On the Minimum Number of Multiplications Necessary for Universal Hash Functions

Mridul Nandi(✉)

Applied Statistics Unit, Indian Statistical Institute, Baranagar, India
mridul@isical.ac.in

Abstract. Let $d \geq 1$ be an integer and R_1 be a finite ring whose elements are called **block**. A d-block universal hash over R_1 is a vector of d multivariate polynomials in message and key block such that the maximum *differential probability* of the hash function is "low". Two such single block hashes are pseudo dot-product (PDP) hash and Bernstein-Rabin-Winograd (BRW) hash which require $\frac{n}{2}$ multiplications for n message blocks. The Toeplitz construction and d independent invocations of PDP are d-block hash outputs which require $d \times \frac{n}{2}$ multiplications. However, here we show that *at least* $(d-1) + \frac{n}{2}$ *multiplications are necessary* to compute a universal hash over n message blocks. We construct a *d-block universal hash, called* EHC*, which requires the matching* $(d-1)+\frac{n}{2}$ *multiplications for* $d \leq 4$. Hence it is optimum and our lower bound is tight when $d \leq 4$. It has similar parllelizibility, key size like Toeplitz and so it can be used as a light-weight universal hash.

Keywords: Universal hash · AXU hash · Multivariate polynomial · Error correcting code · Vandermonde matrix · Toeplitz hash

1 Introduction

Universal hash function and its close variants *ΔU hash* [10,13,40,42,43] are used as building blocks of several cryptographic constructions, e.g., *message authentication codes* [10,49], domain extension of *pseudorandom functions* [2,4], extractors [15,32] and quasi-randomness [44]. It also has close connection with *error correcting codes* and other combinatorial objects [13,43].

Informally, a universal hash function h takes two inputs, a key k and a *message m of arbitrary length*, and produces a *fixed-length* output $h_k(m) := h(k,m)$. For a universal (or ΔU) hash function h the following holds: for any two distinct messages m_1, m_2, the collision probability $\mathsf{Pr}[h_k(m_1) = h_k(m_2)]$ (or differential probability $\max_\delta \mathsf{Pr}[h_k(m_1) - h_k(m_2) = \delta]$) is small for an uniformly chosen key k. Formal definitions can be found in Sect. 2.

A very popular application of universal hash is to obtain a domain extension of pseudorandom function (or PRF) and message authentication code (or MAC). Let f be a PRF over fixed length input. When h has low collision probability, the composition function $f \circ h$ is a PRF [4] over arbitrary length. Thus h behaves

C. Cid and C. Rechberger (Eds.): FSE 2014, LNCS 8540, pp. 489–508, 2015.
DOI: 10.1007/978-3-662-46706-0_25

as a preprocessor to reduce the problem of designing arbitrary size input PRF to a fixed small size input PRF. Similarly, we can show that mapping (N, M) to $f(N) \oplus h(M)$ is a MAC [6] over arbitrary length messages when N is used as a nonce (i.e., not repeating) and h has low differential probability. These methods are only useful when we process long message and use a much faster h (than a PRF f). So our main question of this paper is that how fast a universal hash function could be in a reasonable computational model?

Multiplication Complexity. The above question has been answered [26,28] in terms of order in different circuit level computational models, e.g., branching model. In this paper, we consider "multiplication complexity" in "algebraic computation model" [24] in which a polynomial or a rational function is computed by using addition and multiplication (or division) over an underlying ring (or a field) R_1 in a sequence (see Definition 4 for more details). For example, to compute $x_1x_2 + x_1x_3$, we can compute it as $v_1 + v_2$ where $v_1 = x_1x_2$ and $v_2 = x_1x_3$. This computation requires two multiplications. However, the same polynomial can be computed as $x_1(x_2 + x_3)$ which requires only one multiplication. We define multiplication complexity of a multivariate polynomial as the minimum number of multiplications required for all possible computations of H. The multiplication complexities of some standard multivariate polynomials have been studied before and a brief survey is given in Appendix. Our target question of this paper is to **obtain a lower bound** of multiplication complexities among all ΔU hash functions and to show the **tightness of the bound by producing an example**.

1.1 Our Contribution and Outline of the Paper

In the following we assume a universal hash function hashes all messages from R_1^ℓ to R_1^d (usually $d = 1$) and hence multiplication complexity is measured in terms of ℓ and d.

Optimality of pseudo dot-product and BRW hash. In this paper **we prove that a hash function with low differential probability must have multiplication complexity at least** $\ell/2$ (see Theorem 3 in Sect. 5). We show it by proving contrapositive. If a function has multiplication complexity $c < \ell/2$ then there are $2c$ multiplicands. As we have ℓ message blocks and key blocks are linear in multiplicands we are able to solve for two distinct messages from R_1^ℓ which map to all $2c$ multiplicands identically for all keys. Hence differential probability is one. Even though the lower bound seems intuitive, to the best of our knowledge, it was not known before. The pseudo dot-product [46] based hash PDP (e.g. NMH hash [14], NMH* [14], NH [4] and others [8,21]) defined as (for even ℓ)

$$\mathsf{PDP}_{k_1,\ldots,k_\ell}(m_1, \ldots, m_\ell) = (m_1 + k_1)(m_2 + k_2) + \cdots + (m_{\ell-1} + k_{\ell-1})(m_\ell + k_\ell)$$

and Bernstein-Rabin-Winograd or BRW hash [7,36] are two known examples which achieve this bound ($\ell/2$ multiplications for ℓ message blocks).

Optimality of multiple block hash. We also extend this bound for multiple block hash outputs (such as Toeplitz construction [26] or independent applications of a single block hash function). To compute

$$(H_1 := x_1x_2 + x_3x_4, \quad H_2 := x_1x_3 + x_2x_4),$$

we can compute H_1 by two multiplications (it can be shown that H_1 or H_2 individually can not be computed in one multiplication only) and then compute $H_2 = (x_1 + x_4)(x_2 + x_3) - H_1$ by one multiplication. Similarly for d polynomials $H_1, \ldots, H_d$ (with individual multiplication complexity c) there is a scope of computing all d polynomials simultaneously in less than cd multiplications. In Theorem 5 (Sect. 5), **we prove that to obtain d block hash outputs on ℓ block messages, we need at least $(d-1) + \ell/2$ multiplications**.

Construction with Matching Complexity. So far, no construction is known achieving this lower bound for $d > 1$. Note that both Toeplitz and independent invocation applied to the PDP requires $\ell d/2$ multiplications.[1] So there is a possibility of scope of a hash construction having better multiplication complexity. In this paper, for $d \leq 4$ **we provide a d-block Δ-universal hash, called** xxx **EHC or encode-hash-combiner** (see Algorithm 1, in Sect. 4). The main ingredient of the construction was introduced in [17]. Here, we first encode the input using an efficiently computable linear error correcting code [27] with minimum distance d so that codewords for different inputs will differ in at least d places; then we feed the i^{th} **encoded symbol** (e.g., a pair of blocks for PDP) through its underlying universal hash function (e.g., PDP which requires one multiplication for a symbol or two blocks); and then apply another efficient linear combiner to the all hash outputs to obtain the final d block hash outputs. The optimization in [17] is entirely aimed at linear-time asymptotic encodings [41], which don't have much connection to concrete performance. Moreover, codewords can not be computed in online manner with small buffer and requires at least ℓ-blocks of memory (in addition to input and output size). This is the possible reason that community has not found any interest to implement it (even for smaller ℓ, i.e. for small messages).

Our choice of code and combiners (satisfying some desired property) are based on Vandermonde matrices which can be computed with small buffer. Number of multiplication will essentially depend on size of codewords and due to choice of MDS code we need exactly $(d-1) + \ell/2$ multiplications. Hence, the construction is optimum and our bound is tight. In terms of key size and parallelizibilty, both Toeplitz and EHC are similar. The idea trivially does not extend for $d > 4$ as we do not find any appropriate error correcting code with distance $d > 5$.

2 Definitions: Universal and Δ-universal Hash Function

ΔU Hash Function. A hash function h is a $(\mathcal{K}, D, R)$-family of functions $\{h_k := h(k, \cdot) : D \to R\}_{k \in \mathcal{K}}$ defined on its domain or message space D, taking

[1] Applying the Theorem 1 in [47], we can prove that these constructions have multiplication complexity $\ell d/2$.

values on a group R, called *output space* and indexed by the **key space** $\mathcal{K}$. Usual choices of R are (i) $\mathbb{Z}_p$ (the field of modulo a prime p), (ii) $\mathbb{Z}_{2^w}$ (the ring of modulo 2^w) (iii) $\mathbb{F}_{2^n}$ (Galois field of size 2^n) and (iv) R_1^d with **coordinate wise operation**, where R_1 is one of the previous choices. In the last example when $d > 1$, h is also called **multi** or d**-block** hash. An element of R (or R_1 for the multi-block) is called block. In general, we write R_1 even for $d = 1$. However, the output space is always denoted by $R = R_1^d$, $d \geq 1$. Except for $(\mathbb{Z}_p)^d$, R can be viewed as the set $\{0,1\}^N$ by using the canonical encodings and we say that hash size is N.

Definition 1 (ϵ-ΔU hash function). *A* $(\mathcal{K}, D, R)$*-family* h *is called* ϵ-Δ**U (universal) hash function** *if for any two distinct* x *and* x' *in* D *and a* $\delta \in R$*, the* δ-differential probability $\mathsf{diff}_{h,\delta}[x,x'] := \mathsf{Pr}_{\mathbf{K}}[h_{\mathbf{K}}(x) - h_{\mathbf{K}}(x') = \delta] \leq \epsilon$ *where the random variable* $\mathbf{K}$ *is uniformly distributed over the set* $\mathcal{K}$*.*

Unless mentioned explicitly, we always mean key $\mathbf{K}$ to be chosen uniformly from its key space. The *maximum* δ*-differential probability* over all possible of two distinct inputs x, x' is denoted by $\Delta_{h,\delta}$. The *maximum differential probability* $\Delta_h := \max_\delta \Delta_{h,\delta}$. If the addition is bit-wise xor "$\oplus$" on $R = \{0,1\}^N$, we call the hash family ϵ**-AXU** (almost-xor-universal) hash function [37].

Universal Hash Function. When $\delta = 0$, the 0-differential event is equivalent to collision. So we write $\mathsf{diff}_{h,0}[x,x']$ and $\Delta_{h,0}$ by $\mathsf{coll}_h[x,x']$ and coll_h respectively and we call them collision probabilities.

Definition 2 (ϵ-U hash function). *A hash family* h *is called* ϵ**-universal** *(or* ϵ*-U) if* $\mathsf{coll}_h := \max_{x \neq x'} \mathsf{Pr}_{\mathbf{K}}[h_{\mathbf{K}}(x) = h_{\mathbf{K}}(x')] \leq \epsilon$*.*

Balanced Hash Function. We call h ϵ**-balanced** [23,31] on a subset $D' \subseteq D$ if $\mathsf{Pr}[h_{\mathbf{K}}(x) = y] \leq \epsilon$ for all $x \in D'$, $y \in R$. If $D' = D$ then we call it ϵ-balanced. Note that ϵ is always at least $1/|R|$ for ϵ-ΔU (shown in [43]) and ϵ-balanced function (easy to check from definition) but not necessarily for an ϵ-U hash function [43]. An ϵ-balanced functions are useful to prove ϵ-ΔU property whenever h_K's are linear [23]. More precisely, for a linear hash, ϵ-ΔU is equivalent to ϵ-balanced function on $R \setminus \{0\}$.

3 Analysis Methods of Universal Hash Functions

In this section all messages (and possibly key) blocks are elements of the underlying field R_1 of size q.

3.1 Multi-linear Hash and Poly-Hash

The hash mapping $(m_1, \ldots, m_l) \mapsto m_1 \cdot \mathbf{K}_1 + \cdots + m_\ell \cdot \mathbf{K}_\ell$ can be shown to be q^{-1}-ΔU hash function.[2] It is known as *multi-linear hash* ML[13,49]. Later MMH

[2] One can also prove it by applying Lemma 2 as it is a sum hash.

was proposed [14] with a performance record. It is a multi-linear hash with a specific choice of $R_1 = \mathbb{Z}_{2^{64}}$ and a post-processor. All these constructions above requires (at least) ℓ multiplications and ℓ many independent key blocks.

By counting roots of a polynomial, one can show that $(m_1, \ldots, m_\ell) \mapsto m_1 \cdot \mathbf{K} + m_2 \cdot \mathbf{K}^2 + \cdots + m_\ell \cdot \mathbf{K}^\ell$ is an $\ell \times q^{-1}$-ΔU hash function. This is known as poly-hash [3,9,45]. Some examples are Ghash used in GCM [22], poly1305 [6], polyQ, polyR [25] (combination of two poly-hashes), and others [18,20] etc. The speed report of these constructions are given in [30,40].

Bernstein-Rabin-Winograd hash or BRW [7,36] hash is a multi-variate polynomial hash which is non-linear in message blocks. It requires $\ell/2$ multiplication and one key. As the algorithm is recursive and binary tree based, it requires $O(\log \ell)$ storage. This construction uses minimum number of keys (single block) and requires minimum number of multiplications (as we show in Theorem 3 of Sect. 5).

3.2 Composition of Universal Hashes

Given an ϵ_1-universal $(\mathcal{K}, D, D')$-hash function h and ϵ_2-ΔU $(\mathcal{K}', D', R_1)$-hash function h' the composition hash function defined below

$$(h' \circ h)_{k,k'}(m) = h'_{k'}(h_k(m)),\ \forall m \in D$$

is $(\epsilon_1 + \epsilon_2)$-$\Delta$U-hash function on D. Whenever h' is assumed to be only ϵ_2-U hash function, the composition is $(\epsilon_1 + \epsilon_2)$-U-hash function [40]. This composition results are useful to play with domain and range for different choices and has been used in several constructions [4,25,39].

3.3 Pseudo Dot-Product

The notion of pseudo dot product hash is introduced for preprocessing some cost in matrix multiplications [46]. The construction NMH [14] uses this idea. NMH* and NH are variants of these construction. Later on NH has been modified to propose some more constructions [8,19,21,31]. A general form of pseudo dot-product PDP is $(m_1+\mathbf{K}_1)(m_2+\mathbf{K}_2)+\ldots+(m_{\ell-1}+\mathbf{K}_{\ell-1})(m_\ell+\mathbf{K}_\ell)$ which is same as multi-linear hash plus a function of messages and a function of keys separately. The main advantage of PDP is that, unlike multi-linear hash, it requires $\ell/2$ multiplications to hash ℓ message blocks. We first prove a general statement which is used to prove ΔU property of PDP.

Lemma 1. *Let h be an ϵ-ΔU $(\mathcal{K}, D, R_1)$-hash function where R_1 is an additive group. Then the following $(\mathcal{K}, D, R_1)$-hash function h'*

$$h'_k(m) = h_k(m) + f(k) + g(m). \tag{1}$$

is ϵ-ΔU hash function for any two functions f and g mapping to R_1.

Proof. For any $m \neq m'$ and δ, $h'_k(m) - h'_k(m') = \delta$ implies that $h_k(m) - h_k(m') = \delta' := \delta + g(m') - g(m)$. So for all $m \neq m'$ and δ,

$$\Pr[h'_k(m) - h'_k(m') = \delta] \leq \max_{\delta'} \Pr[h_k(m) - h_k(m') = \delta'] \leq \epsilon$$

and hence the result follows. □

Corollary 1 (Pseudo dot-product hash). *Let R_1 be a field of size q. The hash function $(m_1, m_2, \ldots, m_\ell) \mapsto (m_1 + \mathbf{K}_1)(m_2 + \mathbf{K}_2) + \ldots + (m_{\ell-1} + \mathbf{K}_{\ell-1})(m_\ell + \mathbf{K}_\ell)$ is q^{-1}-ΔU hash function for a fixed ℓ.*

Corollary 2 (Square hash [11]). *Let R_1 be a field of size q. The hash function $(m_1, m_2, \ldots, m_\ell) \mapsto (m_1 + \mathbf{K}_1)^2 + \cdots + (m_\ell + \mathbf{K}_\ell)^2$ is q^{-1}-ΔU hash function for a fixed ℓ.*

3.4 Message Space Extension: Sum Hash Construction

Now we provide some easy generic tools to hash larger message. Let h be an ϵ-ΔU hash function from D to R_1 with key space $\mathcal{K}$. A hash function is called **sum hash** (based on h), denoted h^{sum} if it is defined as

$$h^{\mathrm{sum}}_{k_1,\ldots,k_s}(m_1, \ldots, m_s) = \sum_{i=1}^{s} h_{k_i}(m_i), \tag{2}$$

The multi-linear hash ML and PDP are two examples of sum-hash.

Lemma 2. *If h is an ϵ-ΔU hash function from D to R_1 with key space $\mathcal{K}$ then h^{sum} is an ϵ-ΔU $(\mathcal{K}^s, D^s, R_1)$-hash function.*

Proof. One can verify it in a straightforward manner once we condition all keys $\mathbf{K}_j$'s except a key $\mathbf{K}_i$ for which $m_i \neq m'_i$ (the i^{th} elements of two distinct inputs m and m'). □

Universal Hash for Variable Length. The above sum-hash is defined for a fixed number of message blocks. Now we define a method which works for arbitrary domain $\overline{D} := \{0,1\}^{\leq t}$. To achieve this, we need a padding rule which maps $\overline{D}$ to $D^+ = \cup_{i \geq 1} D^i$. A padding rule $\mathsf{pad} : \overline{D} \to D^+$ is called D'-restricted if it is an injective function and for all $m \in D$ and $\mathsf{pad}(m) = (m_1, \ldots, m_s)$ we have $m_s \in D'$.

Lemma 3 (Extension for $\overline{D} := \{0,1\}^{\leq t}$). *Let h be an ϵ-ΔU $(\mathcal{K}, D, R_1)$-hash function and ϵ-balanced on $D' \subseteq D$ and $\mathsf{pad} : \overline{D} \to D^{\leq L}$ be a D'-restricted padding rule. The sum-hash $h^{\mathsf{pad},\mathrm{sum}}$, defined below, is an ϵ-ΔU $(\mathcal{K}^L, \{0,1\}^{\leq t}, R_1)$-hash function.*

$$h^{\mathsf{pad},\mathrm{sum}}_{K_1,\ldots,K_L}(m) = \sum_{i=1}^{s} h_{K_i}(m_i), \quad \mathsf{pad}(m) = (m_1, \ldots, m_s) \tag{3}$$

The proof is similar to fixed length sum-hash except that for two messages with different block numbers. In this case, the larger message uses an independent key for the last block which is not used for the shorter message and hence the result follows **by using balanced property of the hash**.

Remark 1. Note that ML is clearly not universal hash function for variable length messages. It is a sum hash applied on the hash $m \cdot \mathbf{K}$ which is not balanced on the field R_1 (the 0 message maps to 0 with probability one). However, it is q^{-1}-balanced for $R_1 \setminus \{0\}$. Hence for any padding rule pad which is injective and the last block is not zero will lead to an universal hash for ML construction. For example, the popular "10-padding" pads a bit 1 and then a sequence of zeros, if required, to make it a tuple of the binary field elements. This ensures that the last block has the bit 1 and hence it is non-zero.

The pseudo dot-product is $2q^{-1}$-balanced on R_1 and hence any injective padding rule for PDP will give a $2q^{-1}$-ΔU hash function.

An Alternative Method: Hashing Length. A generic way to handle arbitrary length is as follows: Let h be an ϵ-ΔU hash function on D_i, $1 \le i \le r$ and h' be an ϵ-ΔU hash function on $\{1, 2, \dots, r\}$. Then the hash function $H_{k,k'}(m) = h_k(m) + h'_{k'}(i)$ where $m \in D_i$ is an ϵ-ΔU hash function on $D := \cup_i D_i$. We apply this approach in our construction to define over arbitrary messages.

3.5 Toeplitz Construction: A Method for Multi-block Hash

One straightforward method to have a d-block universal hash is to apply d independent invocation of universal hash h. More precisely, for d independent keys $\mathbf{K}_1, \dots, \mathbf{K}_d$, we define a d-block hash as $h^{(d)} = (h_{\mathbf{K}_1}(m), \dots, h_{\mathbf{K}_d}(m))$. We call it *block-wise hash*. It is easy to see that if h is ϵ-U (or ΔU) then $h^{(d)}$ is ϵ^d-U (or ΔU) hash function. The construction has d times larger key size. However, for a sum-hash h^{sum} we can apply **Toeplitz construction**, denoted $h^{T,d}$, which requires only d additional key blocks where h is an ϵ-ΔU $(\mathcal{K}, D, R)$-hash function.

$$h_i^{T,d}(m_1, \dots, m_l) = h_{\mathbf{K}_i}(m_1) + h_{\mathbf{K}_{i+1}}(m_2) + \dots + h_{\mathbf{K}_{l+i-1}}(m_l), \quad 1 \le i \le d. \tag{4}$$

We define $h^{T,d}_{\mathbf{K}_1, \dots, \mathbf{K}_{l+d-1}}(m) = (h_1^{T,d}, \dots, h_d^{T,d})$. Note that it requires $d-1$ additional keys than the sum construction for single-block hash. However the number of hash computations is multiplied by d times. Later we propose a better approach for a d-block construction which requires much less multiplications.

Lemma 4. *h is ϵ-ΔU $(\mathcal{K}, D, R_1)$-hash $\Rightarrow h^{T,d}$ is ϵ^d-ΔU $(\mathcal{K}^{l+d-1}, D^l, R_1^d)$-hash.*

Proof. For two distinct messages $m \neq m'$ it must differ at some index. Let i be the first index where they differ i.e., $m_i \neq m'_i$ and $m_1 = m'_1, \dots, m_{i-1} = m'_{i-1}$. Now condition all keys except $\mathbf{K}' := (\mathbf{K}_i, \dots, \mathbf{K}_{i+d-1})$. Denote H_i and H'_i for the i^{th} block hash outputs for the messages m and m' respectively. Now, $H_d - H'_d = \delta_d$ leads a differential equation of h for the key $\mathbf{K}_{i+d-1}$ and so this would contribute probability ϵ. Condition on any such $\mathbf{K}_{i+d-1}$, the previous equation $H_{d-1} - H'_{d-1} = \delta_{d-1}$ can be expressed as an differential equation of $\mathbf{K}_{i+d-2}$ and so on. The result follows once we multiply all these probabilities. □

The above proof argument has similarities in solving a system of upper triangular linear equations. So we start from solving the last equation and once we solve it we move to the previous one and so on until we solve the first one.

Toeplitz Construction Applied to an Arbitrary Length. Now we describe how Toeplitz construction can be used for arbitrary length inputs. If h is ϵ-balanced on a set $D' \subseteq D$ then we need a padding rule which maps a binary string to $(m_1, \ldots, m_s) \in D^s$ such that $m_s \in D'$. This condition is same as sum hash construction for arbitrary length.

Lemma 5 (Toeplitz construction for $\overline{D} := \{0,1\}^{\leq t}$). *Let h be an ϵ-ΔU $(\mathcal{K}, D, R_1)$-hash function and ϵ-regular on $D' \subseteq D$ and pad be D'-restricted. Then the Toeplitz hash $h^{T,d,\mathsf{pad}}(m) = h^{T,d}(\mathsf{pad}(m))$ is an ϵ^d-ΔU $(\mathcal{K}^{L+d-1}, \{0,1\}^{\leq t}, R_1^d)$ hash function.*

The proof is similar to the fixed length proof and hence we skip the proof. The 10-padding rule (as mentioned for ML hash) for Toeplitz construction in ML can be used [38]. Similarly, for PDP one can use any injective padding rule. Generalized linear hash [38], LFSR-based hash [23], CRC construction or Division hash [23,40], Generalized division hash [40], Bucket hash [37], a variant of Toeplitz construction [31] etc. are some other examples of multi-block hash.

4 Our Constructions

4.1 Error-Correcting Coding

Let A be an alphabet. Any injective function $e : D \to A^n$ is called an *encoding function* of length n. Any element in the image of the encoding function is called code word. For any two elements $x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n) \in A^n$ we define hamming distance $d_{\mathrm{ham}}(x, y) = |\{i : x_i \neq y_i\}|$, the number of places two n-tuples differ. We can extend the definition for arbitrary size. Let $x = (x_1, \ldots, x_n) \in A^n, y = (y_1, \ldots, y_m) \in A^m$ where $m \leq n$. We define $d^*_{\mathrm{ham}}(x, y) = (n - m) + d_{\mathrm{ham}}(x', y)$ where $x' = (x_1, \ldots, x_m)$.

Definition 3. *The minimum distance for an encoding function $e : D \to A^{\leq L} := \cup_{i \leq L} A^i$ is defined as $d(e) \stackrel{\Delta}{=} \min_{M \neq M' \in D} d^*_{\mathrm{ham}}(e(M), e(M'))$.*

We know from coding theory that for any coding $e : A^k \to A^n$ we have $d(e) \leq n - k + 1$ (**singleton bound**). Moreover, there is a linear code[3] e, called MDS or **maximum distance separable** code, such that $d(e) = n - k + 1$. However, if we consider sequence of MDS codes applied to different length the combined coding

[3] There is a generator matrix $G_{k \times n}$ over the field $\mathbb{F}$ such that $e(x) = x \cdot G$.

may have minimum distance only one since we may find two distinct messages M, M' such that $e(M) = (x_1, \ldots, x_m)$ and $e(M') = (x_1, \ldots, x_m, x_{m+1})$. If the generator matrix of MDS code is of the systematic form $G = (I_k : S_{k\times(n-k)})$ with identity matrix I_k then S is called MDS matrix. A characterization of MDS matrix is that every square sub-matrix has full rank. There are some known systematic form of MDS code based on Vandermonde matrix [5]. We call a matrix $S_{k\times(n-k)}$ d-MDS if every square submatrix of size d has full rank. Thus, S is a MDS matrix if and only if it is d-MDS for all $1 \le d \le \min\{k, n-k\}$. Now we give examples of MDS and d-MDS matrix using a general form of Vandermonde matrix.

Vandermonde Matrix. We first define a general form of Vandermonde matrix $V_d := V_d(\alpha_1, \ldots, \alpha_n)$ over a finite field $\mathbb{F}_q$ where $d \le n$ are positive integers and $\alpha_1, \ldots, \alpha_n$ are distinct elements of the field. It is an $d \times n$ matrix whose $(i, j)^{\text{th}}$ entry is α_j^{i-1}. If $n = d$ then the matrix is invertible and it is popularly known as Vandermonde matrix, denoted $V(\alpha_1, \ldots, \alpha_s)$. Moreover note that any r' columns of V_d are linearly independent where $r' \le d$. In particular, V_d is a d-MDS matrix.

Lemma 6. *The matrix V_d defined above for n distinct elements $\alpha_1, \ldots, \alpha_n$ is d-MDS matrix.*

Proof. Let us take d columns $i_1, \ldots, i_d$ then the submatrix is the Vandermonde matrix of size d with d distinct elements $\alpha_{i_1}, \ldots, \alpha_{i_d}$. As the Vandermonde matrix is invertible the result follows. □

4.2 A General Construction

Let $d \ge 1$ be an integer. Our construction has three basic components.

(1) Let e be an error correcting code from a message space D to $A^{\le L}$ with the minimum distance d.
(2) Let $h : \mathcal{K} \times A \to R_1$ be an ϵ-ΔU and ϵ-balanced hash function.
(3) For each $l \ge d$, let $V_{d,l}$ be a d-MDS matrix (any d columns are linearly independent) of dimension $d \times l$ whose entries are from R_1.

We define a hash on D which is a composition of three basic steps encoding or expansion, block-wise-Hash and a linear combination. We apply the encoding function e to expand the message $m \in D$ to an l-tuple $(m_1, \ldots, m_l) \in A^l$. In this process we ensure at least d places would differ for two distinct messages. Then we apply the hash h alphabet-wise to obtain $h := (h_1, \ldots, h_l)$. Finally, we apply a linear combiner on the hash blocks to obtain d-block hash output $V_{d,l} \cdot h$. We call this general construction method EHC or encode-hash-combiner. The description of it is given in Algorithm 1.

Theorem 1. *If e has minimum distance d (i.e. $d(e) = d$) and h is ϵ-ΔU and ϵ-balanced function then the extend function H is ϵ^d-Δ universal hash function.*

Input: $m \in D$
Output: $H \in R_1^d$
Key: $(k_1, \ldots, k_L) \in \mathcal{K}^L$

Algorithm $\mathsf{EHC}(m)$
$e(m) = (m_1, \ldots, m_l) \in A^l$. \\ Apply encoding function
For all $j = 1$ to l \\ Apply hash block-wise.
    $h_j = h_{k_j}(m_j)$
$H = V_{d,l} \cdot (h_1, \ldots, h_l)^{\mathrm{tr}}$ \\ Apply d-MDS combiner .
Return H

Algorithm 1. A General Δ-universal hash construction. It uses an error correcting code $e : D \to R_1^+$ with a minimum distance d, a family of d-MDS matrix $V_{d \times l}$, $l \geq d$, and ΔU hash h from A to R_1 with key space $\mathcal{K}$.

Proof. Let $m \neq m'$ and $x = e(m) = (m_1, \ldots, m_l)$, $x' = e(m') = (x'_1, \ldots, x'_l)$. By definition of minimum distance of e, $d^*(e(m), e(m')) \geq d$. W.l.o.g we assume that $l \geq l'$ and $i_1 \leq \ldots \leq i_d \leq l$ are distinct indices at which the encoded messages differ. We condition all keys except $\mathbf{K}_{i_1}, \ldots, \mathbf{K}_{i_d}$. Now for any $\delta \in R_1^d$, $H(m) - H(m') = \delta$ implies that $V \cdot (a_{\mathbf{K}_{i_1}}, \ldots, a_{\mathbf{K}_{i_d}})^{tr} = \delta$ where $a_{\mathbf{K}_{i_j}} = h_{\mathbf{K}_{i_j}}(m_{i_j}) - h_{\mathbf{K}_{i_j}}(m'_{i_j})$ if $i_j \leq l'$ (we use ΔU property), otherwise, $a_{\mathbf{K}_{i_j}} = h_{\mathbf{K}_{i_j}}(m_{i_j})$ (we use balancedness property). Moreover, V is the sub-matrix of $V_{d,l}$ with the columns $i_1, \ldots, i_d$. Note that V is invertible and hence given differential event is equivalent to $(a_{\mathbf{K}_{i_1}}, \ldots, a_{\mathbf{K}_{i_d}})^{tr} = V^{-1} \cdot \delta = \delta'$. Since $\mathbf{K}_{i_j}$'s are independent $\mathsf{Pr}[a_{\mathbf{K}_{i_j}} = \delta'_j] \leq \epsilon$ (because h is ϵ-ΔU hash and ϵ balanced function). So the differential probability of H is at most ϵ^d. □

Remark 2. Note that the only non-linear part in key and message blocks appears in underlying hash computations. As the error correcting code and combiners are linear we only need to apply constant multiplications (which is also a linear function). For appropriate choices of constants, such as primitive element of the field R_1, the constant multiplication is much more efficient compare to non-constant multiplication.

4.3 Specific Instantiations

Specific Instantiations for Fixed Length. Let $d = 4$. Let R_1 be the Galois field of size 2^n and α be a primitive element. Note that R_1^2 can also be viewed as the Galois field of size 2^{2n} and let β be a its primitive element. The following coding function C_4 has minimum distance 4. $C_4(m_1, \ldots, m_t) = (m_1, \ldots, m_t, m_{t+1}, m_{t+2}, m_{t+3})$ where

- $m_{t+1} = \bigoplus_i m_i$,
- $m_{t+2} = \bigoplus_i m_i \beta^{i-1}$ and
- $m_{t+3} = \bigoplus_i m_i \beta^{2(i-1)}$.

Let $l = t+3$. The base hash function $h_{k,k'}(x, x') = (x \oplus k) \cdot (x' \oplus k')$ mapping $R_1^2 \to R_1$ with key space R_1^2. It is the pseudo-dot -product hash. Finally the d-MDS matrix can be replaced by Vandermonde matrix $V_{d,l} := V_d(1, \alpha, \alpha^2, \ldots, \alpha^l)$ wher α.

Proposition 1. *The coding C_4 defined above has minimum distance 4 over the alphabet R_1^2 for a fixed length encoded message.*

Proof. This coding has systemic form $(I : S)$ where $S = V_l(1, \beta, \beta^2)$. It is known that S is 3-MDS matrix. Now we show that it is also 1-MDS and 2-MDS matrix. Showing 1-MDS matrix is obvious as every entry of the matrix is non-zero. To show that S is 2-MDS we need to choose two columns of the three columns. If we include the first column then the sub-matrix is again a Vandermonde matrix. If we choose the last two columns and i_1 and i_2^{th} rows then the determinant of the sub-matrix is $\beta^{i_1+i_2-2}(\beta^{i_2} - \beta^{i_1})$. □

It is easy to see that if we drop the last column or last two columns we have error correcting code with distance 3 and 2 respectively. Similarly, one can have a specific instantiation with $d = 2$. We do not know so far any coding function for $d > 4$ which can be efficiently computed for any arbitrary length input in an online manner without storing the whole message. However, for short messages one can apply some pre-specified MDS codes. Note that it is not necessary to apply MDS code. However, applying MDS-code make the key size and the number of multiplication as low as possible.

Variable Length ΔU Hash. The above construction works for fixed size input. Note that C_4 does not have minimum distance (with extended definition) four for arbitrary length blocks. However, with the extended definition of distance, we observe that C_4 has minimum distance over $D_0 := \cup_{i \equiv 0 \mod 4} R_1^i$. Similarly, it has minimum distance 4 over D_1, D_2 and D_3. Let $\mathbf{K}^{(1)}, \mathbf{K}^{(2)} \in \mathcal{K}$ be dedicated keys for length, i.e. not used to process message. Now we define our hash function ECH^* for arbitrary length message m as follows.

$$\mathsf{ECH}^*(m) = \mathsf{ECH}(m) + b_1 \cdot \mathbf{K}^{(1)} + b_2 \cdot \mathbf{K}^{(2)}, b_1, b_2 \in \{0,1\}, l \equiv b_1 + 2b_2 \mod 4$$

where $e(m) = A^l$. Basically, we hash the length of codeword modulo 4. To analyze it works, we consider three cases for $e(m) \in D_j$ and $e(m') \in D_{j'}$.

1. If $j = j'$ then the previous theorem for fixed length works.
2. If $j \neq j'$ then the differential probability will be low due to the hash $b_1 \cdot \mathbf{K}^{(1)} + b_2 \cdot \mathbf{K}^{(2)}$ applied to two different two-bit string (b_1, b_2).

Theorem 2. *If h is an ϵ-ΔU hash function then the construction* EHC^* *is ϵ^d-ΔU hash function for variable length inputs.*

5 Lower Bound on Multiplications or Non-Linear Operations

A polynomial can be computed through a sequence of addition and multiplication. Moreover, we can combine all consecutive additions to a linear function.

When we multiply the multiplicands can be expressed as linear functions of all the variables and previously computed multiplication results. For example, poly hash $H := m_0 + km_1 + k^2m_2$ can be computed through the following sequence of computations.

$$v_1 = k \cdot m_2, v_2 = (v_1 + m_1) \cdot k, H = v_2 + m_0.$$

Definition 4 *An* **algebraic computation.** *A is defined by the tuple of linear functions $(L_1, \ldots, L_{2t}, L^1, \ldots, L^d)$ where L_{2i-1} and L_{2i} are linear functions over variables $m = (m_1, \ldots, m_l), k = (k_1, \ldots, k_s), v_1, \ldots, v_{i-1}$, $1 \le i \le t$ and L^i's are linear over m, k and $v_1, \ldots, v_t$. When we identify v_i by $L_{2i-1} \cdot L_{2i}$ recursively $1 \le i \le t$, L^i are multivariate polynomials (MVP). We call t the* multiplication complexity *of A.*

We also say that A computes the d-tuple of function $H := (L^1, \ldots, L^d)$. Multiplication complexity of H is defined as the minimum multiplication complexity of all algebraic computation which computes the d polynomials H. Note that while counting multiplication complexity, we ignore the constant multiplications which are required in computing L. This is fine when we are interested in providing lower bounds. However, for a concrete construction, one should clearly mention the constant multiplications also as it could be significant for a large number of such multiplications.

Let R be a ring. A linear function in the variables $x_1, \ldots, x_s$ over R is a function of the form $L(x_1, \ldots, x_s) = a_0 + a_1x_1 + \ldots + a_sx_s$ where $a_i \in R$. We denote the constant term a_0 by c_L. We also simply write the linear function by $L(x)$ where $x = (x_1, \ldots, x_s)$ is the vector of variables. We add or subtract two vectors coordinate-wise. Note that if $c_L = 0$ then $L(x - x') = L(x) - L(x')$.

Notation. We denote the partial sum $a_1x_1 + \ldots + a_ix_i$ by $L[x[1..i]]$ where $x[1..i]$ represents $x_1, \ldots, x_i$. If L is a linear function in the vectors of variables x and y then clearly, $L = a_L + L[x] + L[y]$. Now we state two useful lemmas which would be used to prove lower bounds of multiplication complexities of universal hashes.

Lemma 7 [43]. *Let H be a ϵ-ΔU hash function from S to T then $\epsilon \ge \frac{1}{|T|}$.*

Lemma 8. *Let R be a finite ring. Let $V : \mathcal{K} \times \mathcal{M} \xrightarrow{*} R^t$ be a hash function and L is a linear function on R^t. For any functions f and g, the following keyed function H*

$$H(K, x) = L(V(K, x)) + f(x) + g(K)$$

is ϵ-ΔU hash function if V is ϵ-ΔU hash function. Moreover, $\epsilon \ge \frac{1}{|R|^t}$.

Proof. By above lemma we have $x \ne x'$ and δ_1 such that $\mathsf{Pr}_K[V(K, x) - V(K, x') = \delta_1] \ge \frac{1}{|T|}$. Let $\delta = L(\delta_1) + (f(x) - f(x'))$ and hence $V(K, x) - V(K, x') = \delta_1 \Rightarrow H(K, x) - H(K, x') = \delta$. This proves the result. □

5.1 Minimum Number of Multiplications for ΔU Hash Function

Now we show our first lower bound on the number of multiplications for a ΔU hash function over a field $\mathbb{F}$ which is computable by addition and multiplication. Clearly, it must be a multivariate polynomial in key and message block and we call it multivariate polynomial or MVP hash function. The theorem shows that a ΔU MVP hash function requiring s multiplications can process at most $2s$ blocks of messages. In other words, any MVP hash function computable in s multiplications processing $2s+1$ message blocks has differential probability one. Intuitive reason is that if we multiply s times then there are $2s$ many linear functions of message m only. Thus, mapping $2s+1$ blocks to $2s$ linear functions would not be injective and hence we can find a collision. The detail follows.

Theorem 3. *Let $H(K, m_1, \ldots, m_l)$ be a MVP hash computable by using s multiplications with $2s+1 \leq l$. Then there are two distinct vectors $a, a' \in \mathbb{F}^l$ and $\delta \in \mathbb{F}$ such that $H(K, a) = H(K, a') + \delta$. for all keys K*

Proof. As H can be computed by s multiplications we have $2s+1$ linear functions $\ell_1, \ell_2, \ldots, \ell_{2s}$ and L such that ℓ_{2i-1} and ℓ_{2i} are linear functions over m, K and $v_1, \ldots, v_{i-1}$ where $v_i = \ell_{2i-1} \cdot \ell_{2i}$. Moreover, L is a linear function over m, K and $v = (v_1, \ldots, v_s)$ with $H = L$. Note that there are $2s$ many linear equations $\ell_i[m]$'s (the partial linear functions on x only) over at least $2s+1$ variables $m_1, \ldots, m_l$, we must have a non-zero solution $\Delta \in \mathbb{F}^l$ of $\ell_i[m]$'s. More precisely, there is non-zero $\Delta \in \mathbb{F}^l$ such that $\ell_i[\Delta] = 0$ for all $1 \leq i \leq 2s$. Let $a \in \mathbb{F}^l$ be any vector and $a' = a + \Delta$. Let us denote $v_i(K, a)$ and $v_i(K, a')$ by v_i and v_i' respectively.

Claim: $v_i = v_i'$ for all $1 \leq i \leq s$.

We prove the claim by induction on i. Note that

$$v_1 = (\ell_1[a] + \ell_1[K] + c_{\ell_1}) \cdot (\ell_2[a] + \ell_2[K] + c_{\ell_2})$$

and similarly for v_1'. We already know that $\ell_1[a] = \ell_1[a']$, $\ell_2[a] = \ell_2[a']$ and hence $v_1 = v_1'$. Suppose the result is true for all $j < i$. Then,

$$v_i = (\ell_{2i-1}[a] + \ell_{2i-1}[K] + \ell_{2i-1}[v_1, \ldots, v_{i-1}] + c_{\ell_i})$$
$$\times(\ell_{2i}[a] + \ell_{2i}[K] + \ell_{2i}[v_1, \ldots, v_{i-1}] + c_{\ell_2})$$

and similarly for v_i'. By using equality $\ell_{2i-1}[a] = \ell_{2i-1}[a']$ and $\ell_{2i}[a] = \ell_{2i}[a']$, and the induction hypothesis $v_1 = v_1', \ldots, v_{i-1} = v_{i-1}'$ we have $v_i = v_i'$.

Thus, $V : \mathcal{K} \times \mathbb{F}^l \to \mathbb{F}^s$, mapping (K, x) to $(v_1(K, x), \ldots, v_s(K, x))$ has collision probability 1. The hash function $H(K, x)$ is defined as $L[V(K, x)] + L[K] + L[x] + c_L$. So by using Lemma 8 the result follows. □

Corollary 3. *The pseudo dot product hash* PDP *is optimum in number of multiplications.*

Remark 3.

1. The above result holds even if we ignore the cost involving key only, such as stretching the key by using pseudorandom bit generator or squaring the key (it does for BRW hash) etc. Hence the BRW hash is also optimum if key processing is allowed.
2. From the proof one can actually efficiently construct a, a' and δ. We only need to solve $2s$ equations $\ell_i[x]$. By previous remark, the result can be similarly extended if we ignore cost involving message only, e.g., we apply cryptographic hash to message blocks. More precisely, v_i is defined as product of f_i and g_i where $f_i = f_i^1(x) + f_i^2(K) + f_i^3(v_1, \ldots, v_{i-1})$ and similarly g_i. By using non-injectivity of $x \mapsto (f_1, g_1, \ldots, f_s, g_s)$ we can argue that there are distinct a and a' such that the f_i and g_i values for a and a' are same. However, this gives an existential proof of a and a' (which is sufficient to conclude the above theorem).
3. Our bound is applicable when we replace multiplication by any function. More precisely, we have the following result.

Theorem 4. *Let $H(x_1, \ldots, x_l, y_1, \ldots, y_k)$ be a function where x_1, …, x_l, y_1, …, y_k are variables. Let $f_i : \mathbb{F}^k \times \mathbb{F}^{r_i} \to \mathbb{F}$ be some functions, $1 \le i \le m$. Suppose $H(\cdot))$ can be computed by s_i invocations of f_i, $1 \le i \le m$. If $l \ge \sum_i s_i r_i + 1$ then there are two distinct vectors $a = (a_1, \ldots, a_l)$ and $a' = (a'_1, \ldots, a'_l)$ from $\mathbb{F}^l$ and $\delta \in \mathbb{F}$ such that*

$$H(a, y_1, \ldots, y_k) = H(a', y_1, \ldots, y_k) + \delta,\ \forall y_1, \ldots, y_k.$$

The proof is similar to the above theorem and hence we skip.

Now we extend our first theorem to a multi-block hash output, e.g. Toeplitz hash function. So we work in the field $\mathbb{F}$ however, the hash output is an element of $\mathbb{F}^d$ for some $d \ge 1$. Thus, it can be written as $(H_1, \ldots, H_d)$. Again we restrict to those hash functions which can be computed by adding and multiplying (like previous remark, we will allow any processing involving message or key only). So H_i is a MVP hash function and we call H to be d-MVP hash function.

Theorem 5. *Let $H = (H_1, \ldots, H_d)$ be a vector of d polynomials in $m = (m_1$, …, $m_l)$ and K over a field $\mathbb{F}$ which can be computed by s multiplications. If $l \ge 2(s - r) + 1$ with $r \le d$, then there are $a \ne a'$, elements of $\mathbb{F}^r$ and $\delta \in \mathbb{F}$ such that*

$$\mathsf{Pr}_{\mathbf{K}}[H_{\mathbf{K}}(a) = H_{\mathbf{K}}(a') + \delta] \ge \frac{1}{|\mathbb{F}|^r}.$$

Proof. Suppose H can be computed by exactly s multiplications then we have $2s + d$ linear functions $\ell_1, \ell_2, \ldots, \ell_{2s}$ and $L_1, \ldots, L_d$ such that

(i) ℓ_{2i-1} and ℓ_{2i} are linear functions over m, K and $v_1, \ldots, v_{i-1}$
(ii) $v_i = \ell_{2i-1} \cdot \ell_{2i}$ and
(iii) L_i's are linear functions over x, y and $v = (v_1, \ldots, v_s)$.

Moreover, $H_i = L_i$ for all $1 \leq i \leq d$. The linear functions ℓ_i and L_i can be written as $\ell_i[m] + \ell_i[K] + \ell_i[v] + c_{\ell_i}$ and $L_i[m] + L_i[K] + L_i[v] + c_{L_i}$.

The first $2(s-r)$ many linear equations $\ell_i[m]$'s over at least $2(s-r)+1$ variables. Hence these will have a non-zero solution $\Delta \in \mathbb{F}^l$. Let a be any vector and $a' = a + \Delta$. It is easy to see that $v_i(a, K) = v_i(a', K)$ for all $i \leq s-r$ (similar to proof of Theorem 3). Now consider the mapping $f : \mathbb{F}^k \to \mathbb{F}^r$ mapping

$$K \mapsto (v_{s-r+1}(a, K) - v_{s-r+1}(a', K), \ldots, v_s(a, K) - v_s(a', K)).$$

There must exist $\delta_1 \in \mathbb{F}^r$ such that $\mathsf{Pr}_K[f(K) = \delta_1] \geq \frac{1}{|\mathbb{F}|^r}$. Now we define $\delta = (L_i[M] - L_i[M'] + L_i((0, \ldots, 0, \delta_1))_i$. For this choice of a, a' and δ the result holds. This completes the proof. □

Corollary 4. *The construction* EHC *is optimum when a MDS error correcting code is used. Thus the specific instantiations of* EHC*, given in Sect. 4.3, is optimum for d-block hash outputs,* $2 \leq d \leq 4$.

6 Conclusion and Research Problem

We already know that there is a close connection between error correcting code and universal hash. Here we apply error correcting code and Vandermonde matrix to construct a multi-block universal hash which require minimum number of multiplication. The minimum is guaranteed by showing a lower bound on the number of multiplication required. Previously in different context the lower bound on the number of multiplication has been considered. In this paper for the first time we study "concrete lower bound" (in terms of order a lower bound was known) for universal hash function. Similar lower bound was known for computations of polynomial of specific forms. See Appendix for a brief survey on it. However, we would like to note that those results can not be directly applicable as the contexts are differ ent.

To get a lower bound we take the popular algebraic computation model in which the time of multiplications are separated. We try to equate all the linear functions which are multiplied. Our construction performs better than Toeplitz construction in terms of number of multiplication.

This paper studies the relationship between complexity and security of universal hash. There are some relationship known for complexity and key-size however the picture is incomplete. Moreover, nothing is known involving these three important parameters: (i) security level, (ii) complexity, and (iii) key size. This could be possible future research direction in this topic. Our construction optimizes d block hash output for sum hash functions. It would be interesting to see how one adopts this for multi block polynomial hash using few keys.

In the view of the performance, the ongoing future research of us is to have a lightweight implementation of the universal hash function.

Acknowledgement. This work was supported by the Centre of Excellence in Cryptology, Indian Statistical Institute, Kolkata. The author would like to thank the anonymous reviewers for their useful comments. The author would like to thank Professor Palash Sarkar for his motivating discussion.

References

1. Belaga, G.E.: Some problems in the computation of polynomials. Doklady Akademii Nauk SSSR **123**, 775–777 (1958). ISSN 00023264. Citations in this document: §6
2. Bellare, M.: New proofs for NMAC and HMAC: security without collision-resistance. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 602–619. Springer, Heidelberg (2006)
3. Bierbrauer, J., Johansson, T., Kabatianskii, G.A., Smeets, B.J.M.: On families of hash functions via geometric codes and concatenation. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 331–342. Springer, Heidelberg (1994)
4. Black, J., Halevi, S., Krawczyk, H., Krovetz, T., Rogaway, P.: UMAC: fast and secure message authentication. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 216–233. Springer, Heidelberg (1999). Citations in this document: §1, §1.1
5. Blaum, M., Bruck, J., Vardy, A.: MDS array codes with independent parity symbols. IEEE Trans. Inf. Theor. **42**(2), 529–542 (1996). Citations in this document: §4.1
6. Bernstein, D.J.: The Poly1305-AES message-authentication code. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 32–49. Springer, Heidelberg (2005). Citations in this document: §1, §3.1
7. Bernstein, D.J.: Polynomial evaluation and message authentication. URL: http://cr.yp.to/papers.html#pema. ID b1ef3f2d385a926123e 1517392e20f8c (2007)
8. Boesgaard, M., Christensen, T., Zenner, E.: Badger – a fast and provably secure MAC. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 176–191. Springer, Heidelberg (2005)
9. den Boer, B.: A simple and key-economical unconditional authentication scheme. J. Comput. Secur. **2**, 65–71 (1993)
10. Carter, L., Wegman, N.M.: Universal classes of hash functions. J. Comput. Syst. Sci. **18**(2), 143–154 (1979)
11. Etzel, M., Patel, S., Ramzan, Z.: Square hash: fast message authentication via optimized universal hash functions. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 234–251. Springer, Heidelberg (1999). Citations in this document: §2
12. Eve, J.: The evaluation of polynomials. Numer. Math. **6**, 17–21 (1964). Citations in this document: §6
13. Gilbert, N.E., MacWilliams, F.J., Neil, J.A.: Sloane, Codes which detect deception. Bell Syst. Tech. J. **53**, 405–424 (1974)
14. Halevi, S., Krawczyk, H.: MMH: software message authentication in the Gbit/second rates. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 172–189. Springer, Heidelberg (1997). Citations in this document: §1.1, §1.1, §3.1, §3.3
15. Hastad, J., Impagliazzo, R., Levin, L.A., Luby, M.: Construction of pseudorandom generators from any one-way function. SIAM J. Comput. **28**(4), 1364–1396 (1999)
16. Horner, W.G.: Philosophical transactions. Roy. Soc. Lond. **109**, 308–335 (1819). Citations in this document: §6

17. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with constant computational overhead. STOC **2008**, 433–442 (2008). Citations in this document: §1.1, §1.1
18. Johansson, T.: Bucket hashing with a small key size. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 149–162. Springer, Heidelberg (1997)
19. Kaps, J., Yüksel, K., Sunar, B.: Energy scalable universal hashing. IEEE Trans. Comput. **54**(12), 1484–1495 (2005)
20. Kohno, T., Viega, J., Whiting, D.: CWC: a high-performance conventional authenticated encryption mode. In: Roy, B., Meier, W. (eds.) FSE 2004. LNCS, vol. 3017, pp. 408–426. Springer, Heidelberg (2004)
21. Krovetz, T.: Message authentication on 64-bit architectures. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 327–341. Springer, Heidelberg (2007)
22. McGrew, D.A., Viega, J.: The security and performance of the Galois/Counter mode (GCM) of operation. In: Canteaut, A., Viswanathan, K. (eds.) INDOCRYPT 2004. LNCS, vol. 3348, pp. 343–355. Springer, Heidelberg (2004). Citations in this document: §3.1
23. Krawczyk, H.: LFSR-based hashing and authentication. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 129–139. Springer, Heidelberg (1994). Citations in this document: §2, §3.5
24. Knuth, D.: The Art of Programming. Addison-Wesley, Boston (1969). Citations in this document: §1, §6
25. Krovetz, T., Rogaway, P.: Fast universal hashing with small keys and no preprocessing: the PolyR construction. In: Won, D. (ed.) ICISC 2000. LNCS, vol. 2015, pp. 73–89. Springer, Heidelberg (2001). Citations in this document: §3.1
26. Mansour, Y., Nissan, N., Tiwari, P.: The computational complexity of universal hashing. In: Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing (1990), pp. 235–243. ACM Press (1990) Citations in this document: §1.1
27. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error-Correcting Codes. Elsevier, North-Holland (1977). Citations in this document: §1.1
28. Mehlhorn, K.: On the program size of perfect and universal hash functions. In: FOCS, pp. 170–175 (1982)
29. Motzkin, T.S.: Evaluation of polynomials and evaluation of rational functions. Bull Xmer. Math SOC. **61**, 163 (1955). Citations in this document: §6, §6
30. Nevelsteen, W., Preneel, B.: Software performance of universal hash functions. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, p. 24. Springer, Heidelberg (1999)
31. Nguyen, L.H., Roscoe, A.W.: Short-output universal hash functions and their use in fast and secure data authentication. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 326–345. Springer, Heidelberg (2012). Citations in this document: §3.5
32. Nisan, N., Zuckerman, D.: Randomness is linear in space. J. Comput. Syst. Sci. **52**(1), 43–53 (1996)
33. Ostrowski, A.M.: On two problems in abstract algebra connected with Homers rule. Studies Presented to R. von Mises. Academic Press, New York (1954). Citations in this document: §6
34. Ya, V.: Pan, Methods of computing values of polynomials. Russ. Math. Surv. **21**, 105–136 (1966). Citations in this document: §6, §6, §6
35. Paterson, M.S., Stockmeyer, L.J.: On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM J. Comput. **2**(1), 60–66 (1973). Citations in this document: §6, §6

36. Rabin, M.O., Winograd, S.: Fast evaluation of polynomials by rational preparation. Commun. Pure Appl. Math. **XXV**, 433–458 (1972). Citations in this document: §6
37. Rogaway, P.: Bucket hashing and its application to fast message authentication. J. Cryptology. **12**(2), 91–115 (1999). Citations in this document: §2, §3.5
38. Sarkar, P.: A new multi-linear universal hash family. Des. Codes Crypt. **69**, 1–17 (2012). Springer. Citations in this document: §3.5, §3.5
39. Sarkar, P.: A trade-off between collision probability and key size in universal hashing using polynomials. Des. Codes Crypt. **3**, 271–278 (2011)
40. Shoup, V.: On fast and provably secure message authentication based on universal hashing. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 313–328. Springer, Heidelberg (1996). Citations in this document: §3.2, §3.5
41. Spielman, D.A.: Linear-time encodable and decodable error-correcting codes. STOC 1995 **4**(4), 388–397 (1995). Citations in this document: §1.1
42. Stinson, D.: Universal hashing and authentication codes. Des. Codes Crypt. **4**(4), 369–380 (1994)
43. Stinson, D.R.: On the connections between universal hashing, combinatorial designs and error-correcting codes. Proc. Congressus Numerantium **114**, 7–27 (1996). Citations in this document: §2, §2, §7
44. Stinson, D.R.: Universal Hash Families and the Leftover Hash Lemma, and Applications to Cryptography and Computing. Research report (University of Waterloo. Faculty of Mathematics) (2001). http://books.google.co.in/books?id=edldNAEACAAJ Citations in this document: §1
45. Taylor, R.: An integrity check value algorithm for stream ciphers. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 40–48. Springer, Heidelberg (1994)
46. Winograd, S.: A new algorithm for inner product. IEEE Trans. Comput. **17**, 693–694 (1968). Citations in this document: §1.1, §3.3
47. Winograd, S.: On the number of multiplications necessary to compute certain functions. Commun. Pure Appl. Math. **XXIII**, 165–179 (1970). Citations in this document: §1, §6
48. Winograd, S.: On the number of multiplications required to compute certain functions. Proc. National Acad. Sci. USA **58**(5), 1840–1842 (1967)
49. Wegman, M., Carter, L.: New hash functions and their use in authentication and set equality. J. of Comp. Syst. Sci. **1981**(22), 265–279 (1981)

Appendix: Brief Survey on the Computation of $a_0 + a_1 x + \ldots + a_n x^n$

We provide a brief survey on the lower bound of multiplications for computing a polynomial. Note that our interest in this paper is to provide a lower bound on number of multiplications for computing a multi variate polynomial which is an universal hash. Even though these two issues are very much related (some of the ideas in proving results are also similar), some immediate differences can be noted. For example, the existing bounds depend on the degree of the polynomials whereas we provide bound on the number of message blocks (degree could be arbitrarily higher). The existing works consider multivariate polynomials which has a special form: $P(a_0, \ldots, a_n, x_1, \ldots, x_m) := a_0 + \sum_{i=1}^{n} a_i \cdot \Phi_i(x_1, x_2, \ldots, x_m)$ where

Φ_i's are rational functions of $x_1, \ldots, x_m$. For an universal hash, the lower bound of our paper works for any multivariate polynomial (or even rational functions).

The function x^n can be computed in at most $2\lceil \log_2 n \rceil$ multiplications by using well known "square and multiply" algorithm. One can also compute $1 + x + \ldots + x^{n-1}$ using at most $2\lceil \log_2 n \rceil$ multiplications, one division and two subtractions since it is same as $\frac{x^n-1}{x-1}$ whenever $x \neq 1$. These are some simple examples of polynomials and there are some specific methods to simplify some polynomials. How does one can compute "*generically*" an arbitrary polynomial $f(x) = a_0 + a_1 x + \ldots + a_n x^n$, $a_i \in R$ (an underlying ring or field), of degree n with minimal number of operations, mainly multiplication and division? By generically we mean an algorithm which takes any a_i's and x as its inputs and computes the polynomial $f(x)$ (similar to an algorithm in uniform model). We know Horner's rule [16][4] to compute $f(x)$ in n multiplications and n additions.

§Minimum Number of Multiplications. Can we do better than n multiplications for computing an arbitrary polynomial? Or, can we prove that there are some polynomials for which n multiplications and division are necessary? The above question regarding the minimum number of multiplications to compute a given polynomial of small degree, was first investigated by Ostrowski [33]. He showed that at least n multiplications are required to evaluate a polynomial $f(x)$ of degree n for $1 \leq n \leq 4$. The results were further proved for any positive integer n by Pan [34] and a more general statement by Winograd [47]. Moreover, even if divisions are allowed, at least n multiplications/divisions are necessary to evaluate it. Belega [1] moreover proved that at least n additions or subtractions are required to compute f.

§General statement. The general statement by Winograd gives a lower bound for computation of any multivariate polynomial of the form

$$P(a_0, \ldots, a_n, x_1, \ldots, x_m) := a_0 + \sum_{i=1}^{n} a_i \cdot \Phi_i(x_1, x_2, \ldots, x_m)$$

where Φ_i's are rational functions of $x_1, \ldots, x_m$. If the rank (the maximum number of linear independent elements) of the set $S = \{1, \Phi_1, \ldots, \Phi_n\}$ is $u + 1$ then at least u multiplication and division are necessary. In particular, when $m = 1$, $\Phi_i(x_1) = x_1^i$ we have $P = f(x_1)$ and $u = n$. Thus, the result of Pan [34] is a simple corollary of it. When $m = n$, $\Phi_i(x_1, \ldots, x_n) = x_i$ and $a_0 = 0$ we have the classical dot-product $a_1 \cdot x_1 + \ldots + a_n \cdot x_n$ and the rank is again $n + 1$. So it also proves that to compute the dot-product we need at least n multiplications.

§Evaluation of a given Polynomial with Preprocessing. In the above results all types of multiplications are counted. More formally, the computation of the multivariate polynomial $F(a_0, \ldots, a_n, x) = a_0 + a_1 x + \ldots + a_n x^n$ have been considered in which coefficients are treated as variables or inputs of algorithms.

[4] Around 1669, Isaac Newton used the same idea which was later known as Newton's method of root finding (see 4.6.4, page 486 of [24]).

One of the main motivations of the above issue is to evaluate approximation polynomials of some non-algebraic functions, such as trigonometric functions. As the polynomials (i.e., a_i's) are known before hand, one can do some preprocessing or adaptation on coefficients to reduce some multiplications. To capture this notion, one can still consider the computation of F but the operations involving only a_i's are said to be the preprocessing of a_i's. Knuth [24] (see Theorem E, 4.6.4), Eve [12], Motzkin [29] and Pan [34] provide methods for F requiring $\lceil \frac{n}{2} \rceil$ multiplications ignoring the cost of preprocessing. However, these require preprocessing of *finding roots of higher degree equations* which involves a lot of computation and may not be exact due to numerical approximation. However, it is an one-time cost and is based on only coefficients. Later on, whenever we want to compute the polynomial for a given x, it can be computed faster requiring about $\lceil \frac{n}{2} \rceil$ multiplications. Rabin-Winograd [36] and Paterson-Stockmeyer [35] provide methods which require *rational preprocessing on coefficients (i.e., computing rational functions of coefficients only)* and afterwards about $\frac{n}{2} + O(\log n)$ multiplications for a given x.

§Minimum Number of Multiplications after Preprocessing. We have already seen that total n multiplication is necessary to compute F generically and Horner's rule is one algorithm which shows the tightness of the lower bound. Similarly, with preprocessing, $\lceil n/2 \rceil$ **multiplications for computing the multivariate polynomial F has been proved to be optimum** by Motzkin [29] and later on a more general statement by Winograd [47,48]. The bound $\lceil n/2 \rceil$ does not work for computing a known polynomial f since multiplication by constant could be replaced by addition, e.g. in $\mathbb{Z}$, $a_i \cdot x = x + \ldots + x$ (a_i times). In fact, Paterson and Stockmeyer [35] provided **methods which require about $O(\sqrt{n})$ multiplications** and **showed the bound is optimum**. Note that this method does not compute the polynomial generically which means that for every polynomial $f(x) = a_0 + a_1 x + \ldots + a_n x^n$ there is an algorithm $C_{a_0,\ldots,a_n}$ depending on the coefficients which computes $f(x)$ given x in $O(\sqrt{n})$ multiplications. This result and those by [29,36,47,48] (one algorithm works for F, i.e. for all polynomials f) can be compared with non-uniform and uniform complexity of Turing machine respectively. This justifies two different bounds of computation of a polynomial.