

# Package ‘digest’

January 2, 2012

**Version** 0.5.1

**Date** \$Date: 2011-09-20 17:15:07 -0500 (Tue, 20 Sep 2011) \$

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

**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/clients/rocksoft/papers/crc\\_v3.txt](ftp://ftp.rocksoft.com/clients/rocksoft/papers/crc_v3.txt). For md5, sha-1 and sha-256, this package uses small standalone implementations that were provided by Christophe Devine. For crc32, code from the zlib library 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>

**Repository** CRAN

**Date/Publication** 2011-09-21 19:07:31

## R topics documented:

digest . . . . .	2
hmac . . . . .	5

<b>Index</b>	<b>8</b>
--------------	----------

digest	<i>Create hash function digests for arbitrary R objects</i>
--------	---

### 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"),
       serialize=TRUE, file=FALSE, length=Inf, skip="auto", ascii=FALSE,
       raw=FALSE)
```

### Arguments

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

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](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 operations; 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.

## 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](ftp://ftp.rocksoft.com/cliens/rocksoft/papers/crc_v3.txt).

<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](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",
  "abcdefghijklmnopqrstuvwxy",
  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxy0123456789",
  paste("123456789012345678901234567890123456789012345678901234567890123456789012",
        "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]))
}

# 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"), serialize = FALSE, raw = FALSE, ...)
```

## Arguments

<code>key</code>	An arbitrary character or numeric vector, to use as pre-shared secret key.
<code>object</code>	An arbitrary R object which will then be passed to the <code>serialize</code> function, unless the <code>serialize</code> argument is set to <code>FALSE</code> .

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](ftp://ftp.rocksoft.com/cliens/rocksoft/papers/crc_v3.txt).

<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](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

\*Topic **misc**  
digest, 2  
hmac, 5

digest, 2, 6

hmac, 5

md5sum, 3

serialize, 2, 3, 5