Reading Guide 2:
Keying Hash Functions for Message Authentication

Hyrum Mills, Chris Soghoian, Jon Stone, Malene Wang

September 10, 2004

1 Summary

The paper presented this week is Bellare, Canetti, and Krawczyk's paper, *Keying Hash Functions For Message Authentication*, in which they introduce NMAC, a MAC scheme based on cryptographic hash functions such as MD5 or SHA-1, and HMAC, a special variant of NMAC. In addition to describing the algorithms, they also provide quantitative analyses of the security of these schemes, descriptions of their advantages, the applicability of well-known cryptanalytic attacks, and comparisons to other existing keyed hash function MAC schemes.

Section 1 is a brief overview of some of the technical aspects of the paper, and the security implications, performance issues, and advantages of the NMAC/HMAC schemes. In reading the paper, it may be helpful to skim section 1 first, and come back to it after reading sections 2-5, where the technical terminology and the inner workings of NMAC/HMAC are explained. Portions of the paper also make references to various attacks, which are described in section 6 and also the Attacks section of this reading guide.

Section 1.1 briefly describes the purpose of MACs and how to use them. Section 1.2 discusses why one would use cryptographic hash functions to build a MAC, and some of benefits and obstacles to doing so. Section 1.3 presents the key advantages to the NMAC/HMAC schemes. In particular, the authors emphasize that the security of NMAC/HMAC are tightly coupled to the security of the underlying hash function, and also the black-box usage of hash functions. Section 1.4 begins to get more technical and may or may not confuse the reader, depending on the reader's background. Those who do not understand it immediately should not be too worried; come back to this part after reading sections 2-5 thoroughly, and it will serve as a nice recap. See the definitions section of this guide for a definition of "provably secure". All in all, section 1 is fairly straightforward, as long as the reader has read and understood sections 2-5.

Section 2.1 describes MACs in general. Make sure to understand exactly what is transmitted between two parties using a MAC scheme, what information the adversary can glean from these transmissions to utilize in her attacks, and what known/chosen message attacks are. Also pay special attention to the definition of "(ϵ, t, q, L)-security." This paper uses a_i to denote authentication tags, but bear in mind that the convention is to use t_i.

Section 2.2 deals with cryptographic hash functions. Note the security properties of hash functions, especially the definition of "collision resistance," and also the randomness-like properties. It is important to understand how iterated hash constructions work, including its formal mathematical definition; Figure 2 is a good graphical representation. Pay attention to the difference between compression functions, denoted by f, versus hash functions F, which are generated by iterating the compression function multiple times on certain inputs. The paper says "a b-bit value IV is fixed;" this is a typo, the IV is actually l-bits. The paper also uses commas to indicate concatenation and to separate inputs to a function, which is confusing, so bear in mind that commas used in conjunction with hash functions Fs (e.g. $F(k, x)$) denote concatenation, while those used with compression functions fs (e.g. $f(k, x)$) denote separate inputs.

Section 3 presents keyed hash functions, where the goal is to introduce a secret key into the hash function somehow in order to make it usable as a MAC between parties possessing the key. Specifically, the approach is to use the key as the IV. Paragraph 3 of this section gives a formal mathematical definition of keyed hash functions. Note that this definition is the same as that of iterated hash functions in general (section 2.2). Pay special attention to Definition 3.1 (seem familiar?), and the implication that the security of hash functions is based on their collision-resistance properties.

Section 4 is where NMAC is officially introduced and defined. The formal definition is, of course, important. Make note of the fact when the outer function F_{k_1} is iterated only once, as it is in the NMAC construction, it is equivalent to the the compression function f_{k_1} acting on $\overline{F_{k_2}(x_i)}$, which is the inner hash function padded to a full block size (signified by the bar symbol). In other words, $F_{k_1}(F_{k_2}(x)) = f_{k_1}(\overline{F_{k_2}(x)})$. The padding is necessary because the output of F_{k_2} is only l bits and the compression function f requires b bit inputs.

Theorem 4.1 and its proof are crucial to understanding how and why NMAC is secure. When going through the proof, keep in mind that it is requiring that the keyed compression function f_{k_1} is a MAC on b-bit blocks in and of

itself, which is why Theorem 4.1 describes f as an "(ϵ_f, q, t, b)-secure MAC on messages of b bits." In the proof, the "messages" that f is MACing are the padded results of the inner hash function, $\overline{F_{k_2}(x_i)}$ (not the individual blocks x_i of the message x itself). In contrast, F_{k_2} is simply a collision-resistant hash function that hashes the message x of length L. To summarize, the full keyed hash function F must be collision resistant and the keyed compression function f must be a secure MAC on messages of b bits. Assuming both of these conditions hold, the proof shows that the NMAC construction is a secure MAC on messages of any length.

If at first you don't comprehend the proof, (try^n) again (where $n > 1$). We will go through the proof the Thursday to try and clarify the confusing parts. Remarks 4.2-9 are fairly interesting and straightforward. All of them, with the exception of 4.4 and 4.5, can be understood without first understanding the proof. In Remark 4.6, DES-MAC-CBC is an example of a MAC that is built out of a block cipher. The point is that any MAC can be used, not only things that are based on hash functions.

Having explained the operation of NMAC and its security, section 5 now turns to HMAC, a fixed IV variant of NMAC. Make sure to understand how HMAC relates to NMAC and why it can be considered a variant (it's all in how the secret keys are defined). The paper mentions that $k_1 = f(\overline{k} \oplus opad)$, and $k_2 = f(\overline{k} \oplus ipad)$, but it would be better to express them as $k_1 = f(IV, \overline{k} \oplus opad)$, and $k_2 = f(IV, \overline{k} \oplus ipad)$. With respect to the security of HMAC, understand why it is weaker in theory, but usually inconsequential in practice. Also note the reasons for choosing those specific values of *opad* and *ipad* (see Definitions for "Hamming distance"). Last but not least, note the truly black-box usage of hash functions, which is primary advantage of HMAC over NMAC.

Section 6 lists various well-known cryptanalytical attacks on MAC schemes and two other works, why these attacks are not applicable, and how NMAC/HMAC is better than those other works. For a better understanding of these attacks, see the Attacks section of this guide. In particular, notice that the part on extension attacks explains why the outer function is needed to make NMAC secure.

2 Definitions

A *hash function* is a function that reduces a message of any length to an output message of a fixed size. The output message is called the *hash value*. A one-way hash function is both *preimage* and *second preimage* resistant:

- a hash function is *preimage resistant* if it is computationally infeasible to find any input which corresponds to a particular output (i.e. given $g(x)$ it is difficult to compute x)
- a hash function is *second preimage resistant* if it is infeasible to find any second input which hashes to the same output as any specified input. (i.e. given x it is difficult to compute a $y \neq x$ such that $f(y) = f(x)$)

Finally, a hash function is *collision resistant* if it is computationally infeasible to find two distinct inputs that hash to the same result.

A *message authentication code* (MAC) is used for ascertaining message integrity and authenticating the source. The message sender creates the MAC by running a function which takes both a key and the message body, using a key known previously by both sender and receiver. This MAC is then appended to the outgoing message. The recipient generates the message's MAC again, using the same shared key and message, and compares the newly generated MAC to the original received from the sender. If the two are identical, the sender must have the correct key, and the message has not been altered in transit.

A scheme is *provably secure* if its security can be tied closely to the security of an atomic primitive. NMAC and HMAC are provably secure because the two schemes are solely dependent upon the security attributes of the underlying hash functions. While failure in the security of the underlying functions would simply require the replacement of the hash in NMAC/HMAC with a better one, the hashes are very widely used in other contexts and a failure would in that sense be catastrophic.

Hamming distances measure the number of bits that differ between two n-bit binary strings. For instance, the two strings 10011010 and 00101101 have a Hamming distance of 6.

3 Attacks

The paper refers to several different types of attacks — collision attacks, birthday attacks, extension attacks, and divide and conquer attacks. A *collision attack* finds two messages with the same hash, but the attacker doesn't necessarily get to pick the form of the messages. For example, an attacker might be able to find two meaningless messages that collide, but be unable to find any two messages that contain English text which do.

The *birthday attack* derives it's name and background from what is known as the 'birthday paradox,' a probabalistic statement which says that if 23 people are in a group then there is slightly more than a 50% chance that at least two of them will have the same birthday. For 60 or more people, the probablility is greater than 99%. The 'birthday paradox' is not really a paradox at all; it simply contradicts common intuition.

The attack uses the mathematics underlying the birthday paradox in order to minimize both the time and space required to run the attack. The math states that if a given function returns any of n unique outputs with equal probability and n is a large enough value, then after running $1.2\sqrt{n}$ different values through the function we can expect to have found a pair of arguments x_1 and x_2 where $x_1 \neq x_2$ but $f(x_1) = f(x_2)$ – known as a collision. If the function range is unevenly distributed, a collision may occur even faster.

The premise behind an *extension attack* is directly addressed by the authors, however it is worthwhile to describe it briefly. Imagine a keyed iterated hash functioned where the key is simply prepended to the message, i.e. $F_k(x) = F(k||x)$. In the case of a message that doesn't require padding, since the hash at any given stage of such a process contains the hash of all the text of the message until that point, the end hash of a legitimate message can be considered to be just another stage in the function. New text can be added to the message, the original hash and new text run through the hash function, and a new legitimate hash would result.

For example, the message 'Send the check' may be sent with valid MAC `0xabc`. However, by using an extension attack the attacker may add the phrase 'to Hyrum Mills' and start the hash function as if it had just left off, giving it `0xabc` as the IV and the extension text. Given that the shared key was hashed in at the beginning of the original message and the original MAC was valid, the function will generate an accurate MAC – and the check will be sent to the wrong place.

The general concept of a *divide and conquer attack* is to solve several easier problems rather than attacking the entire construction directly. In constructions that use n keys of length l, this is exemplified by attacks that can search for each key individually ($O(l \cdot 2^n)$) rather than having to search for a set of keys that work simultaneously ($O(2^{ln})$).