The proceedings version of this paper appears in *Advances in Cryptology—CRYPTO '99* [7]. This is the full version. It is available from `www.cs.ucdavis.edu/~rogaway/umac/`

UMAC: Fast and Secure Message Authentication

J. BLACK * S. HALEVI † H. KRAWCZYK ‡ T. KROVETZ * P. ROGAWAY *

September 8, 1999

Abstract

We describe a message authentication algorithm, UMAC, which authenticates messages (in software, on contemporary machines) roughly an order of magnitude faster than current practice (e.g., HMAC-SHA1), and about twice as fast as previously reported times for the universal hash-function family MMH. To achieve such speeds, UMAC uses a new universal hash-function family, NH, and a design which allows effective exploitation of SIMD parallelism. The "cryptographic" work of UMAC is done using standard primitives of the user's choice, such as a block cipher or cryptographic hash function; no new heuristic primitives are developed here. Instead, the security of UMAC is rigorously proven, in the sense of giving exact and quantitatively strong results which demonstrate an inability to forge UMAC-authenticated messages assuming an inability to break the underlying cryptographic primitive. Unlike conventional, inherently serial MACs, UMAC is parallelizable, and will have ever-faster implementation speeds as machines offer up increasing amounts of parallelism. We envision UMAC as a practical algorithm for next-generation message authentication.

Key words: Fast software cryptography, Message authentication codes, Provable security, Universal hashing.

* Dept. of Computer Science, University of California, Davis, California, 95616, USA. E-mail: `{blackj, krovetz, rogaway}@cs.ucdavis.edu` WWW: `www.cs.ucdavis.edu/~{blackj, krovetz, rogaway}/`

† IBM T.J. Watson Research Center, Yorktown Heights, New York, 10598, USA. E-mail: `shaih@watson.ibm.com`

‡ Department of Electrical Engineering, Technion, Haifa, Israel, and IBM T.J. Watson Research Center, New York, USA. Email: `hugo@ee.technion.ac.il`

Contents

1 Introduction

This paper describes a new message authentication code, UMAC, and the theory that lies behind it. UMAC has been designed with two main goals in mind:

- **Extreme speed.** We have aimed to create the fastest MAC ever described, and by a wide margin. We are speaking of speed with respect to software implementations on contemporary general-purpose computers.
- **Provable security.** We insist that the MAC be demonstrably secure, by virtue of proofs carried out in the sense of provable-security cryptography. Namely, if the underlying cryptographic primitives are secure (we use pseudorandom functions) then the MAC is secure, too.

There were of course other goals, avoiding excessive conceptual and implementation complexity being principal among them.

UMAC is certainly fast. On our 350 MHz Pentium II PC, one version of UMAC (where the adversary has 2^{-60} chance of forgery) achieves peak-performance of 2.9 Gbits/sec (**0.98** cycles/byte). Another version of UMAC (with 2^{-30} chance of forgery) achieves peak performance of 5.6 Gbits/sec (**0.51** cycles/byte). For comparison, our SHA-1 implementation runs at **12.6** cycles/byte. Note that SHA-1 speed upper bounds the speed of HMAC-SHA1 [3], a software-oriented MAC representative of the speeds achieved by current practice. The previous speed champion among proposed universal hash functions (the main ingredient for making a fast MAC; see below) was MMH [13], which runs at about **1.2** cycles/byte (for 2^{-30} chance of forgery) under its originally envisioned implementation.

How has it been possible to achieve these speeds? Interestingly, we have done this with the help of our second goal, provable security. We use the well-known universal-hashing approach to message authentication, introduced by [26], making innovations in its realization. Let us now review this approach and its advantages, and then describe what we have done to make it fly.

1.1 Universal-Hashing Approach

UNIVERSAL HASHING AND AUTHENTICATION. Our starting point is a universal hash-function family [8]. (Indeed the "U" in UMAC is meant to suggest the central role that universal hash-function families play in this MAC.) Remember that a set of hash functions is said to be "ϵ-universal" if for any pair of distinct messages, the probability that they collide (hash to the same value) is at most ϵ. The probability is over the random choice of hash function.

As described in [26], a universal hash-function family can be used to build a secure MAC. The parties share two things: a secret and randomly chosen hash function from the universal hash-function family, and a secret encryption key. A message is authenticated by hashing it with the shared hash function and then encrypting the resulting hash (using the encryption key). Wegman and Carter showed that when the hash-function family is strongly universal (a similar but stronger property than the one we defined) and the encryption is realized by a one-time pad, the adversary cannot forge with probability better than that obtained by choosing a random string for the MAC.

WHY UNIVERSAL HASHING? As suggested many times before, the above approach is a promising one for building a highly-secure and ultra-fast MAC [16, 23, 13]. The reasoning is like this. The speed of a universal-hashing MAC depends on the speed of the hashing step and the speed of the encrypting step. But if the hash function compresses messages well (i.e., its output is short) then the encryption shouldn't take long simply because it is a short string that is being encrypted. On the other hand, since the combinatorial property of the universal hash-function family is mathematically proven (making no cryptographic hardness assumptions), it needs no "over-design" or "safety margin" the way a cryptographic primitive would. Quite the opposite: the hash-function family might as well be the fastest, simplest thing that one can prove universal. And that should be much faster than any sort of "cryptographic" primitive.

Equally important, the above approach makes for desirable security properties. Since the cryptographic primitive is applied only to the (much shorter) hashed image of the message, we can select a cryptographically conservative design for this step and pay with only a minor impact on speed. Further, the underlying cryptographic primitive is used only on short and secret messages, eliminating the typical avenues of attack. Under this approach security and efficiency are not conflicting requirements—quite to the contrary, they go hand in hand.

THE QUEST FOR FAST UNIVERSAL HASHING. At least in principle, the universal-hashing paradigm has reduced the problem of fast message authentication to the problem of fast universal hashing. Thus there has been much recent work on the design of fast-to-compute universal hash-function families. Here is a brief overview of some of this work; a more complete overview is given in Section 1.3. Krawczyk [16] describes a "cryptographic CRC" construction, which has very fast hardware implementations and reasonably fast software implementations; it needs about 6 cycles/byte, as shown by Shoup [24]. Rogaway's "bucket hashing" construction [23] was the first universal hash-function family explicitly targeted for fast software implementation; it runs in about 1.5–2.5 cycles/byte. Halevi and Krawczyk devised the MMH hash-function family [13], which takes advantage of current CPU trends and hashes at about 1.5–3 cycles/byte on modern CPUs.

With methods now in hand which hash so very quickly, one may ask if the hash-design phase of making a fast MAC is complete; after all, three cycles/byte may already be fast enough to keep up with high-speed network traffic. However, authenticating information at the rate that it is generated or transmitted is not the real goal. The goal is to use the smallest possible fraction of the CPU's cycles by the simplest possible hash mechanism, and having the best proven bounds. In this way most of the machine's cycles are available for other work.

1.2 Our Contributions

UMAC represents the next step in the quest for a fast and secure MAC. Here we describe the main contributions associated to its design.

NEW HASH FUNCTION FAMILIES AND THEIR TIGHT ANALYSES. A hash-function family named NH underlies hashing in UMAC. It is a simplification of the MMH and NMH families described in [13]. The family NH works like this. The message M to hash is regarded as a sequence of an even number ℓ of integers, $M = (m_1, \ldots, m_\ell)$, where each $m_i \in \{0, \ldots, 2^w - 1\}$ corresponds to a w-bit word (e.g., $w = 16$ or $w = 32$). A particular hash function is named by a sequence of $n \geq \ell$ w-bit integers $K = (k_1, \ldots, k_n)$. The NH function is then computed as

$$\mathsf{NH}_K(M) = \left(\sum_{i=1}^{\ell/2} ((m_{2i-1} + k_{2i-1}) \bmod 2^w) \cdot ((m_{2i} + k_{2i}) \bmod 2^w) \right) \bmod 2^{2w} . \tag{1}$$

The novelty of this method is that all the arithmetic is "arithmetic that computers like to do"—no finite fields or non-trivial modular reductions come into the picture.

Despite the non-linearity of this hash function and despite its being defined using two different rings, $Z/2^w$ and $Z/2^{2w}$, not a finite field, we manage to obtain a tight and desirable bound on the collision probability: 2^{-w}. Earlier analyses of related hash-function families had to give up a small constant in the analysis [13]. We give up nothing.

After proving our bounds on NH we go on and extend the method using the "Toeplitz construction." This is a well-known approach to reduce the error probability without much lengthening of the key [17, 16]. Prior to our work the Toeplitz construction was known to work only for *linear* functions over *fields*. Somewhat surprisingly, we prove that it also works for NH. Here again our proof achieves a tight bound for the collision probability. We then make further extensions to handle length-issues, and allow other optimizations, finally arriving at the hash-function family actually used in UMAC.

COMPLETE SPECIFICATION. Previous work on universal-hash-paradigm MACs dealt with fast hashing but did not address in detail the next step of the process—how to embed a fast-to-compute hash function into a *concrete, practical, and fully analyzed* MAC. For some hash-function constructions (e.g., cryptographic CRCs) this step would be straightforward. But for the fastest hash families it is not, since these hash functions have some unpleasant characteristics, including specialized domains, long key-lengths, long output-lengths, or good performance only on very long messages. It had never been demonstrated that these difficulties could be overcome in a practical way which would deliver on the promised speed. This paper shows that they can. We provide a complete specification of UMAC, a ready-to-use MAC, in a separate specification document [6]). The technical difficulties encountered in embedding our hash-function family into a MAC are not minimized; they are treated with the same care as the hash-function family itself. The construction is fully analyzed, beginning-to-end. What is analyzed is exactly what is specified; there is no "gap" which separates them. This has only been possible by co-developing the specification document and the academic paper.

PRF(HASH, NONCE) CONSTRUCTION. Previous work has assumed that one hashes messages to some fixed length string (e.g., 64 bits) and then the cryptographic primitive is applied. But using universal hashing to reduce a very long message into a fixed-length one can be complex, require long keys, or reduce the quantitative security. Furthermore, hashing the message down by a factor of 128, say, already provides much of the speed and security benefits. So we reduce the length of the message by some pre-set constant, concatenate a sender-generated nonce, and then apply a pseudorandom function (PRF) to the shorter (but still unbounded-length) string. The nonce is communicated along with the image of the PRF. We call this the PRF(HASH, Nonce)-construction. We analyze it and compare it with alternatives.

EXTENSIVE EXPERIMENTATION. In defining UMAC we have been guided by extensive experimentation. Through this we have identified several parameters that influence the speed which UMAC delivers. Our studies illuminate how these algorithmic details shape performance. Our experiments have made clear that while any reasonable version of UMAC (i.e., any reasonable setting of the parameters) should out-perform any conventional MAC, the fastest version of UMAC for one platform will often be different from the fastest version for another platform. We have therefore kept UMAC a parameterized construction, allowing some specific choices to be fine-tuned to the application or platform at hand. In this paper and in [6] we consider a few reasonable settings for these parameters.

SIMD EXPLOITATION. Unlike conventional, inherently serial MACs, UMAC is parallelizable, and will have ever-faster implementations as machines offer up increasing amounts of parallelism. Our algorithm and our specification were specifically designed to allow implementations to exploit the form of parallelism offered up in current and emerging SIMD architectures (Single Instruction

Multiple Data). These architectures provide some long registers that can, in certain instructions, be treated as vectors of smaller-sized words. For NH to run well we must be able to quickly multiply w-bit numbers ($w = 16$ or $w = 32$) into their w-bit product. Many modern machines let us do this particularly well since we can re-appropriate instructions for vector dot-products that were primarily intended for multimedia computations. One of our fastest implementations of UMAC runs on a Pentium and makes use of its MMX instructions which treat a 64-bit register as a vector of four 16-bit words. UMAC is the first MAC specifically constructed to exploit SIMD designs.

1.3 Related Work

MMH Paper. Halevi and Krawczyk investigated fast universal hashing in [13]. Their MMH construction takes advantage of improving CPU support for integer multiplication, particularly the ability to quickly multiply two 32-bit multiplicands into a 64-bit product. Besides MMH, [13] describes a (formerly-unpublished) hash-function family of Carter and Wegman, NMH*, and a variation of it called NMH. Our NH function, as described in formula (1), is a (bound-improving) simplification of NMH. The difference between NH and NMH is that NMH uses an additional modular reduction by a prime close to 2^w, followed by a reduction modulo 2^w. Similarly, NMH* is the same as NH, as given in formula (1) except the mods are omitted and the arithmetic is in Z/p, where p is prime.

Other Work On Universal Hashing For MACs. Krawczyk describes a “cryptographic CRC” which has very fast hardware implementations [16]. Shoup later studied the software performance of this construction, and gave several related ones [24]. In [16] one also finds the Toeplitz construction used in a context similar to ours. An earlier use of the Toeplitz construction, in a different domain, can be found in [17].

A hash-function family specifically targeted for software was Rogaway's “bucket hashing” [23]. Its peak speed is fast but its long output length makes it suitable only for long messages.

Nevelsteen and Preneel give a performance study of several universal hash functions proposed for MACs [18]. Patel and Ramzan give an MMH-variant that can be more efficient than MMH in certain settings [9].

Bernstein reports he has designed and implemented a polynomial-evaluation style hash-function family that has a collision probability of around 2^{-96} and runs in 4.5 Pentium cycles/byte [5]. Other recent work about universal hashing for authentication includes [1, 14].

Other Types Of MACs. Most concrete MACs have been constructed from other cryptographic primitives. The most popular choice of cryptographic primitive has been a block cipher, with the most popular construction being the CBC-MAC [2]. This MAC was analyzed by [4]. An extension of this analysis was carried out by [19], this extension being relevant for how we suggest using a block cipher to make a PRF.

More recently, MACs have been constructed from cryptographic hash-functions. It has usually been assumed that this would lead to faster MACs than the CBC-MAC. A few such methods are described in [25, 15], and analysis appears in [20, 21]. An increasingly popular MAC of this cryptographic-hash-function variety is HMAC [3, 12]. In one version of UMAC we suggest using HMAC as the underlying PRF. One can view UMAC as an alternative to HMAC, with UMAC being faster but more complex.

Other Versions. The proceedings version of this paper appears in [7].

Subkey generation:
Using a PRG, map Key to $K = K_1K_2 \cdots K_{1024}$, with each K_i a 32-bit word, and
to A, where $|A| = 512$.

Hashing the message Msg ***to*** $HM = \mathsf{NHX}_{Key}(Msg)$:
Let Len be $|Msg| \bmod 4096$, encoded as a 2-byte string.
Append to Msg the minimum number of 0 bits to make $|Msg|$ divisible by 64.
Let $Msg = Msg_1 \parallel Msg_2 \parallel \cdots \parallel Msg_t$ where each Msg_i is 1024 words except for Msg_t,
which has between 2 and 1024 words.
Let $HM = \mathsf{NH}_K(Msg_1) \parallel \mathsf{NH}_K(Msg_2) \parallel \cdots \parallel \mathsf{NH}_K(Msg_t) \parallel Len$

Computing the authentication tag:
The tag is $Tag = \text{HMAC-SHA1}_A\,(Nonce \parallel HM)$

Figure 1: *An illustrative special case of* UMAC. *The algorithm above computes a 160-bit tag given Key, Msg, and Nonce. See the accompanying text for the definition of* NH.

2 Overview of UMAC

Unlike many MACs, our construction is *stateful* for the sender: when he wants to authenticate some string Msg he must provide as input to UMAC (along with Msg and the shared key Key) a 64-bit string $Nonce$. The sender must not reuse the nonce within the communications session. Typically the nonce would be a counter which the sender increments with each transmitted message.

The UMAC algorithm specifies how the message, key, and nonce determine an *authentication tag*. The sender will need to provide the receiver with the message, nonce, and tag. The receiver can then compute what "should be" the tag for this particular message and nonce, and see if it matches the tag actually received. The receiver might also wish to verify that the nonce has not been used already; doing this is a way to avoid replay attacks.

Like many modern ciphers, UMAC employs a *subkey generation process* in which the underlying (convenient-length) key is mapped into UMAC's internal keys. In typical applications subkey generation is done just once, at the beginning of a long-lived session, and so subkey-generation is usually not performance-critical.

UMAC depends on a few different *parameters*. We begin by giving a description of UMAC as specialized to one particular setting of these parameters. Then we briefly explore the role of various parameters.

2.1 An Illustrative Special Case

Refer to Figure 1. The underlying key Key (which might be, say, 128 bits) is first expanded into internal keys K and A, where K is 1024 words (a *word* being 32-bits) and A is 512 bits. How Key determines K and A is a rather unimportant and standard detail (it can be handled using any PRG), and so we omit its description.

Figure 1 refers to the hash function NH, which is applied to each block $Msg_1, \ldots, Msg_t$ of Msg. Let $M = Msg_j$ be one of these blocks. Regard M as a sequence $M = M_1 \cdots M_\ell$ of 32-bit words, where $2 \le \ell \le 1024$. The hash function is named by $K = K_1 \cdots K_{1024}$, where K_i is 32 bits. We let

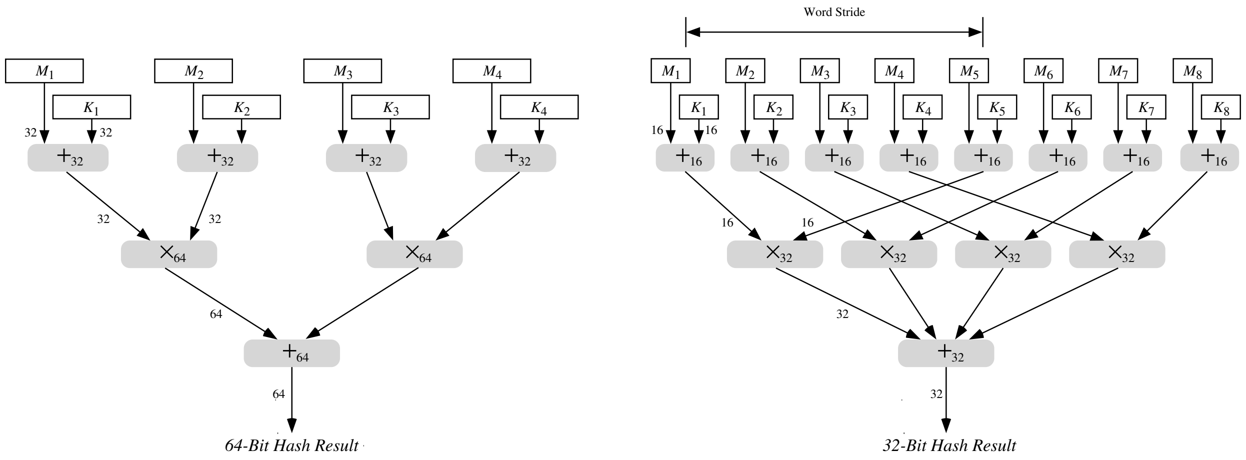


Figure 2: ***Left:*** *The* NH *hash function in its "basic" form, using a wordsize of* $w = 32$ *bits. This type of hashing underlies* UMAC-STD-30 *and* UMAC-STD-60. ***Right:*** *The "strided" form of* NH*, using a wordsize of* $w = 16$ *bits. This type of hashing underlies* UMAC-MMX-30 *and* UMAC-MMX-60.

$\mathsf{NH}_K(M)$ be the 64-bit string

$$\mathsf{NH}_K(M) \;=\; (M_1 +_{32} K_1) \times_{64} (M_2 +_{32} K_2) \;+_{64}\; \cdots \;+_{64}\; (M_{\ell-1} +_{32} K_{\ell-1}) \times_{64} (M_\ell +_{32} K_\ell)$$

where $+_{32}$ is computer addition on 32-bit strings to give their 32-bit sum, $+_{64}$ is computer addition on 64-bit strings to give their 64-bit sum, and $\times_{64}$ is computer multiplication on unsigned 32-bit strings to give their 64-bit product. This description of NH is identical to Equation (1) (for $w = 32$) but emphasizes that all the operations we are performing directly correspond to machine instructions of modern CPUs. See the left-hand side of Figure 2 for a picture.

Theorem 4.2 says that NH is 2^{-32}-universal with respect to strings of equal and appropriate length. Combining with Proposition 6.1 gives that NHX (as described in Figure 1) is 2^{-32}-universal, but now for any pair of strings. Now by Theorem 8.2, if an adversary could forge a message with probability $2^{-32} + \delta$ then an adversary of essentially the same computational complexity could break HMAC-SHA1 (as a PRF) with advantage $\delta - 2^{-160}$. But analysis in [3] indicates that one can't break HMAC-SHA1 (as a PRF) given generally-accepted assumptions about SHA-1.

2.2 UMAC Parameters

The full name of the version of NH just described is $\mathsf{NH}[n, w]$, where $n = 1024$ and $w = 32$: the *wordsize* is $w = 32$ bits and the *blocksize* is $n = 1024$ words. The values of n and w are examples of two of UMAC's parameters. Let us describe a few others.

Naturally enough, the *pseudorandom function* (PRF) which gets applied to *Nonce* || *HM* is a parameter. We used HMAC-SHA1, but any PRF is allowed. Similarly, it is a parameter of UMAC how *Key* gets mapped to K and A.

The universal hashing we used in our example had collision probability 2^{-32}. We make provisions for lowering this (e.g., to 2^{-64}). To square the collision probability one could of course hash the message twice, using independent hash keys, and concatenate the results. But an important optimization in UMAC is that the two keys that are used are not independent; rather, one key is the "shift" of the other, with a few new words coming in. This is the well-known "Toeplitz construction." We prove in Theorem 5.1 that, for NH, the probability still drops according to the square.

In our example we used a long subkey K—it had 4096 bytes. To get good compression with a shorter subkey we can use two-level (2L) hashing. If a hash key of length n_1 gives compression ratio λ_1 and a hash key of length n_2 gives compression ratio λ_2 then using two levels of hashing gives compression ratio $\lambda_1\lambda_2$ with key size $\ell_1 + \ell_2$. Our specification allows for this. In fact, we allow 2L hashing in which the Toeplitz shift is applied at each level. It turns out that this only loses a factor of two in the collision probability. The analysis is rather complex, and is omitted.

To accommodate SIMD architectures in the computation of NH we allow slight adjustments in the indexing of NH. For example, to use the MMX instructions of the Pentium processor, instead of multiplying $(M_1 +_{16} K_1)$ by $(M_2 +_{16} K_2)$ and $(M_3 +_{16} K_3)$ by $(M_4 +_{16} K_4)$, we compute

$$(M_1 +_{16} K_1) \times_{32} (M_5 +_{16} K_5) \ +_{32}\ (M_2 +_{16} K_2) \times_{32} (M_6 +_{16} K_6) \ +_{32}\ \ldots$$

There are MMX instructions which treat each of two 64-bit words as four 16-bit words, corresponding words of which can be added or multiplied to give four 16-bit sums or four 32-bit products. Reading $M_1 \| M_2 \| M_3 \| M_4$ into one MMX register and $M_5 \| M_6 \| M_7 \| M_8$ into another, we are well-positioned to multiply $M_1 +_{16} K_1$ by $M_5 +_{16} K_5$, not $M_2 +_{16} K_2$. See Figure 2.

There are a few more parameters. The *sign* parameter indicates whether the arithmetic operation $\times_{64}$ is carried out thinking of the strings as unsigned (non-negative) or signed (two's-complement) integers. The signed version of NH requires a slightly different analysis and loses a factor of two in the collision probability. This loss is inherent in the method; our analysis is tight.

If the input message is sufficiently short there is no speed savings to be had by hashing it. The *min-length-to-hash* parameter specifies the minimum-length message which should be hashed.

An *endian* parameter indicates if the MAC should favor big-endian or little-endian computation.

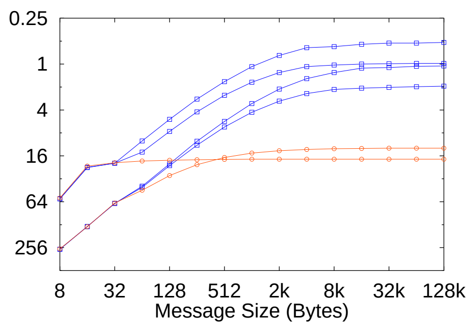
Named Parameter Sets. In [6] we suggest five different settings for the vector of parameters, giving rise to UMAC-STD-30, UMAC-STD-60, UMAC-MMX-15, UMAC-MMX-30, and UMAC-MMX-60. We summarize their salient features.

UMAC-STD-30 and UMAC-STD-60 use a wordsize of $w = 32$ bits. They employ 2L hashing with a compression factor of 32 followed by a compression factor of 16. This corresponds to a subkey K of about 400 Bytes. They employ HMAC-SHA1 as the underlying PRF. They use signed arithmetic. The difference between UMAC-STD-30 and UMAC-STD-60 are the collision bounds (and therefore forgery bounds): 2^{-30} and 2^{-60}, respectively, which are achieved by hashing either once or twice. These two versions of UMAC perform well on a wide range of contemporary processors.

UMAC-MMX-15, UMAC-MMX-30 and UMAC-MMX-60 are well-suited for exploiting the SIMD-parallelism available in the MMX instruction set of Intel processors. They use wordsize $w = 16$ bits. Hashing is accomplished with a single-level scheme and a hash key of about 4 KBytes, which yields the same overall compression ratio as the 2L scheme used in the UMAC-STD variants. These MACs use the CBC-MAC of a software-efficient block cipher as the basis of the underlying PRF. Our tests were performed using the block cipher RC6 [22]. Arithmetic is again signed. The difference between UMAC-MMX-15, UMAC-MMX-30 and UMAC-MMX-60 is the maximal forgery probability: 2^{-15}, 2^{-30} and 2^{-60}, respectively.

3 UMAC Performance

Fair performance comparisons are always difficult to make, but they are particularly difficult for UMAC. First, UMAC depends on a number of parameters, and the most desirable setting for these



	Pentium II		PowerPC		Alpha	
UMAC-STD-60	1.49	(1.93)	1.81	(1.58)	1.03	(2.78)
UMAC-STD-30	2.79	(1.03)	2.28	(1.26)	1.79	(1.60)
UMAC-MMX-60	2.94	(0.98)	4.21	(0.66)	0.287	(10.0)
UMAC-MMX-30	5.66	(0.51)	7.20	(0.39)	0.571	(5.02)
UMAC-MMX-15	8.47	(0.33)	10.5	(0.27)	0.981	(2.85)
CBC-MAC-RC6	0.162	(17.7)	0.210	(13.7)	0.068	(42.5)
HMAC-SHA1	0.227	(12.6)	0.228	(12.6)	0.117	(24.5)

Figure 3: UMAC Performance. ***Left:*** *Performance over various message lengths on a Pentium II, measured in machine cycles/byte. The lines in the graph correspond to the following MACs (beginning at the top-right and moving downward):* UMAC-MMX-30, UMAC-MMX-60, UMAC-STD-30, UMAC-STD-60, HMAC-SHA1 *and* CBC-MAC-RC6. ***Right:*** *Peak performance for three architectures measured in Gbits/sec (cycles/byte). The Gbits/sec numbers are normalized to 350 MHz.*

parameters differs from platform to platform. In some usage scenarios it is realistic to assume the most desirable set of parameters, though often it is not. Thus it is unclear which parameters should be selected and tested on which platforms. Second, UMAC performance varies with message length. The advantage UMAC delivers over conventional MACs increases as messages get longer. What message lengths should be chosen for performance tests?

To address the first issue we report performance characteristics for a few different parameter choices. To address the second issue we report timing data for a variety of message lengths.

TEST ENVIRONMENT. We implemented the four UMAC variants named in Section 2.2 on three representative hardware platforms: a 350 MHz Intel Pentium II, a 200 MHz IBM/Motorola PowerPC 604e, and a 300 MHz DEC Alpha 21164. We also implemented the UMAC-MMX variants on a pre-release PowerPC 7400 equipped with AltiVec SIMD instructions and running at 266 MHz. Performing well on these platforms is important for acceptance of UMAC but also is an indication that the schemes will work well on future architectures. The SIMD operations in the MMX and AltiVec equiped processors and the 64-bit register size of the Alpha are both features we expect to become more prevalent in future processors.

All tests were written in C with a few functions written in assembly. For the Pentium II we wrote assembly for RC6, SHA-1, and the first-level NH hashes. For the PowerPC 604e we wrote in assembly just the first-level NH hashes. In both cases, the number of lines of assembly written was small—about 70–90 lines. No assembly code was written for the Alpha or AltiVec implementations.

For each combination of options we determined the scheme's throughput on variously sized messages, eight bytes through 512 KBytes. The experimental setup ensured that messages resided in level-1 cache regardless of their length. For comparison the same tests were run for HMAC-SHA1 [10] and the CBC-MAC of a fast block cipher, RC6 [22].

RESULTS. The graph in Figure 3 shows the throughput of five versions of UMAC, as well as HMAC-SHA1 and CBC-MAC-RC6, all run on our Pentium II. The table gives peak throughput for the same MACs, but does so for all three architectures. When reporting throughput in Gbits/second the meaning of "Gbit" is 10^9 bits (as opposed to 2^{30} bits). The performance curves for the Alpha and PowerPC look similar to the Pentium II—they perform better than the reference MACs at around the same message length, and level out at around the same message length.

UMAC performs best on long messages because the hash function is then most effective at reducing the amount of data acted upon by the PRF. For our choice of tested parameters the maximum compression ratio is achieved on messages of 4 KBytes and up, which is why the curves in Figure 3 level out at around 4 KBytes. Still, reasonably short messages (a couple hundred bytes) already see substantial benefit from the hashing. For short messages about half the time spent by UMAC-MMX-60 authenticating is spent in the PRF, but that ratio drops to around 7% for longer messages.

When messages are short the portion of time spent computing the PRF is higher, making the choice of PRF more significant. Our tests show that, all other parameters being the same, an RC6-based UMAC is about 50% faster than one based on SHA-1 for messages of length 32 bytes, but this advantage drops to 10% - 20% for 4 KByte messages and nearly vanishes for 64 KByte messages.

In general, the amount of work spent hashing increases as word size decreases because the number of arithmetic operations needed to hash a fixed-length message is inversely related to word size. The performance of UMAC on the Alpha demonstrates this clearly. On UMAC-MMX-60 ($w = 16$) it requires 10 cycles per byte to authenticate a long message, while it requires 2.8 for UMAC-STD-60 ($w = 32$) and only 1.7 for a test version which uses $w = 64$ bit words[1]. Perhaps our most surprising experimental finding was how, on some processors, we could dramatically improve performance by going from words of $w = 32$ bits to words of $w = 16$ bits. Such a reduction in word size might appear to vastly increase the amount of work needed to get to a given collision bound. But a single MMX instruction which UMAC uses heavily can do four 16-bit multiplications and two 32-bit additions, and likewise a single AltiVec instruction can do eight 16-bit multiplications and eight 32-bit additions. This is much more work per instruction than the corresponding 32-bit instructions.

UMAC-STD uses only one-tenth as much hash key as UMAC-MMX to achieve the same compression ratio. The penalty for such 2L hashing ranges from 8% on small messages to 15% on long ones. To lower the amount of key material we could have used a one-level hash with a smaller compression ratio, but experiments show this is much less effective: relative to UMAC-MMX-60, which uses about 4 KBytes of hash key, a 2 KBytes scheme goes 85% as fast, a 1 KByte scheme goes 66% as fast, and a 512 bytes scheme goes 47% as fast.

The endian-orientation of the UMAC version being executed has little effect on performance for the PowerPC and Pentium II systems. Both support swapping the endian-orientation of words efficiently. On the Alpha, and most other systems, such reorientation is not part of the architecture and must be done with an expensive series of primitive operations. Key setup on our 350 MHz Pentium II took 450 μs for UMAC-MMX-60 and 130 μs for UMAC-STD-60, or about 158,000 and 46,000 cycles, respectively.

To answer some questions we had about UMAC performance we implemented it with several variations. In one supplementary experiment we replaced the NH hash function used in UMAC-STD-30 by MMH [13]. Peak performance dropped by 24%. We replaced the NH hash function of UMAC-MMX-30 by a 16-bit MMH (implemented using MMX instructions) and peak performance dropped by 5%. Thus our performance gains (compared to prior art) are due more to SIMD exploitation than the difference between NH and MMH.

The hash family MMH can be simplified by eliminating its last two modular reductions. We refer to the resulting hash function family as MH, and it is defined by $\mathsf{MH}_K(M) = \sum_{i=1}^{\ell} m_i k_i \bmod 2^{2w}$. An MH-version of UMAC-MMX-30 ($w = 16$) was about 5% slower than (our NH-version of)

[1]Another reason UMAC-MMX-60 does poorly on non-SIMD processors is because of additional overhead required to load small words into registers and then sign-extend or zero the upper bits of those registers.

UMAC-MMX-30, while an MH-version of UMAC-STD-30 ($w = 32$) was about 26% slower than (our NH-version of) UMAC-STD-30. Finally, we coded UMAC-STD-30 and UMAC-STD-60 for a Pentium processor which lacked MMX. Peak speeds were 2.2 cycles/byte and 4.3 cycles/byte—still 3 to 6 times faster than methods like HMAC-SHA1. Even though these versions of UMAC do not use MMX instructions, Intel processors without MMX are also slower than the Pentium II at integer multiplication, an important operation in UMAC.

4 The NH Hash Family

We now define and analyze NH, the hash-function family which underlies UMAC. Recall that NH is not, by itself, the hash-function family which UMAC uses, but the basic building block from which we construct UMAC's hash families.

4.1 Preliminaries

FUNCTION FAMILIES. A *family of functions* (with domain $A \subseteq \{0,1\}^*$ and range $B \subseteq \{0,1\}^*$) is a set of functions $\mathsf{H} = \{h : A \to B\}$ endowed with some distribution. When we write $h \leftarrow \mathsf{H}$ we mean to choose a random function $h \in \mathsf{H}$ according to this distribution. A family of functions is also called a *family of hash functions* or a *hash-function family*.

Usually we specify a family of functions H by specifying some finite set of strings, Key, and explaining how each string $K \in \mathsf{Key}$ names some function $\mathsf{H}_K \in \mathsf{H}$. We may then think of H not as a set of functions from A to B but as a single function $\mathsf{H} : \mathsf{Key} \times A \to B$, whose first argument we write as a subscript. A random element $h \in \mathsf{H}$ is determined by selecting uniformly at random a string $K \in \mathsf{Key}$ and setting $h = \mathsf{H}_K$.

UNIVERSAL HASHING. We are interested in hash-function families in which "collisions" (when $h(M) = h(M')$ for distinct M, M') are infrequent. The formal definition follows.

Definition 4.1 Let $\mathsf{H} = \{h : A \to B\}$ be a family of hash functions and let $\epsilon \geq 0$ be a real number. We say that H is ϵ-universal, denoted ϵ-AU, if for all distinct $M, M' \in A$, we have that $\Pr_{h \leftarrow \mathsf{H}}[h(M) = h(M')] \leq \epsilon$. We say that H is ϵ-universal on equal-length strings if for all distinct, equal-length strings $M, M' \in A$, we have that $\Pr_{h \leftarrow \mathsf{H}}[h(M) = h(M')] \leq \epsilon$. ▮

4.2 Definition of NH

Fix an even $n \geq 2$ (the "blocksize") and a number $w \geq 1$ (the "wordsize"). We define the family of hash functions $\mathsf{NH}[n, w]$ as follows. The domain is $A = \{0,1\}^{2w} \cup \{0,1\}^{4w} \cup \cdots \cup \{0,1\}^{nw}$ and the range is $B = \{0,1\}^{2w}$. Each function in $\mathsf{NH}[n, w]$ is named by a string K of nw bits; a random function in $\mathsf{NH}[n, w]$ is given by a random nw-bit string K. We write the function indicated by string K as $\mathsf{NH}_K(\cdot)$.

Let U_w and U_{2w} represent the sets $\{0, \ldots, 2^w - 1\}$ and $\{0, \ldots, 2^{2w} - 1\}$, respectively. Arithmetic done modulo 2^w returns a result in U_w; arithmetic done modulo 2^{2w} returns a result in U_{2w}. We overload the notation introduced in Section 2.1: for integers x, y let $(x +_w y)$ denote $(x+y) \bmod 2^w$. (Earlier this was an operator from strings to strings, but with analogous semantics.)

Let $M \in A$ and denote $M = M_1 \cdots M_\ell$, where $|M_1| = \cdots = |M_\ell| = w$. Similarly, let $K \in \{0,1\}^{nw}$ and denote $K = K_1 \cdots K_n$, where $|K_1| = \cdots = |K_n| = w$. Then $\mathsf{NH}_K(M)$ is defined as

$$\mathsf{NH}_K(M) \;=\; \sum_{i=1}^{\ell/2} (k_{2i-1} +_w m_{2i-1}) \cdot (k_{2i} +_w m_{2i}) \bmod 2^{2w}$$

where $m_i \in U_w$ is the number that M_i represents (as an unsigned integer), where $k_i \in U_w$ is the number that K_i represents (as an unsigned integer), and the right-hand side of the above equation is understood to name the (unique) $2w$-bit string which represents (as an unsigned integer) the U_{2w}-valued integer result. Henceforth we shall refrain from explicitly converting from strings to integers and back, leaving this to the reader's good sense. (We comment that for everything we do, one could use any bijective map from $\{0,1\}^w$ to U_w, and any bijective map from U_{2w} to $\{0,1\}^{2w}$.)

When the values of n and w are clear from the context, we write NH instead of $\mathsf{NH}[n,w]$.

4.3 Analysis

The following theorem bounds the collision probability of NH.

Theorem 4.2 *For any even $n \geq 2$ and $w \geq 1$, $\mathsf{NH}[n,w]$ is 2^{-w}-AU on equal-length strings.*

Proof: Let M, M' be distinct members of the domain A with $|M| = |M'|$. We are required to show

$$\Pr_{K \leftarrow \mathsf{NH}}\left[\, \mathsf{NH}_K(M) = \mathsf{NH}_K(M') \,\right] \leq 2^{-w}.$$

Converting the message and key strings to n-vectors of w-bit words we invoke the definition of NH to restate our goal as requiring

$$\Pr\left[\sum_{i=1}^{\ell/2} (k_{2i-1} +_w m_{2i-1})(k_{2i} +_w m_{2i}) = \sum_{i=1}^{\ell/2} (k_{2i-1} +_w m'_{2i-1})(k_{2i} +_w m'_{2i})\right] \leq 2^{-w}$$

where the probability is taken over uniform choices of $(k_1, \ldots, k_n)$ with each k_i in U_w. Above (and for the remainder of the proof) all arithmetic is carried out in $Z/2^{2w}$.

Since M and M' are distinct, $m_i \neq m'_i$ for some $1 \leq i \leq n$. Since addition and multiplication in a ring are commutative, we lose no generality in assuming $m_2 \neq m'_2$. We now prove that for any choice of $k_2, \cdots, k_n$ we have

$$\Pr_{k_1 \in U_w}\Bigg[\, (m_1 +_w k_1)(m_2 +_w k_2) \;+\; \sum_{i=2}^{\ell/2} (m_{2i-1} +_w k_{2i-1})(m_{2i} +_w k_{2i}) =$$

$$(m'_1 +_w k_1)(m'_2 +_w k_2) \;+\; \sum_{i=2}^{\ell/2} (m'_{2i-1} +_w k_{2i-1})(m'_{2i} +_w k_{2i}) \,\Bigg] \leq 2^{-w}$$

which will imply the theorem. Collecting up the summations, let

$$y = \sum_{i=2}^{\ell/2} (m_{2i-1} +_w k_{2i-1})(m_{2i} +_w k_{2i}) \;-\; \sum_{i=2}^{\ell/2} (m'_{2i-1} +_w k_{2i-1})(m'_{2i} +_w k_{2i})$$

and let $c = (m_2 +_w k_2)$ and $c' = (m'_2 +_w k_2)$. Note that c and c' are in U_w, and since $m_2 \neq m'_2$, we know $c \neq c'$. We rewrite the above probability as

$$\Pr_{k_1 \in U_w}\left[\, c(m_1 +_w k_1) - c'(m'_1 +_w k_1) + y = 0 \,\right] \leq 2^{-w}.$$

In Lemma 4.3 below, we prove that there can be at most one k_1 in U_w satisfying $c(m_1 +_w k_1) - c'(m'_1 +_w k_1) + y = 0$, yielding the desired bound. ▮

The above proof reduced establishing our bound to the following useful lemma, which is used again for Theorem 5.1.

Lemma 4.3 *Let c and c' be distinct values from U_w. Then for any $m, m' \in U_w$ and any $y \in U_{2w}$ there exists at most one $k \in U_w$ such that $c(k +_w m) = c'(k +_w m') + y$ in $Z/2^{2w}$.*

Proof: We note that it is sufficient to prove the case where $m = 0$: to see this, notice that if $c(k +_w m) = c'(k +_w m') + y$, then also $c(k^* +_w 0) = c'(k^* +_w m^*) + y$, where we define $k^* = (k +_w m)$ and $m^* = (m' -_w m)$. If follows that if there exist $k_1 \neq k_2$ satisfying the former equality, then there must also exist $k_1^* \neq k_2^*$ satisfying the latter.

Assuming $m = 0$, we proceed to therefore prove that for any $c, c', m' \in U_w$ with $c \neq c'$ and any $y \in U_{2w}$ there is at most one $k \in U_w$ such that $kc = (k +_w m')c' + y$ in $Z/2^{2w}$. Since $k, m' < 2^w$, we know that $(k +_w m')$ is either $k + m'$ or $k + m' - 2^w$, depending on whether $k + m' < 2^w$ or $k + m' \geq 2^w$ respectively. So now we have

$$k(c - c') = m'c' + y \quad \text{and} \quad k < 2^w - m' \tag{2}$$

$$k(c - c') = (m' - 2^w)c' + y \quad \text{and} \quad k \geq 2^w - m' \tag{3}$$

A simple lemma (Lemma 4.4, presented next) shows that there is at most one solution to each of the equations above. The remainder of the proof is devoted to showing there cannot exist $k = k_1 \in U_w$ satisfying (2) and $k = k_2 \in U_w$ satisfying (3) in $Z/2^{2w}$. Suppose such a k_1 and k_2 did exist. Then we have $k_1 < 2^w - m'$ with $k_1(c - c') = m'c' + y$ and $k_2 \geq 2^w - m'$ with $k_2(c - c') = (m' - 2^w)c' + y$. Subtracting the former from the latter yields

$$(k_2 - k_1)(c' - c) = 2^w c' \tag{4}$$

We show (4) has no solutions in $Z/2^{2w}$. To accomplish this, we examine two cases:

Case 1: $c' > c$. Since both $(k_2 - k_1)$ and $(c' - c)$ are positive and smaller than 2^w, their product is also positive and smaller than 2^{2w}. And since $2^w c'$ is also positive and smaller than 2^{2w}, it is sufficient to show that (4) has no solutions in Z. But this clearly holds, since $(k_2 - k_1) < 2^w$ and $(c' - c) \leq c'$, and so necessarily $(k_2 - k_1)(c' - c) < 2^w c'$.

Case 2: $c' < c$. Here we show $(k_2 - k_1)(c - c') = -2^w c'$ has no solutions in $Z/2^{2w}$. As before, we convert to Z, to yield $(k_2 - k_1)(c - c') = 2^{2w} - 2^w c'$. But again $(k_2 - k_1) < 2^w$ and $(c - c') < (2^w - c')$, so $(k_2 - k_1)(c - c') < 2^w(2^w - c') = 2^{2w} - 2^w c'$. ∎

Let $D_w = \{-2^w + 1, \ldots, 2^w - 1\}$ contain the values attainable from a difference of any two elements of U_w. We now prove the following lemma, used in the body of the preceding proof.

Lemma 4.4 *Let $x \in D_w$ be nonzero. Then for any $y \in U_{2w}$, there exists at most one $a \in U_w$ such that $ax = y$ in $Z/2^{2w}$.*

Proof: Suppose there were two distinct elements $a, a' \in U_w$ such that $ax = y$ and $a'x = y$. Then $ax = a'x$ so $x(a - a') = 0$. Since x is nonzero and a and a' are distinct, the foregoing product is $2^{2w}k$ for nonzero k. But x and $(a - a')$ are in D_w, and therefore their product is in $\{-2^{2w} + 2^{w+1} - 1, \ldots, 2^{2w} - 2^{w+1} + 1\}$, which contains no multiples of 2^{2w} other than 0. ∎

Remarks. The bound given by Theorem 4.2 is tight: let $M = 0^w 0^w$ and $M' = 1^w 0^w$ and note that any key $K = K_1 K_2$ with $K_2 = 0^w$ causes a collision.

Although we do not require any stronger properties than the above, NH is actually 2^{-w}-AΔU under the operation of addition modulo 2^{2w}; only trivial modifications to the above proof are required. See [16] for a definition of ϵ-AΔU.

Several variants of NH fail to preserve collision probability $\epsilon = 2^{-w}$. In particular, replacing the inner addition or the outer addition with bitwise-XOR increases ϵ substantially. However, removing the inner moduli retains $\epsilon = 2^{-w}$ (but significantly degrades performance).

4.4 The Signed Construction: NHS

To this point we have assumed our strings are interpreted as sequences of *unsigned* integers. But often we will prefer they be *signed* integers. Surprisingly, the signed version of NH has slightly higher collision probability: we shall now prove that the collision bound increases by a factor of two to 2^{-w+1}-AU on equal-length strings. This helps explain the 2^{-30} and 2^{-60} forging probabilities for the four UMAC versions named in Section 2.2 and the performance measured in Section 3; in actuality, all four algorithms use the signed version of NH.

Let S_w and S_{2w} be the sets $\{-2^{w-1}, \ldots, 2^{w-1}-1\}$ and $\{-2^{2w-1}, \ldots, 2^{2w-1}-1\}$, respectively. For this section, assume all arithmetic done modulo 2^w returns a result in S_w and all arithmetic done modulo 2^{2w} returns a result in S_{2w}. The family of hash functions NHS is defined exactly like NH except that each $M_i \in \{0,1\}^w$ and $K \in \{0,1\}^w$ is bijectively mapped to an $m_i \in S_w$ and a $k_i \in S_w$.

Theorem 4.5 *For any even $n \geq 2$ and $w \geq 1$, $\mathsf{NHS}[n,w]$ is 2^{-w+1}-AU on equal-length strings.*

The proof of Theorem 4.5 is identical to the proof for Theorem 4.2 with two exceptions: instead of U_w we take elements from S_w, and in the end we use Lemma 4.7, which guarantees a bound of *two* on the number of k_1 values satisfying $c(m_1 +_w k_1) - c'(m'_1 +_w k_1) + y = 0$. To prove Lemma 4.7 we begin by restating Lemma 4.4 for the signed case. Recall that $D_w = \{-2^w+1, \ldots, 2^w-1\}$.

Lemma 4.6 *Let $x \in D_w$ be nonzero. Then for any $y \in S_{2w}$, there exists at most one $a \in S_w$ such that $ax = y$ in $Z/2^{2w}$.*

Proof: Notice that the set of possible differences of any two elements of S_w is again D_w. (This follows from the fact that the obtainable differences for two elements of *any* set of j consecutive integers is always the same, namely $\{-j+1, \ldots, j-1\}$.) Since the proof of Lemma 4.4 depends only on D_w, we may recycle the proof without modification. ∎

Lemma 4.7 *Let c and c' be distinct values from S_w. Then for any $m, m' \in S_w$ and any $y \in S_{2w}$ there exist at most two $k \in S_w$ such that $c(k +_w m) = c'(k +_w m') + y$ in $Z/2^{2w}$.*

Proof: As in the proof to Lemma 4.3, we again notice it is sufficient to prove the case where $m = 0$. We now consider two cases depending on whether $m' < 0$ or $m' \geq 0$. We show in either case there can be at most two values of k satisfying $c(k +_w m) = c'(k +_w m') + y$ in $Z/2^{2w}$.

CASE 1: $m' < 0$. Since $k \in S_w$, we know that $(k +_w m')$ is either $k + m' + 2^w$ (if $k + m' < -2^{w-1}$), or $k + m'$ (if $-2^{w-1} \leq k + m'$). Clearly we cannot have $k + m' \geq 2^{w-1}$. Substituting these values and moving k to the left yields

$$k(c-c') = (m'+2^w)c' + y \quad \text{and} \quad k+m' < -2^{w-1}$$

$$k(c-c') = m'c' + y \quad \text{and} \quad k+m' < 2^{w-1}$$

But Lemma 4.6 tells us there can be at most one solution to each of the above equations.

CASE 2: $m' \geq 0$. Since $k \in S_w$, we now have that $(k +_w m')$ is either $k + m'$ (if $k + m' < 2^{w-1}$) or $k + m' - 2^w$ (if $k + m' \geq 2^{w-1}$). Clearly we cannot have $k + m' < -2^{w-1}$. Substituting these values and moving k to the left yields

$$k(c - c') = m'c' + y \quad \text{and} \quad k + m' < 2^{w-1}$$
$$k(c - c') = (m' - 2^w)c' + y \quad \text{and} \quad k + m' \geq 2^{w-1}$$

But again Lemma 4.6 tells us there can be at most one solution to each of the above equations. ∎

A LOWER BOUND. We now show that the bound for NHS is nearly tight by exhibiting two equal-length messages where the probability of collision is very close to 2^{-w+1}.

Theorem 4.8 *Fix an even $n \geq 2$ and let $m_i = 0$ for $1 \leq i \leq n$. Let $m'_1 = m'_2 = -2^{w-1}$ and $m'_i = 0$ for $3 \leq i \leq n$. Then*

$$\Pr_{K \leftarrow \mathsf{NHS}} [\, \mathsf{NHS}_K(M) = \mathsf{NHS}_K(M') \,] \;\geq\; 2^{-w+1} - 2^{1-2w}$$

where the M and M' are mapped in the usual way from m_i and m'_i, respectively.

Proof: As usual, assume henceforth that all arithmetic is in $Z/2^{2w}$. Invoking the definition of NHS, we will show

$$(k_1 +_w m_1)(k_2 +_w m_2) = (k_1 +_w m'_1)(k_2 +_w m'_2) + y$$

has at least $2^{w+1} - 2$ solutions in $k_1, k_2 \in S_w$. This will imply the theorem. As in the proof to Theorem 4.2, y is the collection of terms for the m_i, m'_i, and k_i where $i > 2$. Since we have $m_i = m'_i = 0$ for $i > 2$, y is clearly 0. Therefore we wish to prove

$$k_1 k_2 = (k_1 -_w 2^{w-1})(k_2 -_w 2^{w-1})$$

has at least $2^{w+1} - 2$ solutions. To remove the inner moduli, we let $i_1 = 1$ if $k_1 < 0$ and $i_1 = 0$ otherwise. Define $i_2 = 1$ if $k_2 < 0$ and $i_2 = 0$ otherwise. Now we may re-write the above as

$$k_1 k_2 = (k_1 - 2^{w-1} + i_1 2^w)(k_2 - 2^{w-1} + i_2 2^w).$$

Multiplying the right side out and rearranging terms we have

$$2^{w-1}(k_1(2i_2 - 1) + k_2(2i_1 - 1)) + 2^{2w-2} = 0.$$

Multiplying through by 4, we arrive at

$$2^{w+1}(k_1(2i_2 - 1) + k_2(2i_1 - 1)) = 0.$$

We notice that all $k_1, k_2 \in S_w$ such that $|k_1| + |k_2| = 2^{w-1}$ will satisfy the above. Now we count the number of k_1, k_2 which work: we see that for each of the $2^w - 2$ choices of $k_1 \in S_w - \{0, -2^{w-1}\}$ there are two values of k_2 causing $|k_1| + |k_2| = 2^{w-1}$, yielding $2^{w+1} - 4$ solutions. And of course two further solutions are $k_1 = -2^{w-1}$, $k_2 = 0$ and $k_1 = 0$, $k_2 = -2^{w-1}$. And so there are at least $2^{w+1} - 2$ total solutions. ∎

5 Reducing the Collision Probability: Toeplitz Extensions

The hash-function families NH and NHS are not yet suitable for use in a MAC: they operate only on strings of "convenient" lengths (ℓw-bit strings for even $\ell \le n$); their collision probability may be higher than desired (2^{-w} when one may want 2^{-2w} or 2^{-4w}); and they guarantee low collision probability only for strings of equal length. We begin the process of removing these deficiencies, here describing a method to square the collision probability at a cost far less than doubling the key length.

5.1 The Toeplitz Approach

To reduce the collision probability for NH, we have a few options. Increasing the wordsize w yields an improvement, but architectural characteristics dictate the natural values for w. Another well-known technique is to apply several random members of our hash-function family to the message, and concatenate the results. For example, if we concatenate the results from, say, four independent instances of the hash function, the collision probability drops from 2^{-w} to 2^{-4w}. But this solution requires four times as much key material. A superior (and well-known) idea is to use the Toeplitz-extension of our hash-function families: given one key we "left shift" to get another key, and we hash again.

For example, to reduce the collision probability to 2^{-64} for $\mathsf{NH}[n, 16]$, we can choose an underlying hash key $K = K_1 \parallel \cdots \parallel K_{n+6}$ and then hash with the four derived keys $K_1 \parallel \cdots \parallel K_n$, $K_3 \parallel \cdots \parallel K_{n+2}$, $K_5 \parallel \cdots \parallel K_{n+4}$, and $K_7 \parallel \cdots \parallel K_{n+6}$. This trick not only saves key material but it can also improve performance by reducing memory accesses, increasing locality of memory references, and increasing parallelism.

Since the derived keys are related it is not clear that the collision probability drops to the desired value of 2^{-64}. Although there are established results which yield this bound (e.g., [17]), they only apply to linear hashing schemes over fields. Instead, NH is non-linear and operates over a combination of rings ($Z/2^w$ and $Z/2^{2w}$). In Theorem 5.1 we prove that the Toeplitz construction nonetheless achieves the desired bound in the case of NH.

5.2 The Unsigned Case

We define the hash-function family $\mathsf{NH}^\mathsf{T}[n, w, t]$ ("Toeplitz-NH") as follows. Fix an even $n \ge 2$, $w \ge 1$, and $t \ge 1$ (the "Toeplitz iteration count"). The domain $A = \{0,1\}^{2w} \cup \{0,1\}^{4w} \cup \cdots \cup \{0,1\}^{nw}$ remains as it was for NH, but the range is now $B = \{0,1\}^{2wt}$. A function in $\mathsf{NH}[n, w, t]$ is named by a string K of $w(n + 2(t-1))$ bits. Let $K = K_1 \parallel \cdots \parallel K_{n+2(t-1)}$ (where each K_i is a w-bit word), and let the notation $K_{i..j}$ represent $K_i \parallel \cdots \parallel K_j$. Then for any $M \in A$ we define $\mathsf{NH}^\mathsf{T}_K(M)$ as

$$\mathsf{NH}^\mathsf{T}_K(M) = \mathsf{NH}_{K_{1..n}}(M) \parallel \mathsf{NH}_{K_{3..n+2}}(M) \parallel \cdots \parallel \mathsf{NH}_{K_{(2t-1)..(n+2t-2)}}(M).$$

When clear from context we write NH^T instead of $\mathsf{NH}^\mathsf{T}[n, w, t]$.

The following shows that NH^T enjoys the best bound that one could hope for.

Theorem 5.1 *For any $w, t \ge 1$ and any even $n \ge 2$, $\mathsf{NH}^\mathsf{T}[n, w, t]$ is 2^{-wt}-AU on equal-length strings.*

Proof: We refer to $\mathsf{NH}^\mathsf{T}[n, w, t]$ as NH^T and to $\mathsf{NH}[n, w]$ as NH, where the parameters n, w, and t are as in the theorem statement.

Let M and M' be distinct members of the domain A with $|M| = |M'|$. We are required to show

$$\Pr_{K \leftarrow \mathsf{NH}^\mathsf{T}} \left[\mathsf{NH}^\mathsf{T}_K(M) = \mathsf{NH}^\mathsf{T}_K(M') \right] \le 2^{-wt}.$$

As usual, we view M, M', K as sequences of w-bit words, $M = M_1 \parallel \cdots \parallel M_\ell$, $M' = M'_1 \parallel \cdots \parallel M'_\ell$ and $K = K_1 \parallel \cdots \parallel K_{n+2(t-1)}$, where the M_i, M'_i and K_i are each w-bits long. We denote by m_i, m'_i, and k_i the w-bit integers corresponding to M_i, M'_i, and K_i, respectively. Again, for this proof assume all arithmetic is carried out in $Z/2^{2w}$. For $j \in \{1, \ldots, t\}$, define E_j as

$$E_j: \ \sum_{i=1}^{\ell/2} (k_{2i+2j-3} +_w m_{2i-1})(k_{2i+2j-2} +_w m_{2i}) = \sum_{i=1}^{\ell/2} (k_{2i+2j-3} +_w m'_{2i-1})(k_{2i+2j-2} +_w m'_{2i})$$

and invoke the definition of NH^T to rewrite the above probability as

$$\Pr_{K \leftarrow \mathsf{NH}^\mathsf{T}} [E_1 \wedge E_2 \wedge \cdots \wedge E_t]. \tag{5}$$

We call each term in the summations of the E_j a "clause" (for example, $(k_1 +_w m_1)(k_2 +_w m_2)$ is a clause). We refer to the jth equality in (5) as Equality E_j.

Without loss of generality, we can assume that M and M' disagree in the last clause (i.e., that $m_{\ell-1} \ne m'_{\ell-1}$ or $m_\ell \ne m'_\ell$). To see this, observe that if M and M' agree in the last few clauses, then each E_j is satisfied by a key K if and only if it is also satisfied when omitting these last few clauses. Hence, we could truncate M and M' after the last clause in which they disagree, and still have exactly the same set of keys causing collisions.

Assume now that $m_{\ell-1} \ne m'_{\ell-1}$. (The proof may easily be restated if we instead have $m_\ell \ne m'_\ell$. This case is symmetric due to the fact we shift the key by two words.) We proceed by proving that for all $j \in \{1, \ldots, t\}$, $\Pr[E_j \text{ is true} \mid E_1, \ldots, E_{j-1} \text{ are true}] \le 2^{-w}$, implying the theorem.

For E_1, the claim is satisfied due to Theorem 4.2. For $j > 1$, notice that Equalities E_1 through E_{j-1} depend only on key words $k_1, \ldots, k_{\ell+2j-4}$, whereas Equality E_j depends also on key words $k_{\ell+2j-3}$ and $k_{\ell+2j-2}$. Fix k_1 through $k_{\ell+2j-4}$ such that Equalities E_1 through E_{j-1} are satisfied, and fix any value for $k_{\ell+2j-3}$. We prove that there is at most one value of $k_{\ell+2j-2}$ satisfying E_j. To achieve this we follow the same technique used in Theorem 4.2. Let

$$y = \sum_{i=1}^{\ell/2-1} (k_{2i+2j-3} +_w m_{2i-1})(k_{2i+2j-2} +_w m_{2i}) - \sum_{i=1}^{\ell/2-1} (k_{2i+2j-3} +_w m'_{2i-1})(k_{2i+2j-2} +_w m'_{2i})$$

and let $c = (k_{\ell+2j-3} +_w m_{\ell-1})$ and $c' = (k_{\ell+2j-3} +_w m'_{\ell-1})$, and then rewrite E_j as

$$c(k_{\ell+2j-2} +_w m_\ell) + y = c'(k_{\ell+2j-2} +_w m'_\ell).$$

Since we assumed $m_{\ell-1} \ne m'_{\ell-1}$ we know $c \ne c'$; clearly $m_\ell, m'_\ell, c, c' \in U_w$ so Lemma 4.3 tells us there is at most one value of $k_{\ell+2j-2}$ satisfying this equation, which completes the proof. ▮

COMMENT. With UMAC, we sometimes use shift amounts of more than two words; this clearly does not interfere with the validity of our proof, provided the number of shifted words is even.

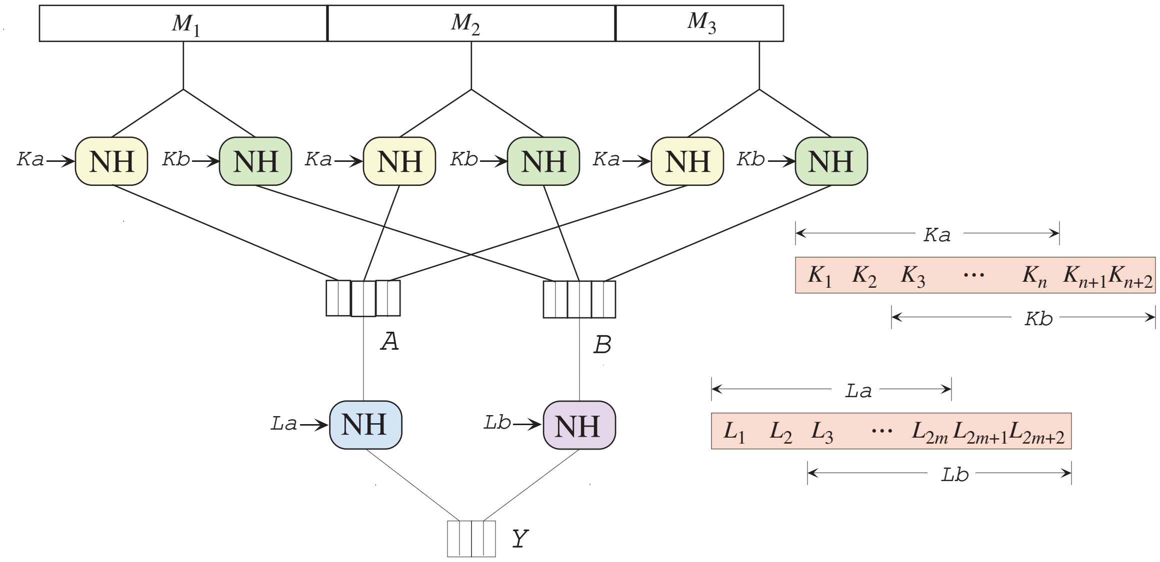


Figure 4: *The T2 scheme. Each block of the message is hashed twice, first using a key Ka, and then using a key Kb. Key Kb is the shift of key Ka. The outputs of the Ka-hashes are concatenated and La hashed, and the outputs of the Kb-hashes are concatenated and then Lb hashed. Key Lb is once again the shift of key La.*

5.3 The Signed Case

Now we consider the Toeplitz construction on NHS, the signed version of NH. The only significant change in the analysis for NH^{T} will be the effect of the higher collision probability of NHS.

We define the hash family $\mathsf{NHS}^{\mathsf{T}}[n, w, t]$ ("Toeplitz-NHS") exactly as we did for NH, but we instead use NHS as the underlying hash function. Now we restate Theorem 5.1 for the signed case.

Theorem 5.2 *For any $w, t \geq 1$ and any even $n \geq 2$, $\mathsf{NHS}^{\mathsf{T}}[n, w, t]$ is $2^{t(-w+1)}$-AU on equal-length strings.*

Proof: The proof is precisely the same as the proof to Theorem 5.1 except we use Theorem 4.5 in place of Theorem 4.2 and Lemma 4.7 in place of Lemma 4.3. Then, using the same notation as Theorem 5.1, $\Pr_{K \leftarrow \mathsf{NHS}^{\mathsf{T}}}[E_j \mid E_1, \ldots, E_{j-1}] \leq 2^{-w+1}$, and so $\Pr_{K \leftarrow \mathsf{NHS}^{\mathsf{T}}}[E_1 \wedge E_2 \wedge \cdots \wedge E_t] \leq 2^{t(-w+1)}$, yielding our result. ∎

5.4 Shorter Keys: T2

Fix an even $n \geq 2$ and $m, w \geq 1$. We describe the family of hash functions $\mathsf{NH}^{\mathsf{T2}}[n, m, w]$ and the family of hash functions $\mathsf{NHS}^{\mathsf{T2}}[n, m, w]$. Both families have domain $A = \{0,1\}^{2w} \cup \{0,1\}^{4w} \cup \cdots \cup \{0,1\}^{nmw}$ and range $B = \{0,1\}^{4w}$. (That is, the input consists of an even number of w-bit words—at least two words and at most mw words. The output is four words.) A function $\mathsf{NH}^{\mathsf{T2}}(K \parallel L, \cdot)$ from the family $\mathsf{NH}^{\mathsf{T2}}[n, m, w]$ is named by a key $K \parallel L$ having $(n+2)w + (2m+2)w$ bits. (The part of the key denoted K has $n+2$ words, and the part of the key denoted L has $2m+2$ words.) We define $\mathsf{NH}^{\mathsf{T2}}(K \parallel L,\ M)$ below. See Figure 4 as well.

`function` $\mathsf{NH}^{\mathsf{T2}}(K \parallel L,\ M)$

1. Let $Ka = K[1..nw]$
2. Let $Kb = K[2w+1..(n+2)w]$
3. Let $La = L[1..2mw]$
4. Let $Lb = L[2w+..(2m+2)w]$
5. Let $M_1 \parallel \cdots \parallel M_t = M$, where $t = \lceil |M|/(nw) \rceil$,
   $|M_1| = \cdots = |M_{t-1}| = nw$, and $2w \le |M_t| \le nw$
6. $A = \mathsf{NH}_{Ka}(M) \parallel \cdots \parallel \mathsf{NH}_{Ka}(M_t)$
7. $B = \mathsf{NH}_{Kb}(M) \parallel \cdots \parallel \mathsf{NH}_{Kb}(M_t)$
8. Return $\mathsf{NH}_{La}(A) \parallel \mathsf{NH}_{Lb}(B)$

The family of hash functions $\mathsf{NHS}^{\mathsf{T2}}[n,m,w]$ is defined exactly as above except for using NHS instead of NH throughout the construction. We can show the following two theorems.

Theorem 5.3 *For any $m, w \ge 1$ and any even $n \ge 2$, $\mathsf{NH}^{\mathsf{T2}}[n,m,w]$ is 2^{-2w+2}-AU on equal-length strings.*

Theorem 5.4 *For any $m, w \ge 1$ and any even $n \ge 2$, $\mathsf{NHS}^{\mathsf{T2}}[n,m,w]$ is 2^{-2w+4}-AU on equal-length strings.*

6 Arbitrary-Length Messages: Padding, Concatenation, Length Annotation

With NH^{T} we can decrease the collision probability to any desired level but we still face the problem that this function operates only on strings of "convenient" length, and that it guarantees this low collision probability only for equal-length strings. We solve these problems in a generic manner, with a combination of padding, concatenation, and length annotation.

MECHANISM. Let $\mathsf{H} : \{A \to B\}$ be a family of hash functions where functions in H are defined only for particular input lengths, up to some maximum, and all the hash functions have a fixed output length. Formally, the domain is $A = \bigcup_{i \in I} \{0,1\}^i$ for some finite nonempty index set $I \subseteq \mathsf{N}$ and the range is $B = \{0,1\}^\beta$, where β is some positive integer. Let a (the "blocksize") be the length of the longest string in A and let $\alpha \ge \lceil \lg_2 a \rceil$ be large enough to describe $|M| \bmod a$. Then we define $\mathsf{H}^* = \{h^* : \{0,1\}^* \to \{0,1\}^*\}$ as follows. Each function $h \in \mathsf{H}$ gives rise to a corresponding function $h^* \in \mathsf{H}^*$, A random function of H^* is determined by taking a random function $h \in \mathsf{H}$ and following the procedure below.

First partition the message Msg into some number of "full blocks" (each containing exactly a bits) and then a "last block" (which might contain fewer bits). Hash each of the full blocks by applying h, and then hash the last block by first zero-padding it to the next domain point and then applying h. Finally, concatenate all the hash values (each having β bits), and append (an encoding of) the length of the last block prior to padding. [2] Pseudocode follows.

`function` $h^*(Msg)$

1. If $M = \lambda$ then return 0^α
2. View Msg as a sequence of "blocks", $Msg = Msg_1 \parallel \cdots \parallel Msg_t$,
   with $|Msg_j| = a$ for all $1 \le j < t$, and $1 \le |Msg_t| \le a$

[2] Since the length of the last block (prior to padding) is between 1 and a, it suffices to take this number modulo a; it still specifies the length of the last block. The length annotation can equivalently be regarded as the length of the original message Msg modulo a.

3. Let Len be an α-bit string that encodes $|Msg| \bmod a$
4. Let $i \geq 0$ be the least number such that $Msg_t \parallel 0^i \in A$
5. $Msg_t = Msg_t \parallel 0^i$
6. Return $h(Msg_1) \parallel \cdots \parallel h(Msg_t) \parallel Len$

ANALYSIS. The following proposition indicates that we have correctly extended H to H^*.

Proposition 6.1 *Let $I \subseteq \mathsf{N}$ be a nonempty finite set, let $\beta \geq 1$ be a number, and let $\mathsf{H} = \{h : \bigcup_{i \in I} \{0,1\}^i \to \{0,1\}^\beta\}$ be a family of hash functions. Let $\mathsf{H}^* = \{h^* : \{0,1\}^* \to \{0,1\}^*\}$ be the family of hash functions obtained from H as described above. Suppose H is ϵ-AU on strings of equal length. Then H^* is ϵ-AU (across all strings).*

Proof: The idea is to note that when distinct messages Msg and Msg' have different lengths then, by our definition of H^*, their hashes can never collide; and if, on the other hand, distinct messages Msg and Msg' have the same length, then, because H is ϵ-AU on equal-length strings, the hash under $h^* \leftarrow \mathsf{H}^*$ of Msg and Msg' agree with probability at most ϵ even if one fixes attention on some particular spot where Msg and Msg' differ. Details follows.

We must show that for all distinct $Msg, Msg' \in \{0,1\}^*$, $\Pr_{h^* \in \mathsf{H}^*}[h^*(Msg) = h^*(Msg')] \leq \epsilon$. So fix distinct $Msg, Msg' \in \{0,1\}^*$ and consider the following cases, based on the relative lengths of these strings:

Case 1. Let $t = \lceil |Msg|/a \rceil$ and let $t' = \lceil |Msg'|/a \rceil$. If the length of Msg and Msg' are sufficiently different that $t \neq t'$ then the probability that $h^*(Msg) = h^*(Msg')$ is zero, since $h^*(Msg)$ and $h^*(Msg')$ have different lengths.

Case 2. With t and t' as above, assume $|Msg|$ differs from $|Msg'|$ but in a manner such that $t = t'$. This means that the lengths of the last blocks of Msg and Msg' differ, and therefore the last α bits of $h^*(Msg)$ (representing the length of the last block) differ from the last α bits of $h^*(Msg')$, so the probability of collision is zero.

Case 3. Finally, assume $|Msg| = |Msg'|$. Suppose Msg is partitioned into $Msg_1 \parallel \cdots \parallel Msg_t$ and Msg' is partitioned into $Msg'_1 \parallel \cdots \parallel Msg'_t$. Since $Msg \neq Msg'$ there exists some index i such that $Msg_i \neq Msg'_i$. Fix such an i. Of course $|Msg_i| = |Msg'_i|$. If $i = t$ then we replace Msg_i and Msg'_i by what we get after zero-padding them, but the lengths of the (now padded) blocks Msg_i and Msg'_i will still be identical and these blocks will still differ. Hence we have that $\Pr_{h \in \mathsf{H}}[h(Msg_i) = h(Msg'_i)] \leq \epsilon$. Moreover, $h(Msg_i) \neq h(Msg'_i)$ implies that $h^*(Msg) \neq h^*(Msg')$, since one can infer $h(Msg_i)$ from $h^*(Msg)$ and one can infer $h(Msg'_i)$ from $h^*(Msg')$. (This uses the fact the range of h is fixed-length strings.) It follows that $h^*(Msg)$ and $h^*(Msg')$ coincide with probability at most ϵ. ▌

COMMENT. For implementation convenience the spec for UMAC [6] is byte-oriented, and so the domain of the hash function defined in the spec is actually $(\{0,1\}^8)^*$, not $\{0,1\}^*$.

7 Final Extensions: Stride, Endianness, Key Shifts

There are a few more useful adjustments to enhance the performance, generality, or and ease-of-implementation of the hash families we have constructed. Although we have defined NHX with parameters n, w, t, what is specified [6] actually has additional parameters: "message stride," "endianness," "key shift," "signed/unsigned," and parameters to handle two-level (2L) hashing. We have already analyzed the effect of using signed integers and two-level hashing in Section 4.4 and Section 5.4. Let us now describe the parameters we have not touched on and argue that they do not affect the analysis.

First we make the simple observation that any bijection π applied to the inputs of an ϵ-AU hash-function family leaves the collision bound unchanged. This can be observed by simply looking at the definition of ϵ-AU: the statement requires a certain probability hold for any two distinct inputs M, M' from the domain. But if $M \neq M'$ then certainly $\pi(M) \neq \pi(M')$ and so the bound still holds.

Message Stride. In NH we multiplied $(k_{2i-1} +_w m_{2i-1})$ by $(k_{2i} +_w m_{2i})$ for $1 \leq i \leq \ell/2$. As described in Section 2.2, we may prefer to pair these multiplications differently, like

$$(k_1 +_w m_1)(k_5 +_w m_5) + (k_2 +_w m_2)(k_6 +_w m_6) + \cdots$$

to take advantage of machine architectural characteristics. Such rearrangements do not affect our collision bound since they amount to applying a fixed permutation to the input and key; since the keys are random, permuting them is irrelevant.

Endianness. Whether we read a message using big-endian or little-endian conventions again can be viewed as a bijection on the messages.

Key Shift. In our proof of Theorem 5.1 we shifted our key by two words to acquire the "next" key. In some cases we may wish to shift by more than two words for performance reasons (specifically in the MMX implementations). As noted in the comment following that proof, a larger shift does not invalidate the argument.

8 From Hash to MAC

In this section we describe a way to make a secure MAC from an ϵ-AU family of hash functions (with small ϵ) and a secure pseudorandom function (PRF). We also describe some constructions for suitable PRFs. We begin with the formal definitions for MACs and PRFs.

8.1 Security Definitions

MACs And Their Security. For this paper a MAC scheme $\Sigma = (\textsc{Key}, \textsc{Tag})$ consists of two things: a *key generator* Key and a *tag generator* Tag. There are also four associated nonempty sets: Key, which denotes the set of possible keys, and $\mathsf{Message}$, Nonce, and Tag, which are sets of strings. The sets Nonce and Tag are finite, with $\mathsf{Nonce} = \{0,1\}^\eta$ and $\mathsf{Tag} = \{0,1\}^\tau$. These different sets are used to describe the domain and range of Key and Tag, as we now explain.

The key generator Key takes no arguments and probabilistically produces an element $Key \in \mathsf{Key}$. This process is denoted $Key \leftarrow \textsc{Key}()$. The tag generator Tag takes $Key \in \mathsf{Key}$, $M \in \mathsf{Message}$, and $Nonce \in \mathsf{Nonce}$, and deterministically yields $Tag \in \mathsf{Tag}$. The first argument to Tag will be written as a subscript, as in $Tag = \textsc{Tag}_{Key}(M, Nonce)$.

An adversary F for attacking MAC scheme $\Sigma = (\textsc{Key}, \textsc{Tag})$ is an algorithm with access to two oracles, denoted Tag and Vf. Specifically, F's oracles behave as follows. First the key generator is run to compute $Key \leftarrow \mathsf{Key}()$. From then on, when presented with a query $(M, Nonce) \in \mathsf{Message} \times \mathsf{Nonce}$, the Tag oracle returns $\textsc{Tag}_{Key}(M, Nonce)$. (If the query $(M, Nonce)$ is outside of the indicated domain, the oracle returns the empty string.) When presented with a query $(M, Nonce, Tag) \in \mathsf{Message} \times \mathsf{Nonce} \times \mathsf{Tag}$, the Vf oracle returns 1 if $\textsc{Tag}_{Key}(M, Nonce) = Tag$, and it returns 0 otherwise. (If the query $(M, \mathit{Nonce}, \mathit{Tag})$ is outside of the indicated domain, the oracle returns 0.) To disallow replay attacks and meet other goals one can, alternatively, make the VF oracle stateful and more restrictive with when it returns 1. We will not pursue these possibilities here.

In an execution of an adversary with her oracles, she is said to *forge*, or *succeed*, if she asks the VF oracle some query $(M, \mathit{Nonce}, \mathit{Tag})$ which returns 1 even though (M, Nonce) was not an earlier query to the TAG oracle. The success of adversary F in attacking Σ, denoted $\mathsf{Succ}^{\mathrm{mac}}_{\Sigma}(F)$, is the probability that F succeeds. Informally, a MAC scheme Σ is good if for every reasonable adversary F the value $\mathsf{Succ}^{\mathrm{mac}}_{\Sigma}(F)$ is suitably small. To talk of this conveniently, we overload the notation and let $\mathsf{Succ}^{\mathrm{mac}}_{\Sigma}(t, q_s, q_v, \mu)$ be the maximal value of $\mathsf{Succ}^{\mathrm{mac}}_{\Sigma}(F)$ among adversaries that run in time at most t, ask at most q_s TAG queries, ask at most q_v VF queries, and all of these queries total at most μ bits. One assumes some fixed RAM model of computation and running time is understood to mean the actual running time plus the size of the program description.

REMARKS. Our notion for MAC security is very strong, insofar as we have let the adversary manipulate the nonce (just so long as they *are* nonces), and then we regarded the adversary as successful if she could forge any new message (or even an old message, but with a nonce different from any already used for it). In actuality we expect the nonce to be a counter which is not under the adversary's control. Our notion of security says that even if the adversary *could* control the nonce, *still* she would be unable to forge, even in a very weak sense.

The definition above explicitly allows verification queries. This definitional choice is pretty inconsequential, in the following sense: $\mathsf{Succ}^{\mathrm{mac}}_{\Sigma}(t, q_s, q_v, \mu) \le q_v \cdot \mathsf{Succ}^{\mathrm{mac}}_{\Sigma}(t + O(\mu), q_s, 1, \mu)$ for any MAC scheme Π. That is, suppose we allowed only one verification query. This is equivalent to having the adversary output her one and only attempt at a forgery, $(M, \mathit{Nonce}, \mathit{Tag})$, at the end of her execution. Then generalizing (as our definition does) to $q_v \ge 1$ verification queries will increase the adversary's chance of success by at most q_v. The proof is simple and the observation is well known, so a proof is omitted. But we will use this fact in the proof of Lemma 8.1.

For substantial q_v the value $\mathsf{Succ}^{\mathrm{mac}}_{\Sigma}(t, q_s, q_v, \mu)$ may become larger than desired. For this reason it may be necessary to architecturally limit q_v to 1—for example, by tearing down a connection when a forgery attempt is detected. There are further approaches to keeping $\mathsf{Succ}^{\mathrm{mac}}_{\Sigma}(t, q_s, q_v, \mu)$ small when one is imagining q_v large.

PRFS AND THEIR SECURITY. A pseudorandom function (PRF) (with key-length α, arbitrary argument length, and output-length β) is a family of functions $\mathsf{F} : \{0,1\}^{\alpha} \times \{0,1\}^{*} \to \{0,1\}^{\beta}$. (See Section 4.1). Let $\mathsf{Rand}(\beta)$ be the family of functions from $\{0,1\}^{*}$ to $\{0,1\}^{\beta}$ in which choosing a random $\rho \leftarrow \mathsf{Rand}(\beta)$ means associating to each string $x \in \{0,1\}^{*}$ a random string $\rho(x) \in \{0,1\}^{\beta}$.

An adversary D for attacking the PRF $\mathsf{F} : \{0,1\}^{\alpha} \times \{0,1\}^{*} \to \{0,1\}^{\beta}$ is given an oracle g which is either a random element of F or a random element of $\mathsf{Rand}(\beta)$. The adversary tries to distinguish these two possibilities. Her advantage is defined as $\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(D) = \Pr_{a \in \{0,1\}^{\alpha}}[D^{\mathsf{F}_a(\cdot)} = 1] - \Pr_{\rho \in \mathsf{Rand}(\beta)}[D^{\rho(\cdot)} = 1]$. Informally, a PRF F is good if for every reasonable adversary D the value $\mathsf{Adv}_{\mathsf{F}}(D)$ is small. To talk of this conveniently we overload the notation and let $\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(t, q, \mu)$ be the maximal value of $\mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(D)$ among adversaries that run in time at most t, ask at most q oracle queries, and these queries total at most μ bits.

8.2 Definition of the PRF(HASH, Nonce) Construction

We use a family of (hash) functions $\mathsf{H} = \{h : \{0,1\}^{*} \to \{0,1\}^{*}\}$ and a family of (random or pseudorandom) functions $\mathsf{F} = \{f : \{0,1\}^{*} \to \{0,1\}^{\tau}\}$. These are parameters of the construction.

We also fix a set $\mathsf{Nonce} = \{0,1\}^\eta$ and an "encoding scheme" $\langle \cdot, \cdot \rangle$. The encoding scheme is a linear-time computable function that maps a string $HM \in \{0,1\}^*$ and $Nonce \in \mathsf{Nonce}$ into a string $\langle HM, Nonce \rangle$ of length $|HM| + |Nonce| + O(1)$ from which, again in linear time, one can recover HM and $Nonce$. The MAC scheme $\mathsf{UMAC}[\mathsf{H},\mathsf{F}] = (\text{KEY}, \text{TAG})$ is then defined as follows:

`function` KEY () $f \leftarrow \mathsf{F}$ $h \leftarrow \mathsf{H}$ `return` (f, h)	`function` $\text{TAG}_{(f,h)}$ $(M,\ Nonce)$ `return` $f\ (\ \langle h(M),\ Nonce \rangle\ )$

The keyspace for this MAC is $\mathsf{Key} = \mathsf{H} \times \mathsf{F}$; that is, a random key for the MAC is a random hash function $h \in \mathsf{H}$ together with a random function $f \in \mathsf{F}$.

ANALYSIS. We begin with the information-theoretic version of the scheme.

Lemma 8.1 *Let $\epsilon \geq 0$ be a real number and let $\mathsf{H} = \{h : \{0,1\}^* \to \{0,1\}^*\}$ be an ϵ-AU family of hash functions. Let $\tau \geq 1$ be a number and let $\Sigma = \mathsf{UMAC}[\mathsf{H}, \mathsf{Rand}(\tau)]$ be the MAC scheme described above. Then for every adversary F that makes at most q_v verification queries we have that $\mathsf{Succ}_\Sigma^{\mathrm{mac}}(F) \leq q_v(\epsilon + 2^{-\tau})$.*

Proof: First assume that F prepares a single verification query which, without loss of generality, is its last oracle query. Run adversary F in the experiment which defines success in attacking $\Sigma = (\text{KEY}, \text{TAG})$. In this experiment the adversary is provided an oracle which behaves as follows. First it chooses a random $\rho \leftarrow \mathsf{Rand}(\tau)$ and $h \leftarrow \mathsf{H}$. Now F asks the sequence of tag-generation queries $(M_1, Nonce_1), \ldots, (M_q, Nonce_q)$, getting responses $Tag_1, \ldots, Tag_q$, where $Tag_i = \text{TAG}_{(\rho,h)}(\langle h(M_i), Nonce_i \rangle)$. Based on these answers (and possibly internal coin flips) the adversary produces its verification query $(M, Nonce, Tag)$. Since we are bounding from above the probability that F is successful we may assume of F's behavior that $Nonce_1, \ldots, Nonce_q$ are distinct and that $(M, Nonce) \notin \{(M_1, Nonce_1), \ldots, (M_q, Nonce_q)\}$ for otherwise, by definition, F does not succeed. Let Forge be the event that the adversary is successful: $Tag = \rho(\langle h(M), Nonce \rangle)$. Let $y_j = \langle M_j, Nonce_j \rangle$ for $1 \leq j \leq q$ and let $y = \langle M, Nonce \rangle$. Note that $y_1, \ldots, y_q$ are distinct, since $Nonce_1, \ldots Nonce_q$ are distinct and $\langle \cdot, \cdot \rangle$ is a (reversible) encoding.

Let Repeat be the event that $Nonce \in \{Nonce_1, \ldots, Nonce_q\}$. Observe that $\Pr[\mathsf{Forge} | \overline{\mathsf{Repeat}}] \leq 2^{-\tau}$ since when $\overline{\mathsf{Repeat}}$ holds the adversary must predict $\sigma(y)$ given $\sigma(y_1), \ldots, \sigma(y_q)$, where $y_1, \ldots, y_q$ are all distinct from y.

We wish to bound $\Pr[\mathsf{Forge} | \mathsf{Repeat}]$. Let i be the index such that $Nonce = Nonce_i$. There is a unique such i since $Nonce_1, \ldots, Nonce_q$ are distinct and $Nonce$ is among them. Let $\mathsf{Collision}$ be the event that Repeat holds and $h(M) = h(M_i)$. We note that $\Pr[\mathsf{Forge} | \overline{\mathsf{Collision}}] \leq 2^{-\tau}$ since, as before, the adversary must predict $\sigma(y)$ having seen only $\sigma(y_1), \ldots, \sigma(y_q)$, where $y \notin \{y_1, \ldots, y_q\}$.

We claim $\Pr[\mathsf{Collision}] \leq \epsilon$. By definition of H being ϵ-AU we know if one chooses a random $h \leftarrow \mathsf{H}$ and gives the adversary no information correlated to h then the adversary's chance of producing distinct M and M_i which collide under h is at most ϵ. The point to note is that the adversary has, in fact, obtained no information correlated to h. In response to her queries she obtains the images under ρ of the distinct points $y_1, \ldots, y_q$. These values, $Tag_1, \ldots, Tag_q$, are random τ-bit strings; the adversary F would be provided an identical distribution of views if the oracle were replaced by one which, in response to a query, returned a random τ-bit string.

The result now follows. We have that $\Pr[\mathsf{Forge}] \le \max\{\Pr[\mathsf{Forge}|\mathsf{Repeat}],\ \Pr[\mathsf{Forge}|\overline{\mathsf{Repeat}}]\}$, with the second expression being at most $2^{-\tau}$. On the other hand,

$$\begin{aligned}\Pr[\mathsf{Forge}|\mathsf{Repeat}] &= \Pr[\mathsf{Forge}|\mathsf{Collision}]\Pr[\mathsf{Collision}] + \Pr[\mathsf{Forge}|\overline{\mathsf{Collision}}]\Pr[\overline{\mathsf{Collision}}] \\ &\le \Pr[\mathsf{Collision}] + \Pr[\mathsf{Forge}|\overline{\mathsf{Collision}}] \\ &\le \epsilon + 2^{-\tau},\end{aligned}$$

as we have argued. We conclude that $\Pr[\mathsf{Forge}] \le \epsilon + 2^{-\tau}$. To finish the proof, use the observation, mentioned in the Remarks of Section 8.1, that allowing q_v verification queries at most increases the adversary's chance of success by a multiplicative factor of q_v. ▮

In the usual way we can extend the above information-theoretic result to the complexity-theoretic setting of interest to applications. Roughly, we prove that if the hash-function family is ϵ-AU and no reasonable adversary can distinguish the PRF from a truly random function with advantage exceeding δ then no reasonable adversary can break the resulting MAC scheme with probability exceeding $\epsilon + \delta$.

To make this formal, we use the following notations. If H is a family of hash functions then $\mathsf{Time}_{\mathsf{H}}$ is an amount of time adequate to compute a representation for a random $h \leftarrow \mathsf{H}$, while $\mathsf{Time}_h(\mu)$ is an amount of time adequate to evaluate h on strings whose lengths total μ bits.

The proof of the following, being standard, is omitted.

Theorem 8.2 *Let $\epsilon \ge 0$ be a real number, let $\mathsf{H} = \{h : \{0,1\}^* \to \{0,1\}^*\}$ be an ϵ-AU family of hash functions, let $\tau \ge 1$ be a number and let $F : \{0,1\}^\alpha \times \{0,1\}^* \to \{0,1\}^\tau$ be a PRF. Let $\Sigma = \mathsf{UMAC}[\mathsf{H},\mathsf{F}]$ be the MAC scheme described above. Then*

$$\mathsf{Succ}^{\mathrm{mac}}_{\Sigma}(t, q_s, q_v, \mu) \;\le\; \mathsf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(t', q', \mu') + q_v(\epsilon + 2^{-\tau})$$

where $t' = t + \mathsf{Time}_{\mathsf{H}} + \mathsf{Time}_h(\mu) + O(\mu)$ and $q' = q_s + q_v$ and $\mu' = O(\mu)$. ▮

8.3 Discussion

We have now defined the PRF(HASH, Nonce) construction, but we wish to make a couple of comments about it.

COMPARISON WITH CARTER-WEGMAN METHOD. Compared to the original suggestion of [26], where one encrypts the hash of the message by XOR-ing with a one-time pad, we require a weaker assumption about the hash-function family H: it need only be ϵ-AU. The original approach of [26] needs of H the stronger property of being "XOR almost-universal" [16]: for all distinct M, M' and for all C, one must have that $\Pr_h[h(M) \oplus h(M') = C] \le \epsilon$. Furthermore, it is no problem for us that the range of $h \in \mathsf{H}$ has strings of unbounded length, while the hash functions used for [26] should have fixed-length output. On the other hand, our cryptographic tool is effectively stronger than what the complexity-theoretic version of [26] requires: we need a PRF over $\{0,1\}^*$ (or at least over the domain $\mathsf{Encoding}$ of possible $\langle HM, Nonce\rangle$ encodings), while the complexity-theoretic analog of [26] could conveniently use a (fixed-output-length) PRF on the fixed-length strings Nonce. The security bound one gets is the same; both constructions are as good as one could hope for, from the point of view of concrete security.

COMPARISON WITH PRF(HASH) CONSTRUCTION. Let us now clarify the role of the nonce in the PRF(HASH, Nonce) scheme. The nonce is essential, in the sense that in its absence the quantitative security bounds are much worse and unacceptable for many applications. Let $\overline{\text{UMAC}}[\mathsf{H},\mathsf{F}]$ be the scheme which is the same as UMAC, except that the PRF is applied directly to HM, rather than to $\langle HM, Nonce\rangle$. Then the analog of Lemma 8.1 would say the following: Fix a number $\tau \geq 1$, let $\mathsf{H} = \{h : \{0,1\}^* \to \{0,1\}^*\}$ be an ϵ-AU family of hash functions, and let $\Sigma = \overline{\mathsf{UMAC}}[\mathsf{H}, \mathsf{Rand}(\tau)]$ be the MAC scheme described above. Then for every adversary F, $\mathsf{Succ}_{\Sigma}^{\mathrm{mac}}(F) \leq q^2\epsilon + 2^{-\tau}$.

This is a far cry from our earlier bound of $\epsilon + 2^{-\tau}$. Instead of security which is independent of the number of tags acquired, q, security degrades with the square of that number. When $q = \epsilon^{-1/2}$, there is no security left.

It is important to note that this is not a problem with the analysis, but with the scheme itself. If one asks $\epsilon^{-1/2}$ oracle queries of $\overline{\mathsf{UMAC}}$ then, by the birthday bound, there is a good chance to find distinct messages, M_1 and M_2, which yield the same authentication tag. If τ is large then this implies that, with high probability, the hash of M_1 equals the hash of M_2. This is crucial information about the hash function which has now been leaked. In particular, for some ϵ-AU hash-function families knowing that M_1 and M_2 collide immediately tells us some third message M_3 which collides with both of them. The adversary can now forge an authentication tag for this message.

The conclusion is that the nonce in the PRF(HASH, Nonce)-scheme cannot be removed. For example, without it, using a 2^{-32}-AU hash function would let you authenticate fewer than 2^{16} messages, which is usually insufficient.

8.4 Realizing the PRF

A complete specification of UMAC must describe how to make the requisite PRF. Under our construction the domain of the PRF needs to be strings of arbitrary lengths. The specification associated to this paper [6] suggests two ways of realizing such PRFs: one using a cryptographic hash function and using a secure block cipher. Let us briefly describe the two suggested constructions. The second is somewhat novel.

From A Cryptographic Hash Function. Given a cryptographic hash function H (e.g., SHA-1) we can use HMAC based on H as our PRF [3, 12]. This method defines the PRF F to be $F_A(M) = H(A \oplus \text{opad} \parallel H(A \oplus \text{ipad} \parallel M))$, where $|A|$ is, typically, the blocksize of H (i.e., 512 bits for all well-known constructions) and opad and ipad are distinct $|A|$-bit constants. This construction is shown in [3] to be a PRF under weak (though nonstandard) assumptions about H.

From A Block Cipher. We use a new variant of the CBC-MAC in order to turn a block cipher E into a PRF F which operates over arbitrary-length inputs. Our CBC-MAC variant is more efficient than earlier suggestions but can still be proven secure when E is a good block cipher.

Let $E : \{0,1\}^\alpha \times \{0,1\}^\beta \to \{0,1\}^\beta$ be the block cipher. We define $F_{K1 \parallel K2 \parallel K3}(M)$, where $|K_1| = |K_2| = |K_3| = \alpha$ and $M \in \{0,1\}^*$, as follows.

If M contains an integral number of blocks (i.e., $|M|$ is a nonzero multiple of β) then apply the "basic" CBC-MAC construction to M except use key $K2$ for encrypting the last block and use key $K1$ for encrypting the earlier ones. More precisely, writing $M = M_1 \parallel \cdots \parallel M_n$, where $|M_1| = \cdots = |M_n| = \beta$, and letting $y_0 = 0^\beta$, set $y_i = E_{K1}(M_i \oplus y_{i-1})$ for $1 \leq i < n$, and then define $F_{K1 \parallel K2 \parallel K3}(M)$ as $E_{K2}(M_n \oplus y_{n-1})$.

If $|M| = 0$ or $|M|$ is not a multiple of β then append to M a 1-bit and then the minimum number of 0-bits so that the resulting string M' will have length which is a multiple of β. Now proceed as above, but using key K_3 in lieu of K_2: that is, writing $M' = M'_1 \parallel \cdots \parallel M'_n$, where $|M'_1| = \cdots = |M'_n| = \beta$, and letting $y_0 = 0^\beta$, set $y_i = E_{K1}(M'_i \oplus y_{i-1})$ for $1 \leq i < n$, and then define $F_{K1 \parallel K2 \parallel K3}(M)$ as $E_{K3}(M'_n \oplus y_{n-1})$.

Building on [19], which in turn builds on [4], it is possible to show that if E is a good PRF (on its domain) then F is a good PRF (on its domain). The quantitative bounds are as in [19]. Further details are omitted.

We comment that since F_A is being applied to adversarially-unknown points, even if it were to have some cryptographic weakness as a PRF, still this might not, by itself, lead to the resulting MAC being insecure. This is true regardless of how the PRF F is realized.

9 Directions

An interesting possibility (suggested to us by researchers at NAI Labs—see acknowledgments) is to restructure UMAC so that a receiver can verify a tag to various forgery probabilities—e.g., changing UMAC-MMX-60 to allow tags to be verified, at increasing cost, to forging probabilities of 2^{-15}, 2^{-30}, 2^{-45}, or 2^{-60}. Such a MAC need be no longer than it is now, and perhaps it need take no longer to generate. Such a feature is particularly attractive for authenticating broadcasts to receivers of different security policies or computational capabilities.

Our efforts to further improve upon UMAC continue, and, at the time of this writing, are working on a new version of the specification document. We expect that UMAC 2.0, as we are calling it, will be a little bit faster and simpler than what we have currently specified in [6]. It will also achieve the property mentioned above, and it will no longer require a variable-input-length PRF. Additionally, the specification document will "export" the underlying hash function, for use in contexts beyond message authentication.

For the authors, who like to mix up theory and practice, the development of UMAC has involved an unprecedented intermixing of theory, experimentation, and painstaking engineering. We think that this has led to a MAC which is faster and more versatile than anything you could devise in the absence of any of these three elements. Thus we hope that UMAC will not only get used, but will help further the underlying design approach we used.

Finally, we comment that, to the best of our knowledge, UMAC is completely patent unencumbered.

Acknowledgments

Thanks to Mihir Bellare and the CRYPTO '99 Committee for their suggestions. Thanks to Dave Balenson and Dave Carman (of NAI Labs, the security research division of Network Associates; DARPA F30602-98-C-0215) for the idea mentioned in Section 9. Thanks to them, and to Bill Aiello, for pointing out the usefulness of high-forgery-probability MACs, such as UMAC-MMX-15, to applications like telephony and multimedia.

Rogaway, Black, and Krovetz were supported by Rogaway's NSF CAREER Award CCR-962540, and by MICRO grants 97-150 and 98-129, funded by RSA Data Security, Inc., and ORINCON Corporation. Much of Rogaway's work was carried out while on sabbatical at Chiang Mai University, under Prof. Krisorn Jittorntrum and Prof. Darunee Smawatakul.

References

[1] Afanassiev, V., Gehrmann, C., and Smeets, B. Fast message authentication using efficient polynomial evaluation. In *Proceedings of the 4th Workshop on Fast Software Encryption* (1997), vol. 1267, Springer-Verlag, pp. 190–204.

[2] ANSI X9.9. American national standard — Financial institution message authentication (wholesale). ASC X9 Secretariat – American Bankers Association, 1986.

[3] BELLARE, M., CANETTI, R., AND KRAWCZYK, H. Keying hash functions for message authentication. In *Advances in Cryptology – CRYPTO '96* (1996), vol. 1109 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 1–15.

[4] BELLARE, M., KILIAN, J., AND ROGAWAY, P. The security of cipher block chaining. In *Advances in Cryptology – CRYPTO '94* (1994), vol. 839 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 341–358.

[5] BERNSTEIN, D. Guaranteed message authentication faster than MD5. Unpublished manuscript, 1999.

[6] BLACK, J., HALEVI, S., HEVIA, A., KRAWCZYK, H., KROVETZ, T., AND ROGAWAY, P. UMAC — Message authentication code using universal hashing. Unpublished specification, `www.cs.ucdavis.edu/~rogaway/umac`, 1999.

[7] BLACK, J., HALEVI, S., KRAWCZYK, H., KROVETZ, T., AND ROGAWAY, P. UMAC: Fast and secure message authentication. In *Advances in Cryptology – CRYPTO '99* (1999), vol. 1666 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 216–233.

[8] CARTER, L., AND WEGMAN, M. Universal hash functions. *J. of Computer and System Sciences*, 18 (1979), 143–154.

[9] ETZEL, M., PATEL, S., AND RAMZAN, Z. Square hash: Fast message authentication via optimized universal hash functions. In *Advances in Cryptology – CRYPTO '99* (1999), vol. 1666 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 234–251.

[10] FIPS 180-1. Secure hash standard. NIST, US Dept. of Commerce, 1995.

[11] GOLDWASSER, S., MICALI, S., AND RIVEST, R. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing 17*, 2 (Apr. 1988), 281–308.

[12] H. KRAWCZYK, M. B., AND CANETTI, R. HMAC: Keyed hashing for message authentication. IETF RFC-2104, 1997.

[13] HALEVI, S., AND KRAWCZYK, H. MMH: Software message authentication in the Gbit/second rates. In *Proceedings of the 4th Workshop on Fast Software Encryption* (1997), vol. 1267, Springer-Verlag, pp. 172–189.

[14] JOHANSSON, T. Bucket hashing with small key size. In *Advances in Cryptology – EUROCRYPT '97* (1997), Lecture Notes in Computer Science, Springer-Verlag.

[15] KALISKI, B., AND ROBSHAW, M. Message authentication with MD5, 1995. Technical newsletter of RSA Laboratories.

[16] KRAWCZYK, H. LFSR-based hashing and authentication. In *Advances in Cryptology – CRYPTO '94* (1994), vol. 839 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 129–139.

[17] MANSOUR, Y., NISSAN, N., AND TIWARI, P. The computational complexity of universal hashing. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing* (1990), ACM Press, pp. 235–243.

[18] NEVELSTEEN, W., AND PRENEEL, B. Software performance of universal hash functions. In *Advances in Cryptology — EUROCRYPT '99* (1999), vol. 1592 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 24–41.

[19] PETRANK, E., AND RACKOFF, C. CBC MAC for real-time data sources. Manuscript 97-10 in `http://philby.ucsd.edu/cryptolib.html`, 1997.

[20] PRENEEL, B., AND VAN OORSCHOT, P. MDx-MAC and building fast MACs from hash functions. In *Advances in Cryptology — CRYPTO '95* (1995), vol. 963 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 1–14.

[21] Preneel, B., and van Oorschot, P. On the security of two MAC algorithms. In *Advances in Cryptology — EUROCRYPT '96* (1996), vol. 1070 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 19–32.

[22] Rivest, R., Robshaw, M., Sidney, R., and Yin, Y. The RC6 block cipher. Available from `http://theory.lcs.mit.edu/~rivest/publications.html`, 1998.

[23] Rogaway, P. Bucket hashing and its application to fast message authentication. In *Advances in Cryptology – CRYPTO '95* (1995), vol. 963 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 313–328.

[24] Shoup, V. On fast and provably secure message authentication based on universal hashing. In *Advances in Cryptology – CRYPTO '96* (1996), vol. 1109 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 74–85.

[25] Tsudik, G. Message authentication with one-way hash functions. In *Proceedings of Infocom '92* (1992), IEEE Press.

[26] Wegman, M., and Carter, L. New hash functions and their use in authentication and set equality. In *J. of Comp. and System Sciences* (1981), vol. 22, pp. 265–279.