Keep Your Secret

This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

# Course "Keep your secret under control - Applied cryptography"

### Promo

Hello and welcome to the course Keep yOUr secrets under control! This course is about applied cryptography in a way that you decide how yuo generate your private key, with which software you do it and more importantly you control entire process yourself!!! At the end of the course you will be able to create your own Private Keys, be capable to encrypt and decrypt your information. At the end of this course you will be able to securely store your passwords, secret thoughts or share credentials over the internet...

You may ask me, why to reinvent the wheel? Encryption software was already invented and can be used even for free!

Indeed I am not aiming to compete with them, rather proposing free open source alternative with R Statistical Software. I trully believe this course will be extremely valuable for several reasons:

The most important advantage is about the possibility to have entire encryption process clear and under control. Creating Private Keys, Encrypting/Decription process will be available without using locked software.

Second advantage is about applicability of the course. You will get several practical ways of using encryption on the day to day basis. You will even create your own private key on the usb key and get simple user interface to encrypt your secrets... What to say I am using this myself!

Of course let me also higlight you that course scope will be limited to encrypting of simple text tables and particular focus will be made on all the risks related to loosing information.

Course will be extremely interactive, easy to follow with a bunch of practical material you will continue use even after the course...

Are you getting lost in managing your passwords? Are you worried to dedicate your secrets to unknown locked software? Join this course to get the solution - and keep your secrets under your control!!!

Two notes about this course: First -> you should be aware that there is a risk to loose your information so you should be very carefull Second -> we will not go to deep in details on how encryption work instead we will focus on how to use it to achieve simple objectives. For example how you can keep you passwords secure wihtout need of using locked software, how you can share your secrets with selected people and so on

This course will be fun and interactive. No complex math but step by step actions to achieve the goal - be able to encrypt / decrypt bunch of passwords for email accounts. You will be able to follow along and learn by doing.

so join this course and keep your secrets under your control

How do you keep your secrets like passwords for email or ebanking? Piece of paper? Perhaps a file in a computer? Perhaps you keep password stored in your browser? May be you have an ecnryption software for that but at the end you have some doubts about whether it's still secure or not... Perhaps it is all correct until you forget your password or loose piece of paper or worse if someone will hack it. Then it's a nightmare... Because I don't want to have such a nightmare myself I decided to study the subject and create an encryption tool that help me to keep my secrets under my control.

Well, I don't want to re-invent the wheel but at least I want to make sure that my secrets are kept secure with least risks possible and I know exactly what its happening when I am using it.

So please check it up and see how we can tackle this problem!

### Section 1 Introduction

#### Lecture 1 Introduction

Hello and thank you for spending your time here. This is an introduction video to the course keep your secrets under control. In the next few minutes my goal is to explain you what will be in this course and how can you benefit from it

what: The main goal of this course is to create a way all of us can use open source encryption algorithm to keep and protect our secrets. Passwords, sensitive records and so on will be protected without usage of paid or free locked software. We will achieve our objective without reinventing the wheel or new encryption algorythm but just using existing open source and free R Statistical Software. To make our life easier we will make a nice user friendly interface in a form of Shiny App. You will learn how to generate and use your own private keys. Even further you will learn how to share your secrets with your colleagues or loved ones... This will be very practical and interactive course where you can practice and follow along. Entire material will be available to you and you can keep experimenting futher to reach desired results

why: now speaking frankly I am not pretend to be a cybersecurity expert or super advanced programmer so I am not going to reveal you anything top secret algorythm nor I can promise you something completely secure. However I strongly believe you will still get a huge advantages of this course.

you will... know exactly all the steps you need to take during encryption/decryption process. you will create your own routine on managing passwords routine you know and finally trust

you will not rely anymore on paid or free locked software that you don't know and have no idea what is happening behind the scenes. you don't risk to download computer virus or trojan when you are getting this software

you will also get multiple practice exercises so you will keep yourself engaged during the course.

finally you will highly reduce the risk of not being hacked because your secrets are hacked or you lost your passwords

at the moment I am creating this course I plan to release it for free so you have nothing to loose but just get benefits. In any case your passwords or secrets are yours and it's really up to you how you would keep them. My goal is just suggest you one method to use.

I am looking forward to see you in the course keep your secrets under control...

#### Lecture 2 How to take full advantage of the course

You will need for the course:

* usb key
* mac or pc windows, probably linux will work as well
* internet connection
* r statistical software
* from 1 to 4 hours of your time to complete the course
* read disclamer, risk of loosing information!!!
* optional fork github repository
* two ways of completing the course: -- start with theory if you prefer -- start with case study lecture if you prefer start exploring possibilities and want to solve this problems quick (get the solution and start using it)

Assess your situation: A. Beginner -> proceed slowly B. Advanced & want to get things done quick -> Disclamer -> Lecture "Summary Encryption Process in 5 minutes" -> Pick up desired solution for your use C. Expert -> All steps from B & adapt and extend desired solution

#### Lecture 3 Disclamer

By joining this course or using material from the course you agree that all materials of the course are solely provided "as is" without any guarantee of functionality. All methods are provided for training purposes. There is a risk that you can loose your information in case you are not properly follow the instructions or by loosing your private key or alike. Make sure you know what you do at any time when using the material of the course. Always test your knowledge on dummy data before applying it to real data...

Further risks and ways to mitigate them:

A. Encrypting information with Public key while not having proper corresponding copy of Private Key. In this case you loose your information if you are deleting the original data after encryption. To minimize this risk: Possibly Only use Private Key all the time; Always delete and renew corresponding public key when exchanging Private Key. Regularly review your Keys and update them all at once.

B. Losing a copy of Private Key. Risk may materialize when you keep your key in Laptop that is stolen or broken. Always keep at least one copy of Private Key in a secure place. For example keep a copy of Private Key on your USB Key. Regularly review your Keys and update them all at once.

C. Loosing a password to your Private Key File. When you write your private key to a file and setting a password you may forget it. To minimize this risk it may be recommended to keep a copy of Private Key either stored without password or simply stored as a file with function **saveRDS()**. Regularly review your Keys and update them all at once.

D. Loosing encrypted information. You may be loosing control over encrypted information. For example, you can simply forget where it is stored and with which key it was encrypted. In order to minimize this risk you should create solid storage system with specific rules and regularly review (use) the data you encrypt. Do not hesitate to keep a copy of encrypted information in the cloud storage such as OneDrive or Google Drive, etc

E. Forgetting the method of Decryption. You may loose your information if you performing wrong operations of Encryption/Decryption. As a rule of thumb you must write the procedure on the paper, set up reminder and decrypt / encrypt this information every 3 month. When you don't use encryption/decryption method for a long period of time always use a dummy data to practice and 'remember' the method!!!

F. Special characters. You may accidentally loose text information during encryption process. Make sure to check on dummy data first

G. Storing your private key public or keepin it together with encrypted information. Make sure you don't 'commit' private key to a public repository when using Version Control

### Section 2 Basics on encryption and tools

#### Lecture 4 basic encryption

Hello Thank you for continuing the course. In this lecture I would give you some intuitive understanding on encryption process but without overcomplication. My goal of the course is not really study how it works but how to use it to our advantage. You may find a link to wiki article if you want to learn more...

Let me just very briefly discuss on this fascinating subject... in easy and digestable way

I think the first time I came across encryption myself was long time ago. I remember as a kids we were playing games with piece of paper with matrices underneeth that it was possible to rotate in order to encrypt and decrypt the message. A kind of very basic example but quite fun to play... for example this one... <https://i.pinimg.com/736x/11/01/21/11012101d70299dd32c7c4cfbb915aa6--detective-theme-detective-crafts.jpg>

It was very fun to create secret messages at these times... just for fun. Then all those stories about WW2 that entire success of army operations was almost relying on encryption methods!!! Today encryption is a very hot topic because potential advantages for economy, industry and even politics! Well, internet communication is today totally relying on this topic!!!

I believe that majoirty of world population does have a minimum idea about the encryption in general. However I believe that very tify fraction of the world population are really capable to use it themselves. Themselves I mean using some sort of software and be capable to encrypt/decrypt their secrets... For example I was talking to my friend today and he told me that all his passwords he just keeping in his head. Just having four passwords that are recycled amond emails accounts, social networks accounts, online banks, and so on

#### Lecture 5 Get R Statistical Software

-- reuse lecture from the course Identify Problems with AI -- option install it on USB key <https://www.youtube.com/watch?v=earYvB_nRi4>

#### Lecture 6 Load openssl library

Hello and thank you for continuing with the course keep your secrets under control. In this lecture we will learn about our computer engine that will encrypt and decrypt our secrets. If you are already fluent in R you are welcome to follow along otherwise I recommend you to follow along by pausing the video and executing commands by yourself

Open your R Studio and type in the console

#install.packages("openssl")

This will download and install open source Toolkit for Encryption, Signatures and Certificates based on OpenSSL. The next step for you is to type in the console

library(openssl)

This will load this library into the environment so we can use it.

The next step I will recommend is to simply type question mark openssl and execute that in the console. This will bring you to the help page to this package. We will not use all of the functionality of this package but rather focus on practical usage of this library. Of course if you are keen you can study further and read documentation from this library...

Before we end this lecture I want to highlight to you why we are using this library. In a nutshell this is a tool that is open source and therefor very transparent. We like this concept because thousands of people can see the source code. because we don't need any other paid software or software that is closed in principle. The risk here is to get a virus like trojan or any other undesired software. Once you are using openssl and you are downloading it from CRAN you are sure to avoid that. You can only get what you get - the package that helps you to encrypt or decrypt your secrets, can help you to create your keys and manage your secrets. In this course you will get all necessary understanding about how to use this package and to greatly reduce all passwords nighmares risks

This is why we will use it in this course.

#### Quiz 1 What is encryption

### Section 3 Encrypt your secrets basics

Hello and welcome to the

Add OVERVIEW table of actions needed to ENCRYPT <--> DECRYPT

#### Lecture 7 generate your private key and store it

Welcome back to the course. In this lecture we will focus on just two things - understanding how to create our private key using **openssl** library and how to store it

Let us define four learning objectives for this lecture:

* understand how to generate private key
* look what is the structure of private key
* write our key to the file
* risks of loosing keys == risk of loosing your data

We will go very slowly with every step so you can follow it in your own pace. For those who already advanced in R and perhaps don't need such detailed explanations feel free to jump to the end of this lecture to review the code chunk you need to use to create the key.

Open your R Studio and create a new script. Start loading a package with a command library(openssl). Inside this library we have function that can create our private key... this function is named rsa underscore keygen. By typing question mark keygen you can read help for this function

# load openssl library  
library(openssl)  
  
# optional read function documentation  
?keygen

## starting httpd help server ...

## done

Now we can simply type private underscore key and generate our key by typing:

# generating private key  
private\_key <- rsa\_keygen()

Let's understand what happened now. First of all Look to the Environment. YOu have just generated an object named 'private\_key'. I want to draw your full attention now: this is a UNIQUE key. If you run command again this will be ANOTHER key. So be careful!!!

Secondly lets understand what kind of object we just generated... If you look to environment again you will find that this is a list of 4. You can easily see what is inside of this list by using str function. Just type str open parentesis private underscore key and close parentesis... this way you will see a summary on what is in the list

# understand what is in the key  
str(private\_key)

## List of 4  
## $ type : chr "rsa"  
## $ size : int 2048  
## $ pubkey:List of 5  
## ..$ type : chr "rsa"  
## ..$ size : int 2048  
## ..$ ssh : chr "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDVe2i1w ..."  
## ..$ fingerprint:Classes 'hash', 'md5' raw [1:16] 8f f4 79 50 ...  
## ..$ data :List of 2  
## .. ..$ e:Class 'bignum' raw [1:3] 01 00 01  
## .. ..$ n:Class 'bignum' raw [1:257] 00 d5 7b 68 ...  
## $ data :List of 8  
## ..$ e :Class 'bignum' raw [1:3] 01 00 01  
## ..$ n :Class 'bignum' raw [1:257] 00 d5 7b 68 ...  
## ..$ p :Class 'bignum' raw [1:129] 00 f3 b8 be ...  
## ..$ q :Class 'bignum' raw [1:129] 00 e0 3c ab ...  
## ..$ d :Class 'bignum' raw [1:257] 00 9e 15 3e ...  
## ..$ dp:Class 'bignum' raw [1:129] 00 98 83 6c ...  
## ..$ dq:Class 'bignum' raw [1:129] 00 9c f3 d9 ...  
## ..$ qi:Class 'bignum' raw [1:128] 54 9e 97 12 ...

you can access each element just by adding dollar sign and the element of the list. For example if I type private underscore key dolalr type i will get a type of the key:

private\_key$type

## [1] "rsa"

notice we have there some important elements. one of them is a public key on this we will talk in the next lecture and data. What we can see as well that data list of the public key is also a part of the data of the private key...

Let us also make an experiment and tryto play with bits parameters. first we will try to specify bits parameter to be 2000. We call it key2000. After generating this key we will look the structure...

key2000 <- rsa\_keygen(bits = 2000)  
str(key2000)

## List of 4  
## $ type : chr "rsa"  
## $ size : int 2000  
## $ pubkey:List of 5  
## ..$ type : chr "rsa"  
## ..$ size : int 2000  
## ..$ ssh : chr "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAA+wCTtl9/C ..."  
## ..$ fingerprint:Classes 'hash', 'md5' raw [1:16] 89 a7 9f 8d ...  
## ..$ data :List of 2  
## .. ..$ e:Class 'bignum' raw [1:3] 01 00 01  
## .. ..$ n:Class 'bignum' raw [1:251] 00 93 b6 5f ...  
## $ data :List of 8  
## ..$ e :Class 'bignum' raw [1:3] 01 00 01  
## ..$ n :Class 'bignum' raw [1:251] 00 93 b6 5f ...  
## ..$ p :Class 'bignum' raw [1:126] 00 c2 de 3e ...  
## ..$ q :Class 'bignum' raw [1:126] 00 c2 0d 13 ...  
## ..$ d :Class 'bignum' raw [1:250] 50 de 7f d9 ...  
## ..$ dp:Class 'bignum' raw [1:125] 64 1e db be ...  
## ..$ dq:Class 'bignum' raw [1:125] 65 cb 4c 6f ...  
## ..$ qi:Class 'bignum' raw [1:125] 62 55 4b 20 ...

now we will do the same for bits 5000 ...

key5000 <- rsa\_keygen(bits = 5000)  
str(key5000)

## List of 4  
## $ type : chr "rsa"  
## $ size : int 5000  
## $ pubkey:List of 5  
## ..$ type : chr "rsa"  
## ..$ size : int 5000  
## ..$ ssh : chr "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACcgDNeIJSl ..."  
## ..$ fingerprint:Classes 'hash', 'md5' raw [1:16] b4 2a 3d 2b ...  
## ..$ data :List of 2  
## .. ..$ e:Class 'bignum' raw [1:3] 01 00 01  
## .. ..$ n:Class 'bignum' raw [1:626] 00 cd 78 82 ...  
## $ data :List of 8  
## ..$ e :Class 'bignum' raw [1:3] 01 00 01  
## ..$ n :Class 'bignum' raw [1:626] 00 cd 78 82 ...  
## ..$ p :Class 'bignum' raw [1:313] 0e fa a9 d9 ...  
## ..$ q :Class 'bignum' raw [1:313] 0d b7 94 aa ...  
## ..$ d :Class 'bignum' raw [1:625] 1e 56 d5 ff ...  
## ..$ dp:Class 'bignum' raw [1:313] 04 3f 42 65 ...  
## ..$ dq:Class 'bignum' raw [1:313] 06 3a 60 29 ...  
## ..$ qi:Class 'bignum' raw [1:313] 00 8b 0f 22 ...

what differences can you see? You see the lenght of the elements is different...

length(key2000$data$n)

## [1] 251

length(key5000$data$n)

## [1] 626

If you want to see what is in this long number you can of course simply print it:

private\_key$data$n

## [b] 26949614215355450686583252100371428636594763851154582434274763294244314553789259373435386386643021318956034787356216238623518623103252877825700664869348972424179659312938039681520088413481112052050229540042392869248206672815883684661772177755639979822352003359488355959735493039875606845304123406050880416339608421256487774406668175262821665774462327919074369114911433332223252887637114006434687643993197927649457159157745596852284417923795338337108076876200542805005507619990163079966287517670653673029443848207769180859980304514318012132114588611392519234347009821753287501214618637208026628967363287389555034363551

You can of course study your private key further but we will now concentrate on the way we can keep and retain our key. In fact we should store this key somehow. In fact this will be very important as we can loose the access to our information if we accidentally delete our private key!!! So to store the key we can use another function named write\_pem to actually write our password to the file.

to use this function we must supply the private key we just generated, the name of the file and the password.

Note that we can provide here not only the file name but also a path where we want to keep our file... It is possible to set up a password as well. This way we will restrict our system to read our key from file... this will obviously further increase security

This way we can simply write our file by executing the function write underscore pem. I will now provide password to be 'udemy'

write\_pem(x = private\_key, path = "private.key", password = "udemy")

After executing this function you should go to the Files tab of the project and you will find your file there. You can do whatever you want with this key but I will make the following. I will save my private key to the usb key. For that I will add a path to the usb key to the function...

#write\_pem(x = private\_key, path = "USB/private.key", password = "udemy")

Once I am TOTALLY sure that my private key is saved to the file I must delete the private key from the environment. To do this we can run function rm and provide names of the elements we want to remove.

# delete the private key  
rm(private\_key)

If you are using R studio you can also delete everything from the environment using a brush icon.

We now have one last and important learning objective: is about loosing our private key... Well, it's the same effect as you simply forget your password and there is no one who can help you to reset it. Risk is to have a data you will never decrypt that is it! Remember this was yourself who generated it so it's up to you how to keep this file in a way you never loose it... For the purpose of the course I will be using a usb stick where I will keep my private key for my private secrets!!!

At this time let's officially conclude this lecture by making a quick summary:

* understand how to generate private key We can generate our UNIQUE private key by using command rsa\_keygen and set number of bits we want for our key
* look what is the structure of private key This way we will create a list with important parameters including part of it will be a public key
* write our key to the file In order to store our key outside of computer we can store it to the file. We can still protect our private key file with a password. Once our private key is in the file we must delete our private key from the r studio environment
* risks of loosing keys = risk of loosing your data BE very CAREFUL to not loosing your key file. It's not a simple file but a UNIQUE file make sure you have this under control!!!

And before we end this lecture I will just summarise the code you need to generate your private key, write it to the file and delete it from R environment

library(tidyverse)

## Loading tidyverse: ggplot2  
## Loading tidyverse: tibble  
## Loading tidyverse: tidyr  
## Loading tidyverse: readr  
## Loading tidyverse: purrr  
## Loading tidyverse: dplyr

## Conflicts with tidy packages ----------------------------------------------

## filter(): dplyr, stats  
## lag(): dplyr, stats

openssl::rsa\_keygen(bits = 2099) %>%   
write\_pem(path = "private.key", password = "udemy")

That is all for this lecture where we discussed about private keys how to create, what they contain and how to manage them... I will be looking forward to welcome you to the next lecture where we will talk about using private key to generate public key

////////////// some more research////////////

* dsa\_keygen
* ec\_keygen

ecKey521 <- ec\_keygen(curve = "P-521")  
ecKey384 <- ec\_keygen(curve = "P-384")  
ecKey256 <- ec\_keygen(curve = "P-256")

structure of these file

str(ecKey256)

## List of 4  
## $ type : chr "ecdsa"  
## $ size : int 256  
## $ pubkey:List of 5  
## ..$ type : chr "ecdsa"  
## ..$ size : int 256  
## ..$ ssh : chr "ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzd ..."  
## ..$ fingerprint:Classes 'hash', 'md5' raw [1:16] db f0 c0 c5 ...  
## ..$ data :List of 3  
## .. ..$ curve: chr "P-256"  
## .. ..$ x :Class 'bignum' raw [1:32] 55 19 64 ab ...  
## .. ..$ y :Class 'bignum' raw [1:32] db 08 7d f5 ...  
## $ data :List of 4  
## ..$ curve : chr "P-256"  
## ..$ x :Class 'bignum' raw [1:32] 55 19 64 ab ...  
## ..$ y :Class 'bignum' raw [1:32] db 08 7d f5 ...  
## ..$ secret:Class 'bignum' raw [1:32] 16 7c 81 b2 ...

str(ecKey384)

## List of 4  
## $ type : chr "ecdsa"  
## $ size : int 384  
## $ pubkey:List of 5  
## ..$ type : chr "ecdsa"  
## ..$ size : int 384  
## ..$ ssh : chr "ecdsa-sha2-nistp384 AAAAE2VjZHNhLXNoYTItbmlzd ..."  
## ..$ fingerprint:Classes 'hash', 'md5' raw [1:16] 25 ee 61 02 ...  
## ..$ data :List of 3  
## .. ..$ curve: chr "P-384"  
## .. ..$ x :Class 'bignum' raw [1:48] 02 d7 18 e1 ...  
## .. ..$ y :Class 'bignum' raw [1:48] f6 01 6c 2d ...  
## $ data :List of 4  
## ..$ curve : chr "P-384"  
## ..$ x :Class 'bignum' raw [1:48] 02 d7 18 e1 ...  
## ..$ y :Class 'bignum' raw [1:48] f6 01 6c 2d ...  
## ..$ secret:Class 'bignum' raw [1:48] 3f c9 3b d5 ...

str(ecKey521)

## List of 4  
## $ type : chr "ecdsa"  
## $ size : int 521  
## $ pubkey:List of 5  
## ..$ type : chr "ecdsa"  
## ..$ size : int 521  
## ..$ ssh : chr "ecdsa-sha2-nistp521 AAAAE2VjZHNhLXNoYTItbmlzd ..."  
## ..$ fingerprint:Classes 'hash', 'md5' raw [1:16] f8 19 b5 b6 ...  
## ..$ data :List of 3  
## .. ..$ curve: chr "P-521"  
## .. ..$ x :Class 'bignum' raw [1:66] 01 8d 28 71 ...  
## .. ..$ y :Class 'bignum' raw [1:66] 00 b0 23 4e ...  
## $ data :List of 4  
## ..$ curve : chr "P-521"  
## ..$ x :Class 'bignum' raw [1:66] 01 8d 28 71 ...  
## ..$ y :Class 'bignum' raw [1:66] 00 b0 23 4e ...  
## ..$ secret:Class 'bignum' raw [1:66] 00 ce d5 06 ...

* dsa\_keygen

dsaKey1024 <- dsa\_keygen(bits = 1024)  
dsaKey2024 <- dsa\_keygen(bits = 2024)

structure of these file

str(dsaKey1024)

## List of 4  
## $ type : chr "dsa"  
## $ size : int 1024  
## $ pubkey:List of 5  
## ..$ type : chr "dsa"  
## ..$ size : int 1024  
## ..$ ssh : chr "ssh-dss AAAAB3NzaC1kc3MAAACBALRKuYIMK0+3vCTmV ..."  
## ..$ fingerprint:Classes 'hash', 'md5' raw [1:16] 46 b8 60 63 ...  
## ..$ data :List of 4  
## .. ..$ p:Class 'bignum' raw [1:129] 00 b4 4a b9 ...  
## .. ..$ q:Class 'bignum' raw [1:21] 00 93 7e 04 ...  
## .. ..$ g:Class 'bignum' raw [1:129] 00 8f 9f 56 ...  
## .. ..$ y:Class 'bignum' raw [1:128] 1e eb d6 f9 ...  
## $ data :List of 5  
## ..$ p:Class 'bignum' raw [1:129] 00 b4 4a b9 ...  
## ..$ q:Class 'bignum' raw [1:21] 00 93 7e 04 ...  
## ..$ g:Class 'bignum' raw [1:129] 00 8f 9f 56 ...  
## ..$ y:Class 'bignum' raw [1:128] 1e eb d6 f9 ...  
## ..$ x:Class 'bignum' raw [1:20] 33 1a ca 50 ...

str(dsaKey2024)

## List of 4  
## $ type : chr "dsa"  
## $ size : int 2048  
## $ pubkey:List of 5  
## ..$ type : chr "dsa"  
## ..$ size : int 2048  
## ..$ ssh : chr "ssh-dss AAAAB3NzaC1kc3MAAAEBAPC+LRazfyBZX9DKG ..."  
## ..$ fingerprint:Classes 'hash', 'md5' raw [1:16] 37 2d 5e bc ...  
## ..$ data :List of 4  
## .. ..$ p:Class 'bignum' raw [1:257] 00 f0 be 2d ...  
## .. ..$ q:Class 'bignum' raw [1:21] 00 9d 4f 9c ...  
## .. ..$ g:Class 'bignum' raw [1:256] 52 da 6b 00 ...  
## .. ..$ y:Class 'bignum' raw [1:257] 00 9f 91 75 ...  
## $ data :List of 5  
## ..$ p:Class 'bignum' raw [1:257] 00 f0 be 2d ...  
## ..$ q:Class 'bignum' raw [1:21] 00 9d 4f 9c ...  
## ..$ g:Class 'bignum' raw [1:256] 52 da 6b 00 ...  
## ..$ y:Class 'bignum' raw [1:257] 00 9f 91 75 ...  
## ..$ x:Class 'bignum' raw [1:21] 00 8f f8 4b ...

##### // Some more research -- > trying to generate exact same private key...

set.seed(1)  
myPrivateKey1 <- rsa\_keygen()  
set.seed(1)  
myPrivateKey2 <- rsa\_keygen()

str(myPrivateKey1)

## List of 4  
## $ type : chr "rsa"  
## $ size : int 2048  
## $ pubkey:List of 5  
## ..$ type : chr "rsa"  
## ..$ size : int 2048  
## ..$ ssh : chr "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDJnb31U ..."  
## ..$ fingerprint:Classes 'hash', 'md5' raw [1:16] 53 c5 be 5a ...  
## ..$ data :List of 2  
## .. ..$ e:Class 'bignum' raw [1:3] 01 00 01  
## .. ..$ n:Class 'bignum' raw [1:257] 00 c9 9d bd ...  
## $ data :List of 8  
## ..$ e :Class 'bignum' raw [1:3] 01 00 01  
## ..$ n :Class 'bignum' raw [1:257] 00 c9 9d bd ...  
## ..$ p :Class 'bignum' raw [1:129] 00 e9 a7 fe ...  
## ..$ q :Class 'bignum' raw [1:129] 00 dc e5 64 ...  
## ..$ d :Class 'bignum' raw [1:256] 04 98 99 f9 ...  
## ..$ dp:Class 'bignum' raw [1:129] 00 c1 2f df ...  
## ..$ dq:Class 'bignum' raw [1:127] 19 05 e3 59 ...  
## ..$ qi:Class 'bignum' raw [1:129] 00 aa 84 3b ...

str(myPrivateKey2)

## List of 4  
## $ type : chr "rsa"  
## $ size : int 2048  
## $ pubkey:List of 5  
## ..$ type : chr "rsa"  
## ..$ size : int 2048  
## ..$ ssh : chr "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCr5h0Lb ..."  
## ..$ fingerprint:Classes 'hash', 'md5' raw [1:16] b3 25 fa 0d ...  
## ..$ data :List of 2  
## .. ..$ e:Class 'bignum' raw [1:3] 01 00 01  
## .. ..$ n:Class 'bignum' raw [1:257] 00 ab e6 1d ...  
## $ data :List of 8  
## ..$ e :Class 'bignum' raw [1:3] 01 00 01  
## ..$ n :Class 'bignum' raw [1:257] 00 ab e6 1d ...  
## ..$ p :Class 'bignum' raw [1:129] 00 da fb 58 ...  
## ..$ q :Class 'bignum' raw [1:129] 00 c8 f5 35 ...  
## ..$ d :Class 'bignum' raw [1:256] 6f 40 63 c5 ...  
## ..$ dp:Class 'bignum' raw [1:128] 32 99 17 06 ...  
## ..$ dq:Class 'bignum' raw [1:128] 22 82 29 11 ...  
## ..$ qi:Class 'bignum' raw [1:129] 00 a3 14 da ...

#### Lecture 8 generate your public key with your private key

Hello and thank you for continuing this course. In the last lecture we have created our private key and store it to the file. I stored it to the usb key. One copy I decided to keep in my wife's clothes drawer (... it's a joke) Let's now move closer to understand how to generate a public key using a private key.

This lecture will have three learning objectives:

* reading our private key
* generating our public key

At this stage we have our private key written in the file. Once it's in the file we can't really use it we must load it to some program. As you remember our program of choice will be our R Statistical SOftware.

Once again we will start R studio and load our library openssl. Now we can start reading the file. Let's doublecheck that our environment is now empty. I have here my usb key which I will connect to computer now. in my script I will write: private\_key, assigment operator my function read underscore key and I will specify the path to the file. If I have specified password I must now provide it here:

private\_key <- read\_key(file = "private.key", password = "udemy")

If I would not provide my password argument R Studio will ask me this password in the interactive mode

#private\_key <- read\_key(file = "private.key")

Now check it out in the Environment our exact same key will be back!!!

str(private\_key)

## List of 4  
## $ type : chr "rsa"  
## $ size : int 2099  
## $ pubkey:List of 5  
## ..$ type : chr "rsa"  
## ..$ size : int 2099  
## ..$ ssh : chr "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABBwaVD4Ozr ..."  
## ..$ fingerprint:Classes 'hash', 'md5' raw [1:16] e8 6e ac 81 ...  
## ..$ data :List of 2  
## .. ..$ e:Class 'bignum' raw [1:3] 01 00 01  
## .. ..$ n:Class 'bignum' raw [1:263] 06 95 0f 83 ...  
## $ data :List of 8  
## ..$ e :Class 'bignum' raw [1:3] 01 00 01  
## ..$ n :Class 'bignum' raw [1:263] 06 95 0f 83 ...  
## ..$ p :Class 'bignum' raw [1:132] 03 a6 8c a7 ...  
## ..$ q :Class 'bignum' raw [1:132] 01 cd 96 2c ...  
## ..$ d :Class 'bignum' raw [1:263] 00 98 64 81 ...  
## ..$ dp:Class 'bignum' raw [1:132] 02 a9 31 b9 ...  
## ..$ dq:Class 'bignum' raw [1:131] 0b cc 19 cb ...  
## ..$ qi:Class 'bignum' raw [1:132] 03 77 26 e5 ...

We are ready to the second objective of our lecture which is to create public key. As a reminder public key is required to encrypt or lock our secret... because public key was part of a private key we can always just extract it from our private key. To do this we need to add dollar pubkey and assign the result to the object public underscore key

# one possibility:  
public\_key <- private\_key$pubkey  
# alternative from documentation  
public\_key2 <- as.list(private\_key)$pubkey  
# result is the same...

let's check out what is in our key:

str(public\_key)

## List of 5  
## $ type : chr "rsa"  
## $ size : int 2099  
## $ ssh : chr "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABBwaVD4Ozr ..."  
## $ fingerprint:Classes 'hash', 'md5' raw [1:16] e8 6e ac 81 ...  
## $ data :List of 2  
## ..$ e:Class 'bignum' raw [1:3] 01 00 01  
## ..$ n:Class 'bignum' raw [1:263] 06 95 0f 83 ...

At this moment let's conclude our lecture with a quick summary:

* reading our private key To read our private key from the file we use function read\_key and we are reading our private key back. We can keep our private key in the file
* generating our public key because public key is a part of the private key we can extract it as a part of the list.

For those more experienced users I will summarise entire executed code in the form of the simple code chunk:

# read key  
public\_key <- read\_key(file = "private.key", password = "udemy") %>%   
 # extract element of the list  
 `[[`("pubkey")

very elegantly isn't it?

And with that we are perfectly ready to start creating secrets... right in the next lecture

#### Lecture 9 encrypt your information using your public key

Hello and thank you for continuing the course. Now we starting to be closer to the most interesting part - creating secrets or encrypting information. Now I have prepared my secret message which I will need to encrypt. For this example let it be a table where one column contains user name and anothe column contains passwords. This file is stored in the file PasswordLIstETraining dot r t f. What we will need to do next is to provide this file to a coding machine which will be our R Statistical Software and functions provided by package openssl. This software will digest this object and encrypt it using a public key. And obviously next we will write encrypted file back to the folder in a form of the file...

For that our learning objectives will be the following:

* reminder on data loss risk
* we will read file that contain secret (for example text file) into the object
* we will serialize and encrypt this object
* we will then write this encrypted object to the file

By the end of this lecture we will reach the state to have our secret encrypted and packed to the file to be stored or shared securely...

I would like to mention for more experienced users to scroll video quicker if you understand our code and for others please pause the video and try to repeat this code by yourself...

We will start with a reminder not to use any business critical or personal information during the course. Remember that this course is just providing you education and you should use dummy data for practice not real one. Otherwise there is a chance that you can render your data unusable and loose it completely. Only start using material from the course once you are totally understand your actions

once we are ready we can start acutally to read our file

# object to encrypt  
library(tidyverse)  
secret\_clearText <- read\_csv2("PasswordList.csv")

## Parsed with column specification:  
## cols(  
## `User ID,Password` = col\_character()  
## )

# view our secret...  
secret\_clearText

## # A tibble: 3 x 1  
## User ID,Password  
## <chr>  
## 1 MyFakeUserID1,pass-word-1234-òàèé  
## 2 MyFakeUserID2,word-pass-<U+0430><U+0431><U+0441><U+0434>-9876  
## 3 MyFakeUserID3,abcd-pass-1234-word

This way we have our data INSIDE R Environment. Check this out... You can click on the name in the Environment to actually visualize the object in a form of the table

# visualise the object in a separate file  
 View(secret\_clearText)

This data now needs to be encrypted. Before we do that however we need to "serialize" this data. This is required to convert our message to "standard" form for the algorithm to encrypt the "message". We will not go too deep into explanations here and just taking this as granted. What we know however is that the result of this operation. It will be a vector of specific length. Elements of the vector will be numbers/characters

# serialize the object  
secret\_serialized <- serialize(secret\_clearText, NULL)  
head(secret\_serialized, 20)

## [1] 58 0a 00 00 00 02 00 03 02 05 00 02 03 00 00 00 03 13 00 00

length(secret\_serialized)

## [1] 630

And now it's time to go ahead and finally ENCRYPT our serialized message. We will use function **encrypt\_envelope()** from **openssl** package. All we need to do is to provide our "message" and our public key!!!

# encrypt the object  
secret\_encrypted <- encrypt\_envelope(secret\_serialized, public\_key)

This way now we have our secret!!! How does it look like?

str(secret\_encrypted)

## List of 3  
## $ iv : raw [1:16] 6b db 51 04 ...  
## $ session: raw [1:263] 02 c5 04 ff ...  
## $ data : raw [1:640] cb 0b 69 2e ...

It is a list containing three elements and it is our message that is encrypted. We can't now read it back without a private key.

# attempting to unserialize will result in error  
#original <- unserialize(secret\_encrypted$data)  
# error...

To recap, this object is still now in the R Environment and it's ENCRYPTED. From this object we can return to read it back with a series of manipulations we will discuss in the next lecture.

At this moment however we will proceed to the next point of our learning objective which is to store this object in a form of a file...

Our objective now is to transfer this object to the file so we can keep this "message" persistently elsewhere. For example in the Computer Hard Disk or in the USB key, send by email and so on. At the end of the day we can be sure that any person with normal computer tools can not read this data without the Private Key.

To achieve this task in R we can use function **write\_rds()** or **saveRDS()**. We will just use the first function:

# write encrypted data to File  
write\_rds(secret\_encrypted, "PasswordList.Encrypted")  
# other options with same result  
#write\_rds(secret\_encrypted, "PasswordListGZCompr.Encrypted",compress = "gz")  
#saveRDS(secret\_encrypted, "PasswordList\_saveRDS.Encrypted")

At this moment we have a file in our project directory named "PasswordList.Encrypted". This file now containing our secret...

Now let's review steps we did by looking to our diagram... //slide// ... in total we just made five steps:

* We had our file with secrets
* We read this file into R software and make it a data frame object
* We have serialized this object
* We have encrypted this serialized object and got a list of raw data vectors
* We wrote the resulted encrypted object into the file we can store or distribute

Obviously consider that we are moving slowly and consistenlty just to have our better understanding on the process. Once we are ready to put this process in operation it will be executed almost instantly...

I will summarise this process in just few R code lines below. **notice that** I am using a "pipe" operator that transfer the result of the function as a first argument to the next function). This way I will not leave any traces of objects that may remain in the R Environment:

# Summary of steps to generate encrypted message and store it to the file  
library(tidyverse)  
read\_csv2("PasswordList.csv") %>%   
 # serialize the object  
 serialize(connection = NULL) %>%   
 # encrypt the object  
 encrypt\_envelope(public\_key) %>%   
 # write encrypted data to File  
 write\_rds("PasswordList.Encrypted")

## Parsed with column specification:  
## cols(  
## `User ID,Password` = col\_character()  
## )

And with that we are perfectly ready to start backward process to actually DECRYPT our file ... right in the next lecture

#### Lecture 10 decrypt your information using your private key

Hello and welcome back to the course Keep Your Secret under Your Control. I hope you are enjoying the process able to follow it and find something new and useful for your day to day use. In this exciting lecture you will be doing the most fascinating part which is to decrypting your secret from the messy encrypted data...

Just to bring you back upspeed I will show back the steps we did so far in the form of the overview //slide//

We have our file with encrypted data that we sill need to decrypt and read back our file

As in the previous lectures we will do step by step code execution to verify what is happening in the decryption process. Also in the end of this lecture we will summarize the executed code to get the overall view of all steps...

The steps we will perform to decrypt our file will be almost exact opposite to the steps we used to ENCRYPT our secret //slide//. We will do the following things:

* Read file with encrypted data to the R Environment
* Read back our private key in to R Environment [optional]
* Decrypt our file
* unserialize our message
* Write file with our secret back to the Computer File System

Let's confront all those arguments...

First thing will be to read our file with Encrypted Information. We can do so using **read\_rds()** function.

# read file with Encrypted Information (from Computer File System to R Environment)  
secret\_encrypted <- read\_rds("PasswordList.Encrypted")

Now you can observe now the object "secret\_encrypted" directly in your R environment. We call this object exactly the same as it was before...

Our private key is not kept in the R environment so we will need to have it to be able to DECRYPT our Secret. We will do that for the demostration purposes:

# read private key. Step is shown only for demostration purposes  
private\_key <- read\_key("private.key", password = "udemy")

Pay attention that this step is really optional because we can provide just a file with Private key to the decrypt function

Now we can decrypt this object using function **decrypt\_envelope()** from the **openssl** package. We must provide five arguments:

* $data - or the element of the list with encrypted raw data vector
* $iv - or the element of the list 16 byte raw vector returned by encrypt\_envelope.
* $session - or the element of the list raw vector with encrypted session key as returned by encrypt\_envelope.
* key which will be our private key or file path. we will use our file named "private.key"
* password string or a function to read protected keys. we will use our password "udemy"

# decrypting the list from R Environment  
secret\_serialized <- decrypt\_envelope(data = secret\_encrypted$data,  
 iv = secret\_encrypted$iv,  
 session = secret\_encrypted$session,  
 key = "private.key",  
 password = "udemy")

Magic, our secret was successfully decripted and we have got back the vector named secret\_serialized

It's time now to unserialize this object using function **unserialize()**

# getting back original object in a form of the data frame  
secret\_clearText <- unserialize(secret\_serialized)

Wonderful, we can have our passwords list back!!! They are in R Environment

# having our secrets back to R Environment  
secret\_clearText

## # A tibble: 3 x 1  
## User ID,Password  
## <chr>  
## 1 MyFakeUserID1,pass-word-1234-òàèé  
## 2 MyFakeUserID2,word-pass-<U+0430><U+0431><U+0441><U+0434>-9876  
## 3 MyFakeUserID3,abcd-pass-1234-word

They are exactly the same as we left them!!!

Final stage will be to write them back as original file named PasswordList dot csv

# write dataframe to the csv file  
write\_csv(secret\_clearText, "PasswordList.csv")

Great! It's time to conclude this lecture with a quick overview of steps we just took in this lecture to complete our decryption process...

Simply put we read our file containing encrypted information into R Environment, after reading we got a list of raw vectors. then we decrypted this list using private key file. Then we unserialized the decrypted vector into the human readable dataframe and finally we wrote this table into comma separated value file

For sake of to get a summary we will make a short and elegant summary of R code that does the same process in one chunk of R code

# read file with Encrypted Information (from Computer File System to R Environment)  
secret\_encrypted <- read\_rds("PasswordList.Encrypted")  
  
# decrypting the list from R Environment  
decrypt\_envelope(data = secret\_encrypted$data,  
 iv = secret\_encrypted$iv,  
 session = secret\_encrypted$session,  
 key = "private.key",  
 password = "udemy") %>%   
 # getting back original object in a form of the data frame  
 unserialize(secret\_serialized) %>%   
 # write dataframe to the csv file  
 write\_csv("PasswordList.csv")  
  
# remove secret\_encrypted object  
rm(secret\_encrypted)

That is all! We were able to complete the process from reading sensitive data, encrypting it, decrypting and reading it again. We did it in the step=by=step manner so you perfectly understood the process. Not to the very deep internals of course but just enough to achieve the purpose to keep our secret secure...

Attached to this lecture I am placing a short summary of code you need to have to encrypt/decrypt comma separated value file

Now it's time for me to thank you for reaching this far in the course. I would love to see you joining in to the further chapter of the course which will be the case studies where we will be discussing various practical situations of using our new CRYPTOGRAPHY expertise!!!

CURIOSITY -- NOT FOR THE COURSE

///================== CURIOUSITY ================================ /// ///// further comments about the time you need to encrypt/decrypt the data...

# measure time required to encrypt the object...  
start.time <- Sys.time()  
  
# Summary of steps to generate encrypted message and store it to the file  
library(tidyverse)  
read\_csv2("PasswordList.csv") %>%   
 # serialize the object  
 serialize(connection = NULL) %>%   
 # encrypt the object  
 encrypt\_envelope(public\_key) %>%   
 # write encrypted data to File  
 write\_rds("PasswordList.Encrypted")

## Parsed with column specification:  
## cols(  
## `User ID,Password` = col\_character()  
## )

end.time <- Sys.time()  
time\_taken\_encrypt <- end.time - start.time  
time\_taken\_encrypt

## Time difference of 0.01559997 secs

# measure time required to dencrypt the object...  
start.time <- Sys.time()  
  
# read file with Encrypted Information (from Computer File System to R Environment)  
secret\_encrypted <- read\_rds("PasswordList.Encrypted")  
  
# decrypting the list from R Environment  
decrypt\_envelope(data = secret\_encrypted$data,  
 iv = secret\_encrypted$iv,  
 session = secret\_encrypted$session,  
 key = "private.key",  
 password = "udemy") %>%   
 # getting back original object in a form of the data frame  
 unserialize(secret\_serialized) %>%   
 # write dataframe to the csv file  
 write\_csv("PasswordList.csv")  
  
# remove secret\_encrypted object  
rm(secret\_encrypted)  
  
end.time <- Sys.time()  
time\_taken\_dencrypt <- end.time - start.time  
time\_taken\_dencrypt

## Time difference of 0 secs

Total Time used for the process...

time\_taken\_encrypt+time\_taken\_dencrypt

## Time difference of 0.01559997 secs

We have been using relatively small password file with secret. It was just 127Byte... Can we understand how this time can be if our file with secret will grow bigger? To estimate we will generate two files with following characteristics:

* file with original size 126Byte
* file with double size 234Byte
* file with 10x size 1100Byte
* file with original size 126Byte

# measure time required to encrypt the object...  
start.time <- Sys.time()  
  
# Summary of steps to generate encrypted message and store it to the file  
library(tidyverse)  
read\_csv2("PasswordList.csv") %>%   
 # serialize the object  
 serialize(connection = NULL) %>%   
 # encrypt the object  
 encrypt\_envelope(public\_key) %>%   
 # write encrypted data to File  
 write\_rds("PasswordList.Encrypted")

## Parsed with column specification:  
## cols(  
## `User ID,Password` = col\_character()  
## )

end.time <- Sys.time()  
time\_taken\_encrypt1x <- end.time - start.time  
time\_taken\_encrypt1x

## Time difference of 0 secs

# measure time required to dencrypt the object...  
start.time <- Sys.time()  
  
# read file with Encrypted Information (from Computer File System to R Environment)  
secret\_encrypted <- read\_rds("PasswordList.Encrypted")  
  
# decrypting the list from R Environment  
decrypt\_envelope(data = secret\_encrypted$data,  
 iv = secret\_encrypted$iv,  
 session = secret\_encrypted$session,  
 key = "private.key",  
 password = "udemy") %>%   
 # getting back original object in a form of the data frame  
 unserialize(secret\_serialized) %>%   
 # write dataframe to the csv file  
 write\_csv("PasswordList.csv")  
  
# remove secret\_encrypted object  
rm(secret\_encrypted)  
  
end.time <- Sys.time()  
time\_taken\_dencrypt1x <- end.time - start.time  
time\_taken\_dencrypt1x

## Time difference of 0.01559997 secs

* file with double size 234Byte
* file with 10x size 1100Byte

Summing up our time

///================== CURIOUSITY ================================ ///

#### QUIZ

THE QUIZ TO make sure student understands the risks of loosing private key

* Question 1: How can I create new Private Key to use for my purposes?

A. Using Paid/Free software B. Using openssl library in R Statistical Software C. Using PuttyGen on Windows/Terminal on Mac D. All the above

* Question 2: How can I create a copy of my Private Key?

A. Copy/Paste file B. Create new Private Key using function set.seed(123) C. No you can not create a copy of Private key

* Question 3: Can I recover information if I loose Private Key?

A. Yes\* B. No

* Question 4: What is the minimum that I need to Encrypt Information?

A. Information, Public Key, Computer wiht R Software B. Information, Private Key, Computer with R Software

* Question 5: What do I need to Decrypt Information?

A. Information, Public Key, Computer with R Software B. Information, Private Key

* Question 6: Do I need to change My Private Key from time to time?

A. No B. Yes\*

* Question 7: Is it necessary to keep copy of my Private Key?

A. Yes\* B. No

* Question 8: Can I Decrypt inforation using Public Key?

A. Yes B. No\*

* Question 9: Can I generate Private Key from Public Key?

A. No\* B. Yes

* Question 10: Can I generate Public Key from Private Key?

A. Yes\* B. No

* Question 11: Can I send my Private Key by Email?

A. Yes B. No\*

* Question 12: What is the sequence of Encryption in R?

A. Read File, Encrypt, Serialize, Write File B. Serialize, Write File, Read Private Key, Encrypt, Write File C. Read File, Serialize, Encrypt using Public Key, Write File D. Read File, Encrypt using Public Key, Serialize, Write File E. Read File, Serialize, Write File, Encrypt

* Question 13: What is the sequence of Decryption in R?

A. Read File, Decrypt, Unserialize, Verify, Write File B. Unserialize, Write File, Read Private Key, Decrypt, Write File C. Read File, Serialize, Decrypt using Public Key, Write File D. Read File, Decrypt using Private Key, unserialize, Write File E. Read File, Serialize, Write File, Decrypt

* Question 14: How to reduce the risk of loosing Private Key?

A. Keep a copy in a folder 'My Documents' B. Print exact numbers of the list and write it on paper C. Save a copy on USB Key and one copy under Computer File System Control D. Send it to yourself by email

* Question 15: How to decrypt my information if I lost my Key?

A. I can generate a new Private Key B. I can not decrypt it anymore C. I can contact 'openssl' package developers or hackers and ask their help

* Question 16: Where can I keep my encrypted information?

A. In a folder 'My Documents' under Computer File System Control B. In the cloud drive or any other independent File System C. Send it to yourself by email D. All the above, important that the encrypted information will not be lost

* Question 17: Where should I normally store my Private Key?

A. In a folder 'My Documents under Computer File System Control B. In the cloud drive or any other independent File System C. Send it to yourself by email D. Google 'Best practices storing private key' E. No exact answer is possible, user must find the balance between Loosing Information vs Loosing Key

* Question 18: What will happen if you forget passphrase to my Private key

A. I will not be able to use my key and decrypt my information\* B. I can use Master password from openssl package developers C. I may try 'brute-force' attack to try different passwords for example using word dictionaries or alike

* Question 19: What should I do to make sure I am not leaking my passphrase?

A. Make sure you close R Session and restart your PC after generating key B. Delete history of your R Session C. Delete R Project after your R Session

* Question 20: What is the benefit of using R Statistical Software for encryption

A. Open Source Algorithm openssl is used B. More understanding of the process C. It is totally free

* Question 21: Can I create exact same Private Key using 'openssl' package?

A. Yes, but only if I use exact same Computer and function 'set.seed()' B. Yes, but only if I do that in the same day C. No, it's not possible

#### Practical Assignment

* ask students to create private key store it to the file delete private key from environment
* provide private key file, provide information that was encrypted, ask students to decrypt
* ask students to generate private, public key, encrypt information, decrypt information
* ask students to delete private key and attempt to read information
* ask students to take big text file and measure the time it would take to encrypt it...

### Section 4 Keep your secrets - Case Studies

#### Lecture 10 Introduction to Case Studies

Hello and welcome to the practical section of the course Keep your secrets under your control. As you have seen in the previous section encryption and decryption process in R can take less than 50 lines of code. Therefore I believe that real value added of the course is in the ready to deploy in real life code. For this I will welcome you into this section where you will find lectures explaining day to day scenarios of using the code provided with this course. We will cover very basic scenarios like generating your private key, encrypting data using R Studio manually or even creating ShinyApp as an interactive user interface tool and other use cases. Check out coming lectures!!! To facilitate our educational process please 'fork' my repository on GitHub or download provided code scripts attached to the lectures

Before I would welcome you to the lectures let me encourage you to go back and watch disclamer lecture. This is just to refresh you on the risks associated to the loosing information. This said I am looking forward to welcome you in the first lecture of the section!

#### Lecture 10 Generate Key, Encrypt/Decrypt in 1 R Script...

Hello and welcome back to the course "Keep your secrets under your control". In this lecture we will cover scenario of simple usage of material of the course to generate your private and public key pairs. The goal of this lecture will be to:

* Creata a private and public key pairs
* encrypting lists of dummy passwords
* and, decrypting our passwords back

Having this script you will be able to start using encryption with basic interface in R-Studio.

To follow this lecture please go to the Lecture 10 branch of the GitHub repository or simply use attached file to this lecture

As a little preparation a quick note on used packages or libraries of R language. We will be using packages: 'openssl' and 'tidyverse'. Make sure you can install those packages to follow this lecture...

# Used Libraries:  
library(openssl)  
library(tidyverse)

finally let's start with the first goal for this lecture or...

##### How do we create a private and public keys...

Actually, Simple two lines of code will make this job. However before you are execuing this code you must be sure that your private key will not be overwritten with your command!!! Just make sure to always, always have a key stored in a separate file, perhaps on USB key. Now we can execute this code. We will see that a new Private key was generated. This private key will be immediately written to the computer file system. In our case - just into the R project folder.

#### KEY MANAGEMENT ####  
# generate your private key (NB: make sure to do back up copy!!!)  
rsa\_keygen(bits = 2099) %>%   
 write\_pem(path = "private.key", password = "udemy")

Optionally we can extract a public key and write it to the file. You may need this in case your friend or colleague will need to encrypt information and send it to you. In this case your colleague or collaborator will be only capable to encrypt but not decrypt this information...

# generate your public key (NB: optional. Use Private Key to encrypt/decrypt)  
read\_key(file = "private.key", password = "udemy") %>%   
 # extract element of the list and write to file  
 `[[`("pubkey") %>% write\_pem("public.key")

After executing this code file named "public dot key" will be written to the R project folder... please feel free to pause the video and execute this code by yourself...

We will now move to the next subject of the agenda which is to know how can we...

##### \* encrypt lists of dummy passwords

I have a passwords saved in csv file which you can see here... this list can be also found in the materials of this lecture. Feel free to use this or create any other siutable file with dummy data that you can use to practice before applying this concept to the real data... The two chunks of code will do the job. First chunk of code is designed to work using a public key while second chunk can be used with a private key...

Note that in both cases we are removing the original file after encryption but only after checking that encrypted file is exists... At this point of time you can save copy of your encrypted information in a desired location and then not to forget to save a private key in the secure place...

Instead we will move to the next chapter which is to Decrypt our passwords and write them back to the file

##### \* and, decrypting our passwords back

The code required to decrypt our secret is available below. We are reading the file into the R environment and then decrypting it with a function decrypt\_envelope. After that we can unserialize it and write to the file that we can read again. Make sure you are pausing the video and fully understand what is going to happen

If you are finding your file back - success...

As a final note I will just recommend you to clear history in the R Environment, eventually delete code from console with CTRL + L, also check .Rhistory file or simply delete it completely.

We are now achieved all the objectives for this lecture. We have a code that we can use to generate our private and public keys. We tried to encrypt and decrypt our dummy data and we cleaned the R Environment simply not to leak passwords.

I think that this method is very fast and simple however I don't really like that we keep our private key directly in the R Project File System. Let's move to the next lecture where we will see how to keep our private key in the USB Key and our Password list as an Excel document...

#### Lecture 11 Keeping your private key on usb key Mac

Hello and welcome back to the course Keep your secrets under your control. By the way thank you for continuing the course. I am sure if you get this far this means that you are most likely find it useful for you. In this lecture we will review how to store your own presonal private key on the usb stick. Once again I would remind you to look on disclamer provided on this course in the lecture 2. Make sure you have another copy of your private key and constantly aware of meaning of performed actions

We will have the same version of this lecture dedicated for MAC users and Windows PC users

Let's start with Mac OS... if you are using PC Windows feel free to jump to the next lecture

Put in any usb key and verify that you have disk displayed in the desktop. If it's not present launch disk utility app. Essentially you can format your disk and erase all data on it... I have formatted my disk as MS-DOS FAT file system and named my drive PDMS-DOS. I will then simply copy my private key there:

I will execute this code in R Studio

You see that we have just read our key from the R Project Directory and wrote it to the usb key. Notice that we also removed private and public keys from R Project directory with file dot remove () function

Let's now simulate the process of encryption usign a public key located in the usb drive...

Our file with passwords is located in the R Project Directory [home work place this file in the desktop...and modify script] I will go to the script core code usb mac os dot r and try to execute the code needed to encrypt this file.

In this case I am using a public key file located in my usb key. Once done I will also execute one more line of code to actually delete the original file with password. Notice that I have constructed a little protection here and I will only remove file in case my encrypted object does exists...

I can then keep my encrypted copy of data named PasswordList.Encrypted in any place I like. But let's review the second part of code that will allow me recover my information or we will perform decryption process... the code is exactly the same as in the previous lecture with the only exception that now we are looking to read private key file from the usb drive.

Once I run this code I have my password list file back to the R Project Files environment...

Before I will end this lecture I would ask you to execute a little homework. Try to modify the code above to read and write information from the computer desktop instead of the R File project environment...

Once again I will remind you to clean the R Studio Environment, Console and .History file... simply to avoid keeping passwords in he R history...

We will instead finish this lecture here and I will be waiting you in the next lecture in which we will be using our usb key with a Private key file on the WIndows PC...

#### Lecture 12 Keeping your private key on usb key Windows PC

Hello and welcome back to the course Keep your secrets under your control. In this lecture we will review how to store your own presonal private key on the usb stick if you are using windows. You may find this lecture useful if you are using both systems at the same time. (like myself:)

Put in any usb key and verify that you have disk displayed in My Computer folder. I would recommend you to format your disk and erase all data on it... I have formatted my disk as MS-DOS FAT file system and named my drive PDMS-DOS. I will then simply copy my private key there:

I will execute this code in R Studio

You see that we have just read our key from the R Project Directory and wrote it to the usb key. Notice that we also removed private and public keys from R Project directory with file dot remove () function

Let's now simulate the process of encryption usign a public key located in the usb drive...

Our file with passwords is located in the R Project Directory [home work place this file in the desktop...and modify script] I will go to the script core code usb mac os dot r and try to execute the code needed to encrypt this file.

In this case I am using a public key file located in my usb key. Once done I will also execute one more line of code to actually delete the original file with password. Notice that I have constructed a little protection here and I will only remove file in case my encrypted object does exists...

I can then keep my encrypted copy of data named PasswordList.Encrypted in any place I like. But let's review the second part of code that will allow me recover my information or we will perform decryption process... the code is exactly the same as in the previous lecture with the only exception that now we are looking to read private key file from the usb drive.

Once I run this code I have my password list file back to the R Project Files environment...

Before I will end this lecture I would ask you to execute a little homework. Try to modify the code above to read and write information from the computer desktop instead of the R File project environment...

Once again I will remind you to clean the R Studio Environment, Console and .History file... simply to avoid keeping passwords in he R history...

We will instead finish this lecture here and I will be waiting you in the next lecture in which we will be using our usb key with a Private key file on the WIndows PC...

Let's define procedure of usage to encrypt data and store encrypted data on the computer memory

// Encrypting document with passwords

To read the passwords back using the private key:

On WINDOWS...

Same procedure would be applicable for Windows the only difference is a path to the USB key...

#### Lecture 13 Using your public/private keys to encrypt/decrypt secrets (\*.bat file)

Hello. The idea of this lecture is to explain potential way of encrypting and de-crypting information using executable files known as '*.bat*' files for windows or, '*.sh*' on Mac.

For example I may create two folders in Computer File System:

1. Encrypted
2. Decrypted

I would place executable file to the folder encrypted named: decrypt.bat and vice versa encrypt.bat into 'Decrypted' folder respectively. I should be able to connect usb key and encrypt / decrypt information just by using those executable files. After clicking those files information should be encrypted or decrypted deleting un-used information...

TDL

#### Lecture 13 Keeping your encrypted secrets on cloud web

#### Lecture 14 Share your secrets securely with collaborators

Explain package 'secret'

#### Lecture 15 User Interface to your secrets

Create shiny app with following characteristics: Main App purpose: - Allow users to encrypt data and store it on the specified location as a file... Users should be able to retrieve this information to the shiny App, load the usb key from the usb file and decrypt the table. User can consult the table, add new rows, passwords, secrets and return to encrypt them. THere should be a possibility to load/download decrypted information to/from app...

* table view passwords is coming from user from file and kept in the table
* user can add 1 row to that table
* user can decrypt and information will appear to the same table...
* when use decrypt information will dissapear from this table...

Other features: - Adding passwords in a structured way - Help text / refresher Publish App on Shinyapps.io web page!!!

##### Test protokol for keep your secret shiny app

*.Open and close App without crashes* .Load Private Key file using a button Load Key \* Load wrong file instead of Private Key, app should give a message, not crash \* Load and decrypt encryted information, information should appear on the table \* Download decrypted information to the file \* Encrypt information from the table \* Download encrypted information to the file *.Generate new private and public key pair and download them to the file*  Import, decrypt information, add new row to the table \* Import, decrypt information, remove one or more rows from the table \* import, decrypt information, save it to the csv file \* import, decrypt check and encrypt information back \* import, decrypt search information, filter and sort data in the table

#### Lecture 16 Encrypting passwords in R-Server scripts

#### Lecture 17 Install R on USB key...

<https://www.youtube.com/watch?v=earYvB_nRi4>

### Section 5 Conclusion

#### Lecture 17 Summary

This time it's great deal!!!