MAC Reforgeability

J. Black [*] M. Cochran [*]

Abstract

Message Authentication Codes (MACs) are a widely-used building-block for secure protocols. Their security is given by a bound on the probability that any adversary can forge messages after at most q calls to an oracle that produces MAC tags for the adversary's queries. However, in certain settings we may wish to know what happens *after* a forgery has occurred. Does the MAC remain relatively secure, or can an adversary efficiently produce further forgeries?

In this paper, we examine the notion of "reforgeability" for MACs. We first give a definition for this new notion, then examine some of the most widely-used and well-known MACs under our definition. We show that for each of these MACs there exists an attack that allows efficient forgeries after the first one is obtained, and we show that simply making these schemes stateful is insufficient. Finally, we exhibit a new scheme, FCHC, which is provably-secure against reforging attacks.

Keywords: Message Authentication Codes, Birthday Attacks, Provable Security.

[*] Department of Computer Science, 430 UCB, Boulder, Colorado 80309 USA. E-mail: jrblack@cs.colorado.edu, Martin.Cochran@colorado.edu WWW: www.cs.colorado.edu/~jrblack/, ucsu.colorado.edu/~cochranm

Contents

1 Introduction **1**

2 Preliminaries **3**

3 Attacks **5**
3.1 Blockcipher Based MACs 6
3.2 Padding Attacks 7
3.3 Effects of Adding State 8
3.4 Attacks on Universal Hash Function Families 9

4 Schemes Resistant to Reforgeability **11**

A Attacks **14**
A.1 Blockcipher-Based MACs 14
A.2 Attacks on Universal Hash Families 14
A.3 Effects of Adding State 17

B Details of the hash127 Attack **17**

C Proof of Theorem 8 **18**

1 Introduction

Message Authentication Codes. Message authentication codes (MACs) are a widely-used means to guarantee message authenticity and integrity in the symmetric-key setting. They work like this: if Alice wishes to send a message M to Bob, she processes M with an algorithm MAC using her shared key K and possibly some state or random bits we denote with s. This produces a short string Tag and she then sends (M, s, Tag) to Bob. Bob runs a verification algorithm VF with key K on the received tuple and VF outputs either ACCEPT or REJECT. The goal is that Bob should virtually never see ACCEPT unless (M, s, Tag) was truly generated by Alice; that is, an imposter should not be able to impersonate Alice and forge valid tuples.

There are a large number of MACs in the literature. Most have a proof of security where security is expressed as a bound on the probability that an attacker will succeed in producing a forgery after making q queries to an oracle that produces MAC tags on messages of his choice. The bound usually contains a term $q^2/2^t$ where q is the total number of tags generated under a given key and t is the tag length in bits. This quadratic term typically comes from the probability that two identical tags were generated by the scheme for two different messages; this event is typically called a "collision" and once it occurs the analysis of the scheme's security no longer holds. The well-known birthday phenomenon is responsible for the quadratic term: if we generate q random uniform t-bit strings independently, the expected value of q when the first collision occurs is about $\sqrt{\pi 2^{t-1}} = \Theta(2^{t/2})$.

Reforgeability. The following is a natural question: if a forgery is observed or constructed by an adversary, what are the consequences? One possibility is that this forgery does not lead to any additional advantage for the adversary: a second forgery requires nearly as much effort to obtain as the first one did. We might imagine using a random function $f : \Sigma^* \to \{0,1\}^t$ as a stateless MAC. Here, knowing a forgery amounts to knowing distinct $M_1, M_2 \in \Sigma^*$ with $f(M_1) = f(M_2)$. However it is obvious this leads to no further advantage for the adversary: the value of f at points M_1 and M_2 are independent of the values of f on all remaining unqueried points.

Practical MAC schemes, however, usually do not come close to truly random functions, even when implemented as pseudorandom functions (PRFs). Instead they typically contain structure that allows the adversary to use the obtained collision to infer information about the inner state of the algorithm. This invariably leads to further forgeries with a minimum of computation.

Applications. One might reasonably ask why we care about reforgeability. After all, aren't MACs designed so that the first forgery is extremely improbable? They are, in most cases, but there are several reasons why we might want to think about reforgeability nonetheless.

It is true that some MACs have such small forgery bounds that it is irrelevant to speak of even one forgery, from a practical standpoint. For example, a MAC with a forgery bound of $q^2/2^{128}$ guarantees that forgeries will occur with probability smaller than 2^{-64} provided no more than about 4 billion messages are processed with a given MAC key. For most settings this is more than enough assurance. But there are other MAC schemes with much larger bounds; for example CBC MAC using triple-DES outputs only 64 bits. To keep the probability of forgery below 2^{-16} we must refrain from MACing more than about 2^{24} messages under a given MAC key. One can easily imagine applications where messages are MACed at a sufficiently high rate that 2^{24} would not be such a large number (for example, a busy access point using a single key for all associations). And a 2^{-16} bound is not all that reassuring.

It might therefore be reasonable to consider the question of reforgeability in this context: if a tag collision occurs or a forgery is obtained, do the floodgates open or is it just an isolated event?

Other applications might intentionally employ a low-security MAC. In sensor nodes, where radio power is far more costly than computing power, short tag-length MACs might be employed to reduce the overhead of sending tags. Of course here we have to accept the risk that reduced security implies we might see some forgeries, but we would want to limit the extent to which forgeries could be generated.

In streaming video applications we might use a low-security MAC with the idea that forging one frame would hardly be noticable to the viewer; our concern would be that the attacker would be unable to efficiently forge arbitrarily many frames, thereby taking over the video transmission.

MAC Scheme	Expected Queries for j Forgeries	Succumbs to Padding Attack	Succumbs to Other Attack	Insecure even when Stateful
CBC MAC	$C_1 + j$		$\surd$	$\surd$
EMAC	$C_1 + j$	$\surd$	$\surd$	$\surd$
XCBC	$C_1 + j$	$\surd$	$\surd$	$\surd$
PMAC	$C_1 + j$		$\surd$	$\surd$
ANSI retail MAC	$C_1 + j$	$\surd$	$\surd$	$\surd$
HMAC	$\sum_i C_i/2^i + j$	$\surd$		$\surd$

UHF in FH mode	Expected Queries for j Forgeries	Reveals Key	Queries for key Recovery
hash127	$C_1 + j$	$\surd$	$C_1 + \log m$
Square Hash	$C_1 + j$	$\surd$	mC_1
LFSR-based Topelitz Hash	$C_1 + j$		
Bucket Hash	$C_1 + j$		
MMH	$C_1 + j$		
NMH	$C_1 + j$		

Figure 1: Summary of Results. The upper table lists each well-known MAC scheme we examined, along with its resistance to reforgeability attacks. Here n is the output length (in bits) of each scheme, and m is the length (in n-bit blocks) of the queries to the MAC oracle; the i-th collision among the tags is denoted by event C_i. For most schemes, the first forgery is made after the first collision among the tags, and each subsequent forgery requires only one further MAC query. For most MAC schemes listed above, the first collision is expected at around $2^{n/2}$ MAC queries, although the exact bound for each scheme differs somewhat. The lower table lists 6 well-known universal hash families, each made into a MAC via the FH construction. These similarly succumb to reforgeability attacks after a collision in the output tags, with hash127 and Square-Hash surrendering their key in the process. The last column gives the expected number of queries for key recovery, where possible.

Finally, the question seems a natural one and answering it should help lend a deeper understanding about one of the fundamental objects in cryptology. The fact that the question of reforgeability has recently arisen in newsgroups, online discussions, and conference hallways lends support to this.

Main Results. In this paper we conduct a systematic study of reforgeability. We first give a definition of reforgeability, both in the stateless and stateful settings. We then examine a variety of well-known MAC schemes and assess their resistance to reforgeability attacks. We find that in every case there exists an attack that enables efficient generation of forgeries given knowledge of an existing one. In some cases this involves fairly constrained modification of just the final block of some fixed message; in other cases we obtain the MAC key and have free rein.

Figure 1 gives a synopsis of our findings. In every case, our attack is based on finding collisions and this in turn leads to a substantial number of subsequent forgeries; the degree to which each scheme breaks is noted in the table and below. We briefly summarize the attacks.

- **CBC MAC.** We show that after an initial collision between two m-block messages, we can forge arbitrary m-block messages where the first two blocks are identical to those of the colliding messages, but the last $m-2$ blocks can be chosen arbitrarily.
- **EMAC [6], XCBC [11], ANSI Retail MAC [1], HMAC [2].** The first three schemes are variants of the basic CBC MAC and succumb to the same attack just mentioned. Additionally all four of these MACs allow varying-length messages (unlike the basic CBC MAC) and therefore admit an additional attack, the “Padding Attack” [26] that allows arbitrary blocks to be appended to colliding pairs at the cost of one additional MAC query.
- **PMAC [12].** For PMAC the best attack we found was quite limited: given a colliding pair of messages, we can arbitrarily alter the last block of one message and produce a forgery after a single additional MAC query using the other.

- **hash127 [8].** Hash127 is a polynomial-hash based on evaluating polynomials over the field $\mathbb{Z}$ mod $2^{127} - 1$. Any collision among tags is catastrophic: given two colliding messages their difference produces a polynomial whose roots include the hash key. Finding roots of polynomials over a finite field is computationally efficient using Berlekamp's algorithm [7] or the Cantor-Zassenhaus algorithm [14].

- **Square Hash [18].** Square-Hash is another fast-to-compute universal hash function family suggested for use in MACs. Once again, any tag collision results in an efficient algorithm that derives the hash key. The attack is specific to the Square-Hash function and we specify it in Section 3.4 where the scheme is described in full.

- **Remaining UHFs.** For each of the remaining universal hash function families we examine [20, 22, 28] we similarly show that collisions in the tag lead to further forgeries for the MAC scheme, provided we use the FH construction that composes a PRF (or PRP) with a member of the hash family. (If a PRF is used, our attacks work only if the tag collision occurs in the underlying universal hash function. This can be efficiently detected.) The idea that multiple forgeries can be obtained after one collision in Carter-Wegman style MACs is not new [29].

Note that we are not claiming the attacks given above are the best possible: there may be even more damaging attacks. But these were sufficient to make us wonder if there exists an efficient and practical MAC scheme resistant to reforgeability attacks. A natural first try is to add state, in the form of a counter inserted in a natural manner, to the schemes above. We show, however, that this approach is insufficient. We therefore devised a new (stateful) scheme, FCHC, that allows a single forgery once a collision is observed, but for which is it nearly as difficult to forge again as it was to forge initially. The scheme is described fully in Section 4 but briefly it works as follows.

Let $\mathcal{H}$ be some $\epsilon-$AU hash family $\mathcal{H} = \{h : D \rightarrow \{0,1\}^l\}$, and $\mathcal{R}$ a set of functions $\mathcal{R} = \text{Rand}(l + b, L)$. Let $\rho_0, \rho_1 \xleftarrow{\$} \mathcal{R}$; the shared key is (ρ_0, ρ_1). Let $\langle cnt \rangle_b$ denote the encoding of cnt using b bits. To MAC a message (M, cnt), the signer first ensures that $cnt < 2^b - 1$ and if so sends $(cnt, \rho_0(\langle cnt \rangle_b \parallel h_{cnt}(M)))$, where h_{cnt} is selected by the first s bits of $\rho_1(\langle cnt \rangle_b \parallel 10^{l-1})$. To verify a received message M with tag (i, t), the verifier computes $\rho_0(\langle i \rangle_b \parallel h_i(M))$ and ensures it equals t.

The scheme is not quite as efficient as we might hope: we must invoke two separate PRFs along with a member of our universal hash family. The PRF keys are fixed, but the hash function key varies per message and for some families this could be expensive in practice. Nonetheless, FCHC might be of use in settings where, say, short tags are required but reforgeability cannot be tolerated. An in-depth discussion of the security properties and tradeoffs of this scheme can be found in section 4.

Related Work. David McGrew and Scott Fluher have recently done some work [23] on a similar subject. They also examine MACs with regard to multiple forgeries, although they view the subject from a different angle. They show that for HMAC, CBC MAC, and Galois Counter Mode (GCM) of operation for blockciphers, reforgeability is possible. However, they examine reforgeability in terms of the number of expected forgeries (parameterized by the number of queries) for each scheme, which is dependent on the precise security bounds for the respective MACs. We instead focus not on the fact that reforgeability is possible with existing MACs (although we demonstrate that this is in fact the case for a large number of well-known MACs), but we examine why this is so and look at different approaches to MACs which may avoid some of these properties.

2 Preliminaries

Let $\{0,1\}^n$ denote the set of all binary strings of length n. For an alphabet Σ, let Σ^* denote the set of all strings with elements from Σ. Let $\Sigma^+ = \Sigma^* - \{\epsilon\}$ where ϵ denotes the empty string. For strings s, t, let $s \parallel t$ denote the concatenation of s and t. For set S, let $s \xleftarrow{\$} S$ denote the act of selecting a member s of S according to a probability distribution on S. Unless noted otherwise, the distribution is uniform. For a binary string s let $|s|$ denote the length of s. For a string s where $|s|$ is a multiple of n, let $|s|_n$ denote $|s|/n$. Unless otherwise noted, given binary strings s, t such that $|s| = |t|$, let $s \oplus t$ denote the bitwise XOR of s and t. For a string M such that $|M|$ is a multiple of n, $|M|_n = m$, then we will use the notation $M = M_1 \parallel M_2 \parallel \ldots \parallel M_m$ such that $|M_1| = |M_2| = \ldots = |M_m|$. Let $\mathrm{Rand}(l, L) = \{f \mid f : \{0,1\}^l \to \{0,1\}^L\}$ denote the set of all functions from $\{0,1\}^l$ to $\{0,1\}^L$.

Universal Hash Families. Universal hash families are used frequently in the cryptographic literature. We now define several notions needed later.

Definition 1 (Carter and Wegman [15]) Fix a domain $\mathcal{D}$ and range $\mathcal{R}$. A finite multiset of hash functions $\mathcal{H} = \{h : \mathcal{D} \to \mathcal{R}\}$ is said to be **Universal** if for every $x, y \in \mathcal{D}$ with $x \neq y$, $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] = 1/|\mathcal{R}|$.

Definition 2 Let $\epsilon \in \mathbb{R}^+$ and fix a domain $\mathcal{D}$ and range $\mathcal{R}$. A finite multiset of hash functions $\mathcal{H} = \{h : \mathcal{D} \to \mathcal{R}\}$ is said to be ϵ-**Almost Universal** (ϵ-AU) if for every $x, y \in \mathcal{D}$ with $x \neq y$, $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq \epsilon$.

Definition 3 (Krawczyk [22], Stinson [30]) Let $\epsilon \in \mathbb{R}^+$ and fix a domain $\mathcal{D}$ and range $\mathcal{R} \subseteq \{0,1\}^r$ for some $r \in \mathbb{Z}^+$. A finite multiset of hash functions $\mathcal{H} = \{h : \mathcal{D} \to \mathcal{R}\}$ is said to be ϵ-**Almost XOR Universal** (ϵ-AXU) if for every $x, y \in \mathcal{D}$ and $z \in \mathcal{R}$ with $x \neq y$, $\Pr_{h \in \mathcal{H}}[h(x) \oplus h(y) = z] \leq \epsilon$.

Message Authentication. Formally, a (stateless) message authentication code is a pair of algorithms, (MAC, VF), where MAC is a 'MACing' algorithm that, upon input of key $K \in \mathcal{K}$ for some key space $\mathcal{K}$, and a message $M \in \mathcal{D}$ for some domain $\mathcal{D}$, computes a τ-bit tag Tag; we denote this by $\mathrm{Tag} = \mathrm{MAC}_K(M)$. Algorithm VF is the 'verification' algorithm such that on input $K \in \mathcal{K}$, $M \in \mathcal{D}$, and $\mathrm{Tag} \in \{0,1\}^\tau$, outputs a bit. We interpret 1 as meaning the verifier *accepts* and 0 as meaning it *rejects*. This computation is denoted $\mathrm{VF}_K(M, \mathrm{Tag})$. Algorithm MAC can be probabilistic, but VF typically is not. A restriction is that if $\mathrm{MAC}_K(M) = \mathrm{Tag}$, then $\mathrm{VF}_K(M, \mathrm{Tag})$ must output 1. If $\mathrm{MAC}_K(M) = \mathrm{MAC}_K(M')$ for some K, M, M', we say that messages M and M' *collide* under that key.

The common notion for MAC security is resistance to adaptive chosen message attack [3]. This notion states, informally, that an adversary *forges* if he can produce a new message along with a valid tag after making some number of queries to a MACing oracle. Because we are interested in *multiple* forgeries, we now extend this definition in a natural way.

Definition 4 [MAC Security—j Forgeries] Let $\Pi = (\mathrm{MAC}, \mathrm{VF})$ be a message authentication code, and let A be an adversary. We consider the following experiment:

```
Experiment $\mathbf{Exmt}_\Pi^{juf\text{-}cma}(A, j)$
$K \xleftarrow{\$} \mathcal{K}$
Run $A^{\mathrm{MAC}_K(\cdot), \mathrm{VF}_K(\cdot,\cdot)}$
If $A$ made $j$ distinct verification queries $(M_i, \mathrm{Tag}_i)$, $1 \leq i \leq j$, such that
— $\mathrm{VF}_K(M_i, \mathrm{Tag}_i) = 1$ for each $i$ from 1 to $j$
— $A$ did not, prior to making verification query $(M_i, \mathrm{Tag}_i)$, query its
$\mathrm{MAC}_K$ oracle at $M_i$
Then return 1 else return 0
```

The *juf-cma advantage* of A in making j forgeries is defined as

$$\mathbf{Adv}_\Pi^{juf\text{-}cma}(A, j) = \Pr\big[\mathbf{Exmt}_\Pi^{juf\text{-}cma}(A, j) = 1\big].$$

For any $q_s, q_v, \mu_s, \mu_v, t \geq 0$ we overload the above notation and define

$$\mathbf{Adv}_\Pi^{juf\text{-}cma}(t, q_s, \mu_s, q_v, \mu_v, j) = \max_A\{\mathbf{Adv}_\Pi^{juf\text{-}cma}(A, j)\}$$

where the maximum is over all adversaries A that have time-complexity at most t, make at most q_s MAC-oracle queries, the sum of those lengths is at most μ_s, and make at most q_v verification queries where the sum of the lengths of these messages is at most μ_v.

The special case where $j = 1$ corresponds to the regular definition of MAC security. If, for a given MAC, $\mathbf{Adv}_{\Pi}^{\textit{juf-cma}}(t, q_s, \mu_s, q_v, \mu_v, j) \leq \epsilon$, then we say that MAC is (j, ϵ)-secure. For the case $j = 1$, the scheme is simply ϵ-secure.

It is worth noting that the adversary is allowed to adaptively query VF_K and is not penalized for queries that return 0. All that is required is for j distinct queries to VF_K return 1, subject to the restriction these queries were not previously made to the MACing oracle.

Stateful MACs. We will also examine stateful MACs that require an extra parameter or counter value. Our model will let the adversary control the counter, but only allow one forgery for each counter value.

A stateful message authentication code is a pair of algorithms, (MAC, VF), where MAC is an algorithm that, upon input of key $K \in \mathcal{K}$ for some key space $\mathcal{K}$, a message $M \in \mathcal{D}$ for some domain $\mathcal{D}$, and a state value S from some prescribed set of states $\mathcal{S}$, computes a τ-bit tag Tag; we denote this by $\mathrm{Tag} = \mathrm{MAC}_K(M, S)$. Algorithm VF is the verification algorithm such that on inputs $K \in \mathcal{K}$, $M \in \mathcal{D}$, $\mathrm{Tag} \in \{0,1\}^\tau$, and $S \in \mathcal{S}$, VF outputs a bit, with 1 representing accept and 0 representing reject. This computation is denoted $\mathrm{VF}_K(M, S, \mathrm{Tag})$. A restriction on VF is that if $\mathrm{MAC}_K(M, S) = \mathrm{Tag}$, then $\mathrm{VF}_K(M, S, \mathrm{Tag})$ must output 1.

Definition 5 [Stateful MAC Security—j Forgeries] Let $\Pi = (\mathrm{MAC}, \mathrm{VF})$ be a stateful message authentication code, and let A be an adversary. We consider the following experiment:

```
Experiment $\mathbf{Exmt}_{\Pi}^{\textit{jsuf-cma}}(A, j)$
$K \xleftarrow{\$} \mathcal{K}$
Run $A^{\mathrm{MAC}_K(\cdot), \mathrm{VF}_K(\cdot,\cdot,\cdot)}$
If $A$ made $j$ distinct verification queries $(M_i, s_i, \mathrm{Tag}_i)$, $1 \leq i \leq j$, such that
— $\mathrm{VF}_K(M_i, s_i, \mathrm{Tag}_i) = 1$ for each $i$ from 1 to $j$
— $A$ did not, prior to making verification query $(M_i, s_i, \mathrm{Tag}_i)$, query
its MAC oracle with $(M_i, s_i)$
— $A$ did not, prior to making verification query $(M_i, s_i, \mathrm{Tag}_i)$, make a
successful verification query $(M_k, s_k, \mathrm{Tag}_k)$ with $s_i = s_k$
Then return 1 else return 0
```

The *jsuf-cma advantage* of A in making j forgeries is defined as

$$\mathbf{Adv}_{\Pi}^{\textit{jsuf-cma}}(A) = \Pr\left[\mathbf{Exmt}_{\Pi}^{\textit{jsuf-cma}}(A, j) = 1\right].$$

For any $q_s, q_v, \mu_s, \mu_v, t, j \geq 0$ we let

$$\mathbf{Adv}_{\Pi}^{\textit{jsuf-cma}}(t, q_s, \mu_s, q_v, \mu_v, j) = \max_{A}\{\mathbf{Adv}_{\Pi}^{\textit{jsuf-cma}}(A, j)\}$$

where the maximum is over all adversaries A that have time-complexity at most t, make at most q_s MACing queries, the sum of those lengths is at most μ_s, and make at most q_v verification queries where the sum of the lengths of the messages involved is at most μ_v.

Note that the adversary is allowed to make repeated queries to either MAC_K or VF_K with the same s value, but is given as credit a maximum of one forgery per value of s.

3 Attacks

As mentioned in the introduction, all MACs we investigate fail to be secure under the definitions of security given above. We first examine stateless MACs based on a blockcipher and show they are insecure. We then examine what occurs when these MACs maintain some form of state, and we find that in almost all cases this does not add significant security. After this we show HMAC and some stateless universal-hash-based schemes are insecure as well.

Preneel and van Oorschot noted that for any iterated hash function one collision can be used to find others by simply appending identical message blocks to the colliding messages [26]. In the same paper they describe why prepending and appending key material or the block length of the message does not prevent this weakness. Several of their ideas are reiterated in what follows. In other instances, where their attacks do not apply, we employ our own methods. In particular, we investigate the composition of functions from a universal hash family with a PRF and ask how easily an adversary, given a colliding pair of messages, can produce another colliding pair of messages.

Many of these attacks in this and other subsections exploit the knowledge of certain types of collisions to forge successfully. Although a typical birthday attack will usually suffice to find these collisions, more efficient attacks may exist for the particular scheme involved. For example, Bellare and Kohno [4] describe a way to find collisions in hash functions with certain properties using computational resources significantly less than that required for a standard birthday attack. Perhaps similar attacks exist for the schemes we describe, but we are instead focusing on what happens after those collisions have occurred, so it is not so important that collisions may be found more easily by using methods other than a standard birthday attack.

3.1 Blockcipher Based MACs

Let $E = \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a mapping such that for a fixed K (called the key), $E(K, \cdot)$ (also denoted by $E_K(\cdot)$) is a permutation on binary strings of n bits. Many MACs use blockciphers as an underlying building block. The security of such schemes usually reduces to the security of the blockcipher used. We present several widely-used MACs based on blockciphers and examine their security. For the purposes of these attacks, we assume no weaknesses of the blockcipher; the attacks work regardless of the family of permutations chosen.

CBC MAC. The tag produced by CBC MAC with key K on message $M \in \{0,1\}^{nm}$, for some fixed m, denoted by $\text{CBCMAC}_K(M)$, is is computed iteratively as follows: Let $h_0 = 0^n$ and $h_i = E_K(M_i \oplus h_{i-1})$ for $1 \leq i \leq m$. Then $\text{CBCMAC}_K(M) = h_m$. The values $h_0, h_1, \ldots, h_m$ are sometimes referred to as the "chaining values." The security of this scheme is dependent on the fact that all input messages are the same length in the number of n-bit blocks, and a security bound is given in [3]. Once a pair of messages (M, M') that collide have been found, we can easily produce other colliding messages based on an attack by Preneel and van Oorschot in [26], which also serves as the basis for the rest of the attacks in this subsection. The best known attack for finding collisions in CBC MAC is a birthday attack, needing an expected $2^{n/2}$ queries to produce a colliding pair of messages (M, M'). Without loss of generality, assume that the fixed length of messages is $2n$ ($m = 2$), and let $M = M_1 \,\|\, M_2$ and $M' = M'_1 \,\|\, M'_2$ such that $|M_1| = |M'_1| = |M_2| = |M'_2| = n$. If $\text{CBCMAC}_K(M) = \text{CBCMAC}_K(M')$ then, because E_K is one-to-one,

$$E_K(M_2 \oplus E_K(M_1)) = E_K(M'_2 \oplus E_K(M'_1)) \Rightarrow M_2 \oplus E_K(M_1) = M'_2 \oplus E_K(M'_1)$$

Let $v \in \{0,1\}^n - 0^n$ be arbitrary and query the MAC oracle on input $M_1 \,\|\, M_2 \oplus v$ to receive tag t^*. Then we can submit the pair $(M'_1 \,\|\, M'_2 \oplus v, t^*)$ as a forgery pair. To see why, consider the following:

$$M_2 \oplus E_K(M_1) = M'_2 \oplus E_K(M'_1)$$
$$\Rightarrow M_2 \oplus v \oplus E_K(M_1) = M'_2 \oplus v \oplus E_K(M'_1)$$
$$\Rightarrow E_K(M_2 \oplus v \oplus E_K(M_1)) = E_K(M'_2 \oplus v \oplus E_K(M'_1))$$

We can repeat this attack as long as we select a distinct v each time. Each additional forgery requires one query to the MACing oracle. If the set length of messages is m blocks, we can query messages which have identical blocks in the last $m-2$ blocks, so that the birthday attack finds a collision in the chaining values after the first two blockcipher invocations during the computation of CBCMAC. This allows the adversary to forge messages for which the last $m-2$ blocks are of the adversary's choice.

XCBC. The XCBC scheme is an extension of CBC MAC that allows for messages of arbitrary length. Given keys $K1, K2, K3$, $|K1| = k$, $|K2| = |K3| = n$, and an input message $M \in \{0,1\}^*$, the tag produced by XCBC on input M, denoted by $\text{XCBC}_{K1,K2,K3}(M)$, is defined by the following: There are two cases. First suppose that $|M|$ is a multiple of n and that $|M|_n = m$ for some m. Let $h_0 = 0^n$ and $h_i = E_{K1}(M_i \oplus h_{i-1})$ for $1 \leq i \leq m-1$. Then the tag produced by XCBC is $E_{K1}(h_{m-1} \oplus M_m \oplus K2)$. Now suppose $|M|$ is not a positive multiple of n. Let M^* be $M \,\|\, 10^l$ where $l = n - 1 - |M| \bmod n$ so that $|M^*| = m$ for some m. Let $h_0 = 0^n$ and $h_i = E_{K1}(M_i^* \oplus h_{i-1})$ for $1 \le i \le m-1$. Then the tag produced by XCBC is $E_{K1}(h_{m-1} \oplus M_m^* \oplus K3)$.

Suppose $\text{XCBC}_K(M) = \text{XCBC}_K(M')$ for $M \neq M'$, and n does not divide $|M|$ or $|M'|$. Then the XOR-ing of $K3$ before the last blockcipher invocation does not prevent the attack used on CBC MAC. Namely, if we assume that M and M' have lengths, in n-bit blocks, of m and m', respectively, then

$$M_m \oplus K3 \oplus E_K(M_{m-1}) = M'_{m'} \oplus K3 \oplus E_K(M'_{m'-1})$$

$$\Rightarrow M_m \oplus K3 \oplus v \oplus E_K(M_{m-1}) = M'_{m'} \oplus K3 \oplus v \oplus E_K(M'_{m'-1})$$

$$\Rightarrow E_K(M_m \oplus K3 \oplus v \oplus E_K(M_{m-1})) = E_K(M'_{m'} \oplus K3 \oplus v \oplus E_K(M'_{m'-1}))$$

Similarly, if $\text{XCBC}_K(M) = \text{XCBC}_K(M')$ for $M \neq M'$ and n divides $|M|$ and $|M'|$, then the XOR-ing of $K2$ before the last blockcipher invocation does not prevent the attack used on CBC. The adversary gets to choose the length of the queried messages, so the adversary may guarantee that a found collision will be of one of these two forms; we will note, however, that a collision between distinct M, M' such that n divides $|M|$ but n does not divide $|M'|$ is apparently not useful to an adversary. Again, an adversary can generate collisions that occur in the second chaining variable so that the last $m-2$ blocks of a forged message are of the adversary's choice and again, one MACing query is required for each additional forgery.

EMAC. The EMAC scheme [6] is an extension of CBC MAC which attains security without requiring that all messages be of a fixed length. Let $M \in (\{0,1\}^n)^+$ such that $|M|_n = m$ for some m. For keys $K1, K2$ let $h_0 = 0^n$ and $h_i = E_{K1}(M_i \oplus h_{i-1})$ for $1 \le i \le m$. Then the tag produced by EMAC with keys $K1, K2$ on message M, denoted by $\text{EMAC}_{K1,K2}(M)$, is $E_{K2}(h_m)$. This extra encryption under the blockcipher keyed with $K2$ does nothing to prevent the attack we described on CBC MAC, so an adversary can forge messages in exactly the same way as the attack described there.

An attack on PMAC similar to the ones presented in this subsection can be found in Appendix A.

3.2 Padding Attacks

Iterated Hash Functions. Cryptographic hash functions are useful in many contexts. A particularly popular methodology, suggested first by Merkle [24] and later by Damgård[16], is the iterated construction. Formally, let $f : \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^l$ and define the iterated hash $H : (\{0,1\}^n)^+ \times \{0,1\}^l \to \{0,1\}^l$ based on f by the following: On inputs $M \in (\{0,1\}^n)^+, \text{IV} \in \{0,1\}^l$ such that $M = M_1 \parallel M_2 \parallel \ldots \parallel M_m$, $H(M, \text{IV}) = h_m$, where $h_0 = \text{IV}$ and $h_i = f(M_i, h_{i-1})$ for $1 \le i \le m$.[1]

Application to MACs. For many MACs, we can think of modeling the MAC abstractly as $g(f(\cdot))$ where f is an iterated hash function and g is a post-processing function, typically a PRF or PRP. There is a conceptual difference in that cryptographic hash functions do not require a secret key and have notably different security goals than that of MACs, but we feel modeling MAC functions in this way is pedagogically useful.

EMAC. EMAC lends itself well to the above abstraction: On input message M such that $|M|_n = m$, we define $f : \mathcal{K} \times (\{0,1\}^n)^+ \to \{0,1\}^n$, $f_{K1}(M) = h_m$ where h_m is as from the description of EMAC earlier. Then define $g : \mathcal{K} \times \{0,1\}^n \to \{0,1\}^n$, $g_{K2}(x) = E_{K2}(x)$ so that $\text{EMAC}_{K1,K2}(M) = g_{K2}(f_{K1}(M))$. Padding attacks work by exploiting known properties in the function f. Namely, in our example of EMAC, it is easy to see that if $f(M) = f(M')$ for some $M, M' \in (\{0,1\}^n)^+$, then for any string $s \in (\{0,1\}^n)^+$, $f(M \parallel s) = f(M' \parallel s)$. This is a property of all iterated hash functions and has been observed by others [21, 26]. This padding attack is effective against EMAC [6], ANSI retail MAC [1], XCBC [11], and HMAC [2].

HMAC. Let $H : (\{0,1\}^l)^+ \times \{0,1\}^L \to \{0,1\}^L$ be an iterated hash function. Given a secret key K and input message M, $\text{HMAC}_K(M)$ is defined as $H(\bar{K} \oplus \mathit{opad} \parallel H(\bar{K} \oplus \mathit{ipad} \parallel M))$ where opad and ipad are

[1]Typically the length of the message ($|M|$) is appended to the message before hashing, but for all attacks presented in this paper the messages queried by the adversary are assumed to be of the same length (unless otherwise noted), so for simplicity we have omitted this extra step.

predefined constants and $\bar{K}$ denotes the unambiguous padding of K to match the input block size of H. HMAC will succumb to the padding attack described above because of its use of an iterated hash function. Let $M, M' \in (\{0,1\}^l)^+$, $|M| = |M'| = m$ be distinct messages that collide under HMAC_K. If we assume that the collision occurs in the hash function keyed by $\bar{K} \oplus \mathit{ipad}$ (see [26] for methods on ensuring this event occurs), then by the observation made above about collisions in iterated hash functions, $M \,\|\, s$ will collide with $M' \,\|\, s$ for $s \in (\{0,1\}^l)^+$. The adversary forges by querying the MACing oracle at $M \,\|\, s$ to receive tag t and querying the verification oracle at $(M' \,\|\, s, t)$. We can also generate the collision within the first iteration of the compression function used in H via the method described in the attack on CBC MAC (common suffixes among all queried messages); this allows an adversary to forge messages for which all but the first message block is of the adversary's choice.

XCBC. Similarly, we can view $\text{XCBC}_{K1,K2,K3}(M)$ as $g(f(M))$ where $g(x) = E_{K1}(x \oplus K2)$ if n divides $|M|$ and $g(x) = E_{K1}(x \oplus K3)$ otherwise. First suppose that $|M|$ is a multiple of n and that $|M|_n = m$ for some m. Let $h_0 = 0^n$ and $h_i = E_{K1}(M_i \oplus h_{i-1})$ for $1 \le i \le m-1$. Then $f(M)$ is defined as $h_{m-1} \oplus M_m$. Now suppose $|M|$ is not a positive multiple of n. Let M^* be $M \,\|\, 10^l$ where $l = n - 1 - |M| \bmod n$ so that $|M^*| = m$ for some m. Let $h_0 = 0^n$ and $h_i = E_{K1}(M_i^* \oplus h_{i-1})$ for $1 \le i \le m-1$. $f(M)$ is defined as $h_{m-1} \oplus M_m^*$. Let $M, M' \in (\{0,1\}^n)^+$ collide under f so that $g(f(M)) = g(f(M'))$. By the properties of iterated functions discussed above, for an arbitrary $v \in (\{0,1\}^n)^+$, $f(M \,\|\, v) = f(M' \,\|\, v) \Rightarrow g(f(M \,\|\, v)) = g(f(M' \,\|\, v))$. The case where the lengths of M and M' are not multiples of n can be handled similarly.

3.3 Effects of Adding State

A natural question to ask is whether adding state to the schemes discussed above adds sufficient security under our definition. For two natural ways to do so the answer is negative, as we shall see.

One obvious way to add state to a stateless MAC $\Pi = (\text{MAC}, \text{VF})$ is to parameterize inputs with a counter, *cnt*. Let $\langle \mathit{cnt} \rangle_b$ denote the b-bit encoding of *cnt*. Upon an input (M, cnt), and with key $K \in \mathcal{K}$, the new stateful algorithm outputs the tag generated by MAC_K on input $\langle \mathit{cnt} \rangle_b \,\|\, M$. Just as naturally, the algorithm can be defined to return the value $\text{MAC}_K(M \,\|\, \langle \mathit{cnt} \rangle_b)$.

CBC MAC. Suppose that we have chosen to add state to CBC MAC by appending an encoding of the state to the messages before MACing. Suppose (M, i) collides with (M', j) and consider the attack on CBC MAC discussed earlier. Because of the way the state is appended to the message, the variable v in the attack is now XOR-ed with the counter values instead of the last blocks of M, M'. Thus, let v be a value such that the counter values k, l defined as $i \oplus v$ and $j \oplus v$, respectively, have not been queried by the adversary. Then the adversary may query on (M, k) to receive tag t, and forge with (M', l, t). Note that in this attack each counter value is queried at most once.

Now suppose an encoding of the state is prepended to each message in the setting of CBC MAC. A successful attack can be made if an adversary queries messages of the form $R_i \,\|\, M$ where R_i is a randomly-chosen value from $\{0,1\}^n$ and M is a fixed, arbitrarily chosen string from $\{0,1\}^{n(m-1)}$ until j distinct collisions of this form have been found. It is expected that j collisions will occur after $\Theta(\sqrt{j2^{n+1}})$ MAC queries, which is clearly less than linear in j and the number of expected queries to find one collision. Suppose $(R_i \,\|\, M, i)$ collides with $(R_j \,\|\, M, k)$. Because the last $m-1$ blocks of the message are the same, we know that, during the computation of the tags, a collision occurs in the second chaining value (h_2) and is propagated through the rest of the computation. This implies that $E_K(\langle i \rangle_n) \oplus R_i = E_K(\langle k \rangle_n) \oplus R_k$. The adversary picks arbitrary $v \in \{0,1\}^n - 0^n$, $M' \in \{0,1\}^{n(m-1)}$ and queries on $(R_i \oplus v \,\|\, M', i)$ to receive tag t and forges with $(R_k \oplus v \,\|\, M', k, t)$. The justification of this claim is almost identical to the justification for the attack on the stateless CBC MAC and is omitted.

At this point we must stop and note that this particular method of adding state does quite a bit better under our definition of security than all previous schemes we have covered. Instead of allowing an adversary to forge one message per query after one collision in the output tags, an attacker must find a collision using new counter values for each forgery he wishes to make. There are two downsides, however. One is that the number of possible forgeries grows as a square in proportion to the number of times an adversary can query $2^{n/2}$ messages. We will see later that this can be improved upon so that the relationship between the number of messages an adversary must query to the number of probable forgeries can be linear. The other downside is that there is no proof of security that the above attack is the best an adversary can do. Again, we do not claim that any of our attacks are the most damaging.

For the same reason that the non-padding attack on CBC MAC worked with only slight alterations for EMAC and XCBC, the attack described above will also work on EMAC and XCBC with the same alterations. A discussion of how adding state affects attacks on HMAC and PMAC can be found in the appendix.

3.4 Attacks on Universal Hash Function Families

There are several MAC paradigms of the Carter-Wegman style [15] described in this paper. One of them is $\mathrm{FH}[\mathcal{H},\mathcal{R}]$. In this paradigm we require the hash function family to be ϵ-AU. The shared key between signer and receiver is (h,ρ), where $h \overset{\$}{\leftarrow} \mathcal{H} = \{h : \mathcal{D} \to \{0,1\}^l\}$ and $\rho \overset{\$}{\leftarrow} \mathcal{R} = \mathrm{Rand}(l, L)$. To MAC message M, the signer sends $\rho(h(M))$. To verify a received message M with tag t, the verifier computes $\rho(h(M))$ and ensures it equals t.

The attack on this scheme is dependent on the family of universal hash functions used. We will show that for each of the families hash127, Square-Hash, LFSR-based Topelitz Hash, Bucket Hash, MMH, NH, and NMH there exists an adversary A such that A can forge j messages in the $\mathrm{FH}[\mathcal{H},\mathcal{R}]$ paradigm in resources comparable to those required for a single forgery.

The adversary works by hashing messages to the birthday bound of h and, with the knowledge of two messages M, M' such that $h(M) = h(M')$, producing two more messages F, F' related to M, M' such that $h(F) = h(F')$. This allows the adversary to forge by querying the MAC oracle on F to receive tag t^* and to forge with (F', t^*). Notably, $h(F) = h(F')$ implies that $\rho(h(F)) = \rho(h(F'))$. We describe the insecurity of the hash functions by showing ways to, given a colliding pair of messages M, M' under that hash function instance, produce a new pair of messages which collide under the same instance without making any additional queries. Of course, if we see a collision in the tags computed by a particular instance of FH on messages M, M', we do not know whether $h(M) = h(M')$ or $h(M) \neq h(M')$ and the collision occurred in ρ. We get around this by assuming the former event until we see evidence to the contrary. That is, we apply the techniques covered throughout the rest of this subsection and if more collisions occur as predicted, we can be reasonably confident that the collision occurred first in h.

Most of the hash function families are examined in the Appendix A, but we have included the analysis of two families here which yield to key-recovery attacks when distinct messages M, M' are found such that $h(M) = h(M')$ for an h in the respective family.

HASH127. Let $M = (M_0, M_1, \ldots, M_{m-1})$ be a sequence of integers in $[-2^{31}, 2^{31}-1]$. For any integer x define $h_x(M) = (x^{m+1} + M_0x^m + M_1x^{m-1} + \ldots + M_{m-1}x) \bmod (2^{127}-1)$. Bernstein proves that the set of such h defined over all x is an ϵ-AU hash family for a small ϵ and describes a way [8] to efficiently compute $h_x(M)$, where x corresponds to the secret key. This function is used to hash a message $M = M_0 \parallel \ldots \parallel M_{m-1}$ by interpreting each M_i as a four byte integer.

Claim 6 *Let $M = (M_0, M_1, \ldots, M_{m-1})$, $M' = (M'_0, M'_1, \ldots, M'_{m-1})$ be two distinct messages such that $h_x(M) = h_x(M')$. Then for an arbitrary non-zero constant $v \in [-2^{31}, 2^{31}-1]$ such that $M_i + v < 2^{31}-1$, $M'_i + v < 2^{31}-1$, the messages $F = (M_0, \ldots, M_{i-1}, M_i + v, M_{i+1}, \ldots, M_{m-1})$ and $F' = (M'_0, \ldots, M'_{i-1}, M'_i + v, M'_{i+1}, \ldots, M'_{m-1})$ will also collide under h_x.*

Proof:

$$\begin{aligned} h_x(F) &= (h_x(M) + x^{m-i}v) \bmod (2^{127}-1) \\ &= h_x(M) \bmod (2^{127}-1) + x^{m-i}v \bmod (2^{127}-1) \\ &= h_x(M') \bmod (2^{127}-1) + x^{m-i}v \bmod (2^{127}-1) \\ &= (h_x(M') + x^{m-i}v) \bmod (2^{127}-1) \\ &= h_x(F'). \end{aligned}$$

▌

One can do better than finding more collisions, however. Let $g(x)$ be the monic polynomial of degree m over the field modulo the prime used in hash127 $(2^{127}-1)$ where the coefficient of the $m+1-i$-th term is

$(M_0 - M_0')^{-1}(M_i - M_i')$ (all arithmetic is done modulo $2^{127}-1$) for $0 \leq i \leq m$. We know g is non-zero because $M \neq M'$. Because $h_x(M) = h_x(M')$, $g(x) = 0$. Using Berlekamp's algorithm [7] for factoring polynomials over large fields, we can find all zeros of g and test them via the MAC oracle to determine x with arbitrarily high probability. There are at most m zeros of g (g may have as factors irreducible polynomials of degree > 1), so a probabilistic algorithm will need an expected $\log m$ queries to the MAC oracle to determine the key with probability close to $1 - 1/2^{127}$. This probability can be brought arbitrarily close to 1 with more queries. The algorithm for doing this is described in Appendix B.

Square-Hash. We describe the universal hash family Square-Hash, first described in [18] as follows: choose a prime number p. For a given secret key $x \in \mathbb{Z}$, and message M, Square-Hash is computed by $h_x(M) = (M+x)^2 \bmod p$. An interesting property of Square-Hash is that when two messages M and M' are found to collide under h_x, it is possible to recover the secret x.

Claim 7 *Let M, M' be two distinct messages such that $h_x(M) = h_x(M')$. Then $x \equiv (2M - 2M')^{-1}((M')^2 - M^2) \bmod p$, where the multiplicative inverse is taken over the field of integers modulo p.*

Proof: By definition, because $h_x(M) = h_x(M')$, we know that

$$\begin{aligned}
(M+x)^2 \bmod p &\equiv (M'+x)^2 \bmod p \Rightarrow \\
(M^2 + 2Mx + x^2) \bmod p &\equiv ((M')^2 + 2M'x + x^2) \bmod p \Rightarrow \\
(M^2 + 2Mx) \bmod p &\equiv ((M')^2 + 2M'x) \bmod p \Rightarrow \\
(2M - 2M')x \bmod p &\equiv ((M')^2 - M^2) \bmod p \Rightarrow \\
x \bmod p &\equiv (2M - 2M')^{-1}((M')^2 - M^2) \bmod p
\end{aligned}$$

▮

To allow messages of greater lengths, Square-Hash was extended to a family SQH* by using a sum.[2] Let $M = M_1 \parallel M_2 \parallel \ldots \parallel M_m$ where $|M_i| = n$ and let x be an m-vector with coordinates $x_1, x_2, \ldots, x_m$ in the integers. Then $\mathrm{SQH}^*_x(M)$ is computed as $\sum_{i=1}^{m}(M_i + x_i)^2 \bmod p$. In this scheme, key recovery is possible using m separate birthday attacks. For $1 \leq i \leq m$, query messages up to the birthday bound of the form $0^{n(i-1)} \parallel R_k \parallel 0^{n(m-i)}$ where $R_k \xleftarrow{\$} \{0,1\}^n$ so that tags are computed using only the secret value x_i and the MAC is reduced to the original Square-Hash. A collision among messages of this form will yield the value of x_i. After m such attacks are completed the entire key x may be recovered.

To forge messages after only one collision has occurred, an attacker may find the appropriate x_i using the attack above then query on an arbitrary message $M = M_1 \parallel M_2 \parallel \ldots \parallel M_m$ to receive tag t. Note that $(M_i + x)^2 \equiv a \bmod p$ is a quadratic residue $\bmod p$ and that there are two distinct values $b, c \bmod p$ such that $b^2 \equiv c^2 \equiv a \bmod p$. Clearly $(M_i + x)^2$ is one of those values. The attacker merely finds the other value and computes M_i' from this value. Then let M' be the message formed by letting $M_j' = M_j$ for $j \neq i$ and M_i' from this value computed earlier. Then $\mathrm{MAC}(M) = \mathrm{MAC}(M')$.

FCH. The counter mode Carter-Wegman paradigm FCH$[\mathcal{H}, \mathcal{R}]$ is described below. Let $\mathcal{H}$ be some $\epsilon-$AU hash family $\mathcal{H} = \{h : \mathcal{D} \to \{0,1\}^l\}$, and $\mathcal{R}$ a set of functions $\mathcal{R} = \mathrm{Rand}(l+b, L)$. Let $\rho \xleftarrow{\$} \mathcal{R}$ and $h \xleftarrow{\$} \mathcal{H}$. (ρ, h) is the shared key between signer and verifier. The signer has a counter, *cnt*, which is an integer variable. To MAC message M, the signer first ensures that $cnt < 2^b - 1$ and if so sends $(cnt, \rho(\langle cnt \rangle_b \parallel h(M)))$. To verify a received message M with tag (i, t), the verifier computes $\rho(\langle i \rangle_b \parallel h(M))$ and ensures it equals t.

An adversary can compute j forgeries successfully as follows: query an expected $2^{l/2}$ messages of the form (M, cnt) to the MACing oracle for some fixed *cnt* and varying M to find messages M, M' such that $h(M) = h(M')$ and $M \neq M'$. The adversary then makes a query (M, cnt') to the MACing oracle for some $cnt' \neq cnt$ and receives tag t. The adversary then forges with (M', cnt', t).

[2]The fully optimized version of Square-Hash has some minute differences from the scheme presented here that complicate the exposition yet do not hinder the general nature of our attack; thus this simplified version is presented.

4 Schemes Resistant to Reforgeability

While FCH$[\mathcal{H}, \mathcal{R}]$ is not secure under a j forgery attack, we introduce a new paradigm for Carter-Wegman MACs based on FCH$[\mathcal{H}, \mathcal{R}]$ that is secure under such an attack. We will call this new paradigm FCHC$[\mathcal{H}, \mathcal{R}]$. It works as follows. Let $\mathcal{H}$ be some $\epsilon-$AU hash family $\mathcal{H} = \{h : D \to \{0,1\}^l\}$, and $\mathcal{R}$ a set of functions $\mathcal{R} = \text{Rand}(l+b, L)$. Without loss of generality assume that $|\mathcal{H}| = 2^s$ for some $s < L$. Let $\rho_0, \rho_1 \xleftarrow{\$} \mathcal{R}$; the shared key between signer and verifier is (ρ_0, ρ_1). Let $\langle cnt \rangle_b$ denote the encoding of cnt using b bits. To MAC message (M, cnt), the signer first ensures that $cnt < 2^b - 1$ and if so sends $(cnt, \rho_0(\langle cnt \rangle_b \,\|\, h_{cnt}(M)))$ as the tag, where h_{cnt} is selected by the first s bits of $\rho_1(\langle cnt \rangle_b \,\|\, 10^{l-1})$. To verify a received message M with tag (i, t), the verifier computes $\rho_0(\langle i \rangle_b \,\|\, h_i(M))$ and ensures it equals t. The proof can be found in Appendix C.

Theorem 8 *Let $\mathcal{H}$ be ϵ-AU and let $\mathcal{R} = \text{Rand}(l+b, L)$. Then* FCHC$[\mathcal{H}, \mathcal{R}]$ *is* $\left(j, \left(\frac{(\epsilon+2^{-L})q^2}{2j^2}\right)^j\right)$*-secure.*

Relative Security of FCHC. What does theorem 8 mean and what sorts of security comparisons are appropriate? The fact that it is stateful requires that extra bits are transmitted with each signed message, and this mitigates the benefits of FCHC in comparison to, say, a stateless scheme which uses those extra bits to push back the birthday phenomenon, but it would be incorrect to say that this fact nullifies all benefits. Consider MACs such that the total number of bits transferred (including state and/or random coins, if applicable) is 64 and consider an adversary that has resources to query 2^{40} input/output pairs to the MAC oracles.[3] For a stateless MAC, all 64 bits are used for the tag. A collision in tags is expected after around 2^{32} queries and if the MAC is one of the stateless MACs covered in this paper, the adversary can now forge once per query thereafter. The expected number of forgeries in this case is close to 2^{39}.

How does FCHC do under such parameters? Say an instance of FCHC uses a PRF and an ϵ-AU hash family which both output 32 bits (so ϵ is near $1/2^{32}$). The other 32 bits are used for the state. Each forgery requires an expected 2^{16} queries, so an adversary can expect to forge around 2^{24} messages, although the first forgery comes much earlier compared to the stateless scheme. This is a drastic difference from the stateless MAC where fully almost half of the messages queried by the adversary are forgeries. With FCHC and these parameters, only one in every 2^{16} queries will be a forgery.

Let's change the parameters slightly. If our instance of FCHC uses a PRF and an ϵ-AU hash family with output lengths of 48 bits, with the other 16 bits used for the state, then each forgery requires an expected 2^{24} queries. The adversary can then expect to forge 2^{16} messages. In some applications, forgery of a few thousand messages may be tolerable, while a complete break where 2^{39} forgeries are allowed would be intolerable.

What about adding state to get beyond the birthday bound? Unintuitively, this can actually make matters worse. Suppose of the 64 bits used, 32 are used for the tag and 32 are used for the state. Let's say we use FCH. In our model, repeated queries with the same tag are allowed, so this just pushes the first collision up so that only 2^{16} queries are needed. Again, almost 2^{39} forgeries are allowed. It may be argued that such a security model is unrealistic and gives and adversary too much power. To address this criticism, let us analyze the scenario where the adversary is not allowed repeat signing queries with the same state value but is allowed repeat queries the verification oracle which use the same state. In this case, the adversary works as follows: make one query (M, s) to the signing query to receive tag t. The adversary then queries an expected 2^{32} message/tag pairs (M_i, s, t) to the verification oracle to find some message M' such that the verification oracle returns 1 on input (M', s, t) - the adversary has found a collision in the UHF (or the outer PRF - if this is the case, additional queries must be made, but the expected number is still on the order of 2^{32}). The MAC is now completely broken, and the adversary obtains one forgery for each additional query thereafter, for, again, an expected 2^{39} forgeries.

What if our stateful MAC uses only 16 bits for the state and we still do not allow repeated signing queries with the same state value? This is where things get a little hairier with regard to comparisons. With only 2^{40} resources, it is unlikely that the strategy above will yield many collisions. However, if the adversary makes 2^{38} verification queries and does find a message M' such that the verification oracle returns 1 on query (M', s, t), then the adversary can again obtain near 2^{39} forgeries. What is the probability that the

[3]We choose 64 bits because it seems more in line with our motivating scenarios (i.e. applications which must use short tags).

adversary finds such a message M'? About one in a thousand. Is it worth the risk that one in a thousand times the security of the MAC will be completely broken? Or is it more tolerable to admit 2^{16} forgeries with assurance that that is the worst that will happen? The answer to these questions will depend on the application and risk model.

EFFICIENCY. The cost of FCHC is roughly twice that of FH or FCH. Typically, the cost of the MAC is dominated by how many blocks the PRF must process. In FH and FCH, high speeds are achieved by only requiring the PRF to process one block per message. In FCHC, two blocks are processed per message by the PRF, one to select the instance of the UHF, and the other to process the output of the UHF instance on the message input. Thus the cost is roughly twice that of FH and FCH. The trade here is that some speed is sacrificed for the benefit of avoiding reforging attacks.

Some may complain that the non-static key to the UHF could potentially seriously degrade performance. While most Universal Hash Families do require key expansion, some are designed with interchangability of keys as a selling point. Bernstein's poly1305 family is such an example [9] that does not rely on any precomputation with respect to the key material.

References

[1] ASSOCIATION, A. B. ANSI X9.19, Financial institution retail message authentication, August 1986. Washington, D. C.

[2] BELLARE, M., CANETTI, R., AND KRAWCZYK, H. Keying hash functions for message authentication. In *CRYPTO* (1996), N. Koblitz, Ed., vol. 1109 of *Lecture Notes in Computer Science*, Springer, pp. 1–15.

[3] BELLARE, M., KILIAN, J., AND ROGAWAY, P. The security of cipher block chaining. In Desmedt [17], pp. 341–358.

[4] BELLARE, M., AND KOHNO, T. Hash function balance and its impact on birthday attacks. In *EUROCRYPT* (2004), C. Cachin and J. Camenisch, Eds., vol. 3027 of *Lecture Notes in Computer Science*, Springer, pp. 401–418.

[5] BELLARE, M., AND ROGAWAY, P. Introduction to modern cryptography, November 2003. Class notes available as `http://www-cse.ucsd.edu/users/mihir/cse207/classnotes.html`.

[6] BERENDSCHOT, A., DEN BOER, B., BOLY, J. P., BOSSELAERS, A., BRANDT, J., CHAUM, D., DAMGARD, I., DISCHTL, M., FUMY, W., VAN DER HAM, M., JANSEN, C. J. A., LANDROCK, P., PRENEEL, B., ROELOFSEN, G., DE ROOIJ, P., AND VANDEWALLE, J. Final report of RACE integrity primitives, 1995.

[7] BERLEKAMP, E. R. Factoring polynomials over large finite fields. *Mathematics of Computation 24* (1970), 713–735.

[8] BERNSTEIN, D. Floating-point arithmetic and message authentication. Draft available as `http://cr.yp.to/papers/hash127.dvi`.

[9] BERNSTEIN, D. The poly1305-AES message-authentication code. Draft available as `http://cr.yp.to/mac.html`.

[10] BLACK, J., HALEVI, S., KRAWCZYK, H., KROVETZ, T., AND ROGAWAY, P. UMAC: Fast and secure message authentication. In Wiener [32], pp. 216–233.

[11] BLACK, J., AND ROGAWAY, P. CBC MACs for arbitrary-length messages: The three-key constructions. In *CRYPTO* (2000), M. Bellare, Ed., vol. 1880 of *Lecture Notes in Computer Science*, Springer, pp. 197–215.

[12] BLACK, J., AND ROGAWAY, P. A block-cipher mode of operation for parallelizable message authentication. In *EUROCRYPT* (2002), L. R. Knudsen, Ed., vol. 2332 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 384–397.

[13] Brassard, G., Ed. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings* (1990), vol. 435 of *Lecture Notes in Computer Science*, Springer.

[14] Cantor, D. G., and Zassenhaus, H. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation 36*, 154 (1981), 587–592.

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

[16] Damgård, I. A design principle for hash functions. In Brassard [13], pp. 416–427.

[17] Desmedt, Y., Ed. *Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings* (1994), vol. 839 of *Lecture Notes in Computer Science*, Springer.

[18] Etzel, M., Patel, S., and Ramzan, Z. Square hash: Fast message authentication via optimized universal hash functions. In Wiener [32], pp. 234–251.

[19] Goldreich, O., Goldwasser, S., and Micali, S. How to construct random functions. *J. ACM 33*, 4 (1986), 792–807.

[20] Halevi, S., and Krawczyk, H. MMH: Software message authentication in the gbit/second rates. In *Fast Software Encryption* (1997), pp. 172–189.

[21] Kaliski, B., and Robshaw, M. Message authentication with MD5. *CryptoBytes* (Spring 1995).

[22] Krawczyk, H. LFSR-based hashing and authentication. In Desmedt [17], pp. 129–139.

[23] McGrew, D. A., and Fluhrer, S. R. Multiple forgery attacks against message authentication codes. Cryptology ePrint Archive, Report 2005/161, 2005. `http://eprint.iacr.org/`.

[24] Merkle, R. C. One way hash functions and DES. In Brassard [13], pp. 428–446.

[25] NIST. Secure Hash Standard (SHS). *Federal Information Processing Standards Publication*, 180-1 (April 1995).

[26] Preneel, B., and van Oorschot, P. C. MDx-MAC and building fast MACs from hash functions. In *CRYPTO* (1995), D. Coppersmith, Ed., vol. 963 of *Lecture Notes in Computer Science*, Springer, pp. 1–14.

[27] Rivest, R. The MD5 message-digest algorithm. *RFC 1321*, 37 (April 1992).

[28] Rogaway, P. Bucket hashing and its application to fast message authentication. *Journal of Cryptology: the journal of the International Association for Cryptologic Research 12*, 2 (1999), 91–115.

[29] Rogaway, P., Bellare, M., and Black, J. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur. 6*, 3 (2003), 365–403.

[30] Stinson, D. R. On the connections between universal hashing, combinatorial designs and error-correcting codes. *Electronic Colloquium on Computational Complexity (ECCC) 2*, 52 (1995).

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

[32] Wiener, M. J., Ed. *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings* (1999), vol. 1666 of *Lecture Notes in Computer Science*, Springer.

A Attacks

A.1 Blockcipher-Based MACs

PMAC. The MAC PMAC is described as follows: for a given blockcipher E and a given message $M = M_1 \parallel M_2 \parallel \ldots \parallel M_m$ for some m, $|M_i| = n$ for $1 \leq i \leq m-1$, we let $X_i = M_i \oplus \gamma_i \cdot L$ for $1 \leq i \leq m$ where the operation '$\cdot$' as well as the constants γ_i and L are given in the original PMAC paper [11]. The tag produced by PMAC with key K on message M of m blocks, denoted by $\mathrm{PMAC}_K(M)$, is $E_K(\mathit{pad}(M_m) \oplus X_m \oplus E_K(X_1) \oplus \ldots \oplus E_K(X_m - 1))$ where *pad* is a function that unambiguously pads strings of length less than n to strings of length n.

For two distinct messages (M, M') that collide with respective lengths, in n-bit blocks, of m and m', we know that the following must be true:

$$E_K(\mathit{pad}(M_m) \oplus X_m \oplus E_K(X_1) \oplus \ldots \oplus E_K(X_m - 1)) =$$
$$E_K(\mathit{pad}(M'_{m'}) \oplus X'_{m'} \oplus E_K(X'_1) \oplus \ldots \oplus E_K(X'_{m'} - 1))$$
$$\Rightarrow \mathit{pad}(M_m) \oplus X_m \oplus E_K(X_1) \oplus \ldots \oplus E_K(X_{m-1}) =$$
$$\mathit{pad}(M'_{m'}) \oplus X'_{m'} \oplus E_K(X'_1) \oplus \ldots \oplus E_K(X'_{m'-1})$$

Let $l = \min\{|M_m|, |M'_{m'}|\}$ and let $v \in \{0,1\}^l - 0^l$ be arbitrary. Let $F = M_1 \parallel \ldots \parallel M_{m-1} \parallel M_m \oplus v$ and let $F' = M'_1 \parallel \ldots \parallel M'_{m'-1} \parallel M'_{m'} \oplus v$. Then $\mathrm{PMAC}_K(F) = \mathrm{PMAC}_K(F')$. Indeed,

$$E_K(\mathit{pad}(M_m \oplus v) \oplus X_m \oplus E_K(X_1) \oplus \ldots \oplus E_K(X_m - 1)) =$$
$$E_K(\mathit{pad}(M'_{m'} \oplus v) \oplus X'_{m'} \oplus E_K(X'_1) \oplus \ldots \oplus E_K(X'_{m'} - 1))$$
$$\Rightarrow \mathit{pad}(M_m) \oplus v \oplus X_m \oplus E_K(X_1) \oplus \ldots \oplus E_K(X_{m-1}) =$$
$$\mathit{pad}(M'_{m'}) \oplus v \oplus X'_{m'} \oplus E_K(X'_1) \oplus \ldots \oplus E_K(X'_{m'-1})$$
$$\Rightarrow \mathit{pad}(M'_{m'}) \oplus X'_{m'} \oplus E_K(X'_1) \oplus \ldots \oplus E_K(X'_{m'-1})$$

To forge an attacker would query the oracle on input F to receive tag t^* and forge with F', t^* The reason that we cannot XOR by a string with more than l bits is that in that case the composition of functions *pad* and XOR is not commutative - if we XOR by a string longer than the original length of M_m or $M'_{m'}$, the messages are not padded in the same way and we are not changing the same bits in both messages. Again, the adversary chooses the lengths of the messages, so this does not hinder the effectiveness of the attack.

A.2 Attacks on Universal Hash Families

LFSR-Based Topelitz Hash. In Carter and Wegman's original paper, they provided an example of a Universal hash family. Fix parameters m and n. Let $\mathbf{A}$ be a random $m \times n$ binary matrix. The family $\mathcal{H} = \{h : \{0,1\}^m \rightarrow \{0,1\}^n\}$ is Universal where a member of the family is specified by the choice of $\mathbf{A}$. We compute $h(M)$ by $\mathbf{A}M$. Krawczyk introduced another family based on this [22], with changes designed to speed up hardware implementations. The changes are not relevant to the attack discussed here, however, because a member of the scheme that Krawczyk describes is still a matrix $\mathbf{A}$, and $h(M)$ is still defined as $\mathbf{A}M$.

Consider distinct messages M, M' in the domain of h such that $h(M) = h(M')$. This means that

$$\mathbf{A}M = \mathbf{A}M' \Rightarrow \mathbf{A}(M - M') = 0$$

Because $M \neq M'$, we have found a non-zero vector vector w such that $\mathbf{A}w = 0$ (clearly $\mathbf{A}$ must be singular for this to occur, but for h to be a compression function $m > n$ anyway, so this assumption is acceptable). Pick F in the domain of h not equal to M or M' arbitrarily. Then let $F' = F - M + M'$.

Claim 9 $h(F) = h(F')$

Proof: $\mathbf{A}F - \mathbf{A}F' = \mathbf{A}(F - F') = \mathbf{A}(F - (F - M + M') = \mathbf{A}(M - M') = 0$ ▌

Bucket Hash. First described by Rogaway in 1995 [28], the bucket hashing scheme is as follows: 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 break M into n words of w bits each. So $M = M_1 \parallel M_2 \parallel \ldots \parallel M_n$ 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 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 $i \neq j$. Formally, let x 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 x as $x_i = \{x_{i1}, x_{i2}, x_{i3}\}$. For any $M \in \{0,1\}^{nw}$ we run the following algorithm:

```
bucket_hash($M$)
for $i \leftarrow 1$ to $N$ do $Y_i \leftarrow O^w$
for $i \leftarrow 1$ to $N$ do
$Y_{x_{i1}} \leftarrow Y_{x_{i1}} \oplus M_i$
$Y_{x_{i2}} \leftarrow Y_{x_{i2}} \oplus M_i$
$Y_{x_{i3}} \leftarrow Y_{x_{i3}} \oplus M_i$
return $Y_1 \parallel Y_2 \parallel \ldots \parallel Y_n$
```

Pick an arbitrary $v \in \{0,1\}^w$ such that $v \neq 0^w$. Define F as the result of XOR-ing every M_i with v, and similarly define F' as the result of XOR-ing every M'_i with v.

Claim 10 *bucket_hash*(F) = *bucket_hash*(F').

Proof: Consider the string returned by bucket_hash(F) which we will label by $Z_1 \parallel Z_2 \parallel \ldots \parallel Z_n$. Also, let bucket_hash$(F') = Z'_1 \parallel Z'_2 \parallel \ldots \parallel Z'_n$. Consider block Z_j. This block is equal to

$$F_{j1} \oplus F_{j2} \oplus \ldots \oplus F_{jk_j}$$

where the indices correspond to the blocks of F that were XOR-ed into bucket j. Similarly,

$$Z'_j = F'_{j1} \oplus F'_{j2} \oplus \cdots \oplus F'_{jk_j}$$

But

$$F_{j1} \oplus F_{j2} \oplus \cdots \oplus F_{jk_j} = M_{j1} \oplus v \oplus M_{j2} \oplus v \oplus \cdots \oplus M_{jk_j} \oplus v =$$

$$M'_{j1} \oplus v \oplus M'_{j2} \oplus v \oplus \cdots \oplus M'_{jk_j} \oplus v = F'_{j1} \oplus F'_{j2} \oplus \cdots \oplus F'_{jk_j}$$

▌

MMH. The MMH family [20] is $\mathcal{H} = \{h : (\{0,1\}^{32})^n \rightarrow \{0,1\}^{32}\}$ 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 ith coordinate of x and M_i the ith coordinate of M. Through some clever implementation tricks, this family is very efficient in software. Consider message M and M' such that $h_x(M) = h_x(M')$. Choose arbitrary non-zero $v \in \{0,1\}^{32}$ and $i_0 \in [1 \ldots n]$. Define F in the following manner: $F_i = M_i$ for all $i \neq i_0$ and $F_{i_0} = M_{i_0} + v \bmod 2^{32}$. Similarly we define F' as $F'_i = M'_i$ for $i \neq i_0$ and $F'_{i_0} = M'_{i_0} + v$.

Claim 11 $h_x(F) = h_x(F')$.

Proof:

$$h_x(F) = \left[\left[\left[vx_{i_0} + \textstyle\sum_{i=1}^{n} M_i x_i\right] \bmod 2^{64}\right] \bmod (2^{32}+15)\right] \bmod 2^{32} =$$

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

$$\left[\left[\left[vx_{i_0}\right] \bmod 2^{64}\right] \bmod (2^{32}+15)\right] \bmod 2^{32} =$$

$$\left[\left[\left[vx_{i_0} + \textstyle\sum_{i=1}^{n} M'_i x_i\right] \bmod 2^{64}\right] \bmod (2^{32}+15)\right] \bmod 2^{32} = h_x(F')$$

The equalities are justified by the fact that modular arithmetic can be distributed over addition. ▌

NMH. Also mentioned in the MMH paper [20] is the adaption of the authors' methods to a family created by Mark Wegman. NMH is defined as $\mathcal{H} = \{h : (\{0,1\}^{32})^n \to \{0,1\}^{32}\}$ where a member of this set is selected by some n-vector x with coordinates in $\{0,1\}^{32}$. We assume here, for simplicity, that n is even. For any message M taken as an n-vector with coordinates in $\{0,1\}^{32}$ we compute $h_x(M)$ as

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

where x_i denotes the ith coordinate of x and M_i the ith coordinate of M.

Consider the case where there are two distinct message M, M' such that $h_x(M) = h_x(M')$. Pick distinct $i_0, i_1 \in [1 \ldots n]$. Without loss of generality assume both i_0 and i_1 are both even. For concision denote $a = M_{i_0-1} - M'_{i_0-1}$ and $b = M_{i_1-1} - M'_{i_1-1}$. Let $v_0 = ab^2$ and $v_1 = -a^2 b$. Define message F in the following manner: $F_i = M_i$ for $i \notin \{i_0, i_1\}$ and $F_{i_b} = M_{i_b} + v_b$ for $b \in [12]$. Define message F' as $F'_i = M_i$ for $i \notin \{i_0, i_1\}$ and $F'_{i_b} = M_{i_b} + v_b$ for $b \in [12]$.

Claim 12 $h_x(F) = h_x(F')$

Proof:

$$h_x(F) = \Big[\Big[\Big[v_0(M_{i_0-1} + x_{i_0-1}) + v_1(M_{i_1-1} + x_{i_1-1}) +$$

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

But note that

$$h_x(F') = \Big[\Big[\Big[v_0(M'_{i_0-1} + x_{i_0-1}) + v_1(M'_{i_1-1} + x_{i_1-1}) +$$

$$\textstyle\sum_{i=1}^{n/2} (M'_{2i-1} + x_{2i-1} \bmod 2^{32})(M'_{2i} + x_{2i} \bmod 2^{32})\Big] \bmod 2^{64}\Big] \bmod (2^{32}+15)\Big] \bmod 2^{32}$$

It will suffice to show that $v_0(M_{i_0-1} + x_{i_0-1}) + v_1(M_{i_1-1} + x_{i_1-1}) = v_0(M'_{i_0-1} + x_{i_0-1}) + v_1(M'_{i_1-1} + x_{i_1-1})$. After subtracting the common terms in x from both sides, note that this is equivalent to showing that $v_0 a = -v_1(b)$. By the way v_0 and v_1 were defined, $v_0 a = a^2 b^2 = -v_1 b$. ▌

The family NH used in UMAC [10] is very similar to NMH - essentially the differences amount to the constants chosen over which to do modular arithmetic. As such, the above attack can be easily adopted to NH.

A.3 Effects of Adding State

HMAC. Recall that given a secret key K and input message M, $\mathrm{HMAC}_K(M)$ is defined as

$$H(\bar{K} \oplus \mathit{opad} \parallel H(\bar{K} \oplus \mathit{ipad} \parallel M))$$

where H denotes some iterated hash function and $\bar{K}$ denotes the unambiguous padding of K to match the input block size of H. Suppose H takes strings of the form $(\{0,1\}^n)^+$ as input and the state is encoded as a string of length n and prepended to the message M to obtain a string M^*; the returned tag is the output of the stateless version of HMAC on input M^*. An adversary can efficiently attack this construction by querying messages of the form (M, i) for varying values of i and an arbitrary, fixed M. This querying is done until j colliding pairs of messages have been found - as mentioned earlier, this will occur with much fewer than j times the number of queries required to produce the first collision. For each pair of colliding messages (M, i), (M, j), the adversary picks an arbitrary $M' \neq M$ in the domain of H, queries the oracle on (M', i) to receive tag t, and forges with (M', j, t). This will be a correct forgery by the properties of iterated hash functions described earlier.

Now suppose the encoding of the state is appended to the message M to obtain message M^*, which is used as the input to HMAC. The adversary first queries, up to the birthday bound, messages of the form (M^i, a) for distinct M^i where n divides $|M^i|$ and fixed a, until a pair of colliding messages (M^i, a), (M^j, a) is found. The attacker can now forge messages by arbitrarily choosing an unqueried counter value k, querying the oracle at (M^i, k) to receive tag t and forging with (M^j, k, t).

PMAC Prepending the state to a message M before MACing does not prevent forgeries for PMAC in our model. The attack is as follows: messages of the form $(R_i \parallel 0^n, i)$, where R_i is a random string from $\{0,1\}^{n(m-1)}$, are queried. Suppose the queries $(R_i \parallel 0^n, i), (R_j \parallel 0^n, j)$ to the MAC oracle return the same tag t. Then by an analysis similar to the stateless (specified) version of PMAC, an adversary may query on $(R_i \parallel 1^n, i)$ to get tag t' and forge with $(R_j \parallel 1^n, j, t')$. The justification for this is the same for the stateless case. Again, j forgeries may be obtained in expected queries within a constant factor of $\sqrt{j2^{n+1}}$.

WC. $\mathrm{WC}[\mathcal{H}, \mathcal{R}]$ is another Carter-Wegman MAC paradigm computed in the following manner. Let $\mathcal{H} = \{h : D \to \{0,1\}^b\}$ and P be an infinite random string, $P = P_1 \parallel P_2 \parallel \ldots$ with $|P_i| = b$. Let $h \xleftarrow{\$} \mathcal{H}$. The shared key between the signer and verifier is (P, h). The signer has a counter, *cnt*, which is an integer variable. To MAC a message $M \in D$, the signer sends $(\mathit{cnt}, P_{\mathit{cnt}} \oplus h(M))$. to verify a received message M with tag (i, t), the verifier computes $P_i \oplus h(M)$ and ensures it equals t.

An adversary can compute j forgeries successfully as follows: query $2^{l/2}$ messages of the form (M_i, cnt) to $\mathrm{MAC}_{P,h}$ for some fixed *cnt* to find messages M_i, M_j such that $h(M_i) = h(M_j)$ and $M_i \neq M_j$. (The adversary will know when this happens because the output of h is XOR-ed with the same pad every time; thus a collision in two tags t_i and t_j implies a collision between $h(M_i)$ and $h(M_j)$.) The adversary then makes a query (M, cnt') to $\mathrm{MAC}_{P,h}$ for some $\mathit{cnt}' \neq \mathit{cnt}$ and receives tag t. The adversary then forges with (M', cnt', t).

B Details of the hash127 Attack

Let us briefly recall the scenario described in Section 3.4. The adversary has knowledge of two messages M, M' such that $h_x(M) = h_x(M')$ for the unknown instance h_x of hash127. The adversary has constructed a polynomial $g(x)$ over the field of integers modulo $p = 2^{127} - 1$, denoted by F_p, one of the roots of which is the secret x. g has at most m roots (where m is the length of the message, in blocks of 32 bits), and these can be found efficiently using Berlekamp's algorithm [7] or the Cantor/Zassenhaus algorithm [14]. Let $x_1, x_2, \ldots, x_k$ denote these roots ($k \leq m$). We assume here that the adversary has made at least one extra query M'' to the MAC oracle (besides the colliding messages), and received in response tag t''. If this is not the case (in which case the adversary was extremely lucky - the first two queries yielded a collision!), then the adversary must make one extra query.

The attack is probabilistic and needs an expected $\log m$ additional queries. The algorithm is described below.

```
**Algorithm** Find_Key
$X \leftarrow \{x_i : 1 \le i \le k\}$
while $|X| > 1$ do:
    • $Z_1 \leftarrow \{x_i : 1 \le i \le \lfloor |X| \rfloor\}$
    • $Z_2 \leftarrow \{x_i : 1 \le i \le \lceil |X| \rceil\}$
    • Let $R \leftarrow \{r_i : 1 \le i \le m - |Z_1|\}$ be randomly-chosen elements from $F_p$.
    • Construct a monic polynomial $f^*(y)$ of degree $m$ such that $f^* \leftarrow \prod_{z \in Z_1}(y - z)\prod_{r \in R}(y - r)$
    • Choose the coefficients of message $M^*$, using simple subtraction, so that the polynomial $f$, whose
      $m+1-i$-th term is $(M''_i - M^*_i)$, is equal to $f^*$.
    • Query the MAC oracle on $M^*$ to receive tag $t^*$.
    • if $t^* = t''$ then $X \leftarrow Z_1$ else $X \leftarrow Z_2$
end do
return contents of $X$
```

The algorithm works by choosing messages M^* such that the polynomial f^* has zeros on half of the remaining possible roots. That is, if the real key x is a root of f^*, then by the way f^* was formed, $h_x(M'') = h_x(M^*)$, and $t^* = t''$. If the real key x is not a root of f^*, then $t^* = t''$ with probability less than $(2m+4)/2^{127} + 1/2^n$ [8], where n is the output size, in bits, of the MAC oracle. The algorithm may be repeated as necessary with different values of M'' (which must be queried) if the adversary suspects the returned value x_i is not the real key x, so that with probability arbitrarily close to 1 the adversary may be sure he has the correct value of x.

C Proof of Theorem 8

This section contains the proof of Theorem 8.

Proof: To review, an adversary A succeeds if it makes $q = q_s + q_v$ queries, q_s queries $Q_s =$

$$\{(M_1, cnt_1), (M_2, cnt_2), \ldots, (M_{q_s}, cnt_{q_s})\}$$

of the MACing oracle and q_v queries $Q_v =$

$$\{(M_1^*, cnt_1^*), (M_2^*, cnt_2^*), \ldots, (M_{q_v}^*, cnt_{q_v}^*)\}$$

of the verification oracle such that j of the q_v queries to the verification oracle returned 1, $(M_i, cnt_i) \neq (M_k^*, cnt_k^*)$ for $1 \le i \le q_s$, $1 \le k \le q_v$, and there are no repeat counter values among the j successful queries to the verification oracle. We must show that A can only succeed with probability that is related to probability of one successful forgery over a term of j^2, with the entire quantity being exponential in j.

Note that, by the way the MAC is parameterized by *cnt*, we are guaranteed that the inputs to ρ_0 for queries $(M, cnt), (M', cnt')$ are distinct so long as $cnt \neq cnt'$. For this reason we can think of *cnt* as parameterizing ρ_0 into a family of functions, with each value selecting an independent member of that family. Also, because a new member of $\mathcal{H}$ is chosen randomly for each value of *cnt*, if we fix *cnt* we can think of the MAC as an instance of FH$[\mathcal{H}, \mathcal{R}]$ completely independent from another instance of FH$[\mathcal{H}, \mathcal{R}]$ chosen by $cnt' \neq cnt$.

Let $q_{s,i}$, $1 \le i \le j$ denote the number of signing queries made for a particular counter value cnt_i such that $\sum_i q_{s,i} = q_s$. Similarly, let $q_{v,i}$, $1 \le i \le j$, denote the number of verification queries made for that same counter value cnt_i such that $\sum_i q_{v,i} = q_v$.[4]

[4]In light of the fact that the adversary is adaptive and has access to random coins, it may seem incorrect to define $q_{s,i}$ and $q_{v,i}$ in this manner. We note that we can parameterize the adversary by a sufficiently long random string, selected at the beginning of the experiment along with the keys to FCHC. At this point, all the random coins have been flipped and the adversary is deterministic. The definitions now make sense and the probabilities are taken over these initial random choices.

Claim 13 *The adversary's advantage in forging correctly for counter value cnt_i is bounded above by $(\epsilon + 1/2^L)q_{v,i} + (\epsilon q_{s,i}(q_{s,i}-1))/2$.*

Proof: Any verification query made before a collision has occurred between two signing queries made with the counter value cnt_i is tantamount to guessing. To see why, note the following. Say a query (M^*, cnt_i, t^*) is made to the verification oracle. There are two cases. In the first case, assume that t^* was not returned as a tag for some previous signing query. Then for the verification oracle to return 1, the adversary must correctly guess an unqueried domain point of ρ_0, which will occur with probability $1/2^L$ because ρ_0 was uniformly selected from $\mathcal{R}$. Now assume that t^* was returned as a tag for some previously queried message (M_k, t_k, cnt_i). The adversary is then correct if $h_{cnt_i}(M^*) = h_{cnt_i}(M_k)$ or if $h_{cnt_i}(M^*) \neq h_{cnt_i}(M_k)$ but $\rho_0(\langle cnt_i\rangle_b \,\|\, h_{cnt_i}(M^*)) = \rho_0(\langle cnt_i\rangle_b \,\|\, h_{cnt_i}(M_k))$. These will occur, respectively, with probabilities of ϵ and $1/2^L$. Thus, the advantage of the adversary for the verification queries is no more than $(\epsilon + 1/2^L)q_{v,i}$.

Now we must show that the advantage from the signing queries is no more than $(\epsilon q_{s,i}(q_{s,i}-1))/2$. This is purely conditioned on the probability that a collision occurs among the tags, at which point we give up and admit a forgery. Let C_ℓ be the event that a collision occurs on the ℓ-th signing query. Clearly, $C_1 = 0$. In general, $C_\ell \leq (\ell-1)(\epsilon + 1/2^L)$ because there are $\ell - 1$ chances to collide and each occurs with probability at most $(\epsilon + 1/2^L)$. Then the probability of a collision is bounded above by

$$\sum_\ell C_\ell \leq \sum_{\ell=1}^{q_{s,i}} (\ell-1)(\epsilon + 2^{-L}) = (\epsilon + 2^{-L}) \sum_{\ell=0}^{q_{s,i}-1} \ell = (\epsilon + 2^{-L})q_{s,i}(q_{s,i}-1)/2$$

▮

Therefore the probability that A successfully forges j messages is bounded above by

$$\prod_{i=1}^{j} \Big((\epsilon + 2^{-L})q_{v,i} + (\epsilon + 2^{-L})q_{s,i}(q_{s,i}-1)/2 \Big)$$

Here we must make a slight diversion into strategies used by the adversary. To simplify the analysis, we only analyze the case of a maximally efficient adversary. All other adversaries by definition will have less of a chance of success. If the adversary is bounded by q total queries split between verification and signing queries, it is clear by the above analysis that an efficient adversary will only attempt a verification query with a counter value of cnt_i after it has found a collision between two signing queries with cnt_i. Before that point the probability of success increases quadratically with respect to the signing queries made, and only linearly with respect to the verification queries.

Now, the probability that an adversary A successfully forges j messages is bounded above by

$$\prod_{i=1}^{j} (\epsilon + 2^{-L})q_{s,i}(q_{s,i}-1)/2 \leq \prod_{i=1}^{j} \frac{(\epsilon + 2^{-L})q_{s,i}^2}{2}$$

(where $\sum_i q_{s,i} \geq q - j$ because all but at most j of the q queries are signing queries). Stated more simply, an adversary must optimize the following product:

$$\prod_{\sum c_i = S} c_i^2$$

where $c_i = q_{s,i}$ and $S = q_s$. This is an equivalent problem to optimizing the following product:

$$\prod_{\sum c_i = S} c_i$$

Now we assume the adversary is adept at solving optimization problems, because the product above can be maximized by thinking about maximizing the content (or hypervolume) of a generalized rectangle in

j-dimensional space where the sum of the length of sides is fixed. This occurs when all the sides are the same length or, in our case, when $q_{s,i} = q_{s,k}$ for $1 \leq i, k \leq j$. That is, $q_{s,i} + 1 \leq q/j$ for $1 \leq i \leq j$.

Then

$$\prod_{i=1}^{j} \frac{(\epsilon + 2^{-L})q_{s,i}^2}{2} \leq \prod_{i=1}^{j} \frac{(\epsilon + 2^{-L})q^2}{2j^2} = \left(\frac{(\epsilon + 2^{-L})q^2}{2j^2}\right)^j$$

To verify the claim above about maximizing content in j-dimensional space, consider the following proof by induction: using the same notation as above, consider the case when $j = 2$. Then the product can be expressed in terms of one variable, x. Namely, $x(S - x)$ where x can range from 0 to S. Using simple techniques from calculus, it is clear that the maximum value to the function is obtained when $x = S/2$. Suppose that for $j \geq 2$, $\prod_{\sum c_i = S, 1 \leq i \leq j} c_i$ is maximized when $c_i = S/j$ for $1 \leq i \leq j$. We want to show that $\prod_{\sum c_i = S, 1 \leq i \leq j+1} c_i$ is maximized when $c_i = S/(j+1)$ for $1 \leq i \leq j+1$. We can express this product as $x \prod_{\sum c_i = S - x, 1 \leq i \leq j} c_i$ for $0 \leq x \leq S$ and now we just need to show that the optimal value for x is $S/(j+1)$. Because of the inductive hypothesis, we know that this product is equivalent to the function, in x, $x(\frac{S-x}{j})^j$. The derivative of this function in x is $(S - x)^j / j^j - x(S - x)^{j-1}/j^{j-1}$. The bounds on x determine that $S - x \geq 0$, and therefore that this function has only one zero (not including the boundaries), when $xj = S - x$. This occurs when $x = S/(j+1)$. ▮