ANOTHER LOOK AT HMAC

NEAL KOBLITZ AND ALFRED MENEZES

Abstract. HMAC is the most widely-deployed cryptographic-hash-function-based message authentication code. First, we describe a security issue that arises because of inconsistencies in the standards and the published literature regarding keylength. We prove a separation result between two versions of HMAC, which we denote HMAC$^{\mathrm{std}}$ and HMAC$^{\mathrm{Bel}}$, the former being the real-world version standardized by Bellare *et al.* in 1997 and the latter being the version described in Bellare's proof of security in his Crypto 2006 paper. Second, we describe how HMAC$^{\mathrm{NIST}}$ (the FIPS version standardized by NIST), while provably secure, succumbs to a practical attack in the multi-user setting. Third, we describe a fundamental defect from a practice-oriented standpoint in Bellare's 2006 security result for HMAC, and show that with this defect removed his proof gives a security guarantee that is of little value in practice. We give a new proof of NMAC security that gives a stronger result for NMAC and HMAC – and solves an "interesting open problem" from Bellare's Crypto 2006 paper – and discuss its limitations.

1. Introduction

In the first two sections our aim is to convey the general ideas using informal language, a minimum of notation and terminology, and no abbreviations or acronyms.[1] Details and formal statements can be found in later sections.

Suppose that Alice and Bob have a shared secret key K for use during a session in which they are exchanging messages M. A message authentication code is a function $t = H(K, M)$, where t is called the "tag" of the message M (under the key K); Alice sends t along with the message M in order to provide Bob with some assurance that the message he receives was truly sent by Alice and was not altered before he got it.

By a *compression function* we mean a function $z = f(x, y)$, where $y \in \{0,1\}^b$ and $x, z \in \{0,1\}^c$. We suppose that $b \geq c$, so the compression function reduces size by at least a factor of 2; typically $b = 512$, $c = 160$. Given a compression function f, the most common way to create an *iterated hash function* h is as follows [13, 23]. Let IV (called the *initializing vector*) be a publicly known bitstring of length c that is fixed once and for all. Suppose that $M = (M_1, \ldots, M_m)$ is a message consisting of $m \leq n$ b-bit blocks (where nb is some bound on message length; for simplicity in this paper we shall suppose that all message lengths are multiples of b). Then we set $x_0 = \mathrm{IV}$, and

Date: February 19, 2012; updated on March 9, 2012.

[1] Except for the one in the title of the paper, which stands for "hash-based message authentication code."

for $i = 1, \ldots, m$ we recursively set $x_i = f(x_{i-1}, M_i)$; finally, we set $h_{\mathrm{IV}}(M) = x_m$. The function h_{IV} takes a message that can be very long and outputs its c-bit hash value.[2]

One of the earliest ideas for converting an iterated hash function h_{IV} into a message authentication code $H(K, M)$ was simply to prepend the secret key K for the session. In other words, one can define $H(K, M) = h_{\mathrm{IV}}(K \| M)$. (If the length k of the key is less than b, as it usually is, then the key can be padded with zero bits or with a fixed string of $b - k$ bits.)

However, it was soon realized that this construction has a security flaw: without knowing K, anyone who knows the tag of a message $M = (M_1, \ldots, M_m)$ can easily compute the tag of any longer message whose first m blocks coincide with M (see Example 9.64 in [22]).

A message authentication code $H(K, M)$ that does not have this flaw can be defined by setting $H(K_1, K_2, M) = h_{K_2}(h_{K_1}(M)^0)$, where K_1 and K_2 are c-bit keys that serve as two different IV's for the hash function h. (Because $h_{K_1}(M)$ is shorter than a message block, it is padded by a sequence of 0-bits or some other fixed $(b - c)$-bit sequence; this is denoted by the superscript 0.) This construction – which is called a "Nested Message Authentication Code" [3] – has no obvious security flaws.

Nevertheless, computer engineers were dissatisfied for two reasons. In the first place, the construction needed two keys rather than one – or, equivalently, a single key of $2c$ bits – and they didn't like having to double the bitlength of the key. More importantly, the construction required that the IV be changed. The engineers wanted to use an off-the-shelf iterated hash function that already had a fixed IV built into it. The message authentication code HMAC was developed in response to these two objections.

First, to deal with the objection to two keys, a single k-bit key K is used, and two keys K_1 and K_2 are obtained from it by XORing with two fixed bitstrings P_1 and P_2: $K_1 = K \oplus P_1$, $K_2 = K \oplus P_2$. Next, in order to use a fixed hash function with given initializing vector IV, the definition of $H(K, M)$ was changed to the following: $H(K, M) = h_{\mathrm{IV}}(K_2^0 \| h_{\mathrm{IV}}(K_1^0 \| M)^0)$, where the zero superscript means that zero bits (or any fixed and publicly known sequences of bits) are appended to fill out the block of b bits.

In [3] Bellare, Canetti, and Krawczyk gave a security proof for the Nested Message Authentication Code, which they then extended to an argument for the security of HMAC. They made two assumptions about the hash function. First, they assumed that the underlying compression function $z = f(x, y)$ is itself a secure message authentication code. This means that an adversary, given access to an oracle that chooses a random key $K \in \{0, 1\}^c$ and responds to any query $M \in \{0, 1\}^b$ with the value $f(K, M)$, cannot in a reasonable length of time with non-negligible probability produce a forgery $f(K, M^*)$ of some message M^* that it didn't query.

The second condition was that the hash function h_{IV} is *collision-resistant*, that is, one cannot in a reasonable length of time find two different messages with the same hash value. But in subsequent years the most commonly-used hash functions were shown not

[2] Before applying an iterated hash function, generally one appends a message block that gives the block-length of the message; with this modification it is possible to give a simple proof that collision-resistance of f implies collision-resistance of h.

to have this property [30, 31].[3] This did not mean that any attack on HMAC had been found, but only that the proof-based guarantee did not apply to HMAC with the hash functions used in practice. The first objective of Bellare's paper [2] was to restore the proof-based guarantee by giving a new proof based on assumptions that had not been invalidated by the work in [30, 31] for the hash functions that are currently in use. A second purpose of [2] was to remove the two-key gap, that is, give a formal proof in the case when the keys K_1 and K_2 are not independent, but rather K_2 is the XOR-shift of K_1 by a fixed and known bitstring.

The first assumption in [2] is that the underlying compression function $z = f(x, y)$ is a pseudorandom function in the following sense. Suppose you choose a random $x \in \{0, 1\}^c$ and flip a coin. Your adversary is allowed to query values of $y \in \{0, 1\}^b$; if the coin was heads, you must always answer the query with the correct value of $f(x, y)$, whereas if the coin was tails, you always reply to the query with a random value of $z \in \{0, 1\}^c$ (subject only to the condition that if the same query is repeated, the same random value of z must be given). Then $f(x, y)$ is said to be a *pseudorandom function* if the adversary, based on your answers to her queries, is unable in a reasonable length of time to determine whether the coin was heads or tails with significantly greater than 50% chance of success.

The second assumption in [2] deals with the fact that K_1 and K_2 are not independent of one another, but rather are connected by the relationship $K_2 = K_1 \oplus P_1 \oplus P_2$, where P_1 and P_2 are fixed and publicly known. In order to rule out the possibility that this relationship weakens HMAC, Bellare assumes that $f(x, y)$ satisfies a weak form of pseudorandomness with respect to its other argument. Namely, fix $x = \mathrm{IV}$, which is the value of x in the first iteration of $f(x, y)$ in the hash function h_{IV}. If $f(\mathrm{IV}, y)$ is evaluated with y set equal to one of the two related keys, that should not give an adversary any significant information about the value of $f(\mathrm{IV}, y)$ with y set equal to the other key. Here the adversary does not know either key, but does know the relationship between them.

More precisely, following [5], Bellare has the related-key adversary attempt to distinguish with success probability significantly greater than 50% between random answers to queries and answers coming from the related keys. That is, suppose you choose a random $K \in \{0, 1\}^b$ and flip a coin. The adversary makes two queries: it asks for $f(\mathrm{IV}, K \oplus P_1)$ and for $f(\mathrm{IV}, K \oplus P_2)$. If your coin was heads, you answer correctly, whereas if it was tails, then you give random replies. If the adversary is unable to guess heads or tails with significantly greater than 50% chance of success, then the compression function $f(x, y)$ (with the fixed IV) is said to be secure against the appropriate type of related-key attack.

2. Overview of results

2.1. Separation between HMAC$^{\mathrm{std}}$ and HMAC$^{\mathrm{Bel}}$. In cryptography often "the devil is in the details." The keylength specifications for HMAC are different in the

[3] It was actually a slightly weaker assumption, called *weak collision-resistance*, that was needed in [3]. However, the collision-finding techniques in [30, 31] show that even this weaker property fails for the commonly-used hash functions.

standards [4] than in the definition of HMAC used in the security proof in [2]. We shall refer to the former variant as $\mathrm{HMAC}^{\mathrm{std}}$ and the latter one as $\mathrm{HMAC}^{\mathrm{Bel}}$.

Theorem 1. *Let f be any compression function that compresses by a factor of at least 3 and satisfies the conditions in* [2] *that are needed for the proof of security for $HMAC^{\mathrm{Bel}}$, and let f^* be the slightly modified function defined in §3 below. Then f^* satisfies the same conditions, and so $HMAC^{\mathrm{Bel}}$ with f replaced by f^* still has the security level established by the proof. But if f is replaced by f^* in the version $HMAC^{\mathrm{std}}$ that is in the standards* [4]*, then $HMAC^{\mathrm{std}}$ has no security at all, because the message tags do not depend on the keys and can be computed by anyone.*

It should be noted that we do not claim that there is any difference in the real-world security of the two versions of HMAC. Rather, this theorem is a theoretical result that points to the need for a stronger related-key attack assumption than the one in [2] if one wants the proof to apply to HMAC as defined in the standards.

2.2. **Attack on HMAC as standardized in** [25]**.** Although the keylength k in the standard [4] is the same as the taglength c, in the security proof in [3] that supports [4] there is nothing that requires that they be the same. In fact, as far as the security proof is concerned, there is no reason to choose k greater than $c/2$. So it is not surprising that the HMAC standardization in [25] with $c = 160$ allows 80-bit keys. The security definitions assume the single-user setting, where there is no known reason to insist on longer keys. However, in §4 we describe a practical attack in the multi-user setting (see also [11]). Thus, even though HMAC as standardized in [25] is "provably secure," it is insecure when there are a large number of users. In fact, if there are 2^a users, then it has at most $80 - a$ bits of security.

In §§3-4 we show how security issues arise because of inconsistences in the standards and security proofs in [3, 4, 25, 2] concerning whether the keylength is c, $c/2$, or b. This discrepancy is quite surprising, given the widespread use of HMAC and the insistence by cryptographers who work in provable security that a careful match between specifications and formal security proofs is crucial in both the design and analysis of protocols.

2.3. **Drawbacks of the main result in** [2]**.** In reducing a problem P' to a problem P, one often uses such phraseology as "there exists an efficient algorithm for P' with an oracle for P." However, the words "there exists" traditionally mean that one has an explicitly described algorithm; they do not have the much weaker meaning that such words have in existence theorems in mathematics. (See [27] for a discussion of this distinction.)

An example of the misuse of the words "there exists an efficient algorithm" would be to say that there exists an efficient algorithm for finding a collision in the most recent improved version of the Secure Hash Algorithm (or any other hash function). Trivially, any function from an extremely large set to a much smaller set has a vast number of collisions. Therefore, a collision exists, and one particular collision can be hardwired into an algorithm that simply outputs the collision. But to say that "an extremely efficient algorithm exists to break the Secure Hash Algorithm" is useless and misleading.

An analogous problematic use of the notion of an algorithm existing occurs several times in [2]. In §6 we identify the places where this occurs and describe the loss of tightness that follows if one gives a realistic interpretation to the argument in [2]. It turns out that the resulting concrete security guarantees are quite weak.

2.4. **New proof.** In §7 we give a new, self-contained proof of security without collision-resistance. Although this proof is based on very similar ideas to the proof in [2], it gives a stronger result. In particular, it solves the "interesting open problem" mentioned in §3.2 of [2]. But even this improved result by itself is probably not good enough to serve as a convincing real-world guarantee of security of HMAC, as we discuss in §8.

Stylistically, our proof resembles the 1996 security proof with collision-resistance in [3] much more than it resembles the proof without collision-resistance in [2]. That is, it is written in a style that was popular in the 1990s before the introduction of turgid notation and "game-hopping" caused many security proofs to become virtually unreadable. Like the proof in [3], our proof is straightforward and is intended to be accessible to anyone with math or computer science background.

3. Keylength

The main difference between the Nested Message Authentication Code (NMAC) and the modified version HMAC that was introduced for reasons of real-world efficiency is that in HMAC the keys are inserted in a way that is less natural, at least from a theoretical point of view. In the first place, the keys enter in the second argument of the compression function $f(x, y)$, $(x, y) \in \{0, 1\}^c \times \{0, 1\}^b$, which typically consists of 512 bits. No one wants to use such long keys, so in [4], where the recommended bitlength of the keys is 128 or 160, they are padded with 384 or 352 zero bits.[4] In the second place, the two keys for HMAC are formed by XORing a single key with two fixed (and publicly known) padding vectors, called ipad and opad.

One of the main goals of [2] is to extend security results from NMAC to HMAC. However, the definition of HMAC used in [2] specifies a random key of bitlength b, and so the security results apply only to this HMAC, not to the version that is implemented in practice, for example in [4]. We next prove Theorem 1, a separation result that shows that the security assumption used for HMAC as defined in [2] is insufficient for the security of HMAC as defined in [4].

For any compression function $f(x, y)$ from $\{0, 1\}^c \times \{0, 1\}^b$ to $\{0, 1\}^c$, define the corresponding function $f^*(x, y)$ as follows: $f^*(x, y) = 0$ if both

(i) $x = \mathrm{IV}$ (the initializing vector for the hash function being used) and
(ii) the last $b - c$ bits of y coincide with the last $b - c$ bits of either of the two padding vectors ipad or opad;

for all other (x, y) we set $f^*(x, y) = f(x, y)$.

Proof of Theorem 1. If $f(x, y)$ satisfies the two assumptions in [2] – namely, pseudorandomness as a function of y for a fixed hidden random x, and immunity from the appropriate related-key attack for fixed known $x = \mathrm{IV}$ and y equal to two related keys – then we claim that $f^*(x, y)$ also satisfies these assumptions. The first property

[4] The two most widely-used hash functions, MD5 and SHA-1, have tags of 128 and 160 bits, respectively. In both cases $b = 512$. Wang *et al.* showed in [31] that collisions can be found for MD5 in roughly 2^{39} operations and in [30] that collisions can be found for SHA-1 in roughly 2^{63} operations (see also [12]). However, no attack faster than a generic birthday attack has yet been found against HMAC with either MD5 or SHA-1.

still holds because there is negligible probability 2^{-c} that $x = \mathrm{IV}$, and the second property holds because there is negligible probability at most 2^{1-c} that for a random b-bit K one has $b - c$ zero bits at the end of either $K \oplus \mathrm{ipad}$ or $K \oplus \mathrm{opad}$. Here we are using the assumption that $f(x, y)$ compresses by a factor of at least 3, that is, $b \geq 2c$. Thus, the assumptions in the proof of security hold for $\mathrm{HMAC}^{\mathrm{Bel}}$ with $f(x, y)$ replaced by $f^*(x, y)$. However, if $f(x, y)$ is replaced by $f^*(x, y)$ in the standardized version $\mathrm{HMAC}^{\mathrm{std}}$, where the last $b - c$ bits of the keys agree with those in either ipad or opad, then the first iteration of the compression function outputs zero in both the inner and outer h_{IV} computations. The tag hence does not depend on the key. □

Thus, in order for the security proof to apply to the version of HMAC that has been standardized in [4] the following stronger related-key assumption is needed. Suppose you choose a random $K \in \{0, 1\}^c$ and flip a coin. Let $K^0 \in \{0, 1\}^b$ denote the key padded by $b - c$ zero bits. The adversary makes two queries: it asks for $f(\mathrm{IV}, K^0 \oplus \mathrm{ipad})$ and for $f(\mathrm{IV}, K^0 \oplus \mathrm{opad})$. If your coin was heads, you answer correctly, whereas if it was tails, then you give random replies. If the adversary is unable to guess heads or tails with significantly greater than 50% chance of success, then the compression function $f(x, y)$ (with the fixed IV) satisfies the desired related-key condition.

4. A practical attack on $\mathrm{HMAC}^{\mathrm{NIST}}$

The attack in this section is a special case of a type of attack described in [11] that applies to a wide range of protocols in the multi-user setting.

Suppose that HMAC is being implemented with keys of 80 bits and tags of c bits, and there are 2^a users (that is, 2^a sessions from which the adversary can make queries). The attacker chooses an arbitrary fixed message M and queries each user for the tag of M under the user's key. The attacker then chooses random keys and for each key computes the corresponding tag of M and looks for a match with a user's tag. Once the attacker finds a match, she hopes (see below) that the collision occurred because the randomly chosen key happens to coincide with the user's secret key, in which case she has broken HMAC (in the sense of existential key recovery, see §5 of [20]). The expected number of keys she has to run through before one of them collides with a user's key is 2^{80-a}.

Often c is greater than 80 – in many applications the recommended value is $c = 160$; for example, see [4]. Then a collision of tags most often comes from a collision of keys, so the attacker really finds a user's key in time roughly 2^{80-a}. However, small taglengths might be allowed in settings when users are not worried about tag-guessing attacks. If $c < 80$ the above attack fails because a collision that's found is most likely just a collision of tags corresponding to different keys. In that case a slight modification of the above attack removes this difficulty. Namely, the attacker queries s different fixed messages, where s is chosen so that $sc > 80$. This in effect lengthens the tags and allows the adversary to be confident that a simultaneous collision of all s tags was caused by a key-collision. The attacker's running time is increased only by a factor of $s \approx 80/c$. For example, the attack on an application with 80-bit keys and 32-bit tags would take just 3 times as long as an attack when the taglength is 160.

5. A remark on a very weak form of collision-resistance

As mentioned in the Introduction, the primary objective of Bellare's paper [2] was to restore the proof-based guarantee for HMAC that had been undermined by the work [30, 31] that found collisions in MD5 and SHA-1. However, after proving the theorem with weak collision-resistance, the authors of [3] had commented:

> **Remark 4.5.** The weak-collision-freeness assumption made in the theorem can be replaced by the significantly weaker assumption that the inner hash function is collision-resistant to adversaries that see the hash value only after it was hashed again with a different secret key.

It is interesting to note that the work of [30, 31] does not compromise this weaker property. Namely, an oracle for weak collision-resistance means one that responds to a message query M by giving $h(K_1, M)$, where K_1 is a hidden random key; whereas an oracle for the "significantly weaker" property in the remark means one that responds by giving $f(K_2, h(K_1, M)^0)$, where K_1 and K_2 are hidden random keys. Given the first type of oracle, the attackers in [30, 31] can simply query an arbitrary initial message block M_1 and then set $\mathrm{IV} = h(K_1, M_1)$. Then the collision they find for h_{IV} will immediately lead to a collision for the original $h(K_1, .)$. However, given the second type of oracle the attackers in [30, 31] are apparently stymied.

This leads us to ask: if the theorem in [3] can be proved with a weaker assumption that still (so far as we know) holds for MD5 and SHA-1, then why was it necessary to write [2] in order to recover the proof-based guarantee? We cannot be sure, but the reason might have been (although this was not mentioned anywhere in [3] or [2]) a loss of tightness in the theorem in [3] that occurs if one passes to the weaker assumption. If one makes the obvious modifications in the proof in [3] needed to accommodate the weaker assumption, one finds a tightness gap of q (the bound on the number of queries). Whether or not this is important in practice depends on how large q is likely to be.

It should also be recalled that another objective of Bellare in [2] was to establish a property of NMAC and HMAC that is stronger than just being a secure message authentication code – namely, being a pseudorandom function.

6. Questions about Bellare's NMAC proof

By far the lengthiest argument in [2] is the proof of the main security result for NMAC (see Theorem 3.3); the extension from NMAC to HMAC is a relatively short proof. In §§6–7 we are concerned with the security result for NMAC and not its extension to HMAC.

We quote a passage from [2] that contains a crucial step in the NMAC proof. In the excerpt A_6 denotes an adversary that takes two messages as input and is attacking the pseudorandomness of a function h (which is the compression function, that is, f in our notation), B^+ is the space of messages, $\|M\|_b$ denotes the number of b-bit blocks in M, and $\mathbf{Adv}_h^{\mathrm{prf}}(A_6(M_1, M_2))$ is a measure of the success probability of A_6. The passage is from the last long paragraph in §3.3:

> Let $M_1^*, M_2^* \in B^+$ be distinct messages such that $\|M_1^*\|_b \le \|M_2^*\|_b \le n$ and
>
> $$\mathbf{Adv}_h^{\mathrm{prf}}(A_6(M_1, M_2)) \le \mathbf{Adv}_h^{\mathrm{prf}}(A_6(M_1^*, M_2^*))$$

> for all distinct $M_1, M_2 \in B^+$ with $\|M_1\|_b \leq \|M_2\|_b \leq n$, where n is as in the Lemma statement. Now let A be the adversary that has M_1^*, M_2^* hardwired in its code and, given oracle g, returns $A_6^g(M_1^*, M_2^*)$. The adversary A has time-complexity as claimed in the Lemma statement.

In other words, M_1^*, M_2^* are defined to be a message-pair for which the adversary A_6 is maximally successful. Such a pair obviously exists. But how in the world could one find such M_1^*, M_2^* algorithmically? It's a tremendous leap to let A be an efficient algorithm that somehow has the pair of optimal messages for A_6 hardwired into its code.

To put it another way, let's ask: What sort of security theorem can come out of this type of unconstructible adversary? Such a theorem essentially says that if NMAC has an adversary with a non-negligible advantage, then an algorithm A *exists* that solves a certain hard problem. However, A exists only in the sense of a mathematical existence theorem, and there is no known way to derive a concrete security bound – that is, one cannot conclude anything about the actual security of NMAC.

We find this type of argument elsewhere in [2]: in the proof of Lemma 3.2 relating the pseudorandomness of NMAC to the almost-universal property and the pseudorandomness of the compression function (see the last paragraph of §3.4); in the proof of the generalization of the main theorem, Theorem 3.4 (see the sentence in parentheses following (21) in §3.5); and in the proof of Lemma 4.2, which is used to prove security under an assumption on the compression function that is weaker than pseudorandomness (see the last sentence of §4). The author justifies these steps by citing "a standard 'coin-fixing' argument" (p. 22), but in fact the steps invalidate any attempt to use the theorems to derive concrete security bounds.

It should be stressed that the general argument in [2] does not depend on the coin-fixing, and so the qualitative result remains valid. However, the tightness of the reduction is greatly affected. Recall that in a reduction from a problem P' to a problem P the *tightness gap* is defined as $t'\epsilon/t\epsilon'$, where (ϵ, t) and (ϵ', t') are the success probability (or "advantage") and running time of algorithms for P and P', respectively. In the main NMAC security result of [2] (see inequality (4) of Theorem 3.3) the tightness gap comes from a term that in our notation would be written $\binom{q}{2}(2n\epsilon')$, where q denotes the number of queries allowed and ϵ' is the advantage of a certain adversary A that attacks the pseudorandomness of f.[5] The other side of inequality (4) of [2] is the advantage ϵ of an adversary (denoted A_{fh} in §7) that attacks the pseudorandomness of the NMAC function. That is, (4) gives $q^2n\epsilon' \geq \epsilon$, so there is a tightness gap of q^2n for the advantages. However, because of the coin-fixing, Theorem 3.3 claims a running time t' for the adversary A of order only nT (that is, the time for an evaluation of the iterated hash function). Neglecting T (that is, replacing it by 1), this reduces the overall tightness gap to $\frac{\epsilon}{\epsilon'} \cdot \frac{t'}{t} \approx (q^2n) \cdot \frac{n}{t} = \frac{q^2n^2}{t}$. This is the computation used in §3.2 of [2] in order to conclude that "the bound justifies NMAC up to roughly $2^{c/2}/n$ queries."

More precisely, Bellare's argument can be summarized as follows. Assuming that there is no f-adversary that's faster than exhaustive search, we can say that the adversary A with running time of order n must have advantage $\epsilon' \leq n/2^c$. Then

[5] In a setting where the adversary is trying to guess a bit with a greater than 50% chance of success, an "advantage" ϵ means that the adversary has at least $(1+\epsilon)/2$ probability of success and at most $(1-\epsilon)/2$ probability of failure, the difference being at least ϵ.

$\epsilon \leq q^2 n\epsilon' \leq q^2 n^2/2^c$, and the result claimed in his theorem has content provided that this is less than 1, i.e., $q < 2^{c/2}/n$.

Without the coin-fixing argument, let's take the running time of A to be roughly t, which can be assumed to have order of magnitude qn. Hence there's another q term in the inequalities for ϵ' and ϵ; that is, $\epsilon \leq q^2 n\epsilon' \leq q^2 n(qn/2^c) = q^3 n^2/2^c$. This means that the bound justifies NMAC only up to roughly $2^{c/3} n^{-2/3}$ queries. Unfortunately, for $c = 160$ and, say, $n = 2^{20}$, Bellare's theorem in [2] now gives just 40 bits of security.

It turns out that a more efficient proof of NMAC security can be given that leads to a significantly tighter result. We do this in the next section. But even our tighter result needs to be interpreted with caution and by itself probably does not give a very convincing security guarantee. In §8 we'll comment on possible interpretations and reasons for skepticism.

7. Proof without game-hopping of NMAC security without collision-resistance

A compression function f that maps from $\{0,1\}^c \times \{0,1\}^b$ to $\{0,1\}^c$ is said to be an (ϵ, t, q)-secure pseudorandom function if no adversary can distinguish between f with a hidden key and a random function with advantage $\geq \epsilon$ in time $\leq t$ with $\leq q$ queries. Suppose that NMAC is constructed from f. Then NMAC is said to be an (ϵ, t, q, n)-secure pseudorandom function if no adversary can distinguish between NMAC with hidden keys and a random function with advantage $\geq \epsilon$ in time $\leq t$ with $\leq q$ queries of block length $\leq n$.

Theorem 2. *Suppose that f is an (ϵ, t, q)-secure pseudorandom function. Then NMAC is a $(3n(3n\epsilon + \binom{q}{2}2^{-c}), t - (2qnT + Cq\log q), q, n)$-secure pseudorandom function. Here C is an absolute constant and T denotes the time for one evaluation of f.*

Proof. We will prove the following equivalent statement: if the compression function f is a $(\frac{1}{3n}(\frac{\epsilon}{3n} - \binom{q}{2}2^{-c}), t + (2qnT + Cq\log q), q)$-secure pseudorandom function, then NMAC is an (ϵ, t, q, n)-secure pseudorandom function. The proof starts by supposing that we have an adversary that defeats the pseudorandomness test for NMAC with advantage $\geq \epsilon$ in time $\leq t$ with at most q queries of block-length at most n, and then proceeds to construct an adversary for f that satisfies the specified parameters.

Let h be the corresponding iterated function, and let fh be the NMAC function, which for a key $K = (K_1, K_2)$ is defined as $fh(M) = f(K_2, h(K_1, M))$, where $M = (M_1, \ldots, M_m)$ is an m-block message, $m \leq n$. For simplicity of notation in what follows we'll disregard the padding; for example, the second argument in f in the definition of fh needs to be padded by $b-c$ bits (denoted by overlining in [3]). Let $g(M)$ denote a random function of messages, and let $g'(M_1)$ denote a random function of 1-block messages. In response to an input of suitable length, g' or g outputs a random c-bit string, subject only to the condition that if the same input is given a second time (in the same run of the algorithm) the output will be the same. In the test for pseudorandomness the oracle is either a random function or the function being tested, as determined by a random bit (coin toss).

The theorem says: If f is a pseudorandom function (prf), then so is fh. To prove this we suppose that we have an adversary A_{fh} that, interacting with its oracle O_{fh}, defeats the prf-test for fh, and we then use A_{fh} to construct a set of adversaries, at least one of which, interacting with the oracle O_f, defeats the prf-test for f. Each adversary makes at most the same number of queries as A_{fh} and has a comparable running time. More precisely, the bound $t + (2qnT + Cq \log q)$ on the running time of one of the adversaries comes from the time required to run A_{fh}, make at most $2q$ computations of h-values, and store at most $2q$ values (coming from oracle responses and h-computations) in lexicographical order and sort them looking for collisions.

For an oracle O we let $O(M)$ denote the response of O to the query M. The adversary A_f is given an oracle O_f and, using A_{fh} as a subroutine, has to decide whether O_f is $f(K_2, .)$ or $g'(.)$. She chooses a random K_1 and presents the adversary A_{fh} with an oracle that is either $f(K_2, h(K_1, .))$ or else $g(.)$ – that is, she simulates O_{fh} (see below). In time $\leq t$ with $\leq q$ queries A_{fh} is able with probability $\frac{1+\epsilon}{2}$ to guess correctly whether O_{fh} is fh with hidden keys or a random function g. Here is how A_f simulates O_{fh}: in response to a query M^i from A_{fh}, she computes $h(K_1, M^i)$, which she queries to O_f, and then gives A_{fh} the value $O_f(h(K_1, M^i))$. Eventually A_{fh} states whether it believes that its oracle O_{fh} is fh or g, at which point A_f states the same thing for the oracle O_f – that is, if A_{fh} said fh, then she says that O_f must have been f, whereas if A_{fh} said that O_{fh} is g, then she says that O_f is g'. Notice that if the oracle O_f is $f(K_2, .)$, then the oracle O_{fh} that A_f simulates for A_{fh} is fh (with random key $K = (K_1, K_2)$); if the oracle O_f is $g'(.)$, then the oracle that A_f simulates for A_{fh} acts as g with the important difference that if $h(K_1, M^i)$ coincides with an earlier $h(K_1, M^j)$ the oracle outputs the same value (even though $M^i \neq M^j$) rather than a second random value.[6] If the latter happens with negligible probability, then this algorithm A_f is as successful in distinguishing f from a random function as A_{fh} is in distinguishing fh from a random function. Otherwise, three sequences of adversaries $A_f^{(m)}$, $B_f^{(m)}$, and $C_f^{(m)}$ come into the picture, as described below.

The general idea of these adversaries is that they each test the oracle O_f by looking for collisions between h-values of two different messages M^i, M^j queried by A_{fh}. More precisely, the m-th adversary in a sequence works not with all of a queried message, but rather with the message with its first $m-1$ (or m) blocks deleted. If a collision is produced, then with a certain probability O_f must be $f(K_2, .)$; however, there is also a possibility that O_f is $g'(.)$ and a collision occurs for the same messages with one further message block deleted, and this brings in the $(m+1)$-st adversary in one of the sequences.

First we make a remark about probabilities, which are taken over all possible coin tosses of the adversary, all possible keys, the oracle's "choice bit" (which determines whether it is the function being tested or a random function), and the coin tosses of the oracle in the case when it outputs a random function.[7] If the adversary's oracle is f or fh with hidden keys, then the adversary's queries in general depend on the keys (upon which the oracle's responses depend) as well as the adversary's coin tosses. However, if the adversary's oracle is a random function – which is the situation when A_f fails and the

[6] If this happens, then A_f has to make a random guess about whether O_f is $f(K_2, .)$ or $g'(.)$, since she knows that she might have failed to simulate O_{fh} and so cannot rely on anything useful coming from A_{fh}.

[7] The term "over all possible coin tosses" means over all possible runs of the algorithm with each weighted by 2^{-s}, where s is the number of random bits in a given run.

sequences of adversaries $A_f^{(m)}$, $B_f^{(m)}$, $C_f^{(m)}$ are needed – then the oracle responses can be regarded simply as additional coin tosses, and the adversary's queries then depend only on the coin tosses and are independent of the keys. This is an important observation for understanding the success probabilities of the adversaries.

For the i-th message query M^i we use the notation M_ℓ^i to denote its ℓ-th block, we let $M^{i,[m]} = (M_1^i, \ldots, M_m^i)$ be the truncation after the m-th block, and we set $M^{i,(m)} = (M_m^i, M_{m+1}^i, \ldots)$, that is, $M^{i,(m)}$ is the message with the first $m-1$ blocks deleted. We say that a message is "non-empty" if its block length is at least 1.

For $m \geq 1$ we define α_m to be the probability, taken over all coin tosses of A_{fh} (including those coming from random oracle responses) and all keys K_1, that the sequence of A_{fh}-queries M^i satisfies the following property:

(1_m) there exist i and j, $j < i$, such that $M^{i,(m+1)}$ and $M^{j,(m+1)}$ are non-empty,

$$M^{i,[m]} = M^{j,[m]}, \qquad \text{and} \qquad h(K_1, M^{i,(m+1)}) = h(K_1, M^{j,(m+1)}).$$

For $m = 0$ we similarly define α_0 as the probability that $h(K_1, M^i) = h(K_1, M^j)$ for some i and j, $j < i$.

For $m \geq 1$ we define β_m to be the probability, taken over all coin tosses of A_{fh} and all pairs of keys (K_1, K_1'), that the sequence of A_{fh}-queries satisfies the following property:

(2_m) there exist i and j, $j < i$, such that $M^{i,(m+1)}$ and $M^{j,(m+1)}$ are non-empty,

$$M^{i,[m]} \neq M^{j,[m]}, \qquad \text{and} \qquad h(K_1, M^{i,(m+1)}) = h(K_1', M^{j,(m+1)}).$$

For $m \geq 2$ we further write $\beta_m = \beta_m' + \beta_m''$, where β_m' denotes the probability of collisions with $M^{i,[m-1]} \neq M^{j,[m-1]}$ and β_m'' denotes the complementary probability, that is, the probability of collisions with $M^{i,[m]}$ and $M^{j,[m]}$ differing only in their m-th block.

Finally, for $m \geq 1$ we define γ_m to be the probability, taken over all coin tosses of A_{fh} and all pairs of keys (K_1, K_1''), that the sequence of A_{fh}-queries satisfies the following property:

(3_m) there exist i and j, $j < i$, such that $M^{i,(m+1)}$ and $M^{j,(m+2)}$ are non-empty,

$$M^{i,[m]} \neq M^{j,[m]}, \qquad \text{and} \qquad h(K_1, M^{i,(m+1)}) = h(K_1'', M^{j,(m+2)}).$$

We now return to the situation where with non-negligible probability α_0 the queries made by A_{fh} lead to at least one collision $h(K_1, M^i) = h(K_1, M^j)$. Note that the advantage of the adversary A_f is at least $\epsilon - \alpha_0$.

The first adversary in the sequence $A_f^{(m)}$ is A_f', which is given the oracle O_f that is either $f(K_1, .)$ with a hidden random key K_1 or else $g'(.)$. As A_f' runs A_{fh}, giving random responses to its queries, he[8] queries O_f with the first block M_1^i of each A_{fh}-query M^i. If $M^{i,(2)}$ is non-empty, he then computes $y_i = h(O_f(M_1^i), M^{i,(2)})$; if $M^{i,(2)}$ is empty, he just takes $y_i = O_f(M_1^i)$. If O_f is $f(K_1, .)$, then y_i will be $h(K_1, M^i)$, whereas if O_f is $g'(.)$, then y_i will be $h(L_i, M^{i,(2)})$ for a random key $L_i = O_f(M_1^i)$ if $M^{i,(2)}$ is non-empty and will be a random value L_i if $M^{i,(2)}$ is empty. As the adversary A_f' gets these values, he looks for a collision with the y_j-values obtained from earlier queries M^j.

[8] We'll alternate the adversaries' genders, not so much for reasons of gender equity, but rather for the purpose of avoiding confusion between two successive adversaries in a discussion.

If a collision occurs, he guesses that O_f is $f(K_1, .)$; if not, he randomly chooses between the two alternatives.

It is, of course, conceivable that even when O_f is $g'(.)$ there is a collision $h(L_i, M^{i,(2)}) = h(L_j, M^{j,(2)})$. Note that $L_i = L_j$ if $M_1^i = M_1^j$, but L_i and L_j are independent random values if $M_1^i \neq M_1^j$. In other words, we have (1_1) or (2_1) with $K_1 = L_i$, $K_1' = L_j$. Recall that the probability that this occurs is $\leq \alpha_1 + \beta_1$.

It is also possible that even when O_f is $g'(.)$ there is a collision involving one or both of the random values L_i or L_j that is produced when $M^{i,(2)}$ or $M^{j,(2)}$ is empty. The probability of this is $\leq \binom{q}{2}2^{-c}$. Bringing these considerations together, we see that the advantage of A_f' is $\geq \alpha_0 - \alpha_1 - \beta_1 - \binom{q}{2}2^{-c}$.

We are now ready to define the three sequences of adversaries $A_f^{(m)}$, $B_f^{(m)}$, $C_f^{(m)}$, $2 \leq m \leq n$, that come into play when the oracle O_{fh} that A_f simulates for A_{fh} fails to perform like a random g even when O_f is g'.

We first describe $A_f^{(m)}$. Like A_f', she runs A_{fh} once and gives random responses to its queries. Let O_f again denote the prf-test oracle for f that $A_f^{(m)}$ can query. As A_{fh} runs, for every query M^i for which $M^{i,(m)}$ is non-empty the adversary $A_f^{(m)}$ queries M_m^i to O_f and computes $y_i = h(O_f(M_m^i), M^{i,(m+1)})$ if $M^{i,(m+1)}$ is non-empty and otherwise takes $y_i = O_f(M_m^i)$. Then she looks for $j < i$ such that $M^{j,(m)}$ is non-empty, $M^{j,[m-1]} = M^{i,[m-1]}$, and $y_i = y_j$. If she finds such a collision, she guesses that O_f is $f(K_1, .)$; otherwise, she randomly guesses whether O_f is $f(K_1, .)$ or $g'(.)$.

The adversary $B_f^{(m)}$ acts as follows. He starts by choosing a random key K_1'. Like $A_f^{(m)}$, for every query M^i for which $M^{i,(m)}$ is non-empty he queries M_m^i to O_f and computes $y_i = h(O_f(M_m^i), M^{i,(m+1)})$ if $M^{i,(m+1)}$ is non-empty and otherwise takes $y_i = O_f(M_m^i)$. But unlike $A_f^{(m)}$, he also computes $z_i = h(K_1', M^{i,(m)})$ whenever $M^{i,(m)}$ is non-empty. He looks for $j < i$ such that $M^{j,(m)}$ is non-empty, $M^{j,[m-1]} \neq M^{i,[m-1]}$, and z_j coincides with y_i. If he finds such a collision, he guesses that O_f is $f(K_1, .)$; otherwise he makes a random guess about whether O_f is $f(K_1, .)$ or $g'(.)$.

Finally, the adversary $C_f^{(m)}$ acts similarly to $B_f^{(m)}$. She also starts by choosing a random key K_1''. Whenever $M^{i,(m)}$ is non-empty she queries M_m^i to O_f and computes $y_i = h(O_f(M_m^i), M^{i,(m+1)})$ if $M^{i,(m+1)}$ is non-empty and otherwise takes $y_i = O_f(M_m^i)$. In addition, whenever $M^{i,(m+1)}$ is non-empty she computes $z_i = h(K_1'', M^{i,(m+1)})$. She looks for $j < i$ such that $M^{j,(m)}$ is non-empty, $M^{j,[m-1]} \neq M^{i,[m-1]}$, and y_j coincides with z_i. (This is the reverse of what $B_f^{(m)}$ looks for.) If she finds such a collision, she guesses that O_f is $f(K_1, .)$, and otherwise makes a random guess about O_f.

We claim that we have the following lower bounds for the advantages of the adversaries, where we set $\delta = \delta(q, c) = \binom{q}{2}2^{-c}$:

A_f: $\epsilon - \alpha_0$;

A_f': $\alpha_0 - \alpha_1 - \beta_1 - \delta$;

$A_f^{(m)}$, $m \geq 2$: $\alpha_{m-1} - \alpha_m - \beta_m'' - \delta$;

$B_f^{(m)}$, $m \geq 2$: $\beta_{m-1} - \gamma_{m-1} - \delta$;

$C_f^{(m)}$, $m \geq 2$: $\gamma_{m-1} - \beta'_m - \delta$.

The adversary $A_f^{(m)}$ takes advantage of the α_{m-1} probability of a collision of the form (1_{m-1}), and if such a collision occurs she guesses that O_f is $f(K_1, .)$. The possibility that O_f is really $g'(.)$ is due to three conceivable circumstances – a collision of the form (1_m) (with $K_1 = O_f(M_m^i)$), a collision of the form (2_m) (with $K_1 = O_f(M_m^i)$ and $K'_1 = O_f(M_m^j)$), or a collision among random values (of which the probability is bounded by δ). The arguments for $B_f^{(m)}$ and $C_f^{(m)}$ are similar. Notice that the purpose of having $C_f^{(m)}$ look for $y_j = z_i$ – as opposed to $z_j = y_i$ as in the case of $B_f^{(m)}$ – is to "level out the messages." That is, whenever the guess by $B_f^{(m)}$ that O_f is $f(K_1, .)$ is wrong because of a collision of the form (3_{m-1}) with i and j reversed and with $K''_1 = O_f(M_m^i)$ we want a wrong guess by $C_f^{(m)}$ to lead back to a situation where M^i and M^j have been truncated by the same amount.

Trivially we have $\alpha_n = \beta_n = \gamma_{n-1} = 0$, and so the adversaries stop at the latest with $A_f^{(n)}$, $B_f^{(n)}$, $C_f^{(n-1)}$. The sum of all the advantages of the $3n-2$ adversaries telescopes and is equal to $\epsilon - (3n-3)\delta$. Thus, one of the $3n-2$ summands must be $\geq \frac{\epsilon}{3n-2} - \frac{3n-3}{3n-2}\delta > \frac{\epsilon}{3n} - \delta$; the corresponding adversary has advantage $> \frac{\epsilon}{3n} - \delta$.

Unfortunately, we have no way of knowing in advance which adversary is the "good" one with this non-negligible advantage. So we need to make a random selection that results in a further loss of tightness of the reduction. That is, the algorithm to attack the pseudorandomness of f consists of randomly choosing one of the $3n-2$ adversaries A_f, A'_f, $A_f^{(m)}$ $(2 \leq m \leq n)$, $B_f^{(m)}$ $(2 \leq m \leq n)$, $C_f^{(m)}$ $(2 \leq m \leq n-1)$, and running it. With probability $1/(3n-2) > 1/(3n)$ you get the "good" adversary. Thus, the advantage of this algorithm is at least $\frac{1}{3n}(\frac{\epsilon}{3n} - \delta)$, as claimed. □

8. Interpretations

How useful is the bound in Theorem 2 as a guarantee of real-world security? That depends on how one chooses to approach the question. We first show that the bound is in some sense optimal. We then discuss its limitations.

8.1. **Optimality of bound.** In §3.2 of [2], Bellare explains that, because of the birthday attack of Preneel and van Oorschot [26], the most one can hope for from a security bound for NMAC is that it justifies NMAC up to roughly $2^{c/2}/\sqrt{n}$ queries. Namely, he argues that a better bound would imply a better generic attack on f than exhaustive key search, and such an attack is not believed to exist. Bellare shows that the bound claimed in his theorem

> justifies NMAC up to roughly $2^{c/2}/n$ queries, off from the number in the above-mentioned attack by a factor of $\sqrt{n}$. It is an interesting open problem to improve our analysis and fill the gap.

(We've changed the notation in the quotation to agree with ours.) Following the method in §3.2 of [2], let's examine our bound in Theorem 2. We want Theorem 2 to give a non-trivial conclusion under the assumption that the best attack on f is exhaustive key search. With that assumption, the advantage of any prf-adversary of f which runs in time at most t and makes at most q queries is $\epsilon \leq t/2^c$. Theorem 2 has content provided that $3n(3n\epsilon + \binom{q}{2}2^{-c}) < 1$. Supposing that $t \ll q^2/3n$, we then have $3n\epsilon \ll q^2 2^{-c}$, and so the condition becomes $3n\binom{q}{2}2^{-c} < 1$. Ignoring a 3/2 factor, this gives us $q^2 < 2^c/n$, and we can say that Theorem 2 justifies NMAC up to roughly $2^{c/2}/\sqrt{n}$ queries. Thus, we could argue that our theorem gives a best possible security bound.

8.2. **Limitations of Theorem 2.** So does that mean that we've proved a nice ironclad security guarantee for NMAC? Hardly. If we were to claim that Theorem 2 is a great improvement over the result in [2] and provides the proof of security that NMAC needs, we would be indulging in boastful hype. We would be ignoring the wise advice given in Ch. 13, v. 8–12 of [1], where it is written that to achieve knowledge the first trait one needs is humility.

And indeed, in interpreting Theorem 2 a huge dollop of humility is called for. Suppose we want to know that NMAC is an (ϵ, t, q, n)-secure pseudorandom function. In order for Theorem 2 to give us the desired assurance, we need f to be roughly an $(\epsilon/(9n^2), t, q)$-secure pseudorandom function. In other words, we have a tightness gap of about $9n^2$. Since values such as 2^{20} and 2^{30} are quite reasonable for the block-length bound, the tightness gap can be gigantic.

A second limitation of Theorem 2 is that it is in the single-user setting, where, as we saw in §4, one might have security assurances that fail in the more realistic multi-user setting.

In the third place, Theorem 2 has a very strong hypothesis – pseudorandomness of the compression function. This property is extremely difficult to evaluate for the compression functions used in practice, for example in SHA-1 and MD5. And one really has to question the value of a theorem if one has no good reason to believe the hypothesis.

We would not want to go out on a limb and say that our Theorem 2 is totally worthless. However, its value as a source of assurance about the real-world security of HMAC is questionable at best.

In our opinion none of the provable security theorems for HMAC with MD5 or SHA-1 (including the proof in [15]) by themselves provide a useful guarantee of security. At most they offer partial evidence of security that must be supplemented by hundreds of person-years of cryptanalysis of the versions of HMAC that are in use.

It is also important to note that the level of security that one needs depends on the particular application. If HMAC is being used only as a message authentication code, and a given session is fairly short-lived, then a bound of 2^{63} or even 2^{50} queries might be reasonable. Moreover, as pointed out in [4], in general only short-term security is needed, because, unlike in the case of encryption, no harm is done if an adversary determines the shared secret key after the session is over.

On the other hand, HMAC can also be used as a pseudorandom function in applications such as key-derivation [14, 17, 21] and one-time passwords [24]. In those settings one often needs a much greater level of assurance than anything that's provided by such theoretical results as our Theorem 2 or the theorems in [2].

9. Conclusions

1. A security proof using standard definitions based on the single-user setting does not necessarily give any useful guarantee of the security of the protocol in a multi-user setting.

2. HMAC with 80-bit keys – such as the version standardized in [25] – should probably not be used in the multi-user setting, because it has at most $80 - a$ bits of security when there are 2^a users.

3. When HMAC is used with a hash function that is not collision-resistant, confidence in its security cannot come from the proof in [2] – or even from our proof in §7 – but rather must be based upon the large number of person-years that engineers and cryptanalysts have devoted to testing it. This is especially the case in an application where one needs pseudorandomness and where short-term security is not enough.

4. The coin-fixing technique, which was described in one form in §7.3 of [6] and appeared in a somewhat different form in [2], should be used with caution. One should ask what concrete bounds can result if such arguments are used. In general this technique cannot be employed as a magic bullet to convert a non-tight bound into a tight one. In addition to the proofs in [2], other proofs that make questionable coin-fixing arguments can be found in [32] (see Lemma 1) and [33] (proof of Theorem 1).

5. A defect or an unstated strong assumption in a theorem – particularly when it's in a paper that appears in the proceedings of a prestigious conference – is likely to propagate to other papers as different authors use it to prove their own results. For example, Theorem 2 of [16] contains a bound that's of questionable practical significance as a result of the authors' reliance on a bound in [2] which, in turn, was derived using a coin-fixing argument.

6. Game-hopping proofs [6, 28] are often especially prone to errors, misunderstandings, and omissions because they are much lengthier than proofs written in a conventional mathematical style. In conferences such as Crypto, program committee members are instructed that they are not responsible for reading anything that is not contained in the main body of a paper, and a strict page limit is imposed on the main body of submissions. Long proofs, such as proofs with sequences of games, are omitted or relegated to appendices that are rarely read by referees. Another reason why game-hopping proofs often receive even less peer review than other proofs is that many people find them especially hard to read. See [18, 19] for further discussion of the drawbacks of game-hopping proofs.

7. In [29] Stern, Pointcheval, Malone-Lee, and Smart comment (in connection with the error in the original security proof for OAEP [7]) that proofs "need time to be validated through public discussion" and that "flaws in security proofs themselves might have a devastating effect on the trustworthiness of cryptography." One can only hope that the research culture in cryptography changes in such a way that proofs start to get the detailed peer review they need.

Acknowledgments

We would like to thank Sanjit Chatterjee, Palash Sarkar, and Greg Zaverucha for helpful technical corrections and comments, and Ann Hibner Koblitz for valuable editorial corrections and suggestions.

Postscript

After the first version of this paper was posted, Bellare contacted us and told us that he strongly objected to our language – especially the word "flaw" – and that Theorem 3.3 of [2] was meant to be understood in the sense of a certain non-uniform model of complexity. In that model a proof is permitted to use an algorithm that exists in the mathematical sense, irrespective of whether or not one can give a feasible way to construct it.

In the paper we explained what we meant by "flaw" in the paragraph that began, "To put it another way, let's ask what sort of security theorem can come out of this type of non-constructive adversary?" That is, the problem with the proof (the coin-fixing step) was not in the area of formal correctness, but rather in its implications for usability of the result. Nevertheless, given the strong objections Bellare raised and the possibility that readers might be misled into believing that the proof is mathematically incorrect, we have removed the word "flaw" in this revision and have also made other similar changes of language.

In this postscript we elaborate on some of the issues from a practice-oriented viewpoint – issues that arise when the results in [2] are interpreted against the backdrop of a complexity model that allows unconstructible algorithms (by which we mean algorithms for which no feasible construction is known).

The pseudorandomness and collision-resistance assumptions. The first point to make is that Bellare's theorem is based on the extremely strong assumption of prf-security even against an unconstructible adversary. That is, the adversary is given tremendous power, namely, access to any information (such as messages with very unusual properties with respect to some function) that exists in the mathematical sense, whether or not anyone knows a feasible way to find it.

In §8 when we explained why our Theorem 2 does not provide convincing assurance of the real-world security of HMAC, we commented that prf-security of the compression function f is a strong property for which there is little evidence in the case of MD5 and SHA-1. This is true even though our theorem was proved in the old-fashioned complexity model going back to Turing, where f was assumed to be prf-secure only against adversaries that can be efficiently constructed.

In contrast, in [2] the prf-security assumption must be interpreted as holding against much more powerful adversaries. In order for Theorem 3.3 of [2] to have content – that is, in order for the right hand side of inequality (4) to be less than 1 – one has to know that *all* low-resource (i.e., at most 2 queries and running time at most $2nT$) prf-adversaries A_2 of the compression function, whether constructible or not, have advantage less than $1/(q^2n)$. In practice, how could one have evidence for such a bound, say in the case of MD5?

Here's an example of an adversary A_2 that would cause the right side of (4) to be greater than 1. Recall that h denotes the iteration of the compression function f. Suppose that there exists a pair of messages (M^1, M^2) with distinct first blocks such that

$$\text{Prob}(h(K, M^1) = h(K, M^2)) - \text{Prob}(h(K', M^{1,(2)}) = h(K'', M^{2,(2)})) > 1/(q^2n),$$

where the first probability is assessed over all keys $K \in \{0,1\}^c$ and the second probability is assessed over all pairs of keys K', K''; here, as in the proof of Theorem 2, $M^{i,(2)}$ means the message with the first block deleted. This pair of messages is hardwired into A_2, which queries the first message blocks M_1^1 and M_1^2 to the prf-oracle O. The adversary then puts the responses into the iterated hash function and sets $y_1 = h(O(M_1^1), M^{1,(2)})$ and $y_2 = h(O(M_1^2), M^{2,(2)})$. If $y_1 = y_2$, he guesses that the oracle is $f(K,.)$, and otherwise he flips a coin to determine his guess about whether O is $f(K,.)$ or random. It is not hard to see that the advantage of this adversary is the difference of probabilities in the above inequality.

Thus, in order for Theorem 3.3 of [2] to give any security assurance at all for MD5, one would have to prove – or at least have some evidence – that the iterated hash function in MD5 satisfies the bound $\text{Prob}(h(K, M^1) = h(K, M^2)) < 1/(q^2n)$ for *all* message pairs with distinct first blocks. (Except for the fact that the inequality needn't hold for message pairs with identical first blocks, this condition is what is called "$1/(q^2n)$-almost universality.") If h is not shown to be $1/(q^2n)$-almost universal, we have to worry about the existence of a single pair (M^1, M^2) that can be hardwired into an adversary A_2 that has advantage $> 1/(q^2n)$. To be sure, a message pair for which $\text{Prob}(h(K, M^1) = h(K, M^2)) \geq 1/(q^2n)$ might not lead to such an adversary if the first blocks of the two messages are identical or if it turns out that $\text{Prob}(h(K', M^{1,(2)}) = h(K'', M^{2,(2)}))$ is non-negligible. Neither is likely, however, and certainly the hope that one of those two circumstances occurs cannot be grounds for confidence that a collision-prone pair will *not* give an adversary with advantage $> 1/(q^2n)$.

Now almost-universality is a very strong assumption to be making about the MD5 hash function. We are unaware of any convincing evidence that it holds. Notice, moreover, that $1/(q^2n)$-almost universality of h is only a *necessary* condition for Theorem 3.3 of [2] to have content, that is, for the right side of inequality (4) to be less than 1. It is not a *sufficient* condition, because it is conceivable that other types of low-resource adversaries A_2 might exist. Because of the complexity model that Bellare uses to justify his proof of Theorem 3.3, one has to know that the advantage is $< 1/(q^2n)$ for all possible adversaries that exist.

Suppose we are interested in the security of HMAC using a hash function such as SHA256 or SHA512 for which no faster-than-generic collision-finding algorithm is known. It seems to us that from the standpoint of collision-resistance of the iterated hash function, the 1996 security theorem in [3] is *stronger* than the 2006 security theorem in [2], because the former theorem is valid in the uniform complexity model, whereas the latter theorem is not. That is, the existence of a collision-prone pair (M^1, M^2) would not invalidate the security assurance coming from the theorem in [3] unless one has an efficiently constructible algorithm to find it. When deriving concrete security assurances from [3], to some extent one can be guided by the state-of-the-art in collision-finding work. However, that is not of much help when evaluating the security guarantee in Theorem 3.3 of [2], where at the very least one needs to know that no collision-prone message pairs *exist*.

Remark 1. Although Theorem 3.3 of [2] does not formally assume the $1/(q^2n)$-almost universality of the iterated hash function h, we saw that this assumption is a necessary condition in order for the theorem to have any real content. In many settings $1/(q^2n)$

is very small. As Dan Bernstein pointed out in 2005 in [10] (see also [8, 9]), with this type of assumption on h one can get a quick proof (in the uniform complexity model, of course) of NMAC security. Because of the complexity model that Bellare uses in proving Theorem 3.3, he is forced implicitly to assume a very strong property of the iterated hash function – a property that, if included explicitly in the hypothesis of the theorem, would have led to a quick proof in the uniform complexity model.

Bellare's tightness analysis. Let's return to Bellare's analysis of tightness of his bound in §3.2 of [2]. He argues that his bound in (4) "justifies NMAC up to roughly $2^{c/2}/n$ queries" (we've replaced his notation for the block-length bound by our notation n), whereas the birthday attack in [26] shows that one cannot expect a security bound that has any content with $2^{c/2}/\sqrt{n}$ queries. He then mentions the open problem of closing the $\sqrt{n}$ gap; our Theorem 2 solves this problem.

Bellare's argument goes essentially as follows. The dominant term on the right in (4) is $q^2 n \cdot \mathbf{Adv}_f^{\mathrm{prf}}(A_2)$. To find a bound on the number of queries for which (4) could have content – that is, for which the right hand side is less than 1 – he sets A_2 equal to the best low-resource algorithm that is known that applies generically – namely, exhaustive key search. For simplicity he sets $T = 1$. In the course of its running time n the adversary A_2 is able to try n keys, and so its advantage is $n2^{-c}$. Setting $q^2 n \cdot n2^{-c}$ equal to 1 leads to his estimate $2^{c/2}/n$ for the number of queries before Theorem 3.3 loses any content.

This argument makes sense in the uniform complexity model, where it is correct to say that there is no known generic attack on the pseudorandomness of f that is faster than exhaustive key search. However, in the complexity model in which Bellare wants us to understand his theorem, this is definitely not the case. We claim that one expects a low-resource adversary A_2 to exist with advantage at least equal to $2^{-c/2}$. To see this, let's suppose that we want to attack the pseudorandomness of a compression function f that has extremely good randomness properties, and in fact let's model f with a random function.

We consider the following adversary A_2. Let $u(x)$ be a fixed bit-valued function of c-bit strings that for random input has equal chance of taking value 0 and 1. For example, $u(x)$ could just pick out the 29-th bit of its input, or it could take the XOR-sum of some fixed subset of its input bits. For a 1-block message M let $\mathrm{Prob}(M)$ denote the probability, assessed over all keys K, that $u(f(K, M)) = 1$, and let M be a fixed message for which this probability is maximal. Just as in Bellare's coin-fixing argument, we hardwire M into A_2. Then A_2 works as follows. It makes only one query M to the prf-oracle O; if $u(O(M)) = 1$, it guesses that the oracle is $f(K, .)$, whereas if $u(O(M)) = 0$, it guesses that the oracle is random. It is not hard to see that the advantage of this A_2 is equal to $\mathrm{Prob}(M) - \frac{1}{2}$.

We claim that there almost certainly exists a 1-block message M such that $\mathrm{Prob}(M) > \frac{1}{2} + 2^{-c/2}$. The reason is that the standard deviation from the starting point in a random walk is equal to the squareroot of the number of steps. That is, running through the 2^c keys K, we can think of $u(f(K, M))$ as a forward step if it's 1 and a backward step if it's 0. For most M one expects to end up roughly $2^{c/2}$ steps away from the starting point. That can, of course, be on either side, but recall that our fixed M was chosen so as to maximize forward progress.

In order for a compression function f *not* to succumb to such an attack with advantage $2^{-c/2}$, it would have to have some very peculiar properties. For example, for any 1-block message M and any $i = 1, \ldots, c$, the i-th bit of $f(K, M)$ for variable K would have to be much more evenly divided between 0's and 1's than would be expected of a random function. It is extremely unlikely that the compression function for either MD5 or SHA-1 satisfies such a property.

Thus, in the non-uniform model there is a generic adversary A_2 with advantage $2^{-c/2}$. Substituting this into inequality (4) of Theorem 3.3 of [2], we see that the theorem loses content if $q > 2^{c/4}/\sqrt{n}$, which is the squareroot of the value claimed in §3.2 of [2]. If, say, $n = 2^{20}$, then the claim in [2] is that Theorem 3.3 justifies NMAC up to roughly 2^{44} queries for MD5 and 2^{60} queries for SHA-1. But in view of our A_2 described above, the Theorem says nothing at all if $q > 2^{22}$ for MD5 and if $q > 2^{30}$ for SHA-1. The mistake in §3.2 of [2] illustrates how difficult it is to appreciate all of the security implications of assuming that a compression function has prf-security even against unconstructible adversaries.

Remark 2. Alternatively, one can define a somewhat more powerful adversary A_2 as follows. For a non-empty subset S of $\{0, 1, \ldots, c\}$ define $u_S(x)$ to be the XOR-sum of the subset S of the c bits of x. For any 1-block message M, any S, and any bit t, consider the probability $\mathrm{Prob}(u_S(f(K, M)) = t)$ as K varies over 2^c keys. Let (M, S, t) be a fixed triple for which this probability is maximal, and hardwire this triple into A_2. Then A_2 queries M to the oracle O; if $u_S(O(M)) = t$, it guesses that the oracle is $f(K, .)$, whereas if $u_S(O(M)) = 1 - t$, it guesses that the oracle is random. This low-resource adversary has advantage $\geq 2^{-c/2}$ unless the compression function f has the property that for every single fixed pair (M, S) (of which there are $\approx 2^{b+c}$) the values $u_S(f(K, M))$ as K varies are much more evenly split between 0 and 1 than would be expected for a random function. No one can reasonably expect such a property to hold for any real-world compression function, including those in MD5, SHA-1, SHA256 or SHA512.

References

[1] Anonymous, *The Bhagavad Gita*, translated by L. L. Patton, Penguin Classics, 2008.

[2] M. Bellare, New proofs for NMAC and HMAC: Security without collision-resistance, *Advances in Cryptology – Crypto '06*, LNCS 4117, Springer-Verlag, 2006, pp. 602-619; extended version available at http://cseweb.ucsd.edu/mihir/papers/hmac-new.pdf

[3] M. Bellare, R. Canetti, and H. Krawczyk, Keying hash functions for message authentication, *Advances in Cryptology – Crypto '96*, LNCS 1109, Springer-Verlag, 1996, pp. 1-15; extended version available at http://cseweb.ucsd.edu/mihir/papers/kmd5.pdf

[4] M. Bellare, R. Canetti, and H. Krawczyk, HMAC: Keyed-hashing for message authentication, Internet RFC 2104, 1997.

[5] M. Bellare and T. Kohno, A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications, *Advances in Cryptology – Eurocrypt '03*, LNCS 2656, Springer-Verlag, 2003, pp. 491-506.

[6] M. Bellare and P. Rogaway, The game-playing technique and its application to triple encryption, available at http://eprint.iacr.org/2004/331.pdf

[7] M. Bellare and P. Rogaway, Optimal asymmetric encryption – how to encrypt with RSA, *Advances in Cryptology – Eurocrypt '94*, LNCS 950, Springer-Verlag, 1994, pp. 92-111.

[8] D. Bernstein, How to stretch random functions: The security of protected counter sums, *J. Cryptology*, **12**, 1999, pp. 185-192.

[9] D. Bernstein, Floating-point arithmetic and message authentication, 2004, available from http://cr.yp.to/papers.html#hash127

[10] D. Bernstein, The Poly1305-AES message-authentication code, 2005, available from http://cr.yp.to/talks/2005.02.15/slides.pdf

[11] S. Chatterjee, A. Menezes and P. Sarkar, Another look at tightness, *Selected Areas in Cryptography — SAC 2011*, LNCS 7118, Springer-Verlag, 2012, pp. 293-319.

[12] M. Cochran, Notes on the Wang *et al.* 2^{63} SHA-1 differential path, available at http://eprint.iacr.org/2007/474.pdf

[13] I. Damgård, A design principle for hash functions, *Advances in Cryptology – Crypto '89*, LNCS 435, Springer-Verlag, 1989, pp. 416-427.

[14] T. Dierks and C. Allen, The TLS protocol, Internet RFC 2246, 1999.

[15] M. Fischlin, Security of NMAC and HMAC based on non-malleability, *Topics in Cryptology – CT-RSA 2008*, LNCS 4964, Springer-Verlag, 2008, pp. 138-154.

[16] P. Fouque, D. Pointcheval, and S. Zimmer, HMAC is a randomness extractor and applications to TLS, *Symposium on Information, Computer and Communications Security – AsiaCCS 2008*, ACM Press, 2008, pp. 21-32.

[17] D. Harkins and D. Carrel, The Internet Key Exchange (IKE), Internet RFC 2409, 1998.

[18] N. Koblitz, Another look at automated theorem-proving, *J. Mathematical Cryptology* **1**, 2007, pp. 385-403.

[19] N. Koblitz, Another look at automated theorem-proving. II, *J. Mathematical Cryptology*, **5**, 2011, pp. 205-225.

[20] N. Koblitz and A. Menezes, Another look at "provable security." II, *Progress in Cryptology – Indocrypt 2006*, LNCS 4329, Springer-Verlag, 2006, pp. 148-175.

[21] H. Krawczyk, Cryptographic extraction and key derivation: The HKDF scheme, *Advances in Cryptology – Crypto 2010*, LNCS 6223, Springer-Verlag, 2010, pp. 631-648.

[22] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[23] R. Merkle, One-way hash functions and DES, *Advances in Cryptology – Crypto '89*, LNCS 435, Springer-Verlag, 1989, pp. 428-446.

[24] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen, HOTP: An HMAC-based one time password algorithm, Internet RFC 4226, 2005.

[25] National Institute of Standards and Technology, The keyed-hash message authentication code (HMAC), FIPS Publication 198, 2002.

[26] B. Preneel and P. van Oorschot, On the security of iterated message authentication codes, *IEEE Transactions on Information Theory*, **45**, 1999, pp. 188-199.

[27] P. Rogaway, Formalizing human ignorance: Collision-resistant hashing without the keys, *Vietcrypt 2006*, LNCS 4341, Springer-Verlag, 2006, pp. 211-228.

[28] V. Shoup, Sequences of games: a tool for taming complexity in security proofs, available at http://eprint.iacr.org/2004/332.pdf

[29] J. Stern, D. Pointcheval, J. Malone-Lee, and N. Smart, Flaws in applying proof methodologies to signature schemes, *Advances in Cryptology – Crypto '02*, LNCS 2442, Springer-Verlag, 2002, pp. 93-110.

[30] X. Wang, Y. L. Yin, and H. Yu, Finding collisions in the full SHA-1, *Advances in Cryptology – Crypto '05*, LNCS 3621, Springer-Verlag, 2005, pp. 17-36.

[31] X. Wang and H. Yu, How to break MD5 and other hash functions, *Advances in Cryptology – Eurocrypt '05*, LNCS 3494, Springer-Verlag, 2005, pp. 561-576.

[32] K. Yasuda, "Sandwich" is indeed secure: How to authenticate a message with just one hashing, *Information Security and Privacy – ACISP 2007*, LNCS 4586, Springer-Verlag, 2007, pp. 355-369.

[33] K. Yasuda, Boosting Merkle-Damgård hashing for message authentication, *Advances in Cryptology – Asiacrypt 2007*, LNCS 4833, Springer-Verlag, 2007, pp. 216-231.

Department of Mathematics, Box 354350, University of Washington, Seattle, WA 98195 U.S.A.
E-mail address: koblitz@uw.edu

Department of Combinatorics & Optimization, University of Waterloo, Waterloo, Ontario N2L 3G1 Canada
E-mail address: ajmeneze@uwaterloo.ca