Package 'secretbase'

April 25, 2024

Type Package

Title Cryptographic Hash and Extendable-Output Functions

Version 0.5.0

Description Fast and memory-efficient streaming hash functions. Performs direct hashing of strings, raw bytes, and files potentially larger than memory, as well as hashing in-memory objects through R's serialization mechanism, without requiring allocation of the serialized object. Implementations include the SHA-256, SHA-3 and 'Keccak' cryptographic hash functions, SHAKE256 extendable-output function (XOF), and 'SipHash' pseudo-random function. The SHA-3 Secure Hash Standard was published by the National Institute of Standards and Technology (NIST) in 2015 at <doi:10.6028/NIST.FIPS.202>. The SHA-256 Secure Hash Standard was published by NIST in 2002 at <https://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>.

License GPL (>= 3)

BugReports https://github.com/shikokuchuo/secretbase/issues

URL https://shikokuchuo.net/secretbase/, https://github.com/shikokuchuo/secretbase/

Encoding UTF-8

Depends R (>= 3.5)

RoxygenNote 7.3.1

NeedsCompilation yes

Author Charlie Gao [aut, cre] (<https://orcid.org/0000-0002-0750-061X>), Hibiki AI Limited [cph]

Maintainer Charlie Gao <charlie.gao@shikokuchuo.net>

Repository CRAN

Date/Publication 2024-04-25 07:40:02 UTC

R topics documented:

`secretbase-package` *secretbase: Cryptographic Hash and Extendable-Output Functions*

Description

Fast and memory-efficient streaming hash functions. Performs direct hashing of strings, raw bytes, and files potentially larger than memory, as well as hashing in-memory objects through R's serialization mechanism, without requiring allocation of the serialized object. Implementations include the SHA-256, SHA-3 and Keccak cryptographic hash functions, SHAKE256 extendable-output function (XOF), and 'SipHash' pseudo-random function.

Author(s)

Charlie Gao `<charlie.gao@shikokuchuo.net>` (ORCID)

See Also

Useful links:

- `https://shikokuchuo.net/secretbase/`
- `https://github.com/shikokuchuo/secretbase/`
- Report bugs at `https://github.com/shikokuchuo/secretbase/issues`

`keccak` *Keccak Cryptographic Hash Algorithms*

Description

Returns a Keccak hash of the supplied object or file.

Usage

```
keccak(x, bits = 256L, convert = TRUE, file)
```

Arguments

`x`	object to hash. A character string or raw vector (without attributes) is hashed 'as is'. All other objects are stream hashed using R serialization (but without allocation of the serialized object).
`bits`	[default 256L] output size of the returned hash. Must be one of 224, 256, 384 or 512.
`convert`	[default TRUE] if TRUE, the hash is converted to its hex representation as a character string, if FALSE, output directly as a raw vector, or if NA, a vector of (32-bit) integer values.
`file`	character file name / path. If specified, 'x' is ignored. The file is stream hashed, thus capable of handling files larger than memory.

Value

A character string, raw or integer vector depending on 'convert'.

R Serialization Stream Hashing

Where this is used, serialization is always version 3 big-endian represenation and the headers (containing R version and native encoding information) are skipped to ensure portability across platforms.

References

Keccak is the underlying algorithm for SHA-3, and is identical apart from the value of the padding parameter.

The Keccak algorithm was designed by G. Bertoni, J. Daemen, M. Peeters and G. Van Assche.

This implementation is based on one by 'The Mbed TLS Contributors' under the 'Mbed TLS' Trusted Firmware Project at `https://www.trustedfirmware.org/projects/mbed-tls`.

Examples

```
# Keccak-256 hash as character string:
keccak("secret base")

# Keccak-256 hash as raw vector:
keccak("secret base", convert = FALSE)

# Keccak-224 hash as character string:
keccak("secret base", bits = 224)

# Keccak-384 hash as character string:
keccak("secret base", bits = 384)

# Keccak-512 hash as character string:
keccak("secret base", bits = 512)

# Keccak-256 hash a file:
file <- tempfile(); cat("secret base", file = file)
```

```
keccak(file = file)
unlink(file)
```

`sha256` *SHA-256 Cryptographic Hash Algorithm*

Description

Returns a SHA-256 hash of the supplied object or file, or HMAC if a secret key is supplied.

Usage

```
sha256(x, key = NULL, convert = TRUE, file)
```

Arguments

`x`	object to hash. A character string or raw vector (without attributes) is hashed 'as is'. All other objects are stream hashed using R serialization (but without allocation of the serialized object).
`key`	[default NULL] If NULL, the SHA-256 hash of 'x' is returned. Alternatively, supply a character string or raw vector as a secret key to generate an HMAC. Note: for character vectors only the first element is used.
`convert`	[default TRUE] if TRUE, the hash is converted to its hex representation as a character string, if FALSE, output directly as a raw vector, or if NA, a vector of (32-bit) integer values.
`file`	character file name / path. If specified, 'x' is ignored. The file is stream hashed, thus capable of handling files larger than memory.

Value

A character string, raw or integer vector depending on 'convert'.

R Serialization Stream Hashing

Where this is used, serialization is always version 3 big-endian represenation and the headers (containing R version and native encoding information) are skipped to ensure portability across platforms.

References

The SHA-256 Secure Hash Standard was published by the National Institute of Standards and Technology (NIST) in 2002 at `https://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf`.

This implementation is based on one by 'The Mbed TLS Contributors' under the 'Mbed TLS' Trusted Firmware Project at `https://www.trustedfirmware.org/projects/mbed-tls`.

Examples

```
# SHA-256 hash as character string:
sha256("secret base")

# SHA-256 hash as raw vector:
sha256("secret base", convert = FALSE)

# SHA-256 hash a file:
file <- tempfile(); cat("secret base", file = file)
sha256(file = file)
unlink(file)

# SHA-256 HMAC using a character string secret key:
sha256("secret", key = "base")

# SHA-256 HMAC using a raw vector secret key:
sha256("secret", key = charToRaw("base"))
```

`sha3` *SHA-3 Cryptographic Hash Algorithms*

Description

Returns a SHA-3 hash of the supplied object or file.

Usage

```
sha3(x, bits = 256L, convert = TRUE, file)
```

Arguments

`x`	object to hash. A character string or raw vector (without attributes) is hashed 'as is'. All other objects are stream hashed using R serialization (but without allocation of the serialized object).
`bits`	[default 256L] output size of the returned hash. Must be one of 224, 256, 384 or 512. For legacy reasons (usage is deprecated), all other values will return the result of `shake256`.
`convert`	[default TRUE] if TRUE, the hash is converted to its hex representation as a character string, if FALSE, output directly as a raw vector, or if NA, a vector of (32-bit) integer values.
`file`	character file name / path. If specified, 'x' is ignored. The file is stream hashed, thus capable of handling files larger than memory.

Value

A character string, raw or integer vector depending on 'convert'.

R Serialization Stream Hashing

Where this is used, serialization is always version 3 big-endian represenation and the headers (containing R version and native encoding information) are skipped to ensure portability across platforms.

References

The SHA-3 Secure Hash Standard was published by the National Institute of Standards and Technology (NIST) in 2015 at doi:10.6028/NIST.FIPS.202.

This implementation is based on one by 'The Mbed TLS Contributors' under the 'Mbed TLS' Trusted Firmware Project at `https://www.trustedfirmware.org/projects/mbed-tls`.

Examples

```
# SHA3-256 hash as character string:
sha3("secret base")

# SHA3-256 hash as raw vector:
sha3("secret base", convert = FALSE)

# SHA3-224 hash as character string:
sha3("secret base", bits = 224)

# SHA3-384 hash as character string:
sha3("secret base", bits = 384)

# SHA3-512 hash as character string:
sha3("secret base", bits = 512)

# SHA3-256 hash a file:
file <- tempfile(); cat("secret base", file = file)
sha3(file = file)
unlink(file)
```

`shake256` *SHAKE256 Extendable Output Function*

Description

Returns a SHAKE256 hash of the supplied object or file.

Usage

```
shake256(x, bits = 256L, convert = TRUE, file)
```

Arguments

`x`	object to hash. A character string or raw vector (without attributes) is hashed 'as is'. All other objects are stream hashed using R serialization (but without allocation of the serialized object).
`bits`	[default 256L] output size of the returned hash. Must be between 8 and 2^24 and coercible to integer.
`convert`	[default TRUE] if TRUE, the hash is converted to its hex representation as a character string, if FALSE, output directly as a raw vector, or if NA, a vector of (32-bit) integer values.
`file`	character file name / path. If specified, 'x' is ignored. The file is stream hashed, thus capable of handling files larger than memory.

Details

To produce single integer values suitable for use as random seeds for R's pseudo random number generators (RNGs), set 'bits' to 32 and 'convert' to NA.

Value

A character string, raw or integer vector depending on 'convert'.

R Serialization Stream Hashing

Where this is used, serialization is always version 3 big-endian represenation and the headers (containing R version and native encoding information) are skipped to ensure portability across platforms.

References

This implementation is based on one by 'The Mbed TLS Contributors' under the 'Mbed TLS' Trusted Firmware Project at https://www.trustedfirmware.org/projects/mbed-tls.

Examples

```
# SHAKE256 hash as character string:
shake256("secret base")

# SHAKE256 hash as raw vector:
shake256("secret base", convert = FALSE)

# SHAKE256 hash to integer:
sha3("secret base", bits = 32L, convert = NA)

# SHAKE256 hash a file:
file <- tempfile(); cat("secret base", file = file)
shake256(file = file)
unlink(file)
```

siphash13 *SipHash Pseudorandom Function*

Description

Returns a fast, cryptographically-strong SipHash keyed hash of the supplied object or file. SipHash-1-3 is optimised for performance. Note: SipHash is not a cryptographic hash algorithm.

Usage

```
siphash13(x, key = NULL, convert = TRUE, file)
```

Arguments

`x`	object to hash. A character string or raw vector (without attributes) is hashed 'as is'. All other objects are stream hashed using R serialization (but without allocation of the serialized object).
`key`	[default NULL] a character string or raw vector comprising the 16 byte (128 bit) key data, or else NULL which is equivalent to '0'. If a longer vector is supplied, only the first 16 bytes are used, and if shorter, padded with trailing '0'. Note: for character vectors only the first element is used.
`convert`	[default TRUE] if TRUE, the hash is converted to its hex representation as a character string, if FALSE, output directly as a raw vector, or if NA, a vector of (32-bit) integer values.
`file`	character file name / path. If specified, 'x' is ignored. The file is stream hashed, thus capable of handling files larger than memory.

Value

A character string, raw or integer vector depending on 'convert'.

R Serialization Stream Hashing

Where this is used, serialization is always version 3 big-endian represenation and the headers (containing R version and native encoding information) are skipped to ensure portability across platforms.

References

The SipHash family of cryptographically-strong pseudorandom functions (PRFs) are described in 'SipHash: a fast short-input PRF', Jean-Philippe Aumasson and Daniel J. Bernstein, Paper 2012/351, 2012, Cryptology ePrint Archive at `https://ia.cr/2012/351`.

This implementation is based on the SipHash streaming implementation by Daniele Nicolodi, David Rheinsberg and Tom Gundersen at `https://github.com/c-util/c-siphash`. This is in turn based on the SipHash reference implementation by Jean-Philippe Aumasson and Daniel J. Bernstein released to the public domain at `https://github.com/veorq/SipHash`.

Examples

```
# SipHash-1-3 hash as character string:
siphash13("secret base")

# SipHash-1-3 hash as raw vector:
siphash13("secret base", convert = FALSE)

# SipHash-1-3 hash using a character string key:
siphash13("secret", key = "base")

# SipHash-1-3 hash using a raw vector key:
siphash13("secret", key = charToRaw("base"))

# SipHash-1-3 hash a file:
file <- tempfile(); cat("secret base", file = file)
siphash13(file = file)
unlink(file)
```

Index