



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Čeština
Deutsch
فارسی
Français
Italiano
日本語
Polski
Português
Русский
Slovenčina
Українська
中文

Edit links

[Create account](#) [Log in](#)

Article [Talk](#)

[Read](#) [Edit](#) [View history](#)

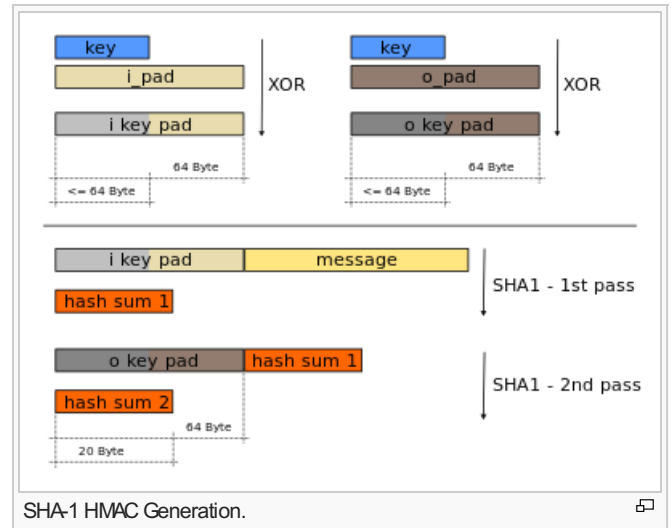
Hash-based message authentication code

From Wikipedia, the free encyclopedia
(Redirected from [Keyed-hash message authentication code](#))

In [cryptography](#), a **keyed-hash message authentication code (HMAC)** is a specific construction for calculating a [message authentication code \(MAC\)](#) involving a [cryptographic hash function](#) in combination with a secret [cryptographic key](#). As with any MAC, it may be used to simultaneously verify both the *data integrity* and the *authentication* of a [message](#). Any cryptographic hash function, such as [MD5](#) or [SHA-1](#), may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-MD5 or HMAC-SHA1 accordingly. The cryptographic strength of the HMAC depends upon the [cryptographic strength](#) of the underlying hash function, the size of its hash output, and on the size and quality of the key.

An iterative hash function breaks up a message into blocks of a fixed size and iterates over them with a [compression function](#). For example, MD5 and SHA-1 operate on 512-bit blocks. The size of the output of HMAC is the same as that of the underlying hash function (128 or 160 bits in the case of MD5 or SHA-1, respectively), although it can be truncated if desired.

The definition and analysis of the HMAC construction was first published in 1996 by [Mihir Bellare](#), Ran Canetti, and Hugo Krawczyk,^[1] who also wrote [RFC 2104](#). This paper also defined a variant called NMAC that is rarely, if ever, used. [FIPS PUB 198](#) generalizes and standardizes the use of HMACs. HMAC-SHA1 and HMAC-MD5 are used within the [IPsec](#) and [TLS](#) protocols.



Contents

- 1 Definition (from [RFC 2104](#))
- 2 Implementation
- 3 Design principles
- 4 Security
- 5 Examples of HMAC (MD5, SHA1, SHA256)
- 6 References
- 7 External links

Definition (from [RFC 2104](#)) [[edit](#)]

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

where

H is a cryptographic hash function,

K is a secret key [padded](#) to the right with extra zeroes to the input block size of the hash function, or the hash of the original key if it is longer than that block size,

m is the message to be authenticated,

|| denotes [concatenation](#),

⊕ denotes [exclusive or \(XOR\)](#),

opad is the outer padding (0x5c5c5c...5c5c, one-block-long [hexadecimal](#) constant),

and *ipad* is the inner padding (0x363636...3636, one-block-long [hexadecimal](#) constant).

Implementation [\[edit\]](#)

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 (key, message)
  if (length(key) > blocksize) then
    key = hash(key) // keys longer than blocksize are shortened
  end if
  if (length(key) < blocksize) then
    key = key // [0x00 * (blocksize - length(key))] // keys shorter than
    blocksize are zero-padded (where // is concatenation)
  end if

  o_key_pad = [0x5c * blocksize] ⊕ key // Where blocksize is that of the
  underlying hash function
  i_key_pad = [0x36 * blocksize] ⊕ key // Where ⊕ is exclusive or (XOR)

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

The following is a [Python](#) implementation of HMAC-MD5:

```
#!/usr/bin/env python

from hashlib import md5

trans_5C = bytearray((x ^ 0x5c for x in range(256)))
trans_36 = bytearray((x ^ 0x36 for x in range(256)))
blocksize = md5().block_size # 64

def hmac_md5(key, msg):
    if len(key) > blocksize:
        key = md5(key).digest()
    key = key + bytearray(blocksize - len(key))
    o_key_pad = key.translate(trans_5C)
    i_key_pad = key.translate(trans_36)
    return md5(o_key_pad + md5(i_key_pad + msg).digest())

if __name__ == "__main__":
    # This is one example from the appendix of RFC 2104
    h = hmac_md5(b"Jefe", b"what do ya want for nothing?")
    print(h.hexdigest()) # 750c783e6ab0b503eaa86e310a5db738
```

Python includes an `hmac` module,^[3] so the function above can be replaced by a shorter version.

```
import hmac
from hashlib import md5

def hmac_md5(key, msg):
    return hmac.HMAC(key, msg, md5)
```

The following is a [PHP](#) implementation of HMAC-SHA1:

```
function sha1_hmac($key,$data,$blockSize=64,$opad=0x5c,$ipad=0x36) {

    // Keys longer than blocksize are shortened
    if (strlen($key) > $blockSize) {
        $key = sha1($key,true);
    }

    // Keys shorter than blocksize are right, zero-padded (concatenated)
    $key = str_pad($key,$blockSize,chr(0x00),STR_PAD_RIGHT);
    $o_key_pad = $i_key_pad = '';
}
```

```

    for($i = 0;$i < $blockSize;$i++) {
        $o_key_pad .= chr(ord(substr($key,$i,1)) ^ $opad);
        $i_key_pad .= chr(ord(substr($key,$i,1)) ^ $ipad);
    }

    return sha1($o_key_pad.sha1($i_key_pad.$data,true),true);
}

$hmact = sha1_hmac('key','The quick brown fox jumps over the lazy dog');

```

The following is an implementation using Qt's QCryptographicHash

```

QByteArray hmacSha256(QByteArray key, const QByteArray& message) {
    const int blocksize = 64;
    if (key.length() > blocksize)
        QCryptographicHash::hash(key, QCryptographicHash::Sha256);

    while (key.length() < blocksize)
        key.append('\0');

    QByteArray o_key_pad('\x5c', blocksize);
    o_key_pad.fill('\x5c', blocksize);

    QByteArray i_key_pad;
    i_key_pad.fill('\x36', blocksize);

    for (int i=0; i<blocksize; i++) {
        o_key_pad[i] = o_key_pad[i] ^ key[i];
        i_key_pad[i] = i_key_pad[i] ^ key[i];
    }

    return QCryptographicHash::hash(o_key_pad
        + QCryptographicHash::hash(i_key_pad + message, QCryptographicHash::Sha256),
        QCryptographicHash::Sha256);
}

```

The following is an example using Node.js's built in crypto module using HMAC-SHA1:

```

var crypto = require('crypto'),
    hmac = crypto.createHmac('sha1', 'Shared Secret');

hmac.update(message);

var hmacHash = hmac.digest('hex');

```

Design principles [\[edit\]](#)

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 $\text{MAC} = \text{H}(\text{key} \parallel \text{message})$. 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 $\text{MAC} = \text{H}(\text{message} \parallel \text{key})$, 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 m_1 and m_2 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 $\text{MAC} = \text{H}(\text{key} \parallel \text{message} \parallel \text{key})$ is better, but various security papers have suggested vulnerabilities with this approach, even when two different keys are used.^{[1][4][5]}

No known extensions attacks have been found against the current HMAC specification which is defined as $\text{H}(\text{key} \parallel \text{H}(\text{key} \parallel \text{message}))$ 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.^[citation needed]

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.^[6]

Security [\[edit\]](#)

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.^{[7][8][9]} 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.^[10]

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.^[11]

In 2011 an informational [RFC 6151](#) [↗](#)^[12] was approved to update the 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.*"

In improperly-secured systems a [timing attack](#) can be performed to find out a HMAC digit by digit.^[13]

Examples of HMAC (MD5, SHA1, SHA256) [\[edit\]](#)

Here are some empty HMAC values:

```
HMAC_MD5("", "") = 0x74e6f7298a9c2d168935f58c001bad88
HMAC_SHA1("", "") = 0xfdbd1d1b18aa6c08324b7d64b71fb76370690e1d
HMAC_SHA256("", "") =
0xb613679a0814d9ec772f95d778c35fc5ff1697c493715653c6c712144292c5ad
```



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") =
0x80070713463e7749b90c2dc24911e275
HMAC_SHA1("key", "The quick brown fox jumps over the lazy dog") =
0xde7c9b85b8b78aa6bc8a7a36f70a90701c9db4d9
HMAC_SHA256("key", "The quick brown fox jumps over the lazy dog") =
0xf7bc83f430538424b13298e6aa6fb143ef4d59a14946175997479dbc2d1a3cd8
```

--

References [\[edit\]](#)


- [^] ^a ^b [Bellare, Mihir; Canetti, Ran; Krawczyk, Hugo \(1996\). "Keying Hash Functions for Message Authentication" \[↗\]\(#\)](#).
- [^] [RFC 2104](#) [↗](#), section 2, "Definition of HMAC", page 3.
- [^] [hmac — Keyed-Hashing for Message Authentication](#) [↗](#), [Python Software Foundation](#), retrieved 7 May 2014
- [^] [Preneel, Bart; van Oorschot, Paul C. \(1995\). "MDx-MAC and Building Fast MACs from Hash Functions" \[↗\]\(#\)](#). Retrieved 28 August 2009.
- [^] [Preneel, Bart; van Oorschot, Paul C. \(1995\). "On the Security of Two MAC Algorithms" \[↗\]\(#\)](#). Retrieved 28 August 2009.
- [^] [Keccak team. "Strengths of Keccak - Design and security" \[↗\]\(#\)](#). 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.*"
- [^] [Bruce Schneier \(August 2005\). "SHA-1 Broken" \[↗\]\(#\)](#). Retrieved 9 January 2009. "*although it doesn't affect applications such as HMAC where collisions aren't important*"
- [^] [IETF \(February 1997\). "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.*"

9. [^] Bellare, Mihir (June 2006). "New Proofs for NMAC and HMAC: Security without Collision-Resistance" [↗](#). In Dwork, Cynthia. *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."
10. [^] 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"  (PDF).
11. [^] Wang, Xiaoyun; Yu, Hongbo; Wang, Wei; Zhang, Haina; Zhan, Tao (2009). "Cryptanalysis on HMAC/NMAC-MD5 and MD5-MAC"  (PDF). Retrieved 15 June 2015.
12. [^] "RFC 6151 – Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms" [↗](#). Internet Engineering Task Force. March 2011. Retrieved 15 June 2015.
13. [^] Briefly mentioned at the end of this session [Sebastian Schinzel: Time is on my Side - Exploiting Timing Side Channel Vulnerabilities on the Web](#) [↗](#) 28th Chaos Communication Congress, 2011.

Notes

- Mihir Bellare, Ran Canetti and Hugo Krawczyk, Keying Hash Functions for Message Authentication, [CRYPTO 1996](#), pp1–15 (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 [\[edit\]](#)

- [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)*  ↗
- [PHP HMAC implementation](#) ↗
- [Python HMAC implementation](#) ↗
- [Perl HMAC implementation](#) ↗
- [Ruby HMAC implementation](#) ↗
- [C HMAC implementation](#) ↗
- [C++ HMAC implementation](#) (part of Crypto++) ↗
- [Java implementation](#) ↗
- [JavaScript MD5 and SHA HMAC implementation](#) ↗
- [JavaScript SHA-only HMAC implementation](#) ↗
- [.NET's System.Security.Cryptography.HMAC](#) ↗
- [Delphi/Free Pascal implementation](#) ↗

v · t · eHash functions & message authentication codes	
	Security summary
Common functions	MD5 · SHA-1 · SHA-2 · SHA-3/Keccak
SHA-3 finalists	BLAKE · Grøstl · JH · Skein · Keccak (winner)
Other functions	FSB · ECOH · GOST · HAS-160 · HAVAL · LMhash · MDC-2 · MD2 · MD4 · MD6 · N-Hash · RadioGatún · RIPEMD · SipHash · Snefru · Streebog · SWIFFT · Tiger · VSH · WHIRLPOOL · crypt(3) (DES)
MAC algorithms	DAA · CBC-MAC · HMAC · OMAC/CMAC · PMAC · VMAC · UMAC · Poly1305-AES
Authenticated encryption modes	CCM · CWC · EAX · GCM · IAPM · OCB
Attacks	Collision attack · Preimage attack · Birthday attack · Brute force attack · Rainbow table · Distinguishing attack · Side-channel attack · Length extension attack
Design	Avalanche effect · Hash collision · Merkle–Damgård construction
Standardization	CRYPTREC · NESSIE · NIST hash function competition
Utilization	Salt · Key stretching · Message authentication
v · t · eCryptography	
	History of cryptography · Cryptanalysis · Cryptography portal · Outline of cryptography
	Symmetric-key algorithm · Block cipher · Stream cipher · Public-key cryptography · Cryptographic hash function · Message authentication code · Random numbers · Steganography

Categories: [Message authentication codes](#) | [Hashing](#)