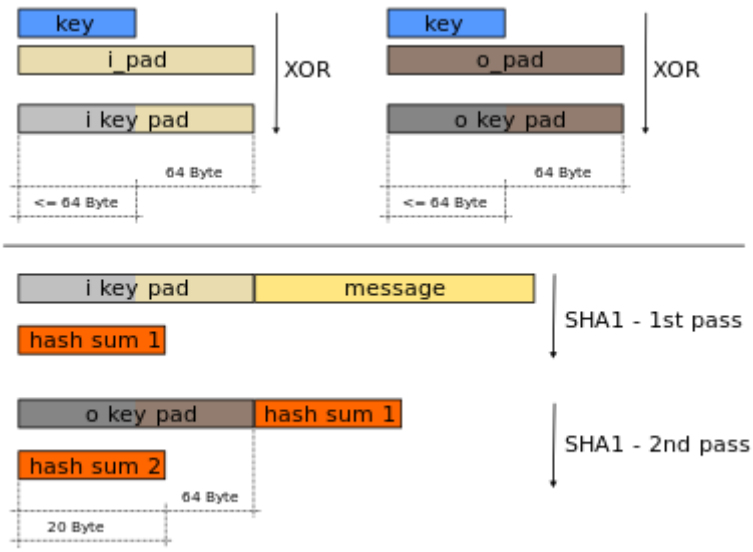
HMAC

In cryptography, an **HMAC** (sometimes expanded as either **keyed-hash message authentication code** or **hash-based message authentication code**) is a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key. It may be used to simultaneously verify both the *data integrity* and the *authentication* of a message, as with any MAC. Any cryptographic hash function, such as SHA-256 or SHA-3, may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-X, where X is the hash function used (e.g. HMAC-SHA256 or HMAC-SHA3). The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and the size and quality of the key



HMAC-SHA1 generation

HMAC uses two passes of hash computation. The secret key is first used to derive two keys – inner and outer. The first pass of the algorithm produces an internal hash derived from the message and the inner key. The second pass produces the final HMAC code derived from the inner hash result and the outer key. Thus the algorithm provides better immunity against length extension attacks.

An iterative hash function breaks up a message into blocks of a fixed size and iterates over them with a compression function. For example, SHA-256 operates on 512-bit blocks. The size of the output of HMAC is the same as that of the underlying hash function (e.g., 256 and 1600 bits in the case of SHA-256 and SHA-3, respectively), although it can be truncated if desired.

HMAC does not encrypt the message. Instead, the message (encrypted or not) must be sent alongside the HMAC hash. Parties with the secret key will hash the message again themselves, and if it is authentic, the received and computed hashes will match.

The definition and analysis of the HMAC construction was first published in 1996 in a paper by Mihir Bellare, Ran Canetti, and Hugo Krawczyk,[1] and they also wrote RFC 2104 in 1997. The 1996 paper also defined a variant called NMAC. FIPS PUB 198 generalizes and standardizes the use of HMACs. HMAC is used within the IPsec and TLS protocols and for JSON Web Tokens.

Contents

Definition

This definition is taken from RFC 2104:

$$\mathrm{HMAC}(K, m) = \mathrm{H}\left(\left(K' \oplus opad\right) \parallel \mathrm{H}\left(\left(K' \oplus ipad\right) \parallel m\right)\right)$$

$$K' = \begin{cases} \mathrm{H}(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

where

H is a cryptographic hash function
m is the message to be authenticated
K is the secret key
K' is a block-sized key derived from the secret key, *K*; either by padding to the right with 0s up to the block size, or by hashing down to less than the block size first and then padding to the right with zeros
∥ denotes concatenation
⊕ denotes bitwise exclusive or (XOR)
opad is the block-sized outer padding, consisting of repeated bytes valued 0x5c
ipad is the block-sized inner padding, consisting of repeated bytes valued 0x36

Implementation

The following pseudocode demonstrates how HMAC may be implemented. Blocksize is 64 (bytes) when using one of the following hash functions: SHA-1, MD5, RIPEMD-128/160.[2]

```
Function hmac
   Inputs:
      key:        Bytes      //array of bytes
      message:    Bytes      //array of bytes to be hashed
      hash:       Function   //the hash function to use (e.g. SHA-1)
      blockSize:  Integer    //the block size of the underlying hash function (e.g. 64 bytes for SHA-1)
      outputSize: Integer    //the output size of the underlying hash function (e.g. 20 bytes for SHA-1)

   //Keys longer than blockSize are shortened by hashing them
   if (length(key) > blockSize)  then
      key ← hash(key)  //Key becomes outputSize bytes long

   //Keys shorter than blockSize are padded to blockSize by padding with zeros on the right
   if (length(key) < blockSize)  then
      key ← Pad(key, blockSize)  //pad key with zeros to make it  blockSize bytes long

   o_key_pad = key xor [0x5c * blockSize]     //Outer padded key
   i_key_pad = key xor [0x36 * blockSize]     //Inner padded key

   return hash(o_key_pad ∥ hash(i_key_pad ∥ message)) //Where ∥ is concatenation
```

Design principles

The design of the HMAC specification was motivated by the existence of attacks on more trivial mechanisms for combining a key with a hash function. For example, one might assume the same security that HMAC provides could be achieved with MAC = **H**(*key* || *message*). However, this method suffers from a serious flaw: with most hash functions, it is easy to append data to the message without knowing the key and obtain another valid MAC ("length-extension attack"). The alternative, appending the key using MAC = **H**(*message* || *key*), suffers from the problem that an attacker who can find a collision in the (unkeyed) hash function has a collision in the MAC (as two messages m1 and m2 yielding the same hash will provide the same start condition to the hash function before the appended key is hashed, hence the final hash will be the same). Using MAC = **H**(*key* || *message* || *key*) is better, but various security papers have suggested vulnerabilities with this approach, even when two different keys are used.[1][3][4]

No known extension attacks have been found against the current HMAC specification which is defined as **H**(*key* || **H**(*key* || *message*)) because the outer application of the hash function masks the intermediate result of the internal hash. The values of *ipad* and *opad* are not critical to the security of the algorithm, but were defined in such a way to have a large Hamming distance from each other and so the inner and outer keys will have fewer bits in common. The security reduction of HMAC does require them to be different in at least one bit.

The Keccak hash function, that was selected by NIST as the SHA-3 competition winner, doesn't need this nested approach and can be used to generate a MAC by simply prepending the key to the message, as it is not susceptible to length-extension attacks.[5]

Security

The cryptographic strength of the HMAC depends upon the size of the secret key that is used. The most common attack against HMACs is brute force to uncover the secret key. HMACs are substantially less affected by collisions than their underlying hashing algorithms alone.[6][7] In particular, in 2006 Mihir Bellare proved that HMAC is a PRF under the sole assumption that the compression function is a PRF.[8] Therefore, HMAC-MD5 does not suffer from the same weaknesses that have been found in MD5.

In 2006, Jongsung Kim, Alex Biryukov, Bart Preneel, and Seokhie Hong showed how to distinguish HMAC with reduced versions of MD5 and SHA-1 or full versions of HAVAL, MD4, and SHA-0 from a random function or HMAC with a random function. Differential distinguishers allow an attacker to devise a forgery attack on HMAC. Furthermore, differential and rectangle distinguishers can lead to second-preimage attacks. HMAC with the full version of MD4 can be forged with this knowledge. These attacks do not contradict the security proof of HMAC, but provide insight into HMAC based on existing cryptographic hash functions.[9]

In 2009, Xiaoyun Wang et al. presented a distinguishing attack on HMAC-MD5 without using related keys. It can distinguish an instantiation of HMAC with MD5 from an instantiation with a random function with 2^{97} queries with probability 0.87.[10]

In 2011 an informational RFC 6151[11] was published to summarize security considerations in MD5 and HMAC-MD5. For HMAC-MD5 the RFC summarizes that – although the security of the MD5 hash function itself is severely compromised – the currently known *"attacks on HMAC-MD5 do not seem to indicate a practical vulnerability when used as a message authentication code"*, but it also adds that *"for a new protocol design, a ciphersuite with HMAC-MD5 should not be included"*.

In May 2011, RFC 6234 was published detailing the abstract theory and source code for SHA-based HMACS.

Examples

Here are some empty HMAC values:

```
HMAC_MD5("", "")    = 74e6f7298a9c2d168935f58c001bad88
HMAC_SHA1("", "")   = fbdb1d1b18aa6c08324b7d64b71fb76370690e1d
HMAC_SHA256("", "") = b613679a0814d9ec772f95d778c35fc5ff1697c493715653c6c712144292c5ad
```

Here are some non-empty HMAC values, assuming 8-bit ASCII or UTF-8 encoding:

```
HMAC_MD5("key", "The quick brown fox jumps over the lazy dog")    = 80070713463e7749b90c2dc24911e275
HMAC_SHA1("key", "The quick brown fox jumps over the lazy dog")   =
de7c9b85b8b78aa6bc8a7a36f70a90701c9db4d9
HMAC_SHA256("key", "The quick brown fox jumps over the lazy dog") =
f7bc83f430538424b13298e6aa6fb143ef4d59a14946175997479dbc2d1a3cd8
```

References

1. Bellare, Mihir; Canetti, Ran; Krawczyk, Hugo (1996). "Keying Hash Functions for Message Authentication" (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.134.8430)

2. "Definition of HMAC" (https://tools.ietf.org/html/rfc2104#section-2) *HMAC: Keyed-Hashing for Message Authentication* (https://tools.ietf.org/html/rfc2104) sec. 2. doi:10.17487/RFC2104 (https://doi.org/10.17487%2FRFC2104). RFC 2104.
3. Preneel, Bart; van Oorschot, Paul C. (1995). "MDx-MAC and Building Fast MACs from Hash Functions" (https://web.archive.org/web/20100604234324/http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.3855) Archived from the original (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.3855) on 4 June 2010. Retrieved 28 August 2009.
4. Preneel, Bart; van Oorschot, Paul C. (1995). "On the Security of Two MAC Algorithms" (https://web.archive.org/web/20090223134840/http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.8908) Archived from the original (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.8908) on 23 February 2009. Retrieved 28 August 2009.
5. Keccak team. "Strengths of Keccak – Design and security" (http://keccak.noekeon.org) Retrieved 30 January 2013. "*Unlike SHA-1 and SHA-2, Keccak does not have the length-extension weakness, hence does not need the HMAC nested construction. Instead, MAC computation can be performed by simply prepending the message with the key*"
6. Bruce Schneier (August 2005). "SHA-1 Broken" (http://www.schneier.com/blog/archives/2005/02/sha1_broken.html) Retrieved 9 January 2009. "*although it doesn't affect applications such as HMAC where collisions aren't important*"
7. IETF (February 1997). "Security" (https://tools.ietf.org/html/rfc2104#section-6) *HMAC: Keyed-Hashing for Message Authentication* (https://tools.ietf.org/html/rfc2104) sec. 6. doi:10.17487/RFC2104 (https://doi.org/10.17487%2FRFC2104). RFC 2104. Retrieved 3 December 2009. "*The strongest attack known against HMAC is based on the frequency of collisions for the hash function H ("birthday attack") [PV, BCK2], and is totally impractical for minimally reasonable hash functions.*"
8. Bellare, Mihir (June 2006). "New Proofs for NMAC and HMAC: Security without Collision-Resistance" (http://cseweb.ucsd.edu/~mihir/papers/hmac-new.html). In Dwork, Cynthia (ed.). *Advances in Cryptology – Crypto 2006 Proceedings*. Lecture Notes in Computer Science 4117. Springer-Verlag. Retrieved 25 May 2010. "*This paper proves that HMAC is a PRF under the sole assumption that the compression function is a PRF. This recovers a proof based guarantee since no known attacks compromise the pseudorandomness of the compression function, and it also helps explain the resistance-to-attack that HMAC has shown even when implemented with hash functions whose (weak) collision resistance is compromised.*"
9. Jongsung, Kim; Biryukov, Alex; Preneel, Bart; Hong, Seokhie (2006). "On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1" (http://eprint.iacr.org/2006/187.pdf) (PDF).
10. Wang, Xiaoyun; Yu, Hongbo; Wang, Wei; Zhang, Haina; Zhan, Tao (2009). "Cryptanalysis on HMAC/NMAC-MD5 and MD5-MAC" (https://www.iacr.org/archive/eurocrypt2009/54790122/54790122.pdf) (PDF). Retrieved 15 June 2015.
11. "RFC 6151 – Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms" (https://tools.ietf.org/html/rfc6151) Internet Engineering Task Force. March 2011. Retrieved 15 June 2015.

Notes

- Mihir Bellare, Ran Canetti and Hugo Krawczyk, Keying Hash Functions for Message Authentication, CRYPTO 1996, pp. 1–15 (PS or PDF).
- Mihir Bellare, Ran Canetti and Hugo Krawczyk, Message authentication using hash functions: The HMAC construction, *CryptoBytes* 2(1), Spring 1996 (PS or PDF).

External links

- RFC2104
- Online HMAC Calculator for dozens of underlying hashing algorithms
- Online HMAC Generator / Tester Tool
- FIPS PUB 198-1, *The Keyed-Hash Message Authentication Code (HMAC)*
- C HMAC implementation
- Python HMAC implementation
- Java implementation