A New Universal Hash Function and Other Cryptographic Algorithms Suitable for Resource Constrained Devices

Palash Sarkar

Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road, Kolkata
India 700108.
email: palash@isical.ac.in

Abstract. A new multi-linear universal hash family is described. Messages are sequences over a finite field $\mathbb{F}_q$ while keys are sequences over an extension field $\mathbb{F}_{q^n}$. A linear map ψ from $\mathbb{F}_{q^n}$ to itself is used to compute the output digest. Of special interest is the case $q = 2$. For this case, we show that there is an efficient way to implement ψ using a tower field representation of $\mathbb{F}_{q^n}$. Such a ψ corresponds to a word oriented LFSR. We describe a method of combining the new universal hash function and a stream cipher with IV to obtain a MAC algorithm. Further, we extend the basic universal hash function to an invertible blockwise universal hash function. Following the Naor-Reingold approach, this is used to construct a tweakable enciphering scheme which uses a single layer of encryption and no finite field multiplications. From an efficiency viewpoint, the focus of all our constructions is small hardware and other resource constrained applications. For such platforms, our constructions compare favourably to previous work.

Keywords: universal hash function, word oriented LFSRs, message authentication codes, blockwise universality, tweakable enciphering schemes

1 Introduction

Universal hash functions are useful in different areas of computer science and are also important in cryptography. These were introduced by Carter and Wegman [6] and have been extensively studied since then. Among their many applications, one of the most important is the construction of message authentication code (MAC) algorithms. This technique was introduced by Wegman and Carter [28]. Previously, work on unconditionally secure authentication codes was done by Gilbert, MacWilliams and Sloane [10]. The approach of using universal hash function for constructing MAC algorithms was later investigated in [25, 5, 3].

1.1 Known Constructions

A well known construction[1] of a multi-linear universal hash function is the following. The message $\mathbf{M} = (M_1, \ldots, M_l)$ and key $\mathbf{K} = (K_1, \ldots, K_l)$ are sequences of elements over a prime field $\mathbb{F}$ and the digest is $K_1M_1 + \cdots + K_lM_l$ which is an element over $\mathbb{F}$. The probability (over random keys) that two distinct messages map to the same value is $1/|\mathbb{F}|$. The map $(M_1, \ldots, M_l) \mapsto K_1M_1 + \cdots + K_lM_l$ is a multi-linear map and we will call this the multi-linear hash function.

This basic idea has been studied and extended later. Halevi and Krawczyk [13] describe an efficient software implementation of this construction over $\mathbb{F}_q$ with $q = 2^{32} + 15$. They also modify the construction to align with 32-bit word boundaries and reduce the total number of modulo q

[1] Attributed to Carter and Wegman [6] in [13] and to Gilbert, MacWilliams and Sloane [10] in [3].

operations. The construction UMAC [5] develops upon [13] using a nonlinear function and is one of the fastest known constructions in software.

Shoup [25] reports implementation results for three kinds of functions over binary fields. The first kind is polynomial evaluation. The second kind is polynomial division due to Krawczyk [17] and is called "LFSR hashing" or cryptographic CRC. The third kind of hash function mentioned in [25] is a generalized division hash (GDH) of which the previous two kinds can be seen as special cases. An earlier work [16] had suggested the use of LFSRs in the construction of authentication codes which can also be seen as universal hash functions.

Polynomial evaluation based hash functions have also been proposed over non-binary fields. PolyR [18] is one such example. A recent proposal is Poly1305 [2], which chooses the prime carefully so as to ensure very fast implementation in software. For GDH, and polynomial evaluation based hash functions, the collision probability degrades with increase in message length.

There are other interesting constructions exploiting different combinatorial structures. See Stinson [26], Bierbrauer et al [4] and Rogaway [23].

1.2 Our Contributions

We generalize the multi-linear hash function mentioned above. The basic idea of the generalization is the following. We work with two fields, a base field $\mathbb{F}_q$ and an extension $\mathbb{F}_{q^n}$, where n is a positive integer. The message consists of elements from $\mathbb{F}_q$, whereas the key consists of elements of $\mathbb{F}_{q^n}$. A linear operator ψ from $\mathbb{F}_{q^n}$ to itself is used. The contribution of $l \leq n$ message elements $M_1, \ldots, M_l$ under a key $K \in \mathbb{F}_{q^n}$ is the sum

$$M_1 K + M_2 \psi(K) + \cdots + M_l \psi^{l-1}(K).$$

The output is an element of $\mathbb{F}_{q^n}$.

This basic idea is developed into a definition of a hash function family LH which is shown to be universal (in fact, XOR universal). We discuss possible choices of q, n and the operator ψ. For $q = 2$, one interesting option is to use a tower field representation of $\mathbb{F}_{2^n}$. The operator ψ can then be implemented very efficiently and corresponds to what is called a word-oriented LFSR in the stream cipher community.

We focus on the choice of $q = 2$ and the tower field representation of ψ. This leads to a universal hash function with a very small hardware footprint. The design is very simple to implement in software. Compared to previous proposals, the new construction produces the least (and optimal) size hash digest for achieving the same collision probability. There are several ways to extend our basic idea. Ideas based on decimation, the Toeplitz construction and multiple linear operators can be used.

Message authentication code. Generic techniques from [5] can be used to obtain a message authentication code (MAC) algorithm from the new universal hash function. Alternatively, we present a simple idea of combining the new hash function and a stream cipher with IV to obtain a MAC algorithm. Choosing a hardware efficient stream cipher (from one of the eSTREAM proposals) leads to a design of MAC algorithm with a very small hardware footprint.

Our technique for obtaining MAC from universal hash function and stream cipher with IV is generic. Consequently, it can also be used to combine software efficient stream ciphers with software efficient universal hash function (such as UMAC) to obtain a software efficient MAC.

Blockwise universal hashing and strong pseudorandom permutation. The universal hash family is used to construct an invertible and blockwise universal hash family. This kind of hash families can be used to construct (tweakable) strong pseudorandom permutation (SPRP). A technique due to Naor and Reingold [22] is used for this purpose. We follow this strategy and design a new SPRP called BEB. Compared to previous work, the unique feature of BEB is that it uses a single layer of encryption and does not use any finite field multiplications. All previous works either used two encryption layers [14, 15, 11] or used finite field multiplications [20, 27, 8, 12, 24]. Disk encryption is an important application of a tweakable SPRP. For this application, compared to previous constructions, BEB provides a design with a very small hardware footprint using a single encryption layer.

Focus of our work. The theoretical contribution of our work consists in designing a new universal hash function based on elementary finite field algebra. This is of some interest in its own right. For the practical aspect, we focus on the simplicity and small hardware size of the new algorithm.

In software, the currently fastest universal hash functions depend crucially on very fast multiplication modulo 2^{32}. This includes UMAC (and its predecessor MMH) and Poly1305. Fast multiplier circuits are present in large general purpose CPUs used in desktops, laptops and servers. As a result, UMAC and Poly1305 are extremely fast on such processors. Our proposal with $q = 2$ processes one bit at a time and even though it is quite simple, it is also slower than either of UMAC or Poly1305.

In practice, however, cryptographic algorithms are often required to be implemented on small CPUs with 8-bit arithmetic and small memory (128 bytes, for example); on small field programmable gate arrays; or as small application-specific integrated circuit. Using multiplication modulo 2^{32} does not seem to be the correct approach for such platforms. The bit-at-a-time approach is more suitable. For resource constrained devices, one would like an algorithm with a small hardware footprint. The new algorithms for the choice of $q = 2$ are ideal for such environments.

2 Linear Algebra Over Finite Fields

We need some elementary results on linear algebra over finite fields. The purpose of this section is to present the basic notation as well as the results that will be required.

Let q be a prime power and $\mathbb{F}_q$ be the finite field of q elements. For a positive integer n, the extension field $\mathbb{F}_{q^n}$ is a vector space over $\mathbb{F}_q$.

Let ψ be a linear transformation from $\mathbb{F}_{q^n}$ to itself. By ψ^i, we denote the usual iterate of ψ, i.e., $\psi^0 = \mathsf{id}$ and $\psi^i = \psi^{i-1} \circ \psi$, where id is the identity map from $\mathbb{F}_{q^n}$ to itself. Let $p(x) \in \mathbb{F}_q[x]$ be of the form $p(x) = a_m x^m + a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$, then by $p(\psi)$ we denote the linear operator $a_m\psi^m + a_{m-1}\psi^{m-1} + \cdots + a_1\psi + a_0\mathsf{id}$. We say that $p(x)$ annihilates ψ, if $p(\psi) = 0$, i.e., $p(\psi)$ maps all elements of $\mathbb{F}_{q^n}$ to zero. The minimum degree monic polynomial in $\mathbb{F}_q$ which annihilates ψ is said to be the minimal polynomial of ψ. The notion of minimal polynomial is relative to the base field. Suppose n_1 divides n. Then $\mathbb{F}_{q^{n_1}}$ is a subfield of $\mathbb{F}_{q^n}$ and the minimal polynomial of ψ over $\mathbb{F}_q$ is not the same as the minimal polynomial of ψ over $\mathbb{F}_{q^{n_1}}$.

If we fix a basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$, then a linear map ψ from $\mathbb{F}_{q^n}$ to itself is uniquely given by an $n \times n$ matrix A with entries from $\mathbb{F}_q$. The minimal polynomial $\tau(x)$ of ψ is also the minimal polynomial of A. Fixing a basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ allows the representation of the elements of $\mathbb{F}_{q^n}$ by n tuples over $\mathbb{F}_q$, so that, with respect to this basis, we can identify $\mathbb{F}_{q^n}$ with $\mathbb{F}_q^n$. For two vectors

$\mathbf{a}, \mathbf{b} \in \mathbb{F}_q^n$, $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the inner product, i.e., $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} a_i b_i$, where $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{b} = (b_1, \ldots, b_n)$.

We will require the following results about linear maps.

Lemma 1. *Let ψ be a linear map from $\mathbb{F}_{q^n}$ to itself such that its minimal polynomial $\tau(x)$ in $\mathbb{F}_q[x]$ is of degree n and is irreducible over $\mathbb{F}_q$. Let $p(x)$ be any polynomial in $\mathbb{F}_q[x]$. If $\tau(x)$ does not divide $p(x)$, then $p(\psi)$ is invertible. Equivalently, either $p(\psi) = 0$ or $p(\psi)$ is invertible.*

Proof: Fix a basis of $\mathbb{F}_{q^n}$ over $\mathbb{F}_q$ and let ψ be given by a matrix A over $\mathbb{F}_q$. Then the minimal polynomial of A is $\tau(x)$ and since $\tau(x)$ is of degree n, it is also the characteristic polynomial of A. Let the characteristic roots of A (over $\mathbb{F}_{q^n}$) be $\zeta_1, \ldots, \zeta_n$. Since $\tau(x)$ is irreducible, none of the ζ_is are zero.

The matrix $p(A)$ represents the linear map $p(\psi)$ with respect to the previously fixed basis. The distinct characteristic roots of $p(A)$ are $p(\zeta_1), \ldots, p(\zeta_n)$. If any of these values is 0, then from the irreducibility of $\tau(x)$, we get that $\tau(x)$ divides $p(x)$. Since $\tau(A) = 0$ (because $\tau(x)$ is the minimal polynomial of A), we have $p(A)$ also to be zero. On the other hand, if none of the $p(\zeta_1), \ldots, p(\zeta_n)$ are zero, the matrix $p(A)$ is non-singular and hence the transformation $p(\psi)$ is invertible. □

Lemma 2. *Let q be a prime power, $n = n_1 \times n_2$ and $q_1 = q^{n_1}$. Let $\rho(\alpha)$ be an irreducible polynomial of degree n_1 over $\mathbb{F}_q$ and $\mathbb{F}_{q^{n_1}} = \mathbb{F}_q[\alpha]/(\rho(\alpha))$. Further, let $\psi : \mathbb{F}_{q_1^{n_2}} \to \mathbb{F}_{q_1^{n_2}}$ be a linear map whose minimal polynomial, $\mu(x)$, over $\mathbb{F}_{q_1}$ is irreducible and of degree n_2. Then the minimal polynomial of ψ over $\mathbb{F}_q$ is of degree n and is irreducible over $\mathbb{F}_q$.*

Proof: Fix a basis of $\mathbb{F}_{q_1^{n_2}}$ over $\mathbb{F}_{q_1}$. Then the linear map ψ is given by an $n_2 \times n_2$ matrix A with entries from $\mathbb{F}_{q_1}$ and whose characteristic polynomial $\mu(x)$ is irreducible over $\mathbb{F}_{q_1}$.

Following [19], let $I(q, n; x)$ be the product of all irreducible polynomials in x of degree n over $\mathbb{F}_q$. Using [19, Theorem 3.31], we have $I(q^{n_1}, n_2; x) = I(q, n_1 n_2; x)$.

By definition, $\mu(x)$ divides $I(q^{n_1}, n_2; x)$. Also, by the Cayley-Hamilton theorem, $\mu(A) = 0$ and so $I(q, n_1 n_2; A) = I(q^{n_1}, n_2; A) = 0$. Now $I(q, n_1 n_2; x)$ is the product of all irreducible polynomials of degree $n = n_1 n_2$ over $\mathbb{F}_q$. By Lemma 1, we have that for any irreducible polynomial $P(x)$ of degree n over $\mathbb{F}_q$, either $P(A) = 0$ or $P(A)$ is invertible. If for all irreducible factors $P(x)$ of $I(q, n; x)$, we have $P(A)$ to be invertible, then we clearly cannot have $I(q, n; A)$ to be zero. Therefore, there must be some irreducible factor $\tau(x)$ of $I(q, n; x)$ such that $\tau(A) = 0$. Since this $\tau(x)$ is a factor of $I(q, n; x)$, it is irreducible and of degree n. This $\tau(x)$ is then the minimal polynomial of A over $\mathbb{F}_q$, which completes our proof. □

3 A Multi-Linear Universal Hash Function

In this section, we describe the new construction.

3.1 Hash Function Definitions

Let $\mathcal{H} = \{H_k\}_{k \in \mathcal{K}}$ be a keyed family of functions, where for each $k \in \mathcal{K}$, $H_k : \mathcal{X} \to \mathcal{Y}$. Here $\mathcal{X}$ and $\mathcal{Y}$ are finite non-empty sets with $|\mathcal{X}| > |\mathcal{Y}|$. Let x and x' be distinct elements of $\mathcal{X}$. The collision probability (over uniform random choice of k from $\mathcal{K}$) of $\mathcal{H}$ associated with the elements x and x' is defined to be $\Pr_k[H_k(x) = H_k(x')]$. Further, if $\mathcal{Y}$ is a commutative (additively written) group,

then for any fixed $\alpha \in \mathcal{Y}$, the differential probability (over uniform random choice of k from $\mathcal{K}$) associated with (x, x', α) is defined to be $\Pr_k[H_k(x) - H_k(x') = \alpha]$. The terminology of collision probability and differential probability in the current context is from [3].

The family $\mathcal{H}$ is said to be ϵ-almost XOR universal (ϵ-AXU) if the differential probability for any (x, x', α) is bounded above by ϵ. The family $\mathcal{H}$ is said to be ϵ-almost universal (ϵ-AU) if the collision probability for any (x, x') is bounded above by ϵ. Clearly, if $\mathcal{H}$ is ϵ-AXU, then it is also ϵ-AU.

$\mathcal{H}$ is said to be universal if it is ϵ-almost universal with $\epsilon = 1/|\mathcal{Y}|$. All the function families that we define in this paper are universal. In fact, for each case, $\mathcal{Y}$ is a commutative group and we show that the differential probabilities are equal to $1/|\mathcal{Y}|$.

3.2 Basic Idea

Fix a field $\mathbb{F}_q$ and an integer $n \geq 1$. Consider the extension field $\mathbb{F}_{q^n}$. Let ψ be a linear map from $\mathbb{F}_{q^n}$ to itself such that the minimal polynomial $\tau(x)$ in $\mathbb{F}_q[x]$ of ψ is of degree n and is irreducible over $\mathbb{F}_q$. For each $K \in \mathbb{F}_{q^n}$, we define a function $G_K : \cup_{l=1}^{n} \mathbb{F}_q^l \rightarrow \mathbb{F}_{q^n}$ as follows. Let $\mathbf{a} = (a_1, \ldots, a_l)$, for some $l \in \{1, \ldots, n\}$. Then

$$\left.\begin{aligned} G_K(\mathbf{a}) &= \langle (a_1, \ldots, a_l), (K, \psi(K), \ldots, \psi^{l-1}(K)) \rangle \\ &= a_1 K + a_2 \psi(K) + \cdots + a_l \psi^{l-1}(K). \end{aligned}\right\} \tag{1}$$

In other words, $G_K(\mathbf{a})$ is the linear combination of $(a_1, \ldots, a_l)$ and $(K, \psi(K), \ldots, \psi^{l-1}(K))$. A term of $G_K()$ is of the form $a_i \psi^{i-1}(K)$, where a_i is an element of $\mathbb{F}_q$ and $\psi^{i-1}(K)$ is an element of $\mathbb{F}_{q^n}$. The efficiency of evaluating the term $a_i \psi^{i-1}(K)$ depends on the representation of $\mathbb{F}_{q^n}$ and the choice of the map ψ. We say more about this later.

Lemma 3. *Fix an l with $1 \leq l \leq n$. The following are true for the function defined in (1).*

1. *For a fixed K, the function G_K restricted to inputs with l components, is a multi-linear function, i.e., it is linear in every component of its input.*
2. *Fix a non-zero $\mathbf{a} \in \mathbb{F}_q^l$. If K is uniformly distributed over $\mathbb{F}_{q^n}$ then so is $G_K(\mathbf{a})$.*
3. *Consequently, for $\mathbf{a}, \mathbf{a}' \in \mathbb{F}_q^l$, $\mathbf{a} \neq \mathbf{a}'$ and for any $\alpha \in \mathbb{F}_{q^n}$, $\Pr_K[G_K(\mathbf{a}) - G_K(\mathbf{a}') = \alpha] = 1/q^n$.*

Proof: It is easy to see that G_K is linear for fixed K. We prove the second statement. Let $\mathbf{a} = (a_1, \ldots, a_l)$ where $a_i \in \mathbb{F}_q$ and define a polynomial $p(x) = a_1 + a_2 x + \cdots + a_l x^{l-1}$. Since $\mathbf{a}$ is non-zero, $p(x)$ is also a non-zero polynomial over $\mathbb{F}_q$. Also, since $l \leq n$, the degree of $p(x)$ is less than n. Recall that the minimal polynomial $\tau(x)$ of ψ is irreducible over $\mathbb{F}_q$ and is of degree n. Thus, $p(x)$ is a non-zero polynomial which is coprime to $\tau(x)$. Using Lemma 1, we have $p(\psi)$ to be an invertible map from $\mathbb{F}_{q^n}$ to itself. Thus, if K is randomly distributed over $\mathbb{F}_{q^n}$, so is $p(\psi)(K)$. The second statement now follows on noting that $p(\psi)(K) = a_1 + a_2 \psi(K) + \cdots + a_l \psi^{l-1}(K) = G_K(\mathbf{a})$.

By linearity of G_K, $G_K(\mathbf{a}) - G_K(\mathbf{a}') = G_K(\mathbf{a} - \mathbf{a}')$. So, the third statement follows directly from the second statement. □

Key Length. For $G_K(\cdot)$, the key is K, which is an element of $\mathbb{F}_{q^n}$. In an efficient implementation of $\mathbb{F}_{q^n}$ the length of representation of K is n times the length of representation of an element of $\mathbb{F}_q$. One K can be used for messages consisting of upto n elements of $\mathbb{F}_q$. Thus, the key length is as long as the message. This feature will be true for the other hash functions that we define. Actually, this is an inherent property of the multi-linear hash construction from Section 1.1 and is present in other extensions of the idea, such as MMH [13] and UMAC [5].

3.3 Extending Message Length

In this and later sections, we use some terminology and notation from [5].

The function G can handle up to n elements of $\mathbb{F}_q$. It is the basic building block used for defining functions which can handle arbitrary length inputs. The idea is the following. Given m elements of $\mathbb{F}_q$, divide into (l_1+1) groups where each of the first l_1 groups has n elements and the last group has l_2 elements with $1 \leq l_2 \leq n$. To obtain the digest, apply G separately (with random and independent keys) to each of the groups and then add together all the individual digests.

More formally, we define a function family which is parametrized as $\mathsf{LH}[q,n,m]$. The domain is $\cup_{i=1}^{m} \mathbb{F}_q^i$, i.e., each element of the domain consists of upto m elements of $\mathbb{F}_q$; the range is $\mathbb{F}_{q^n}$. Let $t = \lceil m/n \rceil$. Each function in $\mathsf{LH}[q,n,m]$ is named by an element $\mathbf{K}$ of $\mathbb{F}_{q^n}^t$; a random function in $\mathsf{LH}[q,n,m]$ is given by a random element of $\mathbb{F}_{q^n}^t$. We write the function indicated by $\mathbf{K}$ as $\mathsf{LH}_{\mathbf{K}}(\cdot)$.

The message $\mathbf{M}$ consists of an l-tuple, $l \leq m$, over $\mathbb{F}_q$. Let $l = l_1 n + l_2$, with $1 \leq l_2 \leq n$. We consider $\mathbf{M}$ to be of the form $(\mathbf{M}_1, \ldots, \mathbf{M}_{l_1}, \mathbf{M}_{l_1+1})$, where $\mathbf{M}_1, \ldots, \mathbf{M}_{l_1}$ are in $\mathbb{F}_q^n$ and $\mathbf{M}_{l_1+1}$ is in $\mathbb{F}_q^{l_2}$. Further, let $\mathbf{K} = (K_1, \ldots, K_t)$ and note that $t \geq l_1 + 1$. Then, $\mathsf{LH}_{\mathbf{K}}(\mathbf{M})$ is defined as

$$\mathsf{LH}_{\mathbf{K}}(\mathbf{M}) = G_{K_1}(\mathbf{M}_1) + G_{K_2}(\mathbf{M}_2) + \cdots + G_{K_{l_1}}(\mathbf{M}_{l_1}) + G_{K_{l_1+1}}(\mathbf{M}_{l_1+1}). \tag{2}$$

Formally, we should write $\mathsf{LH}_{\mathbf{K}}[q,n,m](\mathbf{M})$ instead of $\mathsf{LH}_{\mathbf{K}}(\mathbf{M})$, but, we prefer the second notation as it is simpler. The parameters q, n and m will be clear from the context.

Theorem 1. *For any prime power q, any positive integers n and m, the differential probabilities of $\mathsf{LH}[q,n,m]$ for equal length strings are equal to q^{-n}.*

The previous multi-linear construction. We show that LH is a generalization of the construction mentioned in Section 1.1. Let $n = 1$, then $\mathbb{F}_{q^n} = \mathbb{F}_q$. Consider the function G. The domain, range and the key space of G becomes $\mathbb{F}_q$ and the definition of G is now $G_K(M) = KM$. Now consider the function $\mathsf{LH}[q,1,m]$. The domain is $A = \cup_{i=1}^{m} \mathbb{F}_q$ and the range is $\mathbb{F}_{q^n} = \mathbb{F}_q$. Also $t = \lceil m/n \rceil = m$ and each key $\mathbf{K} = (K_1, \ldots, K_m) \in \mathbb{F}_q^m$. Then for $\mathbf{M} = (M_1, \ldots, M_l)$, $\mathsf{LH}_{\mathbf{K}}(\mathbf{M}) = K_1 M_1 + \cdots + K_l M_l$. Thus, $\mathsf{LH}[q,1,m]$ is the multi-linear hash function from Section 1.1.

Extensions. The basic idea of LH can be extended in several ways. One can use the idea of decimation, the so-called Toeplitz construction or multiple linear operations. These details are given in Section 7.

3.4 Representation of $\mathbb{F}_{q^n}$ and the Linear Transformation ψ

Let $\mathbb{F}_{q^n}$ be represented using a polynomial $\tau(x)$ which is irreducible over $\mathbb{F}_q$. Then an element of $\mathbb{F}_{q^n}$ is given by an n-tuple of elements over $\mathbb{F}_q$. Let $\tau(x) = x^n - t_{n-1}x^{n-1} - \cdots - t_1 x - t_0$ with $t_{n-1}, \ldots, t_0 \in \mathbb{F}_q$. Given an element $(x_0, \ldots, x_{n-1})$ of $\mathbb{F}_{q^n}$, we define $(y_0, \ldots, y_{n-1}) = \psi(x_0, \ldots, x_{n-1})$ in the following manner.

$$\begin{aligned} y_0 &= t_{n-1}x_0 + t_{n-2}x_1 + \cdots + t_0 x_{n-1}; \text{ and} \\ y_i &= x_{i-1} \qquad\qquad\qquad\qquad \text{for } 1 \leq i \leq n-1. \end{aligned} \tag{3}$$

This provides an easy way to implement the map ψ. This is essentially a linear feedback shift register (LFSR). See [19] for a general discussion on LFSRs.

There are two possibilities for choosing the characteristics of $\mathbb{F}_{q^n}$ – characteristics 2 and characteristics a prime $\approx 2^w$. Table 1 provides some examples. The polynomials for the non-binary primes were generated using the PIPS server [7].

Table 1. Example parameters for $\mathsf{LH}[q, n, m]$. In each case, the differential probabilities are $\approx 2^{-128}$.

q	n	$\tau(x)$	diff. prob.
2	128	$x^{128} + x^{107} + x^{64} + x^{13} + 1$	2^{-128}
$2^8 + 1$	16	$x^{16} + 148x^7 + 50$	257^{-16}
$2^{16} + 1$	8	$x^8 + 35035x^4 + 6776$	$(2^{16} + 1)^{-8}$
$2^{32} + 15$	4	$x^4 + 173278966x + 723447520$	$(2^{32} + 15)^{-4}$

Implementation. The choices of q present different possibilities for implementing the algorithm. Also, the extensions of LH given in Section 7 can be combined with the choices of q. These lead to a large variety of possibilities with varying performances on different platforms. In this paper, we have chosen to focus on LH and $q = 2$. The rest of the paper is based on this choice. One of the reasons for making this choice is small hardware footprint of the resulting algorithm. At the same time, we would like to note that other choices of q may lead to software efficient algorithms. For example, $q = 2^{32} + 15$ is suitable for CPUs supporting 32-bit multiplications while $q = 2^8 + 1$ is suitable for CPUs supporting 8-bit multiplications. Developing these options require further work.

4 Tower Field Representation of $\mathbb{F}_{2^n}$

Suppose $n = n_1 n_2$. Fix an irreducible polynomial $\rho(x)$ of degree n_1 over $\mathbb{F}_2$. This gives rise to a representation of $\mathbb{F}_{2^{n_1}}$ as $\mathbb{F}_2[\alpha]/(\rho(\alpha))$. Choose a monic irreducible polynomial $\mu(x)$ of degree n_2 over $\mathbb{F}_2[\alpha]/(\rho(\alpha))$. This gives a representation of $\mathbb{F}_{2^n}$ as a two-part extension ($\mathbb{F}_2 \to \mathbb{F}_{2^{n_1}} \to \mathbb{F}_{2^n}$). The pair of polynomials $(\rho(\alpha), \mu(x))$ defines the particular representation.

Let $\mu(x) = x^{n_2} - c_{n_2-1}x^{n_2-1} - \cdots - c_1 x - c_0$ where c_is are elements of $\mathbb{F}_2[\alpha]/(\rho(\alpha))$. Using $\mu(x)$, we define the linear map ψ as in (3) in the following manner. Given $(x_0, \ldots, x_{n_2-1}) \in \mathbb{F}_{2^{n_1}}^{n_2}$, let $(y_0, \ldots, y_{n_2-1}) = \psi(x_0, \ldots, x_{n_2-1})$ where

$$\begin{aligned} y_0 &= c_{n_2-1}x_0 + c_{n_2-2}x_1 + \cdots + c_0 x_{n_2-1}; \text{ and} \\ y_i &= x_{i-1} \qquad\qquad\qquad\qquad \text{for } 1 \le i \le n_2 - 1. \end{aligned} \tag{4}$$

Evaluating y_0 in general requires n_2 multiplications over $\mathbb{F}_{2^{n_1}}$ and is not a better option than working directly over $\mathbb{F}_2$. On the other hand, if most of the c_is are either 0 or 1, then evaluating y_0 becomes much more efficient. In the best case, we will have all but one of the c_is to be 0 or 1. Table 2 provides examples of tower field representations of $\mathbb{F}_{2^n}$. In all cases, only the constant term of $\mu(x)$ is equal to α and hence (4) can be evaluated by a single multiplication by α modulo the corresponding $\rho(\alpha)$. Our method of obtaining these examples is discussed in Section 4.1.

The idea of using tower fields has been used in the context of stream ciphers such as SNOW [9]. The map ψ has then been called a word oriented LFSR. Tower fields have earlier been suggested to speed up scalar multiplication in elliptic curve cryptography. To the best of our knowledge, however, there is no earlier reference to the use of tower fields in the context of universal hash function construction.

Now we have a linear map ψ from $\mathbb{F}_{2^n}$ to $\mathbb{F}_{2^n}$ (defined via the tower field representation of $\mathbb{F}_{2^n}$). The message $\mathbf{a}$ to be hashed is a bit string. Recall that the definition of $G_K(\mathbf{a})$ assumed that the minimal polynomial $\tau(x)$ of the linear map ψ is in $\mathbb{F}_2[x]$ and is irreducible over $\mathbb{F}_2$. Here, we have defined ψ to be a map from $\mathbb{F}_{(2^{n_1})^{n_2}}$ (where $\mathbb{F}_{2^{n_1}}$ is represented by $\rho(x)$) to itself, whose

Table 2. Examples of $\mathbb{F}_{2^n}$ represented as a tower field. In each case, $\mu(x)$ is a primitive polynomial. The differential (and collision) probabilities are 2^{-n}.

n_1	n_2	$n = n_1 \times n_2$	$\rho(\alpha)$	$\mu(x)$
32	2	64	$\alpha^{32} + \alpha^{31} + \alpha^{29} + \alpha^1 + 1$	$x^2 + x + \alpha$
16	5	80	$\alpha^{16} + \alpha^5 + \alpha^3 + \alpha^2 + 1$	$x^5 + x^3 + \alpha$
96	3	96	$\alpha^{32} + \alpha^{18} + \alpha^9 + \alpha^2 + 1$	$x^3 + x + \alpha$
32	4	128	$\alpha^{32} + \alpha^{18} + \alpha^6 + \alpha^5 + 1$	$x^4 + x^3 + x + \alpha$
16	8	128	$\alpha^{16} + \alpha^{10} + \alpha^9 + \alpha^6 + 1$	$x^8 + x^3 + x + \alpha$
8	16	128	$\alpha^8 + \alpha^7 + \alpha^3 + \alpha^2 + 1$	$x^{16} + x^7 + x + \alpha$

minimal polynomial $\mu(x)$ is in $\mathbb{F}_{2^{n_2}}$ and is irreducible over $\mathbb{F}_{2^{n_2}}$. Lemma 2 assures us that the minimal polynomial of ψ *over* $\mathbb{F}_2$ is an irreducible polynomial of degree n as required.

4.1 Obtaining Tower Field Representations

We are interested in obtaining pairs of polynomials $(\rho(\alpha), \mu(x))$ where $\rho(\alpha)$ is a polynomial of degree n_1 over $\mathbb{F}_2$ while $\mu(x)$ is a polynomial of degree n_2 over $\mathbb{F}_{2^{n_1}}$. Further, we are interested in the condition that only the constant term of $\mu(x)$ is properly in $\mathbb{F}_{2^{n_1}}$ while the coefficients of all other terms are either 0 or 1. As far as we know, there is no known characterization or algorithm which ensures this condition. On the other hand, it is quite easy to test whether a polynomial is irreducible (or primitive) [21]. Based on this, we used the following strategy.

Find a random irreducible polynomial $\rho(\alpha)$. Let $\mathbf{b} = (b_1, \ldots, b_{n_2-1})$ be a bit string of length $n_2 - 1$. Define $\mu_{\mathbf{b}}(x) = x^{n_2} + b_{n_2-1}x^{n_2-1} + \cdots + b_1 x + \alpha$. This $\mu(x)$ is defined over $\mathbb{F}_{2^{n_1}}$. For all non-zero $(2^{n_2-1} - 1)$ bit strings $\mathbf{b}$, check whether $\mu_{\mathbf{b}}(x)$ is irreducible; if any one is indeed found to be irreducible, then report it; else try with another random irreducible $\rho(\alpha)$. Since the number of irreducible polynomials of a fixed degree is quite dense, this procedure succeeds withing a few trials. As mentioned earlier, some examples are given in Table 2.

4.2 Implementation

Hardware footprint. The hardware size of the basic design is very small. Apart from the registers, only XOR gates are required. The main focus of the choice of $q = 2$ and bit-by-bit processing is small CPUs with small memory and resource constrained devices such as small FPGAs or small ASIC structures.

Parallelism. There is a high level of parallelism in LH. Each 128-bit key block is used to process 128 bits of the message. So, separate 128-bit message blocks are processed using separate 128-bit key blocks. Thus, each 128-bit message and key blocks can be processed independently of each other and the final results are simply XORed together. Further, the same instruction applies to each message and key blocks leading to a typical SIMD parallelism. Incorporating parallelism in implementation will increase the throughput. This will also increase the size of hardware to some extent. A designer has to strike a proper balance between these two parameters.

A Simple 'C' Implementation for the Case $n_1 = 32$ and $n_2 = 4$ in Table 2. Four 32-bit words `K0, K1, K2` and `K3` constitute the 128-bit key which is to be used for processing 128 bits of

the message. The simple 'C' code of Figure 1 shows this computation. The macro `NextState` is one application of ψ given by (4) to the key. The macro `PROCESS(M,i)` processes the ith ($0 \leq i \leq 31$) bit of a 32-bit message word `M` in the following manner. It applies `NextState` to update the value of the key words; and if the ith bit of the message is one, then XORs the current value key to the current value of the hash result which is kept in the four 32-bit words `R0, R1, R2` and `R3`.

Note that the code is actually very simple. No finite field or multi-precision multiplications are required. In fact, these operations are so simple that they will be available in almost any processor and not necessarily large CPUs. In contrast, implementation of other algorithms such as UMAC or Poly1305 requires a carefully optimized multi-precision software library including possible assembly language implementation. Consequently, the code sizes of such algorithms are several orders of magnitude larger than the simple code shown in Figure 1. A drawback, however, is that on a general purpose CPU, the speed will be slower than that of UMAC or Poly1305. This is due to the fact that general purpose CPUs do not provide native support for cyclic shift, inner product and other vector operations. For such CPUs, LH should be considered to provide a trade-off between code size and complexity versus computation time.

Fig. 1. A simple 'C' implementation of ψ using a tower field representation of $\mathbb{F}_{2^{128}}$.

```
const unsigned int val1[2] = {0,((1<<18)^(1<<6)^(1<<5)^1)};
const unsigned int val2[2] = {0,0xffffffff};

#define NextState { \
                tmp = (K3<<1)^(val1[K3>>31])^K0^K2; \
                K3=K2; K2=K1; K1=K0; K0=tmp; \
}

#define PROCESS(M,i) {\
                tmp = val2[(M&(1<<i))>>i]; \
                R0 = R0^(K0&tmp); R1 = R1^(K1&tmp); \
                R2 = R2^(K2&tmp); R3 = R3^(K3&tmp); \
}
```

4.3 Tackling Variable Length Inputs

Theorem 1 ensures low differential probability only for equal length inputs. For designing a MAC algorithm, we need to handle variable length messages. We describe how this can be done for $q = 2$. For other values of q, it is possible to suitably modify the proposal.

For $q = 2$, the message is actually a bit string. So, there is no problem of handling partial blocks. The basic technique for handling variable length messages is to append the binary representation of the length to the message. In our case, this simple construction itself will ensure low differential probability for variable length messages. But, such a simple padding rule may result in messages which do not align properly to byte or word boundaries. So, we modify the construction so that the padded length is a multiple of 32. (This can easily be modified to obtain a construction where the padded length is a multiple of 8 or 16.)

We define $\mathsf{UH}[2,n]$ to be a family of functions. The domain of each function in the family consists of all finite length binary strings and the range is $\mathbb{F}_{2^n}$. We have to define the key space for the family. Recall that in our construction, we require the key to be as long as the message. Since, each function of the family UH can handle arbitrary length messages, the key has to be a (one-way) infinite binary string. In other words, the family UH is indexed by the set of all one-way infinite binary strings. This, however, is only a theoretical necessity in making a consistent definition. In practice, one will only work with a key which is only as long as the message.

Let $\mathbf{a} = (a_1, \ldots, a_l)$ be the message, where $l \geq 1$ and each a_i is a bit. Let $\mathsf{pad}(\mathbf{a})$ be the string obtained by padding a minimum number of zeros to $\mathbf{a}$ so as to ensure that the length of $\mathsf{pad}(\mathbf{a})$ is a multiple of 32. Then

$$\mathsf{UH}_{\mathbf{K}}(\mathbf{a}) = \mathsf{LH}_{\mathbf{K}_1}(\mathsf{pad}(\mathbf{a})||\lambda_{64}(l)). \tag{5}$$

By $\widehat{\mathbf{a}}$ we will denote the binary string $\mathsf{pad}(\mathbf{a})||\lambda_{64}(l)$. Here, $\lambda_{64}(l)$ is defined as follows: let y be the minimum length binary representation of l; reverse y and pad on the left with zeros to obtain a binary string of length 64; this is our desired $\lambda_{64}(l)$. For example, if $l = 11$, then $\lambda_{64}(11) = 0^{60}1101$. By construction, the last bit of $\lambda(l)$ is always 1. $\mathbf{K}$ is a one-way infinite binary string while $\mathbf{K}_1$ is the initial segment of $\mathbf{K}$ of length $\mathsf{blklen}_n(|\widehat{\mathbf{a}}|)$ bits, where $\mathsf{blklen}_n(l) = n \times \lceil l/n \rceil$.

Theorem 2. *Let* $\mathbf{a}$ *and* $\mathbf{a}'$ *be two distinct binary strings having lengths* l *and* l' *respectively with* $l \geq l'$. *Let* $\mathbf{K}$ *be a key for* UH *and set* $b = \mathsf{UH}_{\mathbf{K}}(\mathbf{a})$ *and* $b' = \mathsf{UH}_{\mathbf{K}}(\mathbf{a}')$. *Let* $\mathbf{K}_1$ *be the initial segment of* $\mathbf{K}$ *of length equal to* $\mathsf{blklen}_n(|\widehat{\mathbf{a}}|)$ *bits. Let* δ *be any element of* $\mathbb{F}_{2^n}$. *Then*

$$\mathsf{Pr}[\mathsf{UH}_{\mathbf{K}}(\mathbf{a}) - \mathsf{UH}_{\mathbf{K}}(\mathbf{a}') = \delta] = 1/2^n.$$

Here the probability is over random choice of $\mathbf{K}_1$ *(and not the whole of* $\mathbf{K}$*).*

Proof: We are given that $\mathbf{K}_1$ is the initial segment of $\mathbf{K}$ of length equal to $\mathsf{blklen}_n(|\widehat{\mathbf{a}}|)$ bits. Let $\mathbf{K}'_1$ be the initial segment of $\mathbf{K}$ whose length is equal to $\mathsf{blklen}_n(|\widehat{\mathbf{a}'}|)$ bits.

Let $k = |\widehat{\mathbf{a}}|$ and $k' = |\widehat{\mathbf{a}'}|$. By the condition of the theorem, we have $k \geq k'$.

The first point to note is that if $\mathbf{a} \neq \mathbf{a}'$, then $\mathbf{x} = \mathsf{pad}(\mathbf{a})||\lambda_{64}(l) \neq \mathsf{pad}(\mathbf{a}')||\lambda_{64}(l')||0^{k-k'} = \mathbf{x}'$. (Note that $\mathbf{x} = \widehat{\mathbf{a}}$ and $\mathbf{x}' = \widehat{\mathbf{a}'}||0^{k-k'}$.) There are two cases to see this. If $k = k'$, then $\mathbf{x} \neq \mathbf{x}'$ follows directly from $\mathbf{a} \neq \mathbf{a}'$. On the other hand, if $k > k'$, then the last bit of $\mathbf{x}'$ is 0, where as the last bit of $\mathbf{x}$ is 1 from the definition of $\lambda_{64}(l)$. Thus, $\mathbf{x} \neq \mathbf{x}'$ also holds in this case. Note that both $\mathbf{x}$ and $\mathbf{x}'$ have the same length.

The second point is that

$$\begin{aligned}\mathsf{UH}_{\mathbf{K}}(\mathbf{a}) &= \mathsf{LH}_{\mathbf{K}_1}(\mathbf{x}) \text{ and} \\ \mathsf{UH}_{\mathbf{K}}(\mathbf{a}') &= \mathsf{LH}_{\mathbf{K}'_1}(\mathsf{pad}(\mathbf{a}')||\mathsf{bin}_{64}(l')) \\ &= \mathsf{LH}_{\mathbf{K}_1}(\mathbf{x}').\end{aligned}$$

Again this is easy to see for $k = k'$. If $k > k'$, then the padding at the end by zeros does not affect the output of the computation of LH. Now the problem reduces to that of bounding the differential probability of LH for equal length strings. From Theorem 1, we obtain this probability to be $1/2^n$. $\square$

Note. Instead of using separate notations $\mathbf{K}$ and $\mathbf{K}_1$, we will use the convention that the key $\mathbf{K}$ in $\mathsf{UH}_{\mathbf{K}}(\mathbf{a})$ is of length equal to $\mathsf{blklen}_n(|\widehat{\mathbf{a}}|)$ (instead of being an infinite length binary string). This is consistent with the definition of UH, since the later bits of an infinite $\mathbf{K}$ does not affect the computation of the hash value for $\mathbf{a}$.

4.4 Comparison to Some Previously Known Universal Hash Families

The technique of polynomial evaluation based hash is different from the approach taken in this paper. In the former, keys are short but security degrades with increase in length of the message, while, in the latter, keys are as long as the message but collision probability is the best possible. The proposals MMH [13] and UMAC [5] belong to the latter approach. Poly1305 [2] and PolyR [18] belong to the former approach.

Due to the bit-at-a-time approach, software speed of $\mathsf{LH}[2, n, m]$ on general purpose CPUs is not comparable to that of the above mentioned constructions. On the other hand, for resource constrained devices which cannot support large multiplier circuits, the speed of $\mathsf{LH}[2, n, m]$ should be comparable to the others. (We clarify, though, that we have not actually done any speed measurements on such platforms.) For constrained environments, one clear advantage of $\mathsf{LH}[2, n, m]$ is small hardware size; implementing any of UMAC, Poly1305 or PolyR will require substantially more hardware for implementing a multiplier circuit.

The core of UMAC is the non-linear hash function NH^{T}. Here the superscript T stands for the Toeplitz construction and NH^{T} is built from a non-linear hash function NH. Let $\mathbf{M} = (M_1, \ldots, M_l)$, l even, be the message and $\mathbf{K} = (K_1, \ldots, K_n)$, with $n \geq l$ be the key. Each M_i and K_j are w-bit integers. Then $\mathsf{NH}_{\mathbf{K}}(\mathbf{M})$ is computed as

$$\left(\sum_{i=1}^{l/2} ((M_{2i-1} + K_{2i-1}) \bmod 2^w) \cdot ((M_{2i} + K_{2i}) \bmod 2^w) \right) \bmod 2^{2w}. \tag{6}$$

The output is a $2w$-bit integer and the collision probability is shown to be 2^{-w}. Hence, NH is ϵ-almost universal with $\epsilon = 2^{-w}$. In contrast, all the hash families defined in this paper are XOR universal. The cost of computing NH is $l/2$ multiplications modulo 2^{2w} and l additions modulo 2^w. The construction NH^{T} is an extension, which produces a digest of size $2tw$; has collision probability 2^{-tw} and the cost is $tl/2$ multiplications modulo 2^{2w} and l additions modulo 2^w. To provide collision probability of 2^{-128} with $w = 32$ will require a digest of 256 bits, i.e., $t = 4$. The cost will be $2l$ multiplications modulo 2^{64} and $4l$ additions modulo 2^{32}.

Shoup [25] described three methods for defining a universal hash function – polynomial evaluation, division hash (earlier called "LFSR hashing" or cryptographic CRC by Krawzcyk [17]) and generalized division hash. The third method has the first two as special cases. In GDH, the message is a polynomial $m(x)$ over $\mathbb{F}_{2^k}$ of degree less than nl/k. The key is a random monic irreducible polynomial $\tau(x)$ of degree l/k over $\mathbb{F}_{2^k}$. The hash value is $m(x)x^{l/k} \bmod \tau(x)$. The differential probabilities are bounded above by $nl/(k2^l)$.

If $l = k$, then $\tau(x) = x + \alpha$ and the hash function evaluation reduces to computing $\alpha \cdot m(\alpha)$ over $\mathbb{F}_{2^k}$. This is the polynomial evaluation hash over $\mathbb{F}_{2^k}$. If $k = 1$, then the hash function computation reduces to evaluating $m(x)x^l \bmod \tau(x)$, where $m(x)$ is a polynomial over $\mathbb{F}_2$ and $\tau(x)$ is an irreducible polynomial of degree l over $\mathbb{F}_2$. The case $k = 1$ corresponds to Krawzcyk's "LFSR hashing".

Comparing GDH with our approach, for one thing, in our case, the polynomial is part of the specification (i.e., public) and the LFSR is repeatedly applied to the secret key. Secondly, in our case, the collision probability for n-bit digest is 2^{-n} whereas for GDH it is $nl/(k2^n)$, i.e., there is a degradation in security. On the other hand, the key in our construction is as long as the message, while in GDH it is an irreducible polynomial of degree l/k over $\mathbb{F}_{2^k}$. The key in our construction is taken to be the output of a PRG. In contrast, for $l > k$, it is much more complicated to change key in GDH, since this requires choosing a random irreducible polynomial over $\mathbb{F}_{2^k}$.

Table 3 provides the sizes of hash outputs of different schemes for obtaining a collision probability of 2^{-128}. The table shows that LH has the smallest (and optimum) size output.

Table 3. Size of hash outputs of different schemes for attaining collision probability of 2^{-128}. For the first row we assume that the message consists of at most 2^{32} 128-bit blocks.

algorithm	key len	output sz.	coll prob.
poly hash, GDH	constant	160	2^{-128}
NH^{T}	eq to msg len	256	2^{-128}
LH	eq to msg len	128	2^{-128}

Johansson [16] had earlier suggested the use of LFSRs in the construction of authentication codes. These can also be viewed as universal hash functions. The construction from [16] is the following. Let $\mathbb{F}_q$ be the finite field of q elements. Messages, digests and keys are fixed length sequences of length m, n and $(m+n)$ respectively over $\mathbb{F}_q$. Let $\mathbf{K} = (K_1, \ldots, K_m, K_{m+1}, \ldots, K_{m+n})$ and set $\mathbf{K}_a = (K_1, \ldots, K_m)$. Let $\mathcal{L}$ be an LFSR over $\mathbb{F}_q$ whose connection polynomial is of degree m and is irreducible over $\mathbb{F}_q$. We consider $\mathcal{L}$ to be the state update function of the LFSR, i.e., $\mathcal{L}$ is a map from $\mathbb{F}_q^m$ to $\mathbb{F}_q^m$. For a message $\mathbf{M} \in \mathbb{F}_q^m$, the digest is defined to be

$$\left(K_{m+1} + \langle \mathbf{M}, \mathbf{K}_a \rangle, K_{m+2} + \langle \mathbf{M}, \mathcal{L}(\mathbf{K}_a) \rangle, \cdots, K_{m+n} + \langle \mathbf{M}, \mathcal{L}^{n-1}(\mathbf{K}_a) \rangle\right). \tag{7}$$

Each component is an element of $\mathbb{F}_q$ and hence the digest is an element of $\mathbb{F}_q^n$. This construction is different from our proposal. Below, we discuss some of the differences.

1. Our construction uses a linear map over $\mathbb{F}_{q^n}$. Implementations of this map can be done by either an LFSR over $\mathbb{F}_q$ or using a tower field representation of $\mathbb{F}_{q^n}$. There is no scope of using tower field representation in [16].
2. The length of the LFSR in [16] is equal to the length of the message, whereas in the LFSR based implementation of our proposal, the length of the LFSR is equal to the length of the digest.
3. The work [16] does not discuss how to handle variable length messages. Since the length of the LFSR is equal to the length of the message, tackling variable length messages will mean using different length LFSRs for different message lengths. This makes handling variable length messages almost impossible.
4. Suppose that the length m of the message is a multiple of the length n of the digest, i.e., $n|m$. The length of the key in [16] is $m + n$ which means that the key is necessarily longer than the message. For this case, in our construction, the key length is m which is shorter than the key length used in [16].

Based on these points, it is clear that our construction is different from that of [16] and in fact, offers several advantages over that of [16]. The only superficial similarity of [16] to our general framework is the use of LFSRs in [16].

5 A New MAC Construction

There are two known ways to construct a MAC from a universal hash function. Both use a PRF in conjunction with a hash function having low collision or differential probabilities. The first method, due originally to Wegman-Carter (with later analysis by [25, 3]) is $\mathrm{PRF}_{K_f}(\mathrm{Nonce}) \oplus \mathrm{HASH}_{K_h}(M)$. The second method (described in [5]) is $\mathrm{PRF}_{K_f}(\mathrm{HASH}_{K_h}(M), \mathrm{Nonce})$. In both cases, M is the message; K_f is the key for the PRF and K_h is the key for the hash function. Also, the nonce needs to be communicated from the sender to the receiver. Alternatively, this can be achieved if the sender and receiver maintain state.

Though the second method has been proposed for UMAC, the technique described in [5] is generic. As mentioned earlier, the core of UMAC is an almost universal hash function called NH. The key length for NH is as long as the message length. Our construction LH has the same property with respect to key length. In view of this, our first observation on obtaining a MAC from LH is that this can be easily done by simply plugging LH into the generic techniques given in [5].

Alternatively, we describe a different way of combining a universal hash family with a PRF. This technique is suited for universal hash families (like UMAC and our construction), where the key is as long as the message. The way to tackle this is to use a PRG to generate the key. We take this idea further. In practice, a cryptographic PRG is essentially an additive stream cipher. Current designs of stream ciphers include an IV. For example, all the proposals of eSTREAM are stream ciphers with IV.

The formal model of a stream cipher with IV is that of PRF. (This has been folklore in the stream cipher community and has been formalized in [1].) The key for the stream cipher is considered to be the key for the PRF and each function maps an IV to a keystream. We exploit this formal connection to propose a MAC based on a universal hash function and a stream cipher with IV.

Let $\mathsf{SC}_K(\mathrm{IV})$ be a stream cipher with IV; which for a fixed key K produces a "long" keystream from a given IV. The MAC construction using UH is the following. The key for the MAC algorithm is the key for SC. Tag generation is done as follows.

1. Let $\mathbf{a}$ be the message (binary string) to be hashed.
2. Let IV be a nonce.
3. Let $\mathbf{K}$ of length $\mathsf{blklen}_n(|\widehat{\mathbf{a}}|)$ be produced using $\mathsf{SC}_K(\mathrm{IV})$.
4. The tag is $\mathsf{UH}_{\mathbf{K}}(\mathbf{a})$.

Though we have defined this only for UH, this can be easily changed to fit with any universal hash family which has key to be as long as the message.

Let $\mathsf{Adv}_{\mathsf{SC}}(t, q)$ denote an adversary's advantage in breaking SC as a PRF, where the the runtime of the adversary is at most t and it makes at most q queries. Then the following result states the security of the above construction.

Theorem 3. *Let $\mathsf{Adv}_{\mathsf{MAC}}(t, q)$ denote the maximum advantage of an adversary in successfully forging for the above MAC algorithm. Then*

$$\mathsf{Adv}_{\mathsf{MAC}}(t, q) \leq \mathsf{Adv}_{\mathsf{SC}}(t, q) + \frac{1}{2^n}.$$

Proof: As usual one proves the information theoretic version of this result, where the runtime is ignored and only the number of queries made by the adversary is bounded. The proof consists of two games. Let $\mathcal{X}_i$, $i = 0, 1$ be the event that the forging query verifies in Game i.

Game 0. This is the usual forging game for the MAC algorithm.

Game 1. In this game, SC is not used. For each of the tag generation queries made by the adversary $\mathcal{A}$, a new random string is generated and is used as the key to hash the given message. Note that since it is not allowed to repeat nonces (IVs), this does not lead to any inconsistency. But, the nonce in the forging query can be same as one of the earlier nonces. So, there are two cases to consider. If this nonce is not same, then again generate a random string of appropriate length and run the tag verification algorithm. If, on the other hand, this nonce is equal to a previous nonce, then the key corresponding to the previous nonce is used to run the verification algorithm for the forging query. In both cases, the probability that the verification is successful is bounded above by $1/2^n$. For the first case, this follows from Theorem 1 while in the second case, this follows from Theorem 2. So, $\mathsf{Pr}[X_1] = 1/2^n$.

The difference between Games 0 and 1 is in the way the key strings are generated. In the first case, this is generated using SC, while in the second case these are randomly generated. Based on this difference, it is easy to construct an adversary $\mathcal{B}$ which attacks the PRF-property of SC. Basically, $\mathcal{B}$ will be given an oracle which on input an IV returns a string of desired length. At the end of the game, if the forging query made by $\mathcal{A}$ verifies, then $\mathcal{B}$ returns 1, else 0.

If the oracle is SC, then $\mathcal{B}$ is playing Game 0 and if the oracle is random, then $\mathcal{B}$ is playing Game 1. Also, $\mathsf{Pr}[X_0] = \mathsf{Pr}[\mathcal{B} \Rightarrow 1|\text{oracle is real}]$ and $\mathsf{Pr}[X_1] = \mathsf{Pr}[\mathcal{B} \Rightarrow 1|\text{oracle is random}]$. So,

$$\begin{aligned}
\mathsf{Adv}_{\mathsf{MAC}}(t, q) &\leq \mathsf{Pr}[X_0] \\
&\leq |\mathsf{Pr}[X_0] - \mathsf{Pr}[X_1]| + \mathsf{Pr}[X_1] \\
&= |\mathsf{Pr}[\mathcal{B} \Rightarrow 1|\text{oracle is real}] - \mathsf{Pr}[\mathcal{B} \Rightarrow 1|\text{oracle is random}]| + 1/2^n \\
&\leq \mathsf{Adv}_{\mathsf{SC}}(q) + 1/2^n.
\end{aligned}$$

This completes the proof. □

Authenticated encryption (AE). Applications often need to perform both encryption and authentication. It is possible to use $\mathsf{SC}_K()$ for this. Basically use two IVs (as nonces), IV_1 for encryption and IV_2 for authentication. Then the two operations can be done in parallel. It is also possible to use a single IV with alternate keystream segments being used for encryption and authentication. Which approach is preferrable may depend on the actual application.

MAC with small hardware footprint. The construction UH is based on LH which, as discussed earlier, has a very small footprint in hardware. To obtain a MAC algorithm with a small hardware footprint, UH should be combined with a stream cipher also having a small hardware footprint. There are several examples of such ciphers as part of eSTREAM Phase 3, Profile 2 proposals.

Software efficient MAC. Note that our technique of combining a universal hash function with stream cipher with IV is generic. As a result, one can combine a software efficient stream cipher with a software efficient universal hash (such as UMAC) to obtain a MAC algorithm which is very fast in software.

6 Blockwise Universal Hashing and Tweakable Enciphering Scheme

The notion of invertible blockwise universal hash family was considered by Naor and Reingold [22] in the context of constructing a strong pseudorandom permutation (SPRP). Halevi [12] presented a new construction of such a hash family and used it to construct a tweakable SPRP (also called a tweakable enciphering scheme). These constructions are based on polynomial hashing, which requires multiplication over $\mathbb{F}_{2^n}$.

In this section we present a new blockwise universal hash family BEB which does not require any $\mathbb{F}_{2^n}$ multiplication. Also, we show how to construct a tweakable SPRP using the blockwise universal hash family.

We will denote the new blockwise universal hash family by $\mathsf{BUH}[2, n, m]$. This is built using $\mathsf{LH}[2, n, m]$. The key for BUH consists of the key $\mathbf{K}$ for LH along with another element L of $\mathbb{F}_{2^n}$. Recall that the definition of LH requires a linear map ψ from $\mathbb{F}_{2^n}$ to itself and whose minimal polynomial $\tau(x)$ over $\mathbb{F}_2$ is irreducible over $\mathbb{F}_2$. For, the present purpose we will assume $\tau(x)$ to be also primitive. In fact, all the pairs $(\rho(\alpha), \mu(x))$ shown in Table 2 have $\mu(x)$ to be primitive and so the corresponding $\tau(x)$ is also primitive.

The domain and range of $\mathsf{BUH}[2, n, m]$ consists of $\mathbb{F}_{2^n}^m$, i.e., m-tuples over $\mathbb{F}_{2^n}$. The key $\mathbf{K}$ also consists of an m-tuple over $\mathbb{F}_{2^n}$. Given key $(\mathbf{K}, L)$ and an m-tuple $(X_1, \ldots, X_m)$, the output of $\mathsf{BUH}_{\mathbf{K},L}(X_1, \ldots, X_m)$ is given by the following map.

$$(X_1, \ldots, X_m) \mapsto (X_1 + Y, \ldots, X_{m-1} + Y, Y) + (\psi(L), \ldots, \psi^{m-1}(L), L) \tag{8}$$

where $Y = X_m + \mathsf{LH}_{\mathbf{K}}(X_1, \ldots, X_{m-1})$. (This is similar to the construction in [24].)

BUH is an invertible map and this is easy to see.

1. Suppose we are given $(Y_1, \ldots, Y_m) = \mathsf{BUH}_{\mathbf{K},L}(X_1, \ldots, X_m)$;
2. compute $(A_1, \ldots, A_{m-1}, Y) = (Y_1, \ldots, Y_{m-1}, Y_m) - (\psi(L), \ldots, \psi^{m-1}(L), L)$;
3. compute $(X_1, \ldots, X_{m-1}, Y) = (A_1 - Y, \ldots, A_{m-1} - Y, Y)$; and
4. finally compute $X_m = Y - \mathsf{LH}_{\mathbf{K}}(X_1, \ldots, X_{m-1})$.

The main result on BUH is the following which essentially states that the construction is blockwise universal.

Theorem 4. *Fix positive integers n and m and consider the map $\mathsf{BUH}[2, n, m]$. For random choices of $\mathbf{K}$ from $\mathbb{F}_{2^n}^m$ and L from $\mathbb{F}_{2^n}$, and for any two m-tuples $\mathbf{X} = (X_1, \ldots, X_m)$, $\mathbf{X}' = (X'_1, \ldots, X'_m)$ from $\mathbb{F}_{2^n}^m$, let $(Y_1, \ldots, Y_m) = \mathsf{BUH}_{\mathbf{K},L}(X_1, \ldots, X_m)$ and $(Y'_1, \ldots, Y'_m) = \mathsf{BUH}_{\mathbf{K},L}(X'_1, \ldots, X'_m)$. Let $1 \le i, i' \le m$ be such that $(\mathbf{X}, i) \ne (\mathbf{X}', i')$. Then*

$$\mathsf{Pr}_{\mathbf{K},L}[Y_i = Y'_{i'}] = 1/2^n.$$

Proof: The proof consists of several cases.

Case 1 ($i \ne i'$): Without loss of generality assume that $i < i'$. This sub-divides into two cases.

Case 1a ($1 \le i < i' < m$): Then

$$\begin{aligned} Y_i - Y'_{i'} &= (X_i - X'_{i'}) + (Y - Y') + \psi^i(L) - \psi^{i'}(L) \\ &= Z + \psi^i(L - \psi^{i'-i}(L)) \end{aligned}$$

where Z is independent of L. The quantity $\psi^i(L - \psi^{i'-i}(L))$ is uniformly distributed over $\mathbb{F}_{2^n}$ and so $\mathsf{Pr}[Y_i - Y'_{i'}] = 1/2^n$.

Case 1b ($1 \le i < i' = m$)**:** In this case,

$$Y_i - Y'_{i'} = X_i + (Y - Y') + \psi^i(L) - L.$$

An argument as in the previous case settles this case.

Case 2 ($i = i'$)**:** Then $\mathbf{X} \neq \mathbf{X}'$. Let $\Delta X_j = X_j - X'_j$ for $1 \le j \le m$. Again there are two cases.

Case 2a ($1 \le i = i' < m$)**:** Then using the linearity of LH, we have

$$\begin{aligned} Y_i - Y'_{i'} &= \Delta X_i + \Delta X_m + (Y - Y') \\ &= \Delta X_i + \Delta X_m + \mathsf{LH}_{\mathbf{K}}(\Delta X_1, \ldots, \Delta X_{m-1}). \end{aligned}$$

If $(\Delta X_1, \ldots, \Delta X_{m-1}) \neq (0, \ldots, 0)$, then by Theorem 1, we have that $\mathsf{LH}_{\mathbf{K}}(\Delta X_1, \ldots, \Delta X_{m-1})$ is uniformly distributed over $\mathbb{F}_{2^n}$ and so $\mathsf{Pr}_{\mathbf{K}}[\mathsf{LH}_{\mathbf{K}}(\Delta X_1, \ldots, \Delta X_{m-1}) = -(\Delta X_i + \Delta X_m)]$ is $1/2^n$. If on the other hand, $(\Delta X_1, \ldots, \Delta X_{m-1}) = (0, \ldots, 0)$, then we must have $\Delta X_m \neq 0$. But, then $Y_i - Y'_{i'} = \Delta X_m \neq 0$.

Case 2b ($1 \le i = i' = m$)**:** In this case,

$$Y_i - Y'_{i'} = \Delta X_m + \mathsf{LH}_{\mathbf{K}}(\Delta X_1, \ldots, \Delta X_{m-1})$$

and an argument as above settles this case. □

6.1 Tweakable Enciphering Scheme (TES)

A tweakable SPRP is also called a TES. BUH can be used to construct a TES. The basic idea is that of Naor and Reingold – use a layer of ECB encryption between two blockwise universal hash layers. The ECB encryption consists of applying an n-bit block cipher $E_K()$ in parallel to the blocks, i.e., $\mathrm{ECB}_K(X_1, \ldots, X_m) = (E_K(X_1), \ldots, E_K(X_m))$.

First assume that there are no tweaks and messages consists of m blocks, where m is fixed. We call the scheme BEB and the encryption and decryption algorithms for this scheme is shown in Table 4. The notation $\mathsf{BEB}[E]$ denotes the mode of operation BEB instantiated by the block cipher E. Stating and proving the security of BEB requires a rather elaborate machinery. A summary of

Table 4. Encryption and decryption using BEB. The keys consist of K as the block cipher key and $(\mathbf{K}, L)$ as the hashing keys.

$(PP_1, \ldots, PP_m) = \mathsf{BUH}_{\mathbf{K},L}(P_1, \ldots, P_m)$	$(CC_1, \ldots, CC_m) = \mathsf{BUH}_{\mathbf{K},L}(C_1, \ldots, C_m)$
$(CC_1, \ldots, CC_m) = \mathrm{ECB}_K(PP_1, \ldots, PP_m)$	$(PP_1, \ldots, PP_m) = \mathrm{ECB}_K^{-1}(CC_1, \ldots, CC_m)$
$(C_1, \ldots, C_m) = \mathsf{BUH}^{-1}_{\mathbf{K},L}(CC_1, \ldots, CC_m)$	$(P_1, \ldots, P_m) = \mathsf{BUH}^{-1}_{\mathbf{K},L}(PP_1, \ldots, PP_m)$

the necessary preliminaries and the security statement along with a sketch of the proof is given in Section 6.2. The basic BEB design given in Table 4 assumes that there is no tweak and the messages have a fixed number of blocks. These can be made flexible.

Tweakable, but fixed number of blocks. This special case is of interest, since this is required for disk encryption. Let T be an n-bit tweak. Define $L = E_K(T)$. With this change, the encryption and decryption algorithms remain the same.

Tweakable and variable number of blocks. Again let T be an n-bit tweak. Define $R = E_K(T)$ and then $L = E_K(R + \mathsf{bin}_n(m))$, where $\mathsf{bin}_n(m)$ is the n-bit binary representation of m.

Security of the variants. The security proofs for these two variants will be a little different from that for the simple variant. The main difference is in the combinatorial analysis. The corresponding bound on collision probability can be shown to be $4\sigma_n^2/2^n$, where σ_n is the total number of n-bit blocks (including the tweaks) provided by the adversary in all its queries.

Comparison to previous TES. The main novelty of the proposed TES in comparison to all previous TES is that it requires a *single layer* of encryption and *no* $\mathbb{F}_{2^n}$ *multiplications*. The hashing layers are implemented using only LFSRs. For example, if AES-128 is used as the block cipher, then one can use a word oriented LFSR as discussed in Section 4 to implement the hashing layer. From a hardware point of view, this leads to a minimal overhead compared to the use of a multiplication circuit. The trade-off is that the hashing layers require a long key $\mathbf{K}$. Consider for example, the case of disk encryption. Each sector is 512 bytes and so $\mathbf{K}$ is also 512 bytes long. Thus, avoiding multiplications comes at the cost of increase in key size. Though long, the same key will be used for all sectors, i.e., a single 512-byte key is required for the entire disk. This is a tolerable trade-off, especially if we keep in mind that many multiplication-based TES advocate the use of pre-computed tables to speed up the required multiplications.

Disk encryption protocol with small hardware footprint. Since BUH is built from LH, it has a small hardware footprint as already discussed. Hardware space for a disk encryption protocol based on BEB will be dominated by the space to implement an AES block. There is no need to implement a multiplier circuit as required in some of the other proposals. Schemes such as EME also require an AES block as the significant hardware component. Compared to EME, BEB uses a single layer of encryption whereas EME uses two layers of encryption.

6.2 Security of BEB

We summarize some of the relevant preliminaries required to state the security of BEB. Details can be found in [14] and later work. The usual approach is to consider information theoretic security, i.e., security where the block cipher E is instantiated by a permutation chosen randomly from the set of all permutations of $\{0,1\}^n$. This is denoted by $\mathsf{BEB}[\mathrm{Perm}(n)]$, where $\mathrm{Perm}(n)$ denotes the set of all permutations of $\{0,1\}^n$.

The security game assumes that an adversary A interacts with the encryption and decryption oracles. This is denoted by $A^{\mathbf{E}_\pi, \mathbf{D}_\pi}$, where $\mathbf{E}_\pi$ (resp. , $\mathbf{D}_\pi$) denotes the encryption (resp. decryption) oracle for BEB instantiated by a random permutation π. The notation $A^{\mathbf{E}_\pi, \mathbf{D}_\pi} \rightarrow 1$ denotes that the adversary finally outputs the bit 1. For proving security one considers the following advantage of an adversary A.

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{BEB}[\mathrm{Perm}(n)]}(A) = \left| \mathsf{Pr}\left[\pi \xleftarrow{\$} \mathrm{Perm}(n) : A^{\mathbf{E}_\pi, \mathbf{D}_\pi} \rightarrow 1\right] - \mathsf{Pr}\left[A^{\$(.,.),\$(.,.)} \rightarrow 1\right] \right|$$

where $\$(., M)$ returns a random bit string of length $|M|$. The maximum (over all adversaries making q queries) of this advantage is denoted by $\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{BEB}[\mathrm{Perm}(n)]}(q)$.

Theorem 5. *Fix n, m and q to be positive integers. Suppose that an adversary makes a total of q queries, where each query consists of m n-bit strings. Then*

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{BEB}[\mathrm{Perm}(n)]}(q) \leq \frac{2(qm)^2}{2^n}.$$

Proof: The first assumption is that the adversary does not make any pointless queries, i.e., queries for which it already knows the answer or for which it can easily compute the answer. This means that it cannot repeat queries; and neither can it make an encryption query of a string which it had earlier received as answer to a decryption query and vice versa.

The usual proof strategy is a game sequence followed by a combinatorial analysis. The quantities P_i, C_i, PP_i and CC_i corresponding to the sth query will be denoted by P_i^s, C_i^s, PP_i^s and CC_i^s. Details of the games are similar to previous work [14] and so we skip these. The purpose of the sequence is to arrive at a non-interactive game, where there is no adversary. Instead a transcript of q queries is provided, where for each query, both the plaintext and ciphertext blocks are given.

Given the transcript, for $1 \leq s \leq q$, we define $(PP_1^s, \ldots, PP_m^s) = \mathsf{BUH}_{\mathbf{K},L}(P_1^s, \ldots, P_m^s)$ and $(CC_1^s, \ldots, CC_m^s) = \mathsf{BUH}_{\mathbf{K},L}(C_1^s, \ldots, C_m^s)$. The transcript is assumed to maximize the probability (over random choice of the hashing keys) that two of the PPs are equal or two of the CCs are equal. Call this event Coll. Then following the proof technique of [14], it is possible to show

$$\mathbf{Adv}^{\pm\mathrm{rnd}}_{\mathrm{BEB}[\mathrm{Perm}(n)]}(q) \leq (qm)^2/2^n + \mathsf{Pr}[\mathsf{Coll}].$$

By the blockwise universality of BUH, we have that for $(s,i) \neq (t,j)$, $\mathsf{Pr}[PP_i^s = PP_j^t] = 1/2^n$ and $\mathsf{Pr}[CC_i^s = CC_j^t] = 1/2^n$. Thus, the probability that two of the PPs are equal is at most $\binom{qm}{2}/2^n \leq (qm)^2/2^{n+1}$ and similarly for the probability that two of the CCs are equal. Combining these bounds, we obtain the required result. □

7 Extensions of LH

This section describes several extensions of the basic linear hash.

7.1 Decimated Linear Hash

We define a w-decimated linear hash family w-$\mathsf{DLH}[q,n,m]$. The parameter w is not present in the defintion of LH and the essential difference between $\mathsf{LH}[q,n,m]$ and w-$\mathsf{DLH}[q,n,m]$ is in the way the message is formatted and processed. Nevertheless, the details are substantially different and so we give the complete definition.

Fix a prime power q and positive integers n, m and w. The domain of w-$\mathsf{DLH}[q,n,m]$ is $\cup_{i=1}^{mnw} \mathbb{F}_q^i$ and the range is $\mathbb{F}_{q^n}$. The key space is $\mathbb{F}_{q^n}^{mw}$.

Let $\mathbf{M} \in \mathbb{F}_q^l$, $1 \leq l \leq mnw$, be a message in the domain. Let $l = l_1 nw + l_2$ with $1 \leq l_2 \leq nw$ and $l_2 = l_3 w + l_4 = l_4(l_3+1) + (w - l_4)l_3$ with $1 \leq l_4 \leq n$. Given n, w and l, the integers l_1, l_2, l_3

and l_4 are fixed. The message $\mathbf{M}$ consists of a sequence $\mathbf{M}_1, \ldots, \mathbf{M}_{(l_1+1)w}$, where

$$\left.\begin{array}{r} \mathbf{M}_1, \ldots, \mathbf{M}_{l_1 w} \in \mathbb{F}_q^n \\ \mathbf{M}_{l_1 w+1}, \ldots, \mathbf{M}_{l_1 w+l_4} \in \mathbb{F}_q^{l_3+1} \\ \mathbf{M}_{l_1 w+l_4+1}, \ldots, \mathbf{M}_{(l_1+1)w} \in \mathbb{F}_q^{l_3} \end{array}\right\} \tag{9}$$

The key $\mathbf{K}$ is formatted as $\mathbf{K} = (K_1, K_2, \ldots, K_{(l_1+1)w}, \ldots, K_{mw})$ where each $K_i \in \mathbb{F}_{q^n}$. Then, $w\text{-}\mathsf{DLH}_{\mathbf{K}}(\mathbf{M})$ is defined to be

$$w\text{-}\mathsf{DLH}_{\mathbf{K}}(\mathbf{M}) = G_{K_1}(\mathbf{M}_1) + G_{K_2}(\mathbf{M}_2) + \cdots + G_{K_{(l_1+1)w}}(\mathbf{M}_{(l_1+1)w}). \tag{10}$$

The security of this construction is similar to that of Theorem 1.

Theorem 6. *Let q be a prime power and n, m and w be positive integers. Then the differential probabilities of w-$\mathsf{DLH}[q, n, m]$ for equal length strings are equal to q^{-n}.*

The sum on the right hand side of (10) is to be evaluated w summands at a time. So, we consider the evaluation of $G_{K_1}(\mathbf{M}_1) + \cdots + G_{K_w}(\mathbf{M}_w)$. Let $\mathbf{M}_i = (M_{i,1}, \ldots, M_{i,n})$. Expanding, this is equal to

$$\left.\begin{array}{ccccccc} M_{1,1}K_1 & + & M_{1,2}\psi(K_1) & + \cdots + & M_{1,n}\psi^{n-1}(K_1) \\ M_{2,1}K_2 & + & M_{2,2}\psi(K_2) & + \cdots + & M_{2,n}\psi^{n-1}(K_2) \\ \cdots & \cdot & \cdots & \cdot \cdots \cdot & \cdots \\ M_{w,1}K_w & + & M_{w,2}\psi(K_w) & + \cdots + & M_{w,n}\psi^{n-1}(K_w). \end{array}\right\} \tag{11}$$

Define $\psi(K_1, \ldots, K_w) = (\psi(K_1), \ldots, \psi(K_w))$. Then

$$G_{K_1}(\mathbf{M}_1) + \cdots + G_{K_w}(\mathbf{M}_w) = \sum_{i=1}^{n} \langle (M_{1,i}, \ldots, M_{w,i}), \psi^{i-1}(K_1, \ldots, K_w) \rangle. \tag{12}$$

The advantage of the definition in (12) is that ψ can be simultaneously applied to all the $K_1, \ldots, K_w$.

Example: To make this idea concrete, we work with $q = 2$ and $w = 32$. Each key K_i is given by an n-bit string, i.e., $K_i = (K_{i,1}, \ldots, K_{i,n})$. Let ψ be realized using a binary LFSR $\mathcal{L}$ of length n. For example, we can use the first row of Table 1. For $1 \le i \le n$, let $\mathbf{k}_i = (K_{1,i}, K_{2,i}, \ldots, K_{w,i})$ and $\mathbf{m}_i = (M_{1,i}, M_{2,i}, \ldots, M_{w,i})$. Each $\mathbf{k}_i$ fits into a w-bit word. Then $\psi(K_1, \ldots, K_w)$ is found by applying $\mathcal{L}$ *once* to $(\mathbf{k}_1, \ldots, \mathbf{k}_n)$ which we denote by $\mathcal{L}(\mathbf{k}_1, \ldots, \mathbf{k}_n)$. Applying $\mathcal{L}$ to $(\mathbf{k}_1, \ldots, \mathbf{k}_n)$ means the following. Consider $(\mathbf{k}_1, \ldots, \mathbf{k}_n)$ to be a 32×128 matrix of bits and apply $\mathcal{L}$ to each row of this matrix. The advantage is that it is possible to operate on 32-bit words at a time. So, 32 clockings of $\mathcal{L}$ can be completed by a single clocking of $\mathcal{L}$ on 32-bit words. If $\tau(x)$ is a trinomial or a pentanomial, then $\mathcal{L}(\mathbf{k}_1, \ldots, \mathbf{k}_n)$ can be computed using either 2 or 4 XOR operations on w-bit strings.

After applying $\mathcal{L}$ we need to compute the inner product. If $\mathbf{a}$ and $\mathbf{b}_1, \ldots, \mathbf{b}_n$ are w-bit strings, then by $\langle \mathbf{a}, (\mathbf{b}_1, \ldots, \mathbf{b}_n) \rangle$ we denote $(\langle \mathbf{a}, \mathbf{b}_1 \rangle, \ldots, \langle \mathbf{a}, \mathbf{b}_n \rangle)$. Then,

$$G_{K_1}(\mathbf{M}_1) + \cdots + G_{K_w}(\mathbf{M}_w) = \sum_{i=1}^{n} \langle \mathbf{m}_i, \mathcal{L}^{i-1}(\mathbf{k}_1, \ldots, \mathbf{k}_n) \rangle.$$

The cost per w bits of the message is the following. A single application of $\mathcal{L}$ on $(\mathbf{k}_1, \ldots, \mathbf{k}_n)$ and the computation (in parallel) of the n inner products of w-bit strings. Essentially, we are working in a bit-slice manner over the columns of (11). For hardware implementation, this will lead to a significant speed-up.

This can also be useful for software implementation. Basically, the application of $\mathcal{L}$ will be very efficient. However, processors do not have inner product operations as basic operations and so these will form the main bottleneck. On the other hand, for CPUs which support vector operations, DLH will be very fast.

7.2 The Sliding Window (Toeplitz) Construction

The differential (and hence collision) probabilities of LH and DLH are equal to q^{-n}. By suitably choosing q and n, this can be made as low as one desires. On the other hand, it is also possible to introduce additional flexibility based on repeated hashing. We describe the idea for LH. A similar variation can be suggested for DLH.

The idea is the following. Repeatedly hash the same message with independent keys and concatenate the output. If the message is hashed s times, then the collision probability becomes q^{-ns}. But, this approach requires s independent keys. The modification given in [5] is to generate s keys using a sliding window technique. Suppose each hash call requires a key sequence of length t. We start with a sequence of length $s + t - 1$ and slide (one element at a time) a window of length t over this sequence to obtain s different keys each of length t. This is called the Toeplitz procedure as the process can be visualized as a Toeplitz matrix. The same idea works for LH.

We define the hash function family $\mathsf{LH}^{\mathsf{T}}[q, n, m, s]$, where q, n and m are as in Section 3.3 and s is a positive integer. The value $t = \lceil m/n \rceil$, the domain A and the details of message parsing are the same as in Section 3.3. The range B is now an element of $\mathbb{F}_{q^n}^s$. A function in $\mathsf{LH}^{\mathsf{T}}[q, n, m, s]$ is named by a key $\mathbf{K} = (K_1, \ldots, K_{t+s-1}) \in \mathbb{F}_{q^n}^{t+s-1}$. For $1 \leq i \leq s$, let $\mathbf{K}_i = (K_i, K_{i+1}, \ldots, K_{t+i-1})$. For any $M \in A$, we define $\mathsf{LH}^{\mathsf{T}}_{\mathbf{K}}(\mathbf{M})$ as

$$\mathsf{LH}^{\mathsf{T}}_{\mathbf{K}}(\mathbf{M}) = (\mathsf{LH}_{\mathbf{K}_1}(\mathbf{M}), \ldots, \mathsf{LH}_{\mathbf{K}_s}(\mathbf{M})). \tag{13}$$

Theorem 7. *Let q be a prime power and n, m and s be positive integers. Then the differential probabilities of $\mathsf{LH}^{\mathsf{T}}[q, n, m, s]$ for equal length strings are equal to q^{-ns}.*

Proof: As before, since LH^{T} is linear, it is sufficient to show that for any non-zero $\mathbf{M} \in A$ and for any fixed $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_s)$ in $\mathbb{F}_{q^n}^s$, $\Pr_{\mathbf{K}}[\mathsf{LH}^{\mathsf{T}}_{\mathbf{K}}(\mathbf{M}) = \boldsymbol{\alpha}] = q^{-ns}$.

Parse $\mathbf{M}$ as $\mathbf{M}_1, \ldots, \mathbf{M}_{l_1}, \mathbf{M}_{l_1+1}$ as in Section 3.3. Since $\mathbf{M}$ is non-zero, at least one of the $\mathbf{M}_i$'s will be non-zero. Let r be the maximum integer such that $\mathbf{M}_r$ is non-zero. The condition $\mathsf{LH}^{\mathsf{T}}_{\mathbf{K}}(\mathbf{M}) = \boldsymbol{\alpha}$ is equivalent to

$$\left.\begin{array}{rcl} \mathsf{LH}_{\mathbf{K}_1}(\mathbf{M}) = & G_{K_1}(\mathbf{M}_1) + \cdots + G_{K_{r-1}}(\mathbf{M}_{r-1}) + G_{K_r}(\mathbf{M}_r) & = \alpha_1 \\ \mathsf{LH}_{\mathbf{K}_2}(\mathbf{M}) = & G_{K_2}(\mathbf{M}_1) + \cdots + G_{K_r}(\mathbf{M}_{r-1}) + G_{K_{r+1}}(\mathbf{M}_r) & = \alpha_2 \\ \vdots\ \vdots & \vdots & \vdots\ \vdots \\ \mathsf{LH}_{\mathbf{K}_s}(\mathbf{M}) = & G_{K_s}(\mathbf{M}_1) + \cdots + G_{K_{r+s-2}}(\mathbf{M}_{r-1}) + G_{K_{r+s-1}}(\mathbf{M}_r) & = \alpha_s \end{array}\right\} \tag{14}$$

By Lemma 3, each $G_{K_i}(\mathbf{M}_j)$ is uniformly distributed over $\mathbb{F}_{q^n}$. Also, since the K_is are independent and $\mathbf{M}_r$ is non-zero,

$$\begin{aligned}
\Pr[G_{K_r}(\mathbf{M}_r)] &= \Pr[G_{K_r}(\mathbf{M}_r)|G_{K_1}(\mathbf{M}_1),\ldots,G_{K_{r-1}}(\mathbf{M}_{r-1})];\\
\Pr[G_{K_{r+1}}(\mathbf{M}_r)] &= \Pr[G_{K_{r+1}}(\mathbf{M}_r)|G_{K_1}(\mathbf{M}_1),\ldots,G_{K_r}(\mathbf{M}_r),\, G_{K_2}(\mathbf{M}_1),\ldots,G_{K_r}(\mathbf{M}_{r-1})];\\
&\cdots\ \cdot\ \cdots\\
\Pr[G_{K_{r+s-1}}(\mathbf{M}_r)] &= \Pr[G_{K_{r+s-1}}(\mathbf{M}_r)|G_{K_1}(\mathbf{M}_1),\ldots,G_{K_r}(\mathbf{M}_r),\, G_{K_2}(\mathbf{M}_1),\ldots,G_{K_{r+1}}(\mathbf{M}_r),\\
&\qquad\qquad \ldots,\, G_{K_s}(\mathbf{M}_1),\ldots,G_{K_{r+s-2}}(\mathbf{M}_{r-1})].
\end{aligned}$$

Using this it is easy to see that the events given by the rows of (14) are independent and that each row holds with probability q^{-n}. Since there are s rows, the probability that all the rows hold is q^{-ns}. □

Example: Consider $\mathsf{LH}^{\mathsf{T}}[2, 32, m, 4]$, i.e., $q = 2$, $n = 32$ and $s = 4$. Then the output is of size $4 \times 32 = 128$ bits and the differential probabilities are equal to 2^{-128}. The linear operator ψ is implemented using an LFSR of length 32 with $\tau(x) = 1 + x^2 + x^3 + x^7 + x^{32}$. For each bit of the message, we require 4 clockings of a 32-bit LFSR. These clockings are on independent keys and can be done in parallel. We provide some more details.

The message consists of upto m bits and $t = \lceil m/32 \rceil$. The entire key sequence is $\mathbf{K} = (K_1, \ldots, K_t, K_{t+1}, K_{t+2}, K_{t+3})$ and the four keys for the four different hash applications are

$$(K_1, \ldots, K_t),\ (K_2, \ldots, K_{t+1}),\ (K_3, \ldots, K_{t+2}) \text{ and } (K_4, \ldots, K_{t+3}).$$

As before, let $\psi(a, b, c, d) = (\psi(a), \psi(b), \psi(c), \psi(d))$. Then, $\mathsf{LH}^{\mathsf{T}}_{\mathbf{K}}(\mathbf{M})$, where $\mathbf{M} = (M_1, \ldots, M_l)$ with $1 \le l \le m$ is an l-bit message, is computed as

$$\sum_{i=1}^{32} M_i \psi(K_1, K_2, K_3, K_4) + \sum_{i=33}^{64} M_i \psi(K_2, K_3, K_4, K_5) + \cdots$$

Since each M_i is a bit, multiplication by M_i takes no time. The computation per message bit is the application of ψ to the four K_is. Further, ψ is applied simultaneously to four independent keys. This is a typical single-instruction multiple-data (SIMD) senario providing opportunities for parallelism.

For the example above, one can also consider word oriented LFSRs (see Section 4). For example, to achieve collision probability of 2^{-128}, one can take $s = 2$ and implement ψ as a linear map from $\mathbb{F}_{2^{64}}$ to itself by using a word oriented LFSR of length 4 over $\mathbb{F}_{2^{16}}$. Choice of a suitable value of s and the implementation of ψ will depend on the application platform and the resource constraints.

7.3 Multiple Linear Operators

The function family LH as well as the extensions DLH and LH^{T} use one linear operator ψ. In this section, we show that this is a special case of a more general construction which uses multiple linear operators. Using multiple linear operators does not affect (i.e., either improve of reduce) the key size and efficiency. It does, however, provide additional flexibility.

As in the case of LH, we first define a basic hash function which works on bounded length messages and then show how to extend it to arbitrary length messages.

Let q and n be as before. Let $r \ge 1$ be a positive integer and let $\psi_1, \ldots, \psi_r$ be linear operators from $\mathbb{F}_{q^n}$ to itself having $\tau_1(x), \ldots, \tau_r(x)$ as minimal polynomials. We require these to be distinct irreducible polynomials over $\mathbb{F}_q$ each having degree n. For every $\mathbf{K} = (K_1, \ldots, K_r) \in \mathbb{F}^r_{q^n}$, we define

a function $H_{\mathbf{K}} : \cup_{i=1}^{nr} \mathbb{F}_q^i \rightarrow \mathbb{F}_{q^n}$ in the following manner. Let $\mathbf{a} = (a_1, \ldots, a_l) \in \mathbb{F}_q^l$, $1 \leq l \leq nr$. Then

$$H_{\mathbf{K}}(\mathbf{a}) = \sum_{i=1}^{l} a_i(\psi_1^{i-1}(K_1) + \cdots + \psi_r^{i-1}(K_r)). \tag{15}$$

The following result states the basic properties of H.

Lemma 4. *Fix an l with $1 \leq l \leq nr$. The following are true for the function H defined in (15).*

1. *For a fixed $\mathbf{K}$, the function $H_{\mathbf{K}}$ restricted to inputs with l components, is a multi-linear function, i.e., it is linear in every component of its input.*
2. *Fix a non-zero $\mathbf{a} \in \mathbb{F}_q^l$. If $\mathbf{K}$ is uniformly distributed over $\mathbb{F}_{q^n}^r$ then $H_{\mathbf{K}}(\mathbf{a})$ is uniformly distributed over $\mathbb{F}_{q^n}$.*
3. *Consequently, for $\mathbf{a}, \mathbf{a}' \in \mathbb{F}_q^l$, $1 \leq l \leq nr$, $\mathbf{a} \neq \mathbf{a}'$ and for any $\alpha \in \mathbb{F}_{q^n}$, $\Pr_{\mathbf{K}}[H_{\mathbf{K}}(\mathbf{a}) - H_{\mathbf{K}}(\mathbf{a}') = \alpha] = 1/q^n$.*

Proof: The multi-linear property is easy to see. Also, due to linearity, the third statement follows directly from the second statement. So, we only have to prove the second statement. Define the polynomial $p(x) \in \mathbb{F}_q[x]$ to be $p(x) = a_1 x + a_2 x^2 + \cdots + a_l x^l$. Since $\mathbf{a}$ is non-zero, so is $p(x)$.

For $1 \leq i \leq r$, let $p_i(x) = p(x) \bmod \tau_i(x)$. We claim that all the $p_i(x)$ cannot be zeros. Suppose not, then each $\tau_i(x)$ divides $p(x)$ and since the $\tau_i(x)$s are distinct irreducible polynomials, it follows that, $\tau_1(x) \cdots \tau_r(x)$ divides $p(x)$. Now degree of $p(x)$ is at most nr and the constant term of $p(x)$ is zero. So, we can write $p(x) = xp_1(x)$, where $p_1(x)$ is a polynomial of degree less than nr. Since $\tau_1(x) \cdots \tau_r(x)$ is co-prime to x and $\tau_1(x) \cdots \tau_r(x)$ divides $xp_1(x)$, it follows that $\tau_1(x) \cdots \tau_r(x)$ divides $p_1(x)$. But, the degree of $\tau_1(x) \cdots \tau_r(x)$ is nr whereas the degree of $p_1(x)$ is at most $nr - 1$. Thus, we obtain a contradiction.

So, some $p_i(x)$ is non-zero. By Lemma 1, $p_i(\psi_i)$ is an invertible map from $\mathbb{F}_{q^n}$ to itself. Since $\mathbf{K}$ is uniformly distributed over $\mathbb{F}_{q^n}^r$, K_i is uniformly distributed over $\mathbb{F}_{q^n}$ and so is $p_i(\psi_i)(K_i)$. Further, since K_j, $j \neq i$, is independent of K_i, it follows that $H_{\mathbf{K}}(\mathbf{a})$ is uniformly distributed over $\mathbb{F}_{q^n}$. □

Note: The function G_K (defined in Section 3.2) can ensure AXU property for equal length messages consisting of upto n elements. On the other hand, $H_{\mathbf{K}}$ ensures the AXU property for equal length messages consisting of upto nr elements. G_K uses one linear operator ψ, while $H_{\mathbf{K}}$ uses r linear operators. If $r = 1$, then $H_{\mathbf{K}}$ reduces to G_K. Thus, $H_{\mathbf{K}}$ is a proper generalization of G_K.

We next define a function family which is parametrized as $\mathsf{MLH}[q, n, r, m]$. The domain is $A = \cup_{i=1}^{m} \mathbb{F}_q^i$ and the range is $B = \mathbb{F}_{q^n}$. Let $t = \lceil m/(nr) \rceil$. Each function in $\mathsf{MLH}[q, n, r, m]$ is named by an element $\mathbf{K} = (\mathbf{K}_1, \ldots, \mathbf{K}_t)$ with $\mathbf{K}_i \in \mathbb{F}_{q^n}$; a random function in $\mathsf{MLH}[q, n, r, m]$ is given by a random $\mathbf{K}$. We write the function indicated by $\mathbf{K}$ as $\mathsf{MLH}_{\mathbf{K}}(\cdot)$.

The message $\mathbf{M}$ consists of an l-tuple, $l \leq m$, of elements over $\mathbb{F}_q$. Let $l = l_1(nr) + l_2$, with $1 \leq l_2 \leq nr$. We consider $\mathbf{M}$ to be of the form $(\mathbf{M}_1, \ldots, \mathbf{M}_{l_1}, \mathbf{M}_{l_1+1})$, where $\mathbf{M}_1, \ldots, \mathbf{M}_{l_1}$ are from $\mathbb{F}_q^{nr}$ and $\mathbf{M}_{l_1+1}$ is in $\mathbb{F}_q^{l_2}$. Further, let $\mathbf{K} = (\mathbf{K}_1, \ldots, \mathbf{K}_s)$, $\mathbf{K}_i \in \mathbb{F}_{q^n}^r$, and note that $s \geq l_1 + 1$. Then, $\mathsf{MLH}_{\mathbf{K}}(\mathbf{M})$ is defined as

$$\mathsf{MLH}_{\mathbf{K}}(\mathbf{M}) = H_{\mathbf{K}_1}(\mathbf{M}_1) + H_{\mathbf{K}_2}(\mathbf{M}_2) + \cdots + H_{\mathbf{K}_{l_1}}(\mathbf{M}_{l_1}) + H_{\mathbf{K}_{l_1+1}}(\mathbf{M}_{l_1+1}). \tag{16}$$

The proof of the next result follows from Lemma 4 in a manner similar to the way in which Theorem 1 follows from Lemma 3.

Theorem 8. *For any prime power q, positive integers n and m, the differential probabilities of* $\mathsf{MLH}[q, n, r, m]$ *for equal length strings are equal to* q^{-n}.

8 Conclusion

We have presented a new method of constructing families of multi-linear universal hash functions by working over a finite field $\mathbb{F}_q$ and its extension $\mathbb{F}_{q^n}$. Using $q = 2$ and a tower field representation of $\mathbb{F}_{q^n}$, leads to an algorithm with a very small footprint in hardware and reasonable performance in software.

A new MAC algorithm is obtained by combining the new universal hash function with a stream cipher with IV. The basic universal hash function is extended into an invertible blockwise universal hash function. This is then used to construct a tweakable enciphering scheme BEB which has applications to disk encryption algorithm. The novel feature of BEB is that it uses a single layer of encryption and no finite field multiplications. Both the MAC algorithm and BEB are well suited for resource constrained applications.

References

1. Côme Berbain and Henri Gilbert. On the security of IV dependent stream ciphers. In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 254–273. Springer, 2007.
2. Daniel J. Bernstein. The Poly1305-AES message-authentication code. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 32–49. Springer, 2005.
3. Daniel J. Bernstein. Stronger security bounds for Wegman-Carter-Shoup authenticators. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 164–180. Springer, 2005.
4. Jürgen Bierbrauer, Thomas Johansson, Gregory Kabatianskii, and Ben J. M. Smeets. On families of hash functions via geometric codes and concatenation. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 331–342. Springer, 1993.
5. John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. UMAC: Fast and secure message authentication. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 216–233. Springer, 1999.
6. Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.
7. Florent Chabaud and Reynald Lercier. PIPS, the Primitive and Irreducible Polynomial Server and ZEN, A toolbox for fast computation in finite extension over finite rings. `http://fchabaud.free.fr/English/default.php?COUNT=1\&FILE0=Poly`.
8. Debrup Chakraborty and Palash Sarkar. HCH: A new tweakable enciphering scheme using the hash-encrypt-hash approach. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 287–302. Springer, 2006. full version available at `http://eprint.iacr.org/2007/028`.
9. Patrik Ekdahl and Thomas Johansson. A new version of the stream cipher SNOW. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2002.
10. Edgar N. Gilbert, F. Jessie MacWilliams, and Neil J. A. Sloane. Codes which detect deception. *Bell System Technical Journal*, 53:405–424, 1974.
11. Shai Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 315–327. Springer, 2004.
12. Shai Halevi. Invertible universal hashing and the TET encryption mode. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 412–429. Springer, 2007.
13. Shai Halevi and Hugo Krawczyk. MMH: Software message authentication in the gbit/second rates. In Eli Biham, editor, *Fast Software Encryption*, volume 1267 of *Lecture Notes in Computer Science*, pages 172–189. Springer, 1997.

14. Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 482–499. Springer, 2003.
15. Shai Halevi and Phillip Rogaway. A parallelizable enciphering mode. In Tatsuaki Okamoto, editor, *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 2004.
16. Thomas Johansson. A shift register construction of unconditionally secure authentication codes. *Des. Codes Cryptography*, 4(1):69–81, 1994.
17. Hugo Krawczyk. LFSR-based hashing and authentication. In Yvo Desmedt, editor, *CRYPTO*, volume 839 of *Lecture Notes in Computer Science*, pages 129–139. Springer, 1994.
18. Ted Krovetz and Phillip Rogaway. Fast universal hashing with small keys and no preprocessing: The polyr construction. In Dongho Won, editor, *ICISC*, volume 2015 of *Lecture Notes in Computer Science*, pages 73–89. Springer, 2000.
19. R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications, revised edition.* Cambridge University Press, 1994.
20. David A. McGrew and Scott R. Fluhrer. The extended codebook (XCB) mode of operation. Cryptology ePrint Archive, Report 2004/278, 2004. `http://eprint.iacr.org/`.
21. Alfred Menezes, Paul Van Oorschot, and Scott Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
22. Moni Naor and Omer Reingold. On the construction of pseudorandom permutations: Luby-Rackoff revisited. *J. Cryptology*, 12(1):29–66, 1999.
23. Phillip Rogaway. Bucket hashing and its application to fast message authentication. *J. Cryptology*, 12(2):91–115, 1999.
24. Palash Sarkar. Improving upon the TET mode of operation. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *ICISC*, volume 4817 of *Lecture Notes in Computer Science*, pages 180–192. Springer, 2007.
25. Victor Shoup. On fast and provably secure message authentication based on universal hashing. In Neal Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 313–328. Springer, 1996.
26. Douglas R. Stinson. Universal hashing and authentication codes. *Des. Codes Cryptography*, 4(4):369–380, 1994.
27. Peng Wang, Dengguo Feng, and Wenling Wu. HCTR: A variable-input-length enciphering mode. In Dengguo Feng, Dongdai Lin, and Moti Yung, editors, *CISC*, volume 3822 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2005.
28. Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.