Package 'secret'

May 9, 2026

Title Share Sensitive Information in R Packages

Version 1.1.0

Description Allow sharing sensitive information, for example passwords, 'API' keys, etc., in R packages, using public key cryptography.

License MIT + file LICENSE

LazyData true

URL https://github.com/gaborcsardi/secret#readme

BugReports https://github.com/gaborcsardi/secret/issues

RoxygenNote 7.1.0

Imports assertthat, openssl, rprojroot, curl, jsonlite

Suggests covr, mockery, testthat, knitr, rmarkdown, withr

Encoding UTF-8

VignetteBuilder knitr

NeedsCompilation no

Author Gábor Csárdi [aut, cre], Andrie de Vries [aut]

Maintainer Gábor Csárdi <csardi.gabor@gmail.com>

Repository CRAN

Date/Publication 2020-05-07 13:00:02 UTC

Contents

`secret-package` *Share Sensitive Information in R Packages.*

Description

Allow sharing sensitive information, for example passwords, API keys, or other information in R packages, using public key cryptography.

Details

A vault is a directory, typically inside an R package, that stores a number of secrets. Each secret is shared among a group of users. Users are identified using their public keys.

The package implements the following operations:

- Vault:
  - Creating a vault folder: `create_vault()`
  - Creating a package vault: `create_package_vault()`
- User management:
  - Adding a user: `add_user()`, `add_github_user()`.
  - Deleting a user: `delete_user()`.
  - Listing users: `list_users()`.
- Keys:
  - Reading local private key: `local_key()`
- Secrets:
  - Adding a secret: `add_secret()`.
  - Retrieving a secret: `get_secret()`.
  - Updating a secret: `update_secret()`.
  - Deleting a secret: `delete_secret()`.
  - List secrets: `list_secrets()`.
  - Sharing a secret: `share_secret()`. Query or set the set of users that have access to a secret.
  - Unsharing a secret: `unshare_secret()`

Author(s)

Gábor Csárdi and Andrie de Vries

`add_github_user` *Add a user via their GitHub username.*

Description

On GitHub, a user can upload multiple keys. This function will download the first key by default, but you can change this

Usage

```
add_github_user(github_user, email = NULL, vault = NULL, i = 1)
```

Arguments

`github_user`	User name on GitHub.
`email`	Email address of the github user. If NULL, constructs an email as `github-<<github_user>>`
`vault`	Vault location (starting point to find the vault). To create a vault, use `create_vault()` or `create_package_vault()`. If this is `NULL`, then `secret` tries to find the vault automatically: • If the `secret.vault` option is set to path, that is used as the starting point. • Otherwise, if the `R_SECRET_VAULT` environment variable is set to a path, that is used as a starting point. • Otherwise the current working directory is used as the starting point. If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the `inst/vault` folder is used within the package. If no vault can be found, an error is thrown.
`i`	Integer, indicating which GitHub key to use (if more than one GitHub key exists).

See Also

`add_travis_user()`

Other user functions: `add_travis_user()`, `add_user()`, `delete_user()`, `list_users()`

Examples

```
## Not run:
vault <- file.path(tempdir(), ".vault")
create_vault(vault)

add_github_user("hadley", vault = vault)
list_users(vault = vault)
```

```
delete_user("github-hadley", vault = vault)

## End(Not run)
```

`add_secret` *Add a new secret to the vault.*

Description

By default, the newly added secret is not shared with other users. See the users argument if you want to change this. You can also use `share_secret()` later, to specify the users that have access to the secret.

Usage

```
add_secret(name, value, users, vault = NULL)
```

Arguments

`name`	Name of the secret, a string that can contain alphanumeric characters, underscores, dashes and dots.
`value`	Value of the secret, an arbitrary R object that will be serialized using `base::serialize()`.
`users`	Email addresses of users that will have access to the secret. (See `add_user()`)
`vault`	Vault location (starting point to find the vault). To create a vault, use `create_vault()` or `create_package_vault()`. If this is `NULL`, then `secret` tries to find the vault automatically: • If the `secret.vault` option is set to path, that is used as the starting point. • Otherwise, if the `R_SECRET_VAULT` environment variable is set to a path, that is used as a starting point. • Otherwise the current working directory is used as the starting point. If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the `inst/vault` folder is used within the package. If no vault can be found, an error is thrown.

See Also

Other secret functions: `delete_secret()`, `get_secret()`, `list_owners()`, `list_secrets()`, `local_key()`, `share_secret()`, `unshare_secret()`, `update_secret()`

Examples

```
## Not run:
# The `secret` package contains some user keys for demonstration purposes.
# In this example, Alice shares a secret with Bob using a vault.

keys <- function(x){
```

```
  file.path(system.file("user_keys", package = "secret"), x)
}
alice_public  <- keys("alice.pub")
alice_private <- keys("alice.pem")
bob_public  <- keys("bob.pub")
bob_private <- keys("bob.pem")
carl_private <- keys("carl.pem")

# Create vault

vault <- file.path(tempdir(), ".vault")
if (dir.exists(vault)) unlink(vault) # ensure vault is empty
create_vault(vault)

# Add users with their public keys

add_user("alice", public_key = alice_public, vault = vault)
add_user("bob", public_key = bob_public, vault = vault)
list_users(vault = vault)

# Share a secret

secret <- list(username = "user123", password = "Secret123!")

add_secret("secret", value = secret, users = c("alice", "bob"),
           vault = vault)
list_secrets(vault = vault)

# Alice and Bob can decrypt the secret with their private keys
# Note that you would not normally have access to the private key
# of any of your collaborators!

get_secret("secret", key = alice_private, vault = vault)
get_secret("secret", key = bob_private, vault = vault)

# But Carl can't decrypt the secret

try(
  get_secret("secret", key = carl_private, vault = vault)
)

# Unshare the secret

unshare_secret("secret", users = "bob", vault = vault)
try(
  get_secret("secret", key = bob_private, vault = vault)
)


# Delete the secret

delete_secret("secret", vault = vault)
list_secrets(vault)
```

```
# Delete the users

delete_user("alice", vault = vault)
delete_user("bob", vault = vault)
list_users(vault)


## End(Not run)
```

`add_travis_user` *Add a user via their Travis repo.*

Description

On Travis, every repo has a private/public key pair. This function adds a user and downloads the public key from Travis.

Usage

```
add_travis_user(travis_repo, email, vault = NULL)
```

Arguments

`travis_repo`	Name of Travis repository, usually in a format `<<username>>/<<repo>>`
`email`	Email address of the user. This is used to identify users.
`vault`	Vault location (starting point to find the vault). To create a vault, use `create_vault()` or `create_package_vault()`. If this is `NULL`, then `secret` tries to find the vault automatically: • If the `secret.vault` option is set to path, that is used as the starting point. • Otherwise, if the `R_SECRET_VAULT` environment variable is set to a path, that is used as a starting point. • Otherwise the current working directory is used as the starting point. If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the `inst/vault` folder is used within the package. If no vault can be found, an error is thrown.

See Also

Other user functions: `add_github_user()`, `add_user()`, `delete_user()`, `list_users()`

Examples

```
## Not run:
vault <- file.path(tempdir(), ".vault")
create_vault(vault)

add_travis_user("gaborcsardi/secret", vault = vault)
list_users(vault = vault)
delete_user("travis-gaborcsardi-secret", vault = vault)

## End(Not run)
```

`add_user` *Add a new user to the vault.*

Description

By default the new user does not have access to any secrets. See `add_secret()` or `share_secret()` to give them access.

Usage

```
add_user(email, public_key, vault = NULL)
```

Arguments

`email`	Email address of the user. This is used to identify users.
`public_key`	Public key of the user. This is used to encrypt the secrets for the different users. It can be • a string containing a PEM, • a file name that points to a PEM file, • a `pubkey` object created via the `openssl` package.
`vault`	Vault location (starting point to find the vault). To create a vault, use `create_vault()` or `create_package_vault()`. If this is `NULL`, then `secret` tries to find the vault automatically: • If the `secret.vault` option is set to path, that is used as the starting point. • Otherwise, if the `R_SECRET_VAULT` environment variable is set to a path, that is used as a starting point. • Otherwise the current working directory is used as the starting point. If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the `inst/vault` folder is used within the package. If no vault can be found, an error is thrown.

See Also

Other user functions: `add_github_user()`, `add_travis_user()`, `delete_user()`, `list_users()`

Examples

```
## Not run:
# The `secret` package contains some user keys for demonstration purposes.
# In this example, Alice shares a secret with Bob using a vault.

keys <- function(x){
  file.path(system.file("user_keys", package = "secret"), x)
}
alice_public  <- keys("alice.pub")
alice_private <- keys("alice.pem")
bob_public  <- keys("bob.pub")
bob_private <- keys("bob.pem")
carl_private <- keys("carl.pem")

# Create vault

vault <- file.path(tempdir(), ".vault")
if (dir.exists(vault)) unlink(vault) # ensure vault is empty
create_vault(vault)

# Add users with their public keys

add_user("alice", public_key = alice_public, vault = vault)
add_user("bob", public_key = bob_public, vault = vault)
list_users(vault = vault)

# Share a secret

secret <- list(username = "user123", password = "Secret123!")

add_secret("secret", value = secret, users = c("alice", "bob"),
           vault = vault)
list_secrets(vault = vault)

# Alice and Bob can decrypt the secret with their private keys
# Note that you would not normally have access to the private key
# of any of your collaborators!

get_secret("secret", key = alice_private, vault = vault)
get_secret("secret", key = bob_private, vault = vault)

# But Carl can't decrypt the secret

try(
  get_secret("secret", key = carl_private, vault = vault)
)

# Unshare the secret

unshare_secret("secret", users = "bob", vault = vault)
try(
  get_secret("secret", key = bob_private, vault = vault)
```

```
)


# Delete the secret

delete_secret("secret", vault = vault)
list_secrets(vault)

# Delete the users

delete_user("alice", vault = vault)
delete_user("bob", vault = vault)
list_users(vault)


## End(Not run)
```

`create_package_vault` *Create a vault, as a folder or in an R package.*

Description

A vault is a folder that contains information about users and the secrets they share. You can create a vault as either a standalone folder, or as part of a package.

Usage

```
create_package_vault(path = ".")

create_vault(path)
```

Arguments

`path`	Path to the R package. A file or directory within the package is fine, too. If the vault directory already exists, a message is given, and the function does nothing.

Details

A vault is a folder with a specific structure, containing two directories: `users` and `secrets`.

In `users`, each file contains a public key in PEM format. The name of the file is the identifier of the key, an arbitrary name. We suggest that you use email addresses to identify public keys. See also `add_user()`.

In `secrets`, each secret is stored in its own directory. The directory of a secret contains

1. the secret, encrypted with its own AES key, and
2. the AES key, encrypted with the public keys of all users that have access to the secret, each in its own file.

To add a secret, see `add_secret()`

Value

The directory of the vault, invisibly.

Creating a package folder

When you create a vault in a package, this vault is stored in the `inst/vault` directory of the package during development. At package install time, this folder is copied to the `vault` folder.

See Also

`add_user()`, `add_secret()`

Examples

```
## Not run:
# The `secret` package contains some user keys for demonstration purposes.
# In this example, Alice shares a secret with Bob using a vault.

keys <- function(x){
  file.path(system.file("user_keys", package = "secret"), x)
}
alice_public  <- keys("alice.pub")
alice_private <- keys("alice.pem")
bob_public  <- keys("bob.pub")
bob_private <- keys("bob.pem")
carl_private <- keys("carl.pem")

# Create vault

vault <- file.path(tempdir(), ".vault")
if (dir.exists(vault)) unlink(vault) # ensure vault is empty
create_vault(vault)

# Add users with their public keys

add_user("alice", public_key = alice_public, vault = vault)
add_user("bob", public_key = bob_public, vault = vault)
list_users(vault = vault)

# Share a secret

secret <- list(username = "user123", password = "Secret123!")

add_secret("secret", value = secret, users = c("alice", "bob"),
           vault = vault)
list_secrets(vault = vault)

# Alice and Bob can decrypt the secret with their private keys
# Note that you would not normally have access to the private key
# of any of your collaborators!

get_secret("secret", key = alice_private, vault = vault)
```

```
get_secret("secret", key = bob_private, vault = vault)

# But Carl can't decrypt the secret

try(
  get_secret("secret", key = carl_private, vault = vault)
)

# Unshare the secret

unshare_secret("secret", users = "bob", vault = vault)
try(
  get_secret("secret", key = bob_private, vault = vault)
)


# Delete the secret

delete_secret("secret", vault = vault)
list_secrets(vault)

# Delete the users

delete_user("alice", vault = vault)
delete_user("bob", vault = vault)
list_users(vault)


## End(Not run)
```

`delete_secret` *Remove a secret from the vault.*

Description

Remove a secret from the vault.

Usage

```
delete_secret(name, vault = NULL)
```

Arguments

`name`	Name of the secret to delete.
`vault`	Vault location (starting point to find the vault). To create a vault, use `create_vault()` or `create_package_vault()`. If this is `NULL`, then `secret` tries to find the vault automatically: • If the `secret.vault` option is set to path, that is used as the starting point.

- Otherwise, if the R_SECRET_VAULT environment variable is set to a path, that is used as a starting point.
- Otherwise the current working directory is used as the starting point.

If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the inst/vault folder is used within the package. If no vault can be found, an error is thrown.

See Also

Other secret functions: add_secret(), get_secret(), list_owners(), list_secrets(), local_key(), share_secret(), unshare_secret(), update_secret()

---

delete_user *Delete a user.*

Description

It also removes access of the user to all secrets, so if the user is re-added again, they will not have access to any secrets.

Usage

```
delete_user(email, vault = NULL)
```

Arguments

email	Email address of the user.
vault	Vault location (starting point to find the vault). To create a vault, use create_vault() or create_package_vault(). If this is NULL, then secret tries to find the vault automatically: • If the secret.vault option is set to path, that is used as the starting point. • Otherwise, if the R_SECRET_VAULT environment variable is set to a path, that is used as a starting point. • Otherwise the current working directory is used as the starting point. If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the inst/vault folder is used within the package. If no vault can be found, an error is thrown.

See Also

Other user functions: add_github_user(), add_travis_user(), add_user(), list_users()

get_github_key *Get the SSH public key of a GitHub user*

Description

Get the SSH public key of a GitHub user

Usage

```
get_github_key(github_user, i = 1)
```

Arguments

github_user	GitHub username.
i	Which key to get, in case the user has multiple keys. `get_github_key()` retrieves the first key by default.

Value

Character scalar.

get_secret *Retrieve a secret from the vault.*

Description

Retrieve a secret from the vault.

Usage

```
get_secret(name, key = local_key(), vault = NULL)
```

Arguments

name	Name of the secret.
key	The private RSA key to use. It defaults to the current user's default key.
vault	Vault location (starting point to find the vault). To create a vault, use `create_vault()` or `create_package_vault()`. If this is `NULL`, then `secret` tries to find the vault automatically: • If the `secret.vault` option is set to path, that is used as the starting point. • Otherwise, if the `R_SECRET_VAULT` environment variable is set to a path, that is used as a starting point. • Otherwise the current working directory is used as the starting point. If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the `inst/vault` folder is used within the package. If no vault can be found, an error is thrown.

See Also

Other secret functions: add_secret(), delete_secret(), list_owners(), list_secrets(), local_key(), share_secret(), unshare_secret(), update_secret()

Examples

```
## Not run:
# The `secret` package contains some user keys for demonstration purposes.
# In this example, Alice shares a secret with Bob using a vault.

keys <- function(x){
  file.path(system.file("user_keys", package = "secret"), x)
}
alice_public  <- keys("alice.pub")
alice_private <- keys("alice.pem")
bob_public  <- keys("bob.pub")
bob_private <- keys("bob.pem")
carl_private <- keys("carl.pem")

# Create vault

vault <- file.path(tempdir(), ".vault")
if (dir.exists(vault)) unlink(vault) # ensure vault is empty
create_vault(vault)

# Add users with their public keys

add_user("alice", public_key = alice_public, vault = vault)
add_user("bob", public_key = bob_public, vault = vault)
list_users(vault = vault)

# Share a secret

secret <- list(username = "user123", password = "Secret123!")

add_secret("secret", value = secret, users = c("alice", "bob"),
           vault = vault)
list_secrets(vault = vault)

# Alice and Bob can decrypt the secret with their private keys
# Note that you would not normally have access to the private key
# of any of your collaborators!

get_secret("secret", key = alice_private, vault = vault)
get_secret("secret", key = bob_private, vault = vault)

# But Carl can't decrypt the secret

try(
  get_secret("secret", key = carl_private, vault = vault)
)
```

```
# Unshare the secret

unshare_secret("secret", users = "bob", vault = vault)
try(
  get_secret("secret", key = bob_private, vault = vault)
)


# Delete the secret

delete_secret("secret", vault = vault)
list_secrets(vault)

# Delete the users

delete_user("alice", vault = vault)
delete_user("bob", vault = vault)
list_users(vault)


## End(Not run)
```

`get_travis_key` *Retrieve the public key of a Travis CI repository*

Description

Retrieve the public key of a Travis CI repository

Usage

```
get_travis_key(travis_repo)
```

Arguments

`travis_repo`	The repository slug, e.g. `gaborcsardi/secret`.

Value

Character scalar, the key. If the repository does not exist, or it is not user in Travis CI, an HTTP 404 error is thrown.

list_owners *List users that have access to a secret*

Description

List users that have access to a secret

Usage

```
list_owners(name, vault = NULL)
```

Arguments

name	Name of the secret, a string that can contain alphanumeric characters, underscores, dashes and dots.
vault	Vault location (starting point to find the vault). To create a vault, use create_vault() or create_package_vault(). If this is NULL, then secret tries to find the vault automatically: • If the secret.vault option is set to path, that is used as the starting point. • Otherwise, if the R_SECRET_VAULT environment variable is set to a path, that is used as a starting point. • Otherwise the current working directory is used as the starting point. If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the inst/vault folder is used within the package. If no vault can be found, an error is thrown.

See Also

Other secret functions: add_secret(), delete_secret(), get_secret(), list_secrets(), local_key(), share_secret(), unshare_secret(), update_secret()

list_secrets *List all secrets.*

Description

Returns a data frame with secrets and emails that these are shared with. The emails are in a list-column, each element of the email column is a character vector.

Usage

```
list_secrets(vault = NULL)
```

Arguments

vault	Vault location (starting point to find the vault). To create a vault, use create_vault() or create_package_vault(). If this is NULL, then secret tries to find the vault automatically: • If the secret.vault option is set to path, that is used as the starting point. • Otherwise, if the R_SECRET_VAULT environment variable is set to a path, that is used as a starting point. • Otherwise the current working directory is used as the starting point. If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the inst/vault folder is used within the package. If no vault can be found, an error is thrown.

Value

data.frame

See Also

Other secret functions: add_secret(), delete_secret(), get_secret(), list_owners(), local_key(), share_secret(), unshare_secret(), update_secret()

---

list_users *List users*

Description

List users

Usage

```
list_users(vault = NULL)
```

Arguments

vault	Vault location (starting point to find the vault). To create a vault, use create_vault() or create_package_vault(). If this is NULL, then secret tries to find the vault automatically: • If the secret.vault option is set to path, that is used as the starting point. • Otherwise, if the R_SECRET_VAULT environment variable is set to a path, that is used as a starting point. • Otherwise the current working directory is used as the starting point. If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the inst/vault folder is used within the package. If no vault can be found, an error is thrown.

See Also

Other user functions: add_github_user(), add_travis_user(), add_user(), delete_user()

`local_key` *Read local secret key.*

Description

Reads a local secret key from disk. The location of this file can be specified in the USER_KEY environment variable. If this environment variable does not exist, then attempts to read the key from:

- ~/.ssh/id_rsa, and
- ~/.ssh/id_rsa.pem.

Usage

```
local_key()
```

Details

The location of the key is defined by:

```
Sys.getenv("USER_KEY")
```

To use a local in a different location, set an environment variable:

```
Sys.setenv(USER_KEY = "path/to/private/key")
```

See Also

Other secret functions: add_secret(), delete_secret(), get_secret(), list_owners(), list_secrets(), share_secret(), unshare_secret(), update_secret()

`share_secret` *Share a secret among some users.*

Description

Use this function to extend the set of users that have access to a secret. The calling user must have access to the secret as well.

Usage

```
share_secret(name, users, key = local_key(), vault = NULL)
```

Arguments

name	Name of the secret, a string that can contain alphanumeric characters, underscores, dashes and dots.
users	addresses of users that will have access to the secret. (See add_user()).
key	Private key that has access to the secret. (I.e. its corresponding public key is among the vault users.)
vault	Vault location (starting point to find the vault). To create a vault, use create_vault() or create_package_vault(). If this is NULL, then secret tries to find the vault automatically: • If the secret.vault option is set to path, that is used as the starting point. • Otherwise, if the R_SECRET_VAULT environment variable is set to a path, that is used as a starting point. • Otherwise the current working directory is used as the starting point. If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the inst/vault folder is used within the package. If no vault can be found, an error is thrown.

See Also

unshare_secret(), list_owners() to list users that have access to a secret.

Other secret functions: add_secret(), delete_secret(), get_secret(), list_owners(), list_secrets(), local_key(), unshare_secret(), update_secret()

---

unshare_secret *Unshare a secret among some users.*

Description

Use this function to restrict the set of users that have access to a secret. Note that users may still have access to the secret, through version control history, or if they have a copy of the project. They will not have access to future values of the secret, though.

Usage

```
unshare_secret(name, users, vault = NULL)
```

Arguments

name	Name of the secret, a string that can contain alphanumeric characters, underscores, dashes and dots.
users	Email addresses of users that will have access to the secret. (See add_user())
vault	Vault location (starting point to find the vault). To create a vault, use create_vault() or create_package_vault(). If this is NULL, then secret tries to find the vault automatically:

- If the `secret.vault` option is set to path, that is used as the starting point.
- Otherwise, if the `R_SECRET_VAULT` environment variable is set to a path, that is used as a starting point.
- Otherwise the current working directory is used as the starting point.

If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the `inst/vault` folder is used within the package. If no vault can be found, an error is thrown.

See Also

`share_secret()`

Other secret functions: `add_secret()`, `delete_secret()`, `get_secret()`, `list_owners()`, `list_secrets()`, `local_key()`, `share_secret()`, `update_secret()`

`update_secret` *Update a secret in the vault.*

Description

Update a secret in the vault.

Usage

```
update_secret(name, value, key = local_key(), vault = NULL)
```

Arguments

`name`	Name of the secret.
`value`	Value of the secret, an arbitrary R object that will be serialized using `base::serialize()`.
`key`	The private RSA key to use. It defaults to the current user's default key.
`vault`	Vault location (starting point to find the vault). To create a vault, use `create_vault()` or `create_package_vault()`. If this is `NULL`, then `secret` tries to find the vault automatically: • If the `secret.vault` option is set to path, that is used as the starting point. • Otherwise, if the `R_SECRET_VAULT` environment variable is set to a path, that is used as a starting point. • Otherwise the current working directory is used as the starting point. If the starting point is a vault, that is used. Otherwise, if the starting point is in a package tree, the `inst/vault` folder is used within the package. If no vault can be found, an error is thrown.

See Also

Other secret functions: `add_secret()`, `delete_secret()`, `get_secret()`, `list_owners()`, `list_secrets()`, `local_key()`, `share_secret()`, `unshare_secret()`

Index