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^{std} and HMAC^{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 $\text{HMAC}^{\text{NIST}}$ (the FIPS version standardized by NIST), while provably secure (in the single-user setting), 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 because of this defect 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 we discuss why even this stronger result by itself fails to give convincing assurance of HMAC security.

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 [11, 24]. 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 $x_0 = \text{IV}$, and

Date: February 19, 2012; updated on April 25, 2013.

[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 $h_{\rm IV}(M) = x_m$. The function $h_{\rm IV}$ 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 $h_{\rm IV}$ 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 $H(K, M) = h_{\rm IV}(K\|M)$. (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 [23]).

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" [2] – 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: $H(K, M) = h_{\rm IV}(K_2^0\|h_{\rm IV}(K_1^0\|M)^0)$, 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 [2] 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 $h_{\rm IV}$ 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 [33, 34].[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 [1] was to restore the proof-based guarantee by giving a new proof based on assumptions that had not been invalidated by the work in [33, 34] for the hash functions that are currently in use. A second purpose of [1] 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 [1] 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 [1] 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 [2]. However, the collision-finding techniques in [33, 34] show that even this weaker property fails for the commonly-used hash functions.

standards [3] than in the definition of HMAC used in the security proof in [1]. We shall refer to the former variant as HMAC^{std} and the latter one as HMAC^{Bel}.

Theorem 2.1. *Let f be any compression function that compresses by a factor of at least 3 and satisfies the conditions in* [1] *that are needed for the proof of security for $HMAC^{\text{Bel}}$, and let f^* be the slightly modified function defined in §3 below. Then f^* satisfies the same conditions, and so $HMAC^{\text{Bel}}$ with f replaced by f^* still has the security level established by the proof. But if f is replaced by f^* in the version $HMAC^{\text{std}}$ that is in the standards* [3]*, then $HMAC^{\text{std}}$ 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 [1] if one wants the proof to apply to HMAC as defined in the standards.

2.2. **Attack on HMAC as standardized in** [26]**.** Although the keylength k in the standard [3] is the same as the taglength c, in the security proof in [2] that supports [3] 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 [26] 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 [9]). Thus, even though HMAC as standardized in [26] is "provably secure" (in the unrealistic single-user setting), 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 [2, 3, 26, 1] 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** [1]**.** 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 [29] 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 [1]. In §6 we identify the places where this occurs. Then, based on Bellare's

explanation (personal communication) that the main result in [1] is intended to be interpreted in the non-uniform model of complexity, we investigate adversaries of the compression function that exist in that model. We show that the analysis in §3.2 of [1] of the concrete security guarantee provided by the main theorem in that paper is fallacious. We describe a low-resource adversary that renders that theorem useless in practice.

2.4. **Need for stronger pseudorandomness assumption.** In §9 we argue that Pietrzak's recent attack [27] on NMAC shows the deficiencies of the usual definition of pseudorandomness in the context of an iterated hash function, which vastly magnifies the impact of even a small proportion of weak keys. It appears that any reductionist security theorem that assumes only pseudorandomness (in the usual sense) of the compression function will of necessity have such a huge tightness gap as to be worthless in practice.

2.5. **New proof.** In §10 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 [1], it is valid in the uniform model of complexity and for that reason gives a much stronger result.

But even this improved result by itself is, we believe, not nearly good enough to serve as a convincing real-world guarantee of security of HMAC. In §11 we discuss our proof from a practice-oriented standpoint and conclude that it is nearly worthless.

Stylistically, our proof resembles the 1996 security proof with collision-resistance in [2] much more than it resembles the proof without collision-resistance in [1]. 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 [2], 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 [3], 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 [1] is to extend security results from NMAC to HMAC. However, the definition of HMAC used in [1] specifies a random key of bitlength b, and so the security results apply only to this HMAC, not to the version that is implemented

[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 [34] that collisions can be found for MD5 in roughly 2^{39} operations and in [33] that collisions can be found for SHA-1 in roughly 2^{63} operations (see also [10]). However, no attack faster than a generic birthday attack has yet been found against HMAC with either MD5 or SHA-1.

in practice, for example in [3]. We next prove Theorem 2.1, a separation result that shows that the security assumption used for HMAC as defined in [1] is insufficient for the security of HMAC as defined in [3].

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) $x = \text{IV}$ (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 2.1. If $f(x, y)$ satisfies the two assumptions in [1] – namely, pseudorandomness as a function of y for a fixed hidden random x, and immunity from the appropriate related-key attack for fixed known $x = \text{IV}$ and y equal to two related keys – then we claim that $f^*(x, y)$ also satisfies these assumptions. The first property still holds because there is negligible probability 2^{-c} that $x = \text{IV}$, 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 $K \oplus \text{ipad}$ or $K \oplus \text{opad}$. 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 HMAC^{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 HMAC^{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 [3] 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 $f(\text{IV}, K^0 \oplus \text{ipad})$ and for $f(\text{IV}, K^0 \oplus \text{opad})$. 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 $\text{HMAC}^{\text{NIST}}$

The attack in this section is a special case of a type of attack described in [9] 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 [3]. 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 where 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 [1] was to restore the proof-based guarantee for HMAC that had been undermined by the work [33, 34] that found collisions in MD5 and SHA-1. However, after proving the theorem with weak collision-resistance, the authors of [2] 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 [33, 34] 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 [33, 34] 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 [33, 34] are apparently stymied.

This leads us to ask: if the theorem in [2] 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 [1] 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 [2] or [1]) a loss of tightness in the theorem in [2] that occurs if one passes to the weaker assumption. If one makes the obvious modifications in the proof in [2] 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 [1] 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 [1] 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

the sequel we are concerned with the security result for NMAC and not its extension to HMAC.

We quote a passage from [1] 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 $\|M_1\|_b \le \|M_2\|_b \le n$, 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 from a practice-oriented standpoint there is no convincing 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 [1]: 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 greatly complicate any attempt to use the theorems to derive concrete security bounds.

7. Non-uniformity

After the first version of this paper was posted, Bellare contacted us and told us that Theorem 3.3 of [1] 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. We use the term "unconstructible" to refer to such an algorithm or adversary.

The claim that Bellare's proofs are in the non-uniform model of complexity may be puzzling to readers of [1], since the paragraph titled "Techniques" in the Introduction

would lead a careful reader to conclude that Bellare is *not* using unconstructible algorithms in his paper. He writes: "We show (Lemma 3.1) that if a compression function h is a PRF then the iterated compression function h^* is *computationally almost universal* (cAU), a computational relaxation of the standard information-theoretic notion of almost-universality (AU)." But since cAU is trivially equivalent to AU in the face of unconstructible adversaries (similar to the one Bellare uses in §3.3 of [1], see above), this statement makes no sense if the paper [1] implicitly assumes Bellare's non-uniform model. Thus, a careful reader would be led to believe that Bellare's results *are* intended to be valid in the uniform model of complexity.

Moreover, in the next section we show that the analysis of his theorem that Bellare provides in §3.2 of [1] is flawed, because it implicitly assumes that his proof *is* valid in the uniform model of complexity.

When Bellare chooses to prove his theorem in the non-uniform model, a consequence is that the theorem is then based on the extremely strong assumption of prf-security even against an unconstructible adversary. That is, the adversary in the hypothesis of the theorem 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. (For a systematic discussion of issues of non-uniformity and unconstructibility in provable security reductions, see [21].)

Remark 7.1. An indication of the dramatic difference between assuming prf-security against uniform adversaries and assuming prf-security against non-uniform adversaries (in the sense in which Bellare uses the term "non-uniform") is that one can prove a separation result similar to Theorem 2.1 (proved in §3). That is, given a compression function $f(K, M)$, one can modify it slightly so as to obtain a function $f^*(K, M)$ that has the same security against uniform adversaries but no security against non-uniform ones. Here's an example of how to construct f^* when $c = 160$ and $b = 512$. Choose a random 160-bit string y_0, a random 512-bit prime p, and a random 5000-bit prime q, and set $N = pq$. Define $f^*(K, M)$ to equal y_0 for all keys K if $M|N$ and $M > 1$; otherwise define $f^*(K, M) = f(K, M)$. In the uniform model clearly $f^*(K, M)$ has the same prf-security as $f(K, M)$ unless the adversary can factor N, but in the non-uniform model it has no prf-security at all.

In §11 below, where we explain why our Theorem 10.1 does not provide convincing assurance of the real-world security of HMAC, we comment 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 [1] the prf-security assumption must be interpreted as holding against much more powerful adversaries. In order for Theorem 3.3 of [1] 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 $1/(q^2 n)$. In practice, how could one have evidence for such a bound, say in the case of MD5?

In the next section we give a description of a low-resource unconstructible adversary that is very simple but very powerful.

8. Flawed concrete security analysis

Let's turn to Bellare's analysis of tightness of his bound in §3.2 of [1]. 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 [28] shows that one cannot expect a security bound that has any content with $2^{c/2}/\sqrt{n}$ queries.

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, there exists a low-resource (unconstructible) generic adversary A_2 with advantage $2^{-c/2}$. Substituting this into inequality (4) of Theorem 3.3 of [1], 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 [1]. If, say, $n = 2^{20}$, then the claim in [1] 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 [1] 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.[5]

Remark 8.1. Alternatively, one can define a somewhat more powerful adversary A_2 as follows. For a non-empty subset S of $\{1, 2, \ldots, c\}$ 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 $\text{Prob}(u_S(f(K, M)) = t)$ 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.

9. Pietrzak's attack and the need for a stronger pseudorandomness assumption

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.[6] 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 $\leq n$.

In [27] Pietrzak gives a simple but elegant and powerful attack on the pseudorandomness of NMAC that has advantage $\Theta(nq\epsilon)$, where the underlying compression function

[5]According to [13] (comment on page 284), in complexity theory it has been well known for some time that a pseudorandom function loses at least half of its bits of security in the face of non-uniform adversaries.

[6]Recall that in this setting the advantage of the adversary is defined as follows. Let O_f be an oracle that with equal probability is either the compression function f with hidden key or else a random function. The adversary's advantage is the following difference: the conditional probability that the adversary guesses that O_f is f with hidden key given that O_f actually is f with hidden key minus the conditional probability that the adversary guesses that O_f is f with hidden key given that O_f is a random function.

f is an (ϵ, t, q)-secure pseudorandom function. We briefly describe his attack and then discuss its implications.

Let f' be an $(\epsilon/2, t, q)$-secure pseudorandom function, let $\mathcal{K}$ be a subset of keys containing $(\epsilon/2)2^c$ of the keys, and let K_0 be a fixed key in $\mathcal{K}$. Let n_0 be less than n. (Pietrzak chooses $n_0 = n - 1$, K_0 to be the zero vector, and $\mathcal{K}$ to be the keys that start with $\log(2/\epsilon)$ zeros.) Define $f(K, M) = K_0$ for all M if $K \in \mathcal{K}$ and $f(K, M) = f'(K, M)$ otherwise. One checks that f is an (ϵ, t, q)-secure pseudorandom function.

The attacker A chooses $q/2$ random n_0-block messages M^i and then makes his queries in pairs. For each M^i he asks his oracle O_{NMAC} to tag two queries $M^i \| X$ and $M^i \| Y$, where X and Y are distinct message blocks. If O_{NMAC} is NMAC with hidden keys, the adversary wins when there is a collision, which happens any time the iterated hash function hits the set $\mathcal{K}$ of bad keys as it evaluates any of the M^i. The probability of this is at least $(q/2)n_0(\epsilon/2)$, that is, $\Theta(nq\epsilon)$.

Notice that this attack is much worse than just a break in pseudorandomness of NMAC. In fact, once A produces a collision between $M^{i_0} \| X$ and $M^{i_0} \| Y$ he knows the NMAC tag of any message whose first n_0 blocks coincide with M^{i_0}.

This attack shows that if one just assumes that f is an (ϵ, t, q)-secure pseudorandom function, one cannot hope to rule out catastrophic failure of NMAC unless $\epsilon \ll 1/(qn)$. Since values such as $n = 2^{20}$ and $q = 2^{30}$ are reasonable, in practice one would have to take $\epsilon \ll 2^{-50}$. But it is no easy matter in general to get convincing evidence of pseudorandomness of real-world compression functions f; to expect to be able to make a case that a real-world f is an ϵ-secure pseudorandom function for $\epsilon \ll 2^{-50}$ is totally unrealistic.

The basic difficulty with the standard definition of the pseudorandom property is that it allows the function to collapse into triviality at ϵ proportion of the keys.

If there is to be any hope of getting a meaningful practice-oriented pseudorandomness result for NMAC, one must, unfortunately, assume a stronger pseudorandomness property for f. One has to allow the oracle's key to vary during an attack. One way to do this would be to adopt the multiple-oracle model in [4]. In that model the adversary can query any of a set of q oracles which either are all f with different hidden keys or else are all random functions. We propose a definition that is slightly different and not quite as strong.

We shall say that a compression function f is a strongly (ϵ, t, q)-secure pseudorandom function if it satisfies the usual definition of (ϵ, t, q)-security with the addition that the adversary is permitted before any query to 'reset' the oracle O_f, by which we mean that in response to the adversary's request O_f selects a new random key (if O_f is f with hidden key) or a new random function.

With this definition one can remove the q-factor in the tightness gap; we do this in the next section.

10. Uniform proof of NMAC security

Theorem 10.1. *Suppose that f is a strongly (ϵ, t, q)-secure pseudorandom function. Then NMAC is a $(2n(\epsilon + \binom{q}{2}2^{-c}), t - (qnT + Cq\log q), q, n)$-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 strongly $((\frac{\epsilon}{2n} - \binom{q}{2}2^{-c}), t + (qnT + Cq\log q), q)$-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 $fh(M) = f(K_2, \overline{h(K_1, M)})$, where $M = (M_1, \ldots, M_m)$ is an m-block message, $m \leq n$, and the second argument in f needs to be padded by $b - c$ bits (denoted by overlining, as in [2]). 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 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 strongly secure pseudorandom function (prf), then fh is a secure pseudorandom function. 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 $t + (qnT + Cq\log q)$ on the running time of one of the adversaries comes from the time required to run A_{fh}, make at most q computations of h-values, and store at most q 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 $g'(.)$. He chooses a random K_1 and presents the adversary A_{fh} with an oracle that is either $f(K_2, \overline{h(K_1, .)})$ or else $g(.)$ – that is, he 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}, he computes $\overline{h(K_1, M^i)}$, which he queries to O_f, and then gives A_{fh} the value $O_f(\overline{h(K_1, M^i)})$. Eventually (unless the simulation is imperfect, see below) 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 he says that O_f must have been f, whereas if A_{fh} said that O_{fh} is g, then he 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.[7] 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

[7]If A_{fh} fails to produce a guess about the oracle O_{fh} in time t, as can happen if the simulation is imperfect, then A_f guesses that O_f is a random function. Note that the simulation is perfect if O_f is f with hidden key.

in distinguishing fh from a random function. Otherwise, two sequences of adversaries $A_f^{(m)}$ and $B_f^{(m)}$ 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 $m-1$ blocks deleted. If a collision is produced, then with a certain probability O_f must be $f(K_2, .)$; however, one must also account for the possibility that O_f is $g'(.)$, and in the case of $A_f^{(m)}$ this brings in the next adversary in the sequence $A_f^{(m+1)}$.

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.[8] 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 sequences of adversaries $A_f^{(m)}$ and $B_f^{(m)}$ 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^i_ℓ to denote its ℓ-th block, we let $M^{i,[m]} = (M^i_1, \ldots, M^i_m)$ be the truncation after the m-th block, and we set $M^{i,(m)} = (M^i_m, M^i_{m+1}, \ldots)$, that is, $M^{i,(m)}$ is the message with the first $m-1$ blocks deleted. We say that a message M is "non-empty" and write $M \neq \emptyset$ if its block length is at least 1.

We define α_0 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:

there exist i and j, $j < i$, such that $h(K_1, M^i) = h(K_1, M^j)$.

For $m \geq 1$ we define α_m to be the probability, taken over all coin tosses of A_{fh} and all q-tuples of keys $(K_1, K_2, \ldots, K_q)$, that the sequence of A_{fh}-queries M^i satisfies the following property:

(1_m) there exist i and j, $j < i$, such that $M^{i,(m+1)} \neq \emptyset$, $M^{j,(m+1)} \neq \emptyset$,

$$h(K_{\ell_i}, M^{i,(m+1)}) = h(K_{\ell_j}, M^{j,(m+1)}),$$

where for any index i for which $M^{i,(m+1)} \neq \emptyset$ we let ℓ_i denote the smallest index for which $M^{\ell_i,(m+1)} \neq \emptyset$ and $M^{i,[m]} = M^{\ell_i,[m]}$.

Finally, for $m \geq 1$ we define β_m to be the probability, taken over all coin tosses of A_{fh} and all q-tuples of keys $(K_1, K_2, \ldots, K_q)$, that the sequence of A_{fh}-queries M^i satisfies the following property:

[8]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.

(2_m) there exist i and j such that $M^{i,(m+1)} = \emptyset$, $M^{j,(m+1)} \neq \emptyset$,

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

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, she[9] queries O_f with the first block M_1^i of each A_{fh}-query M^i. If $M^{i,(2)}$ is non-empty, she then computes $y_i = h(O_f(M_1^i), M^{i,(2)})$; if $M^{i,(2)}$ is empty, she 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, she looks for a collision with the y_j-values obtained from earlier queries M^j. If a collision occurs, she guesses that O_f is f with hidden key; if not, she guesses that O_f is $g'(.)$.

It is, of course, conceivable that even when O_f is $g'(.)$ there is a collision $h(L_i, M^{i,(2)}) = h(L_j, M^{j,(2)})$ with $M^{i,(2)}$ and $M^{j,(2)}$ non-empty. 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). Recall that the probability that this occurs is α_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. If both are empty, then the probability that $L_i = L_j$ is 2^{-c}. If, say, $M^{j,(2)}$ is non-empty, then in the case $M_1^i \neq M_1^j$ we again have probability 2^{-c} that $L_i = h(L_j, M^{j,(2)})$, whereas in the case $M_1^i = M_1^j$ we have (2_1) with $K_i = L_i$.

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 next describe the sequence of adversaries $A_f^{(m)}$, $m \geq 2$. Let O_f again denote the prf-test oracle for f that $A_f^{(m)}$ can query. Like A'_f, he runs A_{fh} once and gives random responses to its queries. As A_{fh} makes queries, he sorts their prefixes (where we are using the word "prefix" to denote the first $m-1$ blocks of a query that has block-length at least m). If the i-th query has block-length at least m and if its prefix coincides with that of an earlier query, he records the index ℓ_i of the first query that has the same prefix; if it has a different prefix from earlier queries he sets $\ell_i = i$. After running A_{fh}, he goes back to the first query M^{j_1} that has block-length at least m and for all i for which $\ell_i = j_1$ (that is, for all queries that have the same prefix as M^{j_1}) 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 $y_i = O_f(M_m^i)$. Then he resets O_f and goes to the first j_2 such that M^{j_2} has block-length at least m and a different prefix from M^{j_1}. For all i for which $\ell_i = j_2$ 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 $y_i = O_f(M_m^i)$. He continues in this way until he's gone

[9]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.

through all the queries. He then looks for two indices $j < i$ such that $y_j = y_i$. If he finds a collision, he guesses that O_f is f with hidden key; otherwise, he guesses that it is a random function.

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 he guesses that O_f is f with hidden key. The possibility that O_f is really $g'(.)$ is due to two conceivable circumstances – a collision of the form (1_m) or a collision among random values (either a collision between two random values L_i and L_j or between L_i and $h(L_j, M^{j,(m+1)})$, or else a collision of the form (2_m) with $K_i = L_i$ — here the probability of such a collision is bounded by $\binom{q}{2}2^{-c}$ and by β_m, respectively).

Finally, the sequence of adversaries $B_f^{(m)}$, $m \geq 1$, is defined as follows. As usual, O_f denotes the prf-test oracle for f that $B_f^{(m)}$ can query. She runs A_{fh} once and gives random responses to its queries. As A_{fh} makes queries, she sorts their prefixes (where this time we are using the word "prefix" to denote the first m blocks of a query that has block-length at least m). She makes up a list of pairs $(i, S(i))$, where the i-th query has block-length exactly m and coincides with the prefix of at least one other query; in that case $S(i)$ denotes the set of indices $j \neq i$ such that $M^{j,[m]} = M^i$. After running A_{fh}, she chooses a message-block Y that is different from all the blocks M^j_{m+1} of all queries M^j. She goes through all indices i with non-empty $S(i)$. For each such i she queries Y to O_f and for each $j \in S(i)$ she also queries M^j_{m+1} to O_f and computes $y_j = h(O_f(M^j_{m+1}), M^{j,(m+2)}, Y)$. She looks for a collision between $O_f(Y)$ and y_j for $j \in S(i)$. Before going to the next i she resets O_f. If she finds a collision for any of the i, she guesses that O_f is f with hidden key; otherwise, she guesses that it is a random function. The advantage of this adversary is at least $\beta_m - q2^{-c}$, because if O_f is $f(K_i, .)$ and $h(K_i, M^{j,(m+1)}) = K_i$, then $h(O_f(M^j_{m+1}), M^{j,(m+2)}, Y) = f(K_i, Y) = O_f(Y)$, whereas if O_f is a random function, then $O_f(Y)$ has probability only 2^{-c} of coinciding with this h-value.

We thus have the following lower bounds for the advantages of the adversaries:

A_f: $\epsilon - \alpha_0$;

A'_f: $\alpha_0 - \alpha_1 - \beta_1 - \binom{q}{2}2^{-c}$;

$A_f^{(m)}$, $m \geq 2$: $\alpha_{m-1} - \alpha_m - \beta_m - \binom{q}{2}2^{-c}$;

$B_f^{(m)}$, $m \geq 1$: $\beta_m - q2^{-c}$.

Trivially we have $\alpha_n = \beta_n = 0$, and so the adversaries go no farther than $A_f^{(n)}$ and $B_f^{(n-1)}$. The sum of all the advantages of the $2n$ adversaries telescopes and is at least $\epsilon - (2n-1)\binom{q}{2}2^{-c}$.

Since we have no way of knowing which of these adversaries has the greatest advantage, we make a random selection. That is, the adversary A we use to attack the pseudorandomness of f consists of randomly choosing one of the $2n$ adversaries A_f, A'_f, $A_f^{(m)}$ $(2 \leq m \leq n)$, $B_f^{(m)}$ $(1 \leq m \leq n-1)$ and running it. The advantage of the adversary A is the expectation obtained by summing the advantages of the $2n$ adversaries with each one weighted by the probability $1/(2n)$ that we choose the corresponding adversary. This advantage is at least $\frac{1}{2n}(\epsilon - (2n-1)\binom{q}{2}2^{-c})) > (\frac{\epsilon}{2n} - \binom{q}{2}2^{-c})$, as claimed. □

Remark 10.2. In Theorem 2.1 the term $2n(\epsilon+\binom{q}{2}2^{-c})$ can be replaced by $2n\epsilon$, because there is a generic adversary $A_{\text{strong prf}}$ of the strong pseudorandomness property that has advantage of order $(qt/T)2^{-c}$; since one must have $t > qnT$ in order for the theorem to have content, this gives $\epsilon \gg q^2 2^{-c}$. The adversary $A_{\text{strong prf}}$ is constructed as follows. $A_{\text{strong prf}}$ chooses two random message blocks M^1 and M^2 and queries O_f with those messages $q/2$ times, resetting the oracle each time. He then guesses $(t-qT)/2T$ random keys K_i and computes $f(K_i, M^1)$ and $f(K_i, M^2)$ for each key. He looks for a collision with the pair of values $O_f(M^1),\ O_f(M^2)$ for one of the oracle settings. If O_f is f with hidden key, there is a probability roughly $(qt/4T)2^{-c}$ that such a collision will occur, whereas if O_f is a random function that probability is negligible.

Lemma 3.3 of [4] implies that if f is an (ϵ, t, q)-secure pseudorandom function, then f is strongly $(q\epsilon, t - C'(qT + q\log q), q)$ secure for some absolute constant C'. Combined with Theorem 10.1, we obtain:

Corollary 10.3. *Suppose that f is an (ϵ, t, q)-secure pseudorandom function. Then NMAC is a $(2n(q\epsilon + \binom{q}{2}2^{-c}), t - (qnT + Cq\log q), q, n)$-secure pseudorandom function. Here C is an absolute constant and T denotes the time for one evaluation of f.*

Remark 10.4. In the statement of Corollary 10.3 the term $2n(q\epsilon + \binom{q}{2}2^{-c})$ can be replaced simply by $2nq\epsilon$. To see this, recall that the generic key-guessing attack on the pseudorandom property of a compression function has advantage roughly $t2^{-c}$. Since $t > qn$ for Corollary 10.3 to have content, one always has $\epsilon \gg q2^{-c}$.

Remark 10.5. As far as we know, our proof of Corollary 10.3 is the first uniform proof of $O(nq\epsilon)$-security of NMAC assuming ϵ-security of f in the usual sense. Of course, in view of the huge tightness gap of order nq one can justifiably respond to our claim by saying "Who cares?"

11. Interpretations

How useful is the bound in Theorem 10.1 as a guarantee of real-world security? In the case of Corollary 10.3 the answer to the analogous question is clearly "not much," because of the nq tightness gap, which in practice might be greater than 2^{50}. Thus, it is interesting to ask whether or not Theorem 10.1 is equivalent to Corollary 10.3, which would be the case if the converse of Lemma 3.3 of [4] is true, i.e., if strong $(q\epsilon, t, q)$-security of f implies (ϵ, t', q)-security of f in the usual sense (where t' has comparable magnitude to t). We do not know the answer to this question. On the one hand, the key-guessing adversary against the pseudorandom property corresponds to a strong key-guessing adversary $A_{\text{strong prf}}$ that has roughly q times the advantage. On the other hand, $A_{\text{strong prf}}$ needs $q/2$ times as many queries as the former key-guessing adversary. In general, given a prf-adversary A_{prf}, we do not see how to construct from it a strong prf-adversary $A_{\text{strong prf}}$ with magnified advantage but with the same number of queries as A_{prf}. This allows us to conjecture (or at least hope) that the converse of Lemma 3.3 of [4] is false, in which case Theorem 10.1 is a stronger result than Corollary 10.3.

Even if the "true" tightness gap in Theorem 10.1 is only n (rather than nq in disguise, as it would be if Theorem 10.1 is equivalent to Corollary 10.3), one can still question the usefulness of the theorem. Since values such as 2^{20} and 2^{30} are reasonable for the block-length bound, the tightness gap can be quite significant.

A second limitation of Theorem 10.1 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 10.1 has a very strong hypothesis – strong pseudorandomness of the compression function.[10] 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 10.1 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, say, 2^{50} queries might be reasonable. Moreover, as pointed out in [3], 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 [12, 17, 22] and one-time passwords [25]. In those settings one often needs a much greater level of assurance than anything that's provided by theoretical results such as our Theorem 10.1.

12. 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 [26] – 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 [1] – or even from our proof in §10 – 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 [7] and appeared in a somewhat different form in [1], 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.

[10] Note that Theorem 2 needs the compression function to be strongly (ϵ, t, q)-secure for a rather small value of ϵ, since the theorem loses content if $\epsilon > 1/(2n)$.

In addition to the proofs in [1], other questionable uses of coin-fixing arguments can be found in [35] (Lemma 1), [36] (Theorem 1), [37], [6] (Theorem 2), and [14].

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 [1] which, in turn, was derived using a coin-fixing argument.[11]

6. A theorem that has an unstated strong assumption or is accompanied by a fallacious interpretation could also mislead practitioners if it is used as a basis for recommendations. For example, Rogaway's 2011 evaluation of block ciphers [30] for the Government of Japan contains an erroneous statement of the security assurance provided by the main theorem about NMAC security in [1]. §9.4 of Rogaway's report repeats the fallacious argument of Bellare that would have been valid if his proof had been in the uniform model.

7. Game-hopping proofs [7, 31] 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.

8. In [32] Stern, Pointcheval, Malone-Lee, and Smart comment (in connection with the error in the original security proof for OAEP [8]) 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 wish to thank Ronald Cramer, the handling editor for *Designs, Codes, and Cryptography*, for the time he devoted to our paper, undoubtedly much more than the average for submissions. He was the intermediary for correspondence with one (or possibly two) anonymous referees who contributed in two ways to the proof of Theorem 10.1: by finding a gap in the argument concerning the $A_f^{(m)}$ sequence of adversaries, which we fixed by introducing the $B_f^{(m)}$ adversaries; and by pointing out a more efficient argument in the last step that enabled us to improve the result and eliminate an unnecessary factor in the tightness gap. We also wish to thank Krzysztof Pietrzak for sending us a note

[11]The authors of [16] were unaware that Bellare's proof in [1] is invalid in the uniform model of complexity, according to a personal communication from Pointcheval to the second author on 16 April 2012.

[27] describing his attack before it was posted and informing us of the error in an earlier statement of Theorem 10.1, which we mistakenly thought we had proved assuming ϵ-security rather than strong ϵ-security of the compression function.

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

References

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

[2] 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.

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

[4] M. Bellare, R. Canetti, and H. Krawczyk, Pseudorandom functions revisited: The cascade construction and its concerete security, *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, 1996, pp. 514-523; extended version available at http://cseweb.ucsd.edu/users/mihir/papers/cascade.pdf.

[5] M. Bellare and T. Kohno, A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications, *Advances in Cryptology – Eurocrypt 2003*, LNCS 2656, Springer-Verlag, 2003, pp. 491-506.

[6] M. Bellare and T. Ristenpart, Multi-property-preserving hash domain extension and the EMD transform, *Advances in Cryptology – Asiacrypt 2006*, LNCS 4824, Springer-Verlag, 2006, pp. 299-314.

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

[8] 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.

[9] 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.

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

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

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

[13] Y. Dodis and J. Steinberger, Message authentication codes from unpredictable block ciphers, *Advances in Cryptology – Crypto 2009*, LNCS 5677, Springer-Verlag, 2009, pp. 267-285.

[14] S. Dziembowski and K. Pietrzak, Leakage-resilient cryptography, *Proceedings of the 49th Annual IEEE Symposium on the Foundations of Computer Science*, 2008, pp. 293-302.

[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] N. Koblitz and A. Menezes, Another look at non-uniformity, available at http://anotherlook.ca.
[22] H. Krawczyk, Cryptographic extraction and key derivation: The HKDF scheme, *Advances in Cryptology – Crypto 2010*, LNCS 6223, Springer-Verlag, 2010, pp. 631-648.
[23] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.
[24] R. Merkle, One-way hash functions and DES, *Advances in Cryptology – Crypto '89*, LNCS 435, Springer-Verlag, 1989, pp. 428-446.
[25] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen, HOTP: An HMAC-based one time password algorithm, Internet RFC 4226, 2005.
[26] National Institute of Standards and Technology, The keyed-hash message authentication code (HMAC), FIPS Publication 198, 2002.
[27] K. Pietrzak, A closer look at HMAC, personal communication, 1 April 2013; available at http://eprint.iacr.org/2013/212.
[28] B. Preneel and P. van Oorschot, On the security of iterated message authentication codes, *IEEE Transactions on Information Theory*, **45**, 1999, pp. 188-199.
[29] P. Rogaway, Formalizing human ignorance: Collision-resistant hashing without the keys, *Vietcrypt 2006*, LNCS 4341, Springer-Verlag, 2006, pp. 211-228.
[30] P. Rogaway, Evaluation of some blockcipher modes of operation: Evaluation carried out for the Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan, 10 February 2011, available at http://www.cs.ucdavis.edu/~rogaway/papers/modes-cryptrec.pdf.
[31] V. Shoup, Sequences of games: a tool for taming complexity in security proofs, available at http://eprint.iacr.org/2004/332.
[32] J. Stern, D. Pointcheval, J. Malone-Lee, and N. Smart, Flaws in applying proof methodologies to signature schemes, *Advances in Cryptology – Crypto 2002*, LNCS 2442, Springer-Verlag, 2002, pp. 93-110.
[33] X. Wang, Y. L. Yin, and H. Yu, Finding collisions in the full SHA-1, *Advances in Cryptology – Crypto 2005*, LNCS 3621, Springer-Verlag, 2005, pp. 17-36.
[34] X. Wang and H. Yu, How to break MD5 and other hash functions, *Advances in Cryptology – Eurocrypt 2005*, LNCS 3494, Springer-Verlag, 2005, pp. 561-576.
[35] 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.
[36] K. Yasuda, Boosting Merkle-Damgård hashing for message authentication, *Advances in Cryptology – Asiacrypt 2007*, LNCS 4833, Springer-Verlag, 2007, pp. 216-231.
[37] K. Yasuda, Multilane HMAC – Security beyond the birthday limit, *Progress in Cryptology – Indocrypt 2007*, LNCS 4859, Springer-Verlag, 2007, pp. 18-32.

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