Short-output universal hash functions and their use in fast and secure message authentication

Long Hoang Nguyen and Andrew William Roscoe

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

Abstract. Message authentication codes usually require the underlining universal hash functions to have a long output so that the probability of successfully forging messages is low enough for cryptographic purposes. To take advantage of fast operation on word-size parameters in modern processors, long-output universal hashing schemes can be securely constructed by concatenating several instances of short-output primitives. In this paper, we describe a new method for short-output universal hash function termed *digest*() suitable for very fast software implementation and applicable to secure message authentication. The method possesses a higher level of security relative to other well-studied short-output universal hashing schemes. Suppose that the universal hash output is fixed at one word of b bits, then the collision probability of ours is 2^{1-b} compared to 6×2^{-b} of MMH, whereas $2^{-b/2}$ of NH within UMAC is far away from optimality. In addition to message authentication codes, we show how short-output universal hashing is applicable to manual authentication protocols where universal hash keys are used in a very different and interesting way.

1 Introduction

Universal hash functions (or UHFs) first introduced by Carter and Wegman [6, 31] have many applications in computer science, including randomised algorithms, database, cryptography and many others. A UHF takes two inputs which are a key k and a message m: $h(k, m)$, and produces a fixed-length output. Normally what we require of a UHF is that for any pair of distinct messages m and m' the collision probability $h(k, m) = h(k, m')$ is small when key k is randomly chosen from its domain. In the majority of cryptographic uses, UHFs usually have long outputs so that combinatorial search is made infeasible. For example, UHFs can be used to build secure message authentication codes or MAC schemes where the intruder's ability to forge messages is bounded by the collision probability of the UHF. In a MAC, parties share a secret universal hash key and an encryption key, a message is authenticated by hashing it with the shared universal hash key and then encrypting the resulting hash. The encrypted hash value together with the message is transmitted as an authentication tag that can be validated by the verifier. We note however that our new construction presented here applies to other cryptographic uses of universal hashing, e.g., manual authentication protocols as seen later as well as non-cryptographic applications.

Since operating on short-length values of 16, 32 or 64 bits is fast and convenient in ordinary computers, long-output UHFs can be securely constructed by concatenating the results of multiple instances of short-output UHFs to increase computational efficiency. To our knowledge, a number of short-output UHF schemes have been proposed, notably MMH (Multilinear-Modular-Hashing) of Halevi and Krawczyk [9] and NH within UMAC of Black et al. [4]. We note that widely studied polynomial universal hashing schemes PolyP, PolyQ [14] and GHASH [24] can also be designed to produce a short output. While polynomial based UHFs only require short and fixed length keys, they suffer from two unpleasant properties relating to security and computational efficiency as will be discussed later in the paper.

Our main contribution presented in Section 3 is the introduction of a new short-output UHF algorithm termed $digest(k, m)$ that can be efficiently computed on any modern microprocessors. The main advantage of ours is that it provides a higher level of security regarding both collision and distribution probabilities relative to MMH and NH described in Section 4. Our *digest*()

algorithm operates on word-size parameters via word multiplication and word addition instructions, i.e. finite fields or non-trivial reductions are excluded, because the emphasis is on high speed implementation using software.

Let us suppose that the universal hash output is fixed at one word of b bits then the collision probability of ours is 2^{1-b} compared to 6×2^{-b} of MMH, whereas $2^{-b/2}$ of NH is much weaker in security. For clarity, the security bounds of our constructions as well as MMH and NH are independent of the length of message being hashed, which is the opposite of polynomial universal hashing schemes mentioned earlier. For multiple-word output universal hashing constructions as required in MACs, the advantage in security of ours becomes more apparent. When the universal hash output is extended to n words or $n \times b$ bits for any $n \in \mathbb{N}^*$, then the collision probability of ours is 2^{n-nb} as opposed to $6^n \times 2^{-nb}$ of MMH and $2^{-nb/2}$ of NH. There is however a trade-off between security and computational cost as illustrated by our estimated operation counts and software implementations of these constructions. On a 1GHz AMD Athlon processor, one version of *digest*() (where the collision probability ϵ_c is 2^{-31}) achieves peak performance of 0.53 cycles/byte (or cpb) relative to 0.31 cpb of MMH (for $\epsilon_c = 2^{-29.5}$) and 0.23 cpb of NH (for $\epsilon_c = 2^{-32}$). Another version of $digest(k, m)$ for $\epsilon_c = 2^{-93}$ achieves peak performance of 1.54 cpb. For comparison purpose, 12.35 cpb is the speed of SHA-256 recorded on our computer. A number of files that provide the software implementations in C programming language of NH, MMH and our proposed constructions can be downloaded from [1] so that the reader can run them and adapt them for other uses of the short-output universal hash schemes.

We will briefly discuss the motivation of designing (and the elegant graphical structure of) our *digest*() scheme which, we have recently discovered, relates to the well-studied multiplicative universal hashing schemes of Dietzfelbinger et al. [7], Krawczyk [12, 13] and Mansour et al. [18]. The latter algorithms are however not efficient when the input message is of a significant size.

Although researchers from cryptographic community have mainly studied UHFs to construct message authentication codes, we would like to point out that short-output UHF on its own has found applications in manual authentication protocols [2, 8, 15, 17, 19, 10, 20–23, 25, 30]. In the new family of authentication protocols, data authentication can be achieved without the need of passwords, shared private keys as required in MACs, or any pre-existing security infrastructures such as a PKI. Instead human owners of electronic devices who seek to exchange their data authentically would need to manually compare a short string of bits that is often outputted from a UHF. Since humans can only compare short strings, the UHF ideally needs to have a short output of say 16 or 32 bits. There is however a fundamental difference in the use of universal hash keys between manual authentication protocols and message authentication codes, it will be clear in Section 5 that none of the short-output UHF schemes including ours should be used directly in the former. Thus we will propose a general framework where any short-output UHFs can be used efficiently and securely to digest a large amount of data in manual authentication protocols.

While existing universal hashing methods are already as fast as the rate information is generated, authenticated and transmitted in high-speed network traffic, one may ask whether we need another universal hashing algorithm. Besides keeping up with network traffic, as excellently explained by Black et al. [4] — *the goal is to use the smallest possible fraction of the CPU's cycles (so most of the machine's cycles are available for other work), by the simplest possible hash mechanism, and having the best proven bounds.* This is relevant to MACs as well as manual authentication protocols where large data are hashed into a short string, and hence efficient short-output UHF constructions possessing a higher (or optimal) level of security are needed.

Acknowledgements:

Long Nguyen's work on this paper was supported by a research grant from the US Office of Naval Research. Andrew William Roscoe's was partially supported by funding from the US Office of Naval Research.

The authors would like to thank Dr. Andrew Ker at the Oxford University Department of Computer Science for his help with statistical analysis of the digest constructions.

Progresses on the security proof of our digest functions were first made when Long Nguyen visited Professor Bart Preneel and Dr. Frederik Vercauteren at the Computer Security and Industrial Cryptography (COSIC) research group at the Katholieke Universiteit Leuven in September and October 2010. The authors would like to thank them for their time and support as well as Drs Nicky Mouha and Antoon Bosselaers at COSIC for pointing out the relevance of the multiplicative universal hashing scheme of Dietzfelbinger et al.[7] and the literature on hash function implementation and speed measurement for benchmarking.

We also received helpful comments from many anonymous referees as well as had fruitful discussions and technical feedbacks from Professor Serge Vaudenay and Dr. Atefeh Mashatan when Long Nguyen visited the Security and Cryptography Laboratory (LASEC) at the Swiss Federal Institute of Technologies (EPFL) in February and March 2011. The feedbacks significantly improve the technical quality and presentation of the paper. Long Nguyen would like to thank them for their time and support.

2 Notation and definitions

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

Definition 1. [12, 13] A ϵ-balanced universal hash function, $h : R \times X \to Y$, must satisfy that for every $m \in X \setminus \{0\}$ and $y \in Y$: $\Pr_{\{k \in R\}}[h(k,m) = y] \leq \epsilon$

Many existing UHF constructions [4, 9, 12, 13] as well as our newly proposed scheme rely on (integer or matrix) multiplications of message and key, and hence non-zero input message is required; for otherwise $h(k, 0) = 0$ for any key $k \in R$.

Definition 2. [13, 27] A ϵ-almost universal hash function, $h : R \times X \to Y$, must satisfy that for every $m, m' \in X$ $(m \neq m')$: $\Pr_{\{k \in R\}}[h(k,m) = h(k,m')] \leq \epsilon$

Since it is useful particularly in manual authentication protocols discussed later to have both the collision and distribution probabilities bounded, we combine Definitions 1 and 2 as follows

Definition 3. An ϵ_d-balanced and ϵ_c-almost universal hash function, $h : R \times X \to Y$, satisfies

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

3 Integer multiplication construction

We first discuss the multiplicative universal hashing algorithm of Dietzfelbinger et al. [7] which obtains a very high level of security. Although this scheme is not efficient with long input data, it strongly relates to our *digest*() method that make use of word multiplication instructions.

We note that there are two other universal hashing schemes which use arithmetic that computer likes to do to increase computational efficiency, namely MMH of Halevi and Krawczyk [9] and NH of Black et al. [4]. Both of which will be compared against our construction in Section 4.

3.1 Multiplicative universal hashing

Suppose that we want to compute a b-bit universal hash of a M-bit message, then the universal hash key k is drawn randomly from $R = \{1, 3, \ldots, 2^M - 1\}$, i.e. k must be odd. Dietzfelbinger et al. [7] 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 [7]. While this has a simple description, for long input messages of several kilobytes or megabytes, such as documents and images, it will become very time consuming to compute the integer multiplication involved in this algorithm.

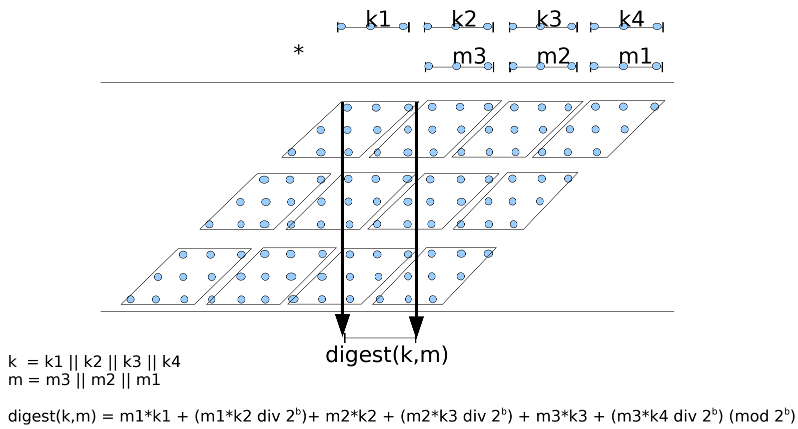

Fig. 1. A b-bit output $digest(k, m)$: each parallelogram represents the expansion of a word multiplication between a b-bit key block and a b-bit message block.

3.2 Word multiplicative construction

In this section, we will define and prove the security of a new short-output universal hashing scheme termed $digest(k, m)$ that can be calculated using word multiplications instead of an arbitrarily long integer multiplication as seen in Equation 1 or an example from Figure 1.

Let us divide message m into b-bit blocks $\langle m_1, \ldots, m_{t=M/b} \rangle$. An $(M + b)$-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

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

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. It should be noted that (div 2^b) is equivalent to a right shift ($>> b$).

To see why this scheme is related to the multiplicative method of Dietzfelbinger et al. [7], one can study Figure 1 where all word multiplications involved in Equation 1 are elegantly arranged into the same shape as the overlap of the expanded multiplication between m and k.[1]

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 [9]:[2]

- ($b = 32$)-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. For a 'C' implementation, please see [1].

```
digest(key, msg)
    Sum = 0
    load key[1]
    for i = 1 to t
        load msg[i]
        load key[i + 1]
        ⟨High1, Low1⟩ = msg[i] * key[i]
        ⟨High2, Low2⟩ = msg[i] * key[i + 1]
        Sum = Sum + Low1 + High2
    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 [9], 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.

The following theorem shows that the switch from a single (arbitrarily long) multiplication of Dietfelbinger et al. into word multiplications of $digest()$ does not weaken the security of the construction. Namely the same collision probability of 2^{1-b} is retained while optimality in distribution is achieved. Moreover this change not only greatly increases computational efficiency but also removes the restriction of odd universal hash key as required in Dietfelbinger et al.

Theorem 1. For any $t, b \geq 1$, $digest()$ of Equation 1 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 $m_1 > m'_1$.[3] A digest collision is equivalent to:

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

[1] If we further ignore the effect of the carry in (word) multiplications of both $digest()$ and the scheme of Dietzfelbinger et al. then they become very similar to the Toeplitz matrix based construction of Krawczyk [12, 13] and Mansour et al. [18] discussed in Annex A. Such a carry-less multiplication instruction is available in a new Intel processor [3].

[2] Although this is a 32-bit machine, the same operation count is applicable to a ($2b = 64$)-bit machine. In the latter, a multiplication of two 32-bit unsigned integer is stored in a single 64-bit register, and $High$ and Low are the upper and lower 32-bit halves of the register.

[3] Please note that when $m_i = m'_i$ for all $i \in \{1, \ldots, j\}$ then in the following calculation we will assume that $m_{j+1} > m'_{j+1}$.

There are two possibilities as follows.

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

$$(m_1 - m'_1)k_1 = y \pmod{2^b} \tag{2}$$

where

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

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 2. 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

$$(m_1 - m'_1)k_1 + (m_1 k_2 \text{ div } 2^b) - (m'_1 k_2 \text{ div } 2^b) + (m_2 - m'_2)k_2 = y \pmod{2^b} \tag{3}$$

where

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

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 3 is satisfied. We arrive at

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

Let us define

$$m_1 k_2 = u 2^b + v$$
$$m'_1 k_2 = u' 2^b + v'$$

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

- When $v \ge v'$: $(m_1 k_2 \text{ div } 2^b) - (m'_1 k_2 \text{ div } 2^b) = (m_1 - m'_1)k_2 \text{ div } 2^b$
- 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 $1 \le c < 2^b$ and:

$$\epsilon_c \le p_1 + p_2$$

where

$$p_1 = \text{Prob}_{\left\{\substack{0 \le k_1 < 2^b \\ 0 \le k_2 < 2^b}\right\}} \Big[c k_1 + (c k_2 \text{ div } 2^b) + d k_2 = y \pmod{2^b}\Big]$$

and

$$p_2 = \text{Prob}_{\left\{\substack{0 \le k_1 < 2^b \\ 0 \le k_2 < 2^b}\right\}} \Big[c k_1 + (c k_2 \text{ div } 2^b) + d k_2 = y - 1 \pmod{2^b}\Big]$$

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

As regards distribution, since $m = m_1 \cdots m_t > 0$ as specified in Definition 3, without loss of generality we can assume that $m_1 \ge 1$. If we fix k_3 through k_{t+1} and for any $y \in \{0, \ldots, 2^b - 1\}$, then the distribution probability ϵ_d is equivalent to:

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

Since $1 \le m_1 < 2^b$, we can use Lemma 1 to deduce that $\epsilon_d = 2^{-b}$. $\square$

Lemma 1. Let $1 \leq c < 2^b$ and $0 \leq d < 2^b$, then for any $y \in \{0, \ldots, 2^b - 1\}$ we have

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

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:

$$2^l sk_1 + (2^l sk_2 \text{ div } 2^b) + ds^{-1}sk_2 = y \pmod{2^b}$$

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:

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

where $x = 1 + ds^{-1}2^{b-l} \pmod{2^b}$ which is always odd because $l < b$, and $z = y - \beta ds^{-1} \pmod{2^b}$. Since z is independent of γ and α, we fix z in our analysis. We can then use Lemma 2 to derive that there is a unique pair (γ, α) satisfying the above equation.

Since $0 \leq \gamma < 2^{b-l}$ and $0 \leq \alpha < 2^l$, γ and α together determine b bits of the combination of k_1 and k_2. Consequently there are at most 2^b different 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

$$2^l\gamma + \alpha x = 2^l\gamma' + \alpha' x = z \pmod{2^b}$$

which implies that

$$2^l(\gamma - \gamma') = (\alpha' - \alpha)x \pmod{2^b}$$

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, i.e. it cannot be reduced to 2^{-b}. 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 $2^{-7} \times 1.875$.

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

$$\text{trunc}_{b'}(digest(k, m)) = \sum_{i=1}^{t} [m_i * k_i + (m_i * k_{i+1} \text{ div } 2^b)] \text{ mod } 2^{b'} \tag{4}$$

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 $b' \in \{1, \ldots, b-1\}$, $\mathrm{trunc}_{b'}(digest())$ of Equation 4 satisfies Definition 3 with the distribution probability $\epsilon_d = 2^{-b'}$ and the collision probability $\epsilon_c = 2^{1-b'}$ on equal length inputs.

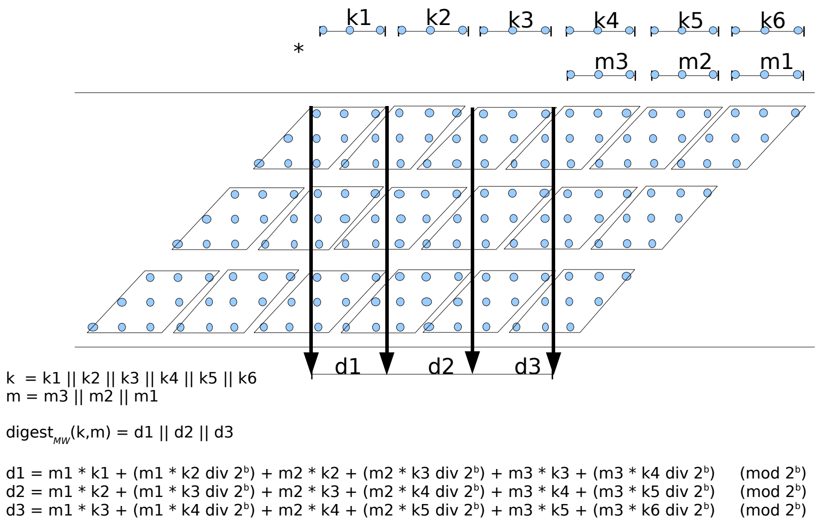

Fig. 2. A $3b$-bit (or three-word) output $digest_{MW}(k, m)$: each parallelogram represents the expansion of a word multiplication between a b-bit key block and a b-bit message block.

3.3 Extending *digest*()

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 word bitlength b that is dictated by architecture characteristics. One possibility is to hash our message with several random and independent keys, and concatenate the results. If we concatenate the results from n independent instances of the digest function, the collision probability drops from 2^{1-b} to 2^{n-nb}. This solution however requires n times as much key material.

A much better and well-studied approach is to use the Toeplitz-extension: given one key we left shift the key by one word to get the next key and digest again. The resulting construction is called $digest_{MW}()$, where MW stands for multiple-word output. The structure of $digest_{MW}()$ is again graphically illustrated by an example in Figure 2 that shows a close connection between $digest_{MW}()$ and the multiplicative universal hashing scheme of Dietfelbinger et al.

We define a n-blocks or $(n \times b)$-bit output $digest_{MW}(k, m)$ as follows. We still divide m into b-bit blocks $\langle m_1, \ldots, m_{t=M/b} \rangle$. However, an $(M + bn)$-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

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

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 Definition 3 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

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

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

$$z = (m'_1 k_{i+1} \text{ div } 2^b) - (m_1 k_{i+1} \text{ div } 2^b) + \sum_{j=2}^{t} \left[(m'_j - m_j) k_{i+j-1} + (m'_j k_{i+j} \text{ div } 2^b) - (m_j k_{i+j} \text{ div } 2^b) \right]$$

we then rewrite Equality E_i as

$$(m_1 - m'_1) k_i = z \pmod{2^b}$$

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

$$f(y_i, x_{i+1}) = 2^l y_i + x_{i+1}[(m_2 - m'_2)s^{-1} 2^{b-l} + 1] \pmod{2^b}$$

$$\begin{aligned} g(k_{i+2}, \ldots, k_{i+t}) = (m'_2 k_{i+2} \text{ div } 2^b) + \sum_{j=3}^{t} \left[m'_j k_{i+j-1} + (m'_j k_{i+j} \text{ div } 2^b) \right] - \\ (m_2 k_{i+2} \text{ div } 2^b) - \sum_{j=3}^{t} \left[m_j k_{i+j-1} + (m_j k_{i+j} \text{ div } 2^b) \right] \pmod{2^b} \end{aligned}$$

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

$$\begin{aligned} (m_1 - m'_1)k_i + ((m_1 - m'_1)k_{i+1} \text{ div } 2^b) + (m_2 - m'_2)k_{i+1} &= g(k_{i+2}, \ldots, k_{i+t}) - b_i \pmod{2^b} \\ 2^l s k_i + (2^l s k_{i+1} \text{ div } 2^b) + (m_2 - m'_2)s^{-1} s k_{i+1} &= g(k_{i+2}, \ldots, k_{i+t}) - b_i \pmod{2^b} \\ 2^l y_i + x_{i+1} + (m_2 - m'_2)s^{-1}(x_{i+1} 2^{b-l} + y_{i+1}) &= g(k_{i+2}, \ldots, k_{i+t}) - b_i \pmod{2^b} \\ 2^l y_i + x_{i+1}[(m_2 - m'_2)s^{-1} 2^{b-l} + 1] &= s' y_{i+1} - b_i + g(k_{i+2}, \ldots, k_{i+t}) \pmod{2^b} \\ f(y_i, x_{i+1}) &= s' y_{i+1} - b_i + g(k_{i+2}, \ldots, k_{i+t}) \pmod{2^b} \end{aligned}$$

Putting Equalities E_1 through E_n together, we have

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

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 address the collision property of an almost universal hash function, their proofs can be easily adapted to show that our constructions are also ϵ_c-almost-Δ-universal [9] 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 $a \in Y$: $\Pr_{\{k \in R\}}[digest(k, m) - digest(k, m') = a] \leq \epsilon_c$.

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 3.2, 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)
For i = 1 to n
    d[i] = 0
    load key[i]
For j = 1 to t
    load msg[j]
    load key[j + n]
    ⟨High[0], Low[0]⟩ = msg[j] * key[j]
    For i = 1 to n
        ⟨High[i], Low[i]⟩ = msg[j] * key[j + i]
        d[i] = d[i] + Low[i − 1] + High[i]
return ⟨d[1] ··· d[n]⟩
```

4 Comparative analysis

In this section, we mainly compare our new digest scheme against well-studied universal hashing algorithms MMH of Halevi and Krawczyk [9] and NH of Black et al. [4] described in Subsections 4.1 and 4.2 respectively. Since $digest()$ can be extended to produce multiple-word output as in the case of MMH and NH to build MACs, our analysis consider both single- and multiple-word output schemes. We note that NH is the building block of not only UMAC but also UHASH16 and UHASH32 [4]. For completeness, we will discuss another widely studied UHF family based on polynomial over finite field, e.g. PolyP, PolyQ, PolyR [14] and GHASH [24]. While the polynomial universal hashing schemes only require short keys, they suffer from two unpleasant properties: (1) the collision probability decreases linearly with the message length, and (2) they are less efficient, especially in software implementation, than our digest functions as well as MMH and NH due to the involved modular arithmetic operations.

The properties of the three main schemes – MMH, NH and *digest*() – are summarised in Table 1 where the upper and lower halves correspond to single-word (b bits) and respectively multiple-word (nb bits) output schemes for any $n \geq 1$. 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. For multiple-word output schemes, this advantage in security of our proposed digest algorithm becomes even more significant as seen in the lower half of Table 1.

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

Table 1. 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.

We end this section by providing implementation results in Table 2 of Section 4.3. As described earlier, C files which contain the implementations of NH, MMH and *digest*() as well as their multiple-word output versions can be downloaded from [1] which allows readers to test the speed of the constructions for themselves.

4.1 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

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

It was proved in [9] that the collision probability of MMH is $\epsilon_c = 6 \times 2^{-b}$ as opposed to only 2^{1-b} of *digest*(). By using the same proof technique presented in [9], it is also not hard to show that the distribution probability of MMH is $\epsilon_d = 2^{2-b}$, as opposed to 2^{-b} of *digest*().

Following is the pseudo-code of MMH take from [9].

MMH(key, msg)
1. $SumHigh = SumLow = 0$
2. for $i = 1$ to t

    load $msg[i]$
    load $key[i]$
    $\langle ProdHigh, ProdLow \rangle = msg[i] * key[i]$
    $SumLow = SumLow + ProdLow$
    $SumHigh = SumHigh + ProdHigh + carry$
Reduce $\langle SumHigh, SumLow \rangle$ mod p and then mod 2^b

For single-word output, each message word in MMH requires 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 considerably weaker than $digest_{MW}()$ $(\epsilon_c = 2^{n-nb}, \epsilon_d = 2^{-nb})$.

4.2 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. Its computational cost is however lower than the other twos, i.e. each message-word requires only $1/2$ $(b \times b)$ MULT, 1 ADD modulo 2^b, and $1/2$ ADD modulo 2^{2b}.

Following is the pseudo-code of NH.

$\mathrm{NH}(key, msg)$
$SumHigh = SumLow = 0$
for $i = 1$ to $t/2$
    load $msg[2i-1]$
    load $msg[2i]$
    load $key[2i-1]$
    load $key[2i]$
    $Left = msg[2i-1] + key[2i-1]$
    $Right = msg[2i] + key[2i]$
    $\langle ProdHigh, ProdLow \rangle = Left * Right$
    $SumLow = SumLow + ProdLow$
    $SumHigh = SumHigh + ProdHigh + carry$
return $\langle SumHigh, SumLow \rangle$

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.

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

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}	0.53	32	6×2^{-32}	0.31	64	2^{-32}	0.23
64	$2^2 \times 2^{-64}$	1.05	64	$6^2 \times 2^{-64}$	0.57	128	2^{-64}	0.39
96	$2^3 \times 2^{-96}$	1.54	96	$6^3 \times 2^{-96}$	0.76	192	2^{-96}	0.62
160	$2^5 \times 2^{-160}$	2.13	160	$6^5 \times 2^{-160}$	1.37	320	2^{-160}	1.15
256	$2^8 \times 2^{-256}$	3.44	256	$6^8 \times 2^{-256}$	2.31	512	2^{-256}	1.90

Table 2. Performance (cycles/byte) of *digest*, MMH and NH constructions. In each row, the length of NH is always twice the length of MMH and *digest*.

4.3 Implementations of MMH, NH and digest constructions

We have 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 codes were 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 UMAC [29]) in our C implementations [1].

For comparison, we recompiled publicly available source codes for SHA-256 and SHA-512 [26] whose reported speeds on our workstation are 12.35 cpb and 8.54 cpb respectively.

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 reported speeds for MMH and NH within UMAC in [4, 9] and our results do not include the cost of key generation.

Table 2 shows the results of the experiments, which were averaged over a large number of random and long data inputs of at least 8 kilobytes. The speeds are in cycles/byte or cpb. Our digest constructions, at the cost of higher security, are slightly slower than MMH and NH due to extra multiplication operations, but still considerably faster than standard cryptographic hash functions SHA-256 and SHA-512.

4.4 Polynomial universal hashing schemes

Since our emphasis of this paper is on fast software implementation of universal hash functions, we have so far mainly considered UHF algorithms using simple arithmetic operations available in most ordinary computers. In this section, we will study another well-studied class of UHF based on polynomial over finite fields, including PolyP, PolyQ, PolyR [14] and GHASH within Galois Counter Mode or GCM [24].

For simplicity, we will give a simple version of polynomial universal hashing that is the core of PolyP, PolyQ, PolyR and GHASH. Let the set of all messages be $\{m = \langle m_1, \ldots, m_t \rangle; m_i \in \mathbb{F}_p\}$, here p is the largest prime number less than 2^b and the message length is $M = tb$ bits. For any key $k \in \mathbb{F}_p$, we define:

$$\text{Poly}(k, m) = m_1 + m_2 k + m_3 k^2 + \cdots + m_t k^{t-1} \pmod{p}$$

Such a scheme does have two nice properties as follows

- The key length of the b-bit output Poly() scheme is fixed at b bits regardless of the message length. In contrast, MMH, NH and *digest*() all require the key length to be greater than or equal to message length.
- Poly() provides collision resistance for both equal and unequal length messages. Suppose that the bit lengths of two different messages m and m' are bt and bt', then the collision probability is $max\{t-1, t'-1\}/p$. On the other hand, MMH, NH and *digest*() only ensure collision

resistance for equal length data, but not unequal length messages. The latter is intuitively because unequal length messages in $digest()$, MMH and NH require unequal length keys, which make them incomparable for collision analysis.

Regarding the first property, as mentioned earlier all of the short-output constructions are usually used to build MACs which reuse a single key for a period of time. Consequently long key generation from a short seed that is done once in a while for $digest()$, MMH or NH will not affect their practical uses in message authentication codes. Without taking into account key generation, MMH, NH and digest functions are significantly faster than PolyP32, PolyQ32 and PolyP64 whose peak performance in Pentium II assembly are 3.69, 3.86 and 6.86 cpb as reported by Krovetz and Rogaway [14]. In addition to 1 MULT and 1 ADD, Poly() requires an extra reduction modulo p per each message word as seen in the pseudo-code below.[4]

The main disadvantage of a polynomial universal hashing scheme is that its collision probability depends on the length of messages, which is the opposite of MMH, NH and $digest()$. Namely, the collision probability of the above scheme is $\epsilon = (t-1)2^{-b}$ that is no where near the level of security obtained by our digest function when message is of a significant size. The security downside of polynomial universal hash functions does have a negative impact on their use in manual authentication protocols where short-output but highly secure universal hash functions are required.

Following is the pseudo-code for Poly().

Poly(key, msg)
1. load $msg[1]$
2. $Sum = msg[1]$
3. for $i = 2$ to t
4. load $msg[i]$
5. $Sum = (Sum + msg[i] * key) \bmod p$
6. return Sum

5 Short-output universal hash functions in manual authentication protocols

In addition to MAC schemes, short-output universal hash functions have found use in manual authentication protocols as explained below.

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, shared private keys as in MACs, or pre-established security infrastructures such as a PKI. Instead authentication is bootstrapped from human trust and interactions.

The authenticated data $m_{A/B}$ might include public keys, images or videos, and so can be of significant size. Using notation taken from authors' work [20–23] the N-indexed arrow ($\longrightarrow_N$) indicates an unreliable and high-bandwidth (or normal) link where messages can be maliciously altered, whereas the E-indexed arrow ($\longrightarrow_E$) represents an authentic and unspoofable (or empirical) 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 $k_{A/B}$ are generated by A/B, and k_A is kept secret until after k_B is revealed in Message 2. Operators $\|$ and $\oplus$ denote bitwise concatenation and exclusive-or.

[4] In line 5 of the pseudo-code of Poly() the operation $Sum + msg[i] * key$ can overflow or be bigger than 2^{2b}, and hence reduction modulo p must be done carefully to obtain the correct result. For example, one might compute $y = msg[i] * key \bmod p$ first, which is followed by $Sum = (Sum + y) \bmod p$.

A pairwise manual authentication protocol [2, 15, 17, 20]
$1. A \longrightarrow_N B : m_A, hash(A \parallel k_A)$
$2. B \longrightarrow_N A : m_B, k_B$
$3. A \longrightarrow_N B : k_A$
$4. A \longleftrightarrow_E B : h(k^*, m_A \parallel m_B)$
where $k^* = k_A \oplus k_B$

To ensure both devices agree on the same data $m_A \parallel m_b$, their human owners manually compare the universal hash value in Message 4. As human interactions are expensive, the universal hash function needs to have a short output of $b \in [16, 32]$ bits.

As seen from the above protocol, the universal hash key k^* always varies randomly and uniformly from one to another protocol run. In other words, no value of k^* is used to hash more than one message because $k_{A/B}$ instrumental in the computation of k^* are randomly chosen in each protocol run. This is fundamentally different from MACs which use the same private key to hash multiple messages for a period of time, and hence attacks which rely on the reuse of a single private key in multiple sessions are irrelevant in manual authentication protocols. What we then want to understand is the collision and distribution properties of the universal hash function. We stress that this analysis is also applicable to group manual authentication protocols [16, 20–22, 30].

Should $digest()$, MMH or NH (or UHASH16/32) be used directly in Message 4 of the above protocol, random and fresh keys $k_{A/B}$ of similar size as $m_A \parallel m_B$ must be generated whenever the protocol is run.[5] Obviously one can generate a long random key stream from a short seed via a pseudo-random number generator, but it can be computationally expensive especially when the authenticated data $m_{A/B}$ are of a significant size. Of course we can use one of the polynomial universal hashing functions (e.g. PolyP32, PolyQ32 or PolyR16_32 all defined in [14]) which require a short key. But since humans only can compare short value over the empirical channel, it is intolerable that the security bound of the universal hash function degrades linearly along with the length of data being authenticated.

One possibility suggested in [2, 8, 25] is to truncate the output of a cryptographic hash function to the b least significant bits:

$$h(k, m) = \mathrm{trunc}_b\left(hash(k \parallel m)\right)$$

Although it can be computationally infeasible to search for a full cryptographic hash collision, it is not clear whether the truncated solution is sufficiently secure because the definition of a hash function does not normally specify the distribution of individual groups of bits.

What we therefore propose is a combination of cryptographic hashing and short-output universal hash functions. We want to stress that among MMH, NH and $digest()$, the least preferable scheme would be NH because it needs to double output length to achieve the same order of security as MMH and $digest()$. The length of universal hash functions must be short in manual authentication protocols because humans can only compare short strings efficiently and accurately.

Without loss of generality, we use our digest method in the following construction which is also applicable to MMH and NH. Let $hash()$ be a B-bit cryptotgraphic hash function, e.g. SHA-2 or SHA-3. First the input key is split into two parts of unequal lengths $k = k_1 \parallel k_2$, where k_1 is $B + b$ bits and k_2 is at least 80 bits. Then our modified digest construction $digest'()$ which takes an arbitrarily length message m is computed as follows

$$digest'(k, m) = digest(k_1, hash(m \parallel k_2))$$

[5] Suppose that the bitlengths of input data and output are M and b then $digest()$ requires $M + b$ bits and both MMH and NH requires M bits for the key.

We hash the concatenation of m and k_2 to make it much harder for the intruder to search for hash collision because a large number of bits of the hash input will not be controlled by the intruder. Consequently the intruder cannot carry out effective off-line searching.

We denotes θ_c the hash collision probability on random messages of $hash()$, and it should be clear that $\theta_c \gg 2^{-b}$ given that $b \in [16, 32]$. The following theorem will demonstrate that this construction preserves both the collision probability except a tiny bias due to the hash function and the distribution probability of $digest()$ regardless of what $hash()$ is. It also removes the restriction on equal length input messages because the hash function $hash()$ always produces a fixed length value.[6]

Theorem 4. *$digest'()$ satisfies Definition 3 with the distribution probability $\epsilon_d = 2^{-b}$ and the collision probability $\epsilon_c = 2^{1-b} + \theta_c$.*

Proof. Let l_1 and l_2 denote the bitlengths of keys k_1 and k_2 respectively.

We first consider collision property of $digest'()$. For any pair of distinct messages m and m', as key k_2 varies uniformly and randomly the probability that $hash(m \parallel k_2) = hash(m' \parallel k_2)$ is bounded above by θ_c. So there are two possibilities:

– When $hash(m \parallel k_2) = hash(m' \parallel k_2)$ then $digest(k_1, hash(m \parallel k_2)) = digest(k_1, hash(m' \parallel k_2))$ for any key $k_1 \in \{0, 1\}^{l_1}$.
– When $hash(m \parallel k_2) \neq hash(m' \parallel k_2)$ then $digest(k_1, hash(m \parallel k_2)) = digest(k_1, hash(m' \parallel k_2))$ with probability 2^{1-b}.

Consequently the collision probability of $digest'()$ is

$$\theta_c + (1 - \theta_c)2^{1-b} < \theta_c + 2^{1-b}$$

As regards distribution probability of $digest'()$, we fix message m of arbitrarily length and a b-bit value y in our analysis.

For each value of k_2, there will be at most 2^{l_1-b} different keys k_1 such that

$$digest(k_1, hash(m \parallel k_2)) = y$$

Since there are 2^{l_2} different keys k_2, there will be at most $2^{l_1-b}2^{l_2} = 2^{l_1+l_2-b}$ different pairs (k_1, k_2) or different keys k such that $digest(k_1, hash(m \parallel k_2)) = y$. The distribution probability of $digest'()$ is therefore 2^{-b} □

We end this section by pointing out that the shortness of UHF output required in manual authentication protocols further implies that UHFs with optimal (or nearly optimal) collision probability are much more sought here than in message authentication codes. Although our proposed $digest'()$ scheme is very near to optimality, we might want to go further. To our knowledge, this is possible but at the expense of involving arithmetic that computers less like to do than word multiplication and addition even when the input data is short. These are bit-wise matrix multiplications in the well-studied Toeplitz matrix hashing construction of [12, 18] that we mentioned in Footnote 1 and finite fields modular reductions in polynomial universal hashing schemes of [5, 11, 28]. Both of these are discussed in Annexes A and B.

[6] We note that there is a subtle difference between $digest'()$ and PolyR16_32() of [14] which is defined as follows PolyR16_32(k, m) = PolyQ32$(k_1$, PolyQ16$(k_2, m_2) \parallel m_1)$. Here PolyQ32 and PolyQ16 produce 32- and 16-bit outputs respectively. The idea here is to hash short messages directly with PolyQ16, but hash significant longer messages with a hybrid scheme. As a result PolyR16_32 is faster than PolyQ32, but its collision probability is still dependent on message length. This is not the case with $digest'()$, which partly explains why we need to a rather longer key of $K = B + b + 80$ bits to reach $\epsilon_c = 2^{1-b}$ regardless of the message length.

References

1. http://www.cs.ox.ac.uk/publications/publication5935-abstract.html
2. *Simple Pairing White Paper.* See: www.bluetooth.com/NR/rdonlyres/0A0B3F36-D15F-4470-85A6-F2CCFA26F70F/0/SimplePairing_WP_V10r00.pdf
3. http://software.intel.com/en-us/articles/carry-less-multiplication-and-its-usage-for-computing-the-gcm-mode/
4. J. Black, S. Halevi, H. Krawczyk, T. Krovetz, P. Rogaway. *UMAC: Fast and Secure Message Authentication.* CRYPTO, LNCS vol. 1666, pp. 216-233, 1999.
5. B. den Boer. *A simple and key-economical unconditional authentication scheme.* Journal of Computer Security **2** (1993), 65-71.
6. J.L. Carter and M.N. Wegman. *Universal Classes of Hash Functions.* Journal of Computer and System Sciences, **18** (1979), 143-154.
7. M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. *A reliable randomized algorithm for the closest-pair problem.* Journal Algorithms, **25** (1997), 19-51.
8. C. Gehrmann, C. Mitchell and K. Nyberg. *Manual Authentication for Wireless Devices.* RSA Cryptobytes, vol. 7, no. 1, pp. 29-37, 2004.
9. S. Halevi and H. Krawczyk. *MMH: Software Message Authentication in the Gbit/second Rates.* FSE, LNCS vol. 1267, pp. 172-189, 1997.
10. ISO/IEC 9798-6, L.H. Nguyen, ed., 2010, *Information Technology – Security Techniques – Entity authentication – Part 6: Mechanisms using manual data transfer.*
11. 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, 1-11.
12. H. Krawczyk. *LFSR-based Hashing and Authentication.* Advances in Cryptology, CRYPTO 1994, LNCS vol. 839, 129-139.
13. H. Krawczyk. *New Hash Functions For Message Authentication.* Advances in Cryptology - Eurocrypt 1995, LNCS vol. 921, pp. 301-310.
14. T. Krovetz and P. Rogaway. em Fast Universal Hashing with Small Keys and no Preprocessing: the PolyR Construction. Information Security and Cryptology - ICICS 2000, LNCS vol. 2015, pp. 73-89, Springer, 2000.
15. S. Laur and K. Nyberg. *Efficient Mutual Data Authentication Using Manually Authenticated Strings.* LNCS vol. 4301, pp. 90-107, 2006.
16. 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.
17. 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.
18. 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.
19. A. Mashatan and D. Stinson. *Practical Unconditionally Secure Two-channel Message Authentication.* Designs, Codes and Cryptography **55** (2010), 169-188.
20. L.H. Nguyen and A.W. Roscoe. *Authentication protocols based on low-bandwidth unspoofable channels: A comparative survey.* Journal of Computer Security **19**(1): 139-201 (2011).
21. L.H. Nguyen and A.W. Roscoe. *Efficient group authentication protocol based on human interaction* FCS-ARSPA 2006, pp. 9-31.
22. L.H. Nguyen and A.W. Roscoe. *Authenticating ad-hoc networks by comparison of short digests.* Information and Computation **206**(2-4) (2008), 250-271.
23. L.H. Nguyen and A.W. Roscoe. *Separating two roles of hashing in one-way message authentication.* FCS-ARSPA-WITS 2008, pp. 195-210.
24. National Institute of Standards and Technology. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.* NIST Special Publication 800-38D, November, 2007.
25. 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.
26. Please see: http://www.aarongifford.com/computers/sha.html
27. D.R. Stinson. *Universal Hashing and Authentication Codes.* Advances in Cryptology - Crypto 1991, LNCS vol. 576, pp. 74-85, 1992.
28. R. Taylor. *An Integrity Check Value Algorithm for Stream Ciphers.* Advances in Cryptology, CRYPTO 1993. LNCS vol. 773, Springer-Verlag, pp. 40-48, 1994.
29. The source code and performance of UMAC can be found on this website: http://fastcrypto.org/umac/

30. 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.
31. 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.

A Toeplitz universal hashing

We first give the definition of a Toeplitz matrix.

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 $(\mathbb{F}_2)^{M\times b}$.

Krawczyk [12] and Mansour [18] independently introduce the following scheme, where the symbol '×' in Equation 5 represents a product of vector m and matrix $A(k)$ over $\mathbb{F}_2$.

$$h^T(k, m) = m \times A(k) \tag{5}$$

If key k is drawn randomly from R, then the collision probability is 2^{-b} which is optimal. For use in manual authentication protocols of Section 5, we define $h(k, m) = h^T(k_1, hash(m \parallel k_2))$ where $k = k_1 \parallel k_2$. This obtains $\epsilon_c = 2^{-b} + \theta_c$ where θ_c is the hash collision probability of $hash()$.

B Polynomial universal hashing

We first define the following n-bit output polynomial universal hashing scheme $\mathrm{PH}_{n,p}$ adapted from [5, 11, 28], where p is the largest prime number less than 2^n. This unversal hash function takes a n-bit key $k \in \mathbb{F}_p$ and a $2n$-bit data $m = m_1 \parallel m_2$, and produces an output in $\mathbb{F}_p$.

$$\mathrm{PH}_{n,p}(k, m) = k * m_1 + m_2 \pmod p$$

It is not difficult to show that the collision probability of this construction is $1/p$.

Suppose that we can hash an arbitrarily long message m into a $4b$-bit value by using a cryptographic hash function then our construction uses two different instances of the above polynomial hashing scheme, namely PH_{b,p_1} and PH_{2b,p_2} where p_1 and p_2 are the biggest prime numbers less than 2^b and 2^{2b} respectively.

$$h(k, m) = \mathrm{PH}_{b,p_1}(k_1, \mathrm{PH}_{2b,p_2}(k_2, hash(m \parallel k_3)))$$

Here $k = k_1 \parallel k_2 \parallel k_3$, where $k_1 \in \mathbb{F}_{p_1}$, $k_2 \in \mathbb{F}_{p_2}$ and k_3 is at least 80 bits.

The collision probability of this construction is therefore $\epsilon_c = 1/p_1 + 1/p_2 + \theta_c$, where θ_c denotes the hash collision probability on random messages of $hash()$. Since $p_2 \gg p_1$ and $1/p_1 \gg \theta_c$, we can deduce that $\epsilon_c \approx 1/p_1 \approx 2^{-b}$.