Message Authentication Codes

by

JOHN R. BLACK, JR.

B.S. (California State University at Hayward) 1988

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Chair

Committee in Charge

2000

Message Authentication Codes

This research was supported by Rogaway's CAREER award CCR-962540, and by MICRO grants 97-150, 98-129, and 99-103 funded by RSA Data Security Inc., ORINCON Corporation, Certicom Corporation, and the State of California.

John Richard Black, Jr.
September 2000
Computer Science

Message Authentication Codes

Abstract

In this thesis, we explore a topic within cryptography called Message Authentication Codes (MACs). A MAC is a collection of algorithms which allows A to send a message to B in such a way that B can be certain (with very high probability) that A did in fact originate the message.

Let's say A wants to send a message (string) M to B using a MAC. First A will run the "tag generation" algorithm on M to produce a string called the "tag." Then A sends M along with the tag to B and B runs the "verification" algorithm to determine whether A was truly the author of M.

The dominant method we use for creating MACs follows an approach first employed by Wegman and Carter [30]. For tag generation we first use a member randomly chosen from a set of hash functions called a "Universal Hash Function Family" to compress M into a smaller string, then we apply some cryptographic primitive to this smaller string to produce the tag. The verification algorithm repeats these steps on M and ensures that the tag generated matches the tag sent.

We examine several known MACs using this paradigm, even when those MACs were not originally designed in this way. This approach often leads to simpler proofs of security for established algorithms. We then relate several new algorithms in the same spirit. The first set of algorithms extends the well-known CBC MAC to accept messages of varying lengths and still retain a high degree of proven security. And in the final chapter we look at a new Universal Hash Function Family called NH which leads to a MAC faster than any currently known.

To Scott R. Nollet,

chess partner, intellectual stimulator, best friend.

Contents

List of Figures

Acknowledgements

My deepest thanks go to my advisor, Phil Rogaway. Not only did I have constant and unlimited access to an incredibly gifted researcher, but I enjoyed the company of a good friend as well. I cannot imagine another human being I would rather have worked with.

Another strong influence in my development as a researcher was Chip Martel; Chip was always available to help me in any way he could, and made a strong positive contribution to my experience here.

Special thanks also to Prof. Fred Chong who was a constant source of good advice as well as a friend and climbing partner.

I also wish to thank my colleagues during the time spent here at Davis. James Pace, Fritz Barnes, and particularly, Steven Samorodin. You have all been great friends. My closest colleague and constant comrade, Ted Krovetz, deserves special mention.

UC Davis was a wonderful place to spend these past five years. I would have gone crazy if not for the wonderful close-knit bunch of climbers in town, especially Trip Armstrong and Jesse Moore.

Thanks also to my close friends who inspired me to pursue my dreams and go for the Ph.D. Roger Kilday, Mike Matrigali, Eric Lundblad, and especially Scott Nollet.

Special thanks to my uncle Paul Doupont, who always believed in me.

Certainly the person who put up with the second-most pain in the writing of this thesis was my girlfriend, Sandra Ruland. Her unflagging support was a constant source of energy and motivation.

Last and not least, I wish to thank Mihir Bellare for his always-friendly guidance.

Chapter 1

Introduction

It's an exciting time to be alive. Computer technology has gone from an obscure toy used by physicists to a commonplace tool in 40% of American homes in the year 1999. This change has taken less than 50 years and has revolutionized (and continues to revolutionize) the way we do business, find entertainment, and communicate with each other. Perhaps the most dramatic force behind recent computerization is the Internet, which has created a global community much faster than the world was ready for it. The economic gains to be realized in this domain have pushed technology in related areas, such as networking, multimedia processing, and security, as well as other domains.

We focus on security: this area is quite vast and encompasses many topics. Intrusion detection, monitoring, and auditing are all areas under the heading of "computer security." But perhaps one of the most interesting areas is cryptography, which has a history pre-dating the electronic computer by many centuries.

1.1 What is Cryptography?

The mention of "Cryptography" evokes in many of us the notions of encryption and decryption. Certainly spy novels and popular movies would have us believe this. However, in recent decades cryptography has come to encompass a much larger array of problems than simply providing private communication via ciphers. At least as important, for example, is message authentication (ie, the ability to verify a message really does originate at a particular source). Rivest defines cryptography as simply "communication in the presence of adversaries" [26].

1.2 Modern Cryptography: A Flawed Science?

Originally, cryptography was done by very simple means: the Romans used simple substitution ciphers for privacy, for example. As cryptanalysis (the science of defeating cryptographic protocols) became more sophisticated, the cryptographic algorithms evolved in an attempt to stay ahead. One could claim that today's very sophisticated block ciphers, such as DES [21] and IDEA [18] are the results of a long evolutionary chain from the simplest ciphers of ancient times through polyalphabetic Vigenère ciphers, rotor machines such as the Enigma of World War II, and various linear feedback shift register machines. However, even with the relatively deep understanding we have achieved in mathematics and theoretical computer science, no one has been able to prove much of anything about the security of these primitives. Much of cryptography requires that there exist *one-way functions*. Informally, a one-way function is a function which is "easy" to compute but "hard" to invert. The goal of the ciphers listed above is to achieve this (strictly speaking, the objects above are not one-way functions but "pseudorandom permutations;" candidate one-way functions can be derived from them via simple constructions).

We do not know if one-way functions exists. Some have said that this unanswered question casts a pall on all of modern cryptography. This claim is incorrect for two reasons: there is a similar unanswered question in complexity theory (does $\mathcal{P} = \mathcal{NP}$?) and this field has made very significant contributions in spite of it. Also, not all techniques in cryptography rely on the existence of one-way functions.

1.3 Provable Security

The modern approach to cryptography requires proofs of security. As just mentioned, we may not be able to prove security to the degree we would like. (If we can forgive some informality, an example might be as follows: it would be pleasing to show that no program exists for a 1000Mhz Pentium III with 1,000 MBytes of RAM and 100 GByte disk which could factor the product of two randomly-chosen 1,000-bit primes in less than 100 years.) The approach adopted by modern researchers has been to follow a similar tack to that used in complexity theory: we use reductions from primitives thought to be "secure" to proposed protocols, and we show that "breaking" the protocol would imply "breaking" the underlying primitive. For example, many cryptographic primitives are based on the belief that factoring the product of two randomly-chosen large primes is intractable. Researchers will then prove that breaking their cryptographic system is equivalent to this factoring problem.

1.4 A Sampling of Problems in Modern Cryptography

Encryption or "privacy" is certainly the best-known cryptographic problem. Typically this problem is stated as follows: Alice (A) wishes to communication with Bob (B) over some communication medium which is subject to interception by some eavesdropper Eve (E). We wish to supply A and B with an algorithm which allows them to communicate in such a way that E can learn everything which passes between A and B and yet can extract no useful information about the content of this communication (we allow for the fact that E learns that something is being communicated (along with its length), though a fair amount of research has been done on how to conceal this as well).

Another important problem in cryptography is "Message Authenticity." In this problem Alice and Bob are at it again, attempting to communicate, and Eve is no longer passively eavesdropping but attempting to impersonate A. The goal is to provide Alice and Bob with an algorithm such that when a message arrives from Alice, Bob can verify that Alice did in fact originate the message.

Several other domains exist under the heading of "cryptography." For example, secret sharing (breaking a secret into many parts and parceling them out in such a way that only authorized subsets of the parts can reconstruct the secret). Also, electronic cash, electronic voting, and a host of other problems are claimed by cryptographic researchers. A good survey is provided by Menezes, van Oorschot, and Vanstone [20].

1.5 Contributions of this Thesis

Much of the first portion of this document is not original. Although the viewpoint is often my own, the ideas in Chapters 1 through 4 are well-known. Some topics may not have been written-up before (eg, the proof of Theorem 4.1.4), but are straightforward exercises, included here for completeness and to give the reader a good base from which to read the later chapters.

Chapter 2 introduces the notation and building blocks used later in the thesis. Chapter 3 introduces formal definitions for MACs and gives a few example MACs. Chapter 4 defines the Carter-Wegman MAC, describing the important concept of a Universal Hash Family, giving many examples. Chapter 5 contains a new proof of Bernstein's "Protected Counter Sums" [5]. In Chapter 6 we begin presenting the new contributions.

Chapter 6 is an adapted version of the paper from CRYPTO 2000, "CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions" by Black and Rogaway. The paper introduces a new problem (extending the domain of the CBC MAC to accept messages of *any* length) and then goes on to show progressively more efficient solutions. Our hope is that these algorithms will be used. The CBC MAC is already widely used, so our hope is that these extensions, with their reassuring proofs of security, will be accepted by the community.

Chapter 7 contains a discussion about a hash family called NH. This family is an integral piece of a fast MACing algorithm "UMAC" which appeared in CRYPTO 1999 in the paper "UMAC: Fast and Secure Message Authentication" by Black, Halevi, Krawczyk, Krovetz, and Rogaway [6]. The family NH was largely developed by Black, Krovetz, and

Rogaway, with Black doing the analytical work, Krovetz the experimental work, and Rogaway driving the design of the project. UMAC is the fastest MAC algorithm yet known and we are excited about both it and a forthcoming second version. Our hope is that those communities who need MACing algorithms which run very fast will be willing to depart from common practice (eg, CBC MAC) and adopt UMAC as their standard. Again, UMAC's security is rigorously proven.

Chapter 2

Preliminaries

Before proceeding to the main topic of the thesis, Message Authentication, we first introduce the notational conventions used throughout and then describe some of the building blocks we will require later on as well.

2.1 Notation

There is a great deal of standard notation used in cryptography. Much of it is standard from other fields (eg. from complexity theory or logic), but some is unique to this field alone. We now briefly introduce this notation.

An integer with a superscript is just the exponentiation of that integer. A symbol with a superscript means a repetition of that symbol, where the number of repetitions is indicated by the exponent. It should be clear from context which meaning is intended. For example 0^{16} means the string of 16 zero bits, but 2^{16} will often mean the quantity 65,536.

A set shown with a superscript is taken to mean the usual cartesian product. If the set is an alphabet (eg. $\{0,1\}$), then taking the set to a power indicates the set of strings generated over that alphabet by the superscript. For example, $\{0,1\}^n$ means all binary strings of length n. The set $\{0,1\}^*$ represents all binary strings, including the empty string. The set $\{0,1\}^+$ is all binary strings *except* the empty string (ie, strings which have length at least 1). The notation $\{0,1\}^{<n}$ denotes all binary strings of length at most $n-1$. Superscripts may be combined, eg., $(\{0,1\}^n)^+$ is the set of non-empty strings whose length is a multiple of n.

A string with a subscript indicates a substring with indices determined by the subscript. If the subscript is a single integer i, we take S_i to mean the ith bit from the left end of string S, counting the first bit as 1. If i is larger than the number of bits of S we regard S_i as undefined. If the subscript is a pair of integers i, j we take $S_{i,j}$ to be the substring of S beginning at S_i and containing all bits up to and including S_j. If $j < i$ or i or j is out of range we again regard $S_{i,j}$ as undefined.

FUNCTIONS. The usual functional notation is used, but sometimes we omit an argument because we don't intend to name that argument. For example, F may be a function with two inputs, but we may write $F(\cdot,\cdot)$ to indicate this rather than $F(x,y)$ which may be confusing if no variables x and y have been quantified. When convenient we may write some of the arguments as a subscript such as writing $F(K,x)$ as $F_K(x)$; we will note this change as it occurs.

We may sometimes wish to pass a function to a function. We will usually use normal functional notation, but sometimes we will prefer to write the passed function as a superscript. For example, let D be an oracle Turing machine and let $\mathcal{O}(\cdot)$ be an oracle which takes one input. We may write $D^{\mathcal{O}(\cdot)} = x$ as a boolean expression testing whether D outputs x after running with oracle $\mathcal{O}$. If we wish to capture the output of D we will write $x \leftarrow D^{\mathcal{O}(\cdot)}$ where x will be of the proper type to accept D's output.

Operations on Strings. When working with strings we will denote concatenation with a dot, so the concatenation of x and y will be written $x \,.\, y$. Sometimes if this is ungainly or unclear we may instead write $x \parallel y$ which means exactly the same thing.

The exclusive-or, or XOR, of two equal-length binary strings x and y is denoted $x \oplus y$. If we use this operator on integers i and j, the integers will always be the same "size" (meaning i and j are taken from some set whose elements can be encoded by strings of a fixed finite length; this length will be clear from context). The result is the integer resulting from a bitwise XOR of the twos-complement representation of i and j.

To measure the size of a string we use vertical bars: $|M|$ represents the number of characters M contains from its underlying alphabet. If the alphabet is not specified we will assume it to be $\{0, 1\}$.

Special Sets. We name two special sets. For sets a and b, $\mathrm{Rand}(a, b)$ indicates all functions from set a to set b. If m and n are integers, we overload the above notation and write $\mathrm{Rand}(m, n)$ to mean the set of all functions from $\{0, 1\}^m$ to $\{0, 1\}^n$. The set $\mathrm{Perm}(a)$ indicates all permutations from set a onto itself. If n is an integer, we overload this notation again: $\mathrm{Perm}(n)$ represents all permutations from the set $\{0, 1\}^n$ onto itself.

Probabilistic Notation. To sample uniformly at random from a set A we write $x \stackrel{R}{\leftarrow} A$ or simply $x \leftarrow A$. For example, to let ρ be a random function from $a = \{0, 1\}^{10}$ to $b = \{0, 1\}$

we could write $\rho \xleftarrow{R} \mathrm{Rand}(a, b)$.

When writing a probability we will either indicate the random choices as a subscript to the probability symbol, or mention them in the surrounding text. If the experiment in the probability symbol is complex, we will sometimes describe it within the brackets, separating the preliminary steps with semicolons and then indicating the final boolean test with a colon. For example,

$$\Pr[c_1 \leftarrow \{\text{heads, tails}\};\ c_2 \leftarrow \{\text{heads, tails}\};\ c_3 \leftarrow \{\text{heads, tails}\} :\ c_1 = c_2 = c_3] = 1/4$$

indicates the experiment of flipping three coins and then determining whether they are all the same.

2.2 Building Blocks

Cryptographic protocols are often built up from lower-level pieces. Oftentimes we describe these pieces as theoretical objects and then prove the security of our protocol based on the assumed security of these pieces (the so-called reductionist approach, cf. Section 1.3). We now discuss two key tools which are useful in the design of algorithms discussed later in this thesis.

2.2.1 Block Ciphers

Perhaps the best-known cryptographic primitive is the block cipher. Cryptographers like to use the alternate name "Finite Pseudorandom Permutation" (Finite PRP) since it is more descriptive. A finite PRP is a family of permutations where a single permutation from the family is selected by a finite string called the "key." Syntactically, we

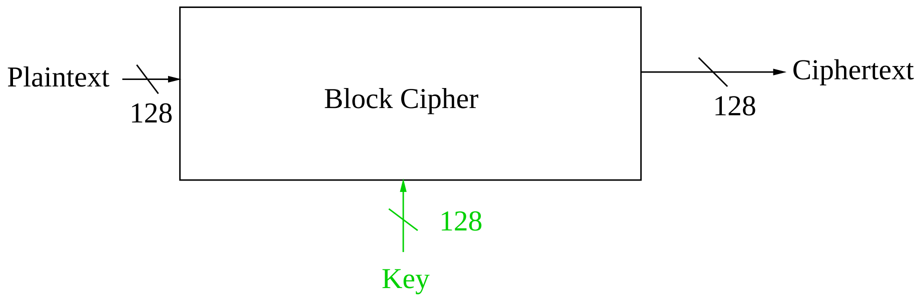

Figure 2.1: A Block Cipher. We can think of a block cipher as a black box: we chose a key of some length (here 128 bits) and give it to the cipher. This produces a permutation on strings of some length (here 128 bits).

define a finite PRP as follows:

Definition 2.2.1 Fix numbers κ and $\ell > 0$. A **finite PRP** or **block cipher**, with key length κ and block length ℓ is a function

$$F \ : \ \{0,1\}^\kappa \ \times \ \{0,1\}^\ell \ \to \ \{0,1\}^\ell$$

where $F(K, \cdot)$ is a permutation for each $K \in \{0,1\}^\kappa$. ∎

The security of a PRP is defined based on its "closeness" to a family of truly random permutations. Since the number of permutations on $\{0,1\}^\ell$ is quite large ($2^\ell!$) any PRP with a short key will have to select only some small subset of the possible permutations. If this subset has some "regularity" about it that can be detected by a "distinguishing algorithm," we consider this an insecurity.

The approach we take is as follows: we consider an adversary which will assume the role of this detecting algorithm. Think of her as a probabilistic oracle Turing machine. We will hand this adversary an oracle either for some PRP or for some random permutation. She will sample from the oracle for some fixed period of time and at the end she outputs a bit. This bit will indicate whether she believes the oracle to be a PRP oracle or a truly random oracle. We quantify over all such adversaries, so if there is *any* exploitable weakness in our PRP, an adversary which exploits this weakness will be considered.

If the adversary is unable to distinguish well between these two types of oracles, we say that the PRP is secure. We now proceed to formally define the notions just discussed.

Definition 2.2.2 Let D be a PRP adversary and let F be a PRP with block length ℓ and key length κ. Define the **advantage** of D as follows:

$$\mathrm{Adv}_D^{\mathrm{prp}}(F) = \Pr[K \leftarrow \{0,1\}^\kappa : \ D^{F_K(\cdot)} = 1] - \Pr[\pi \leftarrow \mathrm{Perm}_\ell : \ D^{\pi(\cdot)} = 1].$$

▮

We can interpret this definition as follows: the left-hand probability is the experiment where we select a key of the appropriate length, then create an F-oracle with this key. The adversary queries this oracle for some amount of time and she outputs a "1" bit with some probability. Now in the right-hand probability the *same* adversary is given a randomly-chosen permutation and she again queries her oracle and outputs "1" with some probability. (We can interpret a "1" as meaning the adversary is guessing that her oracle is a PRP.) The goal of the adversary is to create as large a difference between these two probabilities as possible. We ignore the possibility that advantage can be negative since any adversary which achieves a negative advantage can be replaced by another which achieves the absolute value of the first (just alter the first adversary to output "0" instead of "1", and "1" instead of "0").

We are now ready to define the concept of a secure block cipher.

Definition 2.2.3 Fix some positive key length κ and block length ℓ. Let $F : \{0,1\}^\kappa \times \{0,1\}^\ell$ be a block cipher. We say adversary D can (t, q, ϵ)**-distinguish** F **from a random permutation** if D runs in at most t steps, makes at most q oracle queries and $\text{Adv}_D^{\text{prp}}(F) \geq \epsilon$. ▮

So a secure block cipher will have tiny ϵ even for large q and t for any adversary F.

As a trivial example, let's consider the terrible block cipher X. Let $\kappa = \ell = 64$ and for all $K \in \{0,1\}^\kappa$ and all $M \in \{0,1\}^\ell$ define $X_K(M) = M \oplus K$. Clearly $X_K(\cdot)$ is a permutation for all $K \in \{0,1\}^\kappa$ (indeed it's an involution!). However, it should be intuitively clear that X is not a very secure cipher; let's attack it.

Let's assume we are the adversary D and our oracle is either the PRP oracle $X_K(\cdot)$ or a randomly chosen permutation. If we were allowed only one query, we would not be able to distinguish. But given two queries it's quite easy to win. The simplest attack is to ask for $X_K(0^{64}) = 0 \oplus K = K$. Now if we are in the PRP case we have the key. Now we might ask the query $X_K(1^{64})$; if we get back $K \oplus 1^{64}$ we should output a "1" bit indicating we believe we have a PRP oracle. It's very unlikely that a randomly chosen permutation π would have the property that $\pi(1) = \pi(0) \oplus 1$; in fact the probability of this is $1/(2^{64} - 1)$. So we have

$$\text{Adv}_D^{\text{prp}}(X) = 1 - \frac{1}{2^{64} - 1}$$

which is very close to 1; a very insecure cipher.

Common practical block ciphers are DES [21], IDEA [18], and RC6 [25], among a host of others. We refer the interested reader to the appropriate documents for a description of how these ciphers work, but we again stress that no proofs of security are given for them, and we do not know their exact security parameters. The trust we bestow on these primitives stems principally from the lack of success attackers have had against them. So if we can prove that breaking some higher-level protocol implies low security of these well-known block ciphers, it does provide some assurance that our protocol is sound since no one has been able to mount an effective attack on these ciphers, despite the attempts of scores of clever people.

2.2.2 Pseudorandom Functions

Pseudorandom Functions (or PRFs) [12] are another extremely useful building block used in cryptographic protocols. PRFs are a natural relaxation of PRPs: whereas PRPs were required to be *permutations*, PRFs need only be *functions*. We focus on PRFs with finite domain and range.

The syntax and definition of security are completely analogous to the above but are given here for completeness.

Definition 2.2.4 Fix numbers κ, ℓ, and $L > 0$. A **finite PRF** with key length κ, input length ℓ, and output length L is a function

$$F \;:\; \{0,1\}^{\kappa} \;\times\; \{0,1\}^{\ell} \;\rightarrow\; \{0,1\}^{L}.$$

$\blacksquare$

Definition 2.2.5 Let D be an adversary and let F be a PRF with input length ℓ, output length L, and key length κ. Define the **advantage** of D as follows:

$$\mathrm{Adv}_D^{\mathrm{prf}}(F) = \Pr[K \leftarrow \{0,1\}^{\kappa} :\; D^{F_K(\cdot)} = 1] - \Pr[\rho \leftarrow \mathrm{Rand}(\ell, L) :\; D^{\rho(\cdot)} = 1].$$

$\blacksquare$

Definition 2.2.6 Fix some positive key length κ and input/output lengths ℓ, L. Let $F : \{0,1\}^\kappa \times \{0,1\}^\ell$ be a PRF. We say adversary D can (t, q, ϵ)**-distinguish** F **from a random function** if D runs in at most t steps, makes at most q oracle queries and $\mathrm{Adv}_D^{\mathrm{prf}}(F) \geq \epsilon$. $\blacksquare$

Additional Language. If there does not exist an adversary which can (t, q, ϵ)-distinguish a PRP or PRF from their truly-random counterparts, we say that the PRP or PRF is (t, q, ϵ)-secure. It may sometimes be more convenient to use a functional form: we say a construction is $\epsilon(t, q)$-secure meaning for fixed t and q, no adversary can obtain an advantage larger than $\epsilon(t, q)$, where $\epsilon(\cdot, \cdot)$ is a function with signature $\mathsf{Z} \times \mathsf{Z} \rightarrow [0, 1]$.

Chapter 3

Message Authentication Codes

Message Authentication Codes (or MACs) are used to ensure a message really did originate from a given source (with exceedingly high probability). We include an introduction to the topic.

3.1 Introduction

Message authentication is used to provide authenticity: did a received message actually originate with the purported sender?

Algorithms which provide message authentication allow for a party A to send a message to a party B in such a way that if the message is modified in route to B then B will almost certainly detect this.

The motivation here is obvious: take the setting in Arizona this year where Democrats were allowed to vote in the primary over the internet. If authentication were not provided, certain unethical parties might vote multiple times or might impersonate other

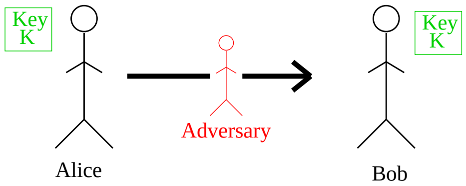

Figure 3.1: The Authentication Problem. User A wishes to send a message along some communications link to user B, but an active adversary may insert information into the channel. Our goal is to provide B with a method to determine which received messages really came from A.

voters hoping to enhance the chances of one candidate over another. (It is interesting to note that the Arizona authorities did not use any cryptographic algorithms, however.)

The starting point for our discussion is to describe a model which best captures the setting for our problem. We will assume a two-party protocol with A and B as the parties. We assume that A and B share some private string K called the "key." This setting is called the "private-key" setting because an identical key is shared by A and B in advance.

In the private-key setting, the tool which enables message authentication is called a *message authentication code*. Here we will examine both the notion of message authentication codes and the question of how to achieve them.

3.2 The Model

Parties A and B share the key K and communicate across a channel on which is present an *active* adversary. The adversary is able to eavesdrop freely and she can modify data in transit or create new messages of her own. Clearly the ability for this adversary to impersonate you would be quite serious: for example, if she could impersonate you to your bank, she might withdraw all your money! (Grad students tend to worry about this less than others, I'm happy to say.)

Realize that we are not concerned about privacy here: we would be happy to let the world see all the communications between A and B. We are concerned only that the adversary cannot "forge" messages to others that make it appear as though we were the original sender.

Our tool for supplying this security is the MAC. A message authentication scheme includes a *MAC generation algorithm* which, on input of a message M and the key K, generates a short *tag*, or *MAC*. Normally we think of this tag as being transmitted along with the message M, as our way of authenticating M. A message authentication scheme also includes a *MAC verification algorithm* which, on input of a message, a key, and a (purported) tag, answers 0 or 1, with 1 indicating that the message M is deemed authentic, and 0 indicating that it is not. We will denote these two algorithms by MAC and VF, writing the key as a subscript. And finally we have a *key generation algorithm* which returns keys (typically random strings) of the appropriate length. We denote this algorithm KGEN.

3.3 Definitions

A message authentication scheme is a 3-tuple $\mathcal{M} = (\mathrm{MAC}, \mathrm{VF}, \mathrm{KGEN})$ called the "MAC generation" (or signing) algorithm, the "MAC verification" algorithm, and the "key generator."

MAC takes a four-tuple (keys, message, state, coins) to a two-tuple (tag, new-state):

$$\mathrm{MAC} : \underbrace{\{0,1\}^*}_{\text{key}} \times \underbrace{\{0,1\}^*}_{\text{Msg}} \times \underbrace{\{0,1\}^*}_{\text{state}} \times \underbrace{\{0,1\}^\infty}_{\text{coins}} \rightarrow \underbrace{\{0,1\}^*}_{\text{tag}} \times \underbrace{\{0,1\}^*}_{\text{new state}}$$

We say "coins" because one may think of flipping coins to get random bits. Really this just means that the MAC algorithm may ingest some random bits as part of its computation if the algorithm is probabilistic. The "state" input and "new state" output are used if the MAC is stateful. We often write simply

$$t \leftarrow \mathrm{MAC}_K(M)$$

where K is the key, M is the message, t is the tag, and the state or coins (if needed) are implicit. For stateful MACs, we reserve one special tag value REKEY which is output when the MAC generation algorithm refuses to continue MACing with the same key.

The verification algorithm, VF, is a mapping from a three tuple (keys, messages, purported-tag) to the binary decision, 1 or 0:

$$\mathrm{VF} : \underbrace{\{0,1\}^*}_{\text{key}} \times \underbrace{\{0,1\}^*}_{\text{Msg}} \times \underbrace{\{0,1\}^*}_{\text{purported tag}} \rightarrow \underbrace{\{0,1\}}_{\text{decision}}$$

The key generator KGEN is a mapping from coins to the key space.

$$\mathrm{KGEN} : \underbrace{\{0,1\}^\infty}_{\text{coins}} \rightarrow \underbrace{\{0,1\}^*}_{\text{key}}$$

Usually the key is just a randomly-chosen string of some designated length.

We demand of any MAC scheme $\mathcal{M}$ that if $t \leftarrow \mathrm{MAC}_K(M)$ then $\mathrm{VF}_K(M,t) = 1$.

3.4 What Makes a MAC Good?

We now know what a MAC is *syntactically*, but of course not just any triple $(\mathrm{MAC}, \mathrm{VF}, \mathrm{KGEN})$ is a good MAC even if it does satisfy the rules above. For a MAC to be considered "good" it must also be difficult for users to forge if they do not hold the key. (A further desirable feature of a good MAC is that it be fast-to-compute; more on this later.)

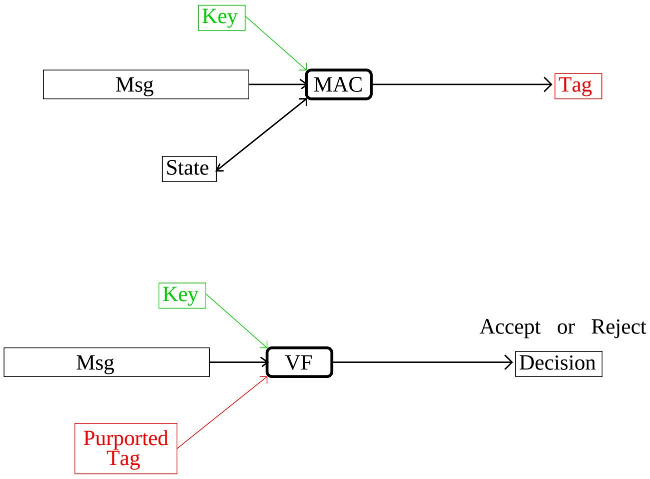

Figure 3.2: The syntax of a MAC. The MAC generator MAC takes the message, Msg, along with the Key and possibly some state and outputs a Tag. The MAC verifier VF, takes the received message, Msg, along with the Key and the purported Tag and outputs either Accept or Reject.

Let's begin by defining our notion of forgery. Here we follow [13] which defined "existential forgery by an adaptive adversary;" they conceived of this notion for a different setting (that of digital signatures instead of MACs), but it was adopted by [3] for our case as well.

Intuitively, we define forgery as follows: let some adversary have a $\text{MAC}_K(\cdot)$ oracle

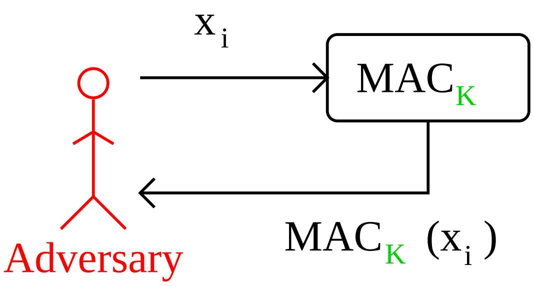

Figure 3.3: Existential Forgery by an Adaptive Adversary. The adversary makes queries $(x_1, \cdots, x_q)$ to the MAC oracle, receiving tags $(\text{MAC}_K(x_1), \cdots, \text{MAC}_K(x_q))$ in return. She then must try and produce a new message with a valid tag; if she succeeds she has "forged."

and let her use this oracle to generate MAC tags for messages of her choice. Of course the key K is concealed within the oracle and cannot be extracted unless she can divine information simply by the tags she generates. When she is done with this phase she is required to produce a *new* message (ie, a message which she did not give to her oracle) along with a valid tag for this message. If she succeeds in this, she is said to have forged. A good MAC has the property that even if the adversary MACs a large number of messages, she forges only with very small probability.

The alert reader might notice that our definition of security allows for the re-transmission of messages. That is, a MAC scheme might be secure against forgeries as defined above and yet does not preclude a message-tag pair from being sent multiple times. A setting where this might be bad is as follows: suppose Alice sends a note to the bank saying "Transfer \$10 from my account to Bob's account" and authenticates this message with tag t. If Bob were dishonest he might intercept this message and then repeat it to the bank multiple times. This is called a replay attack and there are various remedies such as including a timestamp or sequence number on all messages.

Another observation here is that our adversary may be too strong: in real life it is perhaps infeasible for an adversary to acquire MAC tags for messages of her choice. She may instead be limited to collecting tags for messages which are traveling across some channel which she is monitoring. We prefer the definition of this stronger adversary for two reasons:

- One can easily imagine *some* scenarios where the adversary might have this ability (for example in certain entity authentication protocols one player issues a challenge to another in the form of a random string to be MACed).

- Any MAC which is secure against our active adversary is certainly secure against a passive one. And since we *can* achieve efficient MACs secure against active adversaries, we prefer this definition of security.

Let's now give a formal definition for the above notion of security.

Definition 3.4.1 Let $\mathcal{M} = (\mathrm{MAC}, \mathrm{VF}, \mathrm{KGEN})$ be a message authentication scheme. We say that an adversary F (t, q, μ, ϵ)-breaks $\mathcal{M}$ if

$$\begin{aligned}\mathrm{Adv}_F^{\mathrm{mac}} &\stackrel{\mathrm{def}}{=} \Pr\,[K \leftarrow \mathrm{KGEN};\ (m_1, m_2, \cdots, m_q; m^*, t^*) \leftarrow F^{\mathrm{MAC}_K(\cdot)} : \\ &\qquad\qquad \mathrm{VF}_K(m^*, t^*) = 1 \text{ and } m^* \notin \{m_1, m_2, \cdots, m_q\}] \\ &\geq \epsilon\end{aligned}$$

where F runs in time at most t and makes at most q queries to its $\mathrm{MAC}_a(\cdot)$ oracle, the queries totaling at most μ bits. We say $\mathcal{M}$ is (t, q, μ, ϵ)-secure if no adversary (t, q, μ, ϵ)-breaks it. If $\mathcal{M}$ is (t, q, μ, ϵ)-secure for all values of t, q, and μ, we say $\mathcal{M}$ is ϵ-secure. ▮

3.5 A Perfect MAC

Before discussing some of the known MAC algorithms, let's build our intuition by examining a perfect MAC. (Of course our perfect MAC is not really practical or else we would have no reason to continue further on this topic.)

Our perfect MAC will accept any string from $\{0,1\}^*$ and produce an ℓ-bit tag t as its output. Imagine an infinite-length table T with two columns; the left column has all the elements of $\{0,1\}^*$. The right column is filled with randomly-chosen ℓ-bit strings. To compute $\mathrm{MAC}_T(M)$ for any $M \in \{0,1\}^*$ we simply look up M in our table and return the corresponding quantity from the right-hand column.

Why is this MAC good? From our definition of security in Section 3.4, we think of some adversary probing the table at will. Remember, she cannot see the entire table, only specific rows she selects based on which messages she submits to her MACing oracle. After choosing q rows to look at, she then must predict the value in the right column for some row she has *not* yet viewed. But clearly she has learned absolutely nothing from her previous work since we filled each right-hand column slot with independent random ℓ-bit strings! Her chance of correctly guessing this right-hand value is $1/2^\ell$ even if she has unbounded computational power.

Of course this scheme is completely impractical: the table we are using must be shared by users A and B, and its large size makes this infeasible. Even if we were to limit the length of messages we could MAC down to n bits there are still $\ell \cdot 2^n$ bits which need to be shared between A and B. This is prohibitively large for even modest values of n.

So our goal is to find a practical scheme which comes close to the random table described above. The key to practicality is to devise a method of selecting a small proportion of all possible random tables and then to encode these choices by a short string called a "key."

Remark on PRFs: One cryptographic primitive mentioned in Section 2.2 is strongly related to the notions we have been discussing here: the Pseudorandom Function (PRF). PRFs are extraordinarily useful in cryptographic protocol design, and it can be easily shown that any "good" PRF is a good MAC. However the security goal for PRFs is actually stronger than for MACs, so the converse is not true.

3.6 Some Simple Examples

Our goal is a *practical* MAC. Let's make a few stabs just to get a feel for what we are trying to accomplish.

One attempt might be to simply encrypt the message M. In the past some (misguided) people have thought that a correct encryption of M implied that M was also authentic. A simple example of how wrong this thinking is runs as follows: suppose we use a one-time pad to encrypt. This approach involves simply XORing M with $|M|$ bits of key. It is well-known that this method is perfectly secure for the purposes of protecting privacy, but we can easily break this scheme if used as a MAC: let K be an infinite binary string and let i indicate the index within K where bits K_i, K_{i+1}, and so forth have not yet been used. Then the adversary makes one query to her MAC oracle, say $t \leftarrow \mathrm{MAC}_{K_i}(0)$, and then she knows a new message/tag pair:

$$\text{Message} : 1 \qquad\qquad \text{Tag} : \; t \oplus 1.$$

Another approach might be to take a secure block cipher, such as DES [21] and simply compute $\mathrm{MAC}_K(M) = \mathrm{DES}_K(M)$. This works fine provided M is a 64-bit message, but of course we need a method to deal with messages which are much longer. Since it is not an unreasonable restriction to require messages to be a multiple of some constant, let's assume the domain of our MAC is $D = (\{0,1\}^{64})^{+}$. Then to MAC any $M \in D$ we will break M in to 64-bit chunks: $M = M_1.M_2.\cdots.M_j$ where $|M_i| = 64$. And now define

$$\mathrm{MAC}_K(M) = \mathrm{DES}_K\left(\bigoplus_{i=1}^{j} M_i\right).$$

But we may easily observe that this approach is flawed as well: the adversary requests $t \leftarrow \mathrm{MAC}_K(M)$ for $M = 0^{64}.1^{64}$ and then announces the same t works for $M = 1^{64}.0^{64}$.

So something more clever is required. Many approaches which *do* work have been proposed; we now take a look at these.

3.7 A Brief Survey of MAC Algorithms

Good MAC algorithms have been known since the mid-70's. In February of 1977 Campbell [8] published a method where "each group of 64 message bits is passed through the algorithm after being combined with the output of the previous pass." The term "MAC" first appeared around 1980 in the ANSI X9.9 standard [1].

We now proceed to discuss several of the more well-known algorithms for MACing. We will provide a description of the algorithm, but will not discuss proofs of security as this would take us too far afield and is not necessary for an understanding of the thrust of this thesis. The cited references will direct the interested reader to the original paper if further details are desired. Our motivation for a look at these methods now is to foment the idea of what a good MAC looks like and to provide a base for extensions to these MACs later in the thesis.

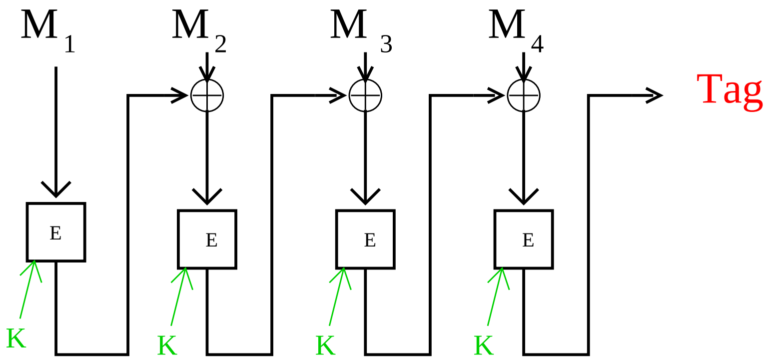

Figure 3.4: The CBC MAC. We begin with a block cipher E with key K and block length ℓ. The message $M \in (\{0,1\}^\ell)^+$ to be MACed is broken into blocks: $M = M_1.\cdots.M_m$ with $|M_i| = \ell$. Then each block is passed through $E_K(\cdot)$ and XORed with the next block.

3.7.1 CBC MAC

The CBC MAC [10, 15] is one of the oldest ideas for a MAC, dating from the mid-1970's. The "CBC" stands for "Cipher Block Chaining" because each message block is chained together with invocations of some block cipher. We now describe how it works.

Fix some positive block length ℓ and some key length κ. Let $E : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^\ell$ be a block cipher (see Section 2.2): it uses a key K to encipher an ℓ-bit block X into an ℓ-bit ciphertext $Y = E_K(X)$. The message space for the CBC MAC is $(\{0,1\}^\ell)^+$, meaning binary strings whose lengths are a positive multiple of ℓ. So let $M = M_1 \cdots M_m$ be a string that we want to MAC, where $|M_1| = \cdots = |M_m| = \ell$. Then $\mathrm{CBC}_{E_K}(M)$, the CBC MAC of M under key K, is defined as C_m, where $C_i = E_K(M_i \oplus C_{i-1})$ for $i = 1, \ldots, m$ and $C_0 = 0^\ell$.

This MAC is probably the most widely used at the present time. The standards describing it include ANSI X9.9, ISO 9797-1, and FIPS 113. CBC MAC was proven secure for fixed length messages by Bellare, Kilian, and Rogaway [3].

3.7.2 XOR MAC

The MACs known as XOR MACs were designed by Bellare, Guérin, and Rogaway in 1995 [2]. We describe here the "randomized" XOR MAC, though the authors also describe a counter-based scheme. They assume the existence of a PRF F with input size ℓ, output size L and key length κ bits. Let M be the message we wish to MAC. We divide M into $\ell/2$-bit blocks: $M = M_1.M_2.\cdots.M_n$ where $|M_i| = \ell/2$. Let $\langle i \rangle$ denote an $\ell/2 - 1$ bit encoding of the integer i. Assume $K \in \{0,1\}^\kappa$ is the shared key. To compute the XOR MAC of a message the sender chooses a random $\ell - 1$ bit string r and then computes the tag (r, t) where

$$t = F_K(0.r) \oplus F_K(1.\langle 1 \rangle.M_1) \oplus F_K(1.\langle 2 \rangle.M_2) \oplus \cdots F_K(1.\langle n \rangle.M_n).$$

The sender then transmits M, t, and r. The verifier simply computes t' from M and r (and K) and ensures $t = t'$. One nice selling point of XOR MACs is that each of the invocations of F can be computed in parallel. This is a nice feature if messages are long and F is an expensive function. Another nice feature is that the MAC is incremental with respect to substitutions: computing the tag for a related message (where, say, only one block is different) requires only a constant amount of computation. [2]

3.7.3 Carter-Wegman MACs

In 1979 Carter and Wegman published the idea of a Universal Hash Family [9], but no mention was made of MACs. Two years later the same authors published a second paper with their idea on how to MAC using Universal Hash Families [30]. Their idea was quite novel: instead of applying some cryptographic primitive to the message M to be

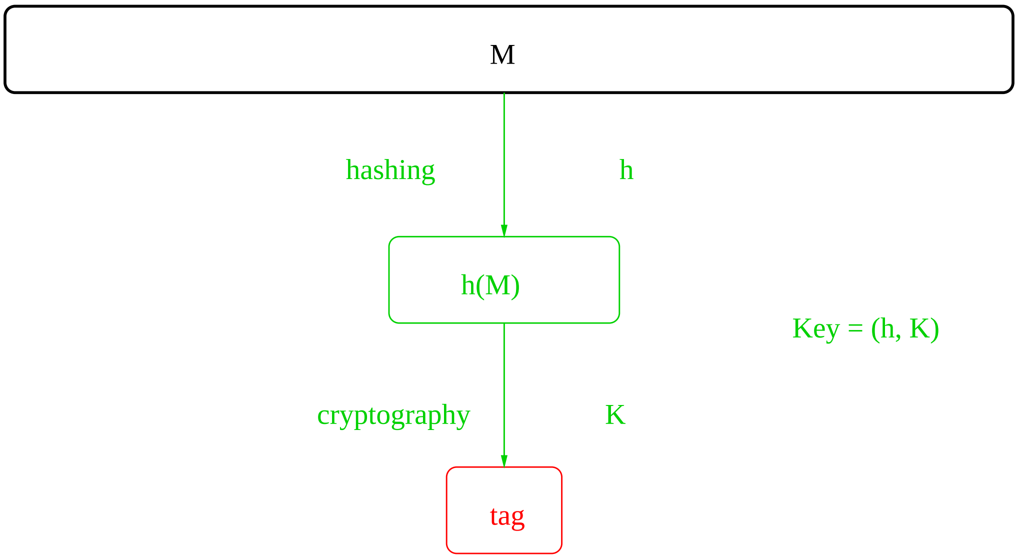

Figure 3.5: Carter-Wegman MACs. The idea is to first apply some hash function h to the message M, compressing the string to a much smaller length. Then we apply the cryptography to a much smaller string.

MACed, they would first hash M down to a smaller size using a hash function which had only a combinatorial property (rather than a cryptographic one). They then would apply a cryptographic primitive–one-time pad encryption–to the smaller resulting string.

In 1983, Brassard [7] proposed using this approach within a complexity-theoretic cryptographic context: whereas Wegman and Carter had encrypted the output using a one-time pad (see Section 3.6), Brassard suggested using a block cipher (eg. DES) for producing a MAC.

Although proposed in the early 1980's, it was not realized that this style of MAC provided an avenue for very efficient software-based MACs until 10 years later. We end this brief discussion at this point and delve more fully into Carter-Wegman MACs in the following chapter.

3.8 A Sample MAC Attack

In Section 3.4 we learned the definition of forgery and in Section 3.6 we saw a simple attack according to this definition. In this section we will give an attack, due to Preneel and van Oorschot [23], which is effective against a strong MAC: the CBC MAC. Of course since the CBC MAC is strong, our attack will require substantial resources in order to have a decent probability of success! However it will establish a lower bound on the probability of forging. This attack is the most effective known against the CBC MAC.

To recapitulate the rules-of-the-game: we are given a MAC oracle for CBC MAC. As mentioned in Section 3.7, CBC MAC is secure only for messages of some fixed length, so this oracle will accept messages of length n blocks, where n is fixed. Let's say the block-length of the underlying cipher is ℓ bits. We generate q messages, $M_1, M_2, \cdots, M_q$, each $n\ell$ bits long, and hand them to the oracle; we receive in response the ℓ-bit tags $t_1, t_2, \cdots, t_q$ corresponding to our messages. Our job, as the attacker, is then to produce a new message M^* which was not previously asked, along with a tag t^* such that t^* is a valid tag for M^*. In other words, if our oracle is $\mathrm{CBC}_\pi(\cdot)$ for permutation $\pi : \{0,1\}^\ell \times \{0,1\}^\ell$, then we must produce an $n\ell$-bit string $M^* \notin \{M_1, \cdots, M_q\}$ and an ℓ-bit string t^* such that $\mathrm{CBC}_\pi(M^*) = t^*$.

Now we give the attack. For simplicity we will assume $n = 2$ (ie, that all messages are two blocks long). The attack can easily be generalized to longer messages (see the full version of [3]). We also assume that q is substantially less than 2^ℓ.

The first step of our attack is to generate $q-1$ random ℓ-bit strings $r_1, r_2, \cdots, r_{q-1}$. Let $\langle i \rangle$ denote the ℓ-bit standard two's complement encoding of the integer i. Then we form the following list of messages to be used as queries to our oracle:

$$\begin{aligned} M_1 &= r_1 \parallel \langle 1 \rangle \\ M_2 &= r_2 \parallel \langle 2 \rangle \\ &\vdots \\ M_{q-1} &= r_{q-1} \parallel \langle q-1 \rangle \end{aligned}$$

In response, we receive back the tags $t_1, t_2, \cdots, t_{q-1}$.

The next step is to check and see if there are any matches in the list of tags. In other words, are there integers $1 \leq i < j \leq q-1$ with $t_i = t_j$? If not we give up and our attack fails. But if there are, we produce a forgery; here is how it's done: we ask one final query (notice that we purposely saved one!). The final query is

$$M_q = r_i \parallel \langle i \rangle \oplus \langle j \rangle \oplus 1^\ell.$$

We obtain tag t_q in response; we will let $t^* = t^q$. And now we give our forgery $M^* = r_j \parallel 1^\ell$.

Of course we must now check two things: we must ensure that M^* is a new message not in $\{M_1, \cdots, M_q\}$, and we must ensure that $\mathrm{CBC}_\pi(M^*) = t^*$. To check the first condition we note that the last ℓ bits of M^* are all 1's. Since q is smaller than $2^\ell - 1$, we know that this string (the two's complement encoding of $2^\ell - 1$) was never the second block of any message previously asked.

To check the second condition we look at the computation performed by the

CBC MAC algorithm. We know

$$t^* = t_q = \pi(\pi(r_i) \oplus \langle i \rangle \oplus \langle j \rangle \oplus 1^\ell) \qquad \text{and} \qquad \pi(\pi(r_i) \oplus \langle i \rangle) = \pi(\pi(r_j) \oplus \langle j \rangle).$$

We need to ensure that $\mathrm{CBC}_\pi(M^*) = t^*$. Expanding this we find we must check if

$$\pi(\pi(r_j) \oplus 1^\ell) = \pi(\pi(r_i) \oplus \langle i \rangle \oplus \langle j \rangle \oplus 1^\ell).$$

Since $\pi()$ is a permutation, this is true iff

$$\pi(r_j) \oplus 1^\ell = \pi(r_i) \oplus \langle i \rangle \oplus \langle j \rangle \oplus 1^\ell,$$

and cancelling the 1^ℓ terms and moving $\langle j \rangle$ to the left we see that the above is true iff

$$\pi(r_j) \oplus \langle j \rangle = \pi(r_i) \oplus \langle i \rangle.$$

And reapplying $\pi()$ on both sides yields the equality we know to be true from the start. And so we have indeed forged.

It remains to compute the probability that this attack will succeed. We must bound from below the chance that there will exist distinct i and j such that $t_i = t_j$. More precisely we wish to bound from below, for all $1 \leq i < j \leq q$ the quantity

$$\Pr_{\pi \leftarrow \mathrm{Perm}(\ell)}[\mathrm{CBC}_\pi(M_i) = \mathrm{CBC}_\pi(M_j)].$$

But for any i, $\mathrm{CBC}_\pi(M_i)$ is simply $\pi(\pi(r_i) \oplus \langle i \rangle)$. Looking more carefully at this, we notice that since r_i was a uniform randomly-selected string, $\pi(r_i)$ will be uniform and random as well. The XOR with some other quantity uncorrelated to $\pi(r_i)$ will not change this fact, so $\pi(r_i) \oplus \langle i \rangle$ is uniformly random as well. And the final application of $\pi()$ again maintains this feature, so for any i, $\mathrm{CBC}_\pi(M_i)$ is simply $\pi(\pi(r_i) \oplus \langle i \rangle)$ is a uniform random ℓ-bit string.

Therefore the probability we seek is simply the chance that the selection of $q-1$ random ℓ-bit strings will contain a repetition. The chance of this is computed by the so-called "birthday bound." The exact analysis of this quantity can be obtained by computing the Ramanujan Q-function for q and 2^ℓ, but a simple analysis (given in Appendix A) tells us that the probability lies between $0.316(q-1)(q-2)/2^\ell$ and $0.5(q-1)(q-2)/2^\ell$.

Interpreting the Result. The attack we gave said, roughly, that the probability of success is at least $0.3q^2/2^\ell$. And so if $\ell = 64$, we would need about 2^{32}, or about 4 billion, queries in order to obtain a chance of at least 0.3 of obtaining a collision (and thus forging). If q is much less than this, our bound quickly gives us no guarantees.

And remember, all of this assumes a model where the attacker has the ability to submit queries of her own making, and then produce a new message which possibly has no meaningful content as a forgery. This may not be a realistic model, but it certainly is stronger than the more intuitive models where the attacker simply listens passively to a communications link and then produces a message of her choosing as a forgery.

Chapter 4

Carter-Wegman MACs

As we saw in Section 3.7, the Carter-Wegman MACs are those which use a function from a Universal Hash Family to compress the message M to be MACed. The output of this hash function is then processed cryptographically to produce the tag. In this chapter we will explore Universal Hash Families and the various Carter-Wegman type constructions along with their security.

4.1 Universal Hash Families

Universal Hash Families have many uses other than MACs. They can be used for solving the Disjoint Set Problem, for typical data retrieval problems, for derandomization, as well as many other uses. There are many different variants of Universal Hash Families, and we now present a few of these as well as some more-general rules which are useful for combining them.

In the following discussion and throughout the thesis will assume that the domain and range of our hash functions are finite sets of binary strings and that the range is smaller than the domain.

Definition 4.1.1 [Carter and Wegman, 1979] Fix a domain D and range R. A finite multiset of hash functions $\mathcal{H} = \{h : D \to R\}$ is said to be **Universal** if for every $x, y \in D$ where $x \neq y$, $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] = 1/|R|$. ∎

An equivalent name to "Universal" is "Universal-2;" the "2" reminds us that we are concerned with *pairs* of distinct points.

The above is the simplest definition, and is the basis for all the variants we'll discuss next. But first we give an example of a Universal family. Probably the most well-known is the "linear hash" where the elements are taken from some field.

Choose any prime p, which we will call the "hash table size." Call the domain D and fix an r such that any $x \in D$ can be written as $r+1$ chunks: $x = \langle x_0, x_1, \ldots, x_r \rangle$ where each chunk is less than p (a modest restriction). Define another set A which contains all possible $(r+1)$-tuples with coordinates drawn from $\{0, \ldots, p-1\}$. We now define our hash family $\mathcal{H} = \{h_a : a \in A\}$ where $a = \langle a_0, a_1, \ldots, a_r \rangle$ and

$$h_a(x) = \sum_{i=0}^{r} a_i x_i \bmod p.$$

So we see that selecting an $(r+1)$-tuple from A in effect selects a hash function from $\mathcal{H}$.

For readers unfamiliar with Universal Hashing it can be quite disorienting to talk about a probability over hash functions rather than over points of the domain. For example, there is nothing in the definition that says that all hash functions in our family are good! Notice in the linear hash just given that one hash function is selected by $a = \langle 0, 0, \cdots, 0 \rangle$

which hashes every point to 0. This terrible hash function always produces collisions no matter what two initial points were chosen! But the fact is that only some small fraction of hash functions will cause a collision on two distinct points (it just so happens here that this bad hash function is always in this fraction). A stronger notion of universal hashing which attempts to make $h(a)$ and $h(b)$ uniform and pairwise independent random variables (and therefore must get rid of such bad hash functions as mentioned above) is called "Strongly Universal Hashing." We omit a discussion here since we have no need for it in this thesis.

The proof that the linear hash family is Universal is quite simple and we present the proof here since many later proofs will be strongly reminiscent.

Theorem 4.1.2 *The linear hash family $\mathcal{H}$ is Universal.*

Proof: Select any two distinct domain elements x and y. We rewrite these as $(r+1)$-tuples $x = \langle x_0, \cdots, x_r\rangle$ and $y = \langle y_0, \cdots, y_r\rangle$. We wish to show that the probability of a collision under a randomly-chosen hash function from $\mathcal{H}$ is $1/p$. For simplicity we write the random choice of hash function as a rather than h_a in the probability symbol. So we wish to prove that

$$\Pr_a[x_0a_0 + x_1a_1 + \cdots + x_ra_r = y_0a_0 + y_1a_1 + \cdots + y_ra_r \bmod p] = 1/p.$$

Since $x \neq y$ there must be some i such that $x_i \neq y_i$. Without loss of generality, assume $i = 0$. Then we rewrite the above as

$$\Pr_a[a_0(x_0 - y_0) = y_1 - x_1 + \cdots + y_r - x_r \bmod p] = 1/p.$$

Let's rewrite $(y_1 - x_1 + \cdots + y_r - x_r)$ as s. Now since $(x_0 - y_0) \neq 0$ and p is prime, we know $(x_0 - y_0)$ has an inverse, so we must simply prove that

$$\Pr_a[a_0 = s(x_0 - y_0)^{-1} \bmod p] = 1/p.$$

Note that $s(x_0 - y_0)^{-1}$ is a uniquely defined field element. And what is the probability that a randomly chosen $a \in A$ will have its first component equal to $s(x_0 - y_0)^{-1}$? Exactly $1/p$ of course. ▮

The linear hash is quite fast on most modern processors, but in the search for efficiency researchers have discovered that even faster methods are possible if we relax slightly the requirement that the collision probability be $1/|R|$. This led to the notion of Almost Universal Hash Families in which we allow the collision probability to be some $\epsilon \geq 1/|R|$.

Definition 4.1.3 Let $\epsilon \in \mathbf{R}^+$ be a positive number. Fix a domain D and range R. A finite multiset of hash functions $\mathcal{H} = \{h : D \rightarrow R\}$ is said to be **ϵ-Almost Universal** (ϵ-AU) if for every $x, y \in D$ where $x \neq y$, $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq \epsilon$. ▮

We will see several examples of ϵ-AU hash families in the next section.

It is interesting to note that in a sense a Universal hash family is ideal: one could not hope to get a lower collision rate than $1/|R|$. This is not immediately obvious (though perhaps intuitive). Let's prove this fact, which may give further insight as to how universal hashing works.

Proposition 4.1.4 *Fix a finite domain D and range R with $|D| > |R|$. If $\mathcal{H} = \{h : D \rightarrow R\}$ is ϵ-AU, then $\epsilon \geq 1/|R|$.*

Domain Elements	Range Elements 0	1	2	3	4	5
a	19	19	19	26	18	19
b	19	19	19	17	28	18
c	17	30	18	19	17	19
d	27	17	19	19	19	19
e	19	19	25	19	19	19
f	18	17	18	17	12	38
g	25	19	19	19	19	19
$\vdots$			$\vdots$			

Figure 4.1: A Example W Matrix with 120 hash functions. The (i, j)-th cell contains the number of hash functions h which have $h(i) = j$. For example there are 18 hash functions h with $h(c) = 2$. Since each row must sum to 120 and there are more rows than columns, we must have some column with two entries larger than $120/6 = 20$. Here that column is column 0.

Proof: Define a weighting function $w : D \times R \to \mathsf{Z}$. For any $x \in D$ and $y \in R$ we define $w(x, y)$ as the number of hash functions $h \in \mathcal{H}$ which cause $h(x) = y$. Now define a weighting matrix, W where $W(x, y) = w(x, y)$. This matrix will have $|D|$ rows and $|R|$ columns. (See Figure 4.1 for an example.) Note that the rows of W must sum to $|\mathcal{H}|$ by definition of $w(\cdot, \cdot)$. Therefore each row *must* have some entry which is at least $m = |\mathcal{H}|/|R|$. Since $|D| > |R|$ we know W has more rows than columns, so by the pigeonhole principle there must be some column j which has two entries at least m. Call these entries (s, j) and (t, j). Since $w(s, j) \geq m$ and $w(t, j) \geq m$, we simply fix the distinct domain points s and t and ask what is the probability that these collide under a randomly chosen $h \in \mathcal{H}$? But we know that $\Pr_h[h(s) = j] \geq m/|\mathcal{H}|$ and $\Pr_h[h(t) = j] \geq m/|\mathcal{H}|$ and so we must have that $\Pr_h[h(s) = h(t)] \geq m/|\mathcal{H}|$. But $m/|\mathcal{H}| = 1/|R|$ and we're done. ▌

Next we state a useful proposition which allows us to combine ϵ-AU hash families by appending their outputs. We will see several examples of ϵ-AU hash families in the next section.

Proposition 4.1.5 *If we have domain D and ranges R_1, R_2, and let $\mathcal{H}_1 = \{h : D \to R_1\}$ be an ϵ_1-AU hash family and let $\mathcal{H}_2 = \{h : D \to R_2\}$ be an ϵ_2-AU hash family, then we define $\mathcal{H}_1.\mathcal{H}_2 = \{h : D \to R_1 \times R_2\}$ where each $h \in \mathcal{H}_1.\mathcal{H}_2$ is an ordered pair (h_1, h_2) and for any $x \in D$ we have $(h_1, h_2)(x) = (h_1(x), h_2(x))$. Then $\mathcal{H}_1.\mathcal{H}_2$ is $\epsilon_1\epsilon_2$-AU.*

The proof is quite trivial but is included for completeness.

Proof: Take two distinct points $x, y \in D$. Then the probability over all $(h_1, h_2) \in \mathcal{H}_1.\mathcal{H}_2$ that $(h_1, h_2)(x) = (h_1, h_2)(y)$ is the same as the probability that $(h_1(x), h_2(x)) = (h_1(y), h_2(y))$. For this we need both $h_1(x) = h_1(y)$ and $h_2(x) = h_2(y)$. The probability of the first condition is ϵ_1 and of the second ϵ_2, so the probability of both occurring is $\epsilon_1\epsilon_2$.

▮

We conclude with yet another universal hash family variant: the Almost XOR Universal hash family.

Definition 4.1.6 Let $\epsilon \in \mathbf{R}^+$ be a positive number. Fix a domain D and range R. A finite multiset of hash functions $\mathcal{H} = \{h : D \to R\}$ is said to be **ϵ-Almost XOR Universal** (ϵ-AXU) if for every $x, y \in D$ where $x \neq y$, and every $c \in R$, $\Pr_{h \in \mathcal{H}}[h(x) \oplus h(y) = c] \leq \epsilon$. ▮

4.2 Some Examples of Fast-To-Compute Hash Families

Although the Wegman-Carter MAC was proposed in 1981, it was quite some time before cryptographers pursued their approach to invent fast MACs. The work by Krawczyk in 1994 [17] was the first in a series of efforts in this direction. Krawczyk described the

"Division Hash", a method akin to CRC computations which he demonstrated to be ϵ-AU for small epsilon. He also described a matrix-multiplication method called "LFSR-Based Toeplitz Hash." Both of these methods focus on exploiting a Linear Feedback Shift Register (LFSR) to allow efficient hardware implementations. We briefly discuss these now.

Division Hash.

We fix positive integers n and ℓ. A message M is broken into n chunks of ℓ-bits each. We then view M as a polynomial $m(x)$ with degree at most $n\ell$ over GF(2). Define the key set as $P = \{p(x) : p(x)$ has degree ℓ and is irreducible over GF(2)$\}$. (There are efficient methods to sample uniformly from P.) The family of hash functions is $\mathcal{H} = \{h : \{0,1\}^{\leq n\ell} \rightarrow \{0,1\}^{\ell}\}$ where a member is selected by the choice of $p(x) \in P$; we compute $h(x)$ for any message $m(x)$ as follows:

$$h(x) = m(x) \cdot x^{\ell} \bmod p(x).$$

Krawczyk shows this family is ϵ-AU for $\epsilon \approx n\ell/2^{\ell}$.

Krawczyk argues that this hash family can be efficiently implemented in hardware since polynomial division over GF(2) can be realized with an LFSR. He also claims that efficient software implementation is possible using some tables with precomputed values based on $p(x)$.

LFSR-Based Toeplitz Hash.

First we review another example of a Universal hash family from Carter and Wegman's original paper [9]. Fix parameters m and n. Let A be a random $m \times n$ binary matrix. Now we construction the family $\mathcal{H} = \{h : \{0,1\}^m \rightarrow \{0,1\}^n\}$ where a member of the family is specified by the choice of A. We compute $h(M)$ by treating the m-bit message M as a binary vector and computing $h(M) = AM$. That this family is Universal is a direct consequence of Theorem 4.1.2 and Proposition 4.1.5.

Krawczyk then notes that we can use a Toeplitz matrix in place of the random matrix A. A Toeplitz matrix has the property that each row is "shifted" once to the right as we read the matrix from top to bottom; the vacated entry in the leftmost column after the shift is filled with a new random value. Formally if $k - i = l - j$ for any $1 \leq i, k \leq n$ and $1 \leq j, l \leq m$, then $A(i, j) = A(k, l)$. It turns out that the family remains Universal even with this strong restriction on the matrix A. Krawczyk then further restricts the columns of A to be of states of an LFSR with a starting state representing the coefficients of a randomly-chosen irreducible polynomial of degree n over GF(2). By using this restricted class of matrices and by using a simple feedback protocol for the LFSR, he produces an ϵ-AU family with $\epsilon = m/2^{n-1}$, which is only slightly worse than the optimal $\epsilon = 1/2^n$ achieved by the Carter-Wegman scheme. Again hardware implementations are very efficient.

Bucket Hashing.

In 1995, Rogaway published his bucket hashing scheme [27]. This marked the first time researchers were trying to create software-efficient hash families to be used in MACs. Both this paper and Shoup's paper (mentioned next) focussed on not only providing fast hash families, but actually began publishing timing results on various platforms. We now describe Rogaway's bucket hash family.

We fix three positive integers: a word-size w, a block size n and a security parameter N (we will call N the "number of buckets"). To hash a message M we first break M

into n words of w bits each. So $M = M_1.M_2.\cdots.M_n$ with each $|M_i| = w$. Then we imagine N "buckets" (which are simply variables of w bits) into which we will XOR the words of M. For concreteness, we might have $w = 32$, $n = 1024$, and $N = 140$. For each word M_i of M we XOR M_i into three randomly chosen buckets. Finally we concatenate all the bucket contents as the output of the hash function. The only restriction on the buckets for any M_i is that they cannot be the same three buckets as were used for any M_j with $i \neq j$. Let's state all of this more formally.

Definition 4.2.1 Fix word-size w, block size n and security parameter N. We define the **bucket hash** family as $\mathcal{H} = \{h : \{0,1\}^{nw} \to \{0,1\}^{Nw}\}$ as follows. Let H be a randomly chosen n-vector with distinct coordinates, each coordinate being a 3-element set of w-bit words. We denote the ith coordinate of H as $H_i = \{h_{i1}\ h_{i2}\ h_{i3}\}$. For any $M \in \{0,1\}^{nw}$, let $M = M_1.M_2.\cdots.M_n$ with each $|M_i| = w$ then run the following algorithm:

bucket_hash(M)
  for $i \leftarrow 1$ **to** N **do** $Y_i \leftarrow 0^w$
  for $i \leftarrow 1$ **to** n **do**
    $Y_{h_{i1}} \leftarrow Y_{h_{i1}} \oplus M_i$
    $Y_{h_{i2}} \leftarrow Y_{h_{i2}} \oplus M_i$
    $Y_{h_{i3}} \leftarrow Y_{h_{i3}} \oplus M_i$
  return $Y_1.Y_2.\cdots.Y_N$

∎

The bucket hash family is very fast in software. Rogaway reports 6 instructions per word on a Silicon Graphics machine. This is about 1.5 instructions per byte, easily the fastest hash family ever invented at its time of publication. However, as Rogaway points out, bucket hashing has several drawbacks.

The timings reported above were measured for compressing a 1024-word message down to 140 words, providing a collision rate of $< 2^{-30}$. While this 7–to–1 compression ratio is sufficient to reduce the crytographic work to a point where a Carter-Wegman MAC will be fast, one certainly might hope for better compression. Also, since many messages could potentially be much shorter (as internet packets tend to be, for example), 1024 words might be viewed as atypically long. And finally, in order to get lower collision rates one needs many more buckets. For example to get to $\epsilon \approx 2^{-40}$ we need almost 400 buckets, resulting in quite long hash outputs.

Division Hash Variants.

In 1996 Shoup [28] published a few variants of the Division Hash family. Shoup published timing results on a Sparc-10 achieving speeds as fast as 7 instructions per byte once the preprocessing was done. Although not nearly as fast as bucket hashing, Shoup's families had shorter outputs and were therefore possibly more practical.

More than anything else, the mid-1990's marked the first time researchers were focussing on both producing efficient MACs with proven security *and* providing detailed experimental data to show that their algorithms really were fast on today's commodity processors.

Multilinear Modular Hashing.

In 1997, Halevi and Krawczyk introduced their Multilinear Modular Hash family (MMH) [14]. Their approach was to return to the simple linear hash (see Section 4.1), but instead of taking a sum mod p, they used a different series of moduli, giving up some small constant in the collision probability yet enhancing performance.

The MMH family is $\mathcal{H} = \{h : (\{0,1\}^{32})^n \to \{0,1\}^{32}\}$ where a member of this set is selected by some n-vector x with coordinates in $\{0,1\}^{32}$. For any message M taken as an n-vector with coordinates in $\{0,1\}^{32}$ we compute $h_x(M)$ as

$$\left[\left[\left[\sum_{i=1}^{n} M_i x_i\right] \bmod 2^{64}\right] \bmod (2^{32}+15)\right] \bmod 2^{32}$$

where x_i denotes the i-th coordinate of x and M_i the i-th coordinate of M. The authors prove that the MMH family is ϵ-AU for $\epsilon = 1.5/2^{30}$.

The significance of the MMH paper was manifold. First, it carried on the search for fast universal hash families giving both proofs of security and detailed experimental results. Second, it redirected attention back to the linear hash as a method for achieving extreme speed without much complexity. The specific innovation of MMH was the choice of moduli. The inner modulus by 2^{64} is implemented by simply using 64-bit registers to accumulate the sums while ignoring carries; the outer modulus by 2^{32} is also trivially implemented: merely ignore any bits beyond the least-significant 32. The middle modulus by $p = 2^{32} + 15$, a prime, is also efficiently implemented, but using a trick: first view any 64-bit number x as two 32-bit numbers a and b such that $x = a2^{32} + b$; now notice that $2^{32} = -15$ in the field Z_p, so $x \bmod p$ is the same as $b - 15a$. This means the middle modulus can computed without any divisions.

Halevi and Krawczyk report speeds of 820 Mbit/sec on a 150 Mhz PowerPC 604 and 1080 Mbit/sec on a 200 Mhz Pentium-Pro. They also give timings for a lower-collision variant of MMH ($\epsilon = 2.25/2^{60}$) which has about 60% of the throughput stated above. They do not report instructions/byte or cycles/byte, but later testing by Ted Krovetz yielded algorithms running at about 1.5-3 cycles/byte on a Pentium II.

Also mentioned in this paper is a family called NMH which is the natural adaptation of the authors' methods to a family created by Mark Wegman. NMH stands for "Nonlinear Modular Hashing," and is defined as $\mathcal{H} = \{g : (\{0,1\}^{32})^n \to \{0,1\}^{32}\}$ where a member of this set is selected by some n-vector x with coordinates in $\{0,1\}^{32}$. We assume here that n is even. For any message M taken as an n-vector with coordinates in $\{0,1\}^{32}$ we compute $g_x(M)$ as

$$\left[\left[\left[\sum_{i=1}^{n/2}(M_{2i-1} + x_{2i-1} \bmod 2^{32})(M_{2i} + x_{2i} \bmod 2^{32})\right] \bmod 2^{64}\right] \bmod (2^{32}+15)\right] \bmod 2^{32}$$

where again x_i denotes the i-th coordinate of x and M_i the i-th coordinate of M. The authors state that good bounds can be achieved for this family but do not provide any further details. An interesting feature of NMH, as compared to MMH, is that only half the total number of multiplications are needed. Since multiplications are more expensive than additions on most processors, NMH is often faster in practice.

4.3 From Hash to MAC

We have reviewed several fast hash families, but our overarching goal is to produce a MAC. In the Carter-Wegman MAC, the hash function does most of the work and consumes most of the time; therefore all of the papers summarized above presented a fast universal hash family but did not provide an actual MAC, assuming that this part of the project was either easy, uninteresting, or simply outside of the paper's scope (in fairness, [27] does provide a "toy" MAC, but admits there is much work left to be done to realize a fully-specified MAC).

In their original paper, Wegman and Carter [30] used perfect encryption to produce their MAC. Subsequently Brassard [7] suggested using a block cipher. We now review these methods and several other variants of these methods for producing a MAC given a universal hash family.

Perfect Encryption

Following the general approach of the original Wegman-Carter MAC, assume we have some ϵ-AXU hash family $\mathcal{H} = \{h : D \to \{0,1\}^b$ (see Definition 4.1.6), and we also have some infinite random bit string $P = P_1.P_2.\cdots$ with $|P_i| = b$. We choose a random $h \in \mathcal{H}$. The shared key between signer and verifier is (h, P). We describe the scheme WC[$\mathcal{H}$]. The signer has a counter, cnt, which is an integer variable, initially 0. To sign message M, the signer sends $(\text{cnt}, P_{\text{cnt}} \oplus h(M))$. To verify a received message M with tag (i, s), the verifier computes $P_i \oplus h(M)$ and ensures it equals s. We now state and prove a theorem on the security of this scheme.

Theorem 4.3.1 *Let $\mathcal{H}$ be ϵ-AXU. Then WC[$\mathcal{H}$] is ϵ-secure.*

Proof: Recall from Section 3.4 our definition of ϵ-secure: we choose a random h from $\mathcal{H}$ and random infinite bit string $P = P_1.P_2.\cdots$ with $|P_i| = b$. We instantiate the MAC oracle $\text{MAC}_{(h,P)}(\cdot)$ which MACs according to our scheme WC[$\mathcal{H}$]. We must show that there cannot exist an adversary E which makes q queries $Q = (M_1, M_2, \cdots, M_q)$ of MAC, in sequence, receiving tags $T = (t_1, t_2, \cdots, t_q)$ and then outputs a new message $M^* \notin Q$ along with a tag t^* and counter value i^* such that $P_i^* \oplus h(M^*) = t^*$ with probability more than ϵ.

We compute the probability that E forges after making q queries. Since E outputs a counter value i^*, there are two possibilities: either $i^* \leq q$ or $i^* > q$. We consider the latter first. If $i^* > q$ then P_{i^*} is a uniformly-distributed random bit string uncorrelated to any value yet seen, we know $P_i^* \oplus h(M^*) = t^*$ is also uniformly distributed and independent of what has been seen. So E's ability to guess t^* is exactly $1/2^b$.

Now consider the case that $i^* \leq q$. In other words, E has chosen a counter value which has been used in the computation of a tag t_{i^*} for a message M_{i^*}. She has both of these values. Now she must produce $M^* \neq M_{i^*}$ and t^* such that $t^* = h(M^*) \oplus P_{i^*}$. But this requires that $h(M^*) \oplus t^* = h(M_{i^*}) \oplus t_{i^*}$, or that $h(M^*) \oplus h(M_{i^*}) = t^* \oplus t_{i^*}$. But $\mathcal{H}$ is ϵ-AXU so the chance of this 0 is at most ϵ.

Theorem 4.1.4 tells us that $\epsilon \geq 1/2^b$, so in either case E's chance of forging is at most ϵ. ∎

The preceding is an “information theoretic” result. In other words, the adversary E could have unbounded computing power and still she could not succeed in producing a forgery with probability exceeding ϵ. But this MAC is obviously impractical since assuming that the sender and verifier share some random infinite-length string is unrealistic. To make this scheme practical, we might settle for pseudorandom bits instead of truly random bits, and use a PRF (see Section 2.2) as a source for these pseudorandom bits. Since a PRF is keyed, the key will provide us with a compact description of the bit stream. To generate our pseudorandom bits given this key, we will simply feed a counter to the PRF as input. Of course we must take care not to allow counter values to repeat or our pseudorandom bits will suddenly become very predictable! We now describe our scheme more formally.

Let $F : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^L$ be a (t, q, δ)-secure PRF. Let $\mathcal{H} = \{h : \{0,1\}^* \to$

$\{0,1\}^L\}$ be a family of hash functions. We define the scheme WC[$\mathcal{H}, F$] as follows: the shared secret key is some randomly-chosen string $a \in \{0,1\}^\kappa$ (the PRF key), and some randomly-chosen hash function $h \in \mathcal{H}$. The MAC generation algorithm is stateful; it maintains a counter cnt which starts at -1. To authenticate a message M, the MAC generation algorithm first increments cnt and ensures cnt $< 2^L - 1$. If this test fails, we refuse to MAC and instead return REKEY indicating that the signer needs to re-key. Otherwise we generate the MAC tag $(\mathrm{cnt}, F_a(\mathrm{cnt}) \oplus h(M))$. To verify a received message M with tag (i, s), the verifier computes $F_a(i) \oplus h(M)$ and ensures it equals s. We now, following [3], state and prove a theorem on the security of this scheme. (See Section 3.4 for the definition of security for a MAC.)

We now discuss the complexity-theoretic version of the $WC[\mathcal{H}, F]$ construction. In order to do so, we must introduce a computational framework. It is standard to assume our computations are performed on a Turing machine, with each step accounting for one unit of time. The description of the machine will be some polynomial in t, the total running time.

Theorem 4.3.2 *Let $\mathcal{H}$ be ϵ-AXU and let* INIT *denote the time required to select a random $h \in \mathcal{H}$. Let* HASH(q) *denote the time required to hash q strings. Let $F : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^L$ be a $\epsilon'(t,q)$-secure PRF. Let E be an adversary running in time at most t, making at most q queries, the length of these queries totalling at most μ bits. Then E forges with probability at most $\epsilon + \epsilon'(t + \textsc{Init} + \textsc{Hash}(q+1) + O(q), q+1)$.*

Proof: To simplify notation, we will denote by δ the quantity $\epsilon'(t+O(q)+\textsc{Init}+\textsc{Hash}(q+1), q+1)$. To begin our proof, we will assume we have an adversary E which attacks our

MAC in time at most t, making at most q queries totalling at most μ bits; we will not make any assumption about the best advantage achievable by E. We will then build a new adversary D which attacks our PRF F using E as a subroutine. Analyzing the advantage of D will then yield our desired bound on E's advantage.

We now describe adversary D. Recall the setting such an adversary runs in: D is presented with an oracle $\mathcal{O}$ which is either the pseudorandom oracle $F(\cdot)$ or a truly-random oracle $\rho(\cdot)$. After some amount of computation, including subroutine calls to E and various calls to this oracle, D outputs either a "1" meaning D believe the oracle is pseudorandom, or "0" meaning D believes the oracle is truly-random.

D runs as follows:

1. Initially, D chooses a random $h \in \mathcal{H}$.
2. We set a counter cnt to -1.
3. Next D runs our MAC adversary E then waits.
4. Eventually E makes a signing query, asking for the MAC of some message M_1. For this and all messages M_i which E wishes to MAC, we increment cnt and return the string $\mathcal{O}(\mathrm{cnt}) \oplus h(M_i)$.
5. After at most q of these oracle queries, E will output some message M^* along with a tag (i, s). This is E's attempt at a forgery.
6. We check if M^* is new and if $s = \mathcal{O}(i) \oplus h(M^*)$; if yes, we output "1", else we output "0."

Before continuing, let's count up the resources used by adversary D. In step 1 we use $\textsc{Init}$ to choose a hash function. Then running E costs us time at most t since E runs in time at most t. Of course E is requesting MAC tags at most q times and this causes us to hash at most $q+1$ times (steps 4 and 6) with a total cost of $\textsc{Hash}(q+1)$. The cost of step 4 is $O(q)$ since it is run at most q times. So our total time is $t+\textsc{Init}+\textsc{Hash}(q+1)+O(q)$. How many oracle queries does D make? At most q in step 4 and one more in step 6, so at most $q+1$. Therefore the maximal advantage obtainable by adversary D is $\epsilon'(t+\textsc{Init}+\textsc{Hash}(q+1)+O(q), q+1)$ which is δ.

Now we return to the problem of computing the forgery probability of adversary E, the MAC adversary. We approach this by writing down the formula for advantage of D:

$$\Pr[K \leftarrow \{0,1\}^\kappa :\ D^{F_K(\cdot)} = 1] - \Pr[\rho \leftarrow \mathrm{Rand}(\ell, L) :\ D^{\rho(\cdot)} = 1] = \delta.$$

To evaluate each of these two probabilities, we first need to understand the operation of D. Clearly D is simulating the MAC's operation, since it chooses a hash function and then builds a MAC tag with its oracle. It then guesses that $\mathcal{O}$ is pseudorandom if its E adversary forges. So if $\mathcal{O}$ is pseudorandom (ie, the 1 probability above) is exactly the same as the probability that E will output a correct forgery. On the other hand, if $\mathcal{O}$ is truly-random (the 1 probability above), D will output 1 if E successfully forges, and here we know from Theorem 4.3.1 that this occurs with probability at most ϵ. With these facts in mind, we can rewrite the above as

$$\Pr[E \text{ forges under } \mathrm{WC}(\mathcal{H}, F)] - \Pr[E \text{ forges under } \mathrm{WC}(\mathcal{H})] = \delta$$

or

$$\Pr[E \text{ forges under } \mathrm{WC}(\mathcal{H}, F)] \le \epsilon + \delta$$

which completes the proof. ▮

Applying a PRF to the Hash Output. Another approach to building a MAC in the Carter-Wegman style is to apply a PRF directly to the output of a universal hash family. Since, in practice, we have PRFs which accept arbitrarily long inputs, this approach is sometimes more flexible since our hash families will sometimes be *compression* functions rather than true hash functions (a "hash function" is usually a function with a fixed-size output). Another advantage of this particular scheme is that it is stateless (ie, there are no counters the signer must keep track of).

This time we need only require our hash family be ϵ-AU (instead of ϵ-AXU in the foregoing scheme). The shared key between signer and verifier is (h, ρ) where $h \in \mathcal{H} = \{h : D \to \{0,1\}^\ell\}$ and ρ is a randomly-chosen function from $\mathcal{R} = \mathrm{Rand}(\ell, L)$. We describe the scheme $\mathrm{FH}[\mathcal{H}, \mathcal{R}]$. We first choose a random $h \in \mathcal{H}$. We fix some number $q > 0$; we agree to MAC at most q messages and then the scheme refuses to MAC any further, outputting REKEY until new keys are chosen. To sign message M, the signer first ensures we've signed fewer than q messages and if so sends $\rho(h(M))$. To verify a received message M with tag s, the verifier computes $\rho(h(M))$ and ensures it equals s. We now state and prove a theorem on the security of this scheme.

Theorem 4.3.3 *Let $\mathcal{H}$ be ϵ-AU and let $\mathcal{R} = \mathrm{Rand}(\ell, L)$. Fix some number $q > 0$, the limit on the number of messages we will MAC. Then FH$[\mathcal{H}, \mathcal{R}]$ is $\epsilon q(q+1)/2$-secure.*

Proof: Suppose that an adversary E makes q queries $Q = (M_1, M_2, \cdots, M_q)$ of the MAC oracle, receiving tags $T = (t_1, t_2, \cdots, t_q)$ and then outputs, with some probability, a new message $M^* \notin Q$ along with a tag t^* such that $\rho(h(M^*)) = t^*$ where $h \xleftarrow{R} \mathcal{H}$ and $\rho \xleftarrow{R} \mathcal{R}$.

We compute the probability that E forges. We begin by examining the hash values produced from E's queries and her attempted forgery; consider the multiset

$$H = \{h(M_1), h(M_2), \cdots, h(M_q), h(M^*)\}.$$

Now we condition on the event that all the members of H are distinct; if there are duplicates in H we say there has been a "collision in the hash function."

First consider the case where there has been no collision in the hash function. Since $h(M^*)$ is distinct from all other members of H, computing $\rho(h(M^*))$ will involve computing $\rho()$ at a domain point which has never been sampled before by E. Therefore the probability that $\rho(h(M^*)) = t^*$ is exactly $1/2^L$.

Next consider the chance that there was a collision in the hash function. We don't know necessarily how E might use this to her advantage, but certainly *some* information about the hash function is being given away here. So we will count *any* collision in the hash function as a bad event (depending on the PRF used, there is often an attack which can exploit collisions in the hash function as we did in Section 3.8). We now compute the probability that a collision occurs in the hash function.

Let C_i be the event that a collision occurs for the first time on the i-th query. We know $\Pr[C_1] = 0$. In general, $\Pr[C_i] \leq (i-1)\epsilon$ since there are $i-1$ distinct values to collide with and at most an ϵ chance of hitting any of them. Therefore our bound is

$$\Pr[\text{collision in hash function}] \leq \sum_{i=1}^{q+1} C_i \leq \sum_{i=1}^{q+1} (i-1)\epsilon = \epsilon \sum_{i=0}^{q} i = \epsilon q(q+1)/2.$$

Since $\epsilon q(q+1)/2 > \epsilon \geq 1/2^L$, in either case above E's chance of forging is at most $\epsilon q(q+1)/2$. ∎

We can once again pass from the information-theoretic case above (where we had access to a truly-random function) to the complexity-theoretic case (where we replace such a truly-random function with a PRF). However, it is quite analogous to the previous case and our bound, once again, involves simply adding the term $\epsilon'(t+\textsc{Init}+\textsc{Hash}(q+1)+O(q), q+1)$. We state the full theorem here, but omit the proof.

Theorem 4.3.4 *Let $\mathcal{H}$ be ϵ-AU and let* Init *denote the time required to select a random $h \in \mathcal{H}$. Let* $\textsc{Hash}(q)$ *denote the time required to hash q strings. Let $F : \{0,1\}^\kappa \times \{0,1\}^\ell \rightarrow \{0,1\}^L$ be a $\epsilon'(t,q)$-secure PRF. Let E be a FH[$\mathcal{H}, F$] MAC adversary running in time at most t, making at most q queries, the length of these queries totalling at most μ bits. Then E forges with probability at most $\epsilon q(q+1)/2 + \epsilon'(t + \textsc{Init} + \textsc{Hash}(q+1) + O(q), q+1)$.*

We obviously have worsened our bound quite a bit by going from the first construction where we had an information-theoretic bound of ϵ to the latter construction where our bound rose to $\epsilon q(q+1)/2$. The problem was that we were vulnerable to any collision in the hash functions. The next construction fixes this problem.

PRF applied to a Counter and the Hash Output. We can ensure that the input to our PRF is distinct every time by employing a counter. This scheme will therefore once again be stateful.

Again we have some ϵ-AU hash family $\mathcal{H} = \{h : D \rightarrow \{0,1\}^\ell\}$, and a set of functions $\mathcal{R} = \text{Rand}(\ell + b, L)$. We choose a random $h \in \mathcal{H}$ and a random $\rho \in \mathcal{R}$. The shared key between signer and verifier is (h, ρ). We describe the scheme FCH$[\mathcal{H}, \mathcal{R}]$. The signer has a counter, cnt, which is an integer variable, initially 0. To sign message M, the signer first ensures that $\text{cnt} < 2^b - 1$ and if so increments cnt and then sends $(\text{cnt}, \rho(\langle \text{cnt} \rangle_b \parallel h(M))$. To verify a received message M with tag (i, s), the verifier computes $\rho(\langle i \rangle_b \parallel h(M))$ and ensures it equals s. We now state and prove a theorem on the security of this scheme.

Theorem 4.3.5 *Let $\mathcal{H}$ be ϵ-AU and let $\mathcal{R} = \text{Rand}(\ell + b, L)$. Then FCH$[\mathcal{H}, \mathcal{R}]$ is $(\epsilon + 1/2^L)$-secure.*

Proof: The proof is strongly reminiscent of our last stateful scheme.

Once again we must show that there cannot exist an adversary E which makes q queries $Q = (M_1, M_2, \cdots, M_q)$ of the MAC oracle receiving tags $T = (t_1, t_2, \cdots, t_q)$ and then outputs a new message $M^* \notin Q$ along with a tag t^* and counter value i^* such that $\rho(\langle i^* \rangle_b \parallel h(M^*)) = t^*$ where $h \overset{R}{\leftarrow} \mathcal{H}$ and $\rho \overset{R}{\leftarrow} \mathcal{R}$.

We compute the probability that E forges after making q queries. Since E outputs a counter value i^*, there are two possibilities: either $i^* \leq q$ or $i^* > q$. We consider the latter first. If $i^* > q$ then $\langle i^* \rangle_b \parallel h(M^*)$ is domain point of ρ which has never been sampled before (we know this because our counter value has a fixed-length encoding, namely, b bits). Therefore $\rho(\langle i^* \rangle_b \parallel h(M^*))$ is a uniformly-distributed random string uncorrelated to any value yet seen. So E's ability to guess t^* is exactly $1/2^L$.

Now consider the case that $i^* \leq q$. She must produce $M^* \neq M_{i^*}$ and t^* such that $t^* = \rho(\langle i^* \rangle_b \parallel h(M^*))$. How can this happen? There are two ways: either $h(M_{i^*}) = h(M^*)$ or this did not happen and yet $\rho(\langle i^* \rangle_b \parallel h(M_{i^*})) = \rho(\langle i^* \rangle_b \parallel h(M^*))$. We might call these cases a "collision in h" and a "collision in ρ," respectively. We will upper-bound the overall probability of forgery in the present case by adding the probabilities of each of these. We know the probability of a collision in h to be 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 $M_1, \ldots, M_q$. These values, $t_1, \ldots, t_q$, are random L-bit strings; the adversary E would be provided an identical distribution of views if the oracle were replaced by one which, in response to a query, returned a random L-bit string.

The probability of a collision in ρ is simply $1/2^L$. So our overall bound is $\epsilon + 1/2^L$. Since $\epsilon + 1/2^L > 1/2^L$, we have $\epsilon + 1/2^L$ as an upper bound in both cases. ▮

As before we may pass to the complexity-theoretic result by adding the appropriate term.

Theorem 4.3.6 *Let $\mathcal{H}$ be ϵ-AU and let* INIT *denote the time required to select a random $h \in \mathcal{H}$. Let* HASH(q) *denote the time required to hash q strings. Fix a $b > 1$. Let $F : \{0,1\}^\kappa \times \{0,1\}^{\ell+b} \to \{0,1\}^L$ be a $\epsilon'(t,q)$-secure PRF. Let E be a FCH$[\mathcal{H}, F]$ MAC adversary running in time at most t, making at most q queries, the length of these queries totalling at most μ bits. Then E forges with probability at most $\epsilon + 1/2^L + \epsilon'(t +$* INIT $+$ HASH$(q+1) + O(q), q+1)$.

Chapter 5

Protected Counter Sums

Many times we can view a MAC in the Carter-Wegman spirit, even when the designers did not view it this way themselves. The advantage is that often the analysis becomes much simpler. We begin with a simple example in this chapter, viewing Berstein's "Protected Counter Sum" construction [5] as a Carter-Wegman MAC, and then proceed on in the following chapter to view a variant of the CBC MAC this way as well.

5.1 Protected Counter Sums

Recall the definition of the XOR MAC from Section 3.7: we have a random function $\rho : \{0,1\}^\ell \to \{0,1\}^L$. We require that ℓ is an even number. If M is the message we wish to MAC, we divide M into $\ell/2$-bit blocks: $M = M_1.M_2.\cdots.M_n$ where $|M_i| = \ell/2$. Let $\langle i \rangle$ denote an $\ell/2 - 1$ bit encoding of the integer i. (So we require $i \leq 2^{\ell/2-1}$.) Then the XOR MAC of M is

$$t = \rho(0.r) \oplus \rho(1.\langle 1 \rangle.M_1) \oplus \rho(1.\langle 2 \rangle.M_2) \oplus \cdots \rho(1.\langle n \rangle.M_n)$$

where r is a random $\ell - 1$ bit string.

Bernstein's construction is based on XOR MAC. His Protected Counter Sum (PCS) construction works as follows: let $\rho : \{0,1\}^\ell \to \{0,1\}^L$ be a random function with $\ell > L$ and $c = \ell - L$. If M is the message we wish to MAC, we divide M into L-bit blocks: $M = M_1.M_2.\cdots.M_n$ where $|M_i| = L$ and $n < 2^c$. Let $\langle i \rangle$ denote an c-bit encoding of the integer i. Then the PCS MAC of M is

$$t = \rho(\langle 0 \rangle \ . \ (\rho(\langle 1 \rangle.M_1) \oplus \rho(\langle 2 \rangle.M_2) \oplus \ \cdots \ \oplus \rho(\langle n \rangle.M_n)).$$

Although Bernstein's analysis takes a very different approach, we might notice that his construction strongly resembles the Carter-Wegman type MACs we've been discussing. To make the resemblance even stronger we will recast PCS MAC in a slightly different (but equivalent) form. Instead of applying ρ with $\langle 0 \rangle$ prepended to the inner XORs, we will instead apply some different, independently chosen, random function ρ'. This scheme, which we call FCS MAC will be viewed as a PRF applied to the hash result (see Section 4.3).

First we describe a hash family we will call the "XOR hash." Fix some positive number $c < \ell$, the "counter size." Let the hash domain $D = (\{0,1\}^{\ell - c})^{<2^c}$. Let $\mathcal{H} = \{h : D \to \{0,1\}^\ell\}$ be the family of functions where members are selected by the random choice of some $\rho \in \mathrm{Rand}(\ell, L)$. For any $M \in D$ we break M into $(\ell - c)$-bit chunks $M_1.M_2.\cdots.M_n$. We then compute $h(M)$ as

$$h(M) = \rho(\langle 1 \rangle.M_1) \oplus \rho(\langle 2 \rangle.M_2) \oplus \ \cdots \ \oplus \rho(\langle n \rangle.M_n)$$

where again $\langle i \rangle$ denotes some c-bit encoding of i.

We now instantiate FCS MAC as $\mathrm{FH}[\mathcal{H}, \mathrm{Rand}(\ell, L)]$. The security of this scheme is, as we have already seen, $\epsilon q(q+1)/2$. Of course we would like to know what kind of value we can prove for ϵ here, which is the next order of business. Although we require only that the XOR hash be ϵ-AU, we will prove that it is ϵ-AXU which clearly implies the former.

Theorem 5.1.1 *Fix some $\ell \geq 1$ and some positive $c < \ell$. Then the XOR hash on domain $D = (\{0,1\}^{\ell-c})^{<2^c}$ is $2^{-\ell}$-AXU.*

Proof: We are required to show that for any two distinct values $M, M' \in D$ and constant $a \in \{0,1\}^\ell$, the probability over uniform random choices of functions $\rho \in \mathrm{Rand}(\ell,\ell)$ that the XOR hashes of M and M' XOR to a is at most $2^{-\ell}$. In other words we must prove that for any $a \in \{0,1\}^\ell$,

$$\Pr_{\rho \in \mathrm{Rand}(\ell,\ell)}[(\rho(\langle 1\rangle.M_1) \oplus \cdots \oplus \rho(\langle n\rangle.M_n)) \oplus (\rho(\langle 1\rangle.M'_1) \oplus \cdots \oplus \rho(\langle n\rangle.M'_{n'})) = a] \leq 2^{-\ell}.$$

We know M and M' are distinct; we now consider two cases: (1) one of these messages is a prefix of the other, or (2) neither is a prefix of the other.

Case 1: Without loss of generality, assume that M' is a prefix of M. This means that $M_1 = M'_1$, $M_2 = M'_2$, $\cdots$, $M_{n'} = M'_{n'}$. Since the XOR of a value with itself is 0, we may cancel terms and rewrite the above as

$$\Pr_{\rho \in \mathrm{Rand}(\ell,\ell)}[(\rho(\langle n'+1\rangle.M_{n'+1}) \oplus \cdots \oplus \rho(\langle n\rangle.M_n)) = a] \leq 2^{-\ell}.$$

Now imagine we choose $\rho \in \mathrm{Rand}(\ell,\ell)$ in two phases: we first determine the value of ρ at all domain points *except* $\langle n\rangle \,.\, M_n$. Then in the second phase we choose the value of $\rho(\langle n\rangle \,.\, M_n)$. It should be clear that we are not at all constrained in the second phase by choices made in the first phase because the counter values ensure that our second-phase input has not yet been determined. Therefore we need only prove

$$\Pr[\rho(\langle n\rangle.M_n) = a \oplus \rho(\langle n'+1\rangle.M_{n'+1}) \oplus \cdots \oplus \rho(\langle n-1\rangle.M'_{n-1}))] \leq 2^{-\ell}$$

where the probability is taken over random choices of $\rho(\langle n\rangle \,.\, M_n)$, and this will imply the theorem. But there is of course only one value which makes the equality true, that being the value on the right-hand side of the equation. Therefore the above probability is indeed $2^{-\ell}$.

Case 2: This case is similar to the above, so we'll be a bit more terse. Since M and M' are distinct, they must differ in some $(\ell - c)$-bit block; without loss of generality, assume $M_1 \neq M'_1$. (This argument could be repeated for any i where $M_i \neq M'_i$.) Again we determine $\rho(\cdot)$ in two phases: first the value of $\rho(\cdot)$ at all points except $\langle 1\rangle \,.\, M_1$ and $\langle 1\rangle \,.\, M'_1$ are determined, then finally we determine $\rho(\langle 1\rangle \,.\, M_1)$ and $\rho(\langle 1\rangle \,.\, M'_1)$. Rearranging terms, we note we now need only prove that

$$\Pr[\rho(\langle 1\rangle.M_1) \oplus \rho(\langle 1\rangle.M'_1) = a \oplus \rho(\langle 2\rangle.M_2) \oplus \cdots \oplus \rho(\langle n\rangle.M'_n))] \leq 2^{-\ell}$$

where the probability is taken over the second-phase choices. Since $M_1 \neq M'_1$ we know $\langle 1\rangle \,.\, M_1 \neq \langle 1\rangle \,.\, M'_1$ and so these points are going to be two independent random values between 0 and $2^\ell - 1$. Therefore the probability that they XOR to some particular value is exactly $2^{-\ell}$. ∎

Given the preceding theorem, we may now invoke Theorem 4.3.3 to conclude that FCS MAC is $q(q+1)/2^{\ell+1}$-secure.

Chapter 6

CBC MAC Variants

As noted in the previous chapter, we can often view an existing MAC as a Carter-Wegman MAC in spite of the fact it may not have been designed as one. In this chapter we will consider variants of the CBC MAC as Carter-Wegman MACs. This will make the analysis easier than it has been when considered from other viewpoints [3, 22], and it will allow us to extend the capabilities of the MAC to efficiently handle messages of any length (rather restrict us to only those messages whose lengths are a multiple of the block length).

6.1 Introduction

Recall from Section 3.7 the definition of the CBC MAC. Let $E : \mathsf{Key} \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher (see Section 2.2): it uses a key $K \in$ Key to encipher an n-bit block X into an n-bit ciphertext $Y = E_K(X)$. The message space for the CBC MAC is $(\{0,1\}^n)^+$, meaning binary strings whose lengths are a positive multiple of n. So let $M = M_1 \cdots M_m$ be a string that we want to MAC, where $|M_1| = \cdots = |M_m| = n$. Then

$\mathrm{CBC}_{E_K}(M)$, the CBC MAC of M under key K, is defined as C_m, where $C_i = E_K(M_i \oplus C_{i-1})$ for $i = 1, \ldots, m$ and $C_0 = 0^n$.

Bellare, Kilian and Rogaway proved the security of the CBC MAC, in the sense of reduction-based cryptography [3]. But their proof depends on the assumption that it is only messages of one fixed length, mn bits, that are being MACed. Indeed when message lengths can vary the CBC MAC is *not* secure. This fact is well-known. As a simple example, notice that given the CBC MAC of a one-block message X, say $T = \mathrm{CBC}_{E_K}(X)$, the adversary immediately knows the CBC MAC for the two-block message $X \,.\, (X \oplus T)$, since this is once again T.

Thus the CBC MAC (in the "raw" form that we have described) has two problems: it can't be used to MAC messages outside of $(\{0,1\}^n)^+$, and all messages must have the same fixed length.

Dealing with variable message lengths: EMAC. When message lengths may vary the CBC MAC must be embellished. There have been several suggestions for doing this. The most elegant one we have seen is to encipher $\mathrm{CBC}_{E_{K1}}(M)$ using a new key, $K2$. That is, the domain is still $(\{0,1\}^n)^+$ but one defines EMAC (for encrypted MAC) by $\mathrm{EMAC}_{E_{K1},E_{K2}}(M) = E_{K2}(\mathrm{CBC}_{E_{K1}}(M))$. This algorithm was developed for the RACE project [4]. It has been analyzed by Petrank and Rackoff [22] who show, roughly said, that an adversary who obtains the MACs for messages which total σ blocks cannot forge with probability better than $2\sigma^2/2^n$.

Among the nice features of EMAC is that one need not know $|M|$ prior to processing the message M. We now proceed to develop several variants of EMAC, all of which will preserve this feature.

6.1.1 Outline of Chapter

EMAC has a domain limited to $(\{0,1\}^n)^+$ and uses $1 + |M|/n$ applications of the block cipher E. In this chapter we refine EMAC in three ways: (1) we extend the domain to $\{0,1\}^*$; (2) we shave off one application of E; and (3) we avoid keying E by multiple keys. Of course we insist on retaining provable security (across all messages lengths).

In Section 6.2 we introduce three refinements to EMAC, which we call ECBC, FCBC, and XCBC. These algorithms are natural extensions of the basic CBC MAC. We would like to think that this is an asset. The point here is to strive for economy, in terms of both simplicity and efficiency.

Figure 6.1.1 summarizes the characteristics of the CBC MAC variants mentioned in this chapter. The top three rows give known constructions (two of which we have now defined). The next three rows are our new constructions. Note that our last construction, XCBC, retains essentially all the efficiency characteristics of the CBC MAC, but extends the domain of correct operation to all of $\{0,1\}^*$.

For each of the new schemes we give a proof of security. Rather than adapt the rather complex proof of [22], or the even more complicated one of [3], we follow a new tack: we view EMAC as an instance of the Carter-Wegman paradigm: with EMAC one is enciphering the output of a universal-2 hash function. This universal-2 hash function is the CBC MAC itself. Since it is not too hard to upper bound the collision probability of the CBC MAC (see Lemma 6.3.2), this approach leads to a simple proof for EMAC, and ECBC as well. We then use the security of ECBC to prove security for FCBC, and then we use

Construct	Domain	$\#E$ Appls	$\#E$ Keys	Key Length
CBC	$\{0,1\}^{nm}$	$\|M\|/n$	1	k
EMAC	$(\{0,1\}^n)^+$	$1+\|M\|/n$	2	$2k$
EMAC*	$\{0,1\}^*$	$1+\lceil(\|M\|+1)/n\rceil$	2	$2k$
ECBC	$\{0,1\}^*$	$1+\lceil\|M\|/n\rceil$	3	$3k$
FCBC	$\{0,1\}^*$	$\lceil\|M\|/n\rceil$	3	$3k$
XCBC	$\{0,1\}^*$	$\lceil\|M\|/n\rceil$	1	$k+2n$

Figure 6.1: *The CBC MAC and five variants. Here M is the message to MAC and E : $\{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ is a block cipher. The third column gives the number of applications of E, assuming $|M| > 0$. The fourth column is the number of different keys used to key E. For* CBC *the domain is actually $(\{0,1\}^n)^+$, but the scheme is secure only on messages of some fixed length, nm.*

the security of FCBC to prove security for XCBC. In passing from FCBC to XCBC we use a general lemma (Lemma 6.5.1) which says, in effect, that you can always replace a pair of random independent permutations $\pi_1(\cdot), \pi_2(\cdot)$ by a pair of functions $\pi(\cdot), \pi(\cdot \oplus K)$, where π is a random permutation and K is a random constant.

We directly prove the security of ECBC. For this we view EMAC as an instance of the Carter-Wegman paradigm: with EMAC one is enciphering the output of a universal-2 hash function. This universal hash function is the (basic) CBC MAC. Under this viewpoint the bulk of proving security is to analyze the collision probability of the CBC MAC: that is, what is the chance, for distinct messages M and M', that $\mathrm{CBC}_\pi(M) = \mathrm{CBC}_\pi(M')$? This is a simple, information-theoretic question. In particular, the approach effectively eliminates the complexity associated with the adversary's adaptivity.

The same method used for ECBC gives a simple proof for EMAC. We include our proof of EMAC in Section 6.3. Petrank and Rackoff's proof was quite complex.

The proof again demonstrates that the adversary's adaptivity does not help her very much in this setting.

New standards. This work was largely motivated by the emergence of the Advanced Encryption Standard (AES). With the AES should come a next-generation standard for using it to MAC. Current CBC MAC standards handle this, for example, in an open-ended informational appendix [15]. We suggest that the case of variable message lengths is the *usual* case in applications of MACs, and that a modern MAC standard should specify only algorithms which will correctly MAC any sequence of bit strings. The methods here are simple, efficient, provably sound, timely, and patent-free—all the right features for a contemporary standard.

6.2 Schemes ECBC, FCBC, and XCBC

Arbitrary-length messages without obligatory padding: ECBC. We have described the algorithm $\mathrm{EMAC}_{E_{K1},E_{K2}}(M) = E_{K2}(\mathrm{CBC}_{E_{K1}}(M))$. One problem with EMAC is that its domain is limited to $(\{0,1\}^n)^+$. What if we want to MAC messages whose lengths are *not* a multiple of n?

The simplest approach is to use obligatory 10^i padding: always append a "1" bit and then the minimum number of "0" bits (possibly none) so as to make the length of the padded message a multiple of n. Then apply EMAC. We call this method EMAC*. Formally, $\mathrm{EMAC}^*_{E_{K1},E_{K2}}(M) = \mathrm{EMAC}_{E_{K1},E_{K2}}(M \,.\, 10^{n-1-|M| \bmod n})$. This construction works fine. In fact, it is easy to see that this form of padding *always* works to extend the domain of a MAC from $(\{0,1\}^n)^+$ to $\{0,1\}^*$.

One unfortunate feature of EMAC* is this: if $|M|$ is already a multiple of n then we are appending an entire extra block of padding, and seemingly "wasting" an application

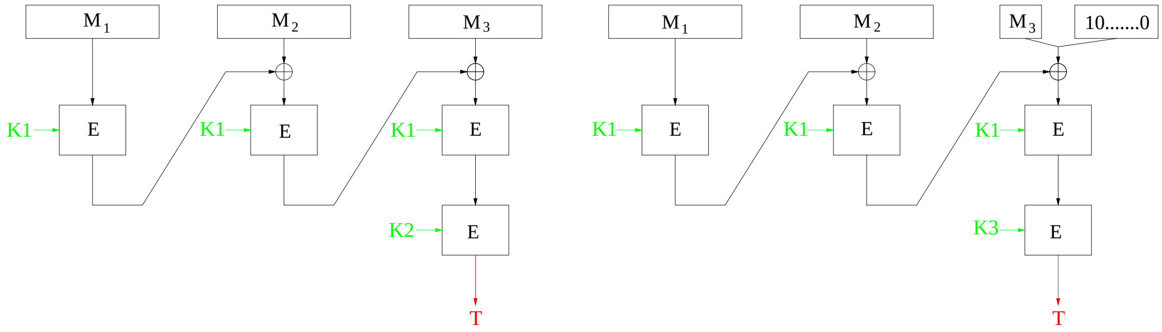

Figure 6.2: *The* ECBC *construction using a block cipher* $E : \mathsf{Key} \times \{0,1\}^n \to \{0,1\}^n$. *The construction uses three keys,* $K1, K2, K3 \in \mathsf{Key}$. *On the left is the case where* $|M|$ *is a positive multiple of* n, *while on the right is the case where* $|M|$ *is not a positive multiple of* n.

of E. People have worked hard to optimize new block ciphers—it seems a shame to squander some of this efficiency with an unnecessary application of E. Moreover, in practical settings we often wish to MAC very short messages, where saving one invocation of the block cipher can be a significant performance gain. In some domains messages of exactly 8 or 16 to 64 bytes may be commonplace.

Our first new scheme lets us avoid padding when $|M|$ is a nonzero multiple of n. We simply make two cases: one for when $|M|$ is a positive multiple of n and one for when it isn't. In the first case we compute $\mathrm{EMAC}_{E_{K1},E_{K2}}(M)$. In the second case we append minimal 10^i padding ($i \geq 0$) to make a padded message P whose length is divisible by n, and then we compute $\mathrm{EMAC}_{E_{K1},E_{K3}}(P)$. Notice the different second key—$K3$ instead of $K2$—in the case where we've added padding. Here, in full, is the algorithm. It is also shown in Figure 6.2.

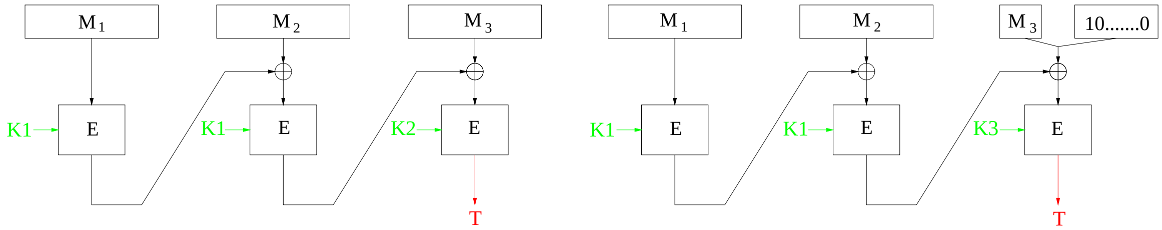

Figure 6.3: *The* FCBC *construction with a block cipher* $E : \mathsf{Key} \times \{0,1\}^n \to \{0,1\}^n$. *The construction uses three keys,* $K1, K2, K3 \in \mathsf{Key}$. *On the left is the case where* $|M|$ *is a positive multiple of* n, *while on the right is the case where* $|M|$ *is not a positive multiple of* n.

$$
\begin{array}{l}
\textbf{Algorithm} \;\; \mathrm{ECBC}_{E_{K1},E_{K2},E_{K3}}(M) \\
\textbf{if} \;\; M \in (\{0,1\}^n)^+ \\
\qquad \textbf{then return} \;\; E_{K2}(\mathrm{CBC}_{E_{K1}}(M)) \\
\qquad \textbf{else return} \;\; E_{K3}(\mathrm{CBC}_{E_{K1}}(M\,.\,10^i)), \text{ where } i = n - 1 - |M| \bmod n
\end{array}
$$

In Section 6.3 we prove that ECBC is secure. We actually show that it is a good pseudo-random function (PRF), not just a good MAC. The security of ECBC does not seem to directly follow from Petrank and Rackoff's result [22]. At issue is the fact that there is a relationship between the key $(K1, K2)$ used to MAC messages in $(\{0,1\}^n)^+$ and the key $(K1, K3)$ used to MAC other messages.

Improving efficiency: FCBC. With ECBC we are using $\lceil |M|/n \rceil + 1$ applications of the underlying block cipher. We can get rid of the $+1$ (except when $|M| = 0$). We start off, as before, by padding M when it is outside $(\{0,1\}^n)^+$. Next we compute the CBC MAC using key $K1$ for all but the final block, and then use either key $K2$ or $K3$ for the final block. Which key we use depends on whether or not we added padding. The algorithm follows, and is also shown in Figure 6.3.

$$
\begin{array}{l}
\textbf{Algorithm} \;\; \mathrm{FCBC}_{E_{K1},E_{K2},E_{K3}}(M) \\
\textbf{if} \;\; M \in (\{0,1\}^n)^+ \\
\qquad \textbf{then} \;\; K \leftarrow K2, \text{ and } P \leftarrow M \\
\qquad \textbf{else} \;\; K \leftarrow K3, \text{ and } P \leftarrow M \,.\, 10^i, \text{ where } i \leftarrow n-1-|M| \bmod n \\
\text{Let } P = P_1 \cdots P_m, \text{ where } |P_1| = \cdots = |P_m| = n \\
C_0 \leftarrow 0^n \\
\textbf{for} \;\; i \leftarrow 1 \textbf{ to } \; m-1 \textbf{ do} \\
\qquad C_i \leftarrow E_{K1}(P_i \oplus C_{i-1}) \\
\textbf{return} \;\; E_K(P_m \oplus C_{m-1})
\end{array}
$$

In Section 6.4 we prove the security of this construction. Correctness follows from the result on the security of ECBC.

Avoiding multiple encryption keys: XCBC. Most block ciphers have a key-setup cost, when the key is turned into subkeys. The subkeys are often larger than the original key, and computing them may be expensive. So keying the underlying block cipher with multiple keys, as is done in EMAC, ECBC, and FCBC, is actually not so desirable. It would be better to use the same key for all of the block-cipher invocations. The algorithm XCBC does this.

We again make two cases. If $M \in (\{0,1\}^n)^+$ we CBC as usual, except that we XOR in an n-bit key, $K2$, before enciphering the last block. If $M \notin (\{0,1\}^n)^+$ then append minimal 10^i padding ($i \geq 0$) and CBC as usual, except that we XOR in a different n-bit key, $K3$, before enciphering the last block. Here, in full, is the algorithm. Also see Figure 6.4. The proof of security can be found in Section 6.5.

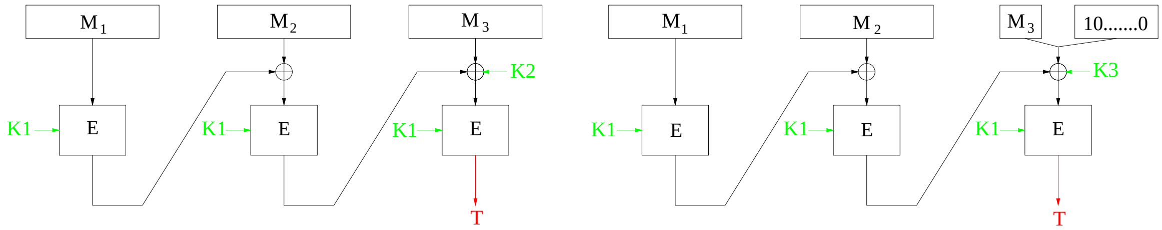

Figure 6.4: *The* XCBC *construction with a block cipher* $E : \mathsf{Key} \times \{0,1\}^n \to \{0,1\}^n$. *We use keys* $K1 \in \mathsf{Key}$ *and* $K2, K3 \in \{0,1\}^n$. *On the left is the case where* $|M|$ *is a positive multiple of* n*; on the right is the case where* $|M|$ *is not a positive multiple of* n.

Algorithm $\mathrm{XCBC}_{E_{K1},E_{K2},E_{K3}}(M)$
if $M \in (\{0,1\}^n)^+$
    then $K \leftarrow K2$, and $P \leftarrow M$
    else $K \leftarrow K3$, and $P \leftarrow M \,.\, 10^i$, where $i \leftarrow n - 1 - |M| \bmod n$
Let $P = P_1 \cdots P_m$, where $|P_1| = \cdots = |P_m| = n$
$C_0 \leftarrow 0^n$
for $i \leftarrow 1$ **to** $m-1$ **do**
    $C_i \leftarrow E_{K1}(P_i \oplus C_{i-1})$
return $E_{K1}(P_m \oplus C_{m-1} \oplus K)$

We have now defined functions $\mathrm{CBC}_{\rho_1}(\cdot)$, $\mathrm{EMAC}_{\rho_1,\rho_2}(\cdot)$, $\mathrm{ECBC}_{\rho_1,\rho_2,\rho_3}(\cdot)$, $\mathrm{FCBC}_{\rho_1,\rho_2,\rho_3}(\cdot)$, and $\mathrm{XCBC}_{\rho_1,\rho_2,\rho_3}(\cdot)$ where $\rho_1, \rho_2, \rho_3\colon \{0,1\}^n \to \{0,1\}^n$. We emphasize that the definitions make sense for any ρ_1, ρ_2, $\rho_3\colon \{0,1\}^n \to \{0,1\}^n$; in particular, we don't require ρ_1, ρ_2, ρ_3 be permutations. Notice that we interchangeably use notation such as ρ_1 and E_{K1}; the key $K1$ is simply naming a function $\rho_1 = E_{K1}$.

Basic facts. It is often convenient to replace random permutations with random functions, or vice versa. The following proposition lets us easily do this. For a proof see Proposition 2.5 in [3].

Lemma 6.2.1 [PRF/PRP Switching] *Fix* $n \geq 1$. *Let* $\mathcal{A}$ *be an adversary that asks at most* p *queries. Then*

$$\left|\Pr[\pi \stackrel{R}{\leftarrow} \mathrm{Perm}(n) : \mathcal{A}^{\pi(\cdot)} = 1] - \Pr[\rho \stackrel{R}{\leftarrow} \mathrm{Rand}(n,n) : \mathcal{A}^{\rho(\cdot)} = 1]\right| \leq p(p-1)/2^{n+1}. \qquad \blacksquare$$

As is customary, we will show the security of our MACs by showing that their information-theoretic versions approximate random functions. As is standard, this will be enough to pass to the complexity-theoretic scenario. Part of the proof is Proposition 2.7 of [3].

Lemma 6.2.2 [Inf. Th. PRF $\Rightarrow$ Comp. Th. PRF] *Fix $n \geq 1$. Let* CONS *be a construction such that* $\mathrm{CONS}_{\rho_1,\rho_2,\rho_3}(\cdot)\colon \{0,1\}^* \to \{0,1\}^n$ *for any* $\rho_1, \rho_2, \rho_3 \in \mathrm{Rand}(n,n)$*. Suppose that if* $|M| \leq \mu$ *then* $\mathrm{CONS}_{\rho_1,\rho_2,\rho_3}(M)$ *depends on the values of* ρ_i *on at most* p *points (for* $1 \leq i \leq 3$*). Let* $E\colon \mathrm{Key} \times \{0,1\}^n \to \{0,1\}^n$ *be a family of functions. Then*

$$\begin{aligned}
\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{CONS}[E]}(t,q,\mu) &\leq \mathrm{Adv}^{\mathrm{prf}}_{\mathrm{CONS}[\mathrm{Perm}(n)]}(q,\mu) + 3\cdot\mathrm{Adv}^{\mathrm{prp}}_{E}(t',p), \text{ and}\\
\mathrm{Adv}^{\mathrm{mac}}_{\mathrm{CONS}[E]}(t,q,\mu) &\leq \mathrm{Adv}^{\mathrm{prf}}_{\mathrm{CONS}[\mathrm{Perm}(n)]}(q,\mu) + 3\cdot\mathrm{Adv}^{\mathrm{prp}}_{E}(t',p) + \frac{1}{2^n}\,,
\end{aligned}$$

where $t' = t + O(pn)$. ∎

6.3 Security of ECBC

In this section we prove the security of the ECBC construction. See Section 6.2 for a definition of $\mathrm{ECBC}_{\rho_1,\rho_2,\rho_3}(M)$, where $\rho_1, \rho_2, \rho_3 \in \mathrm{Rand}(n,n)$.

Our approach is as follows: we view the CBC MAC as an almost universal-2 family of hash functions and prove a bound on its collision probability. Then we create a PRF by applying one of two random functions to the output of CBC MAC and claim that this construction is itself a good PRF. Applying a PRF to a universal-2 hash function is a well-known approach for creating a PRF or MAC [9, 30, 6]. The novelty here is the extension to three keys and, more significantly, the treatment of the CBC MAC as an almost universal-2 family of hash functions. The latter might be against one's instincts because the CBC MAC is a much stronger object than a universal hash-function family. Here we ignore that extra strength and study only the collision probability.

We wish to show if $M, M' \in (\{0,1\}^n)^+$ are distinct strings then $\Pr_\pi[\mathrm{CBC}_\pi(M) = \mathrm{CBC}_\pi(M')]$ is small. By "small" we mean a slowly growing function of $m = |M|/n$ and $m' = |M'|/n$. Formally, for $n, m, m' \geq 1$, define the *collision probability* of the CBC MAC to be

$$V_n(m, m') \stackrel{\text{def}}{=} \max_{M \in \{0,1\}^{nm}, M' \in \{0,1\}^{nm'}, M \neq M'} \{\Pr[\pi \stackrel{R}{\leftarrow} \mathrm{Perm}(n) :\ \mathrm{CBC}_\pi(M) = \mathrm{CBC}_\pi(M')]\}.$$

(The character "V" is meant to suggest collisions.) It is not hard to infer from [22] a collision bound of $V_n(m, m') \leq 2.5\,(m+m')^2/2^n$ by using their bound on EMAC and realizing that collisions in the underlying CBC MAC must show through to the EMAC output. But a direct analysis easily yields a (slightly better) bound.

We can upper bound the collision probability of the CBC MAC in two different ways. One way is to build on Petrank and Rackoff's result [22] which measures the distance between EMAC[Rand(n,n)] and Rand$(\{0,1\}^*, n)$. Another way is with a direct analysis. Here is the former.

Lemma 6.3.1 [First CBC MAC Collision Bound] *Let $n, m, m' \geq 1$. Then*

$$V_n(m, m') \ \leq\ \frac{2.5\,(m+m')^2}{2^n}\ .$$

Proof: Let $\mathcal{A}$ be an adversary that asks queries in $(\{0,1\}^n)^+$ which total $p = m + m'$ blocks. Let $N = 2^n$. From [22] we know that

$$\Pr[\rho, \sigma \stackrel{R}{\leftarrow} \mathrm{Rand}(n,n) :\ \mathcal{A}^{\mathrm{EMAC}_{\rho,\sigma}(\cdot)} = 1] \ - \ \Pr[R \stackrel{R}{\leftarrow} \mathrm{Rand}((\{0,1\}^n)^+, n) :\ \mathcal{A}^{R(\cdot)} = 1]$$

$$\leq\ 2p^2/N\ .$$

The above is true for *any* adversary who asks at most p blocks. Consider the *particular* adversary that asks its oracle the fixed queries $M \in (\{0,1\}^n)^m$ and $M' \in (\{0,1\}^n)^{m'}$, where $M \neq M'$, and then outputs "1" if the return values are the same. The result above tells us that

$$\Pr[\rho, \sigma \stackrel{R}{\leftarrow} \mathrm{Rand}(n,n) : \ \mathrm{EMAC}_{\rho,\sigma}(M) = \mathrm{EMAC}_{\rho,\sigma}(M')] - \\ \Pr[R \stackrel{R}{\leftarrow} \mathrm{Rand}((\{0,1\}^n)^+, n) : \ R(M) = R(M')] \ \leq \ 2p^2/N \ .$$

Of course the second term is exactly $1/N$ and so

$$\Pr[\rho, \sigma \stackrel{R}{\leftarrow} \mathrm{Rand}(n,n) : \ \mathrm{EMAC}_{\rho,\sigma}(M) = \mathrm{EMAC}_{\rho,\sigma}(M')] \ \leq \ \frac{2p^2+1}{N} \ .$$

Lemma 6.2.1 lets us now replace the random function ρ by a random permutation π giving

$$\begin{aligned} \Pr[\pi \stackrel{R}{\leftarrow} \mathrm{Perm}(n); \ \sigma \stackrel{R}{\leftarrow} \mathrm{Rand}(n,n) : \ \mathrm{EMAC}_{\pi,\sigma}(M) = \mathrm{EMAC}_{\pi,\sigma}(M')] & \\ \leq \ & \frac{2p^2+1}{N} + \frac{p(p-1)}{2N} \\ \leq \ & \frac{2.5p^2}{N} \ . \end{aligned}$$

Now we observe that

$$\Pr[\pi \stackrel{R}{\leftarrow} \mathrm{Perm}(n) : \ \mathrm{CBC}_\pi(M) = \mathrm{CBC}_\pi(M')] \\ \leq \ \Pr[\pi \stackrel{R}{\leftarrow} \mathrm{Perm}(n); \ \sigma \stackrel{R}{\leftarrow} \mathrm{Rand}(n,n) : \ \mathrm{EMAC}_{\pi,\sigma}(M) = \mathrm{EMAC}_{\pi,\sigma}(M')]$$

since collisions in $\mathrm{EMAC}_{\pi,\sigma}$ are caused *either* by a collision in $\mathrm{CBC}_\pi(M)$ and $\mathrm{CBC}_\pi(M')$ *or* by a collision in σ at the distinct points $\mathrm{CBC}_\pi(M)$ and $\mathrm{CBC}_\pi(M')$. We conclude that

$$\Pr[\pi \stackrel{R}{\leftarrow} \mathrm{Perm}(n) : \ \mathrm{CBC}_\pi(M) = \mathrm{CBC}_\pi(M')] \ \leq \ \frac{2.5(m+m')^2}{N} \ ,$$

as desired. ▮

We also have a direct argument, much simpler than given in [22], which yields a slightly better bound.

Lemma 6.3.2 [Second CBC Collision Bound] *Fix $n \geq 1$ and let $N = 2^n$. Let $M, M' \in (\{0,1\}^n)^+$ be distinct strings having $m = |M|/n$ and $m' = |M'|/n$ blocks. Assume that $m, m' \leq N/4$. Then*

$$V_n(m, m') \;\leq\; \frac{(m+m')^2}{2^n}\,. \qquad \blacksquare$$

Proof: Although M and M' are distinct, they may share some common prefix. Let k be the index of the last block in which M and M' agree. (If M and M' have unequal first blocks then $k = 0$.)

Each particular permutation π is equally likely among all permutations from $\{0,1\}^n$ to $\{0,1\}^n$. In our analysis, we will view the selection of π as an incremental procedure. This will be equivalent to selecting π uniformly at random. In particular, we view the computation of $\mathrm{CBC}_\pi(M)$ and $\mathrm{CBC}_\pi(M')$ as playing the game given in Figure 6.5. Here the notation M_i indicates the ith block of M. We initially set each range point of π as `undefined`; the notation $\mathrm{Domain}(\pi)$ represents the set of points x where $\pi(x)$ is no longer `undefined`. We use $\mathrm{Range}(\pi)$ to denote the set of points $\pi(x)$ which are no longer `undefined`; we use $\overline{\mathrm{Range}}(\pi)$ to denote $\{0,1\}^n - \mathrm{Range}(\pi)$.

During the game, the X_i are those values produced after XORing with the current message block, M_i, and the Y_i values are $\pi(X_i)$. See Figure 6.6.

We are concerned with the probability that π will cause $\mathrm{CBC}_\pi(M) = \mathrm{CBC}_\pi(M')$, which

$bad \leftarrow \texttt{false}$; **for all** $x \in \{0,1\}^n$ **do** $\pi(x) \leftarrow \texttt{undefined}$; $X_1 \leftarrow M_1$; $X'_1 \leftarrow M'_1$; $BAD \leftarrow \{X_1, X'_1\}$

for $i \leftarrow 1$ **to** k **do**
  if $X_i \in \mathrm{Domain}(\pi)$ **then** $Y_i \leftarrow Y'_i \leftarrow \pi(X_i)$
  else $Y_i \leftarrow Y'_i \stackrel{R}{\leftarrow} \overline{\mathrm{Range}}(\pi)$; $\pi(X_i) \leftarrow Y_i$
    if $i < m$ **then** $X_{i+1} \leftarrow Y_i \oplus M_{i+1}$
      if $X_{i+1} \in BAD$ **then** $bad \leftarrow \texttt{true}$ **else** $BAD \leftarrow BAD \cup \{X_{i+1}\}$
    if $i < m'$ **then** $X'_{i+1} \leftarrow Y'_i \oplus M'_{i+1}$
      if $X'_{i+1} \in BAD$ **then** $bad \leftarrow \texttt{true}$ **else** $BAD \leftarrow BAD \cup \{X'_{i+1}\}$

for $i \leftarrow k+1$ **to** m **do**
  if $X_i \in \mathrm{Domain}(\pi)$ **then** $Y_i \leftarrow \pi(X_i)$
  else $Y_i \stackrel{R}{\leftarrow} \overline{\mathrm{Range}}(\pi)$; $\pi(X_i) \leftarrow Y_i$
    if $i < m$ **then** $X_{i+1} \leftarrow Y_i \oplus M_{i+1}$
      if $X_{i+1} \in BAD$ **then** $bad \leftarrow \texttt{true}$ **else** $BAD \leftarrow BAD \cup \{X_{i+1}\}$

for $i \leftarrow k+1$ **to** m' **do**
  if $X'_i \in \mathrm{Domain}(\pi)$ **then** $Y'_i \leftarrow \pi(X'_i)$
  else $Y'_i \stackrel{R}{\leftarrow} \overline{\mathrm{Range}}(\pi)$; $\pi(X'_i) \leftarrow Y'_i$
    if $i < m$ **then** $X'_{i+1} \leftarrow Y'_i \oplus M'_{i+1}$
      if $X'_{i+1} \in BAD$ **then** $bad \leftarrow \texttt{true}$ **else** $BAD \leftarrow BAD \cup \{X'_{i+1}\}$

Figure 6.5: *Game used in the proof of Lemma 6.3.2. The algorithm gives one way to compute the CBC MAC of distinct messages* $M = M_1 \cdots M_m$ *and* $M' = M'_1 \cdots M'_{m'}$*. These messages are identical up to block* k*, but different afterwards. The computed MACs are* Y_m *and* $Y_{m'}$*, respectively.*

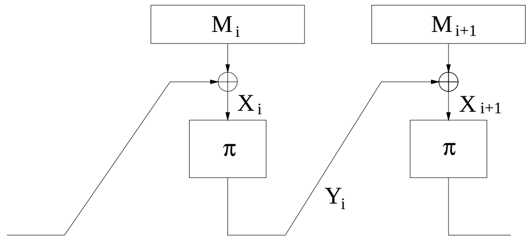

Figure 6.6: *A fragment of the CBC construction showing the labeling convention used in the proof of Lemma 6.3.2.*

will occur in our game iff $Y_m = Y'_{m'}$. Since π is invertible, this occurs iff $X_m = X'_{m'}$. As we shall see, this condition will cause *bad* $=$ `true` in our game. However, we actually set *bad* to `true` in many other cases in order to simplify the analysis.

The idea behind the variable *bad* is as follows: throughout the program (lines 4, 11, and 16) we randomly choose a range value for π at some `undefined` domain point. Since π has not yet been determined at this point, the selection of our range value will be an independent uniform selection: there is no dependence on any prior choice.

If the range value for π were already determined by some earlier choice, the analysis would become more involved. We avoid the latter condition by setting *bad* to `true` whenever such interdependencies are detected. The detection mechanism works as follows: throughout the processing of M and M' we will require π be evaluated at $m+m'$ domain points $X_1, \cdots, X_m$ and $X'_1, \cdots, X'_{m'}$. If all of these domain points are distinct (ignoring duplications due to any common prefix of M and M'), we can rest assured that we are free to assign their corresponding range points without constraint. We maintain a set BAD to track which domain points have already been determined; initially X_1 and X'_1 are the only such points, since future values will depend on random choices not yet made. Of course if $k > 0$ then $X_1 = X'_1$ and BAD contains only one value. Next we begin randomly choosing range points;

if ever any such choice leads to a value already contained in the BAD set, we set the flag bad to `true`.

We now bound the probability of the event that $bad = \texttt{true}$ by analyzing our game. The variable bad can be set `true` in lines 6, 8, 13, and 18. In each case it is required that some Y_i was selected such that $Y_i \oplus M_{i+1} \in BAD$ (or possibly that some Y_i' was selected such that $Y_i' \oplus M_{i+1}' \in BAD$). The set BAD begins with at most 2 elements and then grows by 1 with each random choice of Y_i or Y_i'. We know that on the ith random choice in the game the BAD set will contain at most $i+1$ elements. And so each random choice of Y_i (resp. Y_i') from the co-range of π will cause $Y_i \oplus M_{i+1}$ (resp. $Y_i' \oplus M_{i+1}'$) to be in BAD with probability at most $(i+1)/(N-i+1)$. We have already argued that in the absence of $bad = \texttt{true}$ each of the random choices we make are independent. We make $m-1$ choices of Y_i to produce X_2 through X_m and $m'-1$ choices of Y_i' to determine X_2' through $X_{m'}'$ and so we can compute

$$\Pr[bad = \texttt{true}] \;\leq\; \sum_{i=1}^{m-1+m'-1} \frac{i+1}{N-i+1}.$$

Using the fact that $m, m' \leq N/4$, we can bound the above by

$$\sum_{i=1}^{m+m'-2} \frac{i+1}{N-i} \;\leq\; \frac{2}{N} \sum_{i=1}^{m+m'-2} i+1 \;\leq\; \frac{(m+m')^2}{N}\,.$$

This completes the proof. ∎

Fix $n \geq 1$ and let F: $\text{Key} \times \{0,1\}^n \to \{0,1\}^n$ be a family of functions. Let ECBC[F] be the family of functions $\text{ECBC}_{f1,f2,f3}(\cdot)$ indexed by Key $\times$ Key $\times$ Key. We now use the above to show that ECBC[Perm(n)] is close to being a random function.

Theorem 6.3.3 [ECBC $\approx$ Rand] *Fix $n \geq 1$ and let $N = 2^n$. Let $\mathcal{A}$ be an adversary which asks at most q queries each of which is at most mn-bits. Assume $m \leq N/4$. Then*

$$\Pr[\pi_1, \pi_2, \pi_3 \xleftarrow{R} \mathrm{Perm}(n) :\ \mathcal{A}^{\mathrm{ECBC}_{\pi_1,\pi_2,\pi_3}(\cdot)} = 1] \ -\ \Pr[R \xleftarrow{R} \mathrm{Rand}(\{0,1\}^*, n) :\ \mathcal{A}^{R(\cdot)} = 1]$$
$$\leq\ \frac{q^2}{2} V_n(m,m) + \frac{q^2}{2N}\ \leq\ \frac{(2m^2+1)q^2}{N}.$$

Proof: We will first compute a related probability where the final permutation is a random function; this will simplify the analysis. So we are interested in the quantity

$$\Pr[\pi_1 \xleftarrow{R} \mathrm{Perm}(n);\ \rho_2, \rho_3 \xleftarrow{R} \mathrm{Rand}(n,n) :\ \mathcal{A}^{\mathrm{ECBC}_{\pi_1,\rho_2,\rho_3}(\cdot)} = 1]$$
$$- \Pr[R \xleftarrow{R} \mathrm{Rand}(\{0,1\}^*, n) :\ \mathcal{A}^{R(\cdot)} = 1]\,.$$

For economy of notation we encapsulate the initial parts of our experiments into the probability symbols $\Pr_1$ and $\Pr_2$, rewriting the above as $\Pr_1[\mathcal{A}^{\mathrm{ECBC}_{\pi_1,\rho_2,\rho_3}(\cdot)} = 1] - \Pr_2[\mathcal{A}^{R(\cdot)} = 1]$.

We condition $\Pr_1$ on whether a collision occurred within π_1, and we differentiate between collisions among messages whose lengths are a nonzero multiple of n (which are not padded) and other messages (which are padded). Let $\mathsf{UnpadCol}$ be the event that there is a collision among the unpadded messages and let PadCol be the event that there is a collision among the padded messages. We rewrite the above as

$$\Pr_1[\mathcal{A}^{\mathrm{ECBC}_{\pi_1,\rho_2,\rho_3}(\cdot)} = 1 \mid \mathsf{UnpadCol} \vee \mathsf{PadCol}]\ \Pr_1[\mathsf{UnpadCol} \vee \mathsf{PadCol}]$$
$$+\ \ \Pr_1[\mathcal{A}^{\mathrm{ECBC}_{\pi_1,\rho_2,\rho_3}(\cdot)} = 1 \mid \overline{\mathsf{UnpadCol} \vee \mathsf{PadCol}}]\ \Pr_1[\overline{\mathsf{UnpadCol} \vee \mathsf{PadCol}}]$$
$$-\ \ \Pr_2[\mathcal{A}^R(\cdot) = 1].$$

We observe that in the absence of event $(\mathsf{UnpadCol} \vee \mathsf{PadCol})$, the adversary sees the output of a random function on distinct points; that is, she is presented with random, uncorrelated points over the range $\{0,1\}^n$. Therefore we know

$$\Pr_1[\mathcal{A}^{\mathrm{ECBC}_{\pi_1,\rho_2,\rho_3}(\cdot)} = 1 \mid \overline{\mathsf{UnpadCol} \vee \mathsf{PadCol}}] \;=\; \Pr_2[\mathcal{A}^{R(\cdot)} = 1].$$

Now bounding

$$\Pr_1[\overline{\mathsf{UnpadCol} \vee \mathsf{PadCol}}]$$

and

$$\Pr_1[\mathcal{A}^{\mathrm{ECBC}_{\pi_1,\rho_2,\rho_3}(\cdot)} = 1 \mid \mathsf{UnpadCol} \vee \mathsf{PadCol}]$$

by 1, we reduce the bound on the obtainable advantage to $\Pr_1[\mathsf{UnpadCol} \vee \mathsf{PadCol}] \;\leq\; \Pr_1[\mathsf{UnpadCol}] + \Pr_1[\mathsf{PadCol}]$. To bound this quantity, we break each event into the disjoint union of several other events; define $\mathsf{UnpadCol}_i$ to be the event that a collision in π_1 occurs for the first time as a result of the ith unpadded query. Let q_u be the number of unpadded queries made by the adversary, and let q_p be the number of padded queries. Then

$$\Pr_1[\mathsf{UnpadCol}] = \sum_{i=1}^{q_u} \Pr_1[\mathsf{UnpadCol}_i]\,.$$

Now we compute each of the probabilities on the right-hand side above. If we know no collisions occurred during the first $i-1$ queries we know the adversary has thus far seen only images of distinct inputs under a random function. Any adaptive strategy she adopts in this case could be replaced by a non-adaptive strategy where she pre-computes her queries under the assumption that the first $i-1$ queries produce random points. In other words, any adaptive adversary can be replaced by a non-adaptive adversary which does at least as well. A non-adaptive strategy would consist of generating all q inputs in advance, and from the collision bound on the hash family we know $\Pr_1[\mathsf{UnpadCol}_i] \;\leq\; (i-1)V_n(m,m)\,$.

Summing over i we get

$$V_n(m,m)\sum_{i=1}^{q_u}(i-1) \;\le\; \frac{q_u^2}{2}V_n(m,m)\,.$$

Repeating this analysis for the padded queries we obtain an overall bound of

$$\frac{q_u^2}{2}V_n(m,m) + \frac{q_p^2}{2}V_n(m,m) \;\le\; \frac{q^2}{2}V_n(m,m)\,.$$

Finally we must replace the PRFs ρ_2 and ρ_3 with the PRPs π_2 and π_3. Using Lemma 6.2.1 this costs us an extra $q_u^2/2N + q_p^2/2N \le q^2/2N$, and so our final bound is

$$\frac{q^2}{2}V_n(m,m) + \frac{q^2}{2N}\,,$$

and if we apply the bound on $V_n(m,m)$ from Lemma 6.3.2 we have

$$\frac{q^2}{2}\,\frac{4m^2}{N} + \frac{q^2}{2N} \;\le\; \frac{(2m^2+1)q^2}{N}\,.$$

This completes the proof. ∎

Tightness, and non-tightness, of our bound. Employing well-known techniques (similar to [24]), it is easy to exhibit a known-message attack which forges with high probability (> 0.3) after seeing $q = 2^{n/2}$ message/tag pairs. In this sense the analysis looks tight. The same statement can be made for the FCBC and XCBC analyses. But if we pay attention not only to the number of messages MACed, but also their lengths, then none of these analyses is tight. That is because the security bound degrades quadratically with the total number of message *blocks*, while the the attack efficiency improves quadratically with the total number of *messages*. Previous analyses for the CBC MAC and its variants all shared this same characteristic.

Complexity-Theoretic Result. In the usual way we can now pass from the information-theoretic result to a complexity-theoretic one. We state the result, which follows using Lemma 6.2.2.

Corollary 6.3.4 [ECBC is a PRF] *Fix $n \geq 1$ and let $N = 2^n$. Let E: $\mathrm{Key} \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher. Then*

$$\begin{aligned}
\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{ECBC}[E]}(t,q,mn) &\leq \frac{2m^2q^2+q^2}{N} + 3\cdot \mathrm{Adv}^{\mathrm{prp}}_{E}(t',q'), \quad \textit{and} \\
\mathrm{Adv}^{\mathrm{mac}}_{\mathrm{ECBC}[E]}(t,q,mn) &\leq \frac{2m^2q^2+q^2+1}{N} + 3\cdot \mathrm{Adv}^{\mathrm{prp}}_{E}(t',q')
\end{aligned}$$

where $t' = t + O(mq)$ and $q' = mq$. ▮

It is worth noting that using a very similar argument to Theorem 6.3.3 we can easily obtain the same bound for the EMAC construction [4]. This yields a proof which is quite a bit simpler than that found in [22]. The reader who has already carefully read the previous proof for ECBC will need only skim this proof since it is highly similar and included only for completeness.

Theorem 6.3.5 [EMAC ≈ Rand] *Fix $n \geq 1$ and let $N = 2^n$. Let $\mathcal{A}$ be an adversary which asks at most q queries, each of which is at most m n-bit blocks, where $m \leq N/4$. Then*

$$\begin{aligned}
\Pr[\pi,\sigma \stackrel{R}{\leftarrow} \mathrm{Perm}(n) :\ \mathcal{A}^{\mathrm{EMAC}_{\pi,\sigma}(\cdot)} = 1] - \Pr[\rho \stackrel{R}{\leftarrow} \mathrm{Rand}((\{0,1\}^n)^+, n) :\ \mathcal{A}^{\rho(\cdot)} = 1] \\
\leq \frac{q^2}{2} V_n(m,m) + \frac{q^2}{2N} \\
\leq \frac{(2m^2+1)q^2}{N}\,.
\end{aligned}$$

Before proving the theorem, we begin by stating and proving a general lemma regarding the following construction: given a message M compute $\rho(h(M))$ where h is a hash function drawn from some family of hash functions and ρ is a random function. We evaluate this construction as a PRF. Immediately afterward, we will prove Theorem 6.3.5.

Lemma 6.3.6 [PRF(Hash) as a PRF] *Fix $n \geq 1$. Let $H = \{h\colon \mathsf{Msg} \to \{0,1\}^n\}$ be a family of hash functions. Let $\delta(m)$ be a function such that for all distinct pairs $M, M' \in \mathsf{Msg}$, with M and M' at most mn bits long, $\Pr_{h \xleftarrow{R} H}[h(M) = h(M')] \leq \delta(m)$. Let $\mathcal{A}$ be an adversary that asks q queries from Msg, each query of length at most mn bits. Then*

$$\begin{aligned} \Pr[h \xleftarrow{R} H;\ \rho \xleftarrow{R} \mathrm{Rand}(n,n) :\ \mathcal{A}^{\rho(h(\cdot))} = 1] \quad &- \quad \Pr[R \xleftarrow{R} \mathrm{Rand}(\mathsf{Msg}, n) :\ \mathcal{A}^{R(\cdot)} = 1] \\ &\leq \quad \frac{q^2}{2}\,\delta(m). \end{aligned}$$

Proof: For economy of notation we encapsulate the initial parts of our experiments into the probability symbols, rewriting the difference in probabilities above as $\Pr_1[\mathcal{A}^{\rho(h(\cdot))} = 1] - \Pr_2[\mathcal{A}^{R(\cdot)} = 1]$. Now condition $\Pr_1$ on whether a collision occurred within the selected function h: define the event HCol to indicate that there is some repeated value among $h(m_1), h(m_2), \ldots, h(m_q)$ within the experiment in $\Pr_1$. Then we may rewrite the above as

$$\Pr_1[\mathcal{A}^{\rho(h(\cdot))} = 1 \mid \overline{\mathsf{HCol}}]\,\Pr_1[\overline{\mathsf{HCol}}] \ + \ \Pr_1[\mathcal{A}^{\rho(h(\cdot))} = 1 \mid \mathsf{HCol}]\,\Pr_1[\mathsf{HCol}] \ - \ \Pr_2[\mathcal{A}^{R(\cdot)} = 1].$$

We observe that in the absence of event HCol, the adversary sees the output of a random function on distinct points; that is, she is presented with random, uncorrelated points over the range $\{0,1\}^n$. Therefore we know

$$\Pr_1[\mathcal{A}^{\rho(h(\cdot))} = 1 \mid \overline{\mathsf{HCol}}] = \Pr_2[\mathcal{A}^{R(\cdot)} = 1].$$

Bounding $\Pr_1[\overline{\mathsf{HCol}}]$ and $\Pr_1[\mathcal{A}^{\rho(h(\cdot))} = 1 \mid \mathsf{HCol}]$ by 1, we reduce the bound on the obtainable advantage to $\Pr_1[\mathsf{HCol}]$.

To bound $\Pr_1[\mathsf{HCol}]$, we break the event HCol into the disjoint union of q events; define HCol_i to be the event that a collision in h occurs for the first time as a result of the ith query, then

$$\Pr_2[\mathsf{HCol}] = \sum_{i=1}^{q} \Pr_2[\mathsf{HCol}_i] \ .$$

Now we may easily compute each of the probabilities on the right-hand side above. If we know no collisions occurred during the first $i-1$ queries, we know the adversary has thus far seen only images of distinct inputs under a random function. Any adaptive strategy she adopts in this case could be effectively replaced by an equivalent non-adaptive strategy where she pre-computes her queries under the assumption that the first $i-1$ queries produce random points. In other words, any adaptive adversary can be replaced by an equivalent non-adaptive adversary which would do at least as well. A non-adaptive strategy would consist of generating all q inputs in advance, and from the collision bound on the hash family we know $\Pr_1[\mathsf{HCol}_i]$ must be at most $(i-1)\delta(m)$. Summing over i we get

$$\sum_{i=1}^{q}(i-1)\delta(m) \ \leq \ \frac{q^2}{2}\delta(m)$$

which is the desired bound. ▮

And now we may give the proof of Theorem 6.3.5.

Proof: We begin by invoking Lemma 6.3.6 with the various parameters relevant for EMAC: let the hash family H be the set of functions CBC_π for all $\pi \in \mathrm{Perm}(n)$. Let the message space Msg be $(\{0,1\}^n)^+$, and let $\delta(m) = 4m^2/N$. Then Lemma 6.3.2 gives that

$\Pr_{h \xleftarrow{R} H}[h(M) = h(M')] \leq \delta(m)$. Therefore we may conclude from Lemma 6.3.6 that

$$\Pr[\pi \xleftarrow{R} \mathrm{Perm}(n);\quad \rho \xleftarrow{R} \mathrm{Rand}((\{0,1\}^n)^+, n) : \mathcal{A}^{\rho(\mathrm{CBC}_\pi(\cdot))} = 1] \; - \; \Pr[\rho \xleftarrow{R} \mathrm{Rand}((\{0,1\}^n)^+, n) : \mathcal{A}^{\rho(\cdot)} = 1] \;\leq\; \frac{2q^2m^2}{N}\,.$$

Finally the PRF ρ in the left-hand probability must be replaced with a PRP to properly realize the EMAC construction. Using Lemma 6.2.1 this costs us an extra $q^2/2N$, and so our final bound is

$$\frac{q^2}{2}V_n(m,m) + \frac{q^2}{2N}\,,$$

and if we apply the bound from Lemma 6.3.2 we have

$$\frac{q^2}{2}\,\frac{4m^2}{N} + \frac{q^2}{2N} \;\leq\; \frac{(2m^2+1)q^2}{N}\,,$$

as required. ▮

6.4 Security of FCBC

In this section we prove the security of the FCBC construction, obtaining the same bound we have for ECBC. See Section 6.2 for a definition of $\mathrm{FCBC}_{\rho_1,\rho_2,\rho_3}(M)$, where $\rho_1, \rho_2, \rho_3 \in \mathrm{Rand}(n,n)$.

Theorem 6.4.1 [FCBC ≈ Rand] *Fix $n \geq 1$ and let $N = 2^n$. Let $\mathcal{A}$ be an adversary which asks at most q queries, each of which is at most mn bits. Assume $m \leq N/4$. Then*

$$\Pr[\pi_1,\pi_2,\pi_3 \xleftarrow{R} \mathrm{Perm}(n) : \mathcal{A}^{\mathrm{FCBC}_{\pi_1,\pi_2,\pi_3}(\cdot)} = 1] \;-\; \Pr[R \xleftarrow{R} \mathrm{Rand}(\{0,1\}^*, n) : \mathcal{A}^{R(\cdot)} = 1]$$

$$\leq \frac{q^2}{2}V_n(m,m) + \frac{q^2}{2N}$$

$$\leq \frac{(2m^2+1)q^2}{N}\,.$$

Proof: Let us compare the distribution on functions

$$\{\mathrm{ECBC}_{\pi_1,\pi_2,\pi_3}(\cdot) \mid \pi_1,\pi_2,\pi_3 \stackrel{R}{\leftarrow} \mathrm{Perm}(n)\} \text{ and } \{\mathrm{FCBC}_{\pi_1,\sigma_2,\sigma_3}(\cdot) \mid \pi_1,\sigma_2,\sigma_3 \stackrel{R}{\leftarrow} \mathrm{Perm}(n)\}\ .$$

We claim that these are the *same* distribution, so, information theoretically, the adversary has no way to distinguish a random sample drawn from one distribution from a random sample from the other. The reason is simple. In the ECBC construction we compose the permutation π_1 with the random permutation π_2. But the result of such a composition is just a random permutation, σ_2. Elsewhere in the ECBC construction we compose the permutation π_1 with the random permutation π_3. But the result of such a composition is just a random permutation, σ_3. Making these substitutions—σ_2 for $\pi_2 \circ \pi_1$, and σ_3 for $\pi_3 \circ \pi_1$, we recover the definition of ECBC. Changing back to the old variable names, we now have that that

$$\begin{aligned}&\Pr[\pi_1,\pi_2,\pi_3 \stackrel{R}{\leftarrow} \mathrm{Perm}(n):\ \mathcal{A}^{\mathrm{FCBC}_{\pi_1,\pi_2,\pi_3}(\cdot)} = 1]\\ &\qquad = \Pr[\pi_1,\pi_2,\pi_3 \stackrel{R}{\leftarrow} \mathrm{Perm}(n):\ \mathcal{A}^{\mathrm{ECBC}_{\pi_1,\pi_2,\pi_3}(\cdot)} = 1]\end{aligned}$$

So the bound of our theorem follows immediately from Theorem 6.3.3. ∎

Since the bound for FCBC exactly matches the bound for ECBC, Corollary 6.3.4 applies to FCBC as well.

6.5 Security of XCBC

In this section we prove the security of the XCBC construction. See Section 6.2 for a definition of $\mathrm{XCBC}_{\rho_1,\rho_2,\rho_3}(M)$, where $\rho_1,\rho_2,\rho_3 \in \mathrm{Rand}(n,n)$ and $M \in \{0,1\}^*$.

We first give a lemma which bounds an adversary's ability to distinguish between a pair of random permutations, $\pi_1(\cdot)$, $\pi_2(\cdot)$, and the pair $\pi(\cdot)$, $\pi(K\oplus\cdot)$, where π is a random permutation and K is a random n-bit string. This lemma, and ones like it, may make generally useful tools.

Lemma 6.5.1 [Two Permutations From One] *Fix $n \geq 1$ and let $N = 2^n$. Let $\mathcal{A}$ be an adversary which asks at most p queries. Then*

$$\Big| \Pr[\pi \stackrel{R}{\leftarrow} \mathrm{Perm}(n);\ K \stackrel{R}{\leftarrow} \{0,1\}^n :\ \mathcal{A}^{\pi(\cdot),\,\pi(K\oplus\cdot)} = 1]$$
$$- \Pr[\pi_1, \pi_2 \stackrel{R}{\leftarrow} \mathrm{Perm}(n) :\ \mathcal{A}^{\pi_1(\cdot),\,\pi_2(\cdot)} = 1] \Big|$$
$$\leq \frac{p^2}{N}\,.$$

Proof:

We use the game shown in Figure 6.7 to facilitate the analysis. Call the game in that figure Game 1. We play the game as follows: first, the initialization procedure is executed once before we begin. In this procedure we set each range point of π as `undefined`; the notation Domain(π) represents the set of points x where $\pi(x)$ is no longer `undefined`. We use Range(π) to denote the set of points $\pi(x)$ which are no longer `undefined`. We use $\overline{\mathrm{Range}}(\pi)$ to denote $\{0,1\}^n - \mathrm{Range}(\pi)$. Now, when the adversary makes a query X to her left oracle, we execute the code in procedure $\pi(X)$. When she makes a query X to her right oracle, we execute procedure $\pi(K\oplus X)$. We claim that in this setting we perfectly simulate a pair of oracles where the first is a random permutation $\pi(\cdot)$ and the second is $\pi(K\oplus\cdot)$. To verify this, let us examine each of the procedures in turn.

Initialization:

$S, T \leftarrow \{0,1\}^n$; $\quad K \xleftarrow{R} \{0,1\}^n$; $\quad$ **for all** $X \in \{0,1\}^n$ **do** $\pi(X) \leftarrow$ `undefined`

Procedure "$\pi(X)$" (a left-query of X):

if $X \in \text{Domain}(\pi)$ **then** $bad \leftarrow$ `true`, **return** $\pi(X)$

$Y \xleftarrow{R} S$

if $Y \in \text{Range}(\pi)$ **then** $bad \leftarrow$ `true`, $Y \xleftarrow{R} \overline{\text{Range}}(\pi)$

$\pi(X) \leftarrow Y$; $\quad S \leftarrow S - \{Y\}$; $\quad$ **return** Y

Procedure "$\pi(K \oplus X)$" (a right-query of X):

if $(K \oplus X) \in \text{Domain}(\pi)$ **then** $bad \leftarrow$ `true`, **return** $\pi(K \oplus X)$

$Y \xleftarrow{R} T$

if $Y \in \text{Range}(\pi)$ **then** $bad \leftarrow$ `true`, $Y \xleftarrow{R} \overline{\text{Range}}(\pi)$

$\pi(K \oplus X) \leftarrow Y$; $\quad T \leftarrow T - \{Y\}$; $\quad$ **return** Y

Figure 6.7: *Game used in the proof of Lemma 6.5.1. With the shaded text in place the game behaves like a pair of functions $\pi(\cdot)$, $\pi(K \oplus \cdot)$. With the shaded text removed the game behaves like a pair of independent random permutations $\pi_1(\cdot)$, $\pi_2(\cdot)$.*

For procedure $\pi(X)$, we first check to see if X is in the domain of π. We are assuming that the adversary will not repeat a query, but it is possible that X is in the domain of π if the adversary has previously queried the *second* procedure with the string $K \oplus X$. In this case we faithfully return the proper value $\pi(X)$. If her query is not in the domain of π we choose a random element Y from S which we hope to return in response to her query. However, it may be that Y is already in the range of π. Although we always remove from S any value returned from this procedure, it may be that Y was placed in the range of π by the *second* procedure. If this occurs, we choose a random element from the co-range of π and return it. For procedure $\pi(K \oplus \cdot)$ we behave analogously.

Note during the execution of Game 1, we faithfully simulate the pair of functions $\pi(\cdot)$ and $\pi(K \oplus \cdot)$. That is, the view of the adversary would be exactly the same if we had selected a random permutation π and a random n-bit string K and then let her query $\pi(\cdot)$ and $\pi(K\oplus\cdot)$ directly.

Now consider the game we get by removing the shaded statements of Game 1. Call that game Game 2. We claim that Game 2 exactly simulates two random permutations. In other words, the view of the adversary in this game is exactly as if we had given her two independent random permutations π_1 and π_2. Without loss of generality, we assume the adversary never repeats a query. Then examining procedure $\pi(X)$, we see each call to procedure $\pi(X)$ returns a random element from $\{0,1\}^n$ which has not been previously returned by this procedure. This clearly is a correct simulation of a random permutation $\pi_1(\cdot)$. For procedure $\pi(K\oplus X)$, we offset the query value X with some hidden string K, and then return a random element from $\{0,1\}^n$ which has not been previously returned by this procedure. Note in particular that the offset by K has no effect on the behavior of this procedure relative to the previous one. Therefore this procedure perfectly simulates a random independent permutation $\pi_2(\cdot)$.

In both games we sometimes set a variable *bad* to `true` during the execution of the game. In neither case, however, does this have any effect on the values returned by the game.

We name the event that *bad* gets set to `true` as BAD. This event is well-defined for both Games 1 and 2. Notice that Games 1 and 2 behave identically prior to *bad* becoming `true`. One can imagine having two boxes, one for Game 1 and one for Game 2, each box with a red light that will illuminate if *bad* should get set to `true`. These two boxes are defined as having identical behaviors until the red light comes on. Thus, collapsing the initial parts of the probabilities into the formal symbols $\Pr_1$ and $\Pr_2$, we may now say that

$\Pr_1[\mathcal{A}^{\pi(\cdot),\pi(K\oplus\cdot)} = 1 \mid \overline{\mathsf{BAD}}] = \Pr_2[\mathcal{A}^{\pi_1(\cdot),\pi_2(\cdot)} = 1 \mid \overline{\mathsf{BAD}}]$. And since Games 1 and 2 behave identically until *bad* becomes `true`, we know $\Pr_1[\mathsf{BAD}] = \Pr_2[\mathsf{BAD}]$. We have that

$$\begin{aligned}
&\left|\Pr_1[\mathcal{A}^{\pi(\cdot),\pi(K\oplus\cdot)} = 1] - \Pr_2[\mathcal{A}^{\pi_1(\cdot),\pi_2(\cdot)} = 1]\right| \quad = \\
&\quad \Big|\Pr_1[\mathcal{A}^{\pi(\cdot),\pi(K\oplus\cdot)} = 1 \mid \overline{\mathsf{BAD}}] \cdot \Pr_1[\overline{\mathsf{BAD}}] \quad + \quad \Pr_1[\mathcal{A}^{\pi(\cdot),\pi(K\oplus\cdot)} = 1 \mid \mathsf{BAD}] \cdot \Pr_1[\mathsf{BAD}] \quad - \\
&\qquad \Pr_2[\mathcal{A}^{\pi_1(\cdot),\pi_2(\cdot)} = 1 \mid \overline{\mathsf{BAD}}] \cdot \Pr_2[\overline{\mathsf{BAD}}] \quad - \quad \Pr_2[\mathcal{A}^{\pi_1(\cdot),\pi_2(\cdot)} = 1 \mid \mathsf{BAD}] \cdot \Pr_2[\mathsf{BAD}]\Big| .
\end{aligned}$$

From the previous assertions we know this is equal to

$$\begin{aligned}
& \left|\Pr_1[\mathcal{A}^{\pi(\cdot),\pi(K\oplus\cdot)} = 1 \mid \mathsf{BAD}] \cdot \Pr_1[\mathsf{BAD}] - \Pr_2[\mathcal{A}^{\pi_1(\cdot),\pi_2(\cdot)} = 1 \mid \mathsf{BAD}] \cdot \Pr_2[\mathsf{BAD}]\right| \\
= & \left|\Pr_2[\mathsf{BAD}] \cdot \left(\Pr_1[\mathcal{A}^{\pi(\cdot),\pi(K\oplus\cdot)} = 1 \mid \mathsf{BAD}] - \Pr_2[\mathcal{A}^{\pi_1(\cdot),\pi_2(\cdot)} = 1 \mid \mathsf{BAD}]\right)\right| \\
\le & \Pr_2[\mathsf{BAD}].
\end{aligned}$$

Therefore we may bound the adversary's advantage by bounding $\Pr_2[\mathsf{BAD}]$.

We define p events in Game 2: for $1 \le i \le p$, let BAD_i be the event that *bad* becomes `true`, for the first time, as a result of the ith query. Thus BAD is the disjoint union of $\mathsf{BAD}_1, \ldots, \mathsf{BAD}_p$ and

$$\Pr_2[\mathsf{BAD}] = \sum_{i=1}^{p} \Pr_2[\mathsf{BAD}_i] .$$

We now wish to bound $\Pr_2[\mathsf{BAD}_i]$. To do this, we first claim that adaptivity does not help the adversary to make BAD happen. In other words, the optimal adaptive strategy for making BAD happen is no better than the optimal non-adaptive one. Why is this? Adaptivity could help the adversary if Game 2 released information associated to K. But Game 2 has no dependency on K—the variable is not used in computing return values to the adversary. Thus Game 2 never provides the adversary any information that is relevant for creating a good ith query. So let $\mathcal{A}$ be an optimal adaptive adversary for making BAD happen. By the standard averaging argument there is no loss of generality to assume that $\mathcal{A}$ is deterministic. We can construct an optimal non-adaptive adversary $\mathcal{A}'$ by running $\mathcal{A}$ and simulating two independent permutation oracles. Since Game 2 returns values of the same distribution, for all K, the adversary $\mathcal{A}'$ will do no better or worse in getting BAD to happen in Game 2 if $\mathcal{A}'$ asks the sequence of questions that $\mathcal{A}$ asked in the simulated run. Adversary $\mathcal{A}'$ can now be made deterministic by the standard averaging argument. The resulting adversary, $\mathcal{A}''$, asks a fixed sequence of queries and yet does just as well as the adaptive adversary $\mathcal{A}$.

Now let us examine the chance that *bad* is set `true` in line 2, assuming the adversary has chosen her queries in advance. As we noted above, this can occur when the adversary asks a query X after having previously issued a query $K \oplus X$ to the *second* procedure. What is the chance that this occurs? We can view the experiment as follows: at most $i-1$ queries were asked of the second procedure; let's name those queries $Q = \{X_1, \cdots, X_{i-1}\}$. We wish to bound the chance that a randomly chosen X will cause $K \oplus X \in Q$. But this is simply asking the chance that $K \in \{X \oplus X_1, \cdots, X \oplus X_{i-1}\}$ which is at most $(i-1)/N$.

What is the chance that *bad* becomes `true` in line 4? This occurs when the randomly chosen Y is already in the range of π. Since Y is removed from S each time the first procedure returns it, this occurs when the *second* procedure returned such a Y and it was then subsequently chosen by the first procedure. Since at most $i-1$ values were returned by the second procedure, the chance is at most $(i-1)/N$ that this could occur.

The same arguments apply to lines 6 and 8. Therefore we have $\Pr_2[\mathsf{BAD}_i] \leq 2(i-1)/N$, and

$$\Pr_2[\mathsf{BAD}] \;\leq\; \sum_{i=1}^{p} \frac{2(i-1)}{N} \;\leq\; \frac{p^2}{N}$$

which completes the proof. ▮

We may now state and prove the theorem for the security of our XCBC construction. The bound follows quickly from the the bound on FCBC and the preceding lemma.

Theorem 6.5.2 [XCBC ≈ Rand] *Fix $n \geq 1$ and let $N = 2^n$. Let $\mathcal{A}$ be an adversary which asks at most q queries, each of which is at most mn bits. Assume $m \leq N/4$. Then*

$$\Big| \Pr[\pi_1 \stackrel{R}{\leftarrow} \mathrm{Perm}(n);\ K2, K3 \stackrel{R}{\leftarrow} \{0,1\}^n :\ \mathcal{A}^{\mathrm{XCBC}_{\pi_1,K2,K3}(\cdot)} = 1]$$

$$- \Pr[R \stackrel{R}{\leftarrow} \mathrm{Rand}(\{0,1\}^*, n) :\ \mathcal{A}^{R(\cdot)} = 1] \Big|$$

$$\leq\ \frac{q^2}{2} V_n(m,m) + \frac{(2m^2+1)q^2}{N} \ \leq\ \frac{(4m^2+1)q^2}{N}\,.$$ ▮

Proof:

One version of the triangle inequality states that for any real numbers a, b, c, $|a - b| \leq |a - c| + |c - b|$. We use this to show that the difference above (between XCBC and a random function) is at most the sum of the distance between XCBC and FCBC plus the distance between FCBC and a random function. So the above difference is at most

$$\Big| \Pr[\pi_1, \pi_2, \pi_3 \stackrel{R}{\leftarrow} \mathrm{Perm}(n) :\ \mathcal{A}^{\mathrm{FCBC}_{\pi_1,\pi_2,\pi_3}(\cdot)} = 1] \ - \ \Pr[R \stackrel{R}{\leftarrow} \mathrm{Rand}(\{0,1\}^*, n) :\ \mathcal{A}^{R(\cdot)} = 1] \Big|$$

$$+ \ \Big| \Pr[\pi_1, \pi_2, \pi_3 \stackrel{R}{\leftarrow} \mathrm{Perm}(n) :\ \mathcal{A}^{\mathrm{FCBC}_{\pi_1,\pi_2,\pi_3}(\cdot)} = 1]$$

$$- \ \Pr[\pi_1 \stackrel{R}{\leftarrow} \mathrm{Perm}(n);\ K2, K3 \stackrel{R}{\leftarrow} \{0,1\}^n :\ \mathcal{A}^{\mathrm{XCBC}_{\pi_1,K2,K3}(\cdot)} = 1] \Big|$$

and Theorem 6.4.1 gives us a bound on the first difference above. We now bound the second difference. Clearly this difference is at most

$$\Big| \Pr[\pi_1, \pi_2, \pi_3 \stackrel{R}{\leftarrow} \mathrm{Perm}(n) : \mathcal{A}^{\pi_1(\cdot),\pi_2(\cdot),\pi_3(\cdot)} = 1]$$

$$- \Pr[\pi_1 \stackrel{R}{\leftarrow} \mathrm{Perm}(n);\ K2, K3 \stackrel{R}{\leftarrow} \{0,1\}^n : \mathcal{A}^{\pi_1(\cdot),\pi_1(K2\oplus\cdot),\pi_1(K3\oplus\cdot)} = 1]\Big|$$

since any adversary which does well in the previous setting could be converted to one which does well in this setting. (Here we assume that $\mathcal{A}$ makes at most mq total queries of her oracles). Applying Lemma 6.5.1 twice we bound the above by $2m^2q^2/N$. Therefore our overall bound is

$$\frac{q^2}{2}V_n(m,m) + \frac{q^2}{2N} + \frac{2m^2q^2}{N} \le \frac{q^2}{2}V_n(m,m) + \frac{(2m^2+1)q^2}{N} .$$

And if we apply the bound on $V_n(m,m)$ from Lemma 6.3.2 we have

$$\frac{q^2}{2}\,\frac{4m^2}{N} + \frac{(2m^2+1)q^2}{N} \le \frac{(4m^2+1)q^2}{N}$$

as required. ∎

XOR-AFTER-LAST-ENCIPHERING DOES NOT WORK. We comment that the XCBC-variant that XORs the second key just *after* applying the final enciphering does *not* work. That is, when $|M|$ is a nonzero multiple of the blocksize, we'd have $\mathrm{MAC}_{\pi,K}(M) = \mathrm{CBC}_\pi(M) \oplus K$. This is no good. In the attack, the adversary asks for the MACs of three messages: the message $\mathbf{0} = 0^n$, the message $\mathbf{1} = 1^n$, and the message $\mathbf{1}\,.\,\mathbf{0}$. As a result of these three queries the adversary gets tag $T_0 = \pi(\mathbf{0}) \oplus K$, tag $T_1 = \pi(\mathbf{1}) \oplus K$, and tag $T_2 = \pi(\pi(\mathbf{1})) \oplus K$. But now the adversary knows the correct tag for the (unqueried) message $\mathbf{0}\,.\,(T_0 \oplus T_1)$, since this is just T_2: namely, $\mathrm{MAC}_{\pi,K}(\mathbf{0}\,.\,(T_0 \oplus T_1)) = \pi(\pi(\mathbf{0}) \oplus (\pi(\mathbf{0}) \oplus K) \oplus (\pi(\mathbf{1}) \oplus K)) \oplus K = \pi(\pi(\mathbf{1})) \oplus K = T_2$.

ON KEY-SEARCH ATTACKS. If FCBC or XCBC is used with an underlying block cipher (like DES) which is susceptible to exhaustive key search, then the MACs inherit this vulnerability. (The same can be said of ECBC and EMAC, except that the double encryption which these MACs employ would seem to necessitate a meet-in-the-middle attack, which uses a great deal of memory.) It was such considerations that led the designers of ISO 9797-1 to suggest triple encryption for enciphering the last block [15]. It would seem to be possible to gain this same exhaustive-key-search strengthening by modifying XCBC to *again* XOR the second key ($K2$ or $K3$) with the result of the last encipherment. (If one is using DES, this amounts to switching to DESX for the last encipherment [16].) We call this variant XCBCX. Likely one could prove good bounds for it in the Shannon model. However, none of this is necessary or relevant if one simply starts with a strong block cipher.

COMPLEXITY-THEORETIC RESULT. One can once again pass from the information-theoretic result to the complexity-theoretic one:

Corollary 6.5.3 [XMAC is a PRF] *Fix $n \geq 1$ and let $N = 2^n$. Let E: $\mathrm{Key} \times \{0,1\}^n \to \{0,1\}^n$ be a block cipher. Then*

$$\mathrm{Adv}^{\mathrm{prf}}_{\mathrm{XCBC}[E]}(t, q, mn) \;\leq\; \frac{4m^2q^2 + q^2}{N} + 3 \cdot \mathrm{Adv}^{\mathrm{prp}}_{E}(t', q'), \quad \textit{and}$$

$$\mathrm{Adv}^{\mathrm{mac}}_{\mathrm{XCBC}[E]}(t, q, mn) \;\leq\; \frac{4m^2q^2 + q^2 + 1}{N} + 3 \cdot \mathrm{Adv}^{\mathrm{prp}}_{E}(t', q')$$

where $t' = t + O(mq)$ and $q' = mq$. ▮

Chapter 7

The NH Hash Family

As we mentioned in Chapter 4, we can search for software-efficient Carter-Wegman MACs by finding efficient ϵ-AU hash families. In this chapter we describe the NH hash family introduced by Black, Halevi, Krawczyk, Krovetz, and Rogaway [6]. NH has a good compression ratio, small ϵ, and is very fast on modern processors. The authors go on to describe a complete MAC they call "UMAC" which will not be discussed here; they use the FCH construction from Section 4.3.

7.1 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}. For integers x, y let $(x \ +_w \ y)$ denote $(x + y) \bmod 2^w$.

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

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

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. For a pictorial representation, see Figure 7.1 on page 106. 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 $\mathsf{NH}[n, w]$.

7.2 Analysis

We use a slight variant of the usual concept of a ϵ-AU hash function. Instead of just proving that NH is ϵ-AU over all of its domain, we will prove that it is instead "ϵ-AU on equal-length strings."

Definition 7.2.1 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-AU on equal-length strings if for all distinct, equal-length strings $M, M' \in A$, we have that $\Pr_{h \leftarrow \mathsf{H}}[h(M) = h(M')] \leq \epsilon$. ▮

The following theorem bounds the collision probability of NH.

Theorem 7.2.2 *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 $n \geq 2$, and 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 7.2.3 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 7.6.1.

Lemma 7.2.3 *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

$$k(c - c') = m'c' + y \quad \text{and} \quad k < 2^w - m' \tag{7.1}$$

$$k(c - c') = (m' - 2^w)c' + y \quad \text{and} \quad k \geq 2^w - m' \tag{7.2}$$

A simple lemma (Lemma 7.2.4, 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 Equation (7.1) and $k = k_2 \in U_w$ satisfying Equation (7.2) 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

$$(k_2 - k_1)(c' - c) = 2^w c' \tag{7.3}$$

We show Equation (7.3) 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 Equation (7.3) 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 7.2.4 *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 7.2.2 is tight: let $M = 0^w0^w$ and $M' = 1^w0^w$ and note that any key $K = K_1K_2$ with $K_2 = 0^w$ causes a collision.

Although we do not require any stronger properties than the above, NH is actually 2^{-w}-ΔAU under the operation of addition modulo 2^{2w}; only trivial modifications to the above proof are required. See [17] for a definition of ϵ-ΔAU.

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).

7.3 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.

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}. We are still doing "native" moduli (ie, moduli are the normal machine operations). 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 7.3.1 *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 7.3.1 is identical to the proof for Theorem 7.2.2 with two exceptions: instead of U_w we take elements from S_w, and in the end we use Lemma 7.3.3, 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 7.3.3 we begin by restating Lemma 7.2.4 for the signed case. Recall that $D_w = \{-2^w+1, \ldots, 2^w-1\}$.

Lemma 7.3.2 *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 7.2.4 depends only on D_w, we may recycle the proof without modification. ▮

Lemma 7.3.3 *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 7.2.3, 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 in S_w 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 7.3.2 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 7.3.2 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 7.3.4 *Let $m_1 = m_2 = 0$ and let $m'_1 = m'_2 = -2^{w-1}$. 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

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

has at least $2^{w+1} - 2$ solutions in $k_1, k_2 \in S_w$. This will imply the theorem. 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. ∎

7.4 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. For a description of methods to correct the remaining deficiencies, see [6].

7.5 The Toeplitz Approach

To reduce the collision probability for NH or NHS, 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, from Proposition 4.1.5 we know that if we concatenate the results from, say, four independent instances of a hash function family with collision probability 2^{-w}, the collision probability drops 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. This is similar to Krawczyk's approach mentioned in Section 4.2.

For example, to reduce the collision probability to 2^{-64} for $\mathsf{NH}[n, 16]$, we can choose an underlying hash key $K = K_1 \,.\, \cdots \,.\, K_{n+6}$ and then hash with the four derived keys $K_1 \,.\, \cdots \,.\, K_n$, $K_3 \,.\, \cdots \,.\, K_{n+2}$, $K_5 \,.\, \cdots \,.\, K_{n+4}$, and $K_7 \,.\, \cdots \,.\, K_{n+6}$. 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 (eg,, [19]), they only apply to hashing schemes over fields. Instead, NH operates over a combination of rings ($Z/2^w$ and $Z/2^{2w}$). In Theorem 7.6.1 we prove that the Toeplitz construction nonetheless achieves the desired bound in the case of NH, and in Theorem 7.7.1 we prove this for NHS as well.

7.6 The Unsigned Case

We define the hash-function family $\mathsf{NH}^\mathsf{T}[n,w,t]$ ("Toeplitz-NH") as follows. Fix an even $n \geq 2$, $w \geq 1$, and $t \geq 1$ (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 $K = K_1 . \cdots . K_{n+2(t-1)}$ (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 7.6.1 *For any $w, t \geq 1$ and any even $n \geq 2$, $\mathsf{NH}^\mathsf{T}[n,w,t]$ 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 $|M| = |M'| = w\ell$ where $\ell \leq n$. We are required to show

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

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

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

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

$$\Pr_{K \leftarrow \mathsf{NH}^\mathsf{T}}[E_1 \wedge E_2 \wedge \cdots \wedge E_t]. \tag{7.4}$$

Call each term in the summations of 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 Equation (7.4) as Equality E_j.

The idea here is that each Equality represents a collision of M and M' under a particular position of the key. Since the key is being shifted, we are essentially hashing M and M' under many different positions of the key. In order for a collision to occur in the big picture, we require that collisions occur for every position of the key and therefore each Equality $E_1, \cdots, E_t$ must be simultaneously true.

Without loss of generality, we can assume that M and M' disagree in the last clause (ie,, that $m_{\ell-1} \neq m'_{\ell-1}$ or $m_\ell \neq m'_\ell$). 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 $m_{\ell-1} \neq m'_{\ell-1}$. (The proof may easily be restated if we instead have $m_\ell \neq m'_\ell$. This case is symmetric due to the fact we shift the key by two words.) We proceed by proving that for all $j \in \{1, \ldots, t\}$, $\Pr[E_j \text{ is true} \mid E_1, \ldots, E_{j-1} \text{ are true}] \leq 2^{-w}$, implying the theorem.

For E_1, the claim is satisfied due to Theorem 7.2.2. 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 7.2.2. 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 $m_{\ell-1} \neq m'_{\ell-1}$ we know $c \neq c'$; clearly $m_\ell, m'_\ell, c, c' \in U_w$ so Lemma 7.2.3 tells us there is at most one value of $k_{\ell+2j-2}$ satisfying this equation, which completes the proof. ▮

7.7 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 7.6.1 for the signed case.

Theorem 7.7.1 *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 7.6.1 except we use Theorem 7.3.1 in place of Theorem 7.2.2 and Lemma 7.3.3 in place of Lemma 7.2.3. Then, using the same notation as Theorem 7.6.1, $\Pr_{K \leftarrow \mathsf{NHS}^{\mathsf{T}}}[E_j \mid E_1, \ldots, E_{j-1}] \leq 2^{-w+1}$, 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. ▮

7.8 Performance

The NH hash family is very fast: the operations we perform are simple and typically take only a few cycles each on a modern CPU. There are no moduli by primes (which often requires an expensive division operation). In addition, we are able to exploit certain SIMD architectures to go even faster. We now discuss these issues.

Basic NH. The preferred embodiment for NH on a processor without SIMD instructions is shown in Figure 7.1. The only requirement from the architecture is that we have a multiplication instruction which can multiply two 32-bit words to a 64-bit result (or more generally, the architecture enables two machine-words to be multiplied such that we obtain the full result). This assumption was also made in [14].

Exploiting SIMD. Certain architectures, such as Motorola's AltiVec and Intel's MMX architectures, provisions are made for multimedia support. Typically a "vector" datatype is made available, and certain vector operations such as vector-addition and vector-dot-product are fast single instructions. The intent here is to make sound and graphics efficient

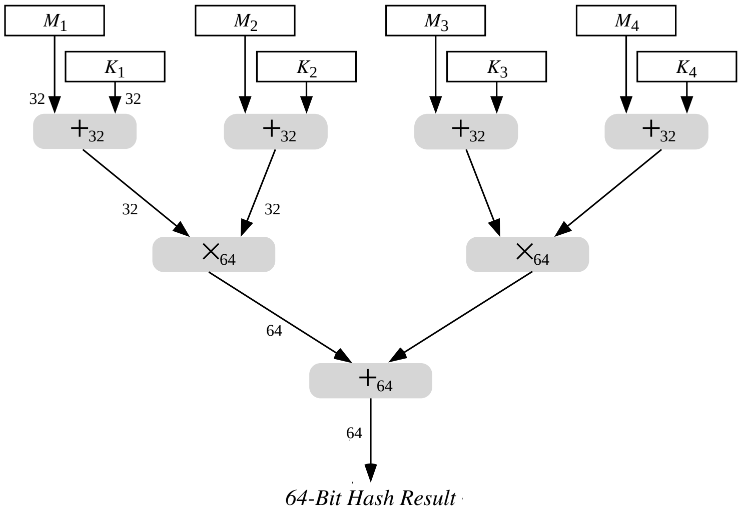

Figure 7.1: The basic NH. In this example, the word size w is 32 bits. Here we use four 32-bit additions, two 32-bit to 64-bit multiplications, and one 64-bit addition to hash our message.

(for example, to transform a point in 3-space we often multiply the 3-coordinate vector which represents the point by a 4×4 matrix). A nice feature of NH is that these instructions can be appropriated for its implementation as well. See Figure 7.2. It is quite remarkable that only three MMX instructions are needed to implement the operation depicted. These three instructions are two vector additions (`paddwd`) and one dot-product (`pmaddwd`). There is a slight amount of additional overhead to add the results into a single 32-bit result at the end as well. As one might guess, this SIMD version of NH is the fastest implementation we tested.

COMMENTS. We should note that the collision probability for the "basic" NH with $w = 32$ listed above is either 2^{-32} or 2^{-31} depending on whether unsigned or signed integers are used, respectively. For the SIMD version using Intel's MMX, the collision probability is 2^{-15} since signed integers are the only option for this instruction set. Therefore, in spite

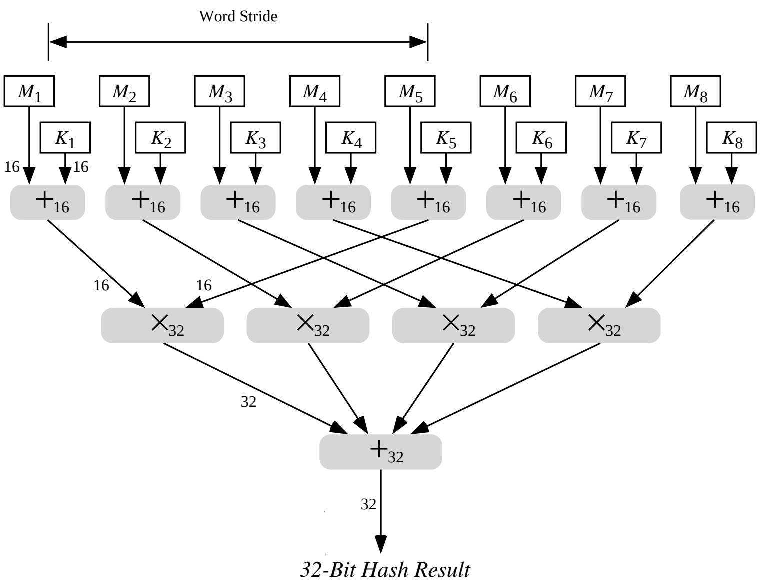

Figure 7.2: The SIMD version of NH. In this example, the word size w is 16 bits. For the Intel MMX instruction set we can implement this picture with three MMX instructions.

of the fact we get extreme speeds for the MMX case, we will most likely have to use our Toeplitz extension to reduce the collision probability which of course costs us in terms of performance. Fortunately, there are ways to efficiently code the Toeplitz-extended hash family so that using a Toeplitz iteration count of 2 does not actually cost us twice as much time as a single NH invocation.

TIMINGS. In the UMAC paper, the authors implement four UMAC variants using four different parameters for NH. These are as follows:

- UMAC-STD-30 –This is basic NHS with $w = 32$.
- UMAC-STD-60 –This is basic NHS with $w = 32$ and a Toeplitz iteration count of 2.
- UMAC-MMX-30 –This is the Intel MMX version of NH with $w = 16$ and a Toeplitz iteration count of 2.

- UMAC-MMX-60 –This is the Intel MMX version of NH with $w = 16$ and a Toeplitz iteration count of 4.

Although UMAC is slightly slower than the underlying NH hash, for reasonably long messages the time to hash dominates the time to MAC. Therefore we examine the speed of these UMAC versions to get an idea of how fast NH is under the options listed above.

The tests were implemented 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 NH but also is an indication that it will work well on future architectures. The SIMD operations in the MMX and AltiVec equipped processors and the 64-bit register size of the Alpha are both features expected to become more prevalent in future processors.

All tests were written in C with a few functions written in assembly. For the Pentium II assembly was written for RC6, SHA-1, and the NH hash code. For the PowerPC 604e assembly was written for just NH. 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 [11] and the CBC-MAC of a fast block cipher, RC6 [25].

	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 7.3: UMAC Performance. *Peak performance for three architectures measured in Gbits/sec (and cycles/byte). The Gbits/sec numbers are normalized to 350 MHz.*

The graph in Figures 7.3 and 7.4 show the throughput of the indicated versions of UMAC, as well as HMAC-SHA1 (a very fast software-based MAC) and CBC-MAC-RC6, all run on a 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.

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. Another reason UMAC-MMX-60 does less wellon 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. Perhaps the most surprising experimental finding was how, on some processors, performance could be

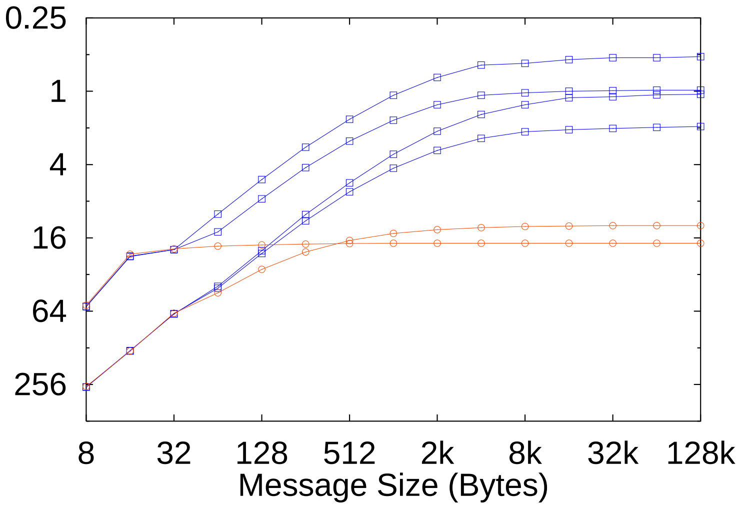

Figure 7.4: Log-scale graph of performance over various message lengths on a Pentium II, measured in machine cycles/byte. The lines in the graph correspond to the following MACs (beginning at the top-right and moving downward): UMAC-MMX-30, UMAC-MMX-60, UMAC-STD-30, UMAC-STD-60, HMAC-SHA1 and CBC-MAC-RC6.

dramatically improved 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.

Bibliography

[1] ANSI X9.9 (REVISED). American national standard–financial institution message authentication (wholesale). Tech. rep., ASC X9 Secretariat–American Bankers Assocation, 1986. (Replaces X9.9–1982).

[2] Bellare, M., Guérin, R., and Rogaway, P. XOR MACs: New methods for message authentication using finite pseudorandom functions. In *Advances in Cryptology – CRYPTO '95* (1995), vol. 963 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 15–28.

[3] Bellare, M., Kilian, J., and Rogaway, P. The security of the cipher block chaining message authentication code. See `www.cs.ucdavis.edu/~rogaway`. Older version appears in *Advances in Cryptology – CRYPTO '94* (1994), vol. 839 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 341–358.

[4] Berendschot, A., den Boer, B., Boly, J., Bosselaers, A., Brandt, J., Chaum, D., Damgård, I., Dichtl, M., Fumy, W., van der Ham, M., Jansen, C., Landrock, P., Preneel, B., Roelofsen, G., de Rooij, P., and Vandewalle, J. *Final Report of Race Integrity Primitives*, vol. 1007 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[5] Bernstein, D. How to stretch random functions: The security of protected counter sums. *Journal of Cryptography 12*, 3 (July 1999).

[6] Black, J., Halevi, S., Krawczyk, H., Krovetz, T., and Rogaway, P. UMAC: Fast and secure message authentication. In *Advances in Cryptology – CRYPTO '99* (1999), Lecture Notes in Computer Science, Springer-Verlag. Full version of this paper available at `www.cs.ucdavis.edu/~rogaway/umac`.

[7] Brassard, G. On computationally secure authentication tags requiring short secret shared keys. In *Advances in Cryptology–CRYPTO 82* (New York, 1983), R. L. Rivest, A. Sherman, and D. Chaum, Eds., Plenum Press, pp. 79–86.

[8] Campbell, C. Design and specification of cryptographic capabilities. In *Computer security and the data encryption standard, NBS Special Publication 500-27*, D. Barnstad, Ed. National Bureau of Standards, Washington, D.C., 1977, pp. 54–66.

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

[10] FIPS 113. Computer data authentication. Federal Information Processing Standards Publication 113, U.S. Department of Commerce/National Bureau of Standards, National Technical Information Service, Springfield, Virginia, 1994.

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

[12] Goldreich, O., Goldwasser, S., and Micali, S. How to construct random functions. *Journal of the ACM 33*, 4 (Oct. 1984), 792–807.

[13] Goldwasser, S., Micali, S., and Rivest, R. L. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing 17*, 2 (Apr. 1988), 281–308.

[14] 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), Springer-Verlag.

[15] Iso/Iec 9797-1. Information technology – security techniques – data integrity mechanism using a cryptographic check function employing a block cipher algorithm. International Organization for Standards, Geneva, Switzerland, 1999. Second edition.

[16] Kilian, J., and Rogaway, P. How to protect DES against exhaustive key search. In *Advances in Cryptology – CRYPTO '96* (1996), vol. 1109 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 252–267.

[17] 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.

[18] Lai, X., and Massey, J. A proposal for a new block encryption standard. In *Advances in Cryptology–EUROCRYPT 90* (1990), I. Damgård, Ed., Springer-Verlag, pp. 389–404. Lecture Notes in Computer Science No. 473.

[19] 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.

[20] Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. *Handbook of Applied Cryptography.* CRC Press, 1997.

[21] National Bureau of Standards. Announcing the data encryption standard. Tech. Rep. FIPS Publication 46, National Bureau of Standards, Jan. 1977.

[22] Petrank, E., and Rackoff, C. CBC MAC for real-time data sources, 1998. Available from `http://philby.ucsd.edu/cryptolib`.

[23] 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.

[24] 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.

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

[26] Rivest, R. L. Cryptography. In *Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity)*, J. van Leeuwen, Ed. Elsevier and MIT Press, 1990, ch. 13, pp. 717–755.

[27] Rogaway, P. Bucket hashing and its application to fast message authentication. *Journal of Cryptography 12* (1999), 91–115.

[28] 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.

[29] Stinson, D. Universal hashing and authentication codes. In *Advances in Cryptology–CRYPTO 91* (1992), J. Feigenbaum, Ed., Springer, pp. 74–85. Lecture Notes in Computer Science No. 576.

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

Appendix A

Birthday Bounds

The following discussion was written by Mihir Bellare (taken from Appendix A of [3]), with some additional details inserted.

We consider the classical "occupancy problem" of throwing q balls at random into N bins. The i-th ball is equally likely to land in bins $1, 2, \cdots, N$. We denote by $C(N, q)$ the probability that at least two balls end up in the same bin after all q have been thrown.

Before stating and proving our main theorem, we establish some basic inequalities which will prove useful in that proof.

Proposition A.0.1 *For* $x \in [0, 1]$,

$$\left(1 - \frac{1}{e}\right) x \leq 1 - e^{-x} \leq x.$$

Proof: We first consider the right-hand inequality. The Taylor expansion for e^{-x} is

$$e^{-x} = 1 - x + x^2/2! - x^3/3! + \cdots .$$

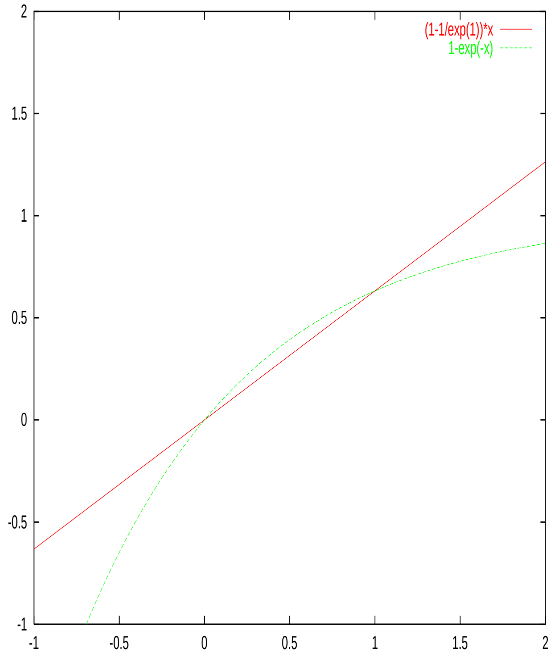

Figure A.1: A plot of $(1-1/e)x$ and $1-e^{-x}$. The former is smaller on the interval $[0,1]$.

Since we are guaranteed $x \in [0,1]$, we can be sure the sum $x^2/2! - x^3/3! + \cdots$ is nonnegative and infer that $e^{-x} \geq 1 - x$. This immediately yields $1 - e^{-x} \leq x$, as desired. (Note that this inequality is actually true for *all* real x, though we don't require it.)

The left-hand inequality is a bit more involved. We wish to show $(1-1/e)x \leq 1 - e^{-x}$ on the interval $[0,1]$. Note that we have equality for $x = 0$ and $x = 1$. Also note that $(1-1/e)x$ is a line and that $1 - e^{-x}$ is concave down on $[0,1]$. (To verify this, take the second derivative of $1-e^{-x}$, which is $-e^{-x}$, and note that it is always negative.) There is only one way for a curve to coincide with a line in two points and yet be concave down, and that is if the curve exceeds the line between these two points. ▮

Now we proceed to the main result of this section. We bound $C(N,q)$ above and below.

Theorem A.0.2 *Let $C(N, q)$ be the probability that at least two balls occupy the same bin when we throw $q \geq 1$ balls randomly into $N \geq q$ bins. Then*

$$1 - e^{-q(q-1)/2N} \;\leq\; C(N,q) \;\leq\; \frac{q(q-1)}{2N},$$

and for $1 \leq q \leq \sqrt{2N}$,

$$0.316\frac{q(q-1)}{N} \;\leq\; C(N,q) \;\leq\; 0.5\frac{q(q-1)}{N}.$$

Proof: The upper bound is easy. Define the event C_i to mean that the i-th ball lands in an occupied bin. Using the sum bound, we can proceed as follows:

$$\begin{aligned}
C(N,q) &= \Pr[C_1 \vee C_2 \vee \cdots \vee C_q] \\
&\leq \Pr[C_1] + \Pr[C_2] + \cdots + \Pr[C_q] \\
&\leq 0/N + 1/N + \cdots + (q-1)/N \\
&= \frac{q(q-1)}{2N}.
\end{aligned}$$

The lower bound is a bit harder. Let the event D_i indicate that there is no collision among the first i balls. It's easy to compute $\Pr[D_{i+1} \mid D_i]$: since exactly i bins are occupied the chance that we will miss all of them is exactly $(N-i)/N$. The probability we're interested in is the opposite of D_q (ie, that some two balls *do* collide). We compute

$$\begin{aligned}
1 - C(N,q) &= \Pr[D_q] \\
&= \Pr[D_q \mid D_{q-1}]\ \Pr[D_{q-1} \mid D_{q-2}]\ \cdots \Pr[D_1]
\end{aligned}$$

and since $\Pr[D_1] = 1$,

$$= \prod_{i=1}^{q-1} \Pr[D_{i+1} \mid D_i]$$

$$= \prod_{i=1}^{q-1} \left(1 - \frac{i}{N}\right).$$

We now use Proposition A.0.1 to substitute $e^{-i/N}$ for $(1 - i/N)$ above. And so the above product is at most

$$\prod_{i=1}^{q-1} e^{-i/N} = e^{-1/N-2/N-\cdots-q(q-1)/N} = e^{-q(q-1)/2N}.$$

This completes the first of the lower bounds in the theorem statement. Next we require the condition that $q \leq \sqrt{2N}$. This means $q(q-1)/2N \leq 1$ and so we can use the other piece of the inequality from Proposition A.0.1 to get

$$C(N, q) \geq \left(1 - \frac{1}{e}\right) \frac{q(q-1)}{2N}.$$

Since $(1 - 1/e)/2 > 0.316$, we have our result. ▮