Package 'secretbase'

September 25, 2026

Type Package

Title Cryptographic Hash Functions and Data Encoding

Version 1.3.1

Description Fast and memory-efficient streaming hash functions, binary/text encoding and serialization. Hashes strings and raw vectors directly. Stream hashes files which can be larger than memory, as well as in-memory objects through R's serialization mechanism. Implements the SHA-256, SHA-3 and 'Keccak' cryptographic hash functions, SHAKE256 extendable-output function (XOF), 'SipHash' pseudo-random function, base64 (including the URL-safe variant) and base58 encoding, 'CBOR' and 'JSON' serialization.

License MIT + file LICENSE

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

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

Depends R (>= 3.5)

Config/build/compilation-database true

Config/roxygen2/markdown TRUE

Config/roxygen2/version 8.1.0

Encoding UTF-8

NeedsCompilation yes

Author Charlie Gao [aut, cre] (ORCID: <https://orcid.org/0000-0002-0750-061X>), Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>), Hibiki AI Limited [cph], The Mbed TLS Contributors [cph] (SHA-3, SHA-256 and base64 code from Mbed TLS), Red Hat, Inc. [cph] (SipHash code from c-siphash), Luke Dashjr [cph] (Base58 code from libbase58)

Maintainer Charlie Gao <charlie.gao@posit.co>

Repository CRAN

Date/Publication 2026-09-25 18:10:02 UTC

Contents

base58dec . . . 2
base58enc . . . 3
base64dec . . . 4
base64enc . . . 5
cbordec . . . 6
cborenc . . . 7
jsondec . . . 8
jsonenc . . . 9
keccak . . . 10
sha256 . . . 11
sha3 . . . 13
shake256 . . . 14
siphash13 . . . 15

Index **18**

`base58dec` *Base58 Decode*

Description

Decodes a character string or raw vector from base58 encoding with checksum.

Usage

```
base58dec(x, convert = TRUE)
```

Arguments

`x`	a character string or raw vector containing base58 encoded data.
`convert`	logical `TRUE` to convert back to a character string, `FALSE` to convert back to a raw vector or `NA` to decode and then unserialize back to the original object.

Details

The 4-byte checksum suffix is verified using double SHA-256 and an error is raised if validation fails. Note: does not expect a version byte prefix (unlike Bitcoin Base58Check).

The value of `convert` should be set to `TRUE`, `FALSE` or `NA` to be the reverse of the 3 encoding operations (for strings, raw vectors and arbitrary objects), in order to return the original object.

Value

A character string, raw vector, or other object depending on the value of `convert`.

References

This implementation is based on 'libbase58' by Luke Dashjr under the MIT licence at https://github.com/luke-jr/libbase58.

See Also

base58enc()

Examples

```
base58dec(base58enc("secret base"))
base58dec(base58enc(as.raw(c(1L, 2L, 4L))), convert = FALSE)
base58dec(base58enc(data.frame()), convert = NA)
```

---

base58enc *Base58 Encode*

Description

Encodes a character string, raw vector or other object to base58 encoding with a 4-byte checksum suffix.

Usage

```
base58enc(x, convert = TRUE)
```

Arguments

x	an object.
convert	logical TRUE to encode to a character string or FALSE to a raw vector.

Details

Adds a 4-byte checksum suffix (double SHA-256) to the data before base58 encoding. Note: does not include a version byte prefix (unlike Bitcoin Base58Check).

A character string or raw vector (with no attributes) is encoded as is, whilst all other objects are first serialized (using R serialisation version 3, big-endian representation).

Value

A character string or raw vector depending on the value of convert.

References

This implementation is based on 'libbase58' by Luke Dashjr under the MIT licence at https://github.com/luke-jr/libbase58.

See Also

base58dec()

Examples

```
base58enc("secret base")
base58enc(as.raw(c(1L, 2L, 4L)), convert = FALSE)
base58enc(data.frame())
```

---

base64dec *Base64 Decode*

Description

Decodes a character string, raw vector or other object from base64 encoding.

Usage

```
base64dec(x, convert = TRUE, url = FALSE)
```

Arguments

x	an object.
convert	logical TRUE to convert back to a character string, FALSE to convert back to a raw vector or NA to decode and then unserialize back to the original object.
url	logical TRUE to use the URL- and filename-safe base64url alphabet without padding, or FALSE (the default) for standard base64.

Details

The value of convert should be set to TRUE, FALSE or NA to be the reverse of the 3 encoding operations (for strings, raw vectors and arbitrary objects), in order to return the original object.

Set url = TRUE to decode the URL- and filename-safe variant (RFC 4648 section 5), which expects the - and _ alphabet without padding; inputs containing +, / or = are then rejected. The value of url must match the variant used to encode.

Value

A character string, raw vector, or other object depending on the value of convert.

References

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

See Also

base64enc()

Examples

```
base64dec(base64enc("secret base"))
base64dec(base64enc(as.raw(c(1L, 2L, 4L))), convert = FALSE)
base64dec(base64enc(data.frame()), convert = NA)
base64dec(base64enc("secret base", url = TRUE), url = TRUE)
```

---

base64enc *Base64 Encode*

Description

Encodes a character string, raw vector or other object to base64 encoding.

Usage

```
base64enc(x, convert = TRUE, url = FALSE)
```

Arguments

x	an object.
convert	logical TRUE to encode to a character string or FALSE to a raw vector.
url	logical TRUE to use the URL- and filename-safe base64url alphabet without padding, or FALSE (the default) for standard base64.

Details

A character string or raw vector (with no attributes) is encoded as is, whilst all other objects are first serialized (using R serialisation version 3, big-endian representation).

Set url = TRUE for the URL- and filename-safe variant (RFC 4648 section 5), which substitutes - and _ for + and / and omits padding (=), making the output safe to use within URLs and filenames without further escaping.

Value

A character string or raw vector depending on the value of convert.

References

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

See Also

base64dec()

Examples

```
base64enc("secret base")
base64enc(as.raw(c(1L, 2L, 4L)), convert = FALSE)
base64enc(data.frame())
base64enc("secret base", url = TRUE)
```

---

cbordec *CBOR Decode*

Description

Decode CBOR (Concise Binary Object Representation, RFC 8949) data to an R object.

Usage

```
cbordec(x)
```

Arguments

x	A raw vector containing CBOR-encoded data.

Details

CBOR types map to R types as follows:

- Integers: integer (if within range) or double
- Float16/Float32/Float64: double
- Byte strings: raw vectors
- Text strings: character
- false/true: logical
- null: NULL
- undefined: NA
- Arrays: lists
- Maps: named lists (keys must be text strings)

Note: CBOR arrays always decode to lists, so R atomic vectors encoded via cborenc() will decode to lists rather than vectors.

Value

The decoded R object.

See Also

cborenc()

Examples

```
# Round-trip encoding
original <- list(a = 1L, b = "test", c = TRUE)
cbordec(cborenc(original))
```

---

cborenc *CBOR Encode*

Description

Encode an R object to CBOR (Concise Binary Object Representation, RFC 8949) format.

Usage

```
cborenc(x)
```

Arguments

x	R object to encode. Supported types: NULL, logical, integer, double, character, raw vectors, and lists (named lists become CBOR maps, unnamed become CBOR arrays).

Details

This implementation supports a minimal CBOR subset:

- Unsigned and negative integers
- Float64
- Byte strings (raw vectors)
- Text strings (UTF-8)
- Simple values: false, true, null, undefined
- Arrays (unnamed lists/vectors)
- Maps (named lists)

Scalars (length-1 vectors) encode as CBOR primitives; longer vectors encode as CBOR arrays. NA values encode as CBOR undefined. Names on atomic vectors are ignored.

Note: atomic vectors do not round-trip perfectly as CBOR arrays decode to lists. Named lists round-trip correctly as CBOR maps.

Value

A raw vector containing the CBOR-encoded data.

See Also

cbordec()

Examples

```
# Encode a named list (becomes CBOR map)
cborenc(list(a = 1L, b = "hello"))

# Round-trip
cbordec(cborenc(list(x = TRUE, y = as.raw(1:3))))
```

---

jsondec *JSON Decode*

Description

Minimal JSON parser. Converts JSON to R objects with proper type handling.

Usage

```
jsondec(x)
```

Arguments

x	Character string or raw vector containing JSON data.

Details

This is a minimal implementation designed for parsing HTTP API responses.

Value

The corresponding R object, or an empty list for invalid input.

Type Mappings

- Object {} -> named list
- Array [] -> unnamed list
- String -> character
- Number -> numeric
- true/false -> logical
- null -> NULL

RFC 8259 Non-conformance

- Invalid JSON returns an empty list instead of erroring.
- Duplicate keys are preserved; R accessors (`$`, `[[`) return first match.
- Non-standard number forms may be accepted (e.g., leading zeros, hexadecimal).
- Invalid escape sequences are output literally (e.g., `\\uZZZZ` becomes `"uZZZZ"`).
- Incomplete Unicode escape sequences for emoji are tolerated.
- Nesting depth is limited to 512 levels.

See Also

`jsonenc()`

Examples

```
jsondec('{"name": "John", "age": 30}')
jsondec('[1, 2, 3]')
jsondec('"a string"')
jsondec('123')
jsondec('true')
```

---

jsonenc *JSON Encode*

Description

Minimal JSON encoder. Converts an R object to a JSON string.

Usage

```
jsonenc(x)
```

Arguments

x	An R object to encode as JSON.

Details

This is a minimal implementation designed for creating HTTP API request bodies.

Value

A character string containing the JSON representation.

Type Mappings

- Named list -> object `{}`
- Unnamed list -> array `[]`
- Character -> string (with escaping)
- Numeric/integer -> number
- Logical -> `true`/`false`
- `NULL`, `NA` -> `null`
- Scalars (length 1) -> primitive value
- Vectors (length > 1) -> array `[]`
- Unsupported types (e.g., functions) -> `null`

See Also

`jsondec()`

Examples

```
jsonenc(list(name = "John", age = 30L))
jsonenc(list(valid = TRUE, count = NULL))
jsonenc(list(nested = list(a = 1, b = list(2, 3))))
jsonenc(list(nums = 1:3, strs = c("a", "b")))
```

---

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 native R serialization.
`bits`	integer output size of the returned hash. Must be one of `224`, `256`, `384` or `512`.
`convert`	logical `TRUE` to convert the hash to its hex representation as a character string, `FALSE` to return directly as a raw vector, or `NA` to return as a vector of (32-bit) integers.
`file`	character file name / path. If specified, `x` is ignored. The file is stream hashed, and the file can be 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 representation and the headers (containing R version and native encoding information) are skipped to ensure portability across platforms.

As hashing is performed in a streaming fashion, there is no materialization of, or memory allocation for, the serialized object.

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 native R serialization.
key	if NULL, the SHA-256 hash of x is returned. If a character string or raw vector, this is used as a secret key to generate an HMAC. Note: for character vectors, only the first element is used.
convert	logical TRUE to convert the hash to its hex representation as a character string, FALSE to return directly as a raw vector, or NA to return as a vector of (32-bit) integers.
file	character file name / path. If specified, x is ignored. The file is stream hashed, and the file can be 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 representation and the headers (containing R version and native encoding information) are skipped to ensure portability across platforms.

As hashing is performed in a streaming fashion, there is no materialization of, or memory allocation for, the serialized object.

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 native R serialization.
bits	integer output size of the returned hash. Must be one of 224, 256, 384 or 512.
convert	logical TRUE to convert the hash to its hex representation as a character string, FALSE to return directly as a raw vector, or NA to return as a vector of (32-bit) integers.
file	character file name / path. If specified, x is ignored. The file is stream hashed, and the file can be 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 representation and the headers (containing R version and native encoding information) are skipped to ensure portability across platforms.

As hashing is performed in a streaming fashion, there is no materialization of, or memory allocation for, the serialized object.

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 native R serialization.
bits	integer output size of the returned hash. Value must be between 8 and 2^24.
convert	logical TRUE to convert the hash to its hex representation as a character string, FALSE to return directly as a raw vector, or NA to return as a vector of (32-bit) integers.
file	character file name / path. If specified, x is ignored. The file is stream hashed, and the file can be 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 representation and the headers (containing R version and native encoding information) are skipped to ensure portability across platforms.

As hashing is performed in a streaming fashion, there is no materialization of, or memory allocation for, the serialized object.

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:
shake256("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 native R serialization.
key	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	logical TRUE to convert the hash to its hex representation as a character string, FALSE to return directly as a raw vector, or NA to return as a vector of (32-bit) integers.
file	character file name / path. If specified, x is ignored. The file is stream hashed, and the file can be 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 representation and the headers (containing R version and native encoding information) are skipped to ensure portability across platforms.

As hashing is performed in a streaming fashion, there is no materialization of, or memory allocation for, the serialized object.

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

base58dec, 2
base58dec(), *4*
base58enc, 3
base58enc(), *3*
base64dec, 4
base64dec(), *6*
base64enc, 5
base64enc(), *5*

cbordec, 6
cbordec(), *8*
cborenc, 7
cborenc(), *6, 7*

jsondec, 8
jsondec(), *10*
jsonenc, 9
jsonenc(), *9*

keccak, 10

sha256, 11
sha3, 13
shake256, 14
siphash13, 15