Package 'sship'

August 5, 2022

Title Tool for Secure Shipment of Content

Version 0.8.0

Maintainer Are Edvardsen <biorakel@gmail.com>

Description Convenient tools for exchanging files securely from within R. By encrypting the content safe passage of files (shipment) can be provided by common but insecure carriers such as ftp and email. Based on asymmetric cryptography no management of shared secrets is needed to make a secure shipment as long as authentic public keys are available. Public keys used for secure shipments may also be obtained from external providers as part of the overall process. Transportation of files will require that relevant services such as ftp and email servers are available.

License GPL-3

Encoding UTF-8

Imports httr, jsonlite, openssl, RCurl, yaml

Suggests httptest, httpuv, rmarkdown, testthat (>= 2.1.0), withr

RoxygenNote 7.1.2

URL https://github.com/Rapporteket/sship

BugReports https://github.com/Rapporteket/sship/issues

NeedsCompilation no

Author Are Edvardsen [aut, cre] (<https://orcid.org/0000-0002-5210-3656>)

Repository CRAN

Date/Publication 2022-08-05 12:20:02 UTC

R topics documented:

`config` *Functions handling sship R package config*

Description

Functions handling sship R package config

Usage

```
create_config(dir = ".")

write_config(config, dir = ".", filename = "_sship.yml")

get_config(dir = ".")

check_config(config)
```

Arguments

`dir`	string providing path to configuration file
`config`	list containing configuration
`filename`	string defining config filename

Value

A status message or list of config

Examples

```
# Create a new config file from package default
create_config(dir = tempdir())

# Get config
config <- get_config(system.file("sship.yml", package = "sship"))

# Check if config is valid
check_config(config)

# Write config to file
write_config(config, dir = tempdir())
```

dec *Unpack shipment and decrypt content*

Description

This function tries to reverse the process of enc and hence depend on the conventions used there.

Usage

```
dec(tarfile, keyfile = "~/.ssh/id_rsa", target_dir = ".")
```

Arguments

tarfile	Character string providing full path to the gzip-compressed tarball holding the shipment payload, including encrypted files.
keyfile	Character string providing the full path to the private RSA key to be used for decryption of the encrypted key that is part of the shipment. Default value is set to `~/.ssh/id_rsa` which is the usual path for unix type operating systems.
target_dir	Character string providing the full path to where the decrypted file is to be written. Defaults to the current directory `"."`, *e.g.* where this function is being called from.

Details

Some of the functions used here might be vulnerable to differences between systems running R. Possible caveats may be the availability of the (un)tar-function and how binary streams/files are treated.

Value

Invisibly a character string providing the file path of the decrypted file.

See Also

enc

Examples

```
# Please note that these examples will write files to a local temporary
# directory.

## Make temporary workspace
wd <- tempdir()

## Make a private-public key pair named "id_rsa" and "id_rsa.pub"
keygen(directory = wd, type = "rsa", overwrite_existing = TRUE)

## Make a secured (encrypted) file
```

```
saveRDS(iris, file = file.path(wd, "secret.rds"), ascii = TRUE)
pubkey <- readLines(file.path(wd, "id_rsa.pub"))
secure_secret_file <-
  enc(filename = file.path(wd, "secret.rds"),
      pubkey_holder = NULL,
      pubkey = pubkey)

## Decrypt secured file using the private key
secret_file <-
  dec(tarfile = secure_secret_file,
      keyfile = file.path(wd, "id_rsa"),
      target_dir = wd)
```

---

enc *Encryption of shipment content*

Description

Various functions and helper functions to establish encrypted files. To secure the content (any file) the Advanced Encryption Standard (AES) is applied with an ephemeral key consisting of 256 random bits. This key is only used once for encryption (and then one more time during decryption at a later stage). A random 128 bit initialization vector (iv) is also applied during encryption. There is no extra security gain in this since the key will never be re-used for encryption/decryption. So, just for good measures then :-) After the content has been encrypted the key itself is encrypted by applying a public key offered by the recipient. This key is obtained from a public provider. Currently, GitHub is the only option. The three files: encrypted content, the encrypted key and the (cleartext) iv is then bundled into a tarball ready for shipment.

Usage

```
enc_filename(filename)

make_pubkey_url(pubkey_holder = "github", pid)

get_pubkey(pubkey_holder, pid)

enc(filename, pubkey_holder, pid, pubkey = NULL)
```

Arguments

`filename`	Character string with fully qualified path to a file.
`pubkey_holder`	Character string defining the provider of the public key used for encryption of the symmetric key. Currently, 'github' is the only valid pubkey holder. If a local pubkey is to be used (see parameter `pubkey`, `pubkey_holder` may be set to NULL or some other value.
`pid`	Character string uniquely defining the user at `pubkey_holder` who is also the owner of the public key.
`pubkey`	Character string representing a valid public key. Default is NULL in which case the key will be obtained as per `pubkey_holder`.

Details

Encrypted files can be decrypted outside R using the OpenSSL library. Both the key and the initialization vector (iv) are binary and this method uses the key directly (and not a [hashed] passphrase). OpenSSL decryption need to be fed the key (and iv) as a string of hex digits. Methods for conversion from binary to hex may vary between systems. Below, a bash shell (unix) example is given

Step 1: decrypt symmetric key (open envelope) using a private key

```
openssl rsautl -decrypt -inkey ~/.ssh/id_rsa -in key.enc -out key
```

Step 2: decrypt content by key obtained in step 1, also converting key and iv to strings of hexadecimal digits

```
openssl aes-256-cbc -d -in data.csv.enc -out data.csv \
-K $(hexdump -e '32/1 "%02x"' key) -iv $(hexdump -e '16/1 "%02x"' iv)
```

Value

Character string providing a filename or a key

See Also

dec

Examples

```
# Please note that these examples will write files to a local temporary
# directory.

## Define temporary working directory and a secret file name
wd <- tempdir()
secret_file_name <- "secret.rds"

## Add content to the secret file
saveRDS(iris, file = file.path(wd, secret_file_name), ascii = TRUE)

## Make a private-public key pair named "id_rsa" and "id_rsa.pub"
keygen(directory = wd, type = "rsa", overwrite_existing = TRUE)

## Load public key
pubkey <- readLines(file.path(wd, "id_rsa.pub"))

## Make a secured file (ready for shipment)
secure_secret_file <- enc(filename = file.path(wd, "secret.rds"),
                          pubkey_holder = NULL, pubkey = pubkey)
```

`github` *Make calls to the github API*

Description

Provides a structured list of the specified resource from the the github API.

Usage

```
gh(path, proxy_url = NULL, token = NULL)

github_api(path, proxy_url = NULL, token = NULL)

rate_limit(proxy_url = NULL, token = NULL)
```

Arguments

`path`	Character string with path to the API resource.
`proxy_url`	Character string defining a network proxy in the form host:port. Default is NULL in which case the API call will not use a proxy.
`token`	Character string holding a github personal access token (PAT) to be used for requests that requires authorization. Default value is NULL in which case the request will be unauthorized unless PAT can be obtained from the environmental variable GITHUB_PAT.

Details

For most use cases only `gh()` will be relevant. The helper function `github_api()` do the actual lifting while `rate_limit()` handles API rate limits.

Value

A list of class github_api containing the parsed content, API resource path and the response object. For `rate_limit()` the path is always "/rate_limit" and can hence be used to detect if the limit is exceeded (without being counted as a request itself). If the allowed API rate is exceeded `gh()` will return a message stating the fact and simple suggestions on how to remedy the problem.

Examples

```
## Get all branches of a repository. If the api rate limit is exceeded this
## function will return NULL and an informative message
gh("repos/Rapporteket/sship/branches")

## helper functions that will normally not be used
github_api("/rate_limit")
rate_limit()
```

keygen *Make private-public key pair*

Description

Just for the convenience of it, make a key pair that may be used alongside sship. Please note that by default the private key will not be protected by a password.

Usage

```
keygen(
  directory = "~/.ssh",
  type = "rsa",
  password = NULL,
  overwrite_existing = FALSE
)
```

Arguments

directory	Character string with path to directory where the key pair will be written. Default is "~/.ssh".
type	Character string defining the key type. Must be one of `c("rsa", "dsa", "ecdsa", "x25519", "ed25529")`. Key lengths are set to the default as defined in the *openssl*-package. If the key-pair is to be used with this package make sure that type is set to "rsa".
password	Character string with password to protect the private key. Default value is NULL in which case the private key will not be protected by a password
overwrite_existing	Logical whether existing key files with the similar names should be overwritten. Set to FALSE by default.

Value

Nothing will be returned from this function, but a message containing the directory where the keys were written is provided

Examples

```
keygen(directory = tempdir(), overwrite_existing = TRUE)
```

pubkey_filter *Filter ssh public keys by type*

Description

From a vector of ssh public keys, return those that are of a given type.

Usage

```
pubkey_filter(keys, type)
```

Arguments

keys	Vector of strings representing ssh public keys.
type	Character string defining the ssh public key type that will pass the filter. Relevant values are strings returned by `attributes(openssl::read_pubkey(pubkey))$class[2]`, *e.g.* "rsa" and "dsa".

Value

A vector of strings representing (filtered) public keys.

Examples

```
## make ssh public key strings
rsa_pubkey <- openssl::write_ssh(openssl::rsa_keygen()$pubkey)
dsa_pubkey <- openssl::write_ssh(openssl::dsa_keygen()$pubkey)

## filter keys by type
pubkey <- pubkey_filter(c(rsa_pubkey, dsa_pubkey), "rsa")
identical(pubkey, rsa_pubkey)
```

ship *Secure cargo and make shipment (secure shipment)*

Description

First, the content (a file) is encrypted and packed and then shipped to the recipient using the specified vessel (transportation method). If the given vessel is not available the function return an error. Optionally, a declaration can also be associated with the shipment and dispatched immediately after the actual cargo.

Usage

```
sship(content, recipient, pubkey_holder, vessel, declaration = "")

dispatch(recipient, vessel, cargo)

dispatchable(recipient, vessel)

make_url(recipient, vessel)

make_opts(recipient, vessel)
```

Arguments

content	Character string: the full path to the file to be shipped
recipient	Character string: user name uniquely defining the recipient both in terms of the public key used for securing the content and any identity control upon docking. See also *Details*.
pubkey_holder	Character string: the holder of the (recipient's) public key. Currently, the only viable option here is 'github'.
vessel	Character string: means of transportation. Currently one of 'ssh' or 'ftp'.
declaration	Character string: the name of an empty file to be associated with shipment of the cargo itself and dispatched immediately after. The most likely use case is for the recipient to check for this file being present before picking up the cargo itself. Default value is "" in which case no declaration will be used.
cargo	Character vector: all items associated with the current shipment. Used only internally.

Details

Most likely access control will be enforced before docking of the shipment can commence. For each recipient a list of available vessels (transport methods) is defined and must include relevant credentials. Functions used here rely on local configuration (`sship.yml`) to access such credentials.

Value

TRUE if successful

See Also

enc

Index