VHASH Security

Wei Dai[1] and Ted Krovetz[2]

[1] Bitvise Limited

[2] Department of Computer Science
California State University, Sacramento CA 95819 USA

August 28, 2007

Abstract. VHASH is an almost-delta-universal hash family, designed for exceptional performance on computers that multiply 64-bit quantities efficiently. Changes to the algorithm detailed in this note improve both security and performance over the original 2006 version. Speed is improved through a newly analyzed hash construction which allows the use of lower-degree polynomials. Claimed security is higher due primarily to improved analysis and a change in prime modulus. The result is a hash family capable of hashing cache-resident one kilobyte messages on the Intel Core 2 architecture at a rate of about one-half processor cycle per byte of message with a collision probability of less than $1/2^{61}$.

Keywords: Universal hashing, message authentication, Wegman-Carter, VHASH, VMAC.

VMAC is a software-oriented Wegman-Carter message authentication algorithm (MAC) introduced in 2006 [3, 6, 8]. At VMAC's heart is the VHASH hash family, which was introduced at the same time. In 2007, VMAC and VHASH were codified into a formal specification, and during this process VHASH was improved with speed and security optimizations, making the finally specified algorithm different from the original in small but significant ways. This note points out where the newer algorithm differs from the older, and updates the appropriate proofs.

Preliminaries. When s is a string, $|s|$ is its length. The empty string is represented λ. When A is a finite set, $|A|$ is the number of elements in the set. The set $\{0, 1, \ldots, p-1\}$ is written as $\mathbb{Z}_p$. All random choices are made according to the uniform distribution.

Universal hash families. A hash family H is a finite set of functions with each $h \in H$ having common domain A and codomain B. The following define hash properties relevant to this note. The probabilities are over the choice of $h \in H$.

1. H is *ε-almost-universal* (ε-AU) if $\Pr[h(m) = h(m')] \le \varepsilon$ for every $m \ne m' \in A$.
2. H is *ε-almost-delta-universal* (ε-ADU) if $\Pr[h(m) + x = h(m')] \le \varepsilon$ for every $m \ne m' \in A$ and $x \in B$. (B must be a group with operator $+$.)

Carter and Wegman introduced universal and strongly universal hashing and several applications [3, 8]. Stinson defined almost-universal, and Krawczyk and Stinson later each introduced almost-delta-universal [5, 7]. These definitions are relaxations of strongly universal and allow high speed implementations.

1 Revised VHASH Algorithm

As we developed optimized code for the original VHASH, a few changes were made to its specification. Initially, the changes focussed on speed, especially on short messages. These optimizations reduced security, and so subsequent modifications were introduced for both the analysis and the VHASH definition to restore the final collision probability to better than the original. Figure 1 gives the revised VHASH algorithm. The most significant changes to the VHASH algorithm are:

1. The value p (Line 7) is now the result of a degree n polynomial rather than degree $n+1$. This eliminates one 128-bit multiplication and modular reduction for each message being hashed, and means very short messages (those shorter than b bits) will require no multiplications or reductions during the polynomial stage of hashing. Also, k is chosen from a slightly more restricted set, allowing a simpler implementation.

$\mathsf{VHASH}[b](M, K, k, k_1, k_2)$

Inputs:

M, a string of any length

K, a string of length b bits, where $b = 128i$ for some integer $i > 1$

k, an element of $\{w2^{96} + x2^{64} + y2^{32} + z \mid w, x, y, z \in \mathbb{Z}_{2^{29}}\}$

k_1, k_2, integers in the range $0 \ldots 2^{64} - 258$, inclusive

Output:

h, an integer in the range $0 \ldots 2^{64} - 258$, inclusive

Algorithm:

1. $n = \max(\lceil |M|/b \rceil, 1)$
2. Let $M_1, M_2, \ldots, M_n$ be strings so that $M_1||M_2|| \cdots ||M_n = M$ and $|M_i| = b$ for $1 \le i < n$.
3. $\ell_n = |M_n|$
4. Let $M_n = M_n||0^j$ where $j \ge 0$ is the smallest integer so that $|M_n| + j \ \mathbf{mod}\ 128 = 0$
5. Byte-reverse each 64-bit word in M_i for each $1 \le i \le n$
6. $a_i = \mathsf{NH}[b/64, 64](K, M_i) \ \mathbf{mod}\ 2^{126}$ for each $1 \le i \le n$
7. $p = k^n + a_1 k^{n-1} + a_2 k^{n-2} + \cdots + a_n k^0 + \ell_n 2^{64} \ \mathbf{mod}\ (2^{127} - 1)$
8. $p_1 = p \ \mathbf{div}\ (2^{64} - 2^{32})$
9. $p_2 = p \ \mathbf{mod}\ (2^{64} - 2^{32})$
10. $h = (p_1 + k_1) \times (p_2 + k_2) \ \mathbf{mod}\ (2^{64} - 257)$

Fig. 1. *The hash family VHASH is (2^{-61})-ADU for all messages upto 2^{62} bits, when K, k, k_1, k_2 are chosen randomly and $b = 1024$.*

2. The prime modulus used in the final hash stage (Line 10) is increased from $2^{61} - 1$ to $2^{64} - 257$. Although slightly less convenient for divisionless modular reduction, the larger prime increases security. Also, the hash computation changes from $p_1 k_1 + p_2 k_2$ in the original to $(p_1 + k_1) \times (p_2 + k_2)$, substituting one addition for one multiplication.
3. No longer is VHASH-128 defined separately from VHASH-64. A single application of VHASH gives VHASH-64, while two applications of VHASH (with different keys) gives VHASH-128. Although this change introduces a third stage of hashing (Lines 7–10) to VHASH-128 that was not in the original, the increased work is offset by the polynomial degree reduction of Item 1.

We now restate the main security theorem, reflecting these changes and improvements in analysis. For any b that is a positive multiple of 128, Figure 1 specifies the hash family $\mathsf{VHASH}[b]$ where choosing a random function h from the family is achieved by choosing K, k, k_1 and k_2 uniformly at random from their domains and letting $h(\cdot) = \mathsf{VHASH}[b](\cdot, K, k, k_1, k_2)$.

Theorem 1. *Let b be any positive multiple of 128, ℓ be any positive multiple of b and $p = 2^{64} - 257$, then $\mathsf{VHASH}[b]$ is ε-ADU (for addition modulo p) over all binary strings up to length ℓ bits where $\varepsilon = 1/2^{62} + 1/(2^{64} - 257) + (\ell/b)2^{-115}$.*

A corollary changing the co-domain of the hash family to the more convenient $\mathbb{Z}_{2^{64}}$ follows by combining this theorem with Lemma 4.

Corollary 2. *Let b be any positive multiple of 128 and ℓ be any positive multiple of b, then $\mathsf{VHASH}[b]$ is ε-ADU (for addition modulo 2^{64}) over all binary strings up to length ℓ bits where $\varepsilon = 1/2^{61} + 2/(2^{64} - 257) + (\ell/b)2^{-114}$.*

The following section gives several hashing lemmas and shows how they fit into the proof of the theorem.

1.1 Basic VHASH Components

VHASH, in a three stage process, employs separate hash functions to do its work. The first stage of hashing uses NH to compress the message being hashed by a fixed ratio, reducing time spent in later, slower stages.

A polynomial-evaluation hash then produces a fixed length output in the second stage. And finally, the third stage uses a strong hash to shorten and strengthen the VHASH product. We review these basic hash functions here.

NH. The hash family NH was designed as a parameterized hash function [2]. Given positive integer parameters n and w and a key K of length nw bits, then NH can hash any string M that is a multiple of $2w$ bits in length but not longer than nw bits. First M and K are broken into w-bit blocks $M_1, M_2, \ldots, M_\ell$ and $K_1, K_2, \ldots K_n$ where $\ell = |M|/w$. Then, each block is interpreted as a w-bit unsigned integer $m_1, m_2, \ldots, m_\ell$ and $k_1, k_2, \ldots k_n$. Finally, the hash result is computed as

$$\mathrm{NH}[n, w](K, M) = \sum_{i=1}^{\ell/2} ((m_{2i-1} + k_{2i-1} \bmod 2^w) \times (m_{2i} + k_{2i} \bmod 2^w)) \bmod 2^{2w}\,.$$

NH is a hash family, and choosing a random function h from the hash family is done by choosing a random nw-bit key K and letting $h(\cdot) = \mathrm{NH}[n, w](K, \cdot)$. NH is known to be (2^{-w})-ADU (for addition modulo 2^{2w}) over messages of the same length (ie, M and M' are distinct, but $|M| = |M'|$), and small modifications to the original proof show that NH is (2^{-w})-ADU over messages that are any multiple of $2w$ bits in length (but still no longer than nw bits). In the context of VHASH, $w = 64$ and $nw = b = 1024$, making $\mathrm{NH}[16, 64]$ a (2^{-64})-ADU hash family. In VHASH, however, we always follow applications of NH with a modular reduction by 2^{126}. Let us consider $\mathrm{NH}[n, w, r] = \mathrm{NH}[n, w] \bmod r$ to be a hash family (again, functions are selected from the family via an nw bit string K). Corollary 5 tells us immediately that $\mathrm{NH}[16, 64, 2^{126}]$ is (2^{-62})-ADU.

We can extend NH to messages longer than nw through repeated use of NH. Let M be a message with length a multiple of $2w$ and let K be an nw-bit string. We hash M by breaking it into $t = \lceil |M|/nw \rceil$ chunks $M_1, M_2, \ldots, M_t$ with $|M_1| = |M_2| = \cdots = |M_{t-1}| = nw$, and computing (in the case of $\mathrm{NH}[n, w, r]$)

$$\mathrm{NH}^*[n, w, r](K, M) = \big(\mathrm{NH}[n, w, r](K, M_1), \mathrm{NH}[n, w, r](K, M_2), \ldots, \mathrm{NH}[n, w, r](K, M_t)\big)\,.$$

NH^* is a hash family, and choosing a random function from the hash family is done by choosing a random nw-bit key K. If NH is an ε-AU hash function, then so is NH^*.

Polynomial. Another simple and efficient hash method results from polynomial evaluation. Given fixed prime p, message $\mathrm{m} = (m_1, \ldots, m_\ell)$ and key $0 \le k < p$, the hash of m is computed as

$$\mathrm{Poly}(k, \mathrm{m}) = m_1 k^\ell + m_2 k^{\ell-1} + \cdots + m_\ell k^1 \bmod p\,.$$

Two different messages m, m' of the same length ℓ differ by constant d when hashed by this function if

$$\mathrm{Poly}(k, \mathrm{m}) - \mathrm{Poly}(k, \mathrm{m}') = (m_1 - m'_1)k^\ell + (m_2 - m'_2)k^{\ell-1} + \cdots + (m_\ell - m'_\ell)k^1 \bmod p = d\,.$$

Because $\mathrm{m} \neq \mathrm{m}'$, at least one of the coefficients in this polynomial is non-zero. This being a polynomial of degree at most ℓ, at most ℓ values for k cause $\mathrm{Poly}(k, \mathrm{m}) - \mathrm{Poly}(k, \mathrm{m}') - d \pmod p$ to be zero. Poly is a hash family, and choosing a random function h from the hash family is done by choosing a random $0 \le k < p$ and letting $h(\cdot) = \mathrm{Poly}(k, \cdot)$. Poly is (ℓ/p)-ADU (for the operation of addition modulo p) over messages in $\mathbb{Z}_p^\ell$.

One can easily extend this polynomial hash to messages of different lengths. Let m be a vector in $\mathbb{Z}_p^\ell$ and t be a number greater than the length of any message to be hashed. Message m can converted to an element of $\mathbb{Z}_p^t$ by prepending $t - (\ell + 1)$ zeroes and a 1. The resulting vector $(0, 0, \ldots, 0, 1, m_1, m_2, m_3, \ldots, m_\ell)$ is hashed as above, resulting in $k^{\ell+1} + m_1 k^\ell + m_2 k^{\ell-1} + \cdots + m_\ell k^1 \bmod p$. This padding process is bijective, meaning that any messages differing before padding also differ afterward, and produces equal length messages to be hashed. The highest degree possible as a result of padding is t, and so the resulting polynomial hash family is (t/p)-ADU.

NMH. Wegman and Carter invented a hash family that Halevi and Krawczyk named NMH^* [4]. We do not need the general version of NMH^*, which allows hashing long messages. We need only hash short fixed length messages. Let p be a prime, and k_1, k_2, m_1, m_2 all be in $\mathbb{Z}_p$. Define

$$h_{k_1,k_2}(m_1, m_2) = (m_1 + k_1) \times (m_2 + k_2) \bmod p\,,$$

and define $H = \{h_{k_1,k_2} \,|\, k_1, k_2 \in \mathbb{Z}_p\}$. Halevi and Krawczyk claim the following, but give no proof.

Lemma 3. *H is $(1/p)$-ADU (for addition modulo p).*

Proof. Let p be prime and $d, m_1, m_2, m'_1, m'_2 \in \mathbb{Z}_p$ with $(m_1, m_2) \neq (m'_1, m'_2)$. To show H is $(1/p)$-ADU for addition modulo p, we need show that $\Pr_{k_1,k_2}[h_{k_1,k_2}(m_1, m_2) + d = h_{k_1,k_2}(m'_1, m'_2) \bmod p] \leq 1/p$. Without loss of generality, assume $m_2 \neq m'_2$ and fix k_2. Substituting the definition of h, our goal is achieved by evaluating $\Pr_{k_1}[(m_1+k_1)\times(m_2+k_2)+d = (m'_1+k_1)\times(m'_2+k_2) \bmod p]$, which, multiplying and rearranging terms, is $\Pr_{k_1}[k_1(m_2 - m'_2) = m'_1m'_2 + m'_1k_2 - m_1m_2 - m_1k_2 - d \bmod p]$. But $c = m_2 - m'_2$ is nonzero and $d' = m'_1m'_2 + m'_1k_2 - m_1m_2 - m_1k_2 - d$ is constant, so the probability simplifies to $\Pr_{k_1}[k_1c = d' \bmod p]$, which is exactly $1/p$. □

1.2 Analysis

Our next lemma gives a general result, describing what happens when an ε-ADU hash family, with the operation being addition modulo a, is considered instead for the operation of addition modulo b.

Lemma 4. *Let a and b be positive integers and $H = \{h : A \to \mathbb{Z}_a\}$ be ε-ADU for addition modulo a. Define $H_b = \{h \bmod b \mid h \in H\}$. Then, H_b is $(\lceil\frac{2a-1}{b}\rceil\epsilon)$-ADU for addition modulo b.*

Proof: Let c be an element of $\mathbb{Z}_b$ and $m \neq m'$ be elements of A. We need to show $\Pr_h[h(m) + c = h(m') \pmod b] \leq \lceil\frac{2a-1}{b}\rceil\epsilon$. To do so, we must evaluate $\Pr_h[h(m) + c = h(m') \pmod b] = \Pr_h[h(m') - h(m) = c \pmod b] = \Pr_h[h(m') - h(m) = c + ib$ for some integer $i]$. The range of possible values for $h(m') - h(m)$ is $\{-(a-1), \ldots, (a-1)\}$, meaning at most $\lceil\frac{2a-1}{b}\rceil$ values for i make possible $h(m') - h(m) = c + ib$. For every i, the event $h(m') - h(m) = c + ib$ is a subset of the event $h(m') - h(m) = c + ib \pmod a$, so $\Pr_h[h(m') - h(m) = c + ib] \leq \Pr_h[h(m') - h(m) = c + ib \pmod a] \leq \epsilon$. Since there are at most $\lceil\frac{2a-1}{b}\rceil$ values of i for which $h(m') - h(m) = c + ib$, and the probability of each collision is no more than ϵ, we may conclude $\Pr_h[h(m) + c = h(m') \pmod b] \leq \lceil\frac{2a-1}{b}\rceil\epsilon$. □

Notice that choosing a to be a multiple of b can improve this bound. Consider $a = 2^{128}$ and $b = 2^{126}$. The theorem says that at most $\lceil\frac{2a-1}{b}\rceil = 8$ values for i make $h(m') - h(m) = c + ib \pmod a$, when in fact no more than four such i values can exist. With a, b and c fixed and $b|a$, $c + ib \pmod a = c + jb \pmod a$ whenever $i = j \pmod{\frac{a}{b}}$. This is because a is a multiple of b, and so increasing i by a/b has no net effect modulo a. The following corollary is a result of this observation.

Corollary 5. *Let a and b be positive integers where $b|a$, and $H = \{h : A \to \mathbb{Z}_a\}$ be ε-ADU for addition modulo a. Define $H_b = \{h \bmod b \mid h \in H\}$. Then, H_b is $(\frac{a}{b}\epsilon)$-ADU for addition modulo b.*

Our final lemma allows us to look at the first two stages of VHASH hashing from a different perspective, achieving tighter bounds in our analysis. Lines 1–5 of the VHASH definition (Figure 1) prepare message M for hashing by transforming it into $n = \max(\lceil|M|/b\rceil, 1)$ strings $M_1, M_2, \ldots, M_n$ with $|M_1| = |M_2| = \cdots = |M_{n-1}| = b$. (If M is the empty string then so is M_1.) For purposes of analysis, we will group these substrings as

$$\begin{aligned} M_{\text{front}} &= M_1 \,||\, M_2 \,||\, \ldots \,||\, M_{n-1} \text{ and } M_{\text{back}} = M_n && \text{if } |M_n| \neq 0, \\ M_{\text{front}} &= M_1 \,||\, M_2 \,||\, \ldots \,||\, M_n \text{ and } M_{\text{back}} = \lambda && \text{if } |M_n| = b, \text{ and} \\ M_{\text{front}} &= \lambda \text{ and } M_{\text{back}} = \lambda && \text{if } |M| = 0. \end{aligned}$$

Notice that if M and M' are distinct messages, and each go through the process of Lines 1–5 and we interpret the results as just described, producing M_{front}, M_{back}, M'_{front} and M'_{back}, then $(M_{\text{front}}, M_{\text{back}}) \neq (M'_{\text{front}}, M'_{\text{back}})$ always. Lines 6–7 then compute

$$\text{Poly}(k, \text{NH}^*[16, 64, 2^{126}](K, M_{\text{front}})) + \text{NH}[16, 64, 2^{126}](K, M_{\text{back}}) \bmod (2^{127} - 1)\,,$$

which the following lemma allows us to claim as an ε-ADU hash family (for addition modulo $2^{127} - 1$) with $\varepsilon = 1/2^{62} + (\ell/b)2^{-115}$.

Theorem 6. *Let A, B and C be non-empty sets; a, b, c and q be positive integers; and $H_a = \{h : A \to \mathbb{Z}_a\}$, $H_b = \{h : B \to \mathbb{Z}_b\}$ and $H_c = \{h : C \to \mathbb{Z}_c\}$ be families of functions. If $\mathbb{Z}_b \subseteq A$, H_a is ε_a-ADU for addition modulo a, H_b is ε_b-AU and H_c is ε_c-ADU for addition modulo c, then $H = \{h \mid h(x, y) = h_a(h_b(x)) + h_c(y) \pmod q$ with $h_a \in H_a, h_b \in H_b, h_c \in H_c\}$ is ε-ADU for addition modulo q where $\varepsilon = \max(\lceil\frac{2a-1}{q}\rceil\varepsilon_a + \varepsilon_b, \lceil\frac{2c-1}{q}\rceil\varepsilon_c)$.*

	64-bit Tags			128-bit Tags		
Architechture	64B	512B	4KB	64B	512B	4KB
64-bit Intel Core 2 "Merom"	$6.0 \rightarrow 6.0$	$1.2 \rightarrow 1.2$	$0.6 \rightarrow 0.6$	$6.7 \rightarrow 7.0$	$1.7 \rightarrow 1.7$	$1.2 \rightarrow 1.1$
32-bit PowerPC "G4"	$15.2 \rightarrow 13.9$	$6.6 \rightarrow 6.4$	$5.7 \rightarrow 5.6$	$22.5 \rightarrow 21.3$	$11.3 \rightarrow 11.9$	$10.3 \rightarrow 11.1$
32-bit ARM v5TE	$41.5 \rightarrow 38.6$	$12.9 \rightarrow 13.1$	$9.8 \rightarrow 10.1$	$52.6 \rightarrow 51.9$	$22.9 \rightarrow 22.1$	$20.0 \rightarrow 19.9$

Table 1. VMAC Performance. *Changes in VMAC cycle-per-byte performance are listed as "old → new" for various architectures and tag lengths over 64-, 512- and 4096-byte, cache-resident messages.*

Proof: Let d be an element of $\mathbb{Z}_q$, and let $(x, y) \neq (x', y')$ be elements of $A \times B$. We need to show $\Pr_h[h(x, y) + d = h(x', y') \pmod q] \leq \varepsilon$. To do so we must evaluate $\Pr_{h_a,h_b,h_c}[h_a(h_b(x)) + h_c(y) + d = h_a(h_b(x')) + h_c(y') \pmod q]$.

If $x \neq x'$, let $h_b \in H_b$ be chosen randomly and let $h_c \in H_c$ be chosen arbitrarily. (Letting h_c be chosen arbitrarily allows h_b and h_c to be dependent, if desired.) We know $\Pr_{h_b}[h_b(x) = h_b(x')] \leq \varepsilon_b$. Assuming $m = h_b(x)$ and $m' = h_b(x')$ differ and letting $d' = h_c(y) + d - h_c(y')$, our probability becomes $\Pr_{h_a}[h_a(m) + d' = h_a(m') \pmod q]$, which according to Lemma 4 is no more than $(\lceil \frac{2a-1}{q} \rceil \varepsilon_a)$. Because we assumed $h_b(x) \neq h_b(x')$, we must compensate for the possibility that they are equal, so $\Pr_{h_a,h_b,h_c}[h_a(h_b(x)) + h_c(y) + d = h_a(h_b(x')) + h_c(y') \pmod q] \leq \lceil \frac{2a-1}{q} \rceil \varepsilon_a + \varepsilon_b$. Then $\Pr_{h_a,h_b,h_c}[h_a(h_b(x)) + h_c(y) + d = h_a(h_b(x')) + h_c(y')]$

$$
\begin{aligned}
&\leq \Pr_{h_a,h_b,h_c}[h_a(h_b(x)) + h_c(y) + d = h_a(h_b(x')) + h_c(y') \mid h_b(x) \neq h_b(x')] + \Pr_{h_b}[h_b(x) = h_b(x')] \\
&\leq \Pr_{h_a,h_b,h_c}[h_a(h_b(x)) + h_c(y) + d = h_a(h_b(x')) + h_c(y') \mid h_b(x) \neq h_b(x')] + \varepsilon_b \\
&= \Pr_{h_a}[h_a(m) + d' = h_a(m') \mid m \neq m'] + \varepsilon_b \\
&\leq \lceil \frac{2a-1}{q} \rceil \varepsilon_a + \varepsilon_b
\end{aligned}
$$

If $x = x'$ but $y \neq y'$, then $\Pr_{h_a,h_b,h_c}[h_a(h_b(x)) + h_c(y) + d = h_a(h_b(x')) + h_c(y') \pmod q] = \Pr_{h_c}[h_c(y) + d = h_c(y') \pmod q]$ due to like-terms canceling. This probability according to Lemma 4 is no more than $(\lceil \frac{2c-1}{q} \rceil \varepsilon_c)$. □

The remainder of the VHASH construction is straightforward. Lines 8–9 form a reversible mapping from the product of Lines 1–7 to the input of Line 10. Standard theorems about the composition of hash functions (eg, [1]) gives us the claimed bound of Theorem 1.

2 Performance

The changes detailed in this note have had a small effect on performance (see Table 1). Using the revised VHASH in VMAC over a sample of architectures, we saw no performance changes of greater than ten percent, but with a slight trend toward improved performance. Of the eighteen combinations of architecture, tag length and message length listed in Table 1, eleven improve, five deteriorate and three are unchanged when comparing the original VMAC with the revised version.

Why are performance changes not uniform? Conflicting forces are at work, each with its own architecture-dependent influence. Degree reduction in the polynomial hash improves performance in all cases by eliminating one or two 128-bit multiplications, but the gain is most marked on short messages (because constant savings are most pronounced on short messages) and on 32-bit architectures (because 128-bit multiplications are four times more expensive on 32-bit architectures than on 64-bit ones). On the negative side, defining VMAC-128 as two iterations of VMAC-64 introduces an additional 32-bytes of key, a separate polynomial computation and a third hashing stage, all of which are slowing influences. Also, the switch in the third stage from a modulus of $2^{61} - 1$ to $2^{64} - 257$ adds a small constant amount of extra work.

The result is a hash function that is usually a bit faster, but always with better hash collision properties, than the original.

References

1. Bierbrauer J, Johansson T, Kabatianskii G, Smeets B. On families of hash functions via geometric codes and concatenation. In *Advances in Cryptology – CRYPTO 1993*. Springer-Verlag, 1993; 331–342.
2. Black J, Halevi S, Krawczyk H, Krovetz T, Rogaway P. UMAC: Fast and secure message authentication. In *Advances in Cryptology – CRYPTO 1999*. Springer-Verlag, 1999; 216–233.
3. Carter L, Wegman M. Universal classes of hash functions. *J. of Computer and System Sciences* 1981; **22**:265–279.
4. Halevi S, Krawczyk H. MMH: Software message authentication in the Gbit/second rates. In *Fast Software Encryption – FSE 1997*. Springer-Verlag, 1997; 172–189.
5. Krawczyk H. LFSR-based hashing and authentication. In *Advances in Cryptology – CRYPTO 94*. Springer-Verlag, 1994; 129–139.
6. Krovetz T. Message authentication on 64-bit architectures. In *Selected Areas of Cryptography – SAC 2006*. Springer-Verlag, 2006.
7. Stinson D. Universal hashing and authentication codes. *Designs, Codes and Cryptography* 1994; **4**:369–380.
8. Wegman M, Carter L. New hash functions and their use in authentication and set equality. *J. of Computer and System Sciences* 1979; **18**:143–154.