MAC Reforgeability

John Black[1] and Martin Cochran[2]

[1] Dept. of Computer Science, University of Colorado, Boulder CO 80309, USA
jrblack@cs.colorado.edu, www.cs.colorado.edu/~jrblack
[2] Google, Inc. 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
martin.cochran@gmail.com

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 reforgeabiliy, 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.

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.

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

Fig. 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.

Main Results. In this paper we conduct a systematic study of reforgeability, treated first in the literature by McGrew and Fluhrer [23]. 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) [7, 27] MACs, the attack is more severe: nonce misuse yields the universal hash family instance almost immediately.

After an earlier draft of this paper appeared on eprint, many of the attacks in Figure 1 and Figure 2 were subsequently improved in [18] by Handschuh and Preneel. In light of this, we include no attacks within this version of the paper. For attack details we refer the interested reader to the full version of this paper [8] and the other literature on this subject [10, 18, 23, 24].

UHF in FH mode	Expected queries for j forgeries	Reveals key	Queries for key recovery
hash127/Poly1305	$C_1 + \log m + j$	$\surd$	$C_1 + \log m$
VMAC	$C_1 + 2j$		
Square Hash	$C_1 + 2j$	$\surd$	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	$\surd$	$2 + \log m$
VMAC	$C_1 + 2j$	$C_1 + j$		
Square Hash	$3m + j$	m	$\surd$	$3m$
Topelitz Hash	$2j + 2$	1		
Bucket Hash	$2j + 2$	1		
MMH/NMH	$2m + j$	m	$\surd$	$2m$

Fig. 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 [11, 29] 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) [7, 27] 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 [18] for further attacks on these and other hash families in a similar setting.

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 trucation 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 $\epsilon-$AU hash family $\mathcal{H} = \{h : D \to \{0,1\}^l\}$, and $\mathcal{R}$ a set of functions $\mathcal{R} = \mathrm{Rand}(l+b, n)$. 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 $\rho(\langle i\rangle_b \| h(M))$ and ensures it equals Tag.

WHY WMAC? There are essentially four parameters which much 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 [11, 29] 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 $\epsilon \le 2^{-82}$, $b = 8$, 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.[3] 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.)

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 [18] still apply and indeed come within a constant factor of matching the bound given in our security reduction.[4]

[3] Dan Bernstein has recently proposed [5] 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.

[4] Our 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 [18].

Related Work. David McGrew and Scott Fluhrer have also done some work [23] 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 [21], reforgeability is possible. However, they examine reforgeability in terms of the number of expected forgeries (parameterized by the number of queries) for each scheme, which is dependent on the precise security bounds for the respective MACs. 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 [18]. 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.

Outline of the Paper. In the next section we cover the basic notation and security models used. After that, we jump right in to the discussion of WMAC and its security reduction, our main contribution, deferring the attacks that motivated its construction to the full version [8].

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

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

Definition 1. *(Carter and Wegman [11]) Fix a domain $\mathcal{D}$ and range $\mathcal{R}$. A finite multiset of hash functions $\mathcal{H} = \{h : \mathcal{D} \to \mathcal{R}\}$ is said to be* **Universal** *if for every $x, y \in \mathcal{D}$ with $x \neq y$, $Pr_{h\in\mathcal{H}}[h(x) = h(y)] = 1/|\mathcal{R}|$.*

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

Definition 3. *(Krawczyk [20], Stinson [28]) Let $\epsilon \in \mathbb{R}^+$ and fix a domain $\mathcal{D}$ and range $\mathcal{R} \subseteq \{0,1\}^r$ for some $r \in \mathbb{Z}^+$. A finite multiset of hash functions $\mathcal{H} = \{h : \mathcal{D} \to \mathcal{R}\}$ is said to be ϵ-***Almost XOR Universal** *(ϵ-AXU) if for every $x, y \in \mathcal{D}$ and $z \in \mathcal{R}$ with $x \neq y$, $Pr_{h\in\mathcal{H}}[h(x) \oplus h(y) = z] \leq \epsilon$.*

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

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

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

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

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

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

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

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

where the maximum is over all adversaries A that have time-complexity at most 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, $\mathbf{Adv}_{\Pi}^{juf\text{-}cma}(t, q_s, \mu_s, q_v, \mu_v, j) \leq \epsilon$, 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, $(\mathrm{MAC}, \mathrm{VF})$, where MAC is an algorithm that, upon input of key $K \in \mathcal{K}$ for some key space $\mathcal{K}$, a message $M \in \mathcal{D}$ for some domain $\mathcal{D}$, and a state value S from some prescribed set of states $\mathcal{S}$, computes a τ-bit tag Tag; we denote this by $\mathrm{Tag} = \mathrm{MAC}_K(M, S)$. Algorithm VF is the verification algorithm such that on inputs $K \in \mathcal{K}$, $M \in \mathcal{D}$, $\mathrm{Tag} \in \{0, 1\}^\tau$, and $S \in \mathcal{S}$, VF outputs a bit, with 1 representing accept and 0 representing reject. This computation is denoted $\mathrm{VF}_K(M, S, \mathrm{Tag})$. A restriction on VF is that if $\mathrm{MAC}_K(M, S) = \mathrm{Tag}$, then $\mathrm{VF}_K(M, S, \mathrm{Tag})$ must output 1.

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 reforgery attacks.

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

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

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

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

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

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, $\mathbf{Adv}_{\Pi}^{jsuf\text{-}cma}(t, q_s, \mu_s, q_v, \mu_v, j, \alpha) \leq \epsilon$*, 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 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.[5]

[5] Truncating 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.

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

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

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 $\mathbf{Adv}^{jsuf\text{-}cma}_{\Pi}(q, t, \alpha)$ be the maximum of $\mathbf{Adv}^{jsuf\text{-}cma}_{\Pi}(A)$ 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 $\mathbf{Adv}^{jsuf\text{-}cma}_{\Pi}(q, t, \alpha)$ is negligibly small for any reasonably large q and Time.

As an example, the FH and FCH [11, 29] modes of operation are special cases of WMAC where α is set to q_s and 1, respectively.

Theorem 1. *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*

$$\mathbf{Adv}^{jsuf\text{-}cma}_{\mathrm{WMAC}}(A) \le \mathbf{Adv}^{prf}_{F}(B) + \frac{\epsilon(\alpha-1)q_s}{2} + \frac{\epsilon}{2^{n-1}}\left(q_v^2 + q_v q_s + \max\{2^n, q^{\frac{1}{2}} 2^{\frac{n}{2}+3}\} q_v\right) + \delta(j, n, q_v).$$

and where B makes at most q queries, using time proportional to $\mathsf{Time} + \mathrm{Hash}(q)$*, where* $\mathrm{Hash}(1)$ *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*

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

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)[21], 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 1. For a given MAC scheme $\Pi = (\mathrm{MAC}, \mathrm{VF})$, let $E(\mathrm{Forge}_{\Pi}, q_s, q_v)$ 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 [23], 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,

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

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 [23].

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.[6] 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 $(t, h(M))$

[6] This 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.

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.

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 $\perp$.

```
**Procedure Initialize**
  $V \leftarrow \emptyset$, $h \xleftarrow{\$} \mathcal{H}$, $\rho \xleftarrow{\$} \text{Rand}(\mathcal{T} \times \mathcal{R}, \{0,1\}^n)$,
  InitializeMap(Map, $\mathcal{T} \times \mathcal{D}, \mathcal{T} \times \mathcal{R}$), InitializeMap(Map$_o$, $\mathcal{T} \times \mathcal{D}, \mathcal{T} \times \mathcal{R}$)
**Procedure** MAC$(t, x)$
  $v \leftarrow h(x)$
  If $(t, v) \in V$ then { $bad \leftarrow$ true, $\boxed{(t, v) \xleftarrow{\$} \mathcal{T} \times \mathcal{R} \setminus V}$ }
  $V \leftarrow V \cup (t, v)$
  return $\rho(t, v)$
**Procedure** VF$(t, x, \text{Tag})$
  If Map$[(t, x)] = \perp$ then {
    $v \leftarrow h(x)$, Map$[(t, x)] \leftarrow (t, v)$
    $\boxed{\text{If } (t, v) \in V \text{ then } \{ \text{Map}_o[(t, x)] \leftarrow (t, v),\ \text{Map}[(t, x)] \xleftarrow{\$} \mathcal{T} \times \mathcal{R} \setminus V,\ (t, v) \leftarrow \text{Map}[(t, x)] \}}$
    $V \leftarrow V \cup (t, v)$
  }
  If Map$_o[(t, x)] \neq \perp$ then {
    If Tag $= \rho(\text{Map}_o[(t, x)])$ or Tag $= \rho(\text{Map}[(t, x)])$ then { $bad \leftarrow$ true }
  }
  return Tag $= \rho(\text{Map}[(t, x)])$
```

Fig. 3. Game G_0 and $\boxed{\text{Game } G_1}$

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 choosen 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,

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

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

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

(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 $q_v = \sum_{x \in S} q_{v,x}$.)

```
   Procedure Initialize
0    $V \leftarrow \emptyset$, $h \xleftarrow{\$} \mathcal{H}$, $\rho \xleftarrow{\$} \text{Rand}(\mathcal{T} \times \mathcal{R}, \{0,1\}^n)$, InitializeMap(Map, $\mathcal{T} \times \mathcal{D}, \mathcal{T} \times \mathcal{R}$),
     InitializeMap($\text{Map}_o, \mathcal{T} \times \mathcal{D}, \mathcal{T} \times \mathcal{R}$), 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 { $bad \leftarrow \mathsf{true}$, $(t, v) \xleftarrow{\$} \mathcal{T} \times \mathcal{R} \setminus V$ }
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 $\text{Map}[(t, x)] = \bot$ 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 $\text{Map}_o[(t, x)] \neq \bot$ then {
10     If $\text{Tag} = O[\text{Map}_o[(t, x)]]$ or $\text{Tag} = O[\text{Map}[(t, x)]]$ then { $bad \leftarrow \mathsf{true}$ }
     }
11   return $\text{Tag} = O[\text{Map}[(t, x)]]$
```

Fig. 4. Game G_2

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, $\text{MAC}(t, x)$ returns a uniform random value τ from $\{0,1\}^n$ and $\text{VF}(t, x, \text{Tag})$ chooses its outputs 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 $bad \leftarrow \mathsf{true}$.

In Game G_4, we first generate all the random answers to the queries of A, and on ith signing query, save the query and just return the ith random answer.

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

Fig. 5. Game G_3

```
   Procedure Initialize
0    $h \xleftarrow{\$} \mathcal{H}$, $(\tau_1, \ldots, \tau_{q_s + \#q_v}) \xleftarrow{\$} (\{0,1\}^n)^{q_s + \#q_v}$, $i \leftarrow 0$,
     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 $\tau_i$
   Procedure VF$(t, x, \mathrm{Tag})$
3    If $O[(t, x)] = \perp$ then {
4      $i \leftarrow i + 1$, $t_i \leftarrow t$, $x_i \leftarrow x$, $O[(t, x)] \leftarrow \tau_i$, $\mathrm{Tag}_i \leftarrow \mathrm{Tag}$
     }
5    return $\tau_i = \mathrm{Tag}_i$
   Procedure Finalize
6    If $(t_i, h(x_i)) = (t_j, h(x_j))$ for some $i < j \le q_s$, then { $bad \leftarrow \mathsf{true}$ }
7    If $(t_i, h(x_i)) = (t_j, h(x_j))$ for some $i < j$, $q_s < j$ then {
8      If $O[(t_i, x_i)] = \mathrm{Tag}_j$ or $O[(t_j, x_j)] = \mathrm{Tag}_j$ then { $bad \leftarrow \mathsf{true}$ }
     }
```

Fig. 6. Game G_4

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,

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

We will use the fact that

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

It is easy to analyze the probability $\Pr[A^{G_4}$ sets *bad* 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 the elements of $\mathcal{T}$ as $T_1, \ldots,$

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

$$\Pr[A^{G_4} \text{ sets } \mathit{bad} \text{ in line } \mathsf{6}] \le \sum_{i=1}^{|\mathcal{T}|} \epsilon \cdot \frac{q_{s,i}(q_{s,i}-1)}{2} \le \sum_{i=1}^{|\mathcal{T}|} \epsilon \cdot \frac{q_{s,i}(\alpha-1)}{2}$$

$$= \frac{\epsilon(\alpha-1)}{2} \sum_{i=1}^{|\mathcal{T}|} q_{s,i} = \frac{\epsilon(\alpha-1)q_s}{2}.$$

We must also bound the probability $\Pr[A^{G_4} \text{ sets } \mathit{bad} \text{ in line } \mathsf{8}]$. 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 $1 \le i \le q_s$ and $t_i = k$. Similarly, let V_k be the set of indices i such that $q_s < i \le q_s + q_v$ and $t_i = k$. Let g be the number of correctly guessed tags during the verification phase. Let $X_k = \{x_i : i \in S_k \vee (i \in V_k \wedge \mathrm{Tag}_i = \tau_i)\}$ 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 $\mathsf{8}$. Then,

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

$$+ \Pr\left[h(x_i) = h(x) : x \in X_k\}\right] \cdot \Pr[\mathrm{Tag}_i = \tau_i] \tag{2}$$

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

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

$$\le \epsilon \sum_{k\in\mathcal{T}} \sum_{j=0}^{|V_k|-1} (C_k + (\alpha+g)2^{-n} + j2^{-n+1}) \tag{5}$$

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

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

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

On a verification query $(t_j, x_j, \mathrm{Tag}_j)$ we consider two cases where the conditional on line $\mathsf{7}$ is met: $i \le q_s$ and $q_s < i$. Also, on line $\mathsf{8}$, 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 occuring 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.

The full version 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

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

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, mutally 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 [18]. 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 [19], pp. 1–15.
3. Bellare, M., Kilian, J., and Rogaway, P. The security of cipher block chaining. In Desmedt [14], pp. 341–358.
4. Bernstein, D. Floating-point arithmetic and message authentication. Draft available as `http://cr.yp.to/papers/hash127.dvi`.
5. Bernstein, D. Polynomial evaluation and message authentication. Draft available as `http://cr.yp.to/papers.html#pema`.
6. Bernstein, D. J. The Poly1305-AES message-authentication code. In Gilbert and Handschuh [16], pp. 32–49.
7. 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.
8. Black, J., and Cochran, M. MAC reforgeability. Cryptology ePrint Archive, Report 2006/095, 2006.
9. Black, J., Halevi, S., Krawczyk, H., Krovetz, T., and Rogaway, P. UMAC: Fast and secure message authentication. In Wiener [30], pp. 216–233.
10. 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.
11. Carter, J., and Wegman, M. Universal hash functions. *Journal of Computer and System Sciences 18* (1979), 143–154.
12. Dai, W., and Krovetz, T. VHASH security. Cryptology ePrint Archive, Report 2007/338, 2007.
13. den Boer, B. A simple and key-economical unconditional authentication scheme. *Journal of Computer Security 2* (1993), 65–72.
14. 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.

15. Etzel, M., Patel, S., and Ramzan, Z. Square hash: Fast message authentication via optimized universal hash functions. In Wiener [30], pp. 234–251.
16. 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.
17. Halevi, S., and Krawczyk, H. MMH: Software message authentication in the gbit/second rates. In *Fast Software Encryption* (1997), pp. 172–189.
18. 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.
19. 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.
20. Krawczyk, H. LFSR-based hashing and authentication. In Desmedt [14], pp. 129–139.
21. McGrew, D., and Viega, J. The Galois/counter mode of operation (GCM). NIST Special Publication, 2005. `cs-www.ncsl.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-revised-spec.pdf`.
22. 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`.
23. McGrew, D. A., and Fluhrer, S. R. Multiple forgery attacks against message authentication codes. Cryptology ePrint Archive, Report 2005/161, 2005. `http://eprint.iacr.org/`.
24. 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.
25. 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.
26. 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.
27. Shoup, V. On fast and provably secure message authentication based on universal hashing. In Koblitz [19], pp. 313–328.
28. Stinson, D. R. On the connections between universal hashing, combinatorial designs and error-correcting codes. *Electronic Colloquium on Computational Complexity (ECCC) 2*, 52 (1995).
29. 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.
30. 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.