On the construction of digest functions for manual authentication protocols

Long Hoang Nguyen and Andrew William Roscoe

Oxford University Department of Computer Science
Parks Road, Oxford, OX1 3QD
Email: {Long.Nguyen,Bill.Roscoe}@cs.ox.ac.uk

Abstract

A digest function is a sort of universal hash that takes a key and a message as its inputs. This paper will study these functions' properties and design in the context of their application in manual authentication technology. Frequently a digest function needs to have a very short output (e.g. 16–32 bits) and no key is used to digest more than one message. These together with other characteristics represent a new kind of game played between an attacker and honest parties, which is very different from other authentication mechanisms, notably message authentication codes or MACs.

Short digests can be constructed directly or by "condensing" longer functions. We offer an improved method for the latter but concentrate mainly on direct constructions. We propose a digest algorithm which uses word multiplications to obtain a very fast implementation. This digest scheme enjoys strong and provable security properties, namely for a single-word or b-bit output digest function the collision probability is $\epsilon = 2^{1-b}$ on equal and arbitrarily length inputs. The scheme is related to the multiplicative universal hash function of Dietzfelbinger et al., and it improves on several well-studied and efficient universal hashing algorithms, including MMH and NH.

1 Motivation and contribution

We investigate the design, construction and security of a new cryptographic primitive termed a *digest function*, whose specification arises from its use in manual authentication technology [4, 16, 27, 36, 37, 38, 39]. A digest function $digest(k, m)$, which takes a key k and a message m, has similarities to both ϵ-balanced and ϵ-almost universal hash functions [26, 53]. However, the majority of uses of this function in practice require it to have a very short output (16–32 bits as in a password), and thus our constructions introduced here are designed to take advantage of this feature, namely this feature opens the way for efficient and parallelisable constructions as opposed to the cascade structure underlying many long-output (universal) hash functions.

Although other similar cryptographic primitives such as short-output universal hash functions MMH [18] and NH [9] have been designed and used to build message authentication codes, we note that these short-output primitives are not often used on their own in cryptographic mechanisms. In this paper we will focus on the application and security properties of digest functions in manual authentication protocols, which use this function together with existing human trust and interactions to authenticate data without the need for PKI, shared private keys and passwords. Here is an example of how this technology works: for electronic

devices to agree on the same (and potentially large) data that have been exchanged over an insecure and high-bandwidth medium, their human owners need to manually compare a short and non-secret digest of both the data and some key.[1] Since human comparison of a digest via conversations, phones or text messages is time-consuming, a digest function must have a very short output. As this technology has many application in, for example, online payments, e-heathcare and bootstrapping of ad hoc networks, it has been rapidly developed by many researchers and standardised by both ISO and IETF in the last decade. Examples include the Bluetooth v2.1 [4, 29], MANA I [16, 35, 36], ZRTP [61], schemes of Laur and Nyberg [27, 28], Pasini and Vaudenay [41], and the (S)HCBK protocols of the authors [8, 36, 37, 38, 39, 43, 44, 45, 46, 47].

We will see in Section 2 that both security and efficiency properties of a digest function are dictated by a new game played between an attacker and trustworthy parties in these protocols, which is very different in a number of aspects from conventional attacking opportunities present in other authentication mechanisms, such as message authentication codes. For example, the digest key always varies randomly from one to another protocol run, i.e. no key value is used to digest more than one message, and hence attacks which rely on the reuse of a single private key in multiple sessions (as in MACs or any ciphers) are irrelevant. Moreover, the digest key is always revealed and known to everyone (including the intruder) prior to the computation of a digest value in these protocols, and consequently the process of expanding this key into a long keystream via a pseudorandom number generator (PRNG) before being input into a digest function need not to be cryptographically secure. This has a great impact on the efficiency of digest computation because an ordinary PRNG is normally faster than a cryptographically secure one. We note that there have been several papers [16, 41, 37, 38] in which manual authentication protocols have been analysed. However, neither do they study the differences between these protocols and MACs nor rigorously investigate the influence of the interactions (or the new game) between protocol participants and the intruder on the security properties that a digest function must satisfy.

A short digest can be directly constructed or by condensing longer functions. We will give a brief survey on the latter in Section 4 where we present our improvement to one of the existing schemes as well as pointing out their disadvantages. Our discussion on direct constructions starts with the Toeplitz matrix algorithm of Krawczyk [25] and Mansour et al. [30], and then expands to all of Sections 5 and 6.

The main contribution of this paper is a new digest algorithm which has similarity to the well-studied multiplicative universal hashing scheme of Dietzfelbinger et al. [14]. Our construction uses word multiplications which are fast in any modern microprocessor to increase computational efficiency while retains the same level of security. In particular, for a single-word or b-bit output digest scheme, the security proof of this construction demonstrates that the collision probability is $\epsilon_c = \mathbf{2^{1-b}}$. Since there are two other related universal hash functions called MMH of Halevi and Krawczyk [18] and NH of Black et al. [9], we will assess their suitability for use in manual authentication protocols as well as compare them against ours in Section 6. While the underlying ideas of MMH and NH are different from our digest scheme, i.e. they are based on techniques due to Carter and Wegman [12] instead of multiplicative universal hashing, they also make use of word multiplications to have a very good performance in computation. They however obtain lower levels of security: (1) for a b-bit output MMH: $\epsilon_c = \mathbf{6} \times \mathbf{2^{-b}}$, and (2) for a b-bit output NH, $\epsilon_c = \mathbf{2^{-b/2}}$.

[1] There are protocols such as schemes of Vaudenay [57], Mashatan and Stinson [31] which do not use a digest or universal hash function. However the majority of others require the specification of a digest function.

It is important to stress that our digest scheme can be efficiently generalised to a multiple-word or long-output scheme without increasing the word length that is constrained by architecture characteristics. The generalised scheme enjoys the best bound for collision probability one could hope for, i.e. for a n-word or nb-bit digest scheme the collision probability is 2^{n-nb}. Consequently, the digest algorithm can be plugged in to construct message authentication codes and a new type of digital certificate termed *FlexiMAC* [39] that is significantly cheaper to check as well as other computer science applications, including randomised algorithm [14]. The description and security proof for this generalised version are provided in subsection 5.3.

We end this paper by reporting the implementation results of our digest construction as well as MMH, NH and SHA family of hash functions on the same platform: 1GHz AMD Athlon(tm) 64 X2 Dual Core Processor (4600+ or 512 KB caches). With $\epsilon_c = 2^{-31}$ chance of collision the digest scheme achieves peak performance of 1.57 cycles/byte, which is comparable with 1.33 cycles/byte of MMH (for $\epsilon_c = 6 \times 2^{-32}$) and 1.25 cycles/byte of NH (for $\epsilon_c = 2^{-32}$ but the output length of NH is twice of MMH and digest). Please note that these figures include the cost of both universal hashing computation and key expansion via a pseudorandom number generator, the latter was not included in the previously recorded speeds for MMH [18] and NH [9]. For comparison, our SHA1 implementation runs at 5.78 cycles/byte.

2 Characteristics of the new game

In this section, we will describe the main requirements of a digest function by outlining major differences between manual authentication protocols and MACs, both of which seek to authenticate data. This comparison focuses on four aspects: human interactions, key distribution or agreement, key usage, and key expansion. To illustrate the differences, we frequently refer to the following scheme[2] as an example, though our analysis here applies to the majority of other oneway, pairwise and group manual authentication protocols in the literature [4, 27, 28, 37, 38, 41, 58]. We end this section by summarising the comparison in Table 1. The exact definition of a digest function is given in Section 3.

In the following scheme, parties A and B want to authenticate their public data $m_{A/B}$ to each other without the need for passwords, private keys, or pre-established security infrastructures such as a PKI. $m_{A/B}$ include public keys, images or videos, and so can be of significant size. The single arrow ($\longrightarrow$) indicates an unreliable and high-bandwidth link where messages can be maliciously altered, whereas the double arrow ($\Longrightarrow$) represents an authentic and unspoofable channel. The latter is not a private channel (anyone can overhear it) and it is usually very low-bandwidth since it is implemented by humans, e.g., human conversations or manual data transfers between devices. $hash()$ is a cryptographic hash function. Long random keys $k_{A/B}$ generated by A/B are kept secret until the end of Messages 1–2. Operators $\|$ and $\oplus$ denote bitwise concatenation and exclusive-or.

A pairwise authentication protocol [4, 27, 29, 36, 37, 38]		
$1. A \longrightarrow B$	:	$m_A, hash(A \parallel k_A)$
$2. B \longrightarrow A$	:	$m_B, hash(B \parallel k_B)$
$3. A \longrightarrow B$	:	k_A
$4. B \longrightarrow A$	:	k_B
$5. A \Longleftrightarrow B$	:	$digest(k_A \oplus k_B, m_A \parallel m_B)$

[2]This is the pairwise version of the SHCBK protocol [37, 38]. It also closely resembles the Bluetooth v2.1 [4, 29] and the protocol of Laur any Nyberg [27].

Human interactions and security of the protocol: To ensure both devices agree on the same data, their human owners have to compare a digest value manually in Message 5. As human interactions are expensive, the digest function needs to have a short output of $b = 16$–32 digits. Also regardless of what the intruder does with Messages 1–4, the digest key $k^* = k_A \oplus k_B$, which is computed at each node and is instrumental in the computation of the digest value $digest(k^*, m_A \parallel m_B)$, always varies from one to another run in a way that cannot be influenced by an attacker. This is because each party's digest key is randomised by its own subkey, due to the $\oplus$ operator. What we then require of such a digest function is the resistance against a *digest collision attack*. A digest collision for a pair of different messages m and m' is a key k such that $digest(k, m) = digest(k, m')$.

Key distribution and agreement: In the symmetric cryptographic world, we usually assume that parties share a long private key in advance, and no-one could have influenced its creation and distribution. In contrast, as seen in the above scheme, the establishment of the digest key $k^* = k_A \oplus k_B$ is part of the protocol itself, and therefore potentially open to attacks. Those protocols must ensure that, invariably, the digest key remains completely unknown to anyone until very late in the protocol, specifically after everyone's view of $m_{A/B}$ is committed. This is a major advantage because no attacker can manipulate $m_{A/B}$ data by reference to what it might know about the digest key. This is in contrast to attacking strategies on symmetric cryptographic primitives as will be discussed below.

Key usage: The majority of MAC schemes and also ciphers use the same private key to encrypt or hash multiple messages for a long period of time.[3] This opens the way for cryptanalysis as well as adaptive chosen plaintext and ciphertext attacks. As explained previously, no key value is used to digest more than one input message in the new game. The key always varies randomly from one to another protocols, and hence there is no opportunity for traditional cryptanalysis. To some extent, this type of protocols have similarities to password-based schemes where off-line searches are made irrelevant, i.e. the only way to find out a guess of a password is correct is to interact with the protocol. However passwords are often unchanged, and so the chance of a successful attack increases as more attempts are launched to guess the passwords. This is not the case in the above protocol since the digest key is always random and fresh, and so the chance of a successful attack remains unchanged regardless of how many times an attack is launched. Crucially this explains why manual authentication protocols can resist collision attacks even though the digest is short.

Key expansion and pseudorandom number generator (PRNG): All digest and universal hash (MMH and NH) constructions considered here require the key length to be comparable with the message length, which can be very long for large data. One therefore usually generates such a long keystream out of a shorter digest key via a PRNG prior to digest computation, e.g. in Message 5 of the above protocol. Similarly both MAC algorithms and ciphers use shared private keys to derive long keystreams or multiple round keys. However, to generate keystreams as in MACs and the one-time padding scheme, the PRNG must pass the next-bit test (or be unpredictable), and such a PRNG is called a Cryptographically Secure

[3] Although researchers [19] have suggested that we should avoid reusing universal hash keys in MAC schemes, and such an approach has been taken by SNOW 3G [3], both parties still need to agree on the same long-term secret which is used to generate a new keystream each time a MAC is computed. The problem of recovering reused universal hash keys is therefore reduced to the problem of recovering long-term secrets.

	MACs	Manual authentication protocols
Human interactions	None ⇒ Long-output primitives	Compare short authentication strings ⇒ Short-output (digest) functions
Key generation, distribution, and agreement	Generated by trusted parties Distributed in advance ⇒ Unbiased and no attacks	Jointly agreed among parties Not agreed until the end of protocol ⇒ Unbiased but open to collision attacks
Key usage	Reused Revocation and replacement ⇒ Cryptanalysis	Never used more than once Always random and fresh ⇒ No cryptanalysis and no off-line search
Key expansion	Must pass the next-bit test ⇒ CS-PRNG	Do not need to pass the next-bit test ⇒ Good quality (ordinary) PRNG

Table 1: A summary of the differences between MACs and the new game.

PRNG (CS-PRNG).[4]

> **The next-bit test** [32] (or unpredictability): A pseudorandom bit generator is said to pass the next-bit test if there is no polynomial-time algorithm which, on input of the first l bits of an output sequence s, can predict the $(l+1)^{st}$ bit of s with probability significantly greater than $1/2$.

In contrast, the digest key in manual authentication protocols is always revealed and known to everyone prior to digest computation (as seen in Messages 3–4 of the above protocol). Clearly there is no point for the PRNG which expands this *non-secret* key into a long keystream to pass the next-bit test, because the pseudorandom keystream is predictable from the digest key anyway. What we require of this PRNG is a long period and good statistical properties, and such a PRNG is well-studied and efficiently implemented in practice, including linear congruential generators, linear or generalised feedback shift registers. We note that the security of a CS-PRNG often relies on the presumed intractability of an underlying number-theoretic problem, such as the factoring and RSA problems, which use modular arithmetics; thus CS-PRNGs are relatively slow compared to normal PRNGs [32]. As will be demonstrated in subsection 6.3, this will have a major effect on the speed and usability of not only our proposed constructions but also other similar primitives used in manual authentication protocols.

3 Notation and definitions

We define M, K and b the bitlength of message, key and output in a digest or universal hash function. We denote $R = \{0,1\}^K$, $X = \{0,1\}^M$ and $Y = \{0,1\}^b$.

Definition 1. [26] A ϵ-balanced universal hash function, $h : R \times X \to Y$, must satisfy that for every $m \in X \setminus \{0\}$ and $y \in Y$: $\Pr_{\{k \in R\}}[h(k,m) = y] \leq \epsilon$

[4]CS-PRNG is also required to resist a *state-compromise* attack: given the knowledge of the internal state of a PRNG or the last few bits, it is infeasible to reconstruct the preceding bits of the sequence [60].

Definition 2. [26] A ϵ-almost universal hash function, $h : R \times X \to Y$, must satisfy that for every $m, m' \in X$ $(m \neq m')$: $\Pr_{\{k \in R\}}[h(k, m) = h(k, m')] \leq \epsilon$

Combining definitions 1 and 2, we define an (ϵ_d, ϵ_c)-balanced digest function as follows. We note that the majority of manual authentication protocols only need the second requirement of a digest function, but some of those also need the first one.

Definition 3. A (ϵ_d, ϵ_c)-balanced digest function, $digest : R \times X \to Y$, must satisfy

- for every $m \in X \setminus \{0\}$ and $y \in Y$: $\Pr_{\{k \in R\}}[digest(k, m) = y] \leq \epsilon_d$
- for every $m, m' \in X$ $(m \neq m')$: $\Pr_{\{k \in R\}}[digest(k, m) = digest(k, m')] \leq \epsilon_c$

4 Existing digest constructions

There have been a number of publications [4, 16, 41] in which functions playing the role of a digest function have been defined. However, they mainly use methods which create a long hash of authentic data and then condense it to a short digest value. Although we make improvement to one of these, we will show that this strategy suffers from either lack of rigorous proof of security or computational inefficiency.

Since this paper concentrates on direct constructions of digest functions, we first analyse a method based on Toeplitz matrix which directly produces a b-bit digest, and then expand this discussion to a variety of new and existing ones in Sections 5 and 6.

4.1 Condensing a long-output function

One possibility suggested in [4, 16, 41] is to truncate the output of a cryptographic hash function to the b least significant bits: $digest(k, m)$=trunc$_b$ $(hash(k \parallel m))$. This does not exploit the short output and parallel computation to increase computational efficiency, because the design of the underlying long-output cryptographic hash function $hash()$ usually follows the Merkle-Damgård structure [34]. Also it is hard to provide any concrete security evidence for the truncation operation trunc$_b()$ since the definition of a hash function does not normally specify the distribution of individual groups of bits. In contrast, we will show in subsection 5.2 that truncation is secure in our new digest constructions.

Taking a different approach, Gehrmann and Nyberg [16] proposed using error-correcting codes (ECC) to construct a short-output (b-bit) check-value function, which is a variant of the polynomial universal hashing algorithm introduced in [10, 20, 55]. The advantage of this approach is its mathematical structure and its security proof, yet the algorithm puts a severe limit on the message length. To have a perfectly balanced (b-bit) digest function $\epsilon_c = 2^{-b}$, the input message length must not exceed $2b$. More generally, to obtain a collision probability $\epsilon_c = c2^{-b}$ where $c \in [1, 2^b - 1]$, the input message bitlength is bounded above by $(c+1)b$.[5] What Gehrmann and Nyberg suggested was to compress any significant amount of data into $(c+1)b$ bits by using a cryptographic hash function as a first step: $digest(k, m) = ECC(k, hash(m))$.

We note that this construction has to compromise the security (i.e. $\epsilon_c = c2^{-b}$) to digest messages which are longer than $2b$ bits. To minimise this effect, we propose the following improved scheme which is nearly perfectly balanced at the cost of more computation. As in

[5]Suppose $b = 32$ and $\epsilon = 7 * 2^{-16}$, then the input message cannot have more than 256 bits, which is just about good enough to authenticate a single key.

Gehrmann-Nyberg, a large message first needs to be hashed into an $8b$-bit value, but then the length of this hash value will be repeatedly halved by using a perfectly balanced digest construction $ECC()$.

$$digest(k, m) = ECC(k_1, ECC(k_2, ECC(k_3, hash(m))))$$

Here $ECC()$ always takes a message twice the length of the key, and produces an output of length half the message.[6] Keys k_1, k_2, and k_3 are derived from k, and their bitlengths are b, $2b$ and $4b$. The collision probability of this construction is therefore $\epsilon_c = 2^{-b} + 2^{-2b} + 2^{-4b} + \mu \approx 2^{-b}$, where μ is the collision probability of $hash()$ and $\mu \ll 2^{-b}$.

We can easily generalise this algorithm to the whole length of message m, and thus eliminate the need for $hash()$. Although the total length of all keys $(k_1, k_2, \ldots)$ required is the same as the length of m, these keys can be efficiently derived from k via a PRNG. Unfortunately this method will become much more expensive mainly because $ECC()$, which involves modular arithmetics, will be called $\lceil \log M \rceil$ times.

4.2 Toeplitz matrix based construction

Instead of condensing longer functions, one can use the following direct construction invented independently by Krawczyk [26] and Mansour et al. [30]. Since this construction uses a Toeplitz matrix multiplication, we give the definition for a Toeplitz matrix below.

Definition 4. A Toeplitz matrix A is a (not necessary square) matrix where each left-to-right diagonal is fixed, i.e. for all pairs of indexes (i, j): $A_{i,j} = A_{i+1,j+1}$.

If we want to compute a b-bit universal hash of a M-bit message m, then $(M + b - 1)$-bit key k is drawn randomly from $R = \{0, 1\}^{M+b-1}$. We can generate a Toeplitz matrix $A(k)$ of M rows and b columns from key k, i.e. we assume a linear map from $(\mathbb{F}_2)^{M+b-1}$ to the set of Toeplitz matrices in $(\mathbb{F}_2)^{M \times b}$, and then

$$h(k, m) \quad = \quad m \times A(k) \tag{1}$$

The symbol '$\times$' in Equation 1 represents a product of vector m and matrix $A(k)$ over $\mathbb{F}_2$.

If key k is drawn randomly from R, then the collision probability is $\epsilon_c = 2^{-b}$. This construction however has two disadvantages. First it requires many bit operations arising from a (Toeplitz) matrix multiplication, and hence unless special hardware or instruction for binary matrix multiplication is available this is likely to be very inefficient. For large messages, we need to expand a short seed into a long ($M + b - 1$-bit) keystream efficiently, and one therefore frequently uses a linear pseudorandom number generator or one with a low linearity complexity for this purpose. Sadly it is not secure to do so in this case, because this construction is *linear* in the key, i.e. $h(k_1 + k_2, m) = h(k_1, m) + h(k_2, m)$. To justify this argument, in Annex A, we describe an attack on this construction if key expansion is done by a linear feedback shift register (or LFSR) whose feedback rule is fixed and known to an attacker. We note that Krawczyk [26] suggested that k could be drawn from a biased distribution on sequences of length K, however constructions for biased sequences due to Alon et al. [7] would involve the computation of either a Legendre symbol or a modular exponentiation per each pseudorandom bit, both of which are computationally expensive.

[6] With $m = m_0 || m_1$ and $k, m_0, m_1 \in \mathbb{F}_p$ where p is prime, then $ECC(k, m) = m_0 + m_1 k \pmod p$ with $\epsilon_c = 1/p$.

5 A new digest construction

We first discuss the well-studied multiplicative universal hashing algorithm introduced by Dietzfelbinger et al. [14]. Although this scheme is provably secure, it is not efficient with long messages, and consequently we will develop this method further by making use of word multiplication instructions which are fast in software on standard and sometimes basic microprocessors. We note that there have been two other related universal hashing methods which also use word multiplications, namely MMH of Halevi and Krawczyk [18] and NH of Black et al. [9]. Both of which will be compared against our new construction in Section 6.

It is important to point out that our digest function as well as MMH and NH all require a key of comparable size as the input message to be randomly selected from its domain. This is the only assumption made in Theorems 1 and 3 (the security proofs of our digest algorithms). In practice, we need to generate such a long key out of a shorter key via a pseudorandom number generator (PRNG), and the issue whether the PRNG need to be cryptographically secure or not is *independent* of the work reported in this section. Instead it depends on the type of authentication mechanisms which use a digest function as pointed out in Section 2.

- For message authentication codes, since a MAC key must be kept private and unknown to the intruder, the process of expanding this key into a long keystream for being input into a MAC algorithm must pass the next-bit test, i.e. a CS-PRNG is required.
- For manual authentication protocols, a digest key is always revealed prior to digest computation, and hence the PRNG does not need to be cryptographically secure.

5.1 Multiplicative universal hashing

Suppose we want to compute a b-bit universal hash of a M-bit message, then the universal hash key k must be drawn randomly from $R = \{1, 3, 5, \ldots, 2^M - 1\}$, i.e. k is odd. Dietzfelbinger et al. [14] define:

$$h(k, m) = (k * m \bmod 2^M) \text{ div } 2^{M-b}$$

It was proved that the collision probability of this construction is $\epsilon_c = 2^{1-b}$ on equal length inputs [14]. While this has a simple description, for long input messages of several megabytes, such as images, it will become very time consuming to compute the integer multiplication involved in this algorithm. We therefore propose the following change to make it run faster.

5.2 New construction and security proof

In this section, we will show that $digest(k, m)$ can be calculated using word multiplications that are implemented efficiently in just about all processors.

Let us divide message m into b-bit blocks $\langle m_1, \ldots, m_{t=M/b} \rangle$. A $(M + b)$-bit key $k = \langle k_1, \ldots, k_{t+1} \rangle$ is selected randomly from $R = \{0, 1\}^{M+b}$. A b-bit $digest(k, m)$ is defined as

$$digest(k, m) \quad = \quad \sum_{i=1}^{t} [m_i * k_i + (m_i * k_{i+1} \text{ div } 2^b)] \bmod 2^b \tag{2}$$

Here, * refers to a word multiplication of two b-bit blocks which produces a $2b$-bit output, whereas both '+' and $\sum$ are additions modulo 2^b.

To see why this scheme is based on the multiplicative method of Dietzfelbinger et al. [14], one can study Figure 1 where all word multiplications involved in Equation 2 can be arranged into the same shape as the overlap of the expanded multiplication between m and k. If we further ignore the effect of the carry in (word) multiplications then this becomes the same as the Toeplitz matrix based construction of subsection 4.2. And indeed such a carry-less multiplication instruction is available in a new Intel processor [6].

Operation count. To give an estimated operation count for an implementation of *digest*(), which will be subsequently compared against universal hashing schemes MMH and NH, we consider a machine with the same properties as one used by Halevi and Krawczyk [18]:

- ($b = 32$)-bit machine integers, and arithmetic operations are done in registers.
- A multiplication of two 32-bit integers yields a 64-bit result that is stored in 2 registers.

A pseudo-code for *digest*() on such machine may be as follows

digest(key, msg)
1. $Sum = 0$
2. load $key[1]$
3. for $i = 1$ to t
4. load $msg[i]$
5. load $key[i+1]$
6. $\langle High1, Low1\rangle = msg[i] * key[i]$
7. $\langle High2, Low2\rangle = msg[i] * key[i+1]$
8. $Sum = Sum + Low1 + High2$
9. return Sum

This consists of $2t = 2M/b$ word multiplications (MULT) and $2t = 2M/b$ addition modulo 2^b (ADD). That is each message-word requires 1 MULT and 2 ADD operations. As in [18], a MULT/ADD operation should include not only the actual arithmetic instruction but also loading the message- and key-words to registers and/or loop handling.

Theorem 1. For any $t, b \geq 1$, *digest*() of Equation 2 satisfies Definition 3 with the distribution probability $\epsilon_d = 2^{-b}$ and the collision probability $\epsilon_c = 2^{1-b}$ on equal length inputs.

Proof. We first consider the collision property. For any pair of distinct messages of equal length: $m = m_1 \cdots m_t$ and $m' = m'_1 \cdots m'_t$, without loss of generality we assume that $m_1 > m'_1$. Please note that when $t = 1$ or $m_i = m'_i$ for all $i \in \{1, \ldots, t-1\}$ then in the following calculation we will assume that $m_{t+1} = m'_{t+1} = 0$. A digest collision is equivalent to:

$$\sum_{i=1}^{t} [m_i * k_i + (m_i * k_{i+1} \text{ div } 2^b)] = \sum_{i=1}^{t} [m'_i * k_i + (m'_i * k_{i+1} \text{ div } 2^b)]$$

There are two possibilities as follows.

WHEN $m_1 - m'_1$ is odd. The above equality can be rewritten as

$$(m_1 - m'_1)k_1 \quad = \quad y \tag{3}$$

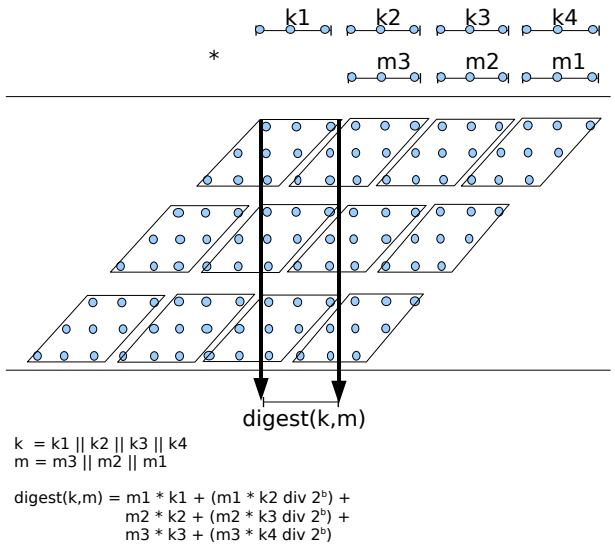

Figure 1: Word multiplication model $digest(k, m)$. Each parallelogram equals the expansion of a word multiplication between a b-bit key block and a b-bit message block.

where

$$\begin{aligned} y &= (m'_1 k_2 \text{ div } 2^b) + \sum_{i=2}^{t} \Big[m'_i * k_i + (m'_i * k_{i+1} \text{ div } 2^b) \Big] - \\ & \quad (m_1 k_2 \text{ div } 2^b) - \sum_{i=2}^{t} \Big[m_i * k_i + (m_i * k_{i+1} \text{ div } 2^b) \Big] \end{aligned}$$

We note that y depends only on keys $k_2, \ldots, k_{t+1}$, and hence we fix k_2 through k_{t+1} in our analysis. Since $m_1 - m'_1$ is odd, i.e. $m_1 - m'_1$ and 2^b are co-prime, there is at most one value of k_1 satisfying Equation 3. The collision probability is therefore $\epsilon_c = 2^{-b} < 2^{1-b}$.

WHEN $m_1 - m'_1$ is even. A digest collision can be rewritten as

$$(m_1 - m'_1)k_1 + (m_1 k_2 \text{ div } 2^b) - (m'_1 k_2 \text{ div } 2^b) + (m_2 - m'_2)k_2 = y \tag{4}$$

where

$$\begin{aligned} y &= (m'_2 k_3 \text{ div } 2^b) + \sum_{i=3}^{t} \Big[m'_i * k_i + (m'_i * k_{i+1} \text{ div } 2^b) \Big] - \\ & \quad (m_2 k_3 \text{ div } 2^b) - \sum_{i=3}^{t} \Big[m_i * k_i + (m_i * k_{i+1} \text{ div } 2^b) \Big] \end{aligned}$$

We note that y depends only on keys $k_3, \ldots, k_{t+1}$. If we fix k_3 through k_{t+1} in our analysis,

we need to find the number of pairs (k_1, k_2) such that Equation 4 is satisfied.

$$\epsilon_c \leq \text{Prob}_{\substack{0 \leq k_1 < 2^b \\ 0 \leq k_2 < 2^b}} \left[(m_1 - m'_1)k_1 + (m_1 k_2 \text{ div } 2^b) - (m'_1 k_2 \text{ div } 2^b) + (m_2 - m'_2)k_2 = y \right]$$

We define

$$\begin{aligned} m_1 k_2 &= u 2^b + v \\ m'_1 k_2 &= u' 2^b + v' \end{aligned}$$

Since we assumed $m_1 > m'_1$, we have $u \geq u'$ and $(m_1 - m'_1)k_2 = (u - u')2^b + v - v'$.

- When $v \geq v'$: $(m_1 k_2 \text{ div } 2^b) - (m'_1 k_2 \text{ div } 2^b) = (m_1 - m'_1)k_2 \text{ div } 2^b$
- When $v < v'$: $(m_1 k_2 \text{ div } 2^b) - (m'_1 k_2 \text{ div } 2^b) = [(m_1 - m'_1)k_2 \text{ div } 2^b] + 1$

Let $c = m_1 - m'_1$ and $d = m_2 - m'_2 \pmod{2^b}$, we then have $1 \leq c < 2^b$ and:

$$\epsilon_c \leq p_1 + p_2$$

where

$$p_1 = \text{Prob}_{\substack{0 \leq k_1 < 2^b \\ 0 \leq k_2 < 2^b}} \left[ck_1 + (ck_2 \text{ div } 2^b) + dk_2 = y \right]$$

and

$$p_2 = \text{Prob}_{\substack{0 \leq k_1 < 2^b \\ 0 \leq k_2 < 2^b}} \left[ck_1 + (ck_2 \text{ div } 2^b) + dk_2 = y - 1 \right]$$

Using Lemma 1, we have $p_1, p_2 \leq 2^{-b}$, and thus $\epsilon_c = 2^{1-b}$.

As regards distribution, since $m = m_1 \cdots m_t > 0$ as specified in Definition 3, without loss of generality we can assume that $m_1 > 0$. If we fix k_3 through k_{t+1}, we need to find the following probability:

$$\epsilon_d \leq \text{Prob}_{\substack{0 \leq k_1 < 2^b \\ 0 \leq k_2 < 2^b}} \left[m_1 k_1 + (m_1 k_2 \text{ div } 2^b) + m_2 k_2 = y \right]$$

Using Lemma 1, we have $\epsilon_d = 2^{-b}$. □

Lemma 1. Let $1 \leq c < 2^b$ and $0 \leq d < 2^b$, then for any $y \in \{0, \ldots, 2^b - 1\}$ there are at most 2^b pairs $k_1, k_2 \in \{0, \ldots, 2^b - 1\}$ such that

$$ck_1 + (ck_2 \text{ div } 2^b) + dk_2 = y \pmod{2^b}$$

Proof. We write $c = s2^l$ with s odd and $0 \leq l < b$. Since s and 2^b are co-prime, there exist a unique inverse modulo 2^b of s, we call it s^{-1}. Our equation now becomes:

$$2^l s k_1 + (2^l s k_2 \text{ div } 2^b) + d s^{-1} s k_2 = y \pmod{2^b}$$

Let $sk_1 = \gamma \pmod{2^{b-l}}$ and $sk_2 = \alpha 2^{b-l} + \beta \pmod{2^b}$, we then have $0 \leq \gamma < 2^{b-l}$ and $0 \leq \alpha < 2^l$. The above equation becomes:

$$\begin{aligned} 2^l \gamma + \alpha + d s^{-1}(\alpha 2^{b-l} + \beta) &= y \\ 2^l \gamma + \alpha(1 + d s^{-1} 2^{b-l}) + \beta d s^{-1} &= y \\ 2^l \gamma + \alpha x &= z \end{aligned}$$

Where $x = 1 + ds^{-1}2^{b-l} \pmod{2^b}$ which is always odd, and $z = y - \beta d s^{-1}$. Using Lemma 2, there is a unique pair (γ, α) satisfying the above equation. Since $0 \leq \gamma < 2^{b-l}$ and $0 \leq \alpha < 2^l$, there are at most 2^b pairs (k_1, k_2) satisfying the condition that we require in this lemma. □

Lemma 2. Let $0 \leq l < b$ and $x \in \{1, 3, \ldots, 2^b - 1\}$ then for any $z \in \{0, \ldots, 2^b - 1\}$ there is a unique pair (γ, α) such that $0 \leq \gamma < 2^{b-l}$, $0 \leq \alpha < 2^l$, and $2^l\gamma + \alpha x = z \pmod{2^b}$.

Proof. If there exist two distinct pairs (γ, α) and (γ', α') satisfying this condition, then

$$2^l\gamma + \alpha x = 2^l\gamma' + \alpha' x = z$$

which implies that

$$2^l(\gamma - \gamma') = (\alpha' - \alpha)x$$

This leads to two possibilities.

- When $\alpha' = \alpha$ then $2^l(\gamma - \gamma') = 0$, which means that $2^{b-l} | (\gamma - \gamma')$. The latter is impossible because $0 \leq \gamma, \gamma' < 2^{b-l}$ and $\gamma \neq \gamma'$.
- When $\alpha' \neq \alpha$ and since x is odd, we must have $2^l | (\alpha' - \alpha)$. This is also impossible because $0 \leq \alpha, \alpha' < 2^l$.

□

REMARKS. The bound given by Theorem 1 for the distribution probability ($\epsilon_d = 2^{-b}$) is tight: let $m = 0^{b-1}1$ and any y and note that any key $k = k_1k_2$ with $k_1 = y$ satisfying this equation $digest(k, m) = y$. The bound given by Theorem 1 for the collision probability $\epsilon_c = 2^{1-b}$ also appears to be tight. To verify this bound, we have implemented exhaustive tests on single-word messages with small value of b. For example, when $b = 7$, we look at all possible pairs of two different ($b = 7$)-bit messages in combination with all ($2b = 14$)-bit keys, the obtained collision probability is $2^{-b} \times 1.875$.

We end this section by pointing out that truncation is secure in this digest construction. For any $b' \in \{1, \ldots, b-1\}$, we define

$$\text{trunc}_{b'}(digest(k, m)) \quad = \quad \sum_{i=1}^{t} [m_i * k_i + (m_i * k_{i+1} \text{ div } 2^b)] \bmod 2^{b'} \tag{5}$$

where $\text{trunc}_{b'}()$ takes the first b' least significant bits of the input. We then have the following theorem whose proof is very similar to the proof of Theorem 1, and hence it is not given here.

Theorem 2. For any $n, t \geq 1$, $b \geq 1$ and any integer $b' \in \{1, \ldots, b-1\}$, $\text{trunc}_{b'}(digest())$ of Equation 5 satisfies the definition of a digest function with the distribution probability $\epsilon_d = 2^{-b'}$ and the collision probability $\epsilon_c = 2^{1-b'}$ on equal length inputs.

5.3 Reducing the collision probability

Although single-word or b-bit digest schemes where $b \in \{8, 16, 32\}$ are suitable for their application in manual authentication protocols, if we want to use digest functions as the main ingredient of a message authentication code, we need to reduce the collision probability without increasing the value of b that is dictated by architecture characteristics. It turns out that using the same technique as in the multiplicative universal hashing method of subsection 5.1, a single-word digest scheme can be straightforwardly generalised to a multiple-word (or nb-bit) digest construction $digest_{MW}()$, where MW stands for multiple-word, as seen in Figure 2 and below.

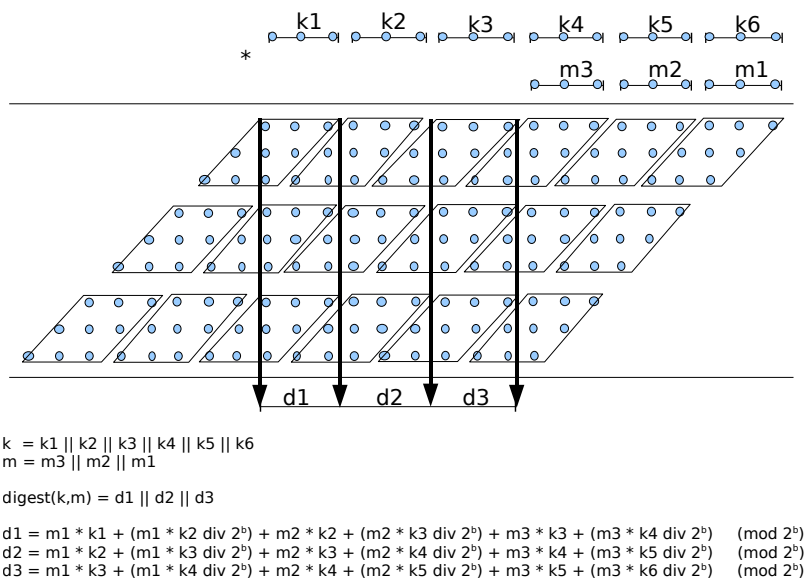

Figure 2: $3b$-bit output $digest_{MW}(k, m)$. Each parallelogram equals the expansion of a word multiplication between a b-bit key block and a b-bit message block.

We still divide m into b-bit blocks $\langle m_1, \ldots, m_{t=M/b} \rangle$. However, a $(M + bn)$-bit key $k = \langle k_1, \ldots, k_{t+n} \rangle$ will be chosen randomly from $R = \{0, 1\}^{M+bn}$ to compute a nb-bit digest.

For all $i \in \{1, \ldots, n\}$, we then define:

$$d_i = digest(k_{i \cdots t+i}, m) = \sum_{j=1}^{t} [m_j k_{i+j-1} + (m_j k_{i+j} \text{ div } 2^b)] \bmod 2^b$$

And

$$digest_{MW}(k, m) = \langle d_1 \cdots \ d_n \rangle$$

Operation count. The advantage of this scheme is the ability to reuse the result of each word multiplication in the computation of two adjacent digest output words as seen in Figure 2 and the following pseudo-code, e.g. the multiplication $m_1 k_2$ is instrumental in the computation of both d_1 and d_2. Using the same machine as specified in subsection 5.2, each message-word therefore requires $(n + 1)$ MULT and $2n$ ADD operations.

A pseudo-code for $digest_{MW}()$ on such machine may be as follows

$digest_{MW}(key, msg)$
1. For $i = 1$ to n
2. $\quad d[i] = 0$
3. $\quad$ load $key[i]$
4. For $j = 1$ to t
5. $\quad$ load $msg[j]$
6. $\quad$ load $key[j + n]$

7. $\langle High[0], Low[0]\rangle = msg[j] * key[j]$
8. For $i = 1$ to n
9. $\quad \langle High[i], Low[i]\rangle = msg[j] * key[j+i]$
10. $\quad d[i] = d[i] + Low[i-1] + High[i]$
11. return $\langle d[1] \cdots d[n]\rangle$

The following theorem and its proof show that $digest_{MW}()$ enjoys the best bound for both collision and distribution probabilities that one could hope for.

Theorem 3. For any $n, t \geq 1$ and $b \geq 1$, $digest_{MW}()$ satisfies the definition of a digest function with the distribution probability $\epsilon_d = 2^{-nb}$ and the collision probability $\epsilon_c = 2^{n-nb}$ on equal length inputs.

Proof. We first consider the collision property of a digest function. For any pair of distinct messages of equal length: $m = m_1 \cdots m_t$ and $m' = m'_1 \cdots m'_t$, without loss of generality we assume that $m_1 > m'_1$. Please note that when $t = 1$ or $m_i = m'_i$ for all $i \in \{1, \ldots, t-1\}$ then in the following calculation we will assume that $m_{t+1} = m'_{t+1} = 0$.

For $i \in \{1, \ldots, n\}$, we define Equality E_i as

$$E_i : \sum_{j=1}^{t} \Big[m_j k_{i+j-1} + (m_j k_{i+j} \text{ div } 2^b)\Big] = \sum_{j=1}^{t} \Big[m'_j k_{i+j-1} + (m'_j k_{i+j} \text{ div } 2^b)\Big]$$

and thus the collision probability is: $\epsilon_c = \text{Prob}_{\{k \in R\}}[E_1 \wedge \cdots \wedge E_n]$.

WHEN $m_1 - m'_1$ is odd. We proceed by proving that for all $i \in \{1, \ldots, n\}$

$$\text{Prob}[E_i \text{ is true} \mid E_{i+1}, \ldots, E_n \text{ are true}] \leq 2^{-b}$$

For Equality E_n, the claim is satisfied due to Theorem 1. We notice that Equalities E_{i+1} through E_n depend only on keys $k_{i+1}, \ldots, k_{n+t}$, whereas Equality E_i depends also on key k_i. Fix k_{i+1} through k_{n+t} such that Equalities E_{i+1} through E_n are satisfied. We prove that there is at most one value of k_i satisfying E_i. To achieve this we let

$$\begin{aligned} z \;=\; & (m'_1 k_{i+1} \text{ div } 2^b) + \sum_{j=2}^{t} \Big[m'_j k_{i+j-1} + (m'_j k_{i+j} \text{ div } 2^b)\Big] - \\ & (m_1 k_{i+1} \text{ div } 2^b) - \sum_{j=2}^{t} \Big[m_j k_{i+j-1} + (m_j k_{i+j} \text{ div } 2^b)\Big] \end{aligned}$$

we then rewrite Equality E_i as

$$(m_1 - m'_1) k_i = z$$

Since we assumed $m_1 - m'_1$ is odd, there is at most one value of k_i satisfying this equation.

WHEN $m_1 - m'_1$ is even. We write $m_1 - m'_1 = 2^l s$ with s odd and $0 < l < b$, and $s' = (m'_2 - m_2)s^{-1}$. We further denote $sk_i = x_i 2^{b-l} + y_i$ for $i \in \{1, \ldots, n+t\}$, where $0 \leq x_i < 2^l$ and $0 \leq y_i < 2^{b-l}$.

For $i \in \{1, \ldots, n\}$, if we define $b_i \in \{0, 1\}$ and

$$f(y_i, x_{i+1}) = 2^l y_i + x_{i+1}[(m_2 - m_2')s^{-1}2^{b-l} + 1]$$

$$\begin{aligned} g(k_{i+2}, \ldots, k_{i+t}) &= (m_2' k_{i+2} \text{ div } 2^b) + \sum_{j=3}^{t} \Big[m_j' k_{i+j-1} + (m_j' k_{i+j} \text{ div } 2^b) \Big] - \\ &\quad (m_2 k_{i+2} \text{ div } 2^b) - \sum_{j=3}^{t} \Big[m_j k_{i+j-1} + (m_j k_{i+j} \text{ div } 2^b) \Big] \end{aligned}$$

then, using similar trick as in the proof of Lemma 1, Equality E_i can be rewritten as

$$\begin{aligned} (m_1 - m_1')k_i + ((m_1 - m_1')k_{i+1} \text{ div } 2^b) + (m_2 - m_2')k_{i+1} &= g(k_{i+2}, \ldots, k_{i+t}) - b_i \\ 2^l s k_i + (2^l s k_{i+1} \text{ div } 2^b) + (m_2 - m_2')s^{-1} s k_{i+1} &= g(k_{i+2}, \ldots, k_{i+t}) - b_i \\ 2^l y_i + x_{i+1} + (m_2 - m_2')s^{-1}(x_{i+1}2^{b-l} + y_{i+1}) &= g(k_{i+2}, \ldots, k_{i+t}) - b_i \\ 2^l y_i + x_{i+1}[(m_2 - m_2')s^{-1}2^{b-l} + 1] &= s' y_{i+1} - b_i + g(k_{i+2}, \ldots, k_{i+t}) \\ f(y_i, x_{i+1}) &= s' y_{i+1} - b_i + g(k_{i+2}, \ldots, k_{i+t}) \end{aligned}$$

Putting Equalities E_1 through E_n together, we have

$$\begin{aligned} E_1 : \ f(y_1, x_2) &= s' y_2 - b_1 + g(k_3, \ldots, k_{1+t}) \\ E_2 : \ f(y_2, x_3) &= s' y_3 - b_2 + g(k_4, \ldots, k_{2+t}) \\ E_3 : \ f(y_3, x_4) &= s' y_4 - b_3 + g(k_5, \ldots, k_{3+t}) \\ &\vdots \quad \vdots \quad \vdots \\ E_{n-1} : \ f(y_{n-1}, x_n) &= s' y_n - b_{n-1} + g(k_{n+1}, \ldots, k_{n+t-1}) \\ E_n : \ f(y_n, x_{n+1}) &= s' y_{n+1} - b_n + g(k_{n+2}, \ldots, k_{n+t}) \end{aligned}$$

We fix k_{n+2} through k_{t+n}. We note that there are 2^{b-t} values for y_{n+1} and two values for b_n. For each pair (y_{n+1}, b_n) there is a unique pair (y_n, x_{n+1}) satisfying Equality E_n due to Lemma 2. Similarly, for each tuple $\langle y_n, k_{n+1}, b_{n-1}, b_n \rangle$ there is also a unique pair (y_{n-1}, x_n) satisfying Equality E_{n-1}. We will continue this process until we reach the pair (y_1, x_2) in Equality E_1. Since Equalities E_1 through E_n do not depend on x_1 and there are 2^l values for x_1, there will be at most $2^l 2^n 2^{b-l} = 2^{n+b}$ different tuples $\langle k_1 \cdots k_{n+1} \rangle$ satisfying Equalities E_1 through E_n. And thus the collision probability $\epsilon_c = 2^{n+b}/2^{(n+1)b} = 2^{n-nb}$.

Similar argument also leads to our bound on the distribution probability $\epsilon_d = 2^{-nb}$. □

REMARKS. Even though Theorems 1 and 3 specify the ϵ_c-almost-universal property of the digest schemes, which is required in manual authentication protocols, their proofs can be easily adapted to show that our constructions are also ϵ_c-almost-Δ-universal as in the case of the MMH scheme considered in the next section. The latter property requires that for every $m, m' \in X$ where $m \neq m'$ and $a \in Y$: $\Pr_{\{k \in R\}}[digest(k, m) - digest(k, m') = a] \leq \epsilon_c$.

6 Comparative analysis

In this section, we compare our new digest scheme against well-studied universal hashing algorithms MMH of Halevi and Krawczyk [18] and NH of Black et al. [9]. While the single-word or b-bit output digest scheme can only be used on its own in manual authentication

protocols, it can perfectly be generalised to have multiple-word output as in the case of MMH and NH to design MAC schemes. Consequently, our comparative analysis consider both single- and multiple-word output schemes.

The main properties of these three schemes are summarised in Table 2, and their implementation results are given in Table 3. The pseudo-codes of both MMH and NH are provided in Annex B.

6.1 MMH

Fix a prime number $p \in [2^b, 2^b + 2^{b/2}]$. The b-bit output MMH universal hash function is defined for any $k = k_1, \ldots, k_t$ and $m = m_1, \ldots, m_t$ as follows

$$\mathrm{MMH}(k, m) = \left[\left[\left[\sum_{i=1}^{t} m_i * k_i\right] \bmod 2^{2b}\right] \bmod p\right] \bmod 2^b$$

It was proved in [18] that MMH can generate digest with the collision probability $\epsilon_c = 6 \times 2^{-b}$ as opposed to only 2^{1-b} obtained by $digest()$. It is also not hard to show that the distribution probability $\epsilon_d = 2^{2-b}$ by using the same proof technique presented in [18].

For single-word output, each message word in MMH requires 1 word of (pseudo)random key, 1 ($b \times b$) MULT, and 1 ADD modulo 2^{2b}. We note however that this does not include the cost of the final reduction modulo p. For n-word output MMH, using "the Toeplitz matrix approach", the scheme is defined as

$$\mathrm{MMH}_{MW}(k, m) = \mathrm{MMH}(k_{1\cdots t}, m) \parallel \mathrm{MMH}(k_{2\cdots t+1}, m) \parallel \cdots \parallel \mathrm{MMH}(k_{n\cdots t+n-1}, m)$$

MMH_{MW} obtains $\epsilon_c = 6^n 2^{-nb}$ and $\epsilon_d = 2^{2n-nb}$, which are significantly worse than $digest_{MW}()$ ($\epsilon_c = 2^{n-nb}, \epsilon_d = 2^{-nb}$).

6.2 NH

The $2b$-bit output NH universal hash function is defined for any $k = k_1, \ldots, k_t$ and $m = m_1, \ldots, m_t$, where t is even, as follows

$$\mathrm{NH}(k, m) = \sum_{i=1}^{t/2} (k_{2i-1} + m_{2i-1})(k_{2i} + m_{2i}) \bmod 2^{2b}$$

The downside of NH relative to MMH and our digest method is the level of security obtained, namely with a $2b$-bit output, which is twice the length of both $digest()$ and MMH, NH was shown to have the collision probability $\epsilon_c = 2^{-b}$ and the distribution probability $\epsilon_d = 2^{-b}$, which are far from optimality. While its computational cost is better than the other twos, i.e. each message-word requires 1 word of (pseudo)random key, only 1/2 ($b \times b$) MULT, 1 ADD modulo 2^b, and 1/2 ADD modulo 2^{2b}, NH is not as suitable as MMH and $digest()$ for being used in manual authentication protocols where humans only can compare very short digest values manually.

For $2n$-word output, also using "the Toeplitz matrix approach", we have $\epsilon_c = 2^{-nb}$ and $\epsilon_d = 2^{-nb}$. Each message-word requires $n/2$ MULT and $3n/2$ ADD operations as seen below.

$$\mathrm{NH}_{MW}(k, m) = \mathrm{NH}(k_{1\cdots t}, m) \parallel \mathrm{NH}\,(k_{3\cdots t+2}, m) \parallel \cdots \parallel \mathrm{NH}(k_{2n-1\cdots t+2(n-1)}, m)$$

Scheme	Message bitlength	Key bitlength	MULT per word	ADD per word	ϵ_c	ϵ_d	Output bitlength
digest	M	$M+b$	2	2	2^{1-b}	2^{-b}	b
MMH	M	M	1	1	6×2^{-b}	2^{2-b}	b
NH	M	M	1/2	3/2	2^{-b}	2^{-b}	$2b$
$digest_{MW}$	M	$M+nb$	$n+1$	$2n$	2^{n-nb}	2^{-nb}	nb
MMH_{MW}	M	$M+(n-1)b$	n	n	$6^n \times 2^{-nb}$	2^{2n-nb}	nb
NH_{MW}	M	$M+2(n-1)b$	$n/2$	$3n/2$	2^{-nb}	2^{-nb}	$2nb$

Table 2: A summary on the main properties of *digest*(), MMH and NH. MULT operates on b-bit inputs, whereas ADD operates on inputs of either b or $2b$ bits.

6.3 Implementations and Summary

The main properties of all three schemes considered here are summarised in Table 2, i.e. the upper and lower halves correspond to single-word (b bits) and respectively multiple-word (nb bits) output schemes. This table indicates that the security level obtained in our digest algorithm is higher than both MMH and NH with respect to the same output length. In particular, the collision probability of *digest*() is a third of MMH, while NH must double the output length to achieve the same order of security which makes it not as suitable for being used in manual authentication protocols as MMH and *digest*(). For multiple-word output schemes, this advantage in security of our digest algorithm becomes even more significant as seen in the lower half of Table 2.

We have also tested the implementations of *digest*(), MMH, NH as well as their multiple-word output versions on a workstation with a 1GHz AMD Athlon(tm) 64 X2 Dual Core Processor (4600+ or 512 KB caches) running the 2.6.30 Linux kernel. All source code was written in C making use of GCC 4.4.1 compiler. The number of cycles elapsed during execution was measured by the *clock*() instruction in the normal way (as in the case of UMAC [56]).

For comparison, we recompiled publicly available source code for the SHA1, SHA256 and SHA512 hash functions [48] whose reported speeds on our workstation are given in Table 4.[7]

As seen in Table 2, the key length is comparable with the message length in all constructions. For application of these primitives in MACs, normally each universal hash key is generated once out of a short seed and reused for a period of time, and hence previously recorded speeds for MMH and NH within UMAC [9, 18] do not include the cost of long key generation. In contrast, for manual authentication protocols described earlier, key generation must be done every time a digest is computed, because each digest key is only used once. The choice of PRNG in our test is based on our observation in Section 2 of the new game arising from manual authentication protocols. Namely this PRNG does not need to be cryptographically secure as usually the case in other cryptographic applications, because the key that seeds a PRNG is always revealed to everyone at the point when the digest value is computed. In addition, unlike the Toeplitz-based construction of Section 4.2 our digest constructions (and MMH and NH) are all non-linear in the input key, and for this reason, very fast and ordinary PRNGs suitable for non-cryptographic purposes can be used here. A typical example is the well-studied Mersenne Twister or its updated version SIMD-oriented

[7] This is slightly faster than the results published by the ECRYPT Benchmarking of Cryptographic Systems [15].

digest				MMH				NH		
Output bitlength	ϵ_c	Speed (cpb)		Output bitlength	ϵ_c	Speed (cpb)		Output bitlength	ϵ_c	Speed (cpb)
32	2×2^{-32}	1.57		32	6×2^{-32}	1.33		64	2^{-32}	1.25
64	$2^2 \times 2^{-64}$	1.93		64	$6^2 \times 2^{-64}$	1.69		128	2^{-64}	1.49
96	$2^3 \times 2^{-96}$	2.36		96	$6^3 \times 2^{-96}$	1.79		192	2^{-96}	1.75
160	$2^5 \times 2^{-160}$	2.79		160	$6^5 \times 2^{-160}$	2.14		320	2^{-160}	2.04
256	$2^8 \times 2^{-256}$	2.89		256	$6^8 \times 2^{-256}$	2.17		512	2^{-256}	2.28

Table 3: Performance (cycles/byte) of *digest*, MMH and NH constructions, which include the cost of both pseudorandom key generation (around 1 cycles/byte) and universal hashing computation. In each row, the length of NH is always twice the length of MMH and *digest*.

Workstation	SHA1	SHA256	SHA512
1GHz AMD Athlon 64 X2	5.78	12.35	8.54

Table 4: Speed of SHA hash functions (cycles/byte) on our workstation.

Fast Mersenne Twister PRNG of Saito and Matsumoto [33, 49], which has a very long period of $2^{19937} - 1$ and passes numerous tests of randomness. The speed of this generator is about 1 cycles/byte (or 1 cpb) on our workstation, which is inline with the results published by its inventors [33]. This is significantly faster than the most efficient CS-PRNG to date, which is based on the AES encryption scheme, whose latest recorded speed [22] due to Käsper and Schwabe is at 7.59 cpb on a modern processor.

Table 3 shows the results of the experiment, which were averaged over a large number of random and long inputs of more than one million bits. The speeds are in cycles/byte or cpb, and for completeness they include the cost of both pseudorandom key generation and universal hashing computation. Our digest constructions, at the cost of higher security, are slightly slower than MMH and NH due to extra multiplication operations, but still significantly faster than cryptographic hash functions SHA1/256/512 as seen in Table 4. Also the speeds of the digest algorithms, MMH, and NH go up very slowly as the output length is multiplied, e.g. doubled, tripled, quintupled, or octupled. This can be explained through the fact that although pseudorandom key generation requires quite an amount computation (i.e. each pseudorandom byte takes around 1 cycle to generate), from Table 2 the amount of pseudorandom key generation per each message-word essentially remains unchanged as we increase the output length.

References

[1] `http://www.intel.com/pressroom/archive/releases/20091202comp_sm.htm`

[2] *ARM9E-S (Rev 1) Technical Reference Manual.* 2000. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0165b/DDI0165.pdf`

[3] 3GPP TS 35.216, Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 and UIA2; Document 2: SNOW 3G specification (March 2006).

[4] *Simple Pairing White Paper*. See: www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf

[5] R.P. Brent and P. Zimmermann. *The great trinomial hunt*. American Mathematical Society Notices, to appear.

[6] http://software.intel.com/en-us/articles/carry-less-multiplication-and-its-usage-for-computing-the-gcm-mode/

[7] N. Alon, O. Goldreich, J. Hastad and R. Peralta. *Simple Constructions of Almost k-wise Independent Random Variables.* In Proceedings of the IEEE Symposium on Foundations of Computer Science, 1990, vol. 2, pp. 544-553.

[8] Chen Bangdao, A.W. Roscoe, R. Kainda, L.H. Nguyen. *The Missing Link: Human Interactive Security Protocols in Mobile Payment*. In Proceedings of the 5th International Workshop on Security, IWSEC, November 2010.

[9] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, P. Rogaway. *UMAC: Fast and Secure Message Authentication.* CRYPTO, LNCS vol. 1666, pp. 216-233, 1999.

[10] B. den Boer. *A simple and key-economical unconditional authentication scheme*. Journal of Computer Security 2 (1993), 65-71.

[11] E. Brickell, D. Pointcheval, S. Vaudenay, and M. Yung. *Design validation for siscrete logarithm based signature schemes.* In PKC 2000, LNCS vol. 1751, pp. 276-292.

[12] J.L. Carter and M.N. Wegman. *Universal Classes of Hash Functions.* Journal of Computer and System Sciences, 18 (1979), 143-154.

[13] I. Damgård. *A Design Principle for Hash Functions*. CRYPTO, LNCS vol. 435 (1989), 416-427.

[14] M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. *A reliable randomized algorithm for the closest-pair problem.* Journal Algorithms, 25:19-51, 1997.

[15] ECRYPT Benchmarking of Cryptographic Systems: http://bench.cr.yp.to/ebash.html

[16] C. Gehrmann, C. Mitchell and K. Nyberg. *Manual Authentication for Wireless Devices.* RSA Cryptobytes, vol. 7, no. 1, pp. 29-37, 2004.

[17] B. Gladman. The publicly available source code for SHA1, SHA256 and SHA512 can be found in this website: http://gladman.plushost.co.uk/oldsite/cryptography_technology/sha/index.php

[18] S. Halevi and H. Krawczyk. *MMH: Software Message Authentication in the Gbit/second Rates.* FSE, LNCS vol. 1267, pp. 172-189, 1997.

[19] H. Handschuh and B. Preneel. *Key-Recovery Attacks on Universal Hash Function Based MAC Algorithms*. CRYPTO, LNCS vol. 5157, pp. 144-161, 2008.

[20] T. Johansson, G.A. Kabatianskii and B. Smeets. *On the relation between A-Codes and Codes correcting independent errors.* Advances in Cryptology - Eurocrypt 1993, LNCS vol. 765, pp. 1-11.

[21] A. Joux. *Multicollisions in Iterated Hash Functions.* CRYPTO 2004, LNCS vol. 3152, pp. 306-316, 2004.

[22] E. Käsper and P. Schwabe. *Faster and Timing-Attack Resistant AES-GCM.* CHES, LNCS vol. 5747, pp. 1-17, 2009.

[23] K. Khoo, F.-L. Wong and C.-W Lim. *On a construction of short digests for authenticating ad-hoc networks.* In Proceedings of ICCSA 2009, LNCS vol. 5593, pp. 863-876.

[24] L.R. Knudsen and W. Meier. *Correlations in reduced round variants of RC6.* Fast Software Encryption, Seventh International Workshop, New York, USA, April 2000, Springer Verlag.

[25] H. Krawczyk. *LFSR-based Hashing and Authentication.* Advances in Cryptology, CRYPTO 1994, LNCS vol. 839, 129-139.

[26] H. Krawczyk. *New Hash Functions For Message Authentication.* Advances in Cryptology - Eurocrypt 1995, LNCS vol. 921, pp. 301-310.

[27] S. Laur and K. Nyberg. *Efficient Mutual Data Authentication Using Manually Authenticated Strings.* LNCS vol. 4301, pp. 90-107, 2006.

[28] S. Laur and S. Pasini. *SAS-Based Group Authentication and Key Agreement Protocols.* In Public Key Cryptography - PKC 2008, 11th International Workshop on Practice and Theory in Public-Key Cryptography, pp. 197-213.

[29] A.Y. Lindell, Comparison-based key exchange and the security of the numeric comparison mode in Bluetooth v2.1, in: *Proceedings of the Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, Lecture Notes in Computer Science, Vol. 5473, M. Fischlin, ed., Springer, 2009, pp. 66-83.

[30] Y. Mansour, N. Nisan and P. Tiwari. *The Computational Complexity of Universal Hashing.* Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, pp. 235-243, 1990.

[31] A. Mashatan and D.R. Stinson, Non-interactive two-channel message authentication based on hybrid-collision resistant hash functions, in: *IET Information Security* **1**(3) (2007), 111-118.

[32] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Handbook of Applied Cryptography.* ISBN: 0-8493-8523-7.

[33] M. Matsumoto and T. Nishimura. Fast Mersenne Twister Homepage where the publicly available source code for this pseudorandom number generator can be found: `http://www.math.sci.hiroshima-u.ac.jp/ m-mat/MT/SFMT/`

[34] R.C. Merkle. *A digital signature based on a conventional encryption function.* Crypto 1987, LNCS vol. 293, pp. 369 - 378.

[35] C. Mitchell (editor). ISO/IEC 9798-6 (2005), first edition: *Information Technology – Security Techniques – Entity authentication – Part 6: Mechanisms using manual data transfer.*

[36] L.H. Nguyen (editor), second edition. ISO/IEC 9798-6 (2010): *Information Technology – Security Techniques – Entity authentication – Part 6: Mechanisms using manual data transfer.*

[37] L.H. Nguyen and A.W. Roscoe. *Authentication protocols based on low-bandwidth unspoofable channels: a comparative survey.* Journal of Computer Security (64 pages), to appear in 2011.

[38] L.H. Nguyen and A.W. Roscoe. *Authenticating ad hoc networks by comparison of short digests.* Information and Computation, vol. 206 (2008), pp. 250-271.

[39] L.H. Nguyen and A.W. Roscoe. *Efficient group authentication protocol based on human interaction.* Proceedings of Workshop on Foundation of Computer Security and Automated Reasoning Protocol Security Analysis, pp. 9-31, 2006.

[40] L.H. Nguyen and A.W. Roscoe. *Separating two roles of hashing in one-way message authentication.* Proceedings of FCS-ARSPA-WITS 2008, pp. 195-210.

[41] S. Pasini and S. Vaudenay. *SAS-based Authenticated Key Agreement.* Public Key Cryptography - PKC 2006: The 9th international workshop on theory and practice in public key cryptography, LNCS vol. 3958, pp. 395-409.

[42] R. Rivest and A. Shamir. *PayWord and MicroMint ? two simple micropayment schemes.* CryptoBytes, 2(1):7-11, Spring 1996.

[43] A.W. Roscoe, Chen Bangdao, and L.H. Nguyen. *Reverse authentication in financial transactions.* In Proceedings of the second International Workshop on Security and Privacy in Spontaneous Interaction and Mobile Device Use (IWSSI/SPMU), May 2010.

[44] A.W. Roscoe, *Human-centred computer security*, 2005. See: `http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/113.pdf`

[45] A.W. Roscoe and L.H. Nguyen, *Security in computing networks*, WO Patent 2007052045.

[46] A.W. Roscoe, B. Chen and L.H. Nguyen, *Improvements in communications security*, WO Patent 2008078101.

[47] A.W. Roscoe and L.H. Nguyen, *Improvements related to the authentication of messages*, WO Patent 2009153585.

[48] Please see: `http://www.aarongifford.com/computers/sha.html`

[49] M. Saito and M. Matsumoto. *SIMD-oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator.* Monte Carlo and Quasi-Monte Carlo Methods 2006, Springer, 2008, pp. 607- 622.

[50] N. Smart. *Cryptography, An Introduction.* ISBN 0 077 09987 7 (PB).

[51] O. Staffelbach and W. Meier. *Cryptographic Significance of the Carry for Ciphers Based on Integer Addition* Advances in Cryptology - Crypto 1990, LNCS vol. 537, pp. 602-614, 1991.

[52] F. Stajano and R. Anderson. *The resurrecting duckling: Security issues for ad-hoc wireless networks.* Workshop on Security Protocols 1999, LNCS vol. 1976, pp. 172-194.

[53] D.R. Stinson. *Universal Hashing and Authentication Codes.* Advances in Cryptology - Crypto 1991, LNCS vol. 576, pp. 74-85, 1992.

[54] D.R. Stinson. *On the Connections Between Universal Hashing, Combinatorial Designs and Error-Correcting Codes.* Congressus Numerantium, vol. 114, pp. 7-27, 1996.

[55] R. Taylor. *An Integrity Check Value Algorithm for Stream Ciphers.* Advances in Cryptology, CRYPTO 1993. LNCS vol. 773, Springer-Verlag, pp. 40-48, 1994.

[56] The source code and performance of UMAC can be found on this website: `http://fastcrypto.org/umac/`

[57] S. Vaudenay. *Secure Communications over Insecure Channels Based on Short Authenticated Strings.* Advances in Cryptology - Crypto 2005, LNCS vol. 3621, pp. 309-326.

[58] J. Valkonen, N. Asokan and K. Nyberg. *Ad Hoc Security Associations for Groups.* In Proceedings of the Third European Workshop on Security and Privacy in Ad hoc and Sensor Networks 2006. LNCS vol. 4357, pp. 150-164.

[59] M.N. Wegman and J.L. Carter. *New Hash Functions and Their Use in Authentication and Set Equality.* Journal of Computer and System Sciences, 22 (1981), 265-279.

[60] A. Yao. *Theory and applications of trapdoor functions.* In Proceedings of the IEEE 23rd Annual Symposium on Foundations of Computer Science, pp. 80-91, 1982.

[61] See `http://zfoneproject.com/prod_zfone.html` and `http://tools.ietf.org/html/draft-zimmermann-avt-zrtp-00`

A An attack on LFSR-based $h(k,m)$ of subsection 4.2

Let $(m,m')_2$ denote the inner product modulo 2 of m and $m' \in (\mathbb{F}_2)^M$.

Suppose that $h(k,m)$ of subsection 4.2 uses a r-bit LFSR to generate a $(K = M+b-1)$-bit key $k' = k_0 \cdots k_{K-1}$ out of the r-bit key $k = k_0 k_1 \cdots k_{r-1}$. In this attack, we need to assume that the feedback rule of the LFSR is fixed and known to the intruder, such a feedback rule can be represented by an irreducible monic polynomial.

$$f(x) \;=\; f_0 + f_1 x + f_2 x^2 + f_3 x^3 + \cdots + f_{r-1} x^{r-1} + x^r \tag{6}$$

For any $n \in [r, K-1]$ we then have

$$k_n \;=\; f_0 k_{n-r} + f_1 k_{n-r+1} + \cdots + f_{r-1} k_{n-1} = \sum_{i=0}^{r-1} f_i k_{n-r+i} \tag{7}$$

Since we need to generate a $M \times b$ Toeplitz matrix $A(k)$ from key k, we use the following linear map from $(\mathbb{F}_2)^{M+b-1}$ to the set of Toeplitz matrices in $(\mathbb{F}_2)^{M\times b}$: $A_{i,j}(k) = k'_{i+j}$. In other words, each column (out of b columns) of $A(k)$ is a continuous sequence of M bits taken from k', and each column $i+1$ is shifted (upward) relative to the column i, with a new element set to the last position of the column $i+1$. This is a matrix where each *right-to-left* diagonal is fixed as opposed to the fixed *left-to-right* diagonal property of a Toeplitz matrix as defined in Definition 4, but it is essentially the same as a Toeplitz matrix.

The input message m can also be represented by a polynomial $m(x)$ of degree $M-1$.

$$m(x) = m_0 + m_1 x + \cdots + m_{M-1} x^{M-1}$$

What we want to prove is the following Lemma and Theorem.

Lemma 3. For any $n \geq r$, the reduction of x^n modulo $f(x)$ is a linear combination of $\{x^0, x^1, \ldots, x^{r-1}\}$, and that this linear combination is identical to the coefficients in the expression of k_n as a linear combination of $\{k_0, \ldots, k_{r-1}\}$.

Proof. Since we always have Equation 7 above, we additionally need to prove that for any $n \geq r$ we also have:

$$x^n \quad = \quad f_0 x^{n-r} + f_1 x^{n-r+1} + \cdots + f_{r-1} x^{n-1} \pmod{f(x)}$$

This can be proved by induction. This is clearly true when $n = r$. We now assume that this is true for $n = i$ such that $i \geq r$: $x^i = f_0 x^{i-r} + f_1 x^{i-r+1} + \cdots + f_{r-1} x^{i-1} \pmod{f(x)}$, then multiplying both sides by x implies that this is also true for $n = i + 1$.

Although this only shows that k_n is a linear combination of $\{k_{n-r}, \ldots, k_{n-1}\}$ which is identical to the coefficients in the expression of x^n as a linear combination of $\{x^{n-r}, \ldots, x^{n-1}\}$, if we repeatedly apply the same result to every element in these linear combinations until k_n and t^n are only repsensented by $\{k_0, \ldots, k_{r-1}\}$ and $\{x^0, x^1, \ldots, x^{r-1}\}$, respectively, we will get the proof. □

Theorem 4. If $f(x)$ devides $m(x)$ then the inner product modulo 2 of m and the first M bits output from a LFSR with a feedback rule $f(x)$ is zero regardless of what the initial r-bit seed is, i.e. if $f(x)|m(x)$ then $(k_0 \cdots k_{M-1}, m_0 \cdots m_{M-1})_2 = 0$ for any value of $k = k_0 \cdots k_{r-1}$.

Proof. We first have

$$(k_0 \cdots k_{M-1}, m_0 \cdots m_{M-1})_2 = \sum_{i=0}^{M-1} k_i m_i = \sum_{m_i=1,\ i \in [0, M-1]} k_i$$

For any $i \geq r$ k_i is a linear combination of $\{k_0, \ldots, k_{r-1}\}$, and thus $\Sigma_{m_i=1} k_i$ is also a linear combination of $\{k_0, \ldots, k_{r-1}\}$. Using Lemma 3, the latter linear combination will be identical to the coefficients in the expression of $\Sigma_{m_i=1} x^i$ as a linear combination of $\{x^0, \ldots, x^{r-1}\}$ modulo $f(x)$. Now suppose that:

$$(k_0 \cdots k_{M-1}, m_0 \cdots m_{M-1})_2 = \sum_{m_i=1,\ i \in [0, M-1]} k_i = a_0 k_0 + a_1 k_1 + \cdots + a_{r-1} k_{r-1}$$

Using Lemma 3 and the above observation, we will have

$$\begin{aligned} a_0 x^0 + a_1 x^1 + \cdots + a_{r-1} x^{r-1} &= \sum_{m_i=1,\ i \in [0, M-1]} \left(x^i \pmod{f(x)} \right) \\ &= \left(\sum_{m_i=1,\ i \in [0, M-1]} x^i \right) \pmod{f(x)} \\ &= m(x) \pmod{f(x)} \end{aligned}$$

As a result, when $f(x)|m(x)$ all of the coefficients in the expression of $(k_0 \cdots k_{M-1}, m_0 \cdots m_{M-1})_2$ as a linear combination of $\{k_0, \ldots, k_{r-1}\}$ will be zero regardless of the value of the initial seed $k = k_0 \cdots k_{r-1}$. □

Since each column in the Toeplitz matrix $A(k)$ is a continuous sequence of M bits which is taken from $k' = k_0 \cdots k_{K-1}$, applying Theorem 4, we will have $h(k, m) = 0$ for any value of k when the polynomial representing the feedback rule of the LFSR divides the polynomial representing the input message.

B Pseudo-code for MMH and NH

The following pseudo-code for MMH is taken from [18], where $b = 32$ and $p = 2^{32} + 15$. The output length of MMH is b bits, whereas NH produces $2b$ bits.

MMH(key, msg)
1. $SumHigh = SumLow = 0$
2. for $i = 1$ to t
3. load $msg[i]$
4. load $key[i]$
5. $\langle ProdHigh, ProdLow \rangle = msg[i] * key[i]$
6. $SumLow = SumLow + ProdLow$
7. $SumHigh = SumHigh + ProdHigh + carry$
8. Reduce $\langle SumHigh, SumLow \rangle$ mod p and then mod 2^b

NH(key, msg)
1. $SumHigh = SumLow = 0$
2. for $i = 1$ to $t/2$
3. load $msg[2i - 1]$
4. load $msg[2i]$
5. load $key[2i - 1]$
6. load $key[2i]$
7. $Left = msg[2i - 1] + key[2i - 1]$
8. $Right = msg[2i] + key[2i]$
9. $\langle ProdHigh, ProdLow \rangle = Left * Right$
10. $SumLow = SumLow + ProdLow$
11. $SumHigh = SumHigh + ProdHigh + carry$
12. return $\langle SumHigh, SumLow \rangle$