

MAC Reforgeability

JOHN BLACK*

MARTIN COCHRAN†

Abstract

Message Authentication Codes (MACs) are core algorithms deployed in virtually every security protocol in common usage. In these protocols, the integrity and authenticity of messages rely entirely on the security of the MAC; we examine cases in which this security is lost.

In this paper, we examine the notion of “reforgeability” for MACs, and motivate its utility in the context of {power, bandwidth, CPU}-constrained computing environments. We first give a definition for this new notion, then examine some of the most widely-used and well-known MACs under our definition in a variety of adversarial settings, finding in nearly all cases a failure to meet the new notion. We examine simple counter-measures to increase resistance to reforgeability, using state and truncating the tag length, but find that both are not simultaneously applicable to modern MACs. In response, we give a tight security reduction for a new MAC, WMAC, which we argue is the “best fit” for resource-limited devices.

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

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

† Google Inc., Mountain View, CA 94043, USA. E-mail: martin.cochran@gmail.com

Contents

1	Introduction	1
2	Preliminaries	5
3	A Fast, Stateful MAC with Short Tags	7
4	Conclusions	12
A	Attacks	16
A.1	Blockcipher Based MACs	16
A.2	Padding Attacks	17
A.3	Effects of Adding State	19
A.4	Attacks on Carter-Wegman MACs	20
B	Details of the hash127 Attack	25
C	A Bound for C	26

1 Introduction

MESSAGE AUTHENTICATION CODES. Message authentication codes (MACs) are the most efficient algorithms to guarantee message authenticity and integrity in the symmetric-key setting, and as such are used in nearly all security protocols. 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 $f : \Sigma^* \rightarrow \{0, 1\}^t$ 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, and for many scenarios this is the correct approach, but there are several settings where we might want to think about reforgeability nonetheless.

- 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.
- Streaming video applications might use a low-security MAC with the idea that forging one frame would hardly be noticeable 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.
- VOIP is another setting where reforgeability is arguably more appropriate than current MAC security models. In this setting, a forged packet probably only corresponds to a fraction of a second of sound and is relatively harmless.

In all cases, if parameters are chosen correctly so that an attacker’s best strategy is to guess tags, the overwhelming number of incorrect guesses can be used to inform users in situations where a forged packet could potentially have serious consequences.

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, partly as a result of the posting of an earlier version of this paper on eprint.iacr.org, the question of reforgeability has arisen in newsgroups, online

MAC scheme	Expected queries for j forgeries	Succumbs to padding attack	Succumbs to other attack	Message freedom
CBC MAC	$C_1 + j$		✓	$m - 2$
EMAC	$C_1 + j$	✓	✓	$m - 2$
XCBC	$C_1 + j$	✓	✓	$m - 2$
PMAC	$C_1 + j$		✓	1
ANSI retail MAC	$C_1 + j$	✓	✓	$m - 2$
HMAC	$\sum_i C_i/2^i + j$	✓		$m - 1$

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. With a general birthday attack, the first collision is expected at around $2^{n/2}$ MAC queries, although the exact number for each scheme can differ somewhat. The last column gives the number of freely-chosen message blocks in the forgery.

discussions, and the fact that industry is now specifically requesting reforgeability resistant MACs [32] lends support to this.

MAIN RESULTS. In this paper we conduct a systematic study of reforgeability, treated first in the literature by McGrew and Fluhrer [33]. 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 for all stateless schemes and many stateful schemes there exists an attack that enables efficient generation of forgeries given knowledge of an existing collision in tags. 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. For each stateful scheme where we could not find an attack, we then turned our attentions to another related problem: nonce misuse. That is, if nonces are reused with the same key, can we forge multiple times? The answer is an emphatic “yes.” For many of these MACs only a single protocol error is required to break the security; querying to the birthday bound is unnecessary.

Figure 1 and Figure 2 give a synopsis of our findings. In most cases, 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. For some Wegman-Carter-Shoup (WCS) [10, 38] MACs, the attack is more severe: nonce misuse yields the universal hash family instance almost immediately.

- **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 [5], XCBC [13], 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” [35] that allows arbitrary blocks to be appended to colliding pairs at the cost of one additional MAC query.
- **PMAC [14].** 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 [7]/ Poly1305[9].** Hash127 and Poly1305 are polynomial-hashes based on evaluating polynomials over the fields $\mathbb{Z} \bmod 2^{127} - 1$ and $\mathbb{Z} \bmod 2^{130} - 5$, respectively. In the FH paradigm, 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 [6] or the Cantor-Zassenhaus algorithm [17]. In the WCS paradigm (in which Poly1305-AES is defined), nonce misuse can be similarly devastating: a single repeated nonce reveals the key.

UHF in FH mode	Expected queries for j forgeries	Reveals key	Queries for key recovery
hash127/Poly1305	$C_1 + \log m + j$	✓	$C_1 + \log m$
VMAC	$C_1 + 2j$		
Square Hash	$C_1 + 2j$	✓	mC_1
Topelitz Hash	$C_1 + 2j$		
Bucket Hash	$C_1 + 2j$		
MMH/NMH	$C_1 + 2j$		

UHF in WCS mode with nonce misuse	Expected queries for j forgeries	Repeated nonce	Reveals key	Queries for key recovery
hash127/Poly1305	$2 + \log m + j$	1	✓	$2 + \log m$
VMAC	$C_1 + 2j$	$C_1 + j$		
Square Hash	$3m + j$	m	✓	$3m$
Topelitz Hash	$2j + 2$	1		
Bucket Hash	$2j + 2$	1		
MMH/NMH	$2m + j$	m	✓	$2m$

Figure 2: Results for Carter-Wegman MACs. The top table lists 6 well-known universal hash families, each made into a MAC via the FH construction [18, 42] where the hash family is composed with a pseudorandom function to produce the MAC tag. These similarly succumb to reforgeability attacks after a collision in the output tags, with hash127/Poly1305 and Square-Hash surrendering their key in the process. The last column gives the expected number of queries for key recovery, where possible. The bottom table considers the same hash families in the Wegman-Carter-Shoup (WCS) [10, 38] paradigm (the most prominent MAC paradigm for ϵ -AU hash families), but where nonces are misused and repeated. With many families, only one repeated nonce query is enough to render the MAC totally insecure. Others reveal the key with a few more queries using the same nonce. See [27] for further attacks on these and other hash families in a similar setting.

- **Square Hash [23].** Square-Hash is another fast-to-compute universal hash function family suggested for use in MACs. Once again, in the FH paradigm 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 A.4 where the scheme is described in full. In the WCS paradigm, nonce reuse also reveals the key after just a handful of out-of-protocol repeated nonces.
- **Remaining UHFs.** For each of the remaining universal hash function families we examine [19, 26, 30, 36] 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 [37]. We also analyze the UHFs under the Wegman-Carter-Shoup mode of operation with misuse of nonces, finding similar weaknesses. Handschuh and Preneel have improved and extended many of attacks found here in [27].

After an earlier draft of this paper appeared on eprint, many of the attacks in Figure 1 and Figure 2 were subsequently improved in [27] by Handschuh and Preneel. In light of this, we have moved the attack details to the appendix. Please refer there or to the other literature on the subject [16, 27, 33, 35].

These attacks 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 nonce inserted in a natural manner, to the schemes above. We show, however, that this approach can be insufficient or insecure when subtly misused. Another approach would be to use a stateless MAC such as HMAC, and truncate the output so a collision in tags does not expose some exploitable internal information. However, this is also somewhat unsatisfactory because all the fastest MACs are stateful WCS-style MACs where truncation severely reduces the security.

We therefore devised a new (stateful) scheme, WMAC, that allows nonce reuse and where for most parameter sizes guessing the tag is the best reforgeability strategy. The scheme is described fully in Section 3 but briefly it works as follows.

Let \mathcal{H} be some ϵ -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 $\rho \xleftarrow{\$} \mathcal{R}$ and $h \xleftarrow{\$} \mathcal{H}$; the shared key is (ρ, h) . 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 $(cnt, \rho(\langle cnt \rangle_b \parallel h(M)))$. To verify a received message M with tag (i, Tag) , the verifier computes $\rho(\langle i \rangle_b \parallel h(M))$ and ensures it equals Tag .

WHY WMAC? There are essentially four parameters which must be balanced when choosing a suitable MAC: speed, security, tag length, and deployment feasibility. WCS MACs provide excellent performance on the first two items, but require long tags and absolutely non-repeatable nonces (which also increases the tag length), a potential deployment problem where the state might have to be consistent across several machines. Stateless MACs, whose tags may be truncated without degrading security and therefore tend to do well on the last two items, lag behind on the first two.

WMAC can be seen as a compromise between the two sets of MACs. It has the speed of the fastest WCS MACs but the tag length may be truncated appropriately and nonces may be reused. A fixed nonce may be used for all queries if desired, effectively yielding the FH [18, 42] scheme as a special case. At the other extreme end, nonces are never repeated and WMAC retains a high degree of security comparable to the WCS setting. For most real-world applications that may already have implicit nonces (via the underlying networking protocol, eg) and that could use the added security benefits from nonces but do not want to enforce nonce uniqueness, WMAC is the best solution.

As an example, consider the following concrete WMAC instantiation. Let $\epsilon \leq 2^{-82}$, $b = 8$, and our PRF will be AES truncated to 24 bits. Then after 2^{32} signing queries and 2^{24} verification queries, one forgery is expected (from guessing the output of the PRF). The hash family can be a variant of the VHASH used in VMAC-128, so that the speed of the family is comparable to VMAC-128.¹ Moreover, the total tag length, including the nonce is only 32 bits. There is no efficient MAC which, using 32 bits for both the tag and nonce, can safely MAC as many messages with so few expected forgeries. (Note that the nonce greatly helps the security in this case; without it an expected 64 forgeries would be possible.)

Because nonce values may be reused, it is possible to use incremental verification in WMAC. In some constrained environments like sensor networks, it is beneficial to have the option to pre-screen incoming MAC tags. First, a low cost check is performed on the message/tag pair. Only if that check is passed will the more expensive MAC be computed. This can be useful when an attacker tries to deplete the power resources in a sensor node by spoofing a large number of messages. The attacker is not necessarily interested in forging messages, but merely requiring the sensor node to perform many expensive calculations. The nonce value may be used as the tag for this first check, computed using a weaker but fast-to-compute MAC. When combined with WMAC's computational efficiency and short tag length, this property makes the scheme ideal for these constrained environments.

We stress that although WMAC offers good tradeoffs for resource-constrained environments where some forgeries may be acceptable, it is still susceptible to attacks that exploit some bad event that occurs during operation, usually related to the value of ϵ for the ϵ -almost universal hash family used. To be clear, the attacks from [27] still apply and indeed come within a constant factor of matching the bound given in our security reduction.²

RELATED WORK. David McGrew and Scott Fluhrer have also done some work [33] on a similar subject, produced concurrently with our work but published earlier. They examine MACs with regard to multiple forgeries, although they view the subject from a different angle. They show that for HMAC, CBC MAC, and GMAC from the Galois Counter Mode (GCM) of operation for blockciphers [31], 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

¹Dan Bernstein has proposed [8] an almost-universal hash family which should be as fast or faster than VMAC-64, but which uses a much smaller key than VMAC. Bernstein's hash would use fewer multiplications and additions than VMAC-128, although those operations are done in some field \mathcal{F} , not modulo 2^n .

²Our bound also highlights interesting behavior with a verification query-only attack when the length of the tag is much smaller than $\lg(\epsilon^{-1})$. This case is also matched by essentially the attacks from [27].

MACs. Although our focus is somewhat different, our work complements their paper by showing their techniques and bounds apply to all major MACs.

Handschuh and Preneel investigated attacks on ϵ -almost universal hash families used in Wegman-Carter-Shoup mode MACs, and found new classes of attacks [27]. Their attacks improve on ours in several ways, probably the most significant of which is that they do not require misuse of nonce values to work.

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 $s \stackrel{\$}{\leftarrow} S$ 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 $M = M_1 \parallel M_2 \parallel \dots \parallel M_m$ such that $|M_1| = |M_2| = \dots = |M_m|$. Let $\text{Rand}(l, L) = \{f \mid f : \{0,1\}^l \rightarrow \{0,1\}^L\}$ 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 [18]) Fix a domain \mathcal{D} and range \mathcal{R} . A finite multiset of hash functions $\mathcal{H} = \{h : \mathcal{D} \rightarrow \mathcal{R}\}$ 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 $\mathcal{H} = \{h : \mathcal{D} \rightarrow \mathcal{R}\}$ 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 [30], Stinson [40]) 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 $\mathcal{H} = \{h : \mathcal{D} \rightarrow \mathcal{R}\}$ is said to be ϵ -**Almost XOR Universal** (ϵ -AXU) if for every $x, y \in \mathcal{D}$ and $z \in \mathcal{R}$ with $x \neq y$, $\Pr_{h \in \mathcal{H}}[h(x) \oplus h(y) = z] \leq \epsilon$.

Throughout the paper we assume that a given value of ϵ for an ϵ -AU or ϵ -AXU family includes a parameter related to the length of the messages. If we speak of a fixed value for ϵ , then we implicitly specify an upper bound on this length.

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 $\text{Tag} = \text{MAC}_K(M)$. Algorithm VF is the ‘verification’ algorithm such that on input $K \in \mathcal{K}$, $M \in \mathcal{D}$, and $\text{Tag} \in \{0,1\}^\tau$, outputs a bit. We interpret 1 as meaning the verifier *accepts* and 0 as meaning it *rejects*. This computation is denoted $\text{VF}_K(M, \text{Tag})$. Algorithm MAC can be probabilistic, but VF typically is not. A restriction is that if $\text{MAC}_K(M) = \text{Tag}$, then $\text{VF}_K(M, \text{Tag})$ must output 1. If $\text{MAC}_K(M) = \text{MAC}_K(M')$ 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 $\Pi = (\text{MAC}, \text{VF})$ be a message authentication code, and let A be an adversary. We consider the following experiment:

Experiment $\text{Exmt}_{\Pi}^{j\text{uf-cma}}(A, j)$

$K \stackrel{\$}{\leftarrow} \mathcal{K}$

Run $A^{\text{MAC}_K(\cdot), \text{VF}_K(\cdot, \cdot)}$

If A made j distinct verification queries (M_i, Tag_i) , $1 \leq i \leq j$, such that

- $\text{VF}_K(M_i, \text{Tag}_i) = 1$ for each i from 1 to j
 - A did not, prior to making verification query (M_i, Tag_i) , query its 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

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

For any $q_s, q_v, \mu_s, \mu_v, \text{Time} \geq 0$ we overload the above notation and define

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

where the maximum is over all adversaries A that have time-complexity at most Time , 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, $\text{Adv}_{\Pi}^{\text{juf-cma}}(t, q_s, \mu_s, q_v, \mu_v, j) \leq \epsilon$, 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 nonce value. Our model will let the adversary control the nonce, but limit the number of MAC queries per nonce. Setting this limit above 1 will simulate a protocol error where nonces are re-used in computing tags.

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 $\text{Tag} = \text{MAC}_K(M, S)$. Algorithm VF is the verification algorithm such that on inputs $K \in \mathcal{K}$, $M \in \mathcal{D}$, $\text{Tag} \in \{0, 1\}^{\tau}$, and $S \in \mathcal{S}$, VF outputs a bit, with 1 representing accept and 0 representing reject. This computation is denoted $\text{VF}_K(M, S, \text{Tag})$. A restriction on VF is that if $\text{MAC}_K(M, S) = \text{Tag}$, then $\text{VF}_K(M, S, \text{Tag})$ must output 1.

As discussed later, all our attacks on stateless MACs work by examining the event of a collision in tag values, by virtue of the birthday phenomenon or otherwise. With stateful MACs an adversary may see collisions in tags, but the state mitigates, and in most cases neutralizes, any potentially damaging information leaked in such an event. With that in mind, we will consider two different security models with regard to stateful MACs. In one, we treat stateful MACs as intended: nonces are not repeated among queries, but repeated nonces may be used with verification queries. Many MACs we examine have security proofs in this model, so it is not surprising that they perform well, even with short tags. Others don't, and we provide the analysis.

We also provide analysis for a plausible and interesting protocol error: that in which nonces are reused. This can happen in several reasonable scenarios: 1) the nonce is a 16- or 32-bit variable, and overflow occurs unnoticed, and 2) the same key is used across multiple virtualized environments. This latter case may happen when MACs in differing virtualized environments are keyed with the same entropy pools, or one environment is cloned from another.

These protocol misuses are captured formally by allowing an adversary a maximum of α queries per nonce between the two oracles. For most MACs we examine, α need only be 2 for successful re forgery attacks.

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

Experiment $\text{Exmt}_{\Pi}^{\text{jsuf-cma}}(A, j, \alpha)$

$K \xleftarrow{\$} \mathcal{K}$

Run $A^{\text{MAC}_K(\cdot), \text{VF}_K(\cdot, \cdot)}$

If A made j distinct verification queries (M_i, s_i, Tag_i) , $1 \leq i \leq j$, such that

- $\text{VF}_K(M_i, s_i, \text{Tag}_i) = 1$ for each i from 1 to j

- A did not, prior to making verification query (M_i, s_i, Tag_i) , query its MAC oracle with (M_i, s_i)
 - A did not make more than α queries to MAC_K with the same nonce.
- Then return 1 else return 0

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

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

For any $q_s, q_v, \mu_s, \mu_v, \text{Time}, j, \alpha \geq 0$ we let

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

where the maximum is over all adversaries A that have time-complexity at most Time , make at most q_s MACing queries, the sum of those lengths is at most μ_s , where no more than α queries were made per nonce, and make at most q_v verification queries where the sum of the lengths of the messages involved is at most μ_v .

If, for a given MAC, $\mathbf{Adv}_{\Pi}^{\text{jsuf-cma}}(t, q_s, \mu_s, q_v, \mu_v, j, \alpha) \leq \epsilon$, then we say that MAC is (j, ϵ) -secure. For the case $j = 1$, the scheme is simply ϵ -secure.

3 A Fast, Stateful MAC with Short Tags

For some stateful MACs discussed in the attacks section, we found no attack, and others are accompanied by a proof of security. Similarly, tag truncation is a simple technique which may be used to ensure that security is retained well after one starts seeing collisions in tags. Perhaps we should be satisfied and consider our search for reforgeability-resistant MACs complete. However, both of these techniques have drawbacks for the applications in mind which require very short tags. Namely, the nonce value must be transmitted with each query, and tag truncation may not be used on the fastest MACs without seriously degrading security.³

It is with these thoughts in mind, and with newfound knowledge of the perils associated with nonce misuse in WCS MACs, that we designed WMAC. WMAC boasts speed comparable to VMAC/Poly1305, can use much shorter tags, and is the first MAC we know of to use repeating nonces, a side effect of which is shorter tags.

WMAC. Let $\mathcal{H} = \{h : \mathcal{D} \rightarrow \mathcal{R}\}$ be a family of ϵ -AU hash functions and let $F : \mathcal{K} \times \mathcal{T} \times \mathcal{R} \rightarrow \{0, 1\}^n$ be a PRF. We define

$$\text{WMAC}[\mathcal{H}, F]_{h, F_K}^t(x) = F_K(t, h(x)),$$

where $t \in \mathcal{T}$, $h \xleftarrow{\$} \mathcal{H}$, $K \xleftarrow{\$} \mathcal{K}$, and $x \in \mathcal{D}$. Informally, once keyed with the selection of $K \in \mathcal{K}$ and AU hash instance h , WMAC accepts a message x and nonce t as inputs and returns $F_K(t, h(x))$ as the tag.

NONCES IN WMAC. WMAC's nonce use can be considered as "flexible" in the sense that the security analysis is done for different uses. To model this, we are mainly interested in an adversary of somewhat limited capability, that is, an adversary which can make at most α signing queries for each nonce $t \in \mathcal{T}$. The adversary's verification queries per nonce are not similarly bounded. We call such an adversary α -limited, and define $\mathbf{Adv}_{\Pi}^{\text{jsuf-cma}}(q, t, \alpha)$ be the maximum of $\mathbf{Adv}_{\Pi}^{\text{jsuf-cma}}(A)$ over every α -limited adversary A which makes at most $q = q_s + q_v$ oracle queries (q_s to the signing oracle and q_v to the verification oracle) and halts within time Time . We say that Π is secure as an α -limited MAC, if $\mathbf{Adv}_{\Pi}^{\text{jsuf-cma}}(q, t, \alpha)$ is negligibly small for any reasonably large q and Time .

As an example, the FH and FCH [18, 42] modes of operation are special cases of WMAC where α is set to q_s and 1, respectively.

³Truncating the tag of VMAC or Poly1305-AES by t bits also effectively grows ϵ for the ϵ -AU family by a multiplicative factor of 2^t . If these MACs were to be revised into FH mode, truncation would be possible, but without nonces they succumb to attacks covered in this paper, and with nonces ϵ needs to be unacceptably reduced to make room for the nonce input.

Theorem 6 For any α -limited adversary A of WMAC which makes at most $q = q_s + q_v$ queries in time Time , there exists an adversary B of F such that

$$\begin{aligned} \text{Adv}_{\text{WMAC}}^{\text{jsuf-cma}}(A) &\leq \text{Adv}_F^{\text{prf}}(B) + \frac{\epsilon(\alpha - 1)q_s}{2} + \\ &\quad \frac{\epsilon}{2^{n-1}} \left(q_v^2 + q_v q_s + \max\{2^n, q^{\frac{1}{2}} 2^{\frac{n}{2}+3}\} q_v \right) + \delta(j, n, q_v). \end{aligned}$$

and where B makes at most q queries, using time proportional to $\text{Time} + \text{Hash}(q)$, where $\text{Hash}(1)$ is the time to compute $h(M)$ for some message $M \in \mathcal{D}$ and $h \xleftarrow{\$} \mathcal{H}$. The term $\delta(j, n, q_v)$ is defined as

$$\sum_{k=j}^{|S|} \sum_{X \in S_k} \left[\prod_{x' \in S: x' \notin X} \left(1 - \frac{q_{v,x'}}{2^n} \right) \prod_{x \in X} \left(\frac{q_{v,x}}{2^n} \right) \right]$$

where S is the set of distinct message-tag pairs seen in all verification queries, S_k is the set of k -tuples in S , and for an element $x \in S$, $q_{v,x}$ is the number of verification queries made for that element.

DISCUSSION OF THE BOUND AND EXPECTED NUMBER OF FORGERIES. McGrew and Fluhrer discuss the expected number of forgeries for GMAC (a WCS MAC)[31], CBC MAC, and HMAC in terms of ϵ , n , and q . Our specific attacks complement their analysis by showing their methods apply to all major stateful and stateless MACs. Essentially, they show that for stateless MACs, the expected number of forgeries is $cq^3 2^{-n} + \mathcal{O}(q^4 2^{-2n})$, where n is output size of the blockcipher or hash function and c is a constant. For WCS MACs, they show the expected number of forgeries is $cq^2 \epsilon + \mathcal{O}(q^3 \epsilon^2)$.

We believe this sort of analysis should supplant the current definition of MAC security for the simple reason that it more accurately quantifies the risks for MACing q messages over the lifetime of one key and, in the case of our bound in particular, makes the bound more easily understood. Rather than giving the traditional security bound and suggesting the number of queries be “well below” a certain value ($2^{n/2}$, usually), producing a specific expected number of forgeries is much superior.

And in this spirit, we give a formula for the expected number of forgeries for WMAC, which also helps to understand the rather obtuse bound in theorem 6. For a given MAC scheme $\Pi = (\text{MAC}, \text{VF})$, let $E(\text{Forge}_{\Pi}, q_s, q_v)$ denote the expected number of forgeries when q_s queries are allowed to the MAC oracle and q_v queries are allowed to the VF oracle.

Following [33], we will assume WMAC uses an ideal random function as the PRF. Unless q_v is unreasonably large, the expected number of forgeries is overwhelmingly influenced by the chance that an adversary sets *bad* to true during one of the q_s queries to the MAC oracle. If this occurs, we give the adversary q_v forgeries. There is a small chance *bad* is set to true in the verification phase and to simplify the analysis we admit q_v forgeries in this case as well. Thus, we bound the expected number of forgeries as q_v times the probability that *bad* is set to true. Finally, we must consider the expected number of forgeries when the adversary merely guesses the correct outputs of the ideal random function, which is $q_v 2^{-n}$. Thus,

$$E(\text{Forge}_{\text{WMAC}}, q) \leq \frac{\epsilon q_v q_s (\alpha - 1)}{2} + \frac{q_v \epsilon}{2^{n-1}} \left(q_v^2 + q_v q_s + 2^{n/2+3} q_v \sqrt{q} \right) + q_v 2^{-n}.$$

It is this formula which is used to give figures in the example from section 1. Note that when $q = q_s = q_v$, letting α take on values in $\{1, q\}$ gives bounds similar to those from [33].

Proof: Without loss of generality, we may assume that A doesn’t ask the same signing query twice, and that A makes all signing queries before making any verification queries.⁴ Our adversary B has access to an oracle $Q(t, x)$. We construct B , which runs A as a subroutine, by directly simulating the oracles A expects. That is, in the startup phase, B randomly selects $h \xleftarrow{\$} \mathcal{H}$. It then runs A , responding to A ’s signing query (t, M) by querying its oracle at $(t, h(M))$ and returning the answer to A . Similarly, B responds to a verification query (t, M, Tag) by querying its oracle at $(t, h(M))$ and returning 1 if the answer is equal to Tag , 0 otherwise. After A has completed all queries, B outputs the same bit as A .

⁴This condition is not required by our security reduction— an adversary may make queries in any order she wishes — but for ease of notation we adopt it.

Consider the games G_0 and G_1 in figure 3, where Game G_1 includes the boxed statement. The function `InitializeMap` takes as arguments a map name, a domain, and a range, and initializes a map with the input name where every map lookup returns \perp .

0	Procedure Initialize
	$V \leftarrow \emptyset, h \xleftarrow{\$} \mathcal{H}, \rho \xleftarrow{\$} \text{Rand}(\mathcal{T} \times \mathcal{R}, \{0, 1\}^n),$
	<code>InitializeMap</code> (Map, $\mathcal{T} \times \mathcal{D}, \mathcal{T} \times \mathcal{R}$), <code>InitializeMap</code> (Map _o , $\mathcal{T} \times \mathcal{D}, \mathcal{T} \times \mathcal{R}$)
	Procedure MAC(t, x)
1	$v \leftarrow h(x)$
2	If $(t, v) \in V$ then { $bad \leftarrow \text{true},$ $(t, v) \xleftarrow{\\$} \mathcal{T} \times \mathcal{R} \setminus V$ }
3	$V \leftarrow V \cup (t, v)$
4	return $\rho(t, v)$
	Procedure VF(t, x, Tag)
5	If <code>Map</code> [(t, x)] = \perp then {
6	$v \leftarrow h(x),$ <code>Map</code> [(t, x)] $\leftarrow (t, v)$
7	If $(t, v) \in V$ then { <code>Map</code>_o[(t, x)] $\leftarrow (t, v),$ <code>Map</code>[(t, x)] $\xleftarrow{\\$} \mathcal{T} \times \mathcal{R} \setminus V,$ <math>(t, v) \leftarrow \text{Map}[(t, x)] }</math>
8	$V \leftarrow V \cup (t, v)$
9	}
10	If <code>Map</code> _o [(t, x)] $\neq \perp$ then {
11	If <code>Tag</code> = $\rho(\text{Map}_o[(t, x)])$ or <code>Tag</code> = $\rho(\text{Map}[(t, x)])$ then { $bad \leftarrow \text{true}$ }
	}
	return <code>Tag</code> = $\rho(\text{Map}[(t, x)])$

Figure 3: Game G_0 and Game G_1

Clearly, A^{G_0} corresponds to the experiment where A is given access to the signing oracle $\rho(t, h(x))$ and verification oracle $\rho(t, h(x)) = \text{Tag}$, and A^{G_1} corresponds to the experiment where the tags for A 's queries (either signing or verification), are chosen as uniform random outputs. Because A doesn't ask the same signing query twice and by the way we constructed B , this is precisely the answers A will get when the signing oracle is a uniform random function and the verification oracle behaves similarly. Finally, when B 's oracle is F_K , B simulates the oracle A expects exactly. Therefore,

$$\begin{aligned}
\text{Adv}_F^{\text{prf}}(B) &= \Pr[1 \leftarrow A^{\text{WMAC}_{K,h}}] - \Pr[1 \leftarrow A^{G_0}] \\
&= \Pr[1 \leftarrow A^{\text{WMAC}_{K,h}}] - \Pr[1 \leftarrow A^{G_1}] + \Pr[1 \leftarrow A^{G_1}] - \Pr[1 \leftarrow A^{G_0}] \\
&\geq \Pr[1 \leftarrow A^{\text{WMAC}_{K,h}}] - \Pr[1 \leftarrow A^{G_1}] - \Pr[A^{G_1} \text{ sets } bad] \\
&= \text{Adv}_{\text{WMAC}}^{\text{jsuf-cma}}(A) - \Pr[1 \leftarrow A^{G_1}] - \Pr[A^{G_1} \text{ sets } bad],
\end{aligned}$$

since G_0 and G_1 are identical-until-*bad* games.

The term $\delta(j, n, q_v)$ represents the probability of A 's success when presented with the oracle of game G_1 . In this case, a verification query (t_i, x_i, τ_i) with a new message-nonce pair (t_i, x_i) 'succeeds' iff $\rho(t_i, h(x_i)) = \tau_i$, and this happens with probability 2^{-n} . Similarly, for ℓ verification queries made with (t_i, x_i) as the message-tag pair, the total success probability is $\ell/2^n$. By summing over all possibilities for correct and incorrect guesses, we have that

$$\sum_{k=j}^{|S|} \sum_{X \in S_k} \left[\prod_{x' \in S: x' \notin X} \left(1 - \frac{q_{v,x'}}{2^n} \right) \prod_{x \in X} \left(\frac{q_{v,x}}{2^n} \right) \right].$$

(A much more intuitive grasp of this term can be obtained by considering its expected value, $q_v 2^{-n}$. This can be seen by the fact that the expected number of forgeries for any one message tag pair $x \in S$ is $q_{v,x} 2^{-n}$; the value follows by linearity of expectation of independent events and the fact that $q_v = \sum_{x \in S} q_{v,x}$.)

Now we must bound the probability that *bad* is set to true, but first we go through some output distribution-preserving game transitions to make the analysis easier. The difference between Game G_1 and Game G_2 is that in G_2 , `MAC`(t, x) returns a uniform random value τ from $\{0, 1\}^n$ and `VF`(t, x, Tag) chooses its outputs

	Procedure Initialize
0	$V \leftarrow \emptyset, h \xleftarrow{\$} \mathcal{H}, \rho \xleftarrow{\$} \text{Rand}(\mathcal{T} \times \mathcal{R}, \{0, 1\}^n), \text{InitializeMap}(\text{Map}, \mathcal{T} \times \mathcal{D}, \mathcal{T} \times \mathcal{R}),$ $\text{InitializeMap}(\text{Map}_o, \mathcal{T} \times \mathcal{D}, \mathcal{T} \times \mathcal{R}), \text{InitializeMap}(O, \mathcal{T} \times \mathcal{R}, \{0, 1\}^n)$
	Procedure $Q(t, x)$
1	$v \leftarrow h(x)$
2	If $(t, v) \in V$ then { $bad \leftarrow \text{true}, (t, v) \xleftarrow{\$} \mathcal{T} \times \mathcal{R} \setminus V$ }
3	$V \leftarrow V \cup (t, v), O[(t, v)] \xleftarrow{\$} \{0, 1\}^n$
4	return $O[(t, v)]$
	Procedure $\text{VF}(t, x, \text{Tag})$
5	If $\text{Map}[(t, x)] = \perp$ then {
6	$v \leftarrow h(x), \text{Map}[(t, x)] \leftarrow (t, v)$
7	If $(t, v) \in V$ then { $\text{Map}_o[(t, x)] \leftarrow (t, v), \text{Map}[(t, x)] \xleftarrow{\$} \mathcal{T} \times \mathcal{R} \setminus V, (t, v) \leftarrow \text{Map}[(t, x)]$ }
8	$V \leftarrow V \cup (t, v), O[(t, v)] \xleftarrow{\$} \{0, 1\}^n$
	}
9	If $\text{Map}_o[(t, x)] \neq \perp$ then {
10	If $\text{Tag} = O[\text{Map}_o[(t, x)]]$ or $\text{Tag} = O[\text{Map}[(t, x)]]$ then { $bad \leftarrow \text{true}$ }
	}
11	return $\text{Tag} = O[\text{Map}[(t, x)]]$

Figure 4: Game G_2

in line 8 from uniform random values from $\{0, 1\}^n$. But in Game G_1 , $\rho(t, v)$ is computed for all distinct (t, v) in line 4 and in line 11 $\rho(\text{Map}[(t, x)])$ is computed for all distinct values of $\text{Map}[(t, x)]$ when distinct (t, x) values are used. Therefore the two games are identical. In Game G_3 , we clean things up by removing the unnecessary ρ , and removing the statement $(t, v) \xleftarrow{\$} \mathcal{T} \times \mathcal{R} \setminus V$. This is possible because this occurs after $bad \leftarrow \text{true}$.

	Procedure Initialize
0	$V \leftarrow \emptyset, O \leftarrow \emptyset, h \xleftarrow{\$} \mathcal{H}, \text{InitializeMap}(\text{Map}, \mathcal{T} \times \mathcal{D}, \mathcal{T} \times \mathcal{R}),$ $\text{InitializeMap}(\text{Map}_o, \mathcal{T} \times \mathcal{D}, \mathcal{T} \times \mathcal{R}), \text{InitializeMap}(O, \mathcal{T} \times \mathcal{R}, \{0, 1\}^n)$
	Procedure $Q(t, x)$
1	$v \leftarrow h(x)$
2	If $(t, v) \in V$ then { $bad \leftarrow \text{true}$ }
3	$V \leftarrow V \cup (t, v), O[(t, v)] \xleftarrow{\$} \{0, 1\}^n$
4	return $O[(t, v)]$
	Procedure $\text{VF}(t, x, \text{Tag})$
5	If $\text{Map}[(t, x)] = \perp$ then {
6	$v \leftarrow h(x), \text{Map}[(t, x)] \leftarrow (t, v)$
7	If $(t, v) \in V$ then { $\text{Map}_o[(t, x)] \leftarrow (t, v), \text{Map}[(t, x)] \xleftarrow{\$} \mathcal{T} \times \mathcal{R} \setminus V, (t, v) \leftarrow \text{Map}[(t, x)]$ }
8	$V \leftarrow V \cup (t, v), O[(t, v)] \xleftarrow{\$} \{0, 1\}^n$
	}
9	If $\text{Map}_o[(t, x)] \neq \perp$ then {
10	If $\text{Tag} = O[\text{Map}_o[(t, x)]]$ or $\text{Tag} = O[\text{Map}[(t, x)]]$ then { $bad \leftarrow \text{true}$ }
	}
11	return $\text{Tag} = O[\text{Map}[(t, x)]]$

Figure 5: Game G_3

In Game G_4 , we first generate all the random answers to the queries of A , and on i th signing query, save the query and just return the i th random answer. The verification queries are handled similarly by using the saved values. We can check whether we should set bad at the finalization step, using the saved query values. Clearly, all games G_2, G_3 , and G_4 preserve the probability that bad gets set. Therefore,

$$\text{Adv}_{\text{WMAC}}^{\text{jsuf-cma}}(A) \leq \text{Adv}_F^{\text{prf}}(B) + \Pr[A^{G_4} \text{ sets } bad] + \delta(j, n, q_v).$$

We will use the fact that

$$\Pr[A^{G_4} \text{ sets } bad] \leq \Pr[A^{G_4} \text{ sets } bad \text{ in line 6}] + \Pr[A^{G_4} \text{ sets } bad \text{ in line 8}].$$

It is easy to analyze the probability $\Pr[A^{G_4} \text{ sets } bad \text{ in line 6}]$; In Game G_4 , the adversary A gets no information about h at all, and the random variables t_i and x_i are independent from h . Let's enumerate all

	Procedure Initialize
0	$h \stackrel{\$}{\leftarrow} \mathcal{H}, (\tau_1, \dots, \tau_{q_s + \#q_v}) \stackrel{\$}{\leftarrow} (\{0, 1\}^n)^{q_s + \#q_v}, i \leftarrow 0,$ InitializeMap($O, \mathcal{T} \times \mathcal{R}, \{0, 1\}^n$)
	Procedure $Q(t, x)$
1	$i \leftarrow i + 1, t_i \leftarrow t, x_i \leftarrow x, O[(t, x)] \leftarrow \tau_i$
2	return τ_i
	Procedure VF(t, x, Tag)
3	If $O[(t, x)] = \perp$ then {
4	$i \leftarrow i + 1, t_i \leftarrow t, x_i \leftarrow x, O[(t, x)] \leftarrow \tau_i, \text{Tag}_i \leftarrow \text{Tag}$
	}
5	return $\tau_i = \text{Tag}_i$
	Procedure Finalize
6	If $(t_i, h(x_i)) = (t_j, h(x_j))$ for some $i < j \leq q_s$, then { <i>bad</i> \leftarrow true }
7	If $(t_i, h(x_i)) = (t_j, h(x_j))$ for some $i < j, q_s < j$ then {
8	If $O[(t_i, x_i)] = \text{Tag}_j$ or $O[(t_j, x_j)] = \text{Tag}_j$ then { <i>bad</i> \leftarrow true }
	}

Figure 6: Game G_4

the elements of \mathcal{T} as $T_1, \dots, T_{|\mathcal{T}|}$, and let $q_{s,i}$ be the number of signing queries (t, x) such that $t = T_i$. Then,

$$\begin{aligned} \Pr[A^{G_4} \text{ sets } bad \text{ in line 6}] &\leq \sum_{i=1}^{|\mathcal{T}|} \epsilon \cdot \frac{q_{s,i}(q_{s,i} - 1)}{2} \leq \sum_{i=1}^{|\mathcal{T}|} \epsilon \cdot \frac{q_{s,i}(\alpha - 1)}{2} \\ &= \frac{\epsilon(\alpha - 1)}{2} \sum_{i=1}^{|\mathcal{T}|} q_{s,i} = \frac{\epsilon(\alpha - 1)q_s}{2}. \end{aligned}$$

We must also bound the probability $\Pr[A^{G_4} \text{ sets } bad \text{ in line 8}]$. The adversary A still learns no information about h , but we must account for an optimal tag guessing strategy with respect to *bad* being set to true. We first focus on the case where A does not guess multiple tags for a message-nonce pair and then handle the general case. For each value $k \in \mathcal{T}$ let S_k be the set of indices i such that $1 \leq i \leq q_s$ and $t_i = k$. Similarly, let V_k be the set of indices i such that $q_s < i \leq q_s + q_v$ and $t_i = k$. Let g be the number of correctly guessed tags during the verification phase. Let $X_k = \{x_i : i \in S_k \vee (i \in V_k \wedge \text{Tag}_i = \tau_i)\}$ and let $X_k^\tau = \{\tau_i : x_i \in X_k\}$. (Note that $\sum_{k \in \mathcal{T}} |X_k| = q_s + g$.) For any value $\tau \in \{0, 1\}^n$, let $G_k(\tau) = \{x_i : \tau_i \in X_k^\tau, \tau = \tau_i\}$. Let $C_k = \max\{|G_k(\tau)| : \tau \in X_k^\tau\}$ and $C = \max\{C_k\}$ and let E_b be the the event that A^{G_4} sets *bad* in line 8. Then,

$$\Pr[E_b] \leq \sum_{k \in \mathcal{T}} \sum_{i \in V_k} \left(\max_{\tau \in X_k^\tau} \Pr[h(x_i) = h(x) : x \in G_k(\tau)] \right) \quad (1)$$

$$+ \Pr[h(x_i) = h(x) : x \in X_k] \cdot \Pr[\text{Tag}_i = \tau_i] \quad (2)$$

$$+ \sum_{j \in V_k, j < i} \Pr[h(x_j) = h(x_i)] \cdot \Pr[\text{Tag}_j = \tau_j \vee \text{Tag}_j = \tau_i] \quad (3)$$

$$\leq \sum_{k \in \mathcal{T}} \sum_{i \in V_k} \left(\epsilon C_k + \epsilon |X_k| 2^{-n} + \sum_{j \in V_k, j < i} \epsilon 2^{-n+1} \right) \quad (4)$$

$$\leq \epsilon \sum_{k \in \mathcal{T}} \sum_{j=0}^{|V_k|-1} (C_k + (\alpha + g) 2^{-n} + j 2^{-n+1}) \quad (5)$$

$$\leq \epsilon \sum_{k \in \mathcal{T}} \left(|V_k| (C + (\alpha + g) 2^{-n}) + 2^{-n+1} \binom{|V_k|}{2} \right) \quad (6)$$

$$\leq \epsilon \left(q_v (C + (\alpha + g) 2^{-n}) + 2^{-n+1} \sum_{k \in \mathcal{T}} \binom{|V_k|}{2} \right) \quad (7)$$

$$\leq \epsilon \left(q_v(C + (\alpha + g)2^{-n}) + 2^{-n+1} \binom{q_v}{2} \right) \quad (8)$$

On a verification query (t_j, x_j, Tag_j) we consider two cases where the conditional on line 7 is met: $i \leq q_s$ and $q_s < i$. Also, on line 8, there are two events that may set *bad* to true: A 's guess may be correct for the unmodified output τ_i , or it may be a correct guess for the modified output τ_j . Suppose *bad* is set to true on line 8, then we distinguish these four events:

- $E_{1,j} : i \leq q_s$ and A 's guess was correct for the unmodified output.
- $E_{2,j} : i \leq q_s$ and A 's guess was correct for the modified output.
- $E_{3,j} : q_s < i$ and A 's guess was correct for the unmodified output.
- $E_{4,j} : q_s < i$ and A 's guess was correct for the modified output.

Then $\Pr[A^{G_4}$ sets *bad* in line 8] on the j -th query is

$$\Pr[E_{1,j} \vee E_{2,j} \vee E_{3,j} \vee E_{4,j}] \leq \Pr[E_{1,j}] + \Pr[E_{2,j}] + \Pr[E_{3,j}] + \Pr[E_{4,j}].$$

Lines (1) and (2) of the set of the equations denote $\Pr[E_{1,j}]$ and $\Pr[E_{2,j}]$, respectively, and line (3) contains $\Pr[E_{3,j} \vee E_{4,j}]$. The justification for line (1) is that an adversary's best strategy when $i \leq q_s$ is to guess the most frequently occurring tag returned during the signing phase (or a tag that is known by being guessed correctly during the verification phase). In lines (2) and (3) the adversary must try to guess an independent uniform random sample from 2^n values once the conditional is met. Line (4) upper bounds the probabilities for these events to occur, lines (5-7) simplify the equation, and the last inequality is justified by the fact that the quantity is maximized by making all verification queries with the same nonce.

Finally, with a simple argument we cover the case where during the verification phase multiple tags are guessed for a particular message-nonce pair. Since A learns nothing about h during the game, A has no way of learning which of its queries caused the conditional on line 7 to be true and gains no advantage from this approach. Indeed, the optimal strategy is to only make one verification query per message-nonce pair, so that the odds of line 8 being reached are increased with each query by forcing more values to be re-mapped as in line 7 of game G_3 .

Appendix C contains a bound for C for values of interest. In particular, $C \leq \max\{1, \frac{q_s+g}{2^n} + 15\sqrt{\frac{q_s+g}{2^{n/2}}}\}$ and the expected value of g is $q_v 2^{-n}$. Putting it together, we have

$$\begin{aligned} \Pr[A^{G_4} \text{ sets } bad] &\leq \frac{\epsilon(\alpha-1)q_s}{2} + \epsilon \left(q_v(C + (\alpha + g)2^{-n}) + 2^{-n+1} \binom{q_v}{2} \right) \\ &\leq \frac{\epsilon(\alpha-1)q_s}{2} + \epsilon \left(\frac{q_v^2}{2^n} + 2q_v C \right) \\ &\leq \frac{\epsilon(\alpha-1)q_s}{2} + \epsilon \left(\frac{q_v^2}{2^n} + \frac{q_v q_s}{2^{n-1}} + \frac{q_v^2}{2^{2n-1}} + \frac{15q_v \sqrt{q}}{2^{n/2}} \right) \\ &\leq \frac{\epsilon(\alpha-1)q_s}{2} + \frac{\epsilon}{2^{n-1}} \left(q_v^2 + q_v q_s + 2^{n/2+3} q_v \sqrt{q} \right). \end{aligned}$$

■

4 Conclusions

We have shown that for most MACs, forging multiple times is not much harder than forging once. We then find that two natural ways of improving resistance to reforgeability are, unfortunately, mutually exclusive

when applied to common MACs. WMAC, which aims to reconcile these two methods with a modern Carter-Wegman-Shoup MAC, is introduced and the security bounds given match the best known attacks [27]. WMAC provides parameter choices that yield constructions with varying security, speed, tag length, and use of state. For this flexibility, the inputs to WMAC are longer than other Wegman-Carter style MAC constructions and therefore messages take slightly longer to process.

Acknowledgments

John Black's work was supported by NSF CAREER-0240000 and a gift from the Boettcher Foundation. Many thanks to Aaram Yum and Yongdae Kim for their thoughtful insights and suggestions on earlier draft versions.

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 Koblitz [29], pp. 1–15.
- [3] BELLARE, M., KILIAN, J., AND ROGAWAY, P. The security of cipher block chaining. In Desmedt [22], 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] 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.
- [6] BERLEKAMP, E. R. Factoring polynomials over large finite fields. *Mathematics of Computation* 24 (1970), 713–735.
- [7] BERNSTEIN, D. Floating-point arithmetic and message authentication. Draft available as <http://cr.yp.to/papers/hash127.dvi>.
- [8] BERNSTEIN, D. Polynomial evaluation and message authentication. Draft available as <http://cr.yp.to/papers.html#pema>.
- [9] BERNSTEIN, D. J. The Poly1305-AES message-authentication code. In Gilbert and Handschuh [24], pp. 32–49.
- [10] BERNSTEIN, D. J. Stronger security bounds for Wegman-Carter-Shoup authenticators. In *EUROCRYPT* (2005), R. Cramer, Ed., vol. 3494 of *Lecture Notes in Computer Science*, Springer, pp. 164–180.
- [11] BIERBRAUER, J., JOHANSSON, T., KABATIANSKII, G., AND SMEETS, B. J. M. On families of hash functions via geometric codes and concatenation. In Stinson [39], pp. 331–342.
- [12] BLACK, J., HALEVI, S., KRAWCZYK, H., KROVETZ, T., AND ROGAWAY, P. UMAC: Fast and secure message authentication. In Wiener [43], pp. 216–233.
- [13] 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.

- [14] 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.
- [15] 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.
- [16] BRINCAT, K., AND MITCHELL, C. J. New CBC-MAC forgery attacks. In *ACISP '01: Proceedings of the 6th Australasian Conference on Information Security and Privacy (London, UK, 2001)*, Springer-Verlag, pp. 3–14.
- [17] CANTOR, D. G., AND ZASSENHAUS, H. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation* 36, 154 (1981), 587–592.
- [18] CARTER, J., AND WEGMAN, M. Universal hash functions. *Journal of Computer and System Sciences* 18 (1979), 143–154.
- [19] DAI, W., AND KROVETZ, T. VHASH security. Cryptology ePrint Archive, Report 2007/338, 2007.
- [20] DAMGÅRD, I. A design principle for hash functions. In Brassard [15], pp. 416–427.
- [21] DEN BOER, B. A simple and key-economical unconditional authentication scheme. *Journal of Computer Security* 2 (1993), 65–72.
- [22] 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.
- [23] ETZEL, M., PATEL, S., AND RAMZAN, Z. Square hash: Fast message authentication via optimized universal hash functions. In Wiener [43], pp. 234–251.
- [24] GILBERT, H., AND HANDSCHUH, H., Eds. *Fast Software Encryption: 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers (2005)*, vol. 3557 of *Lecture Notes in Computer Science*, Springer.
- [25] GONNET, G. Expected length of the longest probe sequence in hash code searching. *Journal of the ACM* 28 (1981), 289–309.
- [26] HALEVI, S., AND KRAWCZYK, H. MMH: Software message authentication in the gbit/second rates. In *Fast Software Encryption (1997)*, pp. 172–189.
- [27] HANDSCHUH, H., AND PRENEEL, B. Key-recovery attacks on universal hash function based MAC algorithms. In *CRYPTO (2008)*, D. Wagner, Ed., vol. 5157 of *Lecture Notes in Computer Science*, Springer, pp. 144–161.
- [28] KALISKI, B., AND ROBshaw, M. Message authentication with MD5. *CryptoBytes* (Spring 1995).
- [29] KOBLITZ, N., Ed. *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings (1996)*, vol. 1109 of *Lecture Notes in Computer Science*, Springer.
- [30] KRAWCZYK, H. LFSR-based hashing and authentication. In Desmedt [22], pp. 129–139.
- [31] MCGREW, D., AND VIEGA, J. The Galois/counter mode of operation (GCM). NIST Special Publication, 2005. cs-www.nsl.nist.gov/groups/ST/toolkit/BGM/documents/proposedmodes/gcm/gcm-revised-spec.pdf.
- [32] MCGREW, D., AND WEIS, B. Requirements on fast message authentication codes. IETF Internet-Draft, February 2008. Intended status: Informational. <http://www.ietf.org/internet-drafts/draft-irtf-cfrg-fast-mac-requirements-01.txt>.

- [33] 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/>.
- [34] MERKLE, R. C. One way hash functions and DES. In Brassard [15], pp. 428–446.
- [35] 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.
- [36] 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.
- [37] 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.
- [38] SHOUP, V. On fast and provably secure message authentication based on universal hashing. In Koblitz [29], pp. 313–328.
- [39] STINSON, D. R., Ed. *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings* (1994), vol. 773 of *Lecture Notes in Computer Science*, Springer.
- [40] STINSON, D. R. On the connections between universal hashing, combinatorial designs and error-correcting codes. *Electronic Colloquium on Computational Complexity (ECCC)* 2, 52 (1995).
- [41] TAYLOR, R. An integrity check value algorithm for stream ciphers. In Stinson [39], pp. 40–48.
- [42] 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.
- [43] 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

As mentioned in the introduction, all stateless MACs we investigate fail to be secure under the definitions of security given above. Furthermore, some stateful schemes with correct nonce use, and all stateful schemes with incorrect nonce use are insecure.

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 [35]. 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. In a similar vein, we analyze nonce misuse in the WCS paradigm, finding devastating consequences for most hash families.

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. In other cases, collision attacks specific to a specific MAC may be more efficient. Regardless, we are instead focusing on what happens after those collisions have occurred.

A.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 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, \dots, 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 [35], 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 $M = M_1 \parallel M_2$ and $M' = M'_1 \parallel M'_2$ 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 $M_1 \parallel M_2 \oplus v$ to receive tag t^* . Then we can submit the pair $(M'_1 \parallel M'_2 \oplus v, t^*)$ as a forgery pair. To see why, consider the following:

$$\begin{aligned} M_2 \oplus E_K(M_1) &= M'_2 \oplus E_K(M'_1) \\ \Rightarrow M_2 \oplus v \oplus E_K(M_1) &= M'_2 \oplus v \oplus E_K(M'_1) \\ \Rightarrow E_K(M_2 \oplus v \oplus E_K(M_1)) &= E_K(M'_2 \oplus v \oplus E_K(M'_1)) \end{aligned}$$

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 in 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 $M \parallel 10^l$ 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 $1 \leq i \leq m - 1$. 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 after padding, in n -bit blocks, of m and m' , respectively, then

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

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 [5] 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 $1 \leq i \leq m$. 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.

A.2 Padding Attacks

ITERATED HASH FUNCTIONS. Cryptographic hash functions are useful in many contexts. A particularly popular methodology, suggested first by Merkle [34] and later by Damgård[20], is the iterated construction. Formally, let $f : \{0, 1\}^n \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ and define the iterated hash $H : (\{0, 1\}^n)^+ \times \{0, 1\}^l \rightarrow \{0, 1\}^l$ based on f by the following: On inputs $M \in (\{0, 1\}^n)^+, IV \in \{0, 1\}^l$ such that $M = M_1 \parallel M_2 \parallel \dots \parallel M_m$, $H(M, IV) = h_m$, where $h_0 = IV$ and $h_i = f(M_i, h_{i-1})$ for $1 \leq i \leq m$.⁵

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 $f : \mathcal{K} \times (\{0, 1\}^n)^+ \rightarrow \{0, 1\}^n$, $f_{K1}(M) = h_m$ where h_m is as from the description of EMAC earlier. Then define $g : \mathcal{K} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, $g_{K2}(x) = E_{K2}(x)$ so that $\text{EMAC}_{K1, K2}(M) = g_{K2}(f_{K1}(M))$.

⁵Typically 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.

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 [28, 35]. This padding attack is effective against EMAC [5], ANSI retail MAC [1], XCBC [13], and HMAC [2].

HMAC. Let $H : (\{0, 1\}^l)^+ \times \{0, 1\}^L \rightarrow \{0, 1\}^L$ be an iterated hash function. Given a secret key K and input message M , $\text{HMAC}_K(M)$ is defined as $H(\bar{K} \oplus \text{opad} \parallel H(\bar{K} \oplus \text{ipad} \parallel M))$ where opad and ipad are 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 $\bar{K} \oplus \text{ipad}$ (see [35] for methods on ensuring this event occurs), then by the observation made above about collisions in iterated hash functions, $M \parallel s$ will collide with $M' \parallel s$ for $s \in (\{0, 1\}^l)^+$. The adversary forges by querying the MACing oracle at $M \parallel s$ to receive tag t and querying the verification oracle at $(M' \parallel s, t)$. 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 $1 \leq i \leq m-1$. 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 $M \parallel 10^l$ 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 $1 \leq i \leq m-1$. $f(M)$ 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 $v \in (\{0, 1\}^n)^+$, $f(M \parallel v) = f(M' \parallel v) \Rightarrow g(f(M \parallel v)) = g(f(M' \parallel v))$. The case where the lengths of M and M' are not multiples of n can be handled similarly.

PMAC. The MAC PMAC is described as follows: for a given blockcipher E and a given message $M = M_1 \parallel M_2 \parallel \dots \parallel M_m$ 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 [13]. The tag produced by PMAC with key K on message M of m blocks, denoted by $\text{PMAC}_K(M)$, is $E_K(\text{pad}(M_m) \oplus X_m \oplus E_K(X_1) \oplus \dots \oplus E_K(X_{m-1}))$ 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:

$$\begin{aligned} E_K(\text{pad}(M_m) \oplus X_m \oplus E_K(X_1) \oplus \dots \oplus E_K(X_{m-1})) &= \\ E_K(\text{pad}(M'_{m'}) \oplus X'_{m'} \oplus E_K(X'_1) \oplus \dots \oplus E_K(X'_{m'-1})) &= \\ \Rightarrow \text{pad}(M_m) \oplus X_m \oplus E_K(X_1) \oplus \dots \oplus E_K(X_{m-1}) &= \\ \text{pad}(M'_{m'}) \oplus X'_{m'} \oplus E_K(X'_1) \oplus \dots \oplus E_K(X'_{m'-1}) & \end{aligned}$$

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

$$\begin{aligned} E_K(\text{pad}(M_m \oplus v) \oplus X_m \oplus E_K(X_1) \oplus \dots \oplus E_K(X_{m-1})) &= \\ E_K(\text{pad}(M'_{m'} \oplus v) \oplus X'_{m'} \oplus E_K(X'_1) \oplus \dots \oplus E_K(X'_{m'-1})) &= \\ \Rightarrow \text{pad}(M_m) \oplus v \oplus X_m \oplus E_K(X_1) \oplus \dots \oplus E_K(X_{m-1}) &= \\ \text{pad}(M'_{m'}) \oplus v \oplus X'_{m'} \oplus E_K(X'_1) \oplus \dots \oplus E_K(X'_{m'-1}) &= \\ \Rightarrow \text{pad}(M'_{m'}) \oplus X'_{m'} \oplus E_K(X'_1) \oplus \dots \oplus E_K(X'_{m'-1}) & \end{aligned}$$

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.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 some natural ways to do so the answer is, surprisingly, “no.” In instances where we found no attack on the stateful schemes with correct nonce management, we turn our attention to how repeated nonces among tags affects security. For all MACs we examine, just a handful of MAC queries with repeated tags are enough to allow j -forgery attacks.

One obvious way to add state to a stateless MAC $\Pi = (\text{MAC}, \text{VF})$ is to parameterize inputs with a nonce, cnt . Let $\langle cnt \rangle_b$ 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 $\langle cnt \rangle_b \parallel M$. Just as naturally, the algorithm can be defined to return the value $\text{MAC}_K(M \parallel \langle cnt \rangle_b)$. In either case we will assume the nonce is encoded using n bits — a full block. Our attacks can easily be adapted to shorter encodings of the state.

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 nonces instead of the last blocks of M, M' . Thus, let v be a value such that the nonce 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 nonce is queried at most once and all but two blocks may be freely chosen by the adversary. That is, perhaps surprisingly, adding state in this way to CBC MAC does not add any security.

Now suppose an encoding of the state is prepended to each message in the setting of CBC MAC. With proper nonce management (ie, no repeated nonces) we found no attack which effectively used information from a pair of colliding messages. However if we allow $\alpha = 2$ queries per nonce, a simple j -forgery attack immediately follows where we need only j MAC queries using prior nonces: An adversary queries messages of the form $R_i \parallel M$ 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 $(R_i \parallel M, i)$ collides with $(R_k \parallel M, k)$. 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 $(R_i \oplus v \parallel M', i)$ to receive tag t and forges with $(R_k \oplus v \parallel M', k, t)$. The justification of this claim is almost identical to the justification for the attack on the stateless CBC MAC and is omitted. This attack first appeared in [16].

This method of adding state does much 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 nonces for each forgery she 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. Ideally, the adversary must work equally hard for each forgery, but we will see later that this is possible with WMAC. 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.

HMAC AND PMAC. For both HMAC and PMAC, simply prepending or appending state prevents attacks, but we will cover the cases of nonce misuse, where for both schemes only j out-of-protocol MAC queries are necessary to obtain j forgeries.

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

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

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 nonce 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 $\sqrt{j}2^{n+1}$.

A.4 Attacks on Carter-Wegman MACs

There are two MAC paradigms of the Carter-Wegman style [18] described in this paper: the stateless mode FH, proposed in Carter and Wegman’s original paper [18], and a stateful mode which we refer to as WCS for Wegman-Carter-Shoup, reflecting the original idea by Carter and Wegman and whose formal security bounds have been more recently improved by Shoup [38] and Bernstein [10].

The attack on each scheme is dependent on the family of universal hash functions used. We will show that for each of the families hash127/Poly1305 [7, 9], Square-Hash[23], LFSR-based Topelitz Hash[30], Bucket Hash[36], MMH[26], NMH[12], and VHASH[19] there exists an adversary A such that A can forge j messages in the FH[\mathcal{H}, \mathcal{R}] paradigm in resources comparable to those required for a single forgery. Informally, the bounds given in [10] show that the first forgery in the WCS mode will likely occur well after one starts to see collisions in tags, so we instead concentrate on potential problems with nonce misuse. For some hash families, nonce misuse can be devastating: if any nonce is repeated, even once, to the MAC oracle an adversary can learn the hash key. To be clear — our results do not contradict any bounds given in [10], and only reinforce the necessity of proper nonce management in WCS MACs.

Following the posting of this paper on eprint.iacr.org, Handschuh and Preneel [27] improved and expanded many of the attacks here, among other results.

FH. The FH paradigm is parameterized by a pseudorandom function family \mathcal{R} and an ϵ -AU hash family \mathcal{H} , written as FH[\mathcal{H}, \mathcal{R}]. The shared key between signer and receiver is (h, ρ) , where $h \xleftarrow{\$} \mathcal{H} = \{h : \mathcal{D} \rightarrow \{0,1\}^l\}$ and $\rho \xleftarrow{\$} \mathcal{R} = \text{Rand}(l, L)$. 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 .

ATTACKS ON FH. 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 . This idea of exploiting ‘internal’ collisions in MACs is not new [35].

HASH127/POLY1305. Let $M = (M_0, M_1, \dots, M_{m-1})$ be a sequence of integers in $[-2^r, 2^r - 1]$ for some r . For any integer x define $h_x(M) = (x^{m+1} + M_0x^m + M_1x^{m-1} + \dots + M_{m-1}x) \bmod (p)$ for some prime $p > 2^r$. When x is thought of as the hash instance or key, this is the well-known polynomial hash, known to be m/p -AU for some time [11, 21, 41]. More recently, Bernstein has described two efficiently computable polynomial hashes, hash127 where $p = 2^{127} - 1$ and Poly1305 where $p = 2^{130} - 5$, in [7, 9].

Claim 7 *Let $M = (M_0, M_1, \dots, M_{m-1})$, $M' = (M'_0, M'_1, \dots, M'_{m-1})$ be two distinct messages such that $h_x(M) = h_x(M')$. Then for an arbitrary non-zero constant $v \in [-2^r, 2^r - 1]$ such that $M_i + v < 2^r - 1$, $M'_i + v < 2^r - 1$, the messages $F = (M_0, \dots, M_{i-1}, M_i + v, M_{i+1}, \dots, M_{m-1})$ and $F' = (M'_0, \dots, M'_{i-1}, M'_i + v, M'_{i+1}, \dots, M'_{m-1})$ will also collide under h_x .*

Proof:

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

■

One can do better than finding more collisions, however. Let $g(x)$ be the monic polynomial of degree m over \mathbb{F}_p , 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 p) 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 [6] 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 $1 - 1/p$. This probability can be brought arbitrarily close to 1 with more queries. The algorithm for doing this

SQUARE-HASH. We describe the universal hash family Square-Hash, first given in [23] 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 8 *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 \mathbb{F}_p .*

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

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

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

■

To allow messages of greater lengths, Square-Hash was extended to a family SQH* by using a sum.⁶ Let $M = M_1 \parallel M_2 \parallel \dots \parallel M_m$ where $|M_i| = n$ and let x be an m -vector with coordinates x_1, x_2, \dots, x_m in the integers. Then $\text{SQH}_x^*(M)$ 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 $M = M_1 \parallel M_2 \parallel \dots \parallel M_m$ to receive tag t . Note that $(M_i + x)^2 \equiv a \bmod p$ is a quadratic residue mod 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 $\text{MAC}(M) = \text{MAC}(M')$.

WEGMAN-CARTER-SHOUP MACS. Let \mathcal{H} be some ϵ -AU hash family $\mathcal{H} = \{h : \mathcal{D} \rightarrow \{0, 1\}^L\}$, and \mathcal{R} a set of functions $\mathcal{R} = \text{Rand}(b, L)$.⁷ The Wegman-Carter-Shoup scheme parameterized by these families is denoted as $\text{WCS}[\mathcal{H}, \mathcal{R}]$. Let $\rho \xleftarrow{\$} \mathcal{R}$ and $h \xleftarrow{\$} \mathcal{H}$. Then (ρ, h) is the shared key between signer and verifier. The signer has a nonce, cnt , which is an integer variable. To MAC message M , the signer first ensures that $\text{cnt} < 2^b - 1$ and if so sends $(\text{cnt}, \rho(\langle \text{cnt} \rangle_b) \oplus h(M))$ where \oplus denotes the operation over some group (for VMAC and Poly1305-AES it is simple addition over the the numbers modulo 2^L). To verify a message M with tag (i, t) , the verifier computes $\rho(\langle i \rangle_b) \oplus h(M)$ and ensures it equals t .

ATTACKS ON WCS. The attacks on hash127/Poly1305 and Square Hash in WCS mode use the same idea to recover the key. Two distinct messages M, M' , of the same length, are queried using the same nonce i , yielding two tags t and t' , respectively. (Note that only *one* errant query is required for this attack.) The value $t' - t$ gives the difference of outputs from the UHF on inputs M' and M . For hash127, Poly1305, and Square Hash this gives a polynomial equation modulo some prime p , evaluated at the hash key. It is a simple process to then use the techniques described in the attack on hash127/Poly1305 in the FH setting to factor the polynomial over the finite field, and test possible values of the hash key via the verification oracle.

This attack demonstrates that proper nonce management is an *extremely* important part of the security of WCS MACs. Even an innocuous-looking “off by one” implementation error can enable an attacker to forge an arbitrary number of messages, with complete message freedom. This susceptibility to insecurity when perhaps subtle programming mistakes are made led us to construct a more fault-tolerant stateful MAC.

OTHER HASH FAMILIES For each remaining universal hash family, we first describe an attack using collisions in tags found in FH mode, then cover an attack in WCS mode with nonce misuse. We stress again that many of these attacks have been subsequently improved in [27].

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 [30], with changes designed to speed up hardware implementations. The changes are not relevant to the attacks 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 $\mathbf{A}M$.

⁶The 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.

⁷The security bounds given in [10, 38] do not require that \mathcal{R} be a family of random functions. \mathcal{R} may also be a family of random permutations.

For the FH scenario, 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 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: $\mathbf{A}F - \mathbf{A}F' = \mathbf{A}(F - F') = \mathbf{A}(F - (F - M + M')) = \mathbf{A}(M - M') = 0$ \blacksquare

The attack in the WCS mode of operation is almost identical. Query two distinct messages M, M' with the same nonce. The difference of their respective tags t^* is equal to the following equation:

$$\mathbf{A}(M - M')$$

The attacker then constructs two message F, F' , using a similar process as described above, such that $h(F) - h(F') = t^*$. A forgery attack follows immediately by querying with (F, j) to receive tag t and forging with $(F', t - t^*)$. Again only *one* MAC query with a repeated nonce is needed.

BUCKET HASH. First described by Rogaway in 1995 [36], 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 $M = M_1 \parallel M_2 \parallel \dots \parallel M_n$ 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 i th 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 ← 1 to N do Y_i ← O^w
for i ← 1 to N do
  Y_{x_{i1}} ← Y_{x_{i1}} ⊕ M_i
  Y_{x_{i2}} ← Y_{x_{i2}} ⊕ M_i
  Y_{x_{i3}} ← Y_{x_{i3}} ⊕ M_i
return Y_1 || Y_2 || ... || Y_n

```

For the attack in the FH setting, assume that a collision has occurred so that we know M, M' such that $\text{bucket_hash}(M) = \text{bucket_hash}(M')$. 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 $\text{bucket_hash}(F) = \text{bucket_hash}(F')$.

The proof is left as an exercise to the interested reader.

For the attack in the WCS setting we again need only one errant MAC query. By the same technique used earlier, query distinct messages M, M' with the same nonce to obtain $\text{bucket_hash}(M) - \text{bucket_hash}(M') = t^*$. Create two messages F, F' by the same method used in the FH setting. Query on (F, j) to get tag t and forge with $(F, j, t - t^*)$.

MMH. The MMH family [26] 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 i th coordinate of x and M_i the i th coordinate of M . Through some clever implementation tricks, this family is very efficient in software. For the attack in the FH setting, consider message M and M' such that $h_x(M) = h_x(M')$. Choose arbitrary non-zero $v \in \{0, 1\}^{32}$ and $i_0 \in [1 \dots n]$. 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:

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

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

Misuse of nonces in the WCS allows complete recovery of the key material, with only n MAC queries with repeated nonces. Namely, for each x_i , query $M' = 0^{32n}$ and M such that $M_j = 0^{32}$ for $j \neq i$ and $M_i = 1$. The difference of the tags produced on MAC queries M and M' is exactly x_i . After all n indices have been queried, the complete key is known.

NMH. Also mentioned in the MMH paper [26] is the adaption of the authors' methods to a family created by Mark Wegman. NMH is defined as $\mathcal{H} = \{h : (\{0, 1\}^{32})^n \rightarrow \{0, 1\}^{32}\}$ 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 i th coordinate of x and M_i the i th coordinate of M .

For FH, consider the case where there are two distinct message M, M' such that $h_x(M) = h_x(M')$. Pick distinct $i_0, i_1 \in [1 \dots n]$. 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 $v_1 = -a^2b$. 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 $b \in 0, 1$. 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 $b \in 0, 1$.

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

Proof:

$$\begin{aligned} h_x(F) &= \left[\left[\left[v_0(M_{i_0-1} + x_{i_0-1}) + v_1(M_{i_1-1} + x_{i_1-1}) + \right. \right. \right. \\ &\quad \left. \left. \left. \sum_{i=1}^{n/2} (M_{2i-1} + x_{2i-1} \bmod 2^{32})(M_{2i} + x_{2i} \bmod 2^{32}) \right] \bmod 2^{64} \right] \bmod (2^{32} + 15) \right] \bmod 2^{32} \end{aligned}$$

But note that

$$\begin{aligned} h_x(F') &= \left[\left[\left[v_0(M'_{i_0-1} + x_{i_0-1}) + v_1(M'_{i_1-1} + x_{i_1-1}) + \right. \right. \right. \\ &\quad \left. \left. \left. \sum_{i=1}^{n/2} (M'_{2i-1} + x_{2i-1} \bmod 2^{32})(M'_{2i} + x_{2i} \bmod 2^{32}) \right] \bmod 2^{64} \right] \bmod (2^{32} + 15) \right] \bmod 2^{32} \end{aligned}$$

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 $v_0a = -v_1(b)$. By the way v_0 and v_1 were defined, $v_0a = a^2b^2 = -v_1b$. \blacksquare

A key recovery attack is possible in the WCS setting, requiring n MAC queries with repeated nonces. The attack is almost identical to the key recovery attack on MMH, and is omitted.

The family NH used in UMAC [12] is very similar to NMH — essentially the differences amount to the constants chosen over which to do modular arithmetic. As such, the above attacks can be easily adopted to NH.

VHASH. The VHASH family is used in VMAC, a successor to UMAC. Because VHASH is the composition of three different hash families, we were not able to find an attack when nonces were misused. We conjecture that there is a simple attack which uses only a small number of queries, but it has so far eluded us. However, if one is allowed to query up to the birthday bound with the same nonce, then tag collisions will occur and we may use the above techniques to detect those collisions which are result of the innermost hash function, based on NH, and apply the attack above.

B Details of the hash127 Attack

Let us briefly recall the scenario described in Section A.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/Poly1305. The adversary has constructed a polynomial $g(x)$ over \mathbb{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 r bits), and these can be found efficiently using Berlekamp's algorithm [6] or the Cantor/Zassenhaus algorithm [17]. Let x_1, x_2, \dots, 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

$X \leftarrow \{x_i : 1 \leq i \leq k\}$

while $|X| > 1$ **do**:

- $Z_1 \leftarrow \{x_i : 1 \leq i \leq \lfloor |X| \rfloor\}$
- $Z_2 \leftarrow \{x_i : 1 \leq i \leq \lceil |X| \rceil\}$
- Let $R \leftarrow \{r_i : 1 \leq i \leq m - |Z_1|\}$ 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 $\sim 1/p + 1/n$, 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 A Bound for C

We seek an answer to the following (maximum occupancy) question: Given a randomly selected q -tuple $S \leftarrow \{\{0, 1\}^n\}^q$, how many times does the maximally-occurring value in the tuple occur? It was shown by Gonnet [25] that if $q2^{-n} = \beta$ for some fixed β , then this number is $\sim \ln q / \ln \ln q$ as $q, 2^n \rightarrow \infty$. In general, however, we do not necessarily expect that $q2^{-n}$ grows as a constant since we are interested in tag truncation.

We can use the normal approximation to give an estimate to the following related question: On average how many values in $\{0, 1\}^n$ occur in S exactly k times? The answer is given as

$$2^n \frac{e^{-x^2}}{\sqrt{2\pi}} + O(1/q)$$

where

$$k = \frac{q}{2^n} + x\sqrt{\frac{q}{2^n}} \text{ and } x = O(1)$$

Letting $x = 15$ will ensure this quantity is $\leq 2^{-64}$ for all cases of practical interest ($n \leq 256$). Thus, $C \leq \max\{1, \frac{q}{2^n} + 15\sqrt{\frac{q}{2^n}}\}$ and in particular when $\frac{q}{2^n} = 2^t$ then $C \leq \max\{1, 2^t + 15 \cdot 2^{t/2}\}$.