On the Minimum Number of Multiplications Necessary for Universal Hash Constructions

Mridul Nandi

Cryptology Research Group
Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road, Kolkata, India 700108
mridul.nandi@gmail.com,

Abstract. Universal hashes are usually based on some multivariate polynomials in message and key blocks (elements of some underlying ring R). These are implemented by using multiplications (which dominates the computational time) and additions. Two such hashes are pseudo dot-product (PDP) hash and Bernstein-Rabin-Winograd (BRW) hash which require $n/2$ multiplications for n message blocks. In this paper we observe that these are optimum in number of multiplications by showing that *at least $n/2$ multiplications or non-linear operations are necessary.* We also extend this lower bound for any multi-block hash construction, i.e., the hash output is an element of R^d. *We show that d block hash outputs requires at least $(d-1)+n/2$ non-linear operations.* The widely used Toeplitz construction for d block hash output requires $nd/2$ multiplications when it is applied for PDP. In this paper, we propose a *d-block universal hash EHC requiring $(d-1)+n/2$ multiplications and hence it is optimum and the bound is tight.* Our construction is roughly d times faster than Toeplitz construction. Moreover, it has similar parllelizibility and key size as in Toeplitz construction.

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, 41, 42] are used as building blocks of several cryptographic constructions, e.g., *message authentication codes* [10, 48], domain extension of pseudorandom functions [2, 4], extractors [15, 32] and quasi-randomness [43]. It also has close connection with *error correcting codes* and other combinatorial objects [13, 42].

Informally, a universal hash function h takes two inputs a key k and a message m, 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 pair of distinct messages the collision (or differential) probability is small when the key is chosen uniformly from its key space. A very popular application of universal hash is to obtain a domain extension of pseudorandom function or PRF. When h has low collision

probability and f is a PRF, then the composition function $f \circ h$ is a PRF [4]. Thus h behaves as a preprocessor to reduce the problem of designing arbitrary size input PRF to a fixed small size input PRF. This method is only useful when we use a much faster h (than a PRF). So our main question of this paper is that how fast a universal hash function could be? It has been answered [26, 28] in terms of order by showing asymptotic lower bounds in circuit level computational models. As these bounds ignore constant factor, there is no way to know which hash functions are optimum. Moreover, almost all known hash functions achieve these bounds. Hence our target question of this paper is to **obtain a concrete lower bound** and to show the **tightness of the bound by producing examples**. To answer it, we consider the *algebraic computation model* (see Knuth [24] Volume 2) 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 in a sequence determined by the algorithm. As multiplication is much more costly operation than addition, we are more interested in *multiplication complexity*, the minimum number of multiplications required to compute hash functions. Note that almost all known universal hashes are based on some polynomials over a ring or field R and hence their multiplication complexity could be compared. This would essentially compare the actual time taken to compute hash values provided the other overheads such as addition is not much.

1.1 Our contribution and outline of the paper

In the following we assume a universal hash function hashes all messages from R^n and hence multiplication complexity is measured in terms of n.

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** $n/2$ (see Theorem 4). We show it by proving the other way, i.e., if a function has multiplication complexity less than $n/2$ then there are two distinct messages from R^n whose differential probability is one. The pseudo dot-product [45] based hash PDP (e.g. NMH hash [14] and its efficient variants NMH* [14], NH [4] and others [8, 21]) and Bernstein-Rabin-Winograd or BRW hash [7, 36] are two types of examples which achieve this bound. Even though our lower bound is intuitive, to the best of our knowledge, it was not known before.

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). In Theorem 6, **we prove that to obtain** d **block hash outputs** (the hash output is now an element of R^d) **we need at least** $(d-1)+n/2$ **multiplications**. Surprisingly, no construction is known so far achieving this lower bound for $d > 1$. Note that both Toeplitz and independent invocation applied to the PDP requires $nd/2$ multiplications. So there is a scope of improving hash construction. In this paper, **we provide a** d**-block** Δ**-universal hash, called EHC or encode-hash-combiner** (see Algorithm

1), using an error correcting code [27] and a linear combiner based on the Vandermonde matrix, which requires exactly $(d-1)+n/2$ 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 basic idea of EHC is described in [17]. However, they used the error correcting code for both encoding and combiner layers which are not practically implementable. The main reason is due to linear space complexity of their constructions. However, our construction requires constant time or memory proportional to the hash size.

Outline of the paper After providing all basic necessary definitions in section 2, we present a survey on existing constructions and analysis methods in section 3. We provide our proposed Δ universal hash functions in section 4. In section 5, we present a brief survey on results of lower bound on the number of multiplications for a polynomial. Later in the same section, we provide the lower bounds on the the number of non-linear operations for a universal hash function. Finally, we conclude with possible future research directions.

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 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-ΔU (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 [42]) and ϵ-balanced function (easy to check from definition) but not necessarily for an ϵ-U hash function [42]. 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

Convert ΔU to an universal hash function. Let h be an ϵ-ΔU $(\mathcal{K}, D, R)$-hash function. Then h' defined below is an ϵ-universal hash on a domain $D \times R$.

$$h'_k(m, z) = h_k(m) + z, \quad m \in D, z \in R.$$

For example, when R is a field of size q with addition "+" and multiplication "$\cdot$", it is easy to see that $m_1 \mapsto \mathbf{K} \cdot m_1$ is a q^{-1}-ΔU hash function and hence $(m_1, m_2) \mapsto m_1 + \mathbf{K} \cdot m_2$ is a q^{-1}-U hash function where $m_1, m_2, \mathbf{K} \in R$.

Multi-linear hash. The hash mapping $(m_1, \ldots, m_l) \mapsto m_1 \cdot \mathbf{K}_1 + \cdots + m_l \cdot \mathbf{K}_l$ is an q^{-1}-ΔU hash function (it is a sum hash and hence one may apply Lemma 2). It is known as *multi-linear hash* ML[13, 48]. Hence $m_1 \cdot \mathbf{K}_1 + \cdots + m_l \cdot \mathbf{K}_l + m_{l+1}$ is an q^{-1}-ΔU hash function. Later MMH was proposed [14] with a performance record. It is a multi-linear hash with a specific choice of $R = \mathbb{Z}_{2^{64}}$ and a post-processor. All these constructions above requires (at least) n multiplications and n many independent key blocks.

(Linear) Poly-hash. By counting roots of a polynomial, one can show that $(m_1, \ldots, m_l) \mapsto m_1 \cdot \mathbf{K} + m_2 \cdot \mathbf{K}^2 + \cdots + m_l \cdot \mathbf{K}^l$ is an $l \times q^{-1}$-ΔU hash function and hence $m_0 + m_1 \cdot \mathbf{K} + m_2 \cdot \mathbf{K}^2 + \cdots + m_l \cdot \mathbf{K}^l$ is an $l \times q^{-1}$-U hash function. These are known as poly-hash [3, 9, 44] which mainly differ by padding rule and choices of R. Padding rule is crucial when we hash variable size messages. We call it linear poly hash as it is linear in message blocks. For example, Ghash used in GCM [22], poly1305 [6], polyQ, polyR [25] (combination of two poly-hashes), and others [20, 18] 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 $n/2$ multiplication and one key. As the algorithm is recursive and binary tree based, it requires $O(\log n)$ storage. This construction uses minimum number of keys (single block) and requires minimum number of multiplications (as we show in Theorem **??**).

3.1 Composition of universal hashes

The conversion for the other direction costs an independent invocation of ΔU-hash function. More precisely, given an ϵ_1-universal $(\mathcal{K}, D, D')$-hash function h and ϵ_2-ΔU $(\mathcal{K}', D', R)$-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. This can be shown in two cases: (i) The collision of $h' \circ h$ implies collision of h or h' (with different inputs) and hence we are done as these two functions have independent keys. (ii) The non-zero differential of $h' \circ h$ implies non-collision of h and differential of h' which can happen with probability at most ϵ_2.

Whenever h' is assumed to be only ϵ_2-U hash function, the composition is $(\epsilon_1 + \epsilon_2)$-U-hash function [40]. One can inductively extend it for a composition of any number of universal hash functions $h_r \circ \cdots \circ h_1$ as long as their domain and output spaces are compatible, i.e., composition is well defined. Whenever the last hash function in the composition is universal (or ΔU) so is the composed hash function. This composition results are useful to play with domain and range for different choices and has been used in several constructions [4, 39, 25].

3.2 Pseudo dot-product

The notion of pseudo dot product hash is introduced for preprocessing some cost in matrix multiplications [45]. 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 [19, 21, 8, 31]. A general form of pseudo dot-product PDP is $(m_1 + \mathbf{K}_2)(m_2 + \mathbf{K}_1) + \ldots + (m_{2l-1} + \mathbf{K}_{2l})(m_{2l} + \mathbf{K}_{2l-1})$ which is same as multi-linear hash plus two functions of messages and keys separately. The main advantage of PDP is that it requires l multiplications to hash $2l$ message blocks. We first prove a general statement which is used to prove ΔU propert of PDP.

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

Proof. For any $m \neq m'$ and δ, $h'_k(m) - h'_k(m') = \delta$ implies that $h_k(m) - h_k(m') = \delta' := \delta + g(m') - g(m)$ and hence the result follows. □

Corollary 1 (Pseudo dot-product hash). *Let R be a field of size q. The hash function $(m_1, m_2, \ldots, m_{2l-1}, m_{2l}) \mapsto (m_1 + \mathbf{K}_1)(m_2 + \mathbf{K}_2) + \ldots + (m_{2l-1} + \mathbf{K}_{2l-1})(m_{2l} + \mathbf{K}_{2l})$ is q^{-1}-ΔU hash function for a fixed l.*

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

3.3 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 with key space $\mathcal{K}$. A hash function is called **sum hash** (based on h), denoted h^{sum} if it is defined as

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

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 with key space $\mathcal{K}$ then h^{sum} is an ϵ-ΔU $(\mathcal{K}^s, D^s, R)$-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'). □

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 $(\mathcal{K}, D, R)$-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 $h^{\mathsf{pad},\text{sum}}$, defined below, is an ϵ-ΔU $(\mathcal{K}^L, \{0,1\}^{\leq t}, R)$-hash function.*

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

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 follows by using balanced property of the hash. 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 (the 0 message maps to 0 with probability one). However, it is q^{-1}-balanced for $R \setminus \{0\}$. 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. Surprisingly, the pseudo dot-product is $2q^{-1}$-balanced on R and hence any injective padding rule for PDP will give a $2q^{-1}$-ΔU hash function.

An Alternate Method A generic way to handle arbitrary length is as follows: Let h be an ϵ-ΔU hash function on D_i, $1 \leq i \leq R$ and h' be an ϵ-ΔU hash function on $\{0,1\}^r$. Suppose $Len : \{1, 2, \ldots, R\} \rightarrow \{0,1\}^r$ is an injective function, such as lenght encoding function. Then the hash function $H_{k,k'}(m) = h_k(m) + h'_{k'}(l_m)$ where $m \in D_i$ and $l_m = Len(i)$ is an an ϵ-ΔU hash function on $D := \cup_i D_i$. In other words, h is an ΔU hash functions for fixed length messages and h' is an ΔU hash function defined on length then the sum of these two functions is an ΔU hash function on variable length input.

3.4 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 $\mathbf{K}_1, \ldots, \mathbf{K}_d$, we define a d-block hash as $h^{(d)} = (h_{\mathbf{K}_1}(m), \ldots, h_{\mathbf{K}_d}(m))$. 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.

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

We define $h^{T,d}_{\mathbf{K}_1,\ldots,\mathbf{K}_{l+d-1}}(m) = (h_1^{T,d}, \ldots, h_d^{T,d})$. 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 $m_1 = m'_1, \ldots, m_{i-1} = m'_{i-1}$. Now condition all keys except $\mathbf{K}' := (\mathbf{K}_i, \ldots, \mathbf{K}_{i+d-1})$. Denote H_i and H'_i for the ith 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 $\overline{D} := \{0,1\}^{\le t}$). *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 $(\mathcal{K}^{L+d-1}, \{0,1\}^{\le t}, R_1^d)$-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. Other popular multi-block constructions are the following:

> 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 examples of multi-block hash.

Discussion: Toeplitz Construction applied to Multi-linear hash. Even though Toeplitz Construction on $\mathbb{F}_{2^n}$ requires few additional key, the number of multiplications is same as that of d independent invocations. More precisely, we need sd multiplications to hash s blocks of size n. Alternatively, one can use a larger field $\mathbb{F}_{2^{nd}}$ to get the same hash size which requires s/d multiplications (note that the number of block is now reduced to s/d as the block size becomes nd) of the field $\mathbb{F}_{2^{nd}}$. So informally speaking Toeplitz construction can give better performance than the later if $\mathbb{F}_{2^{nd}}$ multiplication costs more than d^2 times $\mathbb{F}_{2^n}$ multiplication. However, it is not true as one can always apply the Karatsuba multiplier to have faster implementation which requires more area in hardware. On the other hand, a simple implementation by a primitive element cost d^2 times $\mathbb{F}_{2^n}$ multiplications and it requires the same and similar parallelizibility as the Toeplitz version. Hence, we do not find any clear advantage of Toeplitz construction. Moreover, Toeplitz version requires little bit extra key. The similar discussion is valid when Toeplitz is applied to pseudo dot-product.

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 $d_{\text{ham}}(x, y) = |\{i : x_i \neq y_i\}|$, 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 $m \le n$. We define $d^*_{\text{ham}}(x, y) = (n - m) + d_{\text{ham}}(x', y)$ where $x' = (x_1, \ldots, x_m)$. The minimum distance for an encoding function $e : D \to A^+ := \cup_{i \ge 1} A^i$ is defined as

$$d(e) \stackrel{\Delta}{=} \min_{M \neq M' \in D} d^*_{\text{ham}}(e(M), e(M'))$$

We know from coding theory that for any coding $e : \mathbb{F}^k \to \mathbb{F}^n$ we have $d(e) \leq n-k+1$ (**singleton bound**). Moreover, there is a linear code[1] e, called MDS or **maximum distance separable** code, such that $d(e) = n-k+1$. 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 $1 \leq d \leq \min\{k, n-k\}$. 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 $d \leq n$ 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 $r' \leq d$. 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 e be an error correcting code from a message space D to $A^{\leq L}$ with the minimum distance d. For each $l \geq d$, let $V_{d,l}$ be a d-MDS matrix of dimension $d \times l$ whose entries are from R_1, the underlying ring. Let h be an ϵ-ΔU and ϵ-balanced hash function on A with the output space 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 $(m_1, \ldots, m_l)$ where l depends on m. In this process we ensure at least d places would differ for two distinct messages. Then we apply the hash h block-wise. Finally, we make a linear combiner on the hash blocks to obtain d-block hash output. We call this general construction method EHC or encode-hash-combiner. The description of it is given in Algorithm 1. The basic idea of the construction is described in [17]. However, they used the error correcting code for both encoding and combiner layers which are not practically implementable. The main reason is due to linear space complexity of their constructions. However, our construction requires constant time or memory proportional to the hash size.

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

Input: $m \in D$
Output: $h \in R_1^d$
Key: $k_1, \ldots, k_L$

Algorithm EHC(m)

$e(m) = (m_1, \ldots, m_l) \in D^l$. \\ 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 and a family of d-MDS matrix $V_{d\times l}$, $l \geq d$.

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.*

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.o.l.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 $V \cdot (a_{\mathbf{K}_{i_1}}, \ldots, a_{\mathbf{K}_{i_1}})^{tr} = \delta$ where $a_{\mathbf{K}_{i_j}} = h_{\mathbf{K}_{i_j}}(m_{i_j}) - h_{\mathbf{K}_{i_j}}(m'_{i_j})$ if $i_j \leq l'$, otherwise, $a_{\mathbf{K}_{i_j}} = h_{\mathbf{K}_{i_j}}(m_{i_j})$. 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 $(a_{\mathbf{K}_{i_1}}, \ldots, a_{\mathbf{K}_{i_1}})^{tr} = V^{-1} \cdot \delta$. So the differential probability is at most ϵ^d. □

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. 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

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

Let $l = t + 3$. The base hash function $h_i(k, k', x, x') = (x \oplus k) \cdot (x' \oplus k')$. Finally the d-MDS matrix can be replaced by vandermonde matrix $V_{d,l} := V_d(1, \alpha, \alpha^2, \ldots, \alpha^l)$.

Theorem 2. *The coding C_4 defined above has minimum distance 4 for a fixed length encoded message.*

Proof. This coding has systemic form $(I : S)$ where $S = V_l(1, \alpha, \alpha^2)$. 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. One can check easily that for any two columns and two rows the sub-matrix is non-singular. 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 $\alpha^{i_1+i_2-2}(\alpha^{i_2} - \alpha^{i_1})$. □

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. We modify the definition to incorporate variable length. Note that we already have seen how to incorporate variable length by hashing length with an independent key. Now we provide a more efficient approach (for larger length inputs). We first note that if the longer message m has at least d blocks more than shorter one m' then clearly m is involved by at least d additional keys which are not required for m'. So conditioning on all other keys one can get desired bound.

Let $r \equiv l \mod d$ where $r < d$ where $e(m) = D^r$. Note that r can be expressed using $d' = \log_2 d$ bits. Let $\mathbf{K}^{(1)}, \ldots, \mathbf{K}^{(d')} \in \mathcal{K}$ be dedicated keys for length, i.e. not used to process message. We define the modified hash as follows:

$$\mathsf{ECH}^*(m) = \mathsf{ECH}(m) + (\mathbf{K}^1 \cdot d', \ldots, \mathbf{K}^{d'} \cdot d'),$$

To analyze it works, we consider three cases for $e(m) \in D^l$ and $e(m') \in D^{l'}$.

1. If $l = l'$ then the previous theorem for fixed length works.
2. If $l \geq l' + d$ then as we discussed above the differential probability will be low.
3. Otherwise, modulo d the reduced length r and r' will be different. So we can condition all keys except $\mathbf{K}^{(1)}, \ldots, \mathbf{K}^{(d')}$ and low differential probability follows from the fact that the product hash is ΔU.

Theorem 3. *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 Non-Linear Operations

Definition 3. *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 $m = (m_1, \ldots, m_l), k = (k_1, \ldots, k_s), v_1, \ldots, v_{i-1}$, $1 \leq i \leq t$ 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 $(L^1, \ldots, L^d)$. 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, $L = a_0 + L[x] + L[y]$.

Lemma 7. *[42] 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 and only 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 $V(K, x) - V(K, x') = \delta_1 \Leftrightarrow H(K, x) - H(K, x') = \delta$. 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 4. *Let* $H(K, m_1, \ldots, m_l)$ *be a MVP hash computable by using* s *multiplications with* $2s + 1 \le l$*. 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 above lemma the result follows. □

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

Remark 1.

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 5. *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*

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

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 6. *Let $H = (H_1, \ldots, H_d)$ be a vector of d polynomials in $m = (m_1, \ldots, m_l)$ 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 $1 \le i \le d$. 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 $2(s-r)+1$ variables 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 $i \le s - r$. Now consider the mapping $f : \mathbb{F}^k \to \mathbb{F}^{s-r}$ 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 $\mathsf{Pr}_K[f(y) = \delta_1] \ge \frac{1}{|\mathbb{F}|^r}$. Now we define $\delta = (L_i[M] - L_i[M'] + L_i(\delta_1))_i$. 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 section 4.3, is optimum for d-block hash outputs, $2 \le d \le 4$.*

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 different.

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.

References

[1] G. E. Belaga, *Some problems in the computation of polynomials*, Doklady Akademii Nauk SSSR, ISSN 00023264 **123** (1958), 775–777. Citations in this document: §6.

[2] M. Bellare, *New proofs for NMAC and HMAC: Security without collision-resistance*, Advances in Cryptology Crypto 2006, Lecture Notes in Computer Science, Springer-Verlag **4117** (2006), 602–619.

[3] J. Bierbrauer, T. Johansson, G. Kabatianskii, B. J. M. Smeets, *On families of hash functions via geometric codes and concatenation*, Advances in Cryptology Crypto 1993, Lecture Notes in Computer Science, Springer-Verlag **773**, 331–342.

[4] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, P. Rogaway, *UMAC: Fast and secure message authentication*, Advances in Cryptology Crypto 1999, Lecture Notes in Computer Science, Springer-Verlag **1666**, 216–233. Citations in this document: §1, §1.1.

[5] M. Blaum, J. Bruck, A. Vardy, *MDS Array Codes with Independent Parity Symbols*, IEEE Transaction on Information Theory **42 (2)** (1996), 529–542. Citations in this document: §4.1.

[6] D. J. Bernstein, *The Poly1305-AES message-authentication code*, FSE, Lecture Notes in Computer Science, Springer **3557** (2005), 32–49. Citations in this document: §3.

[7] D. J. Bernstein, *Polynomial evaluation and message authentication*, URL: http://cr. yp. to/papers. html# pema. ID b1ef3f2d385a926123e 1517392e20f8c (2007).
[8] M. Boesgaard, O. Scavenius, T. Pedersen, T. Christensen, E. Zenner, *Badgera fast and provably secure MAC*, Applied cryptography and network security (2005), 176–191.
[9] B. den Boer, *A simple and key-economical unconditional authentication scheme*, Journal of Computer Security **2** (1993), 65–71.
[10] L. Carter, N. M. Wegman, *Universal classes of hash functions.*, J. Comput. Syst. Sci. **18(2)** (1979), 143–154.
[11] M. Etzel, S. Patel, Z. Ramzan, *Square hash: Fast message authentication via optimized universal hash functions*, Advances in Cryptology Crypto 1999, Lecture Notes in Computer Science, Springer-Verlag **1666** (1999), 234–251. Citations in this document: §2.
[12] J. Eve, *The evaluation of polynomials*, Numer. Math. **6** (1964), 17–21. Citations in this document: §6.
[13] N. E. Gilbert, F. J. MacWilliams, J. A. Neil Sloane, *Codes which detect deception.*, Bell System Technical Journal, **53** (1974), 405–424.
[14] S. Halevi, H. Krawczyk, *MMH: Software message authentication in the Gbit/second rates*, In Proceedings of the 4th Workshop on Fast Software Encryption (1997), Springer-Verlag **1267** (1997), 172–189. Citations in this document: §1.1, §1.1, §3, §3.2.
[15] J. Hastad, R. Impagliazzo, L. A. Levin, M. Luby, *Construction of pseudorandom generators from any one-way function*, SIAM Journal on Computing **28(4)** (1999), 1364–1396.
[16] W. G. Horner, , Philosophical Transactions, Royal Society of London **109** (1819), 308–335. Citations in this document: §6.
[17] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Amit Sahai, *Cryptography with Constant Computational Overhead*, STOC 2008 (2008), 433–442. Citations in this document: §1.1, §4.2.
[18] T. Johansson, *Bucket hashing with small key size*, Advances in Cryptology Eurocrypt 1997, Lecture Notes in Computer Science, Springer-Verlag **1233** (1997), 149–162.
[19] J. Kaps, K. Yüksel, B. Sunar, *Energy scalable universal hashing.*, IEEE Transaction in Computers, (2005), 14841495.
[20] T. Kohno, J. Viega, D. Whiting, *CWC: a high-performance conventional authenticated encryption mode*, FSE 2004, Lecture Notes in Computer Science, Springer-Verlag **3017** (2004), 408–426.
[21] T. Krovetz, *Message authentication on 64-bit architectures.*, Selected areas in cryptography, Lecture Notes in Computer Science, Springer Verlag **4356** (2007), 327–341.
[22] D. A. McGrew, J. Viega, *The security and performance of the Galois/counter mode of operation*, URL: http://eprint.iacr.org/2004/193/ (2004). Citations in this document: §3.
[23] H. Krawczyk, *LFSR-based Hashing and Authentication*, Advances in Cryptology Crypto 1994, Lecture Notes in Computer Science, Springer-Verlag **839** (1994), 129–139. Citations in this document: §2, §3.4.
[24] D. Knuth, *The Art of Programming*, Addison-Wesley Pub. Co. **Volume 2** (1969). Citations in this document: §1, §6.

[25] T. Krovetz, P. Rogaway, *Fast Universal Hashing with Small Keys and No Preprocessing: The PolyR Construction*, ICISC 2000, Lecture Notes in Computer Science, Springer-Verlag. **2015** (2000), 73 – 89. Citations in this document: §3.

[26] Y. Mansour, N. Nissan, P. Tiwari, *The computational complexity of universal hashing*, In Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing (1990), ACM Press, (1990), 235–243. Citations in this document: §1.1.

[27] F. J. MacWilliams, N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland (1977). Citations in this document: §1.1.

[28] K. Mehlhorn, *On the Program Size of Perfect and Universal Hash Functions*, FOCS (1982), 170–175.

[29] T. S. Motzkin, *Evaluation of polynomials and evaluation of rational functions*, Bull Xmer. Math, SOC. **61** (1955), 163. Citations in this document: §6, §6.

[30] W. Nevelsteen, B. Preneel, *Software performance of universal hash functions*, In Advances in Cryptology Eurocrypt 1999, Lecture Notes in Computer Science, Springer-Verlag **1592** (1999), 24–41.

[31] L. H. Nguyen, A. W. Roscoe, *Short-Output Universal Hash Functions and Their Use in Fast and Secure Data Authentication*, Fast Software Encryption 2012, Lecture Notes in Computer Science, Springer-Verlag **7549** (2012), 326–345. Citations in this document: §3.4.

[32] N. Nisan, D. Zuckerman, *Randomness is linear in space*, Journal of Computer and System Sciences **52(1)** (1996), 43–53.

[33] A. M. Ostrowski, *On two problems in abstract algebra connected with Homers rule*, Studies Presented to R. von Mises, Academic Press, New York (1954), 40–48. Citations in this document: §6.

[34] V. Ya. Pan, *Methods of computing values of polynomials*, Russian Math. Surveys, 21 (1966), 105–136. Citations in this document: §6, §6, §6.

[35] M. S. Paterson, L. J. Stockmeyer, *On the number of nonscalar multiplications necessary to evaluate polynomials*, SIAM Journal of Computing **2 (1)** (1973), 60–66. Citations in this document: §6, §6.

[36] M. O. Rabin, S. Winograd, *Fast Evaluation of Polynomials by Rational Preparation*, Communications on Pure and Applied Mathematics **XXV** (1972), 433–458. Citations in this document: §6.

[37] P. Rogaway, *Bucket Hashing and Its Application to Fast Message Authentication*, Journal of Cryptology. **12 (2)** (1999), 91–115. Citations in this document: §2, §3.4.

[38] P. Sarkar, *A new multi-linear universal hash family,*, Designs, Codes and Cryptography, Springer (2012), 1–17. Citations in this document: §3.4, §3.4.

[39] P. Sarkar, *A trade-off between collision probability and key size in universal hashing using polynomials*, Des. Codes Cryptography **3** (2011), 271–278.

[40] V. Shoup, *On fast and provably secure message authentication based on universal hashing*, In Advances in Cryptology Crypto 1996, Lecture Notes in Computer Science, Springer-Verlag **1109** (1996), 74–85. Citations in this document: §3.1, §3.4.

[41] D. Stinson, *Universal hashing and authentication codes*, Design Codes and Cryptography **4(4)** (1994), 369–380.

[42] D. R. Stinson, *On the Connections Between Universal Hashing, Combinatorial Designs and Error-Correcting Codes*, In Proc. Congressus Numerantium **114** (1996), 7–27. Citations in this document: §2, §2, §7.

[43] D. R. Stinson, *Universal Hash Families and the Leftover Hash Lemma, and Applications to Cryptography and Computing*, Research report (University of Waterloo. Faculty of Mathematics) **url=http://books.google.co.in/books?id=edldNAEACAAJ,** (2001). Citations in this document: §1.
[44] Richard Taylor, *An integrity check value algorithm for stream ciphers*, Advances in Cryptology Crypto 1993, Lecture Notes in Computer Science, Springer-Verlag (1993), 40–48.
[45] S. Winograd, *A new algorithm for inner product*, IEEE Transactions on Computers **17** (1968), 693–694. Citations in this document: §1.1, §3.2.
[46] S. Winograd, *On the Number of Multiplications Necessary to Compute Certain Functions*, Communications on Pure and Applied Mathematics **XXIII** (1970), 165–179. Citations in this document: §6.
[47] S. Winograd, *On the Number of Multiplications Required to Compute Certain Functions*, Proceedings of the National Academy of Sciences of the United States of America **58 (5)** (1967), 1840–1842.
[48] M. Wegman, L. Carter, *New hash functions and their use in authentication and set equality*, J. of Comp. and System Sciences (1981) **22** (1981), 265–279.

Appendix: Brief Survey on the Computation of $a_0 + a_1x + \ldots + a_nx^n$

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 $f(x) = a_0 + a_1x + \ldots + a_nx^n$, $a_i \in R$ (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][2] to compute $f(x)$ in n multiplications and n additions.

[2] 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])

§ 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 [46]. 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. 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 [46, 47]. 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, 46, 47] (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.