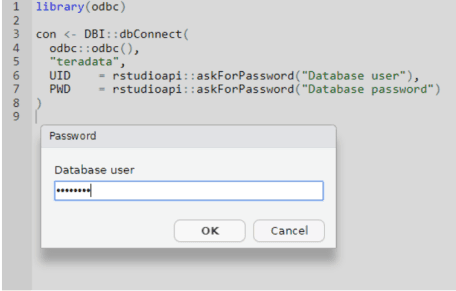
When accessing an API or database in R, it is often necessary to provide credentials such as a login name and password. You may find yourself being prompted with something like this:



When writing an R script that requires a user to provide credentials, you will want a way to have the script prompt the user or, better yet, programatically provided the credentials in the R script. Either way, be careful! You don’t want to put your credentials out there in the clear for all the world to see. Best practices[1](https://rviews.rstudio.com/2019/03/21/how-to-avoid-publishing-credentials-in-your-code/#fn:3) emphatically state:

As with every programming language, it is important to **avoid publishing code with your credentials in plain text**.

So, how can we provide credentials without putting them in the script itself?

I will focus on two cases:

* simply prompting for credentials via rstudioapi
  + suitable for simple credential management
* storing sets of encrypted credentials in a local file via the R keyring package
  + suitable for more complicated credential management

**Prompting for a username and password**

If an R Script requires only one set of credentials and those credentials are easy to remember, it may be easiest to prompt the user for them using rstudioapi. A typical example would be prompting users for their username and password to access a corporate database:

username <- rstudioapi::askForPassword("Database username")

password <- rstudioapi::askForPassword("Database password")

This method may also be convenient if the user’s credentials tend to change over time.

**The R Keyring package**

A more sophisticated option is to use the R keyring package to store and access encrypted credentials locally. This might be more suitable if multiple credentials exist to access a variety of services (think multiple access tokens). With keyring, one password unlocks the keyring which then provides access to all the credentials.

To use the keyring package, a user only needs to to install and load the package[2](https://rviews.rstudio.com/2019/03/21/how-to-avoid-publishing-credentials-in-your-code/#fn:2) and define three strings: the keyring name, a keyring service, and the username that we want to associate our secret credentials with.

The following example shows how to create a keyring name my\_keyring, with credentials to access my\_database as my\_username. We first create a backend\_file type of keyring which will store the encrypted credentials in the user’s home directory ( ~/.config/r-keyring). With keyring\_create, we prompt for the password that will unlock the keyring. Finally, we store a credential in the keyring with set before locking it with keyring\_lock.

library(keyring)

# Set variables to be used in keyring.

kr\_name <- "my\_keyring"

kr\_service <- "my\_database"

kr\_username <- "my\_username"

# Create a keyring and add an entry using the variables above

kb <- keyring::backend\_file$new()

# Prompt for the keyring password, used to unlock keyring

kb$keyring\_create(kr\_name)

# Prompt for the credential to be stored in the keyring

kb$set(kr\_service, username=kr\_username, keyring=kr\_name)

# Lock the keyring

kb$keyring\_lock(kr\_name)

# The encrypted keyring file is now stored at ~/.config/r-keyring/ and can be

# accessed by any R program that provides the keyring password

We can store credentials for multiple usernames per service, and multiple services per keyring. This is ideal in the case of an application that must access a variety of services via access tokens. The encrypted credentials file can either be published with the code, or perhaps for extra security, distributed via a separate channel.

**Retrieving credentials**

To retrieve credentials, set the same three variables and use the keyring get() function, which will prompt us for the keyring password that we set when we called create. A retrieval script might look like this:

library(keyring)

library(DBI)

# Set variables to be used in keyring.

kr\_name <- "my\_keyring"

kr\_service <- "my\_database"

kr\_username <- "my\_username"

# Output the stored password: normally you would not want to do this

keyring::backend\_file$new()$get(service = kr\_service,

user = kr\_username,

keyring = kr\_name)

# Establish connection to Teradata retrieving the password from the keyring.

dbConnect(drv = odbc::odbc(),

dsn = "my\_dsn", # set DSN options in ~/.odbc.ini

pwd = keyring::backend\_file$new()$get(service = kr\_service,

user = kr\_username,

keyring = kr\_name))

With this, we are able to retrieve arbitrary credentials for a particular username and service, allowing us to manage much more complicated sets of credentials with a single password.

So, which is the best way to ensure that plain text credentials are not published with code? If your code relies on a limited number of credentials, an interactive prompt may be the more suitable choice: code users know what their username and password are and can easily enter them interactively.

If the code requires multiple, hard-to-remember, or cumbersome to provide credentials, you might want to consider using keyrings. Users will only need to provide one password, which will unlock the keyring and provide access to all credentials.