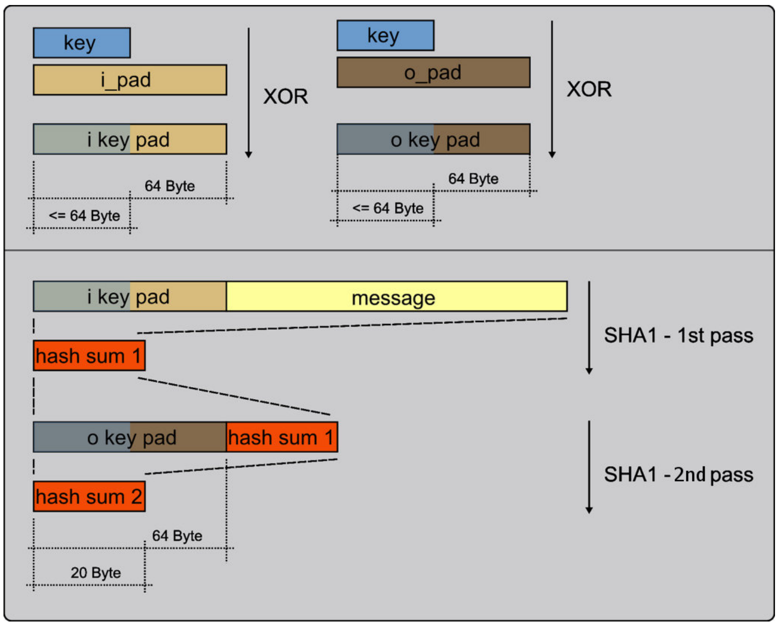
Hash-based message authentication code



SHA-1 HMAC Generation.

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, 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.

Definition (from RFC 2104)

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

where

H is a cryptographic hash function,

K is a secret key padded to the right with extra zeros to the input block size of the hash function, or the hash of the original key if it's 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

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.[1]

```
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 = "".join(chr(x ^ 0x5c) for x in xrange(256))
trans_36 = "".join(chr(x ^ 0x36) for x in xrange(256))
blocksize = md5().block_size

def hmac_md5(key, msg):
    if len(key) > blocksize:
        key = md5(key).digest()
    key += chr(0) * (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__":
    h = hmac_md5("key", "The quick brown fox jumps over the lazy dog")
    print h.hexdigest()  # 80070713463e7749b90c2dc24911e275
```

Example usage

A business that suffers from attackers that place fraudulent Internet orders may insist that all its customers deposit a secret symmetric key with them. Along with an order, a customer must supply the order's HMAC digest, computed using the customer's key. The business, knowing the customer's key, can then verify that the order originated from the stated customer and has not been tampered with.

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 = $\mathbf{H}(key \parallel message)$. 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 = $\mathbf{H}(message \parallel key)$, 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 = $\mathbf{H}(key \parallel message \parallel key)$ is better, however various security papers have suggested vulnerabilities with this approach, even when two different keys are used.

No known extensions attacks have been found against the current HMAC specification which is defined as $\mathbf{H}(key \parallel \mathbf{H}(key \parallel message))$ 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.

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. 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.

In improperly-secured systems a timing attack can be performed to find out a HMAC digit by digit.[2]

Examples of HMAC (MD5, SHA1, SHA256)

Here are some empty HMAC values:

```
HMAC_MD5("", "") = 0x74e6f7298a9c2d168935f58c001bad88

HMAC_SHA1("", "") = 0xfbdb1d1b18aa6c08324b7d64b71fb76370690e1d

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

[1] RFC 2104 (http://tools.ietf.org/html/rfc2104), section 2, "Definition of HMAC", page 3.
[2] Briefly mentioned at the end of this session Sebastian Schinzel:Time is on my Side - Exploiting Timing Side Channel Vulnerabilities on the Web (http://events.ccc.de/congress/2011/Fahrplan/events/4640.en.html) 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) (http://www-cse.ucsd.edu/users/mihir/papers/hmac.html#kmd5-paper).
- Mihir Bellare, Ran Canetti and Hugo Krawczyk, Message authentication using hash functions: The HMAC construction, *CryptoBytes* 2(1), Spring 1996 (PS or PDF) (http://www-cse.ucsd.edu/users/mihir/papers/hmac.html#hmac-cryptobytes).

External links

- RFC2104 (http://www.ietf.org/rfc/rfc2104.txt)
- Online HMAC Calculator for dozens of underlying hashing algorithms (http://quickhash.com/hmac)
- Online HMAC Generator / Tester Tool (http://www.freeformatter.com/hmac-generator.html)
- FIPS PUB 198-1, *The Keyed-Hash Message Authentication Code (HMAC)* (http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf)
- PHP HMAC implementation (http://php.net/manual/en/function.hash-hmac.php)
- Python HMAC implementation (http://docs.python.org/lib/module-hmac.html)
- Perl HMAC implementation (http://cpan.uwinnipeg.ca/htdocs/Digest-HMAC/Digest/HMAC.pm.html)
- Ruby HMAC implementation (http://ruby-hmac.rubyforge.org/)
- C HMAC implementation (http://www.ouah.org/ogay/hmac/)
- C++ HMAC implementation (part of Crypto++) (http://www.cryptopp.com)
- Java implementation (http://docs.oracle.com/javase/1.5.0/docs/guide/security/jce/JCERefGuide.html#HmacEx)
- JavaScript MD5 and SHA HMAC implementation (http://pajhome.org.uk/crypt/md5/instructions.html)
- JavaScript SHA-only HMAC implementation (http://caligatio.github.com/jsSHA/)
- .NET's System.Security.Cryptography.HMAC (http://msdn.microsoft.com/en-us/library/system.security.cryptography.hmac.aspx)

Article Sources and Contributors

Hash-based message authentication code *Source*: http://en.wikipedia.org/w/index.php?oldid=594959925 *Contributors*: 1exec1, Acdx, Angela, Aomelche, Aquntus, ArnoldReinhold, Arvindn, AxelBoldt, Balrog, BenjaminGittins, BorkaD, Caligatio, Capricorn42, CesarB's unpriviledged account, Chrismiceli, Coastside, Conseguenza, Conversion script, CraSH, Crcklssp, Cvk, Cybercobra, DRLB, Daf, Daira Hopwood, Damian Yerrick, DataWraith, Daverocks, Davidgothberg, Depictionimage, Don4of4, Drake Redcrest, Drzepka, Ed Brey, Ehn, Eli of Athens, Eriodega, Eta Aquariids, Firealwaysworks, Fluffy 543, Frap, Fresheneesz, GBL, Grignaak, Grr82, Gurch, Hanche, Heandel, Henridv, IFaqeer, J-Wiki, Jafet, Jaydec, Jdcc, Jesse Viviano, Jsnx, Kate, Kbk, Khazar, Kmag, Lee J Haywood, Martijnthie, Matt Crypto, Michael miceli, Minna Sora no Shita, Mmernex, Modbus.ug, MrOllie, Mrzaius, MusikAnimal, Nichalp, Northox, Ntsimp, ORBIT, Oli Filth, Pedro, Pgimeno, Plfernandez, Polymorphm, Quadrescence, RJFJR, Reikon, Rettetast, Rich Farmbrough, Rjwilmsi, Rossmcm, Runtime, Saimhe, Salih, Schnolle, Shd, Sinar, TyA, VegKilla, Voltin, Waxmop, Ww, Yaronf, Zacchiro, ZeroOne, 174 anonymous edits

Image Sources, Licenses and Contributors

File:Shahmac.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Shahmac.jpg *License*: Creative Commons Attribution 2.0 *Contributors*: Bender235, Danhash, Dawnseeker2000, ESkog, It Is Me Here, Jonerworm, Miserlou, Peachey88, Sfan00 IMG, 1 anonymous edits

License