Security of VSH in the Real World

Markku-Juhani O. Saarinen

Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK.
m.saarinen@rhul.ac.uk

Abstract. In Eurocrypt 2006, Contini, Lenstra, and Steinfeld proposed a new hash function primitive, VSH, *very smooth hash*. In this brief paper we offer commentary on the resistance of VSH against some standard cryptanalytic attacks, including preimage attacks and collision search for a truncated VSH. Although the authors of VSH claim only collision resistance, we show why one must be very careful when using VSH in cryptographic engineering, where additional security properties are often required.

1 Introduction

Many existing cryptographic hash functions were originally designed to be *message digests* for use in digital signature schemes. However, they are also often used as building blocks for other cryptographic primitives, such as pseudorandom number generators (PRNGs), message authentication codes, password security schemes, and for deriving keying material in cryptographic protocols such as SSL, TLS, and IPSec.

These applications may use truncated versions of the hashes with an implicit assumption that the security of such a variant against attacks is directly proportional to the amount of entropy (bits) used from the hash result. An example of this is the HMAC$-n$ construction in IPSec [1]. Some signature schemes also use truncated hashes. Hence we are driven to the following slightly nonstandard definition of security goals for a hash function usable in practice:

1. **Preimage resistance**. For essentially all pre-specified outputs X, it is difficult to find a message Y such that $H(Y) = X$. The difficulty should be $\approx 2^l$ when there are l pre-specified bits in X.
2. **2nd-preimage resistance**. Given a pre-specified message X, it is difficult to find another message Y so that $H(X) = H(Y)$. The difficulty should be $\approx 2^l$ when there are l pre-specified bits that match in the hashes.
3. **Collision resistance**. It should require $\approx 2^{l/2}$ effort to find any two messages X and Y that produce a collision $H(X) = H(Y)$ in l pre-specified bits in the hashes.

In addition to the above three usual goals, we state a fourth, more informal goal – **pseudorandomness**. In essence, we would like a PRNG, stream cipher, or other derived design that relies on a hash function to have at least $\approx 2^{l/2}$ security, as if it was secured with a "real" pseudorandom function.

Pseudorandomness implies that a hash has good statistical properties and resistance against a wide array of distinguishing attacks.

All of the mentioned desirable properties are difficult if not impossible to prove without nonstandard assumptions. We note that proofs based on assumptions are themselves assumptions, whether their origins are in the traditions of symmetric or asymmetric cryptanalysis. An assumption based on the sieving phase of the NFS factoring algorithm may seem like a "hard problem" to a researcher who has spent a lot of time tweaking the sieving phase of the NFS factoring algorithm. On the other hand, a researcher who has dedicated years of effort into symmetric cryptanalysis may feel that symmetric cryptography possesses equally well studied "hard problems", while also allowing more efficient overall implementation.

A "political" standardisation consideration is that (by definition) VSH has a backdoor in the secret factorisation of n. In the past it has been difficult to popularise cryptographic technologies that rely on trusted third parties.

In our opinion VSH is a simple, elegant design that is based on a plausible complexity-theoretic assumption (VSSR: Very Smooth number nontrivial modular Square Root). However, it should not be considered a general-purpose hash function as usually understood in security engineering.

On VSH Security Claims

"VSH is not a Hash Function."
– *Arjen K. Lenstra, Eurocrypt 2006* [1]

Collision resistance is the only property proven for VSH. In Section 3 of the VSH paper [2], short message inversion (equivalent to preimage resistance) is considered and one possible "solution" is provided. As will be shown in Section 2.1 of this paper, the solution is not adequate.

The authors therefore clearly expected VSH to exhibit some level of preimage and 2nd preimage resistance. These are standard requirements in the very definition of a "cryptographic hash function". The authors of VSH are very clear in that "VSH should not be used to model random oracles". Random oracle behaviour is not a standard hash function security requirement.

Some researchers tend to concentrate their efforts on showing that their hash functions provide collision resistance, while ignoring other security properties. However, it is well known that collision resistance does not imply preimage-resistance or other important hash function properties.

To illustrate this point, we present a classical counter-example. Consider an $l+1$-bit hash $H'(x)$ that has been constructed from an l - bit hash H as follows:

$$\text{If } |x| < l-1 \text{ then } H'(x) = x \,||\, 1 \,||\, 0\,0 \cdots 0.$$
$$\text{If } |x| \geq l-1 \text{ then } H'(x) = H(x) \,||\, 1.$$

[1] Quoted with permission. During the conference A.K. Lenstra used some of the results from this note in his presentation, with appropriate credit. This has led some people to mistakenly think that the results in this note were already contained in [2]. All cryptanalytic results presented in this paper are by the author; a draft was circulated with the authors of VSH before Eurocrypt 2006.

That is, if the message x is less than $l-1$ bits long, $H'(x)$ consists of the message itself, a single 1 bit and a padding of zero bits. If the message is $l-1$ bits or longer, the resulting hash consists of a (secure) hash of x, followed by a single 1 bit.

It is easy to show that H' is collision resistant if H is. It is also easy to see that H' is *not* preimage resistant for a large proportion of hash outputs, and that a slightly truncated version is *not* collision resistant.

2 The VSH Algorithm

We describe the VSH algorithm in its most basic form, essentially as it appears in the beginning section 3 of [2]. We note that the attacks can be extended to most of the variants given in the VSH paper, especially the Fast VSH variant in section 3.1 of [2]. [2]

Let $p_1 = 2, p_2 = 3, p_3 = 5, \ldots$ be the sequence of primes. Let n be a large RSA composite. Let k, the block length, be the largest integer such that $\prod_{i=1}^{k} p_i < n$. Let m be a be an l-bit message to be hashed, consisting of bits $m_1, m_2, \ldots, m_l$, and assume that $l < 2^k$. To compute the hash of m:

1. Let x_0 = 1.
2. Let $L = \lceil l/k \rceil$ the number of blocks. Let $m_i = 0$ for $l < i \leq Lk$ (padding).
3. Let $l = \sum_{i=1}^{k} l_i 2^{i-1}$ with $l_i \in \{0, 1\}$ be the binary representation of the message length l and define $m_{Lk+i} = l_i$ for $1 \leq i \leq k$.
4. For $j = 0, 1, \ldots, L$ in succession compute
$$x_{j+1} = x_j^2 \prod_{i=1}^{k} p_i^{m_{(jk+i)}} \mod n.$$
5. Return x_{L+1}.

Selecting a 1024-bit modulus n has been suggested in the original paper, indicating 131-bit block size k.

2.1 Preimage resistance

VSH is multiplicative: Let x, y, and z be three bit strings of equal length, where z consists only of zero bits and the strings satisfy $x \wedge y = z$. It is easy to see that
$$H(z)H(x \vee y) \equiv H(x)H(y) \pmod{n}.$$

This multiplicative property is similar, although simpler, than the one used by Coppersmith to attack (then) Annex D of X.509 [3].

[2] There were many changes to VSH before its final publication, most recently in early March 2006 when message length padding was changed to be performed *after* the message been hashed, rather than at the beginning. Such small changes have significant implications on the development of practical attacks. Remarkably, the "security proof" required no modification. The attacks discussed in this paper apply only to the published Eurocrypt version of VSH; other attacks may be devised on other variants.

As a result VSH succumbs to a classical time-memory trade-off attack that applies to multiplicative and additive hashes. The attack is similar in many aspects to Shanks' baby-step giant-step algorithm for discrete logarithms [5].

We set the secret message m as $(x \vee y)$ and rewrite the equation as

$$H(y) = H(x)^{-1} H(z) H(m) \pmod{n}.$$

To solve the l-bit preimage m of $H(m)$:

1. Tabulate $H(x \,||\, 00\cdots0)^{-1} H(z) H(m) \pmod{n}$ for $0 \leq x < 2^{l/2}$.
2. Do table lookups for $H(00\cdots0 \,||\, y)$ for $y = 0, 1, 2, \ldots$, looking for a match.

The algorithm terminates when $m = x \,||\, y$, in other words before $y < 2^{l/2}$. A preimage attack on VSH therefore has $\approx 2^{l/2}$ complexity rather than $\approx 2^l$ as expected.

Final squarings proposed in section 3 of [2] under subtitle "short message inversion" do not protect against this attack.

This type of attack is extremely serious if VSH is used to secure passwords, a typical application for hash functions. Note that the complexity of attack does not depend on the modulus size n, but on the entropy of the password strings.

Example 1. VSH is being used to secure a 4 character lower case alphabetic password M, stored with ASCII encoding. For demonstration purposes we choose k = 32 and a 169-bit modulus n:

$$\begin{aligned} n &= (2^{84} + 3)(2^{85} - 19) \\ &= 748288838313422294120286382894166426220969123119047. \end{aligned}$$

The hash of the secret is

$$H(m) = 1684412062515461733715906241346671669304986686 4325.$$

In this case $H(z) = 13$; the first iteration yields 1, and the second round 13, the sixth prime, as the length of the message is $2^5 = 32$ bits. We tabulate $H(x)^{-1} H(z) H(m) \pmod{n}$ for $26^2 = 676$ values $T[0 \ldots 675]$:

```
x: aa..   Binary: 01100001 01100001 00000000 00000000
T[0] = 91345572106882035279752100576530653

x: ab..   Binary: 01100001 01100010 00000000 00000000
T[1] = 116156501606261492576199026944080853
                        . . .
x: zz..   Binary: 01111010 01111010 00000000 00000000
T[675] = 384284712674090018973838770853950813384926485216514
```

In the second phase we run through the values of $H(y)$:

```
H(..aa) = 390484467755621620 9933
H(..ab) = 339609581917494930 8197
...
```

A match is found after 83 steps at $H(\texttt{..df}) = 302056604569995827811625594 93$, which matches with $T[18] = H(\texttt{as..})^{-1}H(z)H(m) \pmod{n}$. Hence the secret password M is "asdf".

Note that it is not necessary to store the entire value to the table $T[i]$; appropriate number of least significant bits usually suffices. When the table is indexed by, say, $T[i] \bmod 2^{32}$, search becomes an $O(1)$ operation.

This example illustrates that password cracking time is effectively "square-rooted" by this attack; l-character passwords offer a level of security expected from $l/2$-character passwords.

2.2 One-wayness (of the "Cubing" Variant)

In section 3.4 of the VSH specification, a variant that uses cubing instead of squaring in its compression function is proposed. Using the Jacobi symbol, the compression function

$$x_{j+1} = x_j^3 \prod_{i=1}^{k} p_i^{m_i} \bmod n,$$

becomes

$$\left(\frac{x_{j+1}}{n}\right) = \left(\frac{x_j}{n}\right) \prod_{i=1}^{k} \left(\frac{p_i}{n}\right)^{m_i}.$$

We define a "binary" version of the Jacobi symbol:

$$j(c, n) = \frac{1}{2}\left(1 - \left(\frac{c}{n}\right)\right).$$

We now have a linear equation giving the parity of some message bits:

$$j(x_{j+1}, n) = j(x_j, n) + \sum_{i=1}^{k} j(p_i, n) m_i \pmod{2}.$$

Note that the Jacobi symbol can be very efficiently computed and that $j(p_i, n)$ is essentially randomly 0 or 1 for each randomly generated composite n. If the same message has been hashed with k different moduli n, a system of k linear equations can be obtained, leading to disclosure of bits by solving the system of equations.

The same attack applies to the standard squaring version as well, but it only leaks information about the message length. This was not the case for VSH versions 3.57 and before (ePrint revisions of VSH published before March 2006), where information about the contents of the last message block could be obtained.

One-wayness is implied by the standard hash security requirement of preimage resistance. If one obtains some information about some of the preimage bits easily, one can find the rest faster in an exhaustive search, as the search space is smaller.

Example 2. Assume that a 64-bit password has been hashed with VSH. For demonstration purposes we define the modulus n to be equivalent to the RSA-1024 factoring challenge number $n = 1350..\text{(300 digits)}..7563$ [4].

The Jacobi symbols for the first small primes modulo n are:

$$\left(\frac{2}{n}\right) = -1 \; \left(\frac{3}{n}\right) = -1 \; \left(\frac{5}{n}\right) = -1 \; \left(\frac{7}{n}\right) = 1 \; \left(\frac{11}{n}\right) = 1 \; \left(\frac{13}{n}\right) = -1 \; \cdots$$

Since the length padding (last round) will simply consist of cubing the product of primes and multiplying that with length indicator $p_6 = 13$, we may write

$$\left(\frac{H(m)}{n}\right) = \left(\frac{13}{n}\right) \prod_{i=1}^{64} \left(\frac{p_i}{n}\right)^{m_i}.$$

Using the binary $j(c, n)$ function and knowledge of n, this can be further simplified into the following parity equation:

$$\begin{aligned} j(H(m), n) \equiv\ & 1 + m_1 + m_2 + m_3 + m_6 + m_7 + m_{10} + m_{13} + m_{14} + m_{15} + \\ & m_{16} + m_{17} + m_{22} + m_{24} + m_{25} + m_{26} + m_{27} + m_{28} + m_{29} + \\ & m_{31} + m_{33} + m_{36} + m_{39} + m_{40} + m_{43} + m_{44} + m_{46} + m_{49} + \\ & m_{51} + m_{52} + m_{57} + m_{59} + m_{61} + m_{64} \pmod 2. \end{aligned}$$

We can therefore speed up dictionary search against the password by a factor close to two as half of the password candidates can be rejected with simple bit shift, AND and XOR operations, rather than with computationally expensive modular arithmetic required to compute the full hash.

Note that if the same secret has been hashed with multiple different moduli n, the speedup grows almost exponentially; two distinct moduli yield a speedup factor close to 4 etc.

2.3 Collision Search for Truncated VSH Variants

VSH produces a very long hash (typically 1024 bits). There are no indications that a truncated VSH hash offers security that is commensurate to the hash length. This appears to rule out the applicability of VSH in digital signature schemes which produce signatures shorter than the VSH hash result, such as Elliptic Curve signature schemes.

To illustrate this point, we will describe give an attack on one truncated variant of VSH.

Partial Collision Attacks. We will first discuss a generic technique for turning a partial collision attack into a full collision attack.

Assume that there is a fast $O(1)$ mapping f that causes the hash result of an l-bit hash H to be in some smaller subset of possible outputs: $H(f(x)) \in S$, where $|S| < 2^l$. Typically f would be chosen in such a way that certain hash result bits are forced to have the same constant value. In other words, f forces partial collisions. Note that f itself should not produce too many collisions, i.e. $x_1 \neq x_2$ usually means that $f(x_1) \neq f(x_2)$.

If such an f can be found, and it is fast, the complexity of finding full collisions becomes $\approx \sqrt{|S|}$. Note that f does not need to be able to force the hash to S on each iteration, it is sufficient that it works with reasonable probability. The iteration in low-memory parallel collision search algorithm becomes $s_{i+1} = H(f(s_i))$, and generic parallel collision search algorithms such as those described in [6] can be used.

Attack on VSH Truncated to Least Significant 128 bits. We will instantiate this attack on a VSH variant that only uses the least-significant 128 bits of the hash function result. For basic VSH (1024-bit n, k=131) the result of hashing a 128-bit message $m_1|m_2|\cdots|m_{128}$ can be simplified to:

$$x = \Big(19\big(\prod_{i=1}^{128} p_i^{m_i}\big)^2 \mod n\Big) \mod 2^{128}.$$

The constant $19 = p_8$ is caused by the length padding in the second (and final) round.

It is easy to see that modular reduction by n occurs in this case with less than 50% probability if m is random (or randomised) and its Hamming weight behaves accordingly. This is due to the fact that if only half of the bits in the message are ones, the product of corresponding small primes will be roughly the same bit size as $\sqrt{n}$. The square of this will still be less than n with a significant probability and hence there is no modular reduction by n. Hamming weight of a random bit string is binomially distributed. In practice the modular reduction happens in this case with roughly $P \approx 0.35$ probability. We get the following approximation that is valid with significant probability:

$$x = 19\big(\prod_{i=1}^{128} p_i^{m_i}\big)^2 \mod 2^{128}.$$

Note that the iteration is independent of the RSA modulus n if there is no reduction.

Precomputation phase: For each of the 2^{41} bit strings r of length 41 we compute and store r into a lookup table, indexed by the product

$$\big(\prod_{i=2}^{42} p_i^{r_{i-1}}\big)^{-1} \mod 2^{42}.$$

We will choose the f mapping as follows: Select message bits $m_{43}, m_{44}, \ldots, m_{128}$ from corresponding bits of s_i. Compute the partial product $\prod_{j=43}^{128} p_j^{m_j} \mod 2^{42}$ and use that to select message bits $m_2, m_3, \ldots, m_{42}$ using the lookup table (m_1 is always set to zero).

This will often ($P \approx 0.5$) force the least significant 42 bits to a certain constant value, 19, on each iteration. Note that if the table lookup fails, we may select $m_2, m_3, \ldots, m_{42}$ to be some arbitrary deterministic value; one that satisfies $s_i \equiv 19 \pmod{2^l}$ for some $l < 42$ would be a good choice.

Hence we have can cause the iteration to run in a significantly smaller subset with essentially $O(1)$ effort (constant-factor increase), and collisions can be found significantly faster.

Example 3. We will start with $s_1 = 2^{42} + 19$, and try to produce a sequence satisfying $s_i \equiv 19 \pmod{2^{42}}$ for a significant portion of i.

The partial product $\prod_{i=43}^{128} p_i^{m_i} \mod 2^{42}$ yields $p_{43} = 191$ for s_1. We will then perform a lookup in the precomputed table; it turns out that selecting message bits m_1 through m_{42} as

```
01110010 01010101 00000000 11100001 11110111 00
```

will force the product the desired subset, as the product of primes corresponding to those message bits is

$$3 \cdot 5 \cdot 7 \cdot 17 \cdot 29 \cdot 37 \cdot 43 \cdot 53 \cdot 97 \cdot 101 \cdot 103 \cdot 131 \cdot 137 \cdot 139 \cdot 149 \cdot 151 \cdot 163 \cdot 167 \cdot 173$$
$$= 1164213571911795168635778009100095,$$

and this multiplied by the partial product satisfies

$$191 \cdot 1164213571911795168635778009100095 \equiv 1 \pmod{2^{42}}.$$

Clearly squaring a number that is congruent to $1 \bmod 2^{42}$ maintains that property. The final multiplication by 19 results in that that the second element of the sequence satisfies the desired property $s_2 \equiv 19 \pmod{2^{42}}$. We have

$$\begin{aligned} s_2 &= 19\,(191 \cdot 1164213571911795168635778009100095)^2 \mod 2^{128} \\ &= 79424F79408D6B27F52A500000000013_{16} \end{aligned}$$

With this sequence we only need to rely on a birthday collision in the upper $128 - 42 = 86$ bits of the sequence. Roughly 2^{43} iterations are required with algorithms of [6] to achieve this.

Note that with some probability this algorithm will yield false collisions due to the fact that the inverse of the partial product is not always found in the lookup table. Modular reduction by n may also cause false collisions. This only results in a constant factor increase to the complexity of the algorithm, however; we only need to restart with different starting points until a proper collision is found.

Overall complexity. In essence, the complexity of this attack against VSH truncated to l bits is:

- Pre-computing the table offline: $\approx 2^{\frac{l}{3}}$ time and space.
- Finding collisions: $\approx 2^{\frac{l}{3}}$ iterations.
- Total cost: roughly $\approx 2^{\frac{l}{3}}$, rather than $\approx 2^{\frac{l}{2}}$ as expected from a hash function with good pseudorandomness properties.

We acknowledge that this represents just *one* way of truncating VSH – using, say, the most significant bits of the result would be an even worse option. Many other truncated variants can be attacked using a different f function.

2.4 Other features of VSH

The authors of VSH do not explicitly note this, but the hash function result can be updated after small changes without computing the entire hash again. A "bit flip" in a message will always cause a predictable change in the message result (it becoming multiplied $\bmod n$ by certain power of a small prime or its inverse). This is due to the highly algebraic nature of the hash.

We note such a property may be useful in some applications where rapid update of the hash is required, but it is undesirable in many more as it can facilitate adaptive attacks against some cryptographic protocols. Similar multiplicative property was sufficient for the X.509 Annex D hash function to be considered broken [3].

3 Acknowledgments

The author would like to thank Arjen K. Lenstra and other authors of VSH for encouragement. The paper wouldn't exist if Kenny Paterson wouldn't have pointed out that publication of relatively simple results is important for "real world" security engineers. Keith Martin, Daniel J. Bernstein and anonymous program committee members helped to make the paper significantly easier to read.

References

1. M. BELLARE, R. CANETTI AND H. KRAWCZYK. *HMAC: Keyed-Hashing for Message Authentication. IETF RFC 2104, 1997.*
2. S. CONTINI, A.K. LENSTRA AND R. STEINFELD. *VSH, an efficient and provable collision resistant hash function. Advances in Cryptology – EUROCRYPT 2006, LNCS 4004, Springer-Verlag, 2006.*
3. D. COPPERSMITH. *Analysis of ISO/CCITT Document X.509 Annex D. IBM Research Division, Yorktown Heights, N.Y., 11 June 1989.*
4. RSA LABORATORIES. *RSA-1024 Factoring Challenge Number. Available from:* `http://www.rsasecurity.com/rsalabs/node.asp?id=2093`
5. D. SHANKS. *Class number, a theory of factorization and genera. Proc. Symp. Pure Math. pp. 415 – 550. AMS, Providence, R.I., 1979.*
6. P. VAN OORSCHOT AND M. WIENER. *Parallel collision search with cryptanalytic applications. Journal of Cryptology, 12 (1999), pp. 1 – 28, 1999.*