Guide to the getPass Package

July 18, 2017

Drew Schmidt
wrathematics@gmail.com

Version 0.2-1

Disclaimer

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. The findings and conclusions in this article have not been formally disseminated by the U.S. Department of Health & Human Services nor by the U.S. Department of Energy, and should not be construed to represent any determination or policy of University, Agency, Administration and National Laboratory.

This manual may be incorrect or out-of-date. The author(s) assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

This publication was typeset using LaTeX.

Contents

1 Introduction

getPass [5] is an R package for reading user input in R with masking. There is one exported function, getPass(), which will behave as R's readline() but with masked input. You can pass a message to the password input via the msg argument, similar to the prompt argument in readline().

1.1 Installation

You can install the stable version from CRAN using the usual install.packages():

```
install.packages("getPass")
```

The development version is maintained on GitHub. You can install this version using any of the well-known installer packages available to R:

```
### Pick your preference
devtools::install_github("wrathematics/getPass")
ghit::install_github("wrathematics/getPass")
remotes::install_github("wrathematics/getPass")
```

2 Password Reading

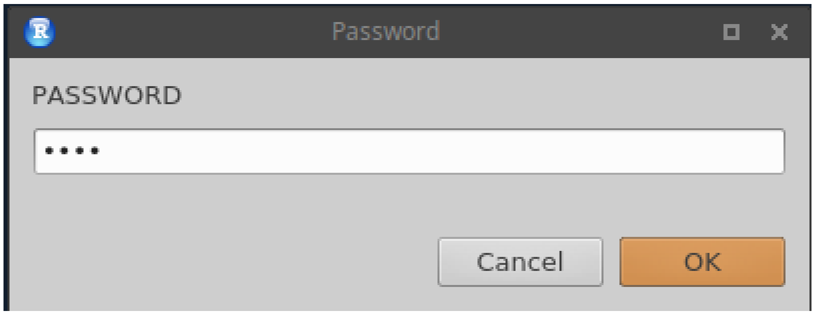
Using the package should mostly amount to calling getPass::getPass(). Currently there are two arguments to getPass(). By setting the msg parameter, you can change what is printed in the password dialogue box:

```
getPass()
## PASSWORD: ****
## [1] "asdf"

getPass(msg="")
## ****
## [1] "asdf"

getPass(msg="shh, it's a secret! ")
## shh, it's a secret! ****
## [1] "asdf"
```

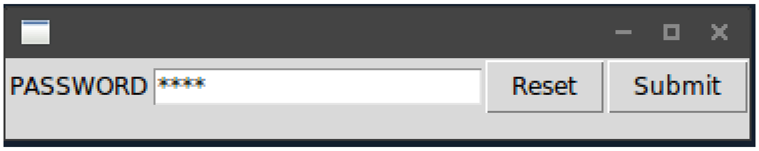
Finally, there is the forcemask flag, which indicates if reading without masking should be possible. By default, if one is running under an environment that does not allow reading with masking, then a warning message will be printed, and R's ordinary readline() will be used. However, if this flag is set to TRUE, then the function will stop with an error.

2.1 Interfaces

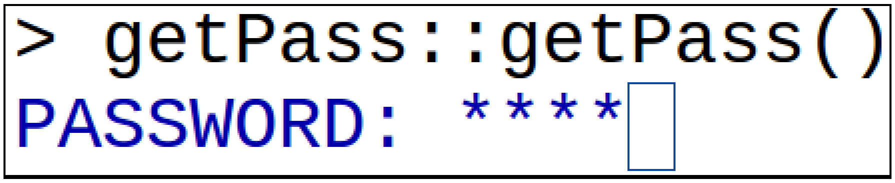
The form that password input takes will vary based on how you interface to R (with implementation details below). If you use **RStudio**, it will look something like this:



If you use **RGui** on Windows or **R.app** (if **tcltk** is supported; see Section 4 below), it will look like:



Finally, if you use the terminal (any OS), it will look like:



```
> getPass::getPass()
PASSWORD: ****
```

We believe this covers pretty much everyone. One notable exclusion is emacs in an environment without **tcltk**. Due to how it handles buffers, I believe it *can't* be supported. If that is incorrect, please let us know!

3 Password Hashing

3.1 The Short Version

After reading in a password that you intend to store (or in some way "pass around"), always hash it using a cryptographic hashing function. Some options for hashing with R include:

- **argon2** `https://cran.r-project.org/package=argon2` [4]
- **sodium** `https://cran.r-project.org/package=sodium` [3]
- **bcrypt** `https://cran.r-project.org/package=bcrypt` [1]
- **openssl** `https://cran.r-project.org/package=openssl` [2]

3.2 The Long(er) Version

In an effort to keep the package as minimal as possible, we do not include any methods for hashing passwords. However, the suggested package **argon2** [4] contains an implementation of the *argon2()* secure password hashing function. Many experts (of which I am not one) have written at length about this topic; and it can quickly get kind of complicated and mathy. The basic idea is: don't store passwords as plaintext. We can use a secure hash function to hash the password, basically turning the input string into a new "garbled" string. Hash functions are hard to "invert", so you can know which hash function I used and know the output, and still (hopefully) not recover the original string.

We can quickly handle this problem without having to think very hard. Say you used **getPass** to read a password into the variable `pass`:

```r
pass
## [1] "myPassw0rd!"
```

An excellent choice to be sure. This is the "plaintext". We can hash it with a call to the **argon2** packages's `pw_hash()` function:

```r
hash <- argon2::pw_hash(pass)
hash
## [1] "$argon2i$v=19$m=8192,t=16,p=2$JeV26p9ZmnlFyHKUWCe/46
   E3q2dtaXzuH06L4Qg15IEgkNr0aw0I5TnxI+6yLFRmLUZG6R4GJK0BTAkZhKgItg$
   MdafxeYEstYyT3RWyj2DDDcBAhfi8dE30tn6L1/
   Xaaus5su5Xiq2fdnD2zCK39DXTUyGWs0TTTZzGxKw104mtg"
attr(,"hashtype")
[1] "argon2"
```

Now say you need to validate a password that's been entered against the hashed password. All you need to do is call `pw_check()`:

```r
argon2::pw_check(hash, pass)
## [1] TRUE
argon2::pw_check(hash, "password")
## [1] FALSE
argon2::pw_check(hash, "1234")
## [1] FALSE
```

So inside of a user-facing application, the process might look something like this:

```r
user_pw <- getPass::getPass()
hash_pw <- argon2::pw_hash(user_pw)
store_user_pw(hash_pw) # pseudocode, but you get the idea
```

There are good reasons to prefer **argon2**: it is lightweight (with no package or system dependencies) and it is believed to be very secure. However, there are other options available in R, including the **bcrypt**, **sodium**, and **openssl** packages.

4 Implementation Details

4.1 RStudio

To use this with RStudio, you need:

- RStudio desktop version >= 0.99.879.
- The rstudioapi package version >= 0.5.

In this case, the `getPass()` function wraps the **rstudioapi** function `askForPassword()`.

4.2 Command Line

Here, the input reader is custom C code. It has been tested successfully on Windows (in the "RTerm" session), Mac (in the terminal, not R.app which will not work!), Linux, and FreeBSD. The maximum length for a password in this case is 255 characters.

On Windows, the reader is just `_getch()`. On 'nix environments (Mac, Linux, ...), masking is made possible via `tcsetattr()`. Special handling for each is provided for handling `ctrl+c` and backspace.

If you discover a problem using this, please file an issue report.

4.3 RGui (Windows)

If you use RGui (the Windows R GUI), then this should use the **tcltk** package. I don't think it's actually possible for **tcltk** to be unavailable on Windows, so if you are an RGui user and have trouble with this, please file an issue report.

4.4 R.app (Mac)

You will need to install dependencies for the **tcltk** package. I'm not completely sure what this process involves for Macs; if you know, please let us know. If **tcltk** is unavailable, then it will use the "unsupported" method below.

4.5 Other/Unsupported Platforms

When a platform is unsupported, the function will optionally default to use R's `readline()` (without masking!) with a warning communicated to the user, or it can stop with an error.

5 Acknowledgements

We thank Kevin Ushey for his assistance in answering questions in regard to supporting RStudio.

The development for this package was supported in part by the project *Harnessing Scalable Libraries for Statistical Computing on Modern Architectures and Bringing Statistics to Large Scale Computing* funded by the National Science Foundation Division of Mathematical Sciences under Grant No. 1418195.

References

[1] Jeroen Ooms. *bcrypt: 'Blowfish' Password Hashing Algorithm*, 2015. R package version 0.2.

[2] Jeroen Ooms. *openssl: Toolkit for Encryption, Signatures and Certificates Based on OpenSSL*, 2016. R package version 0.9.6.

[3] Jeroen Ooms. *sodium: A Modern and Easy-to-Use Crypto Library*, 2017. R package version 1.1.

[4] Drew Schmidt. argon2: Secure password hashing, 2017. R package version 0.2-0.

[5] Drew Schmidt and Wei-Chen Chen. getPass: Masked user input, 2017. R package version 0.2-1.