# Package 'digest'

July 2, 2014

**Version** 0.6.4

**Date** $Date: 2013-12-02 21:56:30 -0600 (Mon, 02 Dec 2013) $

**Author** Dirk Eddelbuettel <edd@debian.org> with contributions by Antoine Lucas, Jarek Tuszynski, Henrik Bengtsson, Simon Urbanek,Mario Frasca, Bryan Lewis, Murray Stokely, Hannes Muehleisen and Duncan Murdoch.

**Maintainer** Dirk Eddelbuettel <edd@debian.org>

**Title** Create cryptographic hash digests of R objects

**Description** The digest package provides a function 'digest()' for the creation of hash digests of arbitrary R objects (using the md5, sha-1,sha-256 and crc32 algorithms) permitting easy comparison of R language objects, as well as a function 'hmac()' to create hash-based message authentication code.

The md5 algorithm by Ron Rivest is specified in RFC 1321, the sha-1 and sha-256 algorithms are specified in FIPS-180-1 and FIPS-180-2,and the crc32 algorithm is described in ftp://ftp.rocksoft.com/cliens/rocksoft/papers/crc_v3.txt.

For md5, sha-1, sha-256 and aes, this package uses a small standalone implementations that were provided by Christophe Devine. For crc32, code from the zlib library is used. For sha-512, an implementation by Aaron D. Gifford is used.

Please note that this package is not meant to be deployed for cryptographic purposes for which more comprehensive (and widely tested) libraries such as OpenSSL should be used.

**Depends** R (>= 2.4.1)

**License** GPL-2

**URL** http://dirk.eddelbuettel.com/code/digest.html

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2013-12-03 07:39:12

# R topics documented:



---

`AES` *Create AES block cipher object*

---

## Description

This creates an object that can perform the Advanced Encryption Standard (AES) block cipher.

## Usage

```
AES(key, mode=c("ECB", "CBC", "CTR"), IV=NULL)
```

## Arguments

| | |
|---|---|
| `key` | The key as a 16, 24 or 32 byte raw vector for AES-128, AES-192 or AES-256 respectively. |
| `mode` | The encryption mode to use. Currently only "electronic codebook" (ECB), "cipher-block chaining" (CBC) and "counter" (CTR) modes are supported. |
| `IV` | The initial vector for CBC mode or initial counter for CTR mode. |

## Details

The standard NIST definition of CTR mode doesn't define how the counter is updated, it just requires that it be updated with each block and not repeat itself for a long time. This implementation treats it as a 128 bit integer and adds 1 with each successive block.

## Value

An object of class `"AES"`. This is a list containing the following component functions:

| | |
|---|---|
| `encrypt(text)` | A function to encrypt a text vector. The text may be a single element character vector or a raw vector. It returns the ciphertext as a raw vector. |
| `decrypt(ciphertext, raw = FALSE)` | A function to decrypt the ciphertext. In ECB mode, the same AES object can be used for both encryption and decryption, but in CBC and CTR modes a new object needs to be created, using the same initial `key` and `IV` values. |
| `IV()` | Report on the current state of the initialization vector. As blocks are encrypted or decrypted in CBC or CTR mode, the initialization vector is updated, so both operations can be performed sequentially on subsets of the text or ciphertext. |
| `block_size(), key_size(), mode()` | Report on these aspects of the AES object. |

## Author(s)

The R interface was written by Duncan Murdoch. The design is loosely based on the Python Crypto implementation. The underlying AES implementation is by Christophe Devine.

## References

United States National Institute of Standards and Technology (2001). "Announcing the ADVANCED ENCRYPTION STANDARD (AES)". Federal Information Processing Standards Publication 197. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

Morris Dworkin (2001). "Recommendation for Block Cipher Modes of Operation". NIST Special Publication 800-38A 2001 Edition. http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

## Examples

```
# First in ECB mode: the repeated block is coded the same way each time
msg <- as.raw(c(1:16, 1:16))
key <- as.raw(1:16)
aes <- AES(key, mode="ECB")
aes$encrypt(msg)
aes$decrypt(aes$encrypt(msg), raw=TRUE)

# Now in CBC mode:  each encoding is different
iv <- sample(0:255, 16, replace=TRUE)
aes <- AES(key, mode="CBC", iv)
code <- aes$encrypt(msg)
code

# Need a new object for decryption in CBC mode
aes <- AES(key, mode="CBC", iv)
aes$decrypt(code, raw=TRUE)

# FIPS-197 examples

hextextToRaw <- function(text) {
  vals <- matrix(as.integer(as.hexmode(strsplit(text, "")[[1]])), ncol=2, byrow=TRUE)
  vals <- vals %*% c(16, 1)
  as.raw(vals)
}

plaintext       <- hextextToRaw("00112233445566778899aabbccddeeff")

aes128key       <- hextextToRaw("000102030405060708090a0b0c0d0e0f")
aes128output    <- hextextToRaw("69c4e0d86a7b0430d8cdb78070b4c55a")
aes <- AES(aes128key)
aes128 <- aes$encrypt(plaintext)
stopifnot(identical(aes128, aes128output))
stopifnot(identical(plaintext, aes$decrypt(aes128, raw=TRUE)))

aes192key       <- hextextToRaw("000102030405060708090a0b0c0d0e0f1011121314151617")
aes192output    <- hextextToRaw("dda97ca4864cdfe06eaf70a0ec0d7191")
```

```
aes <- AES(aes192key)
aes192 <- aes$encrypt(plaintext)
stopifnot(identical(aes192, aes192output))
stopifnot(identical(plaintext, aes$decrypt(aes192, raw=TRUE)))

aes256key     <- hextextToRaw("000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f")
aes256output     <- hextextToRaw("8ea2b7ca516745bfeafc49904b496089")
aes <- AES(aes256key)
aes256 <- aes$encrypt(plaintext)
stopifnot(identical(aes256, aes256output))
stopifnot(identical(plaintext, aes$decrypt(aes256, raw=TRUE)))

# SP800-38a examples

plaintext <- hextextToRaw(paste("6bc1bee22e409f96e93d7e117393172a",
                                "ae2d8a571e03ac9c9eb76fac45af8e51",
                                "30c81c46a35ce411e5fbc1191a0a52ef",
                                "f69f2445df4f9b17ad2b417be66c3710",sep=""))
key <- hextextToRaw("2b7e151628aed2a6abf7158809cf4f3c")

ecb128output <- hextextToRaw(paste("3ad77bb40d7a3660a89ecaf32466ef97",
                                   "f5d3d58503b9699de785895a96fdbaaf",
                                   "43b1cd7f598ece23881b00e3ed030688",
                                   "7b0c785e27e8ad3f8223207104725dd4",sep=""))
aes <- AES(key)
ecb128 <- aes$encrypt(plaintext)
stopifnot(identical(ecb128, ecb128output))
stopifnot(identical(plaintext, aes$decrypt(ecb128, raw=TRUE)))

cbc128output <- hextextToRaw(paste("7649abac8119b246cee98e9b12e9197d",
                                   "5086cb9b507219ee95db113a917678b2",
                                   "73bed6b8e3c1743b7116e69e22229516",
                                   "3ff1caa1681fac09120eca307586e1a7",sep=""))
iv <- hextextToRaw("000102030405060708090a0b0c0d0e0f")
aes <- AES(key, mode="CBC", IV=iv)
cbc128 <- aes$encrypt(plaintext)
stopifnot(identical(cbc128, cbc128output))
aes <- AES(key, mode="CBC", IV=iv)
stopifnot(identical(plaintext, aes$decrypt(cbc128, raw=TRUE)))

ctr128output <- hextextToRaw(paste("874d6191b620e3261bef6864990db6ce",
                                   "9806f66b7970fdff8617187bb9fffdff",
                                   "5ae4df3edbd5d35e5b4f09020db03eab",
                                   "1e031dda2fbe03d1792170a0f3009cee",sep=""))
iv <- hextextToRaw("f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff")
aes <- AES(key, mode="CTR", IV=iv)
ctr128 <- aes$encrypt(plaintext)
stopifnot(identical(ctr128, ctr128output))
aes <- AES(key, mode="CTR", IV=iv)
stopifnot(identical(plaintext, aes$decrypt(ctr128, raw=TRUE)))
```

digest *Create hash function digests for arbitrary R objects*

## Description

The digest function applies a cryptographical hash function to arbitrary R objects. By default, the objects are internally serialized, and either one of the currently implemented MD5 and SHA-1 hash functions algorithms can be used to compute a compact digest of the serialized object.

In order to compare this implementation with others, serialization of the input argument can also be turned off in which the input argument must be a character string for which its digest is returned.

## Usage

```
digest(object, algo=c("md5", "sha1", "crc32", "sha256", "sha512"),
       serialize=TRUE, file=FALSE, length=Inf, skip="auto", ascii=FALSE,
       raw=FALSE)
```

## Arguments

| | |
|---|---|
| object | An arbitrary R object which will then be passed to the serialize function, unless the serialize argument is set to FALSE. |
| algo | The algorithms to be used; currently available choices are md5, which is also the default, sha1, crc32 and sha256. |
| serialize | A logical variable indicating whether the object should be serialized using serialize (in ASCII form). Setting this to FALSE allows to compare the digest output of given character strings to known control output. It also allows the use of raw vectors such as the output of non-ASCII serialization. |
| file | A logical variable indicating whether the object is a file name or a file name if object is not specified. |
| length | Number of characters to process. By default, when length is set to Inf, the whole string or file is processed. |
| skip | Number of input bytes to skip before calculating the digest. Negative values are invalid and currently treated as zero. Special value "auto" will cause serialization header to be skipped if serialize is set to TRUE (the serialization header contains the R version number thus skipping it allows the comparison of hashes across platoforms and some R versions). |
| ascii | This flag is passed to the serialize function if serialize is set to TRUE, determining whether the hash is computed on the ASCII or binary represenataion. |
| raw | A logical variable with a default value of FALSE, implying digest returns digest output as ASCII hex values. Set to TRUE to return digest output in raw (binary) form. |

## Details

Cryptographic hash functions are well researched and documented. The MD5 algorithm by Ron Rivest is specified in RFC 1321. The SHA-1 algorithm is specified in FIPS-180-1, SHA-2 is described in FIPS-180-2. Crc32 is described in ftp://ftp.rocksoft.com/cliens/rocksoft/papers/crc_v3.txt.

For md5, sha-1 and sha-256, this R implementation relies on standalone implementations in C by Christophe Devine. For crc32, code from the zlib library by Jean-loup Gailly and Mark Adler is used.

For sha-512, a standalone implementation from Aaron Gifford is used.

Please note that this package is not meant to be used for cryptographic purposes for which more comprehensive (and widely tested) libraries such as OpenSSL should be used. Also, it is known that crc32 is not collision-proof. For sha-1, recent results indicate certain cryptographic weaknesses as well. For more details, see for example http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html.

## Value

The `digest` function returns a character string of a fixed length containing the requested digest of the supplied R object. For MD5, a string of length 32 is returned; for SHA-1, a string of length 40 is returned; for CRC32 a string of length 8.

## Author(s)

Dirk Eddelbuettel <edd@debian.org> for the R interface; Antoine Lucas for the integration of crc32; Jarek Tuszynski for the file-based operationss; Henrik Bengtsson and Simon Urbanek for improved serialization patches; Christophe Devine for the hash function implementations for sha-1, sha-256 and md5; Jean-loup Gailly and Mark Adler for crc32; Hannes Muehleisen for the integration of sha-512.

## References

MD5: http://www.ietf.org/rfc/rfc1321.txt.

SHA-1: http://www.itl.nist.gov/fipspubs/fip180-1.htm. SHA-256: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf. CRC32: ftp://ftp.rocksoft.com/cliens/rocksoft/papers/crc_v3.txt.

http://www.aarongifford.com/computers/sha.html for the integrated C implementation of sha-512.

http://www.cr0.net:8040/code/crypto for the underlying C functions used here for sha-1 and md5, and further references.

http://zlib.net for documentation on the zlib library which supplied the code for crc32.

http://en.wikipedia.org/wiki/SHA_hash_functions for documentation on the sha functions.

## See Also

serialize, md5sum

## Examples

```
## Standard RFC 1321 test vectors
md5Input <-
  c("",
    "a",
    "abc",
    "message digest",
    "abcdefghijklmnopqrstuvwxyz",
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
    paste("12345678901234567890123456789012345678901234567890123456789012",
          "345678901234567890", sep=""))
md5Output <-
  c("d41d8cd98f00b204e9800998ecf8427e",
    "0cc175b9c0f1b6a831c399e269772661",
    "900150983cd24fb0d6963f7d28e17f72",
    "f96b697d7cb7938d525a2f31aaf161d0",
    "c3fcd3d76192e4007dfb496cca67e13b",
    "d174ab98d277d9f5a5611c2c9f419d9f",
    "57edf4a22be3c955ac49da2e2107b67a")

for (i in seq(along=md5Input)) {
  md5 <- digest(md5Input[i], serialize=FALSE)
  stopifnot(identical(md5, md5Output[i]))
}

sha1Input <-
  c("abc", "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq")
sha1Output <-
  c("a9993e364706816aba3e25717850c26c9cd0d89d",
    "84983e441c3bd26ebaae4aa1f95129e5e54670f1")

for (i in seq(along=sha1Input)) {
  sha1 <- digest(sha1Input[i], algo="sha1", serialize=FALSE)
  stopifnot(identical(sha1, sha1Output[i]))
}

crc32Input <-
  c("abc",
    "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq")
crc32Output <-
  c("352441c2",
    "171a3f5f")

for (i in seq(along=crc32Input)) {
  crc32 <- digest(crc32Input[i], algo="crc32", serialize=FALSE)
  stopifnot(identical(crc32, crc32Output[i]))
}


sha256Input <-
  c("abc",
```

```
    "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq")
sha256Output <-
  c("ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad",
    "248d6a61d20638b8e5c026930c3e6039a33ce45964ff2167f6ecedd419db06c1")

for (i in seq(along=sha256Input)) {
  sha256 <- digest(sha256Input[i], algo="sha256", serialize=FALSE)
  stopifnot(identical(sha256, sha256Output[i]))
}

# SHA 512 example
sha512Input <-
  c("abc",
    "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq")
sha512Output <-
  c(paste("ddaf35a193617abacc417349ae20413112e6fa4e89a97ea20a9eeee64b55d39a",
          "2192992a274fc1a836ba3c23a3feebbd454d4423643ce80e2a9ac94fa54ca49f",
          sep=""),
    paste("204a8fc6dda82f0a0ced7beb8e08a41657c16ef468b228a8279be331a703c335",
          "96fd15c13b1b07f9aa1d3bea57789ca031ad85c7a71dd70354ec631238ca3445",
          sep=""))

for (i in seq(along=sha512Input)) {
  sha512 <- digest(sha512Input[i], algo="sha512", serialize=FALSE)
  stopifnot(identical(sha512, sha512Output[i]))
}

# example of a digest of a standard R list structure
digest(list(LETTERS, data.frame(a=letters[1:5], b=matrix(1:10,ncol=2))))

# test 'length' parameter and file input
fname <- file.path(R.home(),"COPYING")
x <- readChar(fname, file.info(fname)$size) # read file
for (alg in c("sha1", "md5", "crc32")) {
  # partial file
  h1 <- digest(x    , length=18000, algo=alg, serialize=FALSE)
  h2 <- digest(fname, length=18000, algo=alg, serialize=FALSE, file=TRUE)
  h3 <- digest( substr(x,1,18000) , algo=alg, serialize=FALSE)
  stopifnot( identical(h1,h2), identical(h1,h3) )
  # whole file
  h1 <- digest(x    , algo=alg, serialize=FALSE)
  h2 <- digest(fname, algo=alg, serialize=FALSE, file=TRUE)
  stopifnot( identical(h1,h2) )
}

# compare md5 algorithm to other tools
library(tools)
fname <- file.path(R.home(),"COPYING")
h1 <- as.character(md5sum(fname))
h2 <- digest(fname, algo="md5", file=TRUE)
stopifnot( identical(h1,h2) )
```

hmac *compute a hash-based message authentication code*

## Description

The hmac function calculates a message authentication code (MAC) involving the specified cryptographic hash function in combination with a given secret key.

## Usage

```
hmac(key, object,
     algo = c("md5", "sha1", "crc32", "sha256", "sha512"),
     serialize = FALSE, raw = FALSE, ...)
```

## Arguments

| | |
|---|---|
| key | An arbitrary character or numeric vector, to use as pre-shared secret key. |
| object | An arbitrary R object which will then be passed to the serialize function, unless the serialize argument is set to FALSE. |
| algo | The algorithms to be used; currently available choices are md5, which is also the default, sha1, crc32 and sha256. |
| serialize | default value of serialize is here FALSE, not TRUE as it is in digest. |
| raw | This flag alters the type of the output. Setting this to TRUE causes the function to return an object of type "raw" instead of "character". |
| ... | All remaining arguments are passed to digest. |

## Value

The hmac function uses the digest to return a hash digest as specified in the RFC 2104.

## Author(s)

Mario Frasca <mfrasca@zonnet.nl>.

## References

MD5: http://www.ietf.org/rfc/rfc1321.txt.

SHA-1: http://www.itl.nist.gov/fipspubs/fip180-1.htm. SHA-256: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf. CRC32: ftp://ftp.rocksoft.com/cliens/rocksoft/papers/crc_v3.txt.

http://www.aarongifford.com/computers/sha.html for the integrated C implementation of sha-512.

http://www.cr0.net:8040/code/crypto for the underlying C functions used here for sha-1 and md5, and further references.

http://zlib.net for documentation on the zlib library which supplied the code for crc32.

http://en.wikipedia.org/wiki/SHA_hash_functions for documentation on the sha functions.

## See Also

digest

## Examples

```
## Standard RFC 2104 test vectors
current <- hmac('Jefe', 'what do ya want for nothing?', "md5")
target <- '750c783e6ab0b503eaa86e310a5db738'
stopifnot(identical(target, as.character(current)))

current <- hmac(rep(0x0b, 16), 'Hi There', "md5")
target <- '9294727a3638bb1c13f48ef8158bfc9d'
stopifnot(identical(target, as.character(current)))

current <- hmac(rep(0xaa, 16), rep(0xdd, 50), "md5")
target <- '56be34521d144c88dbb8c733f0e8b3f6'
stopifnot(identical(target, as.character(current)))

## SHA1 tests inspired to the RFC 2104 and checked against the python
## hmac implementation.
current <- hmac('Jefe', 'what do ya want for nothing?', "sha1")
target <- 'effcdf6ae5eb2fa2d27416d5f184df9c259a7c79'
stopifnot(identical(target, as.character(current)))

current <- hmac(rep(0x0b, 16), 'Hi There', "sha1")
target <- '675b0b3a1b4ddf4e124872da6c2f632bfed957e9'
stopifnot(identical(target, as.character(current)))

current <- hmac(rep(0xaa, 16), rep(0xdd, 50), "sha1")
target <- 'd730594d167e35d5956fd8003d0db3d3f46dc7bb'
stopifnot(identical(target, as.character(current)))
```

# Index