbitman, Gil Dogon, Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFTX: A Proposal for the SHA-3 Standard. Submission to NIST, 2008.

3. D. Augot, M. Finiasz, and N. Sendrier. A Fast Provably Secure Cryptographic Hash Function. 2003.
4. Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. A family of fast syndrome based cryptographic hash functions. In Ed Dawson and Serge Vaudenay, editors, *Mycrypt*, volume 3715 of *Lecture Notes in Computer Science*, pages 64–83. Springer, 2005.
5. Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *EUROCRYPT*, pages 163–192, 1997.
6. Paul Camion and Jacques Patarin. The Knapsack Hash Function proposed at Crypto'89 can be broken. In *EUROCRYPT*, pages 39–53, 1991.
7. Scott Contini, Arjen K. Lenstra, and Ron Steinfeld. VSH, an efficient and provable collision-resistant hash function. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 165–182. Springer, 2006.
8. Jean-Sebastien Coron and Antoine Joux. Cryptanalysis of a provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2004/013, 2004.
9. Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989.
10. M. Finiasz, P. Gaborit, and N. Sendrier. Improved fast syndrome based cryptographic hash functions. *ECRYPT Hash Function Workshop 2007*, 2007.
11. Matthieu Finiasz. Syndrome based collision resistant hashing. In J. Buchmann and J. Ding, editors, *PQCrypto 2008*, volume 5299, pages 137–147. Springer, 2008.
12. Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Collision-free hashing from lattice problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(42), 1996.
13. Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. In *FOCS*, pages 236–241. IEEE, 1989.
14. Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *J. Cryptology*, 9(4):199–216, 1996.
15. Antoine Joux and Louis Granboulan. A practical attack against knapsack based hash functions (extended abstract). In *EUROCRYPT*, pages 58–66, 1994.
16. Arjen K. Lenstra, Dan Page, and Martijn Stam. Discrete logarithm variants of VSH. In Phong Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 229–242. Springer, 2006.
17. V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. Provably Secure FFT Hashing. *2nd NIST Cryptographic Hash Function Workshop*, 2006.
18. Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 144–155. Springer, 2006.
19. Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: A modest proposal for FFT hashing. In Kaisa Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2008.
20. Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *FOCS*, pages 356–365. IEEE Computer Society, 2002.
21. Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, 2007.
22. Lorenz Minder and Alistair Sinclair. The extended k-tree algorithm. In Claire Mathieu, editor, *SODA*, pages 586–595. SIAM, 2009.

23. Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–519, 1985.
24. Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 145–166. Springer, 2006.
25. Jean-Pierre Tillich and Gilles Zémor. Group-theoretic hash functions. In Gérard D. Cohen, Simon Litsyn, Antoine Lobstein, and Gilles Zémor, editors, *Algebraic Coding*, volume 781 of *Lecture Notes in Computer Science*, pages 90–110. Springer, 1993.
26. Jean-Pierre Tillich and Gilles Zémor. Hashing with SL_2. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 1994.
27. David Wagner. A generalized birthday problem. Long version, `http://www.eecs.berkeley.edu/~daw/papers/genbday-long.ps`.
28. David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.