Package ‘bcrypt’

January 26, 2018

Type Package

Title 'Blowfish' Password Hashing Algorithm

Version 1.1

Description Bindings to the 'blowfish' password hashing algorithm derived from the 'OpenBSD' implementation.

URL `https://github.com/jeroen/bcrypt`
`https://www.openbsd.org/papers/bcrypt-paper.pdf`

BugReports `https://github.com/jeroen/bcrypt/issues`

License BSD_2_clause + file LICENSE

Imports openssl

RoxygenNote 6.0.1.9000

Suggests spelling

Language en-US

Encoding UTF-8

NeedsCompilation yes

Author Jeroen Ooms [cre, aut],
Damien Miller [cph],
Niels Provos [cph]

Maintainer Jeroen Ooms <`jeroen@berkeley.edu`>

Repository CRAN

Date/Publication 2018-01-26 09:08:42 UTC

R topics documented:

bcrypt *Bcrypt password hashing*

Description

Bcrypt is used for secure password hashing. The main difference with regular digest algorithms such as MD5 or SHA256 is that the bcrypt algorithm is specifically designed to be CPU intensive in order to protect against brute force attacks. The exact complexity of the algorithm is configurable via the `log_rounds` parameter. The interface is fully compatible with the Python one.

Usage

```
gensalt(log_rounds = 12)

hashpw(password, salt = gensalt())

checkpw(password, hash)
```

Arguments

`log_rounds`	integer between 4 and 31 that defines the complexity of the hashing, increasing the cost as `2^log_rounds`.
`password`	the message (password) to encrypt
`salt`	a salt generated with `gensalt`.
`hash`	the previously generated bcrypt hash to verify

Details

The `hashpw` function calculates a hash from a password using a random salt. Validating the hash is done by rehashing the password using the hash as a salt. The `checkpw` function is a simple wrapper that does exactly this.

`gensalt` generates a random text salt for use with `hashpw`. The first few characters in the salt string hold the bcrypt version number and value for `log_rounds`. The remainder stores 16 bytes of base64 encoded randomness for seeding the hashing algorithm.

Examples

```
# Secret message as a string
passwd <- "supersecret"

# Create the hash
hash <- hashpw(passwd)
hash

# To validate the hash
identical(hash, hashpw(passwd, hash))
```

```
# Or use the wrapper
checkpw(passwd, hash)

# Use varying complexity:
hash11 <- hashpw(passwd, gensalt(11))
hash12 <- hashpw(passwd, gensalt(12))
hash13 <- hashpw(passwd, gensalt(13))

# Takes longer to verify (or crack)
system.time(checkpw(passwd, hash11))
system.time(checkpw(passwd, hash12))
system.time(checkpw(passwd, hash13))
```

Index