To appear in *J. of Cryptology.* Earlier version appears in *Advances in Cryptology – CRYPTO '95.*

Bucket Hashing and its Application to Fast Message Authentication

Phillip Rogaway
Department of Computer Science, University of California,
Davis, CA 95616. e-mail: rogaway@cs.ucdavis.edu

October 13, 1997

Abstract. We introduce a new technique for constructing a family of universal hash functions. At its center is a simple metaphor: to hash a string x, cast each of its words into a small number of *buckets*; xor the contents of each bucket; then collect up all the buckets' contents. Used in the context of Wegman–Carter authentication, this style of hash function provides a fast approach for software message authentication.

Key words: Cryptography, Hashing, Message authentication codes, Universal Hashing.

1 Introduction

Message authentication. Message authentication is one of the most common cryptographic aims. The setting is that two parties, a signer S and verifier V, share a (short, random, secret) key, k. When S wants to send V a message, x, S computes for it a *message authentication code* (MAC), $\sigma \leftarrow \mathrm{MAC}_k(x)$, and S sends V the pair (x, σ). On receipt of (x', σ'), verifier V checks that $\mathrm{MACV}_k(x', \sigma') = 1$.

To describe the security of a message authentication scheme, an adversary E is given an oracle for $\mathrm{MAC}_k(\cdot)$. The adversary is declared *successful* if she outputs an (x^*, σ^*) such that $\mathrm{MACV}_k(x^*, \sigma^*) = 1$ but x^* was never asked of the $\mathrm{MAC}_k(\cdot)$ oracle. For a scheme to be "good," reasonable adversaries should rarely succeed.

Software-efficient MACs. In the current computing environment it is often necessary to compute MACs frequently and over strings which are commonly hundreds to thousands of bytes long. Despite this, there will usually be no special-purpose hardware to help out: MAC generation and verification will need to be done in software on a conventional workstation or personal computer. So to reduce the impact of message authentication on the machine's overall performance, and to facilitate more pervasive use of message authentication, we need to develop faster techniques. This paper provides one such technique.

Two approaches to message authentication. The fastest software MACs in common use today are exemplified by $\mathrm{MAC}_k(x) = h(k \,\|\, x \,\|\, k)$, with h a (software-efficient) cryptographic hash function, such as $h =$MD5 [22]. Such methods are described in [30]. The algorithm HMAC [3] represents the most refined algorithm in this direction. Schemes like these might seem to be about as software-efficient as one might realistically hope for: after all, we are computing one of the fastest types of cryptographic primitives over a string nearly identical in length to that which we want to authenticate. But it is well-known that this reasoning is specious: in particular, Wegman and Carter [32] showed back in 1981 that we do not have to "cryptographically" transform the entire string x.

In the Wegman–Carter approach communicating parties S and V share a secret key $k = (h, P)$ which specifies both an infinite random string P and a function h drawn randomly from a strongly universal$_2$ family of hash functions $\mathcal{H}$. (Recall that $\mathcal{H}$ is strongly universal$_2$ if for all $x \neq x'$, the random variable $h(x) \parallel h(x')$, for $h \in \mathcal{H}$, is uniformly distributed.) To authenticate a message x, the sender transmits $h(x)$ xor-ed with the next piece of the pad P. The thing to notice is that x is transformed first by a non-cryptographic operation (universal hashing) and only then is it subjected to a cryptographic operation (encryption), now applied to a much shorter string.

A standard cryptographic technique —the use of a pseudorandom function family, F— allows S and V to use a short string a in lieu of the infinite string P. Signer S now MACs the i^{th} message, x_i, with $\text{MAC}_{(h,a)}(x_i) = (i, F_a(i) \oplus h(x_i))$.

As it turns out, to make a good MAC it is enough to construct something weaker than a strongly universal$_2$ family. Carter and Wegman [10] also introduced the notion of an almost universal$_2$ family, $\mathcal{H}$. This must satisfy the weaker condition that $\Pr_{h\in\mathcal{H}}[h(x) = h(x')]$ is small for all $x \neq x'$. As observed by Stinson [27], an almost universal$_2$ family can easily be turned into an almost strongly universal$_2$ family by composing the almost universal$_2$ family with an almost strongly universal$_2$ one. In computing $h_2(h_1(x))$, where h_1 is drawn from an almost universal$_2$ family and h_2 is drawn from a strongly universal$_2$ one, the bulk of the time will typically be spent in computing $h_1(x)$, since x may be a long string but $h_1(x)$ will be a short string, and so h_2 won't have much work left to do. Thus the problem of finding a fast-to-compute MAC has effectively been reduced to finding a family of almost universal$_2$ hash functions whose members are fast to compute.

Bucket hashing. This paper provides a new almost universal$_2$ family of hash functions. We call our hash family *bucket hashing.* It is distinguished by its member functions being extremely fast to compute—as few as 6 elementary machine instructions per word (independent of word size) for the version of bucket hashing we concentrate on in this paper. Putting such a family of hash functions to work in the framework described above will give rise to an efficient software MAC.

A bucket hash MAC will involve significant overhead beyond the time which is spent bucket hashing. For one thing, the output of bucket hashing is too long to use directly; it will need to be composed with an additional layer of hashing. All the same, one can compare the instruction count mentioned above to that of MD5, which uses ≈ 36 instructions per 32-bit word [7], and see that there is potential for substantial efficiency gains even if the true cost of using bucket hashing substantially exceeds 6 instructions/word.

A bucket hash MAC has advantages in addition to speed. Bucket hashing is a *linear* function —it is a special case of matrix multiplication over GF(2)— and this linearity yields many pleasant characteristics for a bucket hash MAC. In particular, bucket hashing is *parallelizable,* since each word of the hash is just the xor of certain words of the message. Bucket hashing is *incremental* in the sense of [4] with respect to both `append` and `substitute` operations. Finally, the only processor instructions a bucket hash needs are word-aligned `load`, `store`, and `xor`; thus a bucket hash MAC is essentially endian-indifferent.

In a bucket hash MAC —indeed in any Wegman-Carter MAC— one is afforded the luxury of conservative (slow) cryptography even in a MAC whose software speed has been aggressively optimized. This is because one arranges that the time complexity for the MAC is dominated by the non-cryptographic work.

One might worry that the linearity or simple character of bucket hashing might give rise to some "weakness" in a MAC which exploits it. But it does not. A bucket hash MAC, like any MAC which follows the Wegman–Carter paradigm, enjoys the assurance advantages of provable security. Moreover, this provable security is achieved under extremely "tight" reductions, so that an adversary who can successfully break the MAC can break the underlying cryptographic primitive (the pseudorandom function F) with essentially identical efficiency.

Previous work. The general theory of unconditional authentication was developed by Simmons; see [26] for a survey. As we have already explained, the universal-hash-and-then-encrypt paradigm is due to Wegman and Carter [32]. The idea springs from their highly influential [10].

In Wegman–Carter authentication the size of the hash family corresponds to the number of bits of shared key—one reason to find smaller families of universal hash functions than those of [10, 32]. Siegel (for other reasons) [25] constructs families of fast-to-compute hash functions which use few bits of randomness and have small description size. Stinson finds small hash families in [27], and also gives general results on the construction of universal hash functions. We exploit some of these ideas here. Subsequent improvements (rooted in coding theory) came from Bierbrauer, Johansson, Kabatianskii and Smeets [6], and Gemmell and Naor [12].

The above work concentrates on universal hash families and unconditionally-secure authentication. Brassard [9] first connects the Wegman–Carter approach to the complexity-theoretic case. The complexity-theoretic notion for a secure MAC is a straightforward adaptation of the definition of a digital signature due to Goldwasser, Micali and Rivest [14]. Their notion of an adaptive chosen message attack is equally at home for defining an unconditionally-secure MAC. Thus we view work like ours as making statements about unconditionally-secure authentication which give rise to corresponding statements and concrete schemes in the complexity-theoretic tradition. To make this translation we regard a finite pseudorandom function (PRF) as the most appropriate tool. Bellare, Kilian and Rogaway [5] were the first to formalize such objects, investigate their usage in the construction of efficient MACs, and suggest them as a desirable starting point for practical, provably-good constructions. Finite PRFs are a refinement of the PRF notion of Goldreich, Goldwasser and Micali [13] to take account of the fixed lengths of inputs and outputs in the efficient primitives of cryptographic practice.

Zobrist [33] gives a hashing technique which predates [10] and essentially coincides with one method from [10]. Arnold and Coppersmith [2] give an interesting hashing technique which allows one to map a set of keys k_i into a set of corresponding values v_i using a table only slightly bigger than $\sum_i v_i$. The proof of our main technical result is somewhat reminiscent of their analysis.

Lai, Rueppel and Woolven [19], Taylor [28], and Krawczyk [18] have all been interested in computationally efficient MACs. The last two works basically follow the Wegman–Carter paradigm. In particular, Krawczyk obtains efficient message authentication codes from hash families which resemble traditional cyclic redundancy codes (CRCs), and matrix multiplication using Toeplitz matrices. Though originally intended for hardware, these techniques are fast in software, too. We recall Krawczyk's CRC-like hash in Section 2.

An earlier version of this paper appeared as [23].

Subsequent work. Shoup [24] has carried out implementations and analysis of hash function families akin to polynomial evaluation. Such hash functions make good candidates for "second level hashing" when a speed-optimized hash function is applied to a long string. The techniques are also fast enough to be gainfully employed all by themselves.

Halevi and Krawczyk describe a family of hash functions, MMH, which achieves extremely impressive software speeds on some modern platforms [15]. To achieve such performance one needs the underlying hardware to be able to quickly multiply two 32-bit integers to form a 64-bit product.

Johansson investigates how to reduce the size of the key for bucket hashing, which, in the current paper, is quite enormous [16].

Organization. We continue in Section 2 by reviewing the definition and basic properties of universal hash families. Sections 3 and 4 give our main result. In the former we formally define our family of hash functions, $\mathcal{B}$; we state a theorem which upper bounds the collision probability of $\mathcal{B}$; and we discuss the efficiency of computing functions drawn from $\mathcal{B}$. In the latter we prove our main theorem, relegating one lemma to Appendix A. Section 5 reviews the Wegman-Carter approach for making a MAC out of a family of universal hash functions, while Section 6 gives a concrete example of this and discusses some of the difficulties involved in constructing a good MAC using bucket hashing. Section 7 considers some extensions and directions for our work.

2 Preliminaries

This section provides background drawn from Carter and Wegman [10, 32], Stinson [27], and Krawczyk [18]. Proofs are omitted.

A *family of hash functions* is a finite multiset $\mathcal{H}$ of string-valued functions, each $h \in \mathcal{H}$ having the same nonempty domain $A \subseteq \{0,1\}^*$ and range $B \subseteq \{0,1\}^b$, for some constant b.

Definition 1 [10] *A family of hash functions $\mathcal{H} = \{h : A \to \{0,1\}^b\}$ is* ϵ-**almost universal**$_2$, *written ϵ-AU$_2$, if for all distinct $x, x' \in A$, $\Pr_{h\in\mathcal{H}}[h(x) = h(x')] \le \epsilon$. The family of hash functions $\mathcal{H}$ is* ϵ-**almost XOR universal**$_2$, *written ϵ-AXU$_2$, if for all distinct $x, x' \in A$, and for all $c \in \{0,1\}^b$, $\Pr_{h\in\mathcal{H}}[h(x) \oplus h(x') = c] \le \epsilon$.*

The value of $\epsilon = \max_{x \ne x'}\{\Pr_h[h(x) = h(x')]\}$ is called the *collision probability*. For us, the principle measures of the worth of an AU$_2$ hash family are how small is its collision probability and how fast can one compute its functions.

To make a fast MAC one may wish to "glue together" various universal hash families. The following are the basic methods for doing this.

First we need a way to make the domain of a hash family bigger. Let $\mathcal{H} = \{h : \{0,1\}^a \to \{0,1\}^b\}$. By $\mathcal{H}^m = \{h : \{0,1\}^{am} \to \{0,1\}^{bm}\}$ we denote the family of hash functions whose elements are the same as in $\mathcal{H}$ but where $h(x_1 x_2 \cdots x_m)$, for $|x_i| = a$, is defined by $h(x_1) \parallel h(x_2) \parallel \cdots \parallel h(x_m)$.

Proposition 2 [27] *If $\mathcal{H}$ is ϵ-AU$_2$ then $\mathcal{H}^m$ is ϵ-AU$_2$.*

Sometimes one needs a way to make the collision probability smaller. Let $\mathcal{H}_1 = \{h : A \to \{0,1\}^{b_1}\}$ and $\mathcal{H}_2 = \{h : A \to \{0,1\}^{b_2}\}$ be families of hash functions. By $\mathcal{H}_1 \& \mathcal{H}_2 = \{h : A \to \{0,1\}^{b_1+b_2}\}$ we mean the family of hash functions whose elements are pairs of functions $(h_1, h_2) \in \mathcal{H}_1 \times \mathcal{H}_2$ and where $(h_1, h_2)(x)$ is defined as $h_1(x) \parallel h_2(x)$.

Proposition 3 *If $\mathcal{H}_1$ is ϵ_1-AU$_2$ and $\mathcal{H}_2$ is ϵ_2-AU$_2$ then $\mathcal{H}_1 \& \mathcal{H}_2$ is $\epsilon_1\epsilon_2$-AU$_2$.*

Next is a way to make the image of a hash function shorter. Let $\mathcal{H}_1 = \{h : \{0,1\}^a \to \{0,1\}^b\}$ and $\mathcal{H}_2 = \{h : \{0,1\}^b \to \{0,1\}^c\}$ be families of hash functions. Then by $\mathcal{H}_2 \circ \mathcal{H}_1 = \{h : \{0,1\}^a \to \{0,1\}^c\}$ we mean the family of hash function whose elements are pairs of functions $(h_1, h_2) \in \mathcal{H}_1 \times \mathcal{H}_2$ and where $(h_1, h_2)(x)$ is defined as $h_2(h_1(x))$.

Proposition 4 [27] *If $\mathcal{H}_1$ is ϵ_1-AU$_2$ and $\mathcal{H}_2$ is ϵ_2-AU$_2$ then $\mathcal{H}_2 \circ \mathcal{H}_1$ is $(\epsilon_1 + \epsilon_2)$-AU$_2$.*

Composition can also be used to turn an AU$_2$ family $\mathcal{H}_1$ whose members hash A to B, and an AXU$_2$ family $\mathcal{H}_2$ whose members hash B to C, into an AXU$_2$ family $\mathcal{H}_2 \circ \mathcal{H}_1$ whose members hash A to C. If $B = \{0,1\}^b$ for some small b, and elements of $\mathcal{H}_2$ are fast to compute on this domain, we have effectively "promoted" $\mathcal{H}_1$ from being AU$_2$ to AXU$_2$ at little cost.

Proposition 5 [27] *Suppose $\mathcal{H}_1 = \{h : A \to B\}$ is ϵ_1-AU$_2$, and $\mathcal{H}_2 = \{h : B \to C\}$ is ϵ_2-AXU$_2$. Then $\mathcal{H}_2 \circ \mathcal{H}_1 = \{h : A \to C\}$ is $(\epsilon_1 + \epsilon_2)$-AXU$_2$.*

We end this section with a sample construction for a software-efficient AXU$_2$ hash family, this one due to Krawczyk [18]. Let $n, \ell \geq 1$ be numbers and let $m \in \{0,1\}^{n\ell}$ be the string we wish to hash. We can view m as a polynomial $m(x)$ over GF(2) of degree $n\ell - 1$ (or less) by viewing the bits of m as the coefficients of $x^{n\ell-1}, \ldots, x^2, x, 1$. We then define a family of hash functions $\mathcal{K}[n,\ell] = \{h : \{0,1\}^{n\ell} \to \{0,1\}^\ell\}$ as follows. A random hash function $h \in \mathcal{K}$ is described by a random irreducible polynomial h over GF(2) of degree ℓ. To hash m using h we compute the degree $\ell - 1$ (or less) polynomial $m(x) \cdot x^\ell \bmod h(x)$. Viewing the coefficients of this polynomial as a string of length ℓ gives us the hash function h evaluated at m.

Theorem 6 [18] *$\mathcal{K}[n,\ell]$ is $\frac{n\ell+\ell}{2^{\ell-1}}$-AXU$_2$.*

The efficiency with which hash functions $h \in \mathcal{K}$ can be computed has been studied by Shoup [24] (who also looked at related hash families). These functions are fast to compute— about 6 instructions/byte on a 32-bit machine, assuming $\ell = 64$, and ignoring the time to "preprocess" the function h. Still, for sufficiently long messages, it will be faster to use the bucket hashing technique from the following section.

We comment that there are many other well-known techniques for universal hashing, such as the linear congruential hash (modulo a prime) [10], the shift register hash [31], or the Toeplitz matrix hash [18].

3 Bucket Hashing

Let $X = X_1 \ldots X_n$ be a string, partitioned into n words. To hash X using bucket hashing we will scatter the words of X into N "buckets," then XOR the contents of each bucket, and then concatenate the bucket contents.

Some ways of scattering the words of X work out better than others. In this paper we analyze a particular bucket hashing scheme, which we denote by $\mathcal{B}$. The scheme will depend on parameters n, N, w. Scheme $\mathcal{B}$ will scatter each word into three buckets.

3.1 Defining the bucket hash family $\mathcal{B}$

Fix a word size $w \geq 1$ and parameters $n \geq 1$ and $N \geq 3$. We will be hashing from domain $D = \{0,1\}^{wn}$ to range $R = \{0,1\}^{wN}$. As a typical example, take $w = 32$, $n = 1024$, and $N = 140$. If we want to be explicit, such a family would be denoted $\mathcal{B}[32, 1024, 140]$. For the scheme we describe to make sense we require that $\binom{N}{3} \geq n$.

Each hash function $h \in \mathcal{B}$ is specified by a length-n list of cardinality-3 subsets of $\{1, \ldots, N\}$. We denote this list by $h = h_1 \cdots h_n$. The three elements of h_i are written $h_i = \{h_{i1}, h_{i2}, h_{i3}\}$.

Choosing a random h from $\mathcal{B}[w, n, N]$ means choosing a random length-n list of three-element subsets of $\{1, \ldots, N\}$ subject to the constraint that no two of these sets are the same. That is, we insist that $h_i \neq h_j$ for all $i \neq j$.

Let $h \in \mathcal{B}$ and let $X = X_1 \cdots X_n$ be the string we want to hash, where each $|X_i| = w$. Then $h(X)$ is defined by the following algorithm. First, for each $j \in \{1, \ldots, N\}$, initialize Y_j to 0^w. Then, for each $i \in \{1, \ldots, n\}$ and $k \in h_i$, replace Y_k by $Y_k \oplus X_i$. When done, set $h(X) = Y_1 \parallel Y_2 \parallel \cdots \parallel Y_N$.

In pseudocode we have:

```
for j ← 1 to N do Y_j ← 0^w
for i ← 1 to n do
      Y_{h_{i1}} ← Y_{h_{i1}} ⊕ X_i
      Y_{h_{i2}} ← Y_{h_{i2}} ⊕ X_i
      Y_{h_{i3}} ← Y_{h_{i3}} ⊕ X_i
return Y_1 || Y_2 || ··· || Y_N
```

The computation of a $h(X)$ can be envisioned as follows. We have N buckets, each initially empty. The first word of X is thrown into the three buckets specified by h_1. The second word of X is thrown into the three buckets specified by h_2. And so on, with the last word of X being thrown into the three buckets specified by h_n. Our N buckets now contain a total of $3n$ words. Compute the xor of the words in each of the buckets (with the xor of no words being defined as the zero-word). The hash of X, $h(X)$, is the concatenation of the final contents of the N buckets.

3.2 Collision probability of the bucket hash family $\mathcal{B}$

The collision probability for $\mathcal{B}[w, n, N]$. is the maximum, over all distinct $x, x' \in \{0,1\}^{nw}$, of the probability that $h(x) = h(x')$. Our main theorem gives an upper bound on the collision probability of $\mathcal{B}$. The bound is about $3312N^{-6}$. In other words, $\mathcal{B}[w, n, N]$ is ϵ-AU$_2$ for $\epsilon \approx 3312N^{-6}$.

Theorem 7 [Main result] *Assume $w \geq 1$, $N \geq 32$ and $n \leq \binom{N}{3}/12$. Let ϵ be the collision probability for $\mathcal{B}[w, n, N]$. Then $\epsilon \leq B(N)$, where $B(N) = \lambda(N)\beta(N)$, for $\lambda(N) = 1/(1 - 6/\binom{N}{3})$ and*

$$\beta(N) = \frac{720(N-3)(N-4)(N-5) + 1944(N-3)(N-4)^2 + 648(N-2)(N-3)^2}{N^3(N-1)^3(N-2)^3}.$$

The proof of Theorem 7 is given in Section 4.

Plot of $B(N)$. In Figure 1 we plot $B(N)$ against N. Consulting the graph we see, for example, that if you hash a string down to 140 words the collision probability is about 2^{-31}.

Comments. In the applications of bucket hashing to message authentication one typically wants a collision probability of, say, $\epsilon \leq 2^{-30}$ or less. As can be seen from Figure 1, getting such a small collision probability requires a fairly large value of N. Since N is the length of our hashed string (in words), large values of N are undesirable and typically require additional layers of hashing. An example of this will be illustrated in Section 5.

Note that our bound shows no dependency on w or n (though there is the technical restriction that $n \leq \binom{N}{3}/12$). Indeed it is easy to see (and the proof of Theorem 7 will show) that the collision probability does not depend on w. In fact, it is a consequence of the proof that, when $4 \leq n \leq \binom{N}{3}/12$, the collision probability does not depend on n, either.

Observe that $\lambda(N) = \mathcal{N}/(\mathcal{N} - 36)$, where $\mathcal{N} = N(N-1)(N-2)$. By our assumption that $N \geq 32$ we have that $1 \leq \lambda(N) \leq 1.002$. So the multiplication by $\lambda(N)$ can effectively be ignored; $B(N) \approx \beta(N)$.

We believe that it is possible to relax the restriction $n \leq \binom{N}{3}/12$ all the way to $n < \binom{N}{3}$. However, doing this would add considerable complexity to the proof, yet have relatively little practical value, since the number of buckets, N, needs to quite large in order to obtain what would usually be regarded as a suitably small collision probability.

Explanation. Here is a bit of intuition for what is going on. Suppose an adversary wants to find a pair of distinct messages $x, x' \in \{0,1\}^{wn}$ which are most likely to collide under a function from $\mathcal{B}$. What two messages should she choose? In the proof of Theorem 7 we recast this question into the following one. An adversary will throw t triples of balls into N buckets. Each of the $3t$ balls will land in a random bucket, except for the following constraints: three *distinct* buckets are selected for the three balls of each toss; and no tosses will land in identical triples of buckets. The adversary's goal is the following: *make every bucket end up with an even number of balls in it.* All the adversary can do is choose how many triples of balls, t, she will disperse. The question we must answer is: what choice of t, where $1 \leq t \leq n$, will maximize the adversary's chance to win this game?

It is not hard to guess the right answer to this question: *four.* Here is an explanation. If the adversary tosses just **one** triple of balls into the buckets she can't possibly win: 3 buckets are guaranteed to have an odd number of balls. If she throws out **two** triples of balls she again can not win, thanks to the constraint that no two triples of balls land in identical triples of buckets. If she throws out **three** triples of balls she again can not win because 9 balls can't be distributed into buckets in such a way that every bucket has an even number of balls. If the adversary throws out **four** triples of balls then, finally, she has a chance to win. This seems like it ought to be the best thing for the adversary to do, because it would seem to become increasingly unlikely to get *every* bucket to have an even number of balls when more balls get tossed into the N buckets. Though this intuition is a long way from being formal, four triples of balls does turn out to be the right answer. Translating back into the adversary's original goal, the adversary can do no better than to choose messages X and X' which differ by exactly 4 words: for X these words are, say, 0^w, while for X' these words are, say, 1^w.

3.3 The efficiency of the bucket hash family $\mathcal{B}$

Instruction counts. To get a feel for the efficiency of bucket hashing, let us do some approximate instruction counts for computing a function $h \in \mathcal{B}$. Though instruction counting is an extremely crude predictor of speed, an analysis like this is still a good implementation-independent way to get some feel for our method's potential efficiency.

To construct a good MAC we will probably want a collision probability of $\epsilon \approx 2^{-30}$ (perhaps less) and so, in view of Figure 1, we will be using a reasonably large value of N, say $N \geq 120$. Thus

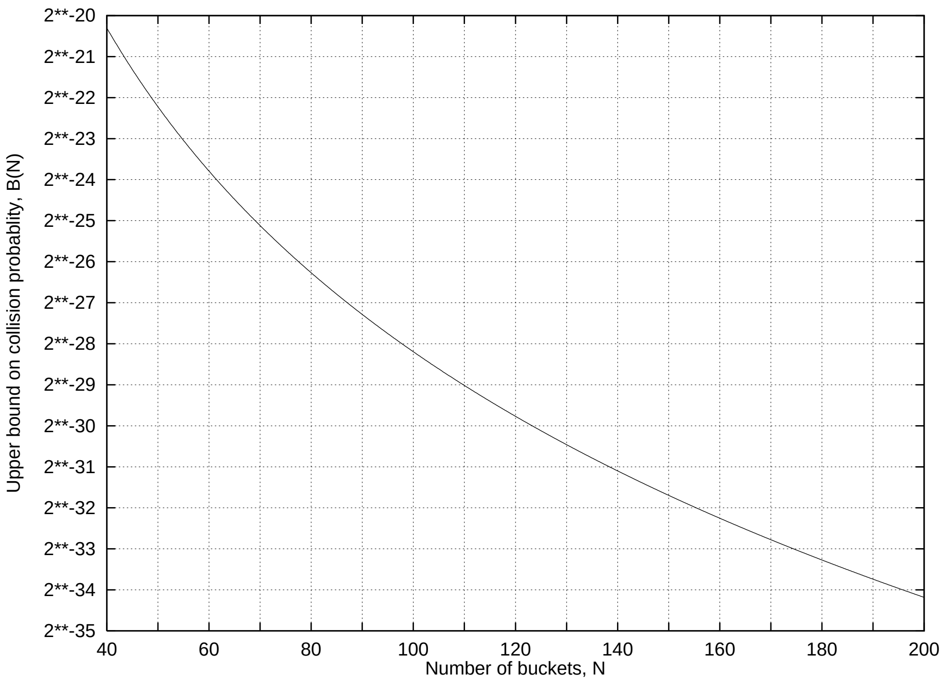

Figure 1: A graphical representation of Theorem 7. We plot of N verses, $B(N)$, our bound on the collision probability of $\mathcal{B}[w, n, N]$.

we will be needing more buckets than can be accommodated by a typical machine's register set. There are then two natural strategies to hash the string $X = X_1 \ldots X_n$, where each X_i is a word of the machine's basic word size:

- **Method-1** (Process words $X_1, \ldots, X_n$). We can read each X_i from memory (in sequence) and then, three times: (1) load from memory the value Y_j of the appropriate bucket j; (2) compute $X_i \oplus Y_j$; (3) store this back into memory, modifying Y_j. Total instruction count is 10 instructions per word (4 reads, 3 writes, 3 xors).
- **Method-2** (Fill buckets $Y_1, \ldots, Y_N$). We can xor together all words that should wind up in bucket 1; then xor all words that go into bucket 2; and so forth, for each of the N buckets. We will need a total of $3n$ reads into $X_1, \ldots, X_n$, plus $3n - N$ xor operations (assuming each bucket contains at least one word). Depending on what we want done with the hash, we may need another N writes to put the hash value back into memory. So the total instruction count is about 6 instructions per word.

Achieving the stated instruction counts requires the use of self-modifying code ("sm-code"); in effect, we implicitly assumed that the representation of $h \in \mathcal{B}$ is the piece of executable code which computes h. In implementation, this can be tricky. If we don't want to use self-modifying code ("$\overline{\text{sm}}$-code") we will need to load from memory the bucket locations (Method-1) or word location (Method-2). This would add 3 loads per word. For Method 2, $\overline{\text{sm}}$-code would further increase the instruction count because of the overhead needed to control the looping: it is h-dependent how many words will fall into a given bucket, so this will have to be read from memory, and loop-unrolling may be difficult. Assuming an additional one instruction per word to account for this work, we have the following approximate instruction counts:

implementation	$\approx$ instrs/wd
Method-1, sm-code	10
Method-1, $\overline{\text{sm}}$-code	13
Method-2, sm-code	6
Method-2, $\overline{\text{sm}}$-code	10

The $\overline{\text{sm}}$-code uses a table to specify h. Assume a machine with a word size of 32 bit. For Method-1 the needed table would typically be $3n$ or $12n$ bytes long (depending on whether one packs bucket indices into bytes or words). For Method-2 that table would typically be be $6n$ or $12n$ bytes long (depending on whether one packs word indices into double-bytes or words), plus an additional N or $4N$ bytes long (depending on whether one packs counter-limits into bytes or words). To get a fast implementation, tables need to fit into cache. Note that there is better locality of reference for Method-1 than Method-2, and this can have a substantial efficiency impact when actually coded.

IMPLEMENTATION. A variety of bucket hashing schemes have been implemented (that is, $\mathcal{B}$ and methods similar to $\mathcal{B}$). The observed performance of these implementations varies enormously according to the particular scheme, the parameters n and N, and the implementation. As a couple points of reference: on a typical 32-bit RISC machine (an SGI with a 150 MHz IP22 processor, 16 KByte data cache, 16 KBytes instruction cache) the most straightforward Method-1/$\overline{\text{sm}}$ implementation ran at 340 Mb/s to hash 1024 words to 140, while a Method-2/sm implementation of a bucket hash family based on the $C[10, 6]$ graph (see Section 7) ran at 1160 Mb/s to hash 909 words to 182.

ROUGH COMPARISONS. Shoup estimates a cost of about 24 instructions/word (6 instructions per byte) for computing a hash function $h \in \mathcal{K}[n, 64]$, where $\mathcal{K}$ is described in Section 2 [24]. Bosselaers, Govaerts and Vandewalle have implemented MD5 at a cost of 36 instructions/word on a Pentium [7] (they obtain a good degree of overlapping instruction-issue, too). In recent work, Halevi and Krawczyk estimate a cost of about 7.5 instructions per word (assuming architectural support for multiplying two 32-bit words to yield a 64-bit product) for their MMH technique [15]. We emphasize that trying to compare such numbers hides many significant factors, including length of hash output (worst for bucket hashing), table sizes and caching issues, and the degree of available parallelism. We have not studied these tradeoffs in detail and do not know if bucket hashing will eventually "win out" in the choice of hash techniques for making a practical MAC.

4 Proof of the Main Theorem

In this section we prove Theorem 7. Throughout this section fix values of n and N satisfying the conditions of the theorem.

Our first two claims show how to simplify the setting.

ONE CAN ASSUME A WORD LENGTH OF $w = 1$. First we argue that, without loss of generality, we can assume that the word length for $\mathcal{B}[w, n, N]$ is $w = 1$. Intuitively, this follows from the "bitwise" character of bucket hashing: when we hash $X_1 \cdots X_n$ down to $Y_1 \cdots Y_N$, where $|X_i| = |Y_j| = w$, the ℓ-th bit of Y_i depends only on $X_1[\ell], \ldots, X_n[\ell]$. For this reason, no advantage can be gained by trying to exploit long words.

Claim 8 $$\max_{\substack{X, X' \in \{0,1\}^{nw} \\ X \neq X'}} \Pr_{H \in \mathcal{B}[w,n,N]} [H(X) = H(X')] = \max_{\substack{x, x' \in \{0,1\}^{n} \\ x \neq x'}} \Pr_{h \in \mathcal{B}[1,n,N]} [h(x) = h(x')] \ .$$

Proof: Let $X, X' \in \{0,1\}^{wn}$ be distinct strings which maximize $\Pr_H[H(X) = H(X')]$. Since $X \neq X'$ there must be some bit position $1 \leq \ell \leq w$ such that the n-bit strings $x = X_1[\ell] \cdots X_n[\ell]$ and $x' = X'_1[\ell] \cdots X'_n[\ell]$ are distinct. Now notice that we can treat any $H \in \mathcal{B}[w,n,N]$ as a hash function $h = H$ from $\mathcal{B}[1,n,N]$, and conversely, because the description of a bucket hash hash function (a sequence of triples of indices) is insensitive to the word length w. Furthermore, $H(X) = H(X')$ implies that $h(x) = h(x')$, and so $\Pr_H[H(X) = H(X')] \leq \Pr_h[h(x) = h(x')]$. We conclude that $\max_{X,X'} \Pr_H[H(X) = H(X')] \leq \max_{x,x'} \Pr_h[h(x) = h(x')]$.

For the opposite inequality, let $x, x' \in \{0,1\}^n$ be distinct strings which maximize $\Pr_h[h(x) = h(x')]$. Write $x = x_1 \ldots x_n$ and $x' = x'_1 \ldots x'_n$, where x_i and x'_i are bits, for all $1 \leq i \leq n$. Define the wn-bit strings $X = X_1 \ldots X_n$ and $X' = X'_1 \ldots X'_n$ by setting $X_i[j] = x_i$ and $X'_i[j] = x'_i$ for each $1 \leq j \leq w$. Clearly $\Pr_h[h(x) = h(x')] = \Pr_H[H(X) = H(X')]$. We conclude that $\max_{x \neq x'} \Pr_h[h(x) = h(x')] \leq \max_{X \neq X'} \Pr_H[H(X) = H(X')]$, as desired. $\diamondsuit$

Given what we have just shown, we henceforth assume a word length as $w = 1$. We will use $\mathcal{B}$ as shorthand for $\mathcal{B}[1,n,N]$.

EXPLOITING LINEARITY. For $0 \leq t \leq n$, let $\mathbf{1}_t = 1^t 0^{n-t}$ and let $\mathbf{0} = 0^N$. For $0 < t \leq n$ define

$$\delta_t \quad = \quad \Pr_{h \in \mathcal{B}}[h(\mathbf{1}_t) = \mathbf{0}] \; .$$

We are trying to bound ϵ, the collision probability of $\mathcal{B}$, which is the maximum, over all distinct $x, x' \in \{0,1\}^n$, of $\Pr_{h \in \mathcal{B}}[h(x) = h(x')]$. We use Claim 8 and the structure of bucket hashing (particularly its linearity) to get the following:

Claim 9 *If* $n \geq 4$ *then* $\epsilon = \max_{t=4,6,8,\ldots} \delta_t$. *If* $n < 4$ *then* $\epsilon = 0$.

Proof: First observe that, for $h \in \mathcal{B}$, computing $h(x)$ amounts to computing a product Ax over GF[2] of an $N \times n$ matrix A and a column vector x. In fact, selecting a random hash function $h \in \mathcal{B}$ corresponds to picking a random binary $n \times N$ matrix A which has three ones in each column and no two identical columns. Writing $\mathcal{A}$ for the set of all such matrices we observe that

$$\begin{aligned}
\epsilon &= \max_{x \neq x'} \Pr_{h \in \mathcal{B}}[h(x) = h(x')] \\
&= \max_{x \neq x'} \Pr_{A \in \mathcal{A}}[Ax = Ax'] \\
&= \max_{x \neq x'} \Pr_{A \in \mathcal{A}}[A(x - x') = \mathbf{0}] \\
&= \max_{x \neq 0^n} \Pr_{A \in \mathcal{A}}[Ax = \mathbf{0}] \\
&= \max_{x \neq 0^n} \Pr_{h \in \mathcal{B}}[h(x) = \mathbf{0}]
\end{aligned}$$

Thus we don't have to think about the probability of distinct strings colliding; it is simpler and more convenient to think about the probability that a non-zero string gets hashed to $\mathbf{0}$.

Next we argue that $\Pr_h[h(x) = \mathbf{0}]$ depends only on the number of ones in x (its Hamming weight), and not on the particular arrangement of zeros and ones within x. Suppose that x has t ones: we claim that $\Pr_A[Ax = \mathbf{0}] = \Pr_A[A\mathbf{1}_t = \mathbf{0}]$. For suppose that the non-zero positions of $x = x_1 \cdots x_n$ are at locations $1 \leq j_1 < \cdots < j_t \leq n$ (meaning that $x_i = 1$ if and only if and only if $i \in \{j_1, \ldots, j_t\}$). Then we pair each matrix $A \in \mathcal{A}$ with a matrix $A' \in \mathcal{A}$ by permuting the columns of A so that columns $j_1, \ldots, j_t$ come first. Then for every $A \in \mathcal{A}$, $Ax = A'\mathbf{1}_t$. Since, for any x, the associated pairing $A \hookrightarrow A'$ is bijective, $\Pr_A[Ax = \mathbf{0}] = \Pr_A[A\mathbf{1}_t = \mathbf{0}]$.

From Claim 8 and what we have just shown, we now know that $\epsilon = \max_{t=1,2,3,\ldots} \delta_t$. So we ask: for which $t \geq 1$ is δ_t largest? One thing is clear: it can not be any any odd-indexed $\mathbf{1}_t$, for if t is odd then $h(\mathbf{1}_t) \neq \mathbf{0}$, because it is impossible to partition $3t$ ones into disjoint sets in such a way that there are an even number of ones in each set. In other words, $\Pr_h[h(\mathbf{1}_t) = \mathbf{0}] = 0$ for odd t. Likewise, $\Pr_A[A\mathbf{1}_2 = \mathbf{0}]$, because of our insistence that no two columns of A are identical. The claim now follows. $\diamondsuit$

Strategy. Our plan is as follows. First we will bound δ_4 from above by $B(N)$. Then we will show that $\delta_t \leq B(N)$ for all even $t \geq 6$. Using Claim 9 we can then conclude that $\epsilon \leq B(N)$.

Our analysis is made possible by using a particular Markov Chain, M. This Markov chain does not accurately describe bucket hashing. But we can correct for the inaccuracy which the chain introduces.

Markov chain model. Consider for a moment an inferior form of bucket hashing: instead of $\mathcal{B}$, where each h_i among $h = h_1 \cdots h_n$ is required to be different from any other, consider the the family of hash functions $\mathcal{C}$, which removes that constraint. In other words, a random $h = h_1 \ldots h_n \in \mathcal{C}[1, n, N]$ is a sequence of random triples, $h_i = \{h_{i1}, h_{i2}, h_{i3}\}$, where $h_{i1}, h_{i2}, h_{i3} \in \{1, \ldots, N\}$ are distinct. This corresponds to a random $N \times n$ binary matrix C with three ones per column.

While there is no natural Markov chain model for $\mathcal{B}$, there is a natural Markov chain M corresponding to $\mathcal{C}$. This chain keeps track of the number of buckets with an odd number of 1's. Thus the Markov chain M has $(N+1)$-states, $\{0, 1, \ldots, N\}$. Being in state i means that i buckets now have an odd number of ones (and $N - i$ buckets have an even number of ones). A transition in M corresponds to throwing three balls into 3 distinct buckets: after each such throw, there is a new number of buckets with an odd number of ones. So state 0 is the start state. Since three balls are tossed with each throw, there can be a non-zero transition probability from states i to j only when $|i - j| \leq 3$. (In fact, the only transitions that can happen are from a state i to a state $j \in \{i-3, i-1, i+1, i+3\} \cap \{0, \ldots, N\}$). The probability of returning to state 0 after t steps corresponds precisely to $\Pr_{h \in \mathcal{C}}[h(\mathbf{1}_t) = \mathbf{0}]$.

Let $\mathcal{N} = N(N-1)(N-2)$. Let P_{ij} denote the transition probability of M: the probability of of moving from state i to state j in a single step. To capture the process $\mathcal{C}$ we have described we need to define M's transition probabilistic as follows:

$$P_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \{(0,3), (N, N-3)\} \\ 3(N-1)(N-2)/\mathcal{N} & \text{if } (i,j) \in \{(1,2), (N-1, N-2)\} \\ (N-1)(N-2)(N-3)/\mathcal{N} & \text{if } (i,j) \in \{(1,4), (N-1, N-4)\} \\ 6(N-2)/\mathcal{N} & \text{if } (i,j) \in \{(2,1), (N-2, N-1)\} \\ 6(N-2)(N-3)/\mathcal{N} & \text{if } (i,j) \in \{(2,3), (N-2, N-3)\} \\ (N-2)(N-3)(N-4)/\mathcal{N} & \text{if } (i,j) \in \{(2,5), (N-2, N-5)\} \\ i(i-1)(i-2)/\mathcal{N} & \text{if } 3 \leq i \leq N-3 \text{ and } j = i-3 \\ 3i(i-1)(N-i)/\mathcal{N} & \text{if } 3 \leq i \leq N-3 \text{ and } j = i-1 \\ 3i(N-i)(N-i-1)/\mathcal{N} & \text{if } 3 \leq i \leq N-3 \text{ and } j = i+1 \\ (N-i)(N-i-1)(N-i-2))/\mathcal{N} & \text{if } 3 \leq i \leq N-3 \text{ and } j = i+3 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Let us give an example of how the above values are computed. Consider P_{ij} for the case associated to $3 \leq i \leq N-3$ and $j = i+1$. In order to go from state i to state $i+1$ in a single step, one ball of

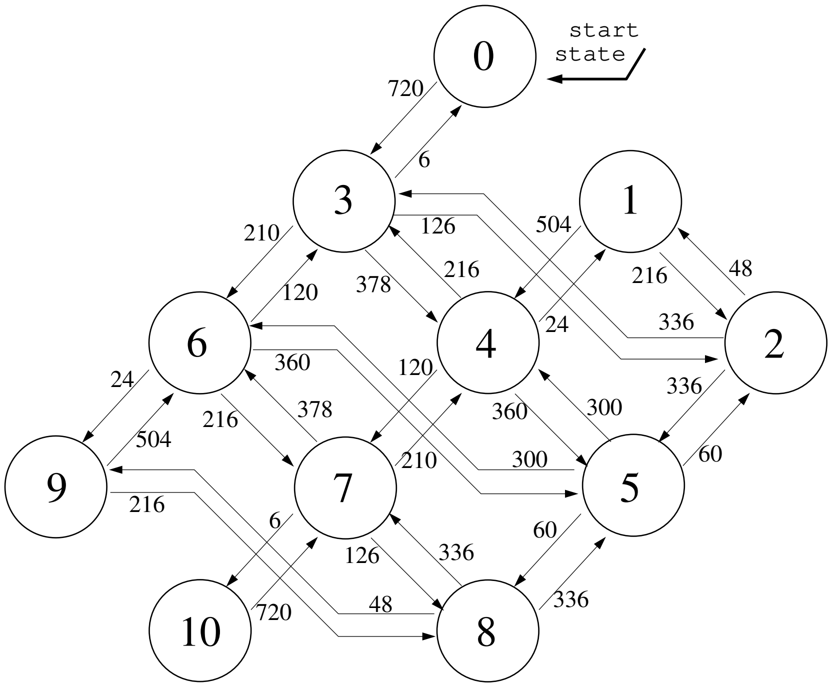

Figure 2: The Markov chain M for $N = 10$ states. The start state is state 0. Divide the number labeling each arc $i \to j$ by $\mathcal{N} = N(N-1)(N-2) = 720$ to get the transition probability P_{ij}.

the three will have to land in one of the i buckets that has an odd number of balls already, while the remaining two balls must land among the $N - i$ remaining buckets. There are $3i(N-i)(N-i-1)$ ordered triples of bucket indices that will accomplish this among the $\mathcal{N}$ ordered triples of bucket indices. (The "3" takes care of the fact that there are $3i$ ways to choose the ball which lands in a bucket with an odd number of balls; after that ball is selected, the remaining two balls have to land in the other $N - i$ buckets.) The reasoning for all of the other P_{ij} values is similar.

In Figure 2 we depict the Markov chain M for some the case where the number of states is $N = 10$. The transition probabilities are computed from Equation 1.

Using M to bound δ_4. We are now ready to show that $\delta_4 \le B(N)$. Recall that $B(N) = \lambda(N) \cdot \beta(N)$, where $\lambda(N)$ and $\beta(N)$ are given by the formulas in the statement of Theorem 7.

Lemma 10 $\delta_4 \le B(N)$.

Proof: First some notation. Let $t \le n$ be a number and let $h_1 \cdots h_t$ be a sequence of triples of distinct elements drawn from $\{1, \ldots, N\}$. We make the following definitions.

- $\mathsf{Parity}(h_1 \cdots h_t)$ is the N-vector whose i-th component, $i \in \{1, \ldots, N\}$, is 0 if i occurs an even number of times in the multiset $h_1 \cup \cdots \cup h_t$, and 1 if i occurs an odd number of times. (Thus $\mathsf{Parity}(h_1 \cdots h_t)$ records the parity of the number of balls in each of the N buckets, if we toss balls according to $h_1, \ldots, h_t$.)
- Given an N-vector of bits $y = y_1 \cdots y_N$, let $\mathsf{NumOnes}(y)$ denote the number of 1-bits in y.
- Define $\mathsf{State}(h_1 \cdots h_t) = \mathsf{NumOnes}(\mathsf{Parity}(h_1 \cdots h_t))$. (Thus $\mathsf{State}(h_1 \ldots h_t)$ records the state of M after hashing $\mathbf{1}_t$ with $h = h_1 \cdots h_t \cdots$. After tossing balls according to $h_1, \ldots, h_t$,

$\mathsf{State}(h_1 \ldots h_t)$ buckets contain an odd number of balls while $N - \mathsf{State}(h_1 \ldots h_t)$ buckets contain an even number of balls.)

- For σ an N-vector of bits, define $\mathsf{State}_\sigma(h_1 \cdots h_t) = \mathsf{NumOnes}(\sigma \oplus \mathsf{Parity}(h_1 \cdots h_t))$. (Thus $\mathsf{State}_\sigma(h_1 \ldots h_t)$ captures the state of M after hashing $\mathbf{1}_t$ with $h = h_1 \cdots h_t \cdots$, given that we start in the configuration specified by σ.)
- Let $\mathsf{Hist}(h_1 h_2 \cdots h_t) = 0\ \mathsf{State}(h_1)\ \mathsf{State}(h_1 h_2)\ \cdots\ \mathsf{State}(h_1 h_2 \cdots h_{t-1})\ \mathsf{State}(h_1 h_2 \cdots h_t)$. This is a list of $t+1$ numbers, each in $\{0, \cdots, N\}$, and it encodes the sequence of states in M one passes through on hashing $\mathbf{1}_t$ according to $h = h_1 h_2 \cdots h_t \cdots$.
- Let $\mathsf{Distinct}(h_1 \cdots h_t)$ be `true` if $h_1, \ldots, h_t$ are all distinct, and `false` otherwise.
- Let R_t ("random") be the uniform distribution on $h_1, \cdots, h_t$ (that is, each h_i is a random triple of distinct points from $\{1, \ldots, N\}$).
- Let D_t ("distinct") be the uniform distribution on distinct $h_1, \ldots, h_t$ (that is, each h_i is a random triple of distinct points from $\{1, \ldots, N\}$, and no two of these triples are identical).
- Let $C(m, t)$ denote the probability of at least one collision in the experiment of throwing t balls, independently and at random, into m bins.

We are now ready to prove the lemma.

$$
\begin{aligned}
\delta_4 &= \Pr_{D_4}[\mathsf{State}(h_1h_2h_3h_4) = 0] \\
&= \Pr_{R_4}[\mathsf{State}(h_1h_2h_3h_4) = 0 \mid \mathsf{Distinct}(h_1h_2h_3h_4)] \\
&= \frac{\Pr_{R_4}[\mathsf{State}(h_1h_2h_3h_4) = 0 \text{ and } \mathsf{Distinct}(h_1h_2h_3h_4)]}{\Pr_{R_4}[\mathsf{Distinct}(h_1h_2h_3h_4)]} \\
&\le \frac{\Pr_{R_4}[\mathsf{Hist}(h_1h_2h_3h_4) \ni \{03630,\ 03430,\ 03230\}]}{1 - C(\binom{N}{3},\ 4)} \qquad (2) \\
&\le \lambda(N) \cdot \Big(\Pr_{R_4}[\mathsf{Hist}(h_1h_2h_3h_4) = 03630] + \Pr_{R_4}[\mathsf{Hist}(h_1h_2h_3h_4) = 03430] + \\
&\qquad \Pr_{R_4}[\mathsf{Hist}(h_1h_2h_3h_4) = 03230] \Big) \\
&= \lambda(N) \cdot \Big(P_{03}P_{36}P_{63}P_{30} \ + \ P_{03}P_{34}P_{43}P_{30} \ + \ P_{03}P_{32}P_{23}P_{30} \Big) \\
&= \lambda(N) \cdot \Big(1 \cdot \frac{(N-3)(N-4)(N-5)}{\mathcal{N}} \cdot \frac{120}{\mathcal{N}} \cdot \frac{6}{\mathcal{N}} \ + \ 1 \cdot \frac{9(N-3)(N-4)}{\mathcal{N}} \cdot \frac{36(N-4)}{\mathcal{N}} \cdot \frac{6}{\mathcal{N}} \ + \\
&\qquad 1 \cdot \frac{18(N-3)}{\mathcal{N}} \cdot \frac{6(N-2)(N-3)}{\mathcal{N}} \cdot \frac{6}{\mathcal{N}} \Big) \qquad (3) \\
&= \lambda(N) \cdot \frac{720(N-3)(N-4)(N-5) + 1944(N-3)(N-4)^2 + 648(N-2)(N-3)^2}{\mathcal{N}^3}
\end{aligned}
$$

$$= \quad B(N)$$

Equation 2 is justified by referring to Figure 2: the only length-4 routes from state 0 back to state 0 are 03630, 03430, 03230, and 03030. The last of these can only arise from non-distinct h_1, h_2, h_3, h_4. For the other three we simply disregard the conjunction with $\mathsf{Distinct}(h_1h_2h_3h_4)$ because we are giving an upper bound. Equation 3 is obtained directly from Equation 1. $\diamondsuit$

USING M TO BOUND $\delta_6, \delta_8, \cdots$. Assume that N is even and $N \geq 6$. We will show, in this case, that $\delta_t \leq B(N)$. Here is the idea. Take a random function $h \in \mathcal{B}$ and look at it's last 6 maps—for convenience of notation, we write $h = h_7 \cdots h_t \;\; h_1h_2h_3h_4h_5h_6$, numbering the final 6 maps $h_1, \ldots, h_6$. Now $h_1, \ldots, h_6$ are statistically correlated to $h_7, \ldots, h_t$ (for example, $h_1 \neq h_7$), yet $h_1, \ldots, h_6$ are not *too* far from being random and independent, in the sense that, for any $h_7 \cdots h_t$, a uniformly selected sequence of maps $h'_1h'_2h'_3h'_4h'_5h'_6$ would have been a valid continuation with probability at least 1/2. (This follows from our assumption that $n \leq \binom{N}{3}/12$.) Thus, up to a factor of 2, we can bound the chance of landing in state 0 on applying h to $\mathbf{1}_t$ by looking at the chance of landing in state 0 after applying a uniformly selected $h_1 \ldots h_6$ starting in some arbitrary (unknown) state of the Markov chain.

To formalize the above argument, let $f_i(t)$ denote the maximum, over all initial states s, of the probability that we arrive in state i in exactly t transitions, given that we start in state s. This is the same as the supremum, over all distributions π on the starting state of M, of the probability that we arrive in state i in exactly t transitions, given that we start in an initial state as chosen by sampling from π. We will need the following lemma about the behavior of Markov chain M.

Lemma 11 $f_0(6) \leq (25920N^8 + 154080N^7)/\mathcal{N}^5$.

The proof is a tedious but straightforward calculation using the transition probabilities of M. It is relegated to Appendix A. The point isn't the specific formula, but only that $f_0(6)$ is less than half $B(N)$ for all sufficiently large N.

Lemma 12 *Assume* $6 \leq t \leq n$. *Then* $\delta_t \leq B(N)$.

Proof: We use the same notation as in the proof of Lemma 10.

$$\begin{aligned}
\delta_t &= \Pr_{h\in\mathcal{B}}[h(\mathbf{1}_t) = \mathbf{0})] \\
&= \Pr_{h_7\cdots h_t\; h_1h_2h_3h_4h_5h_6 \in D_t}[\mathsf{State}(h_7\cdots h_t \;\; h_1h_2h_3h_4h_5h_6) = 0] \\
&= \mathop{\mathbf{E}}_{h_7\cdots h_t \in D_{t-6}}\left[\Pr_{\substack{h_1\cdots h_6 \in D_6 \\ \{h_1,\cdots,h_6\}\cap\{h_7,\cdots,h_t\}=\emptyset}}[\mathsf{State}(h_7\cdots h_t \;\; h_1h_2h_3h_4h_5h_6) = 0]\right] \\
&\leq \max_{h_7\cdots h_t \in D_{t-6}} \Pr_{\substack{h_1\cdots h_6 \in D_6 \\ \{h_1,\cdots,h_6\}\cap\{h_7,\cdots,h_t\}=\emptyset}}[\mathsf{NumOnes}(\mathsf{Parity}(h_7\cdots h_t) \oplus \mathsf{Parity}(h_1h_2h_3h_4h_5h_6)) = 0] \\
&= \Pr_{h_1\cdots h_6 \in E}[\mathsf{NumOnes}(\sigma \oplus \mathsf{Parity}(h_1h_2h_3h_4h_5h_6)) = 0]
\end{aligned}$$

where E and σ are defined by fixing some $h_7 \cdots h_t$ which maximize the probability above and then letting $\sigma = \mathsf{Parity}(h_7 \cdots h_t)$ and letting E be the uniform distribution on $h_1 \cdots h_6$ subject to $h_1, \ldots, h_6$ being distinct from all of $h_7, \ldots, h_t$ and distinct from each other. Continuing, the above expression is:

$$
\begin{aligned}
&= \frac{\Pr_{h_1 \cdots h_6 \in R_6}[\mathsf{State}_\sigma(h_1h_2h_3h_4h_5h_6) = 0 \text{ and } \mathsf{Distinct}(h_1 \cdots h_6\ \ h_7 \cdots h_t)]}{\Pr_{h_1 \cdots h_6 \in D_6}[\mathsf{Distinct}(h_1h_2h_3h_4h_5h_6\ \ h_7 \cdots h_t)]} \\
&\le \frac{\Pr_{h_1 \cdots h_6 \in R_6}[\mathsf{State}_\sigma(h_1h_2h_3h_4h_5h_6) = 0]}{1 - 6\ \ t/\binom{N}{3}} \\
&\le 2 \cdot \Pr_{h_1 \cdots h_6 \in R_6}[\mathsf{State}_\sigma(h_1h_2h_3h_4h_5h_6) = 0] \qquad \textit{//From assumption that } n \le \tbinom{N}{3}/12 \\
&\le 2 \cdot f_0(6) \qquad \textit{//Definition of } f \\
&\le 2 \cdot (25920N^8 + 154080N^7)/\mathcal{N}^5 \qquad \textit{//By Lemma 11} \\
&\le B(N) \text{ for all } N \ge 32
\end{aligned}
$$

For the last inequality: it is easy to verify that this holds for sufficiently large N. The crossover point was determined numerically. $\diamondsuit$

We have now shown that, under the conditions of the theorem, $B(N) \ge \delta_t$ for all $t \ge 1$. This completes the proof. ■

5 From Universal Hash Families to Message Authentication

In this section we review the Wegman-Carter construction (and its complexity-theoretic variant), as well as the formal notion of a message authentication code (MAC) and a pseudorandom function family.

MACs. We follow [14, 5] and define deterministic, counter-based message authentication codes. A MAC scheme $\mathcal{M}$ specifies: constants L and c, determining $\mathsf{Messages} = \{0,1\}^{\le L}$ and $\mathsf{Tags} = \{0,1\}^c$; a set of strings Keys; a number MAX (alternatively, $\mathsf{MAX} = \infty$); and a pair of functions $(\mathrm{MAC}, \mathrm{MACV})$, where

$$
\begin{aligned}
\mathrm{MAC}: &\quad \mathsf{Keys} \times \mathsf{Messages} \times \{1, \ldots, \mathsf{MAX}\} \to \mathsf{Tags}, \quad \text{and} \\
\mathrm{MACV}: &\quad \mathsf{Keys} \times \mathsf{Messages} \times \mathsf{Tags} \to \{0,1\}\ .
\end{aligned}
$$

The first argument to MAC and MACV will usually be written as a subscript. We demand that for any $x \in \mathsf{Messages}$, $k \in \mathsf{Keys}$, and $\mathrm{cnt} \in \{1, \ldots, \mathsf{MAX}\}$, $\mathrm{MACV}_k(x, \mathrm{MAC}_k(x, \mathrm{cnt})) = 1$.

Let $\mathcal{M}$ be a message authentication scheme. A MAC oracle $\mathrm{MAC}_k(\cdot)$ for $\mathcal{M}$ behaves as follows: it answers its first query, x_1, with $\mathrm{MAC}_k(x_1, 1)$; it answers its second query, x_2, with $\mathrm{MAC}_k(x_2, 2)$; and so forth. The MAC oracle responds with the empty string to queries beyond the MAX^{th} or to queries not in the set $\mathsf{Messages}$.

An adversary E for a message authentication scheme $\mathcal{M}$ is an algorithm equipped with a MAC oracle $\mathrm{MAC}_k(\cdot)$. Adversary E is said to forge on a particular execution, this execution having MAC oracle $\mathrm{MAC}_k(\cdot)$, if E outputs a string (x^*, σ^*) where $\mathrm{MACV}_k(x^*, \sigma^*) = 1$ yet E made no oracle query of x^*. When we speak of E forging with a particular probability, that probability is taken over E's coin tosses and a random key $k \in \mathsf{Keys}$ for the MAC oracle. Running times are measured in a standard RAM model of computation, with oracle queries counting as one step. By convention, the running time of E also includes the size of E's description.

One can also provide the adversary with a $\mathrm{MACV}_k(\cdot,\cdot)$ oracle, but this leaves the notion essentially unchanged.

THE WEGMAN-CARTER CONSTRUCTION. Given a family of hash functions $\mathcal{H} = \{A \to \{0,1\}^b\}$ we wish to construct from it a MAC. In the scheme we denote $\mathrm{WC}[\mathcal{H}]$, the Signer and Verifier share a random element $h \in \mathcal{H}$, as well as an infinite random string $P = P_1P_2P_3\cdots$, where $|P_i| = b$. The pair (h, P) is the key shared by the Signer and Verifier. The signer maintains a counter, cnt, which is initially 0. To generate a MAC for the message x the signer increments cnt and then computes the MAC $\sigma = (\mathrm{cnt}, P_{\mathrm{cnt}} \oplus h(x))$ which authenticates x. To verify a MAC $\sigma = (i, s)$ for the message x the Verifier checks if $s = P_i \oplus h(x)$.

The following theorem says that it is impossible (regardless of time, number of queries, or amount of MACed text) to forge with probability exceeding the collision probability.

Proposition 13 [32, 18] *Let $\mathcal{H}$ be ϵ-AXU$_2$ and suppose adversary E forges in the scheme* $\mathrm{WC}[\mathcal{H}]$ *with probability δ. Then $\delta \le \epsilon$.*

PRFS. We follow [13, 5]. A finite pseudorandom function family (PRF) is a map $F : \{0,1\}^\kappa \times \{0,1\}^l \to \{0,1\}^b$. We write $F_a(x)$ in place of $F(a, x)$. Let $R_{l,b}$ be the set of all functions mapping $\{0,1\}^l$ to $\{0,1\}^b$. A *distinguisher* is an algorithm D with access to an oracle. We say that a PRF F is $\epsilon(t, q)$-secure if for every distinguisher D which runs in time t and makes q or fewer queries to its oracle, $\Pr_{k \leftarrow \{0,1\}^\kappa}\left[D^{F_k(\cdot)} = 1\right] - \Pr_{\rho \leftarrow R_{l,b}}\left[D^{\rho(\cdot)} = 1\right] \le \epsilon(t, q)$. Running times are measured in a standard RAM model of computation, with oracle queries counting as one step. By convention, the running time of E also includes the size of E's description.

WEGMAN-CARTER WITH A PRF. A natural complexity-theoretic variant is to use, instead of the random pad P, a random index $a \in \{0,1\}^\kappa$ into a finite PRF $F : \{0,1\}^\kappa \times \{0,1\}^l \to \{0,1\}^b$. The Signer maintains a counter $\mathrm{cnt} \in \{0,1\}^l$, initially 0. (We will not distinguish between numbers and their binary encodings into l-bits.) The Signer and Verifier share a random $a \in \{0,1\}^\kappa$ and a random $h \in \mathcal{H}$. When the Signer wishes to MAC a message x, if $\mathrm{cnt} < 2^l - 1$ then the Signer computes $\sigma = (\mathrm{cnt}, F_a(\mathrm{cnt}) \oplus h(x))$ and increments cnt. (In the unlikely event that cnt reaches $2^l - 1$, a new MAC key is required by the Signer and Verifier.) To verify a MAC $\sigma = (i, s)$ for the message x the Verifier checks if $s = F_a(i) \oplus h(x)$. At most 2^l messages may be MACed (after that, the key a must be changed). We call the scheme just described $\mathrm{WC}[\mathcal{H}, F]$. The following result is obtained by standard techniques.

Proposition 14 *Let $\mathcal{H} = \{h : A \to \{0,1\}^b\}$ be an ϵ-AXU$_2$ family of hash functions. Let $T_{\mathcal{H}}$ denote the time required to compute a representation of a random element $h \in \mathcal{H}$, and let $T_h(q, \mu)$ denote the time required to compute from this representation the hash of q strings, these strings totaling μ bits. Let $F : \{0,1\}^\kappa \times \{0,1\}^l \to \{0,1\}^b$ be an $\epsilon'(t, q)$-secure finite PRF. Let E be an adversary which, in time t, making q queries, these queries totaling μ bits, forges with probability δ against the scheme* $\mathrm{WC}[\mathcal{H}, F]$. *Then $\delta \le \epsilon + \epsilon'(t + \Delta t,\ q + 1)$, where $\Delta t = O(T_h(q, \mu) + T_{\mathcal{H}} + ql + qb)$.*

The value of Δt would usually be insignificant compared to t. Note that in Proposition 13 the forging probability is independent of the number of queries (q) and the length of the queried messages (μ). In Proposition 14 the forging probability depends on these quantities only insofar as they are detrimental to the security of the underlying PRF.

We emphasize that the Signer is stateful in the schemes WC[$\mathcal{H}$] and WC[$\mathcal{H}, F$]. The Signer being stateful improves security (compared with using a random index) and at little practical cost. Note that the Verifier is not stateful. This is possible because our notion of MAC security (Section 5), does not credit the adversary for "replay attacks."

6 Toy Example, and Limitations on Bucket Hashing

In this section we describe a concrete MAC based on the ideas presented so far. This is only a "toy" example; doing a good job at specifying a software-optimized bucket hash MAC would involve much design, experimental, and theoretical work which we have not carried out. Still, the example helps to illustrate the strengths of bucket hashing in making a MAC, as well as the limitations.

TOY EXAMPLE. To keep things simple, suppose the strings we will MAC are of length at most most 4096 bytes. Assume a word size of 4 bytes (32 bits). Let $F : \{0,1\}^\kappa \times \{0,1\}^{64} \to \{0,1\}^{64}$ be a finite PRF (defined, for example, from the compression function of MD5). Here is a way for the Signer to MAC a string X whose length is at most 1024 words. Assume an even number of words. The Signer and Verifier share as a MAC key (i) a random element $h_1 \in \mathcal{B}[32, 1024, 140]$, (ii) a random element $h_2 \in \mathcal{K}[71, 64]$, and a (iii) a random string $a \in \{0,1\}^\kappa$. We use the construction of Proposition 5 (slightly modified to account for length-variability). In the algorithm below, $|X|$ denotes the length of X, encoded as a 2-word string. The function h_1 is extended to strings of length less than 1024 words in the natural way: we stop casting words into buckets when we reach the end of the string. (This is equivalent to 0-padding the string to 1024 words.)

Algorithm $\mathsf{TOY\text{-}MAC}(X)$.
  if $\mathrm{cnt} = 2^{64} - 1$ **then return** error
  $\sigma = \langle \mathrm{cnt},\ F_a(\mathrm{cnt}) \oplus h_2(\,|X|\,.\,h_1(X)) \rangle$
  $\mathrm{cnt} = \mathrm{cnt} + 1$
  return σ

Let us count instructions for TOY-MAC to hash a 4096-byte message. If we bucket hash in 10 instructions per word (Section 3.3), hash using $h_2 \in \mathcal{K}$ in 24 instruction per word ([24]), and compute F with 600 instructions (easy to accomplish), then we will spend $10 + (142/1024) \cdot 24 + 600/1024 = 10 + 3.3 + 0.6 = 13.9$ instructions/word.

Notice that the "cryptographic" contribution to the above time (i.e., the time to compute F) is very small. In a Wegman-Carter MAC one is afforded the luxury of conservative (and slow) cryptography even in an aggressively speed-optimized design. This is because one arranges that the time to compute the MAC is dominated by the non-cryptographic work.

LIMITATIONS ON BUCKET HASHING. If the strings we are MACing are short then, at some point, it makes sense to switch strategies and stop using bucket hashing. In our TOY-MAC, we might hash with only h_2 when the input string has length less than some constant. This is an important limitation on bucket hashing; because the output length is substantial, the technique is simply not useful until the strings to be hashed get long enough. As a consequence, any "real" MAC which employs bucket hashing would likely be a patchwork of different techniques for different message lengths. Therefore a real bucket hash MAC is unlikely to be simple to describe or implement.

On the other hand, if the strings to be hashed are *very* long then, at some point, it makes sense to break the input into blocks and independently bucket has each block, using the construction of Proposition 2. This is because the size of the description of $h \in \mathcal{B}$ grows linearly in the maximal length string which h can hash. We do not want hash functions with excessively long descriptions (certainly the hash function should fit in cache). This is another limitation on the bucket hashing technique, and something which will further complicate the definition of any real bucket hash MAC.

In our TOY-MAC, if we wanted a substantially better collision probability we could apply the construction of Proposition 3, but this would roughly halve the rate for bucket hashing, and perhaps other techniques might then be faster. This is a third limitation on bucket hashing: until better constructions are found, obtaining an extremely small collision probability, say 2^{-50}, would require an excessive number of buckets. That is, the output length of the hash function would be very long, and so the technique would only be useful for hashing extremely long messages.

The last limitation we will mention is the time needed to compute a description of h. In any real MAC scheme the function $h \in \mathcal{B}$ would be determined from some underlying key k with the help of a pseudorandom generator. Because the description of h is large and of a special form, computing h might take a significant amount of time. In most applications of fast message authentications, a one-time key pre-processing delay is not important. But if there is a limited amount of text to be MACed, or if the latency of the first MAC must be minimized, than the time to compute the description of h could be an issue. One approach is to find a version of bucket hashing that uses a small key (ie., a short description for h). This way the underlying pseudorandom generator (if present) is less taxed. This approach has been investigated by [16], who achieves a major reduction in the size of the description the h.

Balanced against these limitations is the possibility of extremely high MAC throughput, at least for long strings.

7 Extensions and Directions

Generalizing $\mathcal{B}$, we call by "bucket hashing" any scheme in which the hash function h is a given by a list $h_1 \cdots h_n$ of "small" subsets of $\{1, \ldots, N\}$ and the hash of $X = X_1 \cdots X_n$, where $|X_i| = w$, is:

for $j \leftarrow 1$ **to** N **do** $Y_j \leftarrow 0^w$
for $i \leftarrow 1$ **to** n **do**
  for each $k \in h_i$ **do**
    $Y_k \leftarrow Y_k \oplus X_i$
return $Y_1 \parallel Y_2 \parallel \cdots \parallel Y_N$

In the general case the distribution on h-values is arbitrary. So $\mathcal{B}$ is just the special case in which we use the uniform distribution on distinct triples in $\{1, \ldots, N\}$.

One could imagine many alternative distributions, some of which will give rise to faster-to-compute hash functions or better bounds on the collision probability. As an example, suppose $h \in \mathcal{H}$ is chosen by randomly re-ordering a list $h_1 \cdots h_n$ of triples which are chosen so that for all sets $I \subseteq \{1, \ldots, n\}$ of cardinality 2 or 4, it is not the case that the multiset $\cup_{h \in I} h$ has an even number of each point $1, \ldots, N$. This new family of hash functions may have substantially smaller collision probability than $\mathcal{B}$ for a given n, N.

The bucket hash scheme of a graph. Hash family $\mathcal{B}$ would have been more efficient had each word gone into two buckets instead of three. One way to specify a scheme where each word lands in two buckets is with a graph G whose N vertices comprise the N buckets and whose m edges $\{1, \ldots, m\}$ indicate the pairs of buckets into which a word may fall. A random hash function from the family is given by a random permutation π on $\{1, \ldots, m\}$. To hash a string $X_1 \ldots X_n$ using π, where $|X_i| = w$ and $n \leq m$, each word X_i is dropped into the two buckets at the endpoints of edge $\pi(i)$. As before, we xor the contents of each bucket and output their concatenation in some canonical order. We call the above scheme the bucket hash of the graph G.

For a graph G to be "good" we want a small number of vertices N, a large number of edges m, and such that for all k where $1 \leq k \leq n \leq m$, if k distinct edges are selected at random from G, then the probability that their union (with multiplicities) comprises a union of cycles is at most some tiny number ϵ.

One possible choice of graphs in this regard are the (d, g)-cages (see [8]). A (d, g)-cage is a smallest d-regular graph whose shortest cycle has g edges. These graphs have been explicitly constructed for various values of (d, g). Though (d, g)-cages are rather large (for even g they have at least $(2(d-1)^{g/2}-2)/(d-2)$ nodes) and the definition of a (d, g)-cage does not exactly correspond to having small collision probability, we conjecture that some (d, g)-cages may still give rise to useful hash families. For example, assume $d-1$ is a prime power. Let $C[d, 6]$ be the $(d, 6)$-cage. This is the the point-line incidence graph of the projective plane of order $d-1$. Bucket hashing with $C[10, 6]$ may be a good way to hash 909 words down to 182 words.

Open questions. The generalized notion of bucket hashing amounts to saying that hashing is achieved for each bit position $1 \ldots w$ by matrix multiplication with a sparse Boolean matrix H. Expressing the method in this generality raises questions like the following: for a given N, n and k, for what distributions $\mathcal{D}$ of binary $N \times n$ matrices H having k ones per column is $\max_{x \in \{0,1\}^n - \{0^n\}} \Pr_{H \in \mathcal{D}} [Hx = \mathbf{0}]$ minimized? What if we also demand that each row has a fixed number of ones? What if, instead of saying that there are k ones per column, we cap the density of the matrix at some value ρ? Answers to such questions may lead to faster bucket hash schemes.

Acknowledgments

Many thanks to the two anonymous referees for their careful reviews. Thanks also to Mihir Bellare, Don Coppersmith, Hugo Krawczyk, and David Zuckerman for their comments and suggestions.

This work was supported in part by NSF CCR-9624560.

References

[1] N. Alon, O. Goldreich, J. Håstad and R. Peralta, Simple constructions of almost k-wise independent random variables, 31st Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1990, pp. 544–553.

[2] R. Arnold and D. Coppersmith, An alternative to perfect hashing, IBM RC 10332 (1984).

[3] M. Bellare, R. Canetti and H. Krawczyk, Keying hash functions for message authentication, *Advances in Cryptology – CRYPTO '96*, Lecture Notes in Computer Science, vol. 1109, Springer-Verlag, 1996, pp. 1–15.

[4] M. Bellare, O. Goldreich and S. Goldwasser, Incremental cryptography: the case of hashing and signing, *Advances in Cryptology – CRYPTO '94*, Lecture Notes in Computer Science, vol. 839, Springer-Verlag, 1994, pp. 216–233.

[5] M. Bellare, J. Kilian and P. Rogaway, The security of cipher block chaining, *Advances in Cryptology – CRYPTO '94*, Lecture Notes in Computer Science, vol. 839, Springer-Verlag, 1994, pp. 341–358.

[6] J. Bierbrauer, T. Johansson, G. Kabatianskii and B. Smeets, On families of hash functions via geometric codes and concatenation, *Advances in Cryptology – CRYPTO '93*, Lecture Notes in Computer Science, vol. 773, Springer-Verlag, 1994, pp. 331–342.

[7] A. Bosselaers, R. Govaerts and J. Vandewalle, Fast hashing on the Pentium, *Advances in Cryptology – CRYPTO 96*, Lecture Notes in Computer Science, vol. 1109, Springer-Verlag, 1996, pp. 298–312.

[8] J. Bondy and U. Murty, *Graph theory with Applications,* North Holland, 1976.

[9] G. Brassard, On computationally secure authentication tags requiring short secret shared keys, *Advances in Cryptology – CRYPTO '82*, Springer-Verlag, 1983, pp. 79–86.

[10] L. Carter and M. Wegman, Universal hash functions, *J. of Computer and System Sciences* 18, 1979, pp. 143–154.

[11] Y. Desmedt, Unconditionally secure authentication schemes and practical and theoretical consequences, *Advances in Cryptology – CRYPTO '85*, Lecture Notes in Computer Science, vol. 218, Springer-Verlag, 1985, pp. 42–45.

[12] P. Gemmell and M. Naor, Codes for interactive authentication, *Advances in Cryptology – CRYPTO '93*, Lecture Notes in Computer Science, vol. 773, Springer-Verlag, 1994, pp. 355–367.

[13] O. Goldreich, S. Goldwasser and S. Micali, How to construct random functions, *Journal of the ACM*, Vol. 33, No. 4, 1986, pp. 210–217.

[14] S. Goldwasser, S. Micali and R. Rivest, A digital signature scheme secure against adaptive chosen-message attacks, *SIAM Journal of Computing,* vol. 17, no. 2 (April 1988), pp. 281–308.

[15] S. Halevi and H. Krawczyk, MMH: Message authentication in software in the Gbit/second rates, *Proceedings of the 4th Workshop on Fast Software Encryption*, Springer-Verlag, 1997.

[16] T. Johansson, Bucket hashing with small key size, *Advances in Cryptology – EUROCRYPT '97,* Lecture Notes in Computer Science, Springer-Verlag, 1997.

[17] T. Johansson, G. Kabatianskii and B. Smeets, On the relation between A-codes and codes correcting independent errors. *Advances in Cryptology – EUROCRYPT '93,* Lecture Notes in Computer Science, vol. 765, Springer-Verlag, 1994, pp. 1–11.

[18] H. Krawczyk, LFSR-based hashing and authentication, *Advances in Cryptology – CRYPTO '94,* Lecture Notes in Computer Science, vol. 839, Springer-Verlag, 1994, pp. 129–139.

[19] X. Lai, R. Rueppel and J. Woollven, A fast cryptographic checksum algorithm based on stream ciphers, *Advances in Cryptology, Proceedings of AUSCRYPT 92,* Lecture Notes in Computer Science, vol. 718, Springer-Verlag, 1992, pp. 339–348.

[20] M. Luby and C. Rackoff, How to construct pseudorandom permutations from pseudorandom functions," *SIAM J. Comput,* vol. 17, no. 2 (April 1988).

[21] P. Pearson, Fast hashing of variable-length text strings, *Communications of the ACM,* vol. 33, no. 6 (1990), pp. 677–680.

[22] R. Rivest, The MD5 message digest algorithm, IETF RFC-1321, 1992.

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

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

[25] A. Siegel, On universal classes of fast high performance hash functions, their time-space trade-off, and their applications, 30th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 1989, pp. 20–25.

[26] G. Simmons, A survey of information authentication, in *Contemporary Cryptography, The Science of Information Integrity,* G. Simmons, editor, IEEE Press, New York, 1992.

[27] D. Stinson, Universal hashing and authentication codes, *Designs, Codes and Cryptography,* vol. 4 (1994), pp. 369–380.

[28] R. Taylor, An integrity check value algorithm for stream ciphers, *Advances in Cryptology – CRYPTO '93*, Lecture Notes in Computer Science, vol. 773, Springer-Verlag, 1994, 40–48.

[29] J. Touch, Performance analysis of MD5, Proc. Sigcomm '95, ACM, 1995, pp. 77–86.

[30] G. Tsudik, Message authentication with one-way hash functions, *Proceedings of Infocom 92,* IEEE Press, 1992.

[31] U. Vazirani, Efficiency considerations in using semi-random sources, *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, ACM Press, 1987, pp. 160–168.

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

[33] A. Zobrist, A new hashing method with applications for game playing, University of Wisconsin, Dept. of Computer Science, TR #88 (April 1970).

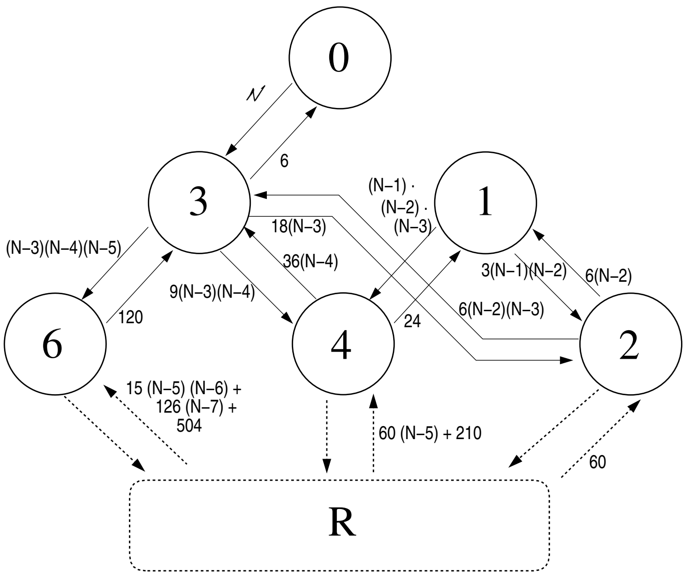

Figure 3: A view of the Markov chain M, where, for purposes of analysis, we have lumped together all states other than $0, 1, 2, 3, 4, 6$. Divide the number labeling each arc $i \to j$ by $\mathcal{N} = N(N-1)(N-2)$ to get the transition probability P_{ij}.

A Proof of Lemma 11

Here we prove Lemma 11, giving a suitable upper bound on $f_0(6)$. We do this by direct calculation, paying attention to the states $0, 1, 2, 3, 4, 6$ and "everything else." To that end, let $R = \{5, 7, 8, \ldots, N\}$ (i.e., "everything else") and define $f_R(t)$ to be the maximum, over all initial states s, of the probability that we arrive at a state $r \in R$ in exactly t transitions, given that we start in state s. Let us write P_{Rj} for $\sum_{r \in R} P_{rj}$. Keep in mind that R is *not* a state of any Markov chain we have defined; this is just a convenient shorthand.

We will establish the bounds indicated in Figure 4, where row t, column i is the upper bound we show for $f_i(t)$. To see how these bounds are computed, refer to Figure 3, which depicts the relevant transition probabilities of M.

We start with the trivial bounds: $f_1(1) \leq 1$, $f_3(1) \leq 1$, $f_R(1) \leq 1$, $f_4(2) \leq 1$, and $f_2(6) \leq 1$. These are obvious, since each $f_i(t)$ represents a probability. Now refer to Figure 3 and calculate. Some of the mundane arithmetic is omitted. In cases such as the calculation of $f_6(4)$, the final inequality is easily seen to hold for sufficiently large N; the crossover point was determined numerically.

$$\begin{aligned} f_0(2) &= f_3(1) \cdot P_{30} \\ &\leq 1 \cdot \frac{6}{\mathcal{N}} \\ &= \frac{6}{\mathcal{N}} \end{aligned}$$

$$\begin{aligned} f_2(2) &\leq f_1(1) \cdot P_{12} + f_3(1) \cdot P_{32} + f_R(1) P_{R2} \\ &\leq 1 \cdot \frac{3(N-1)(N-2)}{\mathcal{N}} + 1 \cdot \frac{18(N-3)}{\mathcal{N}} + 1 \cdot \frac{60}{\mathcal{N}} \end{aligned}$$

	0	**1**	**2**	**3**	**4**	**6**	***R***
$t=1$		1		1			1
$t=2$	$\frac{6}{\mathcal{N}}$		$\frac{3N^2+10N}{\mathcal{N}}$		1	1	
$t=3$		$\frac{42N^3}{\mathcal{N}^2}$		$\frac{54N^4}{\mathcal{N}^2}$			1
$t=4$	$\frac{324N^4}{\mathcal{N}^2}$		$\frac{60}{\mathcal{N}}+\frac{1098N^5}{\mathcal{N}^3}$		$\frac{60N}{\mathcal{N}}+\frac{438N^6}{\mathcal{N}^3}$	$\frac{15N^2}{\mathcal{N}}+\frac{25N^4}{\mathcal{N}^2}$	
$t=5$				$\frac{4320N^8+25680N^7}{\mathcal{N}^4}$			
$t=6$	$\frac{25920N^8+154080N^7}{\mathcal{N}^5}$						

Figure 4: Row t, column i gives our bound on $f_i(t)$. The only value needed is $f_0(6)$; we need that $2f_0(6) \leq B(N)$.

$$
\begin{aligned}
&\leq \frac{3N^2+10N}{\mathcal{N}} \ (N \geq 12) \\
f_1(3) &= f_2(2)\cdot P_{21} + f_4(2)\cdot P_{41} \\
&\leq \frac{3N^2+10N}{\mathcal{N}} \cdot \frac{6(N-2)}{\mathcal{N}} + 1\cdot\frac{24}{\mathcal{N}} \\
&= \frac{(3N^2+10N)(6N-12)+24N(N-1)(N-2)}{\mathcal{N}^2} \\
&\leq \frac{42N^3}{\mathcal{N}^2} \\
f_3(3) &= f_0(2)\cdot P_{03} + f_2(2)\cdot P_{23} + f_4(2)\cdot P_{43} + f_6(2)\cdot P_{63} \\
&\leq \frac{6}{\mathcal{N}}\cdot 1 + \frac{3N^2+10N}{\mathcal{N}}\cdot\frac{6(N-2)(N-3)}{\mathcal{N}} + 1\cdot\frac{36(N-4)}{\mathcal{N}} + 1\cdot\frac{120}{\mathcal{N}} \\
&\leq \frac{54N^4}{\mathcal{N}^2} \\
f_0(4) &= f_3(3)\cdot P_{30} \\
&\leq \frac{54N^4}{\mathcal{N}^3}\cdot\frac{6}{\mathcal{N}} \\
&\leq \frac{324N^4}{\mathcal{N}^3} \\
f_2(4) &\leq f_1(3)\cdot P_{12} + f_3(3)\cdot P_{32} + f_R(3)P_{R2} \\
&\leq \frac{42N^3}{\mathcal{N}^2}\cdot\frac{3(N-1)(N-2)}{\mathcal{N}} + \frac{54N^4}{\mathcal{N}^2}\cdot\frac{18(N-3)}{\mathcal{N}} + 1\cdot\frac{60}{\mathcal{N}} \\
&\leq \frac{60}{\mathcal{N}} + \frac{1098N^5}{\mathcal{N}^3} \ (N \geq 3) \\
f_4(4) &\leq f_1(3)\cdot P_{14} + f_3(3)\cdot P_{34} + f_R(3)\cdot P_{R4}
\end{aligned}
$$

$$
\begin{aligned}
&\leq \frac{42N^3}{\mathcal{N}^2}\cdot 1+\frac{54N^4}{\mathcal{N}^2}\cdot\frac{9(N-3)(N-4)}{\mathcal{N}}+1\cdot\frac{60(N-5)+210}{\mathcal{N}}\\
&\leq \frac{60}{\mathcal{N}}+\frac{438N^6}{\mathcal{N}^3}
\end{aligned}
$$

$$
\begin{aligned}
f_6(4) &\leq f_3(3)\cdot P_{36}+f_R(3)\cdot P_{R6}\\
&\leq \frac{54N^4}{\mathcal{N}^2}\cdot 1+1\cdot\frac{15(N-5)(N-6)+126(N-7)+504}{\mathcal{N}}\\
&= \frac{15N^2}{\mathcal{N}}+\frac{15N^4+189N^3-294N^2-144N}{\mathcal{N}^2}\\
&\leq \frac{15N^4}{\mathcal{N}^2}+\frac{25N^2}{\mathcal{N}}\ (N\geq 18)
\end{aligned}
$$

$$
\begin{aligned}
f_3(5) &= f_0(4)\cdot P_{03}+f_2(4)\cdot P_{23}+f_4(4)\cdot P_{43}+f_6(4)\cdot P_{63}\\
&\leq \frac{324N^4}{\mathcal{N}^3}\cdot 1+\left(\frac{60}{\mathcal{N}}+\frac{1098N^5}{\mathcal{N}^3}\right)\cdot\frac{6(N-2)(N-3)}{\mathcal{N}}+\\
&\qquad\left(\frac{60N}{\mathcal{N}}+\frac{438N^6}{\mathcal{N}^3}\right)\cdot\frac{36(N-4)}{\mathcal{N}}+\left(\frac{15N^2}{\mathcal{N}}+\frac{25N^4}{\mathcal{N}^2}\right)\cdot\frac{120}{\mathcal{N}}\\
&\leq \frac{4320N^8+25680N^7}{\mathcal{N}^4}
\end{aligned}
$$

$$
\begin{aligned}
f_0(6) &= f_3(5)\cdot\frac{6}{\mathcal{N}}\\
&\leq \frac{4320N^8+25680N^7}{\mathcal{N}^4}\cdot\frac{6}{\mathcal{N}}\\
&= \frac{25920N^8+154080N^7}{\mathcal{N}^5}
\end{aligned}
$$

This completes the proof of the lemma. $\diamondsuit$