

WORKSHOP

Data in use Protection Compass

Keep the cape in the Cloud and on the Edge

N

Homomorphic Encryption hands-on lab

Hands-on lab step-by-step

Version 1.0 (Alpha) - June 2020

<https://aka.ms/DataInUseProtectionWS>



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.



Attribution 4.0 International (CC BY 4.0)

Microsoft and any contributors grant you a license to this document under the [Creative Commons Attribution 4.0 International Public License](#), see the [LICENSE](#) file, and grant you a license to any code in the repository under the MIT License, see the [LICENSE-CODE](#) file.

Microsoft, Windows, Microsoft Azure and/or other Microsoft products and services referenced in the document may be either trademarks or registered trademarks of Microsoft in the United States and/or other countries. The license for this document does not grant you rights to use any Microsoft names, logos, or trademarks. Microsoft's general trademark guidelines can be found at <http://go.microsoft.com/fwlink/?LinkID=254653>.

Privacy information can be found at <https://privacy.microsoft.com/en-us/>

Microsoft and any contributors reserve all other rights, whether under their respective copyrights, patents, or trademarks, whether by implication, estoppel or otherwise.

Table of contents

ABSTRACT AND LEARNING OBJECTIVES.....	1
Overview.....	1
Hands-on lab requirements.....	1
BEFORE THE HANDS-ON LAB.....	3
Task 1: Access your lab environment	3
Task 2: Start a VM in the lab	3
Task 3: Connect to a Windows VM in the lab.....	3
Task 3bis: Connect to a Linux VM in the lab.....	4
TRACK FOR C/C++ DEVELOPERS.....	6
Exercise 0: Set up your development environment	6
Windows users (using Visual Studio).....	6
Ubuntu Users (using a terminal).....	8
Exercise 1: Basics.....	9
Task 0: finalize the setup (Windows users)	9
Task 0bis: finalize the setup (Linux users)	10
Task 1: 1_bfv_basics.cpp.....	11
Task 2: 2_encoders.cpp.....	12
Task 3: 5_rotation.cpp (optional)	12
Task 4: 6_serialization.cpp (optional)	12
Task5: 7_performance.cpp (optional).....	13
Exercise 2: Level & Depth.....	13
Task 1: 3_levels.cpp.....	13
Task 2: 4_ckks_basics.cpp.....	14
Exercise 3: Business case	14
Context setting	14
Computation.....	15
Task.....	15
TRACK FOR DATA SCIENTISTS WITH PYTHON.....	17
Exercise 0: Set up your Jupyter notebook environment.....	17
Connect to the VM with port forwarding.....	17
Use the Docker image.....	17
Manual setup.....	18

Exercise 1: Basics of CKKS 20

COMMON FINAL EXERCISE FOR BOTH TRACKS 22

 Use the Docker image..... 22

 Manual setup..... 23

AFTER THE HANDS-ON LAB 25

Task 1: stop a virtual machine 25

APPENDIX 26

Linux VMs private key..... 26

Abstract and learning objectives

Homomorphic Encryption refers to a [new type of encryption technology](#) that allows computation to be directly on encrypted data, without requiring any decryption in the process.

In this hands-on lab, you will better understand how to:

- Use Homomorphic Encryption to secure your data in public cloud services while being able to leverage them appropriately to sustain the intended treatments and processing.
- Leverage the [Microsoft SEAL open source library](#) for your projects.
- And more importantly use this API correctly and securely, with all the relevant explanation and any necessary background material.

Overview

The **Homomorphic Encryption hands-on lab** is a series of exercises that will walkthrough you to the main considerations, the encryption's schemes, etc. that pertain to Homomorphic Encryption, at least through the lens of the Microsoft SEAL library and other open source projects that lie on top of it. A such, and as you will, Microsoft SEAL provides a simple and convenient API with state-of-the-art performance.

To accommodate different backgrounds, in terms of development languages (namely C/C++ vs. Python) and roles (developers/architects vs. data scientist), this hands-on lab is mainly split into two tracks:

- A track devoted to C/C++ developers,
- And another one that more specifically "targets" data scientists.

Please note that, besides common grounded objectives, these tracks are not necessarily meant to be equivalent since they are intended to be adapted for their respective audience.

Regardless of the chosen track, the related hands-on lab's exercises can be implemented on your own. Please note that they do not contain for a vast majority of them a complete set of step-by-step instructions, but rather, to achieve this, refer to:

- A series of pre-baked Jupyter notebooks.
- Specific commented code samples.
- Specific GitHub repos and theirs (Mark Down) readme and quick starts.
- Specific Web pages.

Hands-on lab requirements

- A [Microsoft account](#).
- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- A DevTest lab environment provisioned with virtual machines (VMs) (**completed the day before the lab!**). These WMs are a Windows-based or a Linux-based development environment configured with:

- A code editor, such as [Visual Studio Community 2019](#) or [Visual Studio Code](#) installed.
- A terminal console which allows you to both:
 - Execute Git commands, such as [Git for Windows](#) (2.10 or later).
 - Remotely connect to a VM in SSH, such as [OpenSSH](#), [PuTTY](#).

Before the hands-on lab

Duration: 5-10 minutes

In this pre-exercise, you will set up your environment, i.e. reclaim the virtual machines (VM) for use in the rest of the hands-on lab. You should follow all steps provided *before* attending the hands-on lab.

This pre-exercise applies to both tracks.

IMPORTANT: Most Azure resources require unique names. Throughout this lab you will see the word "SUFFIX" as part of resource names. You should replace this with the value defined as part of the provisioning of your DevTest Lab environment to ensure the resource is uniquely named.

Task 1: Access your lab environment

1. Launch a browser and navigate to <https://portal.azure.com>. Once prompted, login with your Microsoft Azure credentials. If prompted, choose whether your account is an organizational account or just a Microsoft Account. Enter the related credentials to sign in.
2. In the [Azure Portal](#), open the resource group named **HE-HOL-RG-SUFFIX**.
3. Click **DevTest Lab** to access the hands-on lab environment that has been prior provisioned for you.

Task 2: Start a VM in the lab

1. To start a listed Windows virtual machine (VM), right-click the intended VM row under **Claimable virtual machines** and select **Claim machine**. This will start the VM.
2. When the VM is started, it will be displayed in the **My Virtual Machines** pane. After one minute, the status will be **Running**.

You can now connect to the VM. To do so:

- With the **HE-HOL-WINDEV-VM01** Windows lab VM, follow the instructions as per **Task 3: Connect to a Windows VM in the lab**.
- With the **HE-HOL-LINDEV-VM01** Linux lab VM, follow instead the instructions as per **Task 3bis: Connect to a Linux VM in the lab**.

Task 3: Connect to a Windows VM in the lab

1. Under **My Virtual Machines**, select the **HE-HOL-WINDEV-VM01** lab VM row.
2. At the end of line click ..., and then select **Connect**.

RDP SSH BASTION

Connect with RDP

To connect to your virtual machine via RDP, select an IP address, optionally change the port number, and download the RDP file.

IP address *

Public IP address (52.166.88.167) ▼

Port number *

3389

Download RDP File

Can't connect?

[Test your connection](#)

[Troubleshoot RDP connectivity issues](#)

3. Select **Download RDP file**, then open the downloaded RDP file.
4. Click **Connect** on the Remote Desktop Connection dialog.
5. When invited, specify the username and password as detailed in each exercise in the rest of this document. (Do not use your organizational account or your Microsoft Account one.)
6. Click **Yes** to connect, if prompted that the identity of the remote computer cannot be verified.

At the stage, you should be connected on the Windows desktop of the **HE-HOL-WINDEV-VM01** lab VM.

Task 3bis: Connect to a Linux VM in the lab

1. First, save the provided private key to a new text file, for example `keyfile`.

Note Typical SSH private key files don't have any file extension. This however doesn't really matter and if you can't save the file without an extension, any non-binary extension will do. A widely used convention on Windows-based systems consist of using **.pk** for private key and **.pubk** for public keys.

2. Under **My Virtual Machines**, select the intended VM row, i.e. the one for the **CC-HOL-LTEST-01** lab VM.
3. At the end of line click **...**, and then select **Connect**. A SSH connection string will be displayed.

RDP **SSH** BASTION

Connect via SSH with client

1. Open the client of your choice, e.g. [PuTTY](#) or [other clients](#).

2. Ensure you have read-only access to the private key.

```
chmod 400 azureadmin.pem
```

3. Provide a path to your SSH private key file. ⓘ

Private key path

~/ssh/azureadmin

4. Run the example command below to connect to your VM.

```
ssh -i <private key path> azureadmin@13.95.140.218
```

Can't connect?

[Test your connection](#)

[Troubleshoot SSH connectivity issues](#)

4. Now open on your local machine, a prompt command line and enter the provided SSH connection string, using the path to `keyfile` as the <private key path> argument. For example:

```
C:\> ssh -i ~/keys/keyfile azureadmin@13.95.140.218
```

5. When prompted for the authenticity check, ensure the fingerprint is either
SHA256: yW4AyDfXb66iVz64rhGU/KyUMQFPatKsHmJ2m2Hby1U or
MD5: 71:88:bb:ed:8a:96:2c:f4:64:21:8b:ec:05:7a:b7:61, and then type "yes".

(REPLACE these values with your own ones.)

At the stage, you should be connected on the Linux lab VM with a Bash shell.

You should follow all steps provided *before* attending the Hands-on lab.

You are now ready to start the chosen Track!

Track for C/C++ developers

Exercise 0: Set up your development environment

Duration: 5-10 minutes

Objectives:

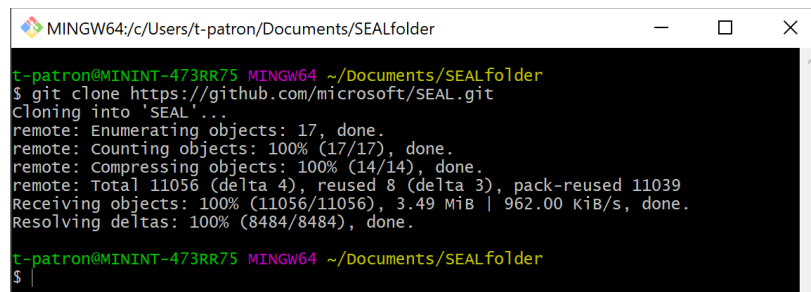
- Set up your development environment depending on the OS you're working on
- Being able to reproduce these steps after this workshop to develop your own programs leveraging SEAL

Windows users (using Visual Studio)

Compile the library and the examples

In this task, you will learn how to compile SEAL and how to use the library with C++ in your own Visual Studio project. To do so, please follow these instructions:

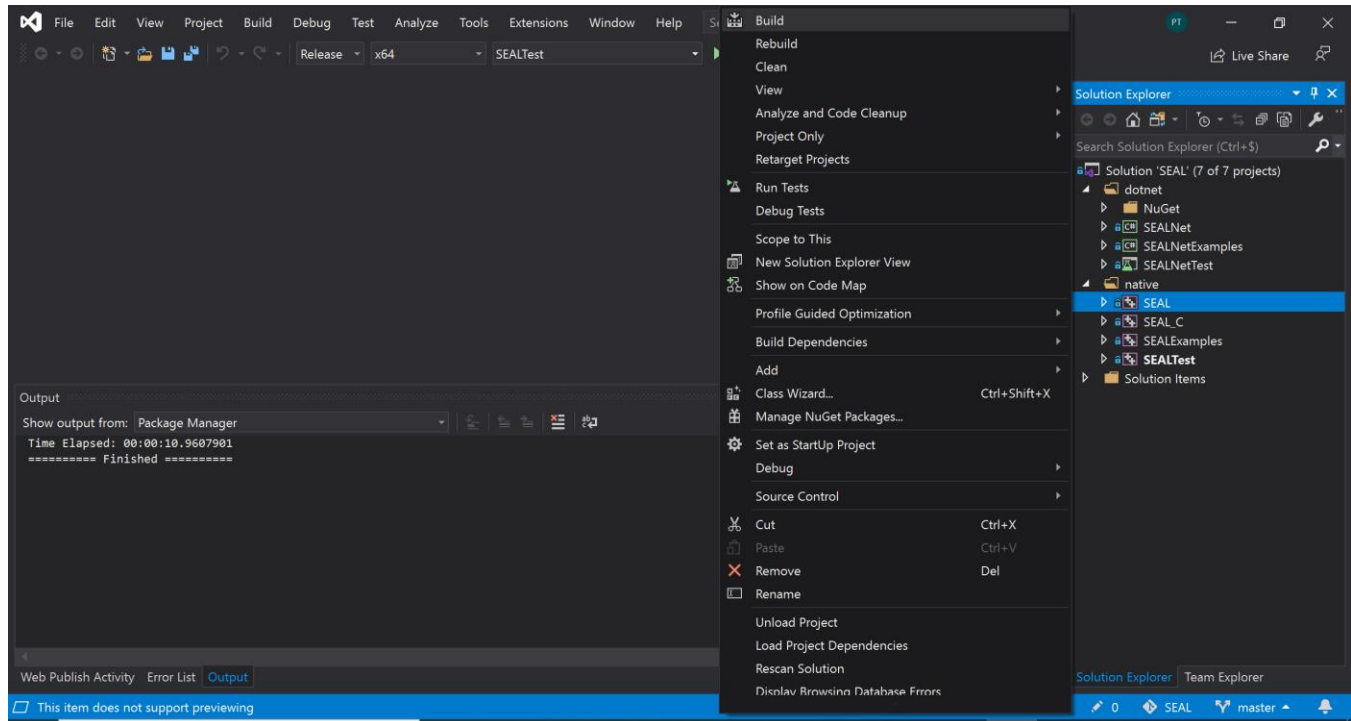
1. Clone the repository in the folder of your choice: open the folder of your choice, write-click in it, select "Git Bash Here" and clone the repository:



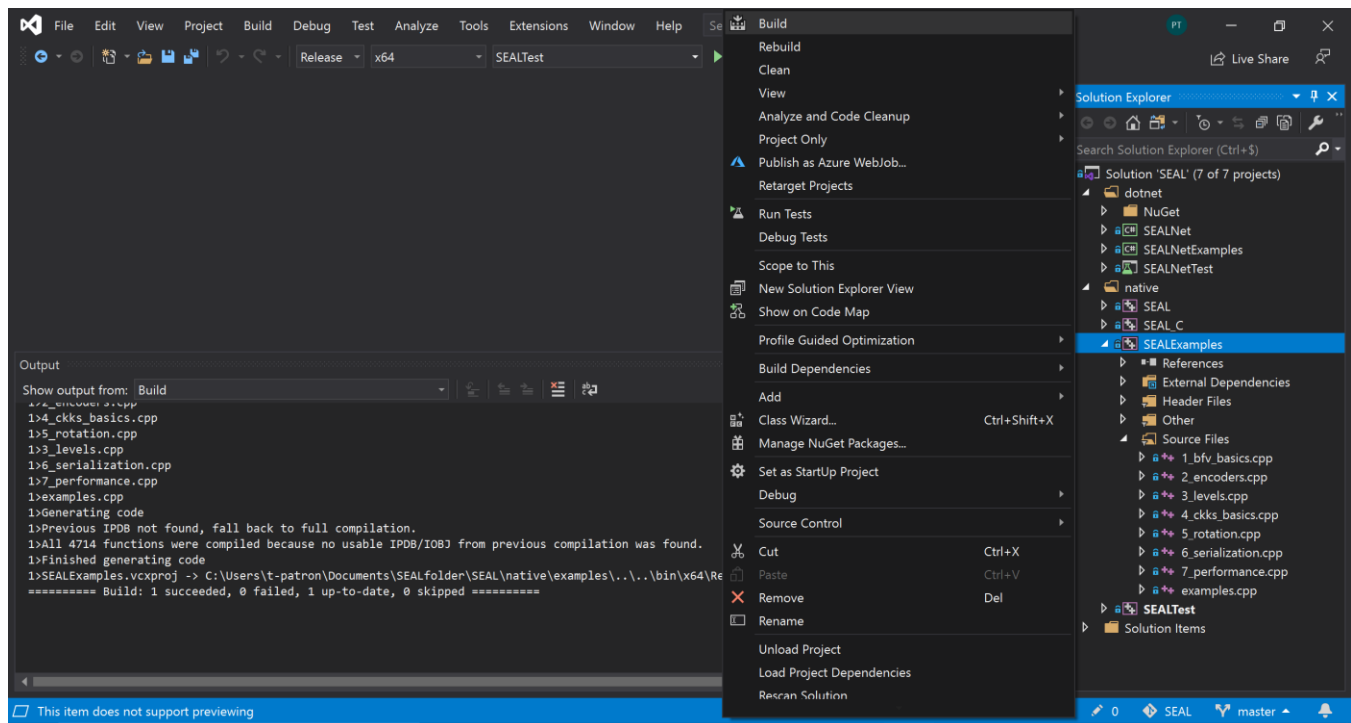
```
MINGW64:/c:/Users/t-patron/Documents/SEALfolder
t-patron@MININT-473RR75 MINGW64 ~/Documents/SEALfolder
$ git clone https://github.com/microsoft/SEAL.git
Cloning into 'SEAL'...
remote: Enumerating objects: 17, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 11056 (delta 4), reused 8 (delta 3), pack-reused 11039
Receiving objects: 100% (11056/11056), 3.49 MiB | 962.00 KiB/s, done.
Resolving deltas: 100% (8484/8484), done.
t-patron@MININT-473RR75 MINGW64 ~/Documents/SEALfolder
$
```

Note When cloning the repo, remember that the "Ctrl + C", "Ctrl + V" do not work with Git for Windows, and add hidden characters. Do not use them when following the instructions, or you might encounter issues when cloning the repo.

2. Build the library (right-click on the `native>SEAL` repo and click **Build**):



3. Build the examples following these instructions:



Note Building the library can take a few minutes, it is normal. If you prefer using C# instead of C++, you'll find the instructions [here](#).

Use the library in your own project

If you want to use the library in your own Visual Studio project after this workshop, you can follow the instructions in the video whose link is given below.

Note Here is a video resuming this task (without the examples part). It is old dated so it uses an early version of SEAL and some commands might differ a little, but it gives a good vision of what you should do, and what things should look like.

Ubuntu Users (using a terminal)

Important note These instructions also apply for Linux (and MacOS) users.

Compile the library and build the examples

In this task, you will learn how to compile SEAL and how to use the library in your own project using a bash terminal. To do so, you can follow the [instructions](#) available from the official Git repository of the library. For this lab session, you just need to build the library ('[Building Microsoft SEAL](#)' under sections **Building Microsoft SEAL / Linux**) and the examples ('[Building examples](#)' under **Building Microsoft SEAL / Linux**). You do not need to install the optional dependencies for this workshop.

Note If you prefer using C# instead of C++, please follow [these instructions](#) instead.

Using the library in your own project

If you want to use the library in your own project after this workshop, you can follow the following [instructions to install SEAL](#).

Note Here is a [video](#) resuming this task (without the examples part). It is old dated so it uses an early version of SEAL and some commands might differ a little, but it gives a good vision of what you should do, and what things should look like.

You should now move to the next exercise.

Exercise 1: Basics

Duration: 15-20 minutes

This exercise, along with the following one, focuses on the examples provided by the GitHub repository. Both exercises consist in running a given example and reading the comments of its code in parallel.

The tasks will indicate the important points to understand, but the real value and considerations remain in the comments of the code. Thus, you can follow exercises 1 and 2 at an appropriate pace for you : please take advantage of this time to understand how BFV and CKKS schemes work before starting exercise 3.

Objectives:

- Understand the structure of a program using SEAL
- Get to know the parameters and which role they have
- Get to know the different types of keys
- Understand the purpose of circuit optimization
- Discover the main operations in HE and their purpose: addition, multiplication, relinearization, rotation

Key concepts: parameters, encoders, circuit optimization, noise budget

Difficulty: 1/5

Task 0: finalize the setup (Windows users)

As explained before, this exercises mostly consist in reading an example's code comments and run it in parallel : for a better understanding, it is highly recommended to display both the terminal and the code on the same screen:

If you completed exercise 0 and everything worked well, you should be able to run the `sealexamples.exe` file (path: `SEAL\bin\x64\Release\sealexamples.exe`) by double-clicking on it. Put the window on the right of your screen.

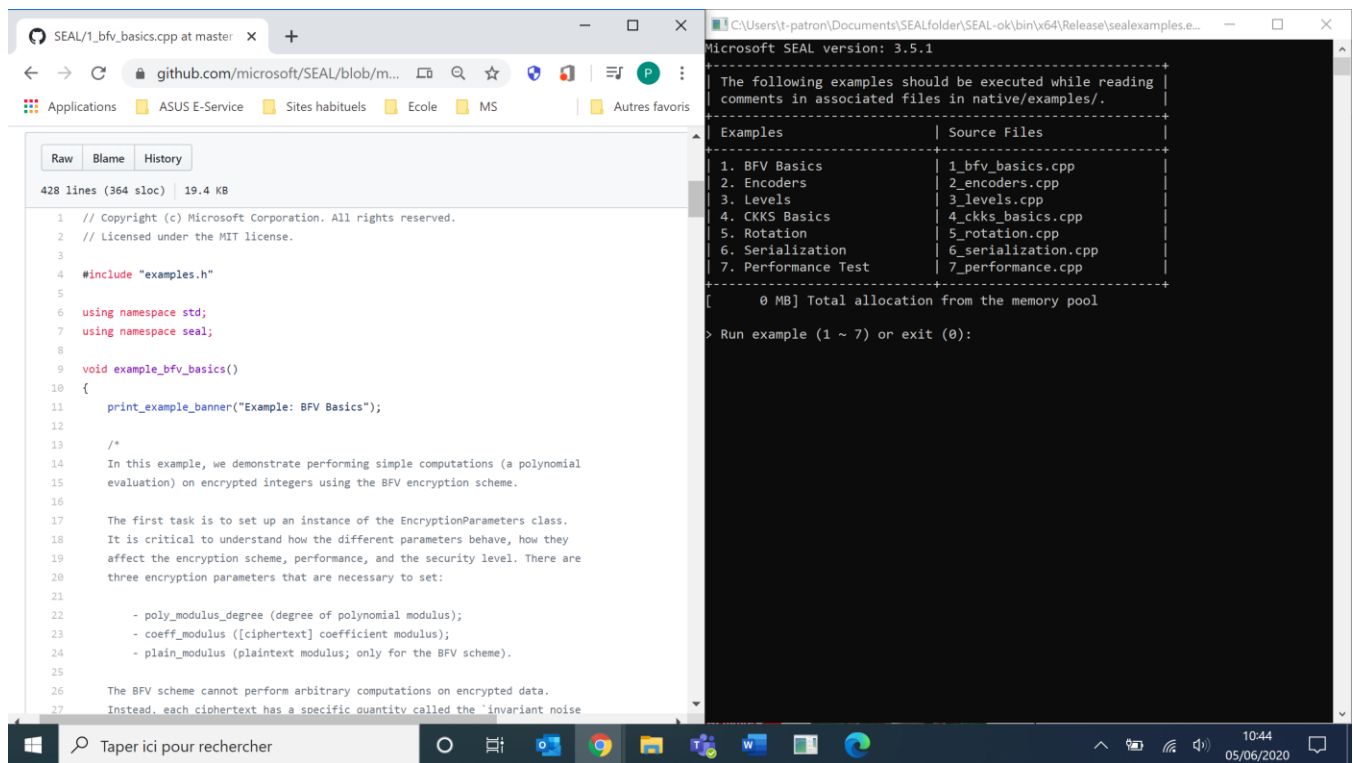
In this exercise and the following, and each example will point to some examples you'll have to run and understand (named in the title of each task). To display the code of the pointed example, you can:

- Open it with Visual Studio in the language of your choice (using the Solution Explorer). Close the Solution Explorer and put the window on the left of your screen. Adjust the zoom in the bottom-left corner of the window.

Note You can open the Solution Explorer by clicking on View -> Solution Explorer.

- Open it in a browser by following this link (replace `$(example.language)` accordingly):
[https://github.com/microsoft/SEAL/blob/master/native/examples/\\$\(example.cpp\)](https://github.com/microsoft/SEAL/blob/master/native/examples/$(example.cpp)) for C++ examples
[https://github.com/microsoft/SEAL/tree/master/dotnet/examples/\\$\(example.cs\)](https://github.com/microsoft/SEAL/tree/master/dotnet/examples/$(example.cs)) for C# examples

Put the code on the left of your screen. It should now look like this:



Task 0bis: finalize the setup (Linux users)

As explained before, this exercises mostly consist in reading an example's code comments and run it in parallel : for a better understanding, it is highly recommended to display both the terminal and the code on the same screen:

Option 1: without Visual Studio Code

If you completed exercise 0 and everything worked well, you should be able to run the `sealexamples.exe` file (path: `SEAL\bin\sealexamples.exe`) by double-clicking on it. Put the terminal on the right of your screen.

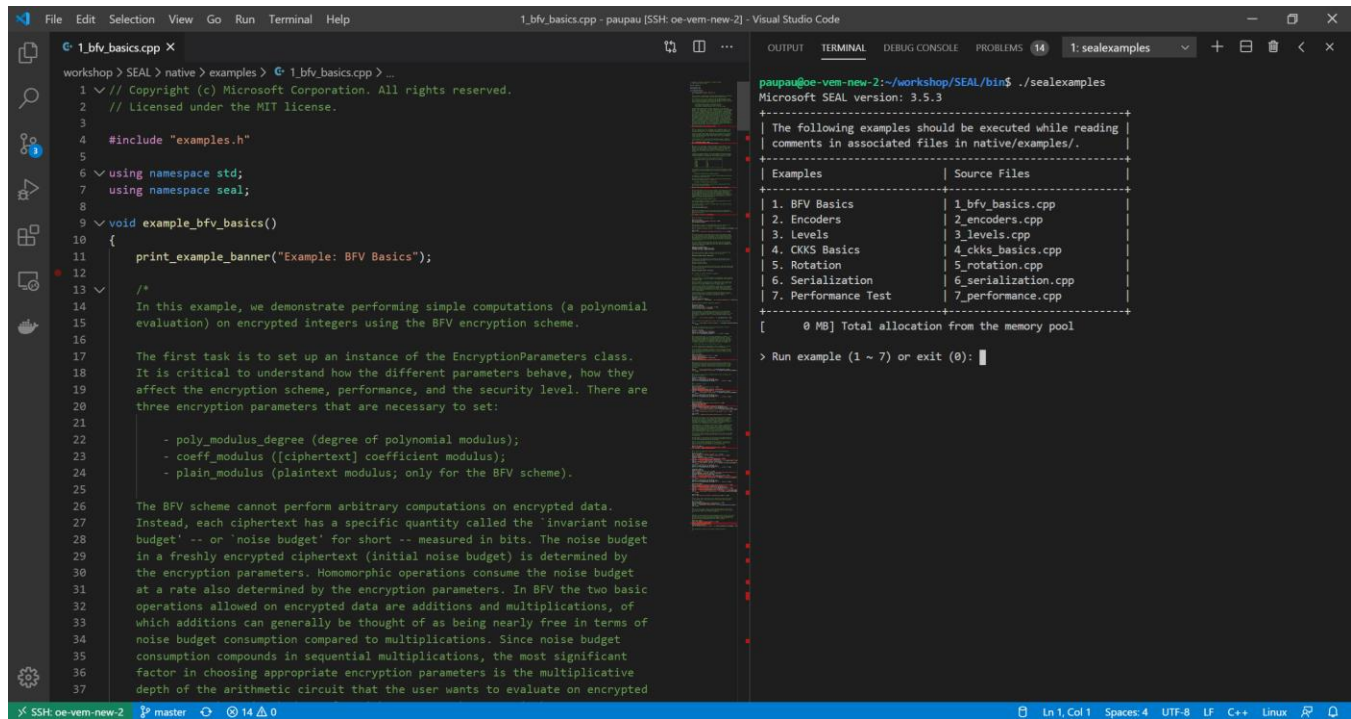
In this exercise and the following, and each example will point to some examples you'll have to run and understand (named in the title of each task). To display the code of the pointed example, open it in a browser by following this link (replace `$(example.language)` accordingly):

- For C++ examples: [https://github.com/microsoft/SEAL/blob/master/native/examples/\\$\(example.cpp\)](https://github.com/microsoft/SEAL/blob/master/native/examples/$(example.cpp))
- For C# examples: [https://github.com/microsoft/SEAL/tree/master/dotnet/examples/\\$\(example.cs\)](https://github.com/microsoft/SEAL/tree/master/dotnet/examples/$(example.cs))

Put the browser window on the left of your screen. It should now approximatively look like the previous screenshot.

Option 2: using Visual Studio Code

If you connected to your machine using SSH, you can also use Visual Studio Code to display the example files and the terminal on the same window, as showed in the following screen shot:



Task 1: 1_bfv_basics.cpp

In this task, you will:

- Discover the structure of a program leveraging the SEAL library (using the BFV scheme)
- Understand how parameters should be fixed in BFV
- Discover the notion of invariant noise budget
- Learn how to check if your parameters are valid with SEAL

After reading and understanding this example, you should be able to answer to the following questions:

- What is the invariant noise budget? What is it derived from? How is it measured and how is it consumed?
- In which order should the parameters be chosen? With which requirements? What is their influence? What is the relationship between the coefficient modulus and the polymodulus degree?
- What is the relationship between invariant noise budget, polymodulus degree and plaintext modulus?
- What is the size of a ciphertext? How is it influenced by native operations?
- Why is factorization advantageous in homomorphic encryption?
- What are the inputs and outputs of the encoder, encryptor, evaluator and decryptor classes?

Task 2: 2_encoders.cpp

In this task, you will:

- Discover the 3 types of encoders and their purpose
- Learn how to smartly encode messages into plaintext elements
- Fully utilize the plaintext polynomial

After reading and understanding this example, you should be able to answer to the following questions:

- *What are the input(s) and output(s) of each encoder? How many message slots are available for each encoder? What happens if the user does not use all of the message slots?*
- *How does each encoder work? Define the overflow issue, which encoder does it concern?*
- *How do the coefficients of a polynomial behave with computation? What is the relationship between plaintext modulus and encoded message?*
- *What is batching? What constraint does it make of plaintext modulus in BFV?*
- *What effect does multiplication have on the scale of a ciphertext in CKKS?*

From here, all the remaining tasks of the exercise are optional. You can skip them and come back later as they are not crucial for this lab. However, they also bring relevant information.

Task 3: 5_rotation.cpp (optional)

In this task, you will:

- Understand what rotation is and why it is used
- Learn the requirement for a noise-free rotation and noise-free relinearization

After reading and understanding this example, you should be able to answer to the following questions:

- *How are rotations performed in SEAL? Is it possible to rotate both rows and columns?*
- *What kind of keys are required to perform a rotation?*
- *Do rotations have an influence on noise budget? Which condition must be respected for a noise-free rotation?*

Task 4: 6_serialization.cpp (optional)

Note This example requires ZLIB to be enabled. ZLIB is a widely used compression library that implements the DEFLATE compression algorithm. It is downloaded as part of SEAL and enabled by default. More information [here](#).

In this task, you will:

- See an example of a client/server model leveraging SEAL
- Understand the purpose of serialization
- Learn how to use symmetric-key encryption with SEAL

After reading and understanding this example, you should be able to answer to the following questions:

- *What is the goal of serialization? What effect does it have on keys and ciphertexts?*
- *Under what condition can serialization be efficient in SEAL?*

- Which objects in SEAL are meant to be serialized and never used locally?
- Why is it sometimes useful to use symmetric-key encryption in homomorphic encryption? What considerations does it involve concerning the trust model of the implementation?

Task5: 7_performance.cpp (optional)

In this task, you will get a bit more details about performance metrics in SEAL for both BFV and CKKS.

You should now move to the next exercise.

Exercise 2: Level & Depth

Duration: 20-25 minutes

This exercise, along with the previous one, focuses on the examples provided by the GitHub repository. Both exercises consist in running a given example and reading the comments of its code in parallel. The tasks will indicate the important points to understand, but the real value and considerations remain in the comments of the code. Thus, you can follow exercises 1 and 2 at an appropriate pace for you : please take advantage of this time to understand how BFV and CKKS schemes work before starting exercise 3.

Objectives:

- Discover the notion of "levels" in homomorphic encryption
- Understand how the depth of a computation influences the choice of the parameters
- Link level and computational depth for both schemes
- Discover the rest of the operations: modulus switching and rescaling

Key concepts: level, depth, scale

Task 1: 3_levels.cpp

In this task, you will:

- Use the BFV scheme
- Discover modulus switching
- Understand the notion of levels and link it to the computational depth

After reading and understanding this example, you should be able to answer to the following questions:

- What is a modulus switching chain? What is the difference between the different levels of the chain?
- What does "going up" the chain mean?
- What is the special prime? To which level of the modulus switching chain does it correspond? What constraints does it have in relation to the other primes in the coefficient modulus?
- What are the data levels?
- Can you describe the modulus switching operation? What effect does it have on the level of the ciphertext? Is it possible to reverse the operation and switch up a ciphertext?
- Is modulus switching necessary?

- What is the benefit of switching modulus? When is it more advantageous? Can you understand the trade-off between noise budget loss and computational performance increasement?

Note In BFV, a multiplicative depth n usually requires $\sim \log(n)$ levels of computation

Task 2: 4_ckks_basics.cpp

In this task, you will:

- Use the CKKS scheme
- Discover the rescaling operation
- Understand that depth n requires n levels of computation

After reading and understanding this example, you should be able to answer to the following questions:

- *What does rescaling mean in CKKS? How is it different from modulus switching?*
- *How does rescaling change the scale (relationship involving a part of the coefficient modulus)?*
- *What is the relationship between multiplicative depth and number of rescaling operations? What is the relationship between multiplicative depth and number of primes in the coefficient modulus?*
- *What is a good strategy when choosing the scale and the coefficient modulus?*
- *How can rescaling have a bad impact on precision if the primes of the coefficient modulus are not wisely chosen?*
- *In which situation can modulus switching be useful in CKKS?*

Note In CKKS, a multiplicative depth n requires n levels of computation, which means at least $n+2$ primes in the coefficient modulus.

You should now move to the next section.

Exercise 3: Business case

Duration: ~30 minutes

Objectives:

- Choose between the BFV and CKKS scheme in a real-life example
- Write your own first program using HE / Choose the parameters on your own
- Deal with the scaling functions

Context setting

The goal of this exercise is to consider homomorphic encryption in a real-life situation. Indeed, even if the technology is still evolving, it is already possible to use it in business (if wisely thought, of course!). With this in mind, let's start with a useful but easy-to-implement computation: linear regression.

Linear regression is a commonly used type of predictive analysis: it attempts to model the relationship between one or several features and an outcome by fitting a linear equation to observed data. In this exercise, the goal is to make a prediction of a house's price based on its characteristics. A lot of estate agencies provide this kind of service on

their website for people who consider selling their house. We can imagine a client/server (estate agency) model, in which the client sends his house's characteristics to a server so that it predicts the house's price.

Note This exercise was based on a Kaggle dataset.

Computation

Our example assumes that the model of linear regression has already been trained. Thus, the relevant features x_i have already been selected for you, and the weights w_i have already been computed using a training and a testing dataset. The computation that the server must evaluate is the following:

$$y = \sum_{i=1}^n w_i x_i + b \quad \text{with } x = [x_1, \dots, x_n] \text{ a house with } n \text{ selected features and } b \text{ the bias}$$

Both weights and features have been normalized so that you will only manipulate floating numbers between 0 and 1. As the price of a house can be really high, and as we do not want to manipulate numbers of too many different orders of magnitude, we set:

$$y = \log(p) \text{ with } p \text{ the price of the house } x$$

Task

Your task is to implement this prediction from the server point of view: given an encrypted input vector x , you have to compute y and send it back to a client that will decrypt it and compute its exponential to obtain the input house's price p . To this end, you can:

- Start your own Visual Studio Project with SEAL as a dependency (Windows)
- Modify the `SEAL\native\examples\examples.cpp` file by:
 1. Replacing `while(true)` by `while(false)` (line 12 in the version 3.5.3)
 2. Copying the `linear_regression()` function from one of our templates before the `main()` function (line 8 in the version 3.5.3)
 3. Adding `#include <math.h>` in the include list (line 5 in the version 3.5.3)
 4. Adding the line `linear_regression();` before the while loop in the main function (line 11 in the version 3.5.3)

If you are using version 3.5.3, you will find an example of the modified version of `examples.cpp` at [this url](#). You can use find the templates at the same location. This folder also contains one [solution](#) for this exercise. If you do not feel comfortable with it, you can directly read it.

Otherwise, the templates will guide you. They already involve an example of input vector with the expected result you should compute. If you chose the template [linear regression template 1](#), you only have to choose the parameters. The other template ([linear regression template 2](#)) requires more autonomy. Here is a set of questions to guide your thinking:

- Which scheme should you choose?
- Which encoding should you use?
- How are you going to set the parameters? How many primes should define the coefficient modulus?

- *What operations do you need to evaluate? In which order?*
- *How many significant numbers should you keep?*
- *How can you optimize your parameters? How precise can your result be?*
- *How can you test your implementation?*

Note You will find the weights and a set of inputs to test your implementation in the Excel file [here](#): the weights are in the sheet named "Weight and bias", and the normalized inputs are in the sheet named "Normalized test inputs". The other sheets are the basis of our exercise and are there for your information, but you won't use them in your task.

You should now move to the Common final exercise for both tracks.

Track for data scientists with Python

Exercise 0: Set up your Jupyter notebook environment

Duration: 10-15 minutes

Objectives:

- Setup all the tools necessary to use SEAL
- Run your Jupyter server with port forwarding enabled

Connect to the VM with port forwarding

Because we will use Jupyter on a remote VM, we need to enable port forwarding, so that when we will connect to the localhost:8888 we will connect with the Jupyter server run in the remote VM.

This first step is common to both setups if you are working on a remote VM, and unnecessary if you install everything on your machine.

Similarly to [Task 3bis: Connect to a Linux VM in the lab](#), login to your Linux VM but this time with the port forwarding enabled, so that you will be able to use Jupyter from your browser:

```
ssh -i <private_key_file> -L 8888:127.0.0.1:8888 <user@vm>
```

Use the Docker image

In this task, you will learn how to pull the Docker image and use it as a developing environment for your Jupyter Notebooks.

Pull the image

1. Pull the image from Docker Hub (note you might need to add sudo to the docker commands):

```
docker pull danielwin/seal-workshop
```

Run the image

1. Run the Docker image with port forwarding enabled:

```
docker run -it -p 8888:8888 seal-workshop
```

2. Finally run the Jupyter notebook within the Docker image:

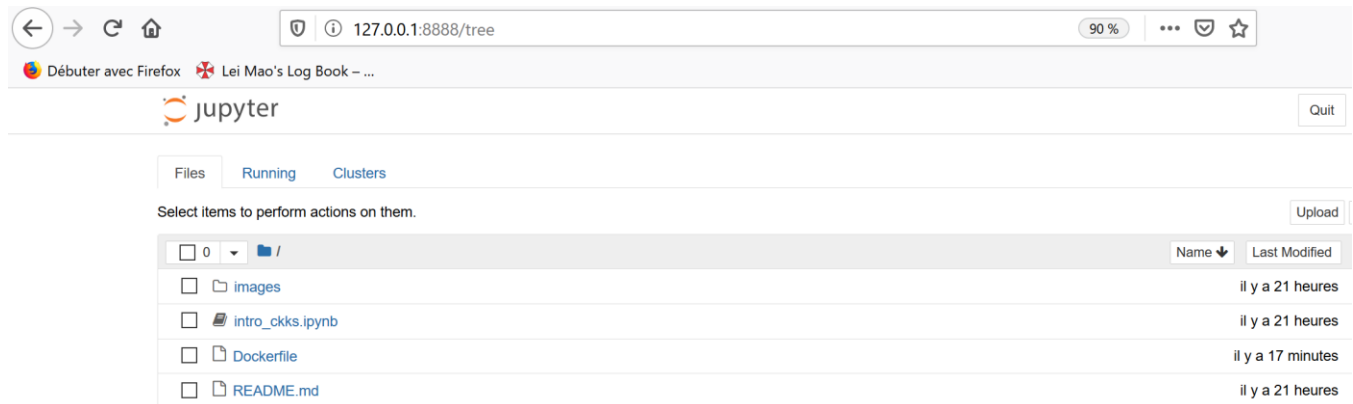
```
jupyter notebook --ip 0.0.0.0 --port 8888 --allow-root
```

This should provide the following result.

```
danywin@daniel-vm:~$ sudo docker run -it -p 8888:8888 seal-workshop
root@0c0692ba5347:/seal-workshop# jupyter notebook --ip 0.0.0.0 --port 8888 --allow-root
[I 05:35:05.886 NotebookApp] Writing notebook server cookie secret to /root/.local/share/jupyter/runtime/notebook_cookie_secret
[I 05:35:06.592 NotebookApp] Serving notebooks from local directory: /seal-workshop
[I 05:35:06.592 NotebookApp] The Jupyter Notebook is running at:
[I 05:35:06.598 NotebookApp] http://0c0692ba5347:8888/?token=1cc7eccab80137740ab72da049fe5bbbb18f4a9ca7beedd5
[I 05:35:06.598 NotebookApp] or http://127.0.0.1:8888/?token=1cc7eccab80137740ab72da049fe5bbbb18f4a9ca7beedd5
[I 05:35:06.598 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 05:35:06.602 NotebookApp] No web browser found: could not locate runnable browser.
[C 05:35:06.602 NotebookApp]

To access the notebook, open this file in a browser:
file:///root/.local/share/jupyter/runtime/nbserver-13-open.html
Or copy and paste one of these URLs:
http://0c0692ba5347:8888/?token=1cc7eccab80137740ab72da049fe5bbbb18f4a9ca7beedd5
or http://127.0.0.1:8888/?token=1cc7eccab80137740ab72da049fe5bbbb18f4a9ca7beedd5
[I 05:35:16.972 NotebookApp] 302 GET /?token=1cc7eccab80137740ab72da049fe5bbbb18f4a9ca7beedd5 (172.17.0.1) 0.51ms
```

3. Using the previous result, copy and paste one of the URLs in your browser:



Manual setup

We will see here how you can instead install all libraries on your computer. We recommend using WSL on Windows.

Install TenSEAL

1. Install Python3 and the dependencies to build SEAL:

```
sudo apt-get update && sudo apt-get install g++ make cmake git python3 python3-dev python3-pip
```

2. Git clone the TenSEAL package:

```
git clone https://github.com/OpenMined/TenSEAL.git
```

3. Go to the TenSEAL folder:

```
cd TenSEAL
```

4. Get the third-party libraries:

```
git submodule init & git submodule update
```

5. Build and install:

```
pip3 install .
```

Get the workshop notebooks

Once the Python libraries are installed, we need to get the notebooks used for the workshop.

1. Clone the notebooks:

```
git clone https://github.com/dhuynh95/seal-workshop.git
```

2. Then install Jupyter Notebook:

```
pip3 install jupyter notebook
```

3. Run your Jupyter environment:

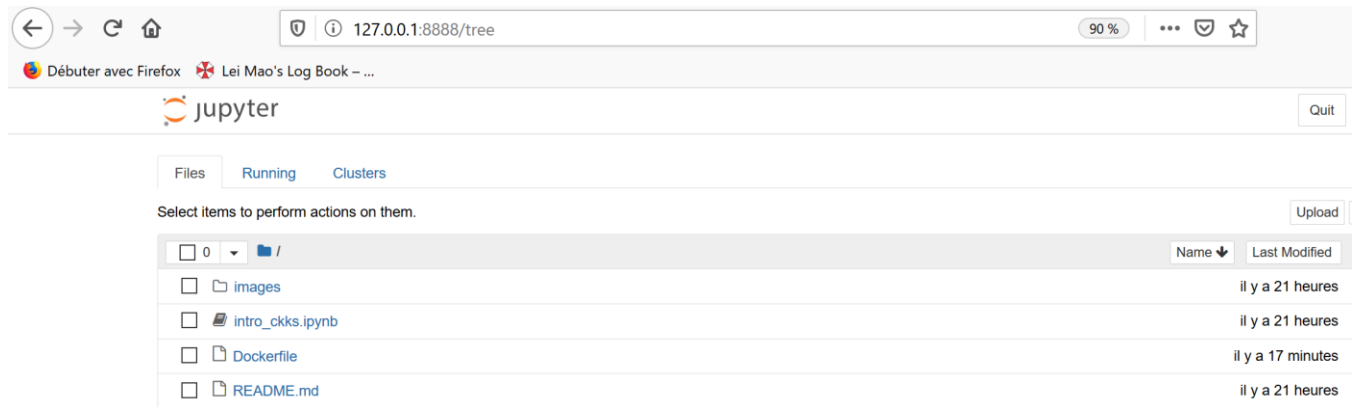
```
jupyter notebook
```

This should give you the following result:

```
danywin@daniel-vm:~/seal-workshop$ jupyter notebook
[I 09:10:50.292 NotebookApp] [jupyter_nbextensions_configurator] enabled 0.4.1
[I 09:10:50.292 NotebookApp] Serving notebooks from local directory: /home/danywin/seal-workshop
[I 09:10:50.292 NotebookApp] The Jupyter Notebook is running at:
[I 09:10:50.292 NotebookApp] http://localhost:8888/?token=46d73d73821088ef816c4a78cd5349a6dd19c5dd43009ac9
[I 09:10:50.292 NotebookApp] or http://127.0.0.1:8888/?token=46d73d73821088ef816c4a78cd5349a6dd19c5dd43009ac9
[I 09:10:50.292 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[W 09:10:50.303 NotebookApp] No web browser found: could not locate runnable browser.
[C 09:10:50.303 NotebookApp]

To access the notebook, open this file in a browser:
    file:///home/danywin/.local/share/jupyter/runtime/nbserver-11306-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/?token=46d73d73821088ef816c4a78cd5349a6dd19c5dd43009ac9
    or http://127.0.0.1:8888/?token=46d73d73821088ef816c4a78cd5349a6dd19c5dd43009ac9
```

4. Open of the URLs in your browser:



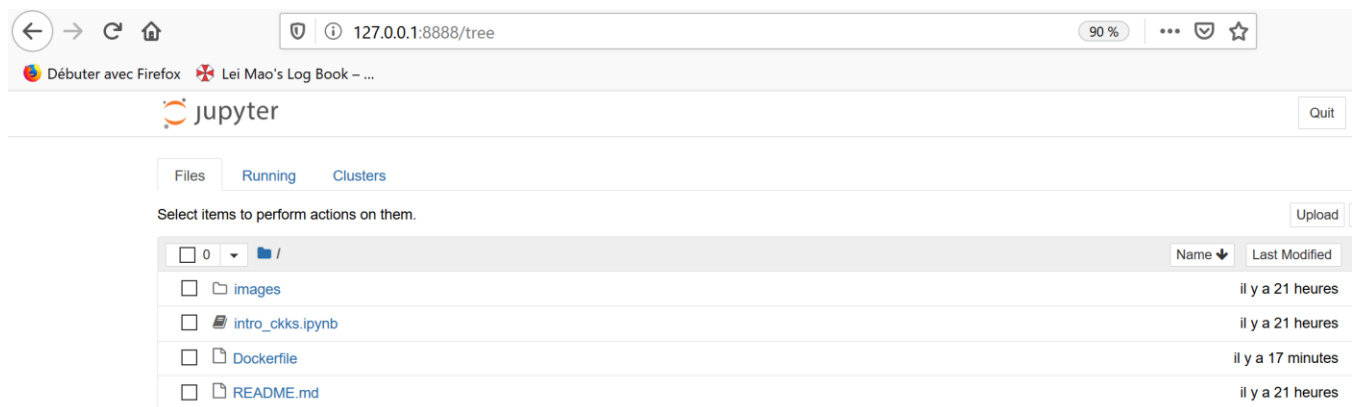
Exercise 1: Basics of CKKS

Duration: 30-60 minutes

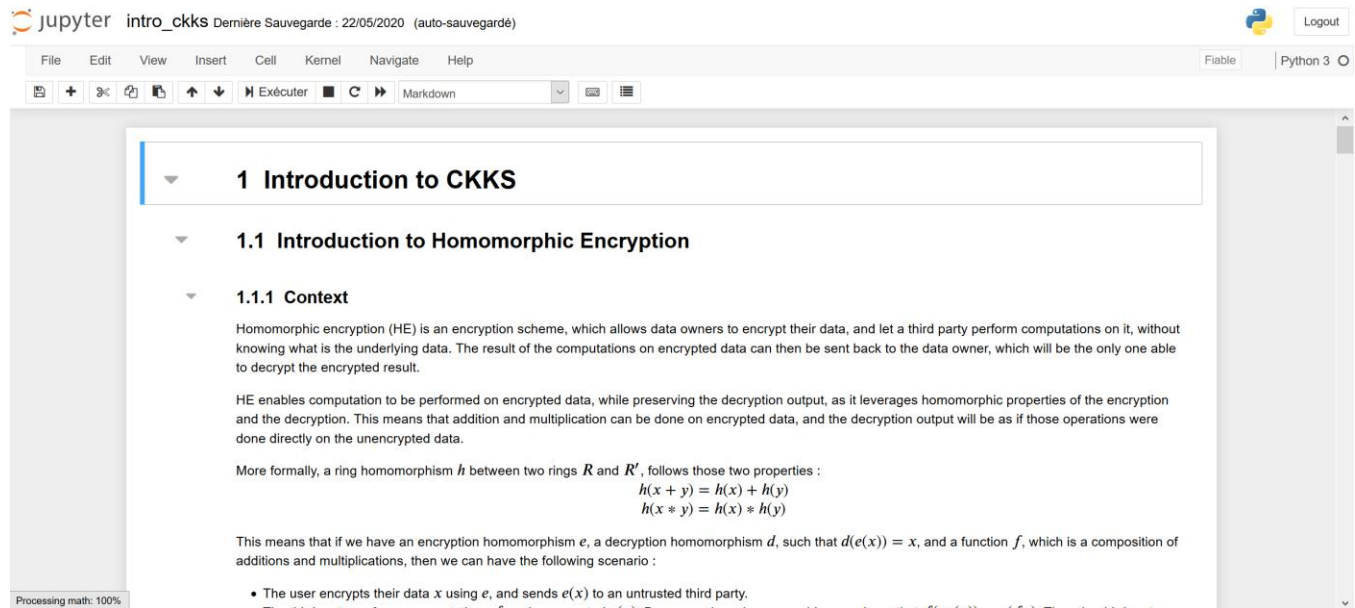
Objectives:

- Get introduced to the fundamental notions of CKKS

Once you managed to run the Jupyter server and connect to it, you will be able to run the notebook `intro_ckks.ipynb`. This notebook contains the basics of CKKS.



This is what the notebook looks like:



You should now move to the Common final exercise for both tracks.

Common final exercise for both tracks

Duration: 15 minutes

Objectives:

- Discover Homomorphic Random Forests and see how to use it on real data

You will now see how an Homomorphic Random Forest can be trained and used for inference on encrypted data. For this exercise we recommend following [Use the Docker image](#) which explains how to setup a Jupyter server within a Docker image.

Use the Docker image

We will pick up at the step after the Docker image was pulled. Don't forget to make sure that port forwarding is enabled if you are connected to a VM.

1. Go to the cryptotree library:

```
cd /cryptotree
```

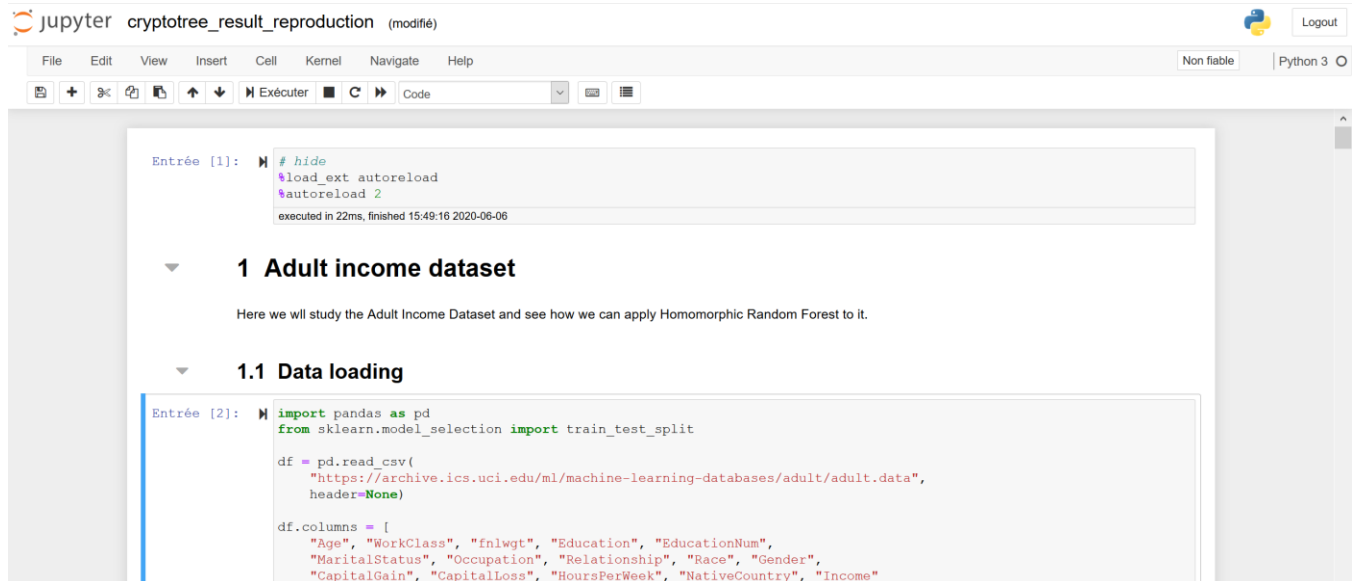
2. Then run the Jupyter server:

```
jupyter notebook --ip 0.0.0.0 --port 8888 --allow-root
```

3. Open the notebook at `examples/adult_income`:



This is how the notebook likes:



Manual setup

It is also possible to directly download the Cryptotree library available at <https://github.com/dhuynh95/cryptotree>. Because Cryptotree uses TenSEAL, you must first install it following the steps in [Manual setup](#).

1. Git clone Cryptotree:

```
git clone https://github.com/dhuynh95/cryptotree.git
```

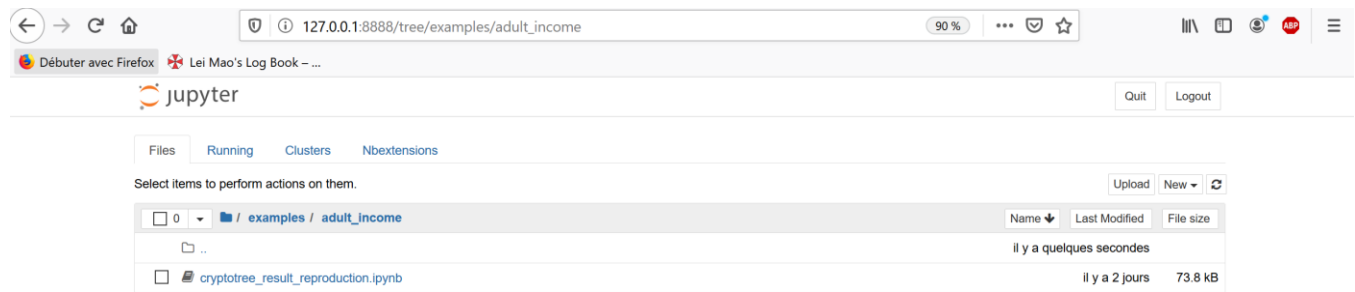
2. Then change the directory and install Cryptotree:

```
cd cryptotree
pip3 install .
```

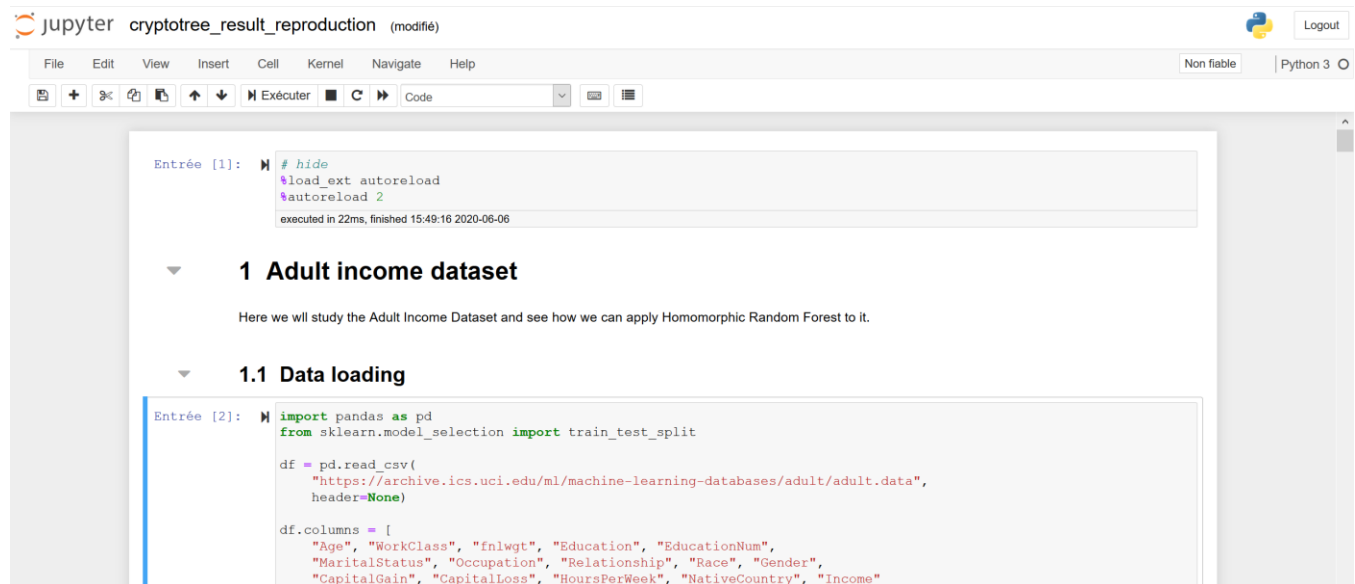
3. Then run the Jupyter server:

```
jupyter notebook --ip 0.0.0.0 --port 8888 --allow-root
```

4. Open the notebook at `examples/adult_income` :



This is how the notebook likes:



You should now move to the next and final section.

After the hands-on lab

Duration: 5 minutes

In this last "exercise", you will stop the VMs that were created in support of the lab.

Task 1: stop a virtual machine

In this task, you will need to stop the **HE-HOL-WINDEV-VM01**, and **HE-HOL-LINDEV-VM01** lab VMs.

1. Under **My Virtual Machines**, select the intended VM row, at the end of line click ..., and then select **Unclaim**.

You should follow all steps provided *after* attending the hands-on lab.

If so, this concludes this hands-on lab.

Appendix

Linux VMs private key

(REPLACE the below private key with your own value.)

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAr01BYI9BjiiWSCThMIjSt8DkRpMYSIfdWLLh/Qmb5vgN
eWZVvrlwM7DoOYWJ/gUeBFZFuRGk9/d1taxLqvJ07MKoI3SjSc69oUbJskL3OIag
Tm4+/MpGMQ9b2189Z1DcQqyeuKwFRlwmeFQS8Hs+bYtjpcdgmyloUtN6bFJEtT0T
qDD0aBOXgL5qBrD+QR+GQ6rVzNjbt9RPbXXtQtLqQfPrdPr88cR2mpddVYW7L9ey
xUD5vCZGi0qp1gV/WhWZ6+OpPIAhJTWudbEUR9FI7AUQDc9VdPoLhqSrJFBt0TY2
CXYMnjV74xI6lxYyn0hnYVcw1oKYZ2v4KHmxQIDAQABAOIBAQCo8Yb/9jwGXFNb
wL14eCI3NzDpRhMcC9hIkFibcpGsZR8PSeXAReE/ZWe051RMWw4odjCvTNSGLXL
MueIsALNifH6cFqQBHIpu+15svC+Yzem570UdFtVMD1AbdgnS0u+80sCIvr7bvH
00sr2M5oDS795Qpsd0Hqu2Tvlppb8JVskpWE49pxFU3IcJjBsQH0tsVXkWDsBLVw
dPyQtWfouCvVpW+MdQxoctzrvKa5C59R0gWYOW6oIgle2G07IFA2+RQHf56uedlR
Au7+ZSq+jGFFQbe2xGMGFzhRitJrIyfbxxraIwo4qIy/Xg3P55W5NceJWM1IRJuZ
bNMSjleBAoGBAPM0aZfE2lpvh2pBb61NXHU9yAthHQ8M800E7u0KfNYbxP/559IY
x/erBaRZYXcQzARQoUQIX9qLQj88r+250hHoJcXZnlsD05YEGlNu1obbxycannpV
own4302dMgo7/2RWsk5Yk+2k/vdNnI9ZWkfX00KXKEhyONTsuzAankr1AoGBAOSs
fli00p5sgjPvYj+i9Ss6RA7H1+0WQirU/8YafvSinICy39zZzJ7K41G1Ab2J2lEZ
YvcMB6pL62AKGU0eCRKon0duTWgizliLYcJn4U9jGzYE5HbN7w2XB61bgmGWLZJJ
tODz+Y6fZyx9bixdAyzNqIepGGkSVCfL/a0B3/1hAoGAX9YR+p7K/he1zEbK8wlq
6GuY4ivDmnifwoGsd9jHymCet9Pg/W16A4Tyr4/yp1D/MMBeJgYrTW/VijuynsjD
NP5VToL1Nqu6pfCuLjGo8vnbTVZiIJh9tePkoKTX40Mu+3Au0l+IzI5fXkHC9p7j
HyMwoPe7EX3APOyvDl0gkKUCgYEAxHtuCDsVJQCJE4TRf2pOjDkBN03KNXPrIJBp
wNcNVLfQD0kizsmZZqtFjNohR7GGE37jq0/+OhYHhTrIKJnxI8YdLawZ+KtHb487
kvuifarj05QSlf42NBAC0ZlS0vV17LdGIq+fMyvF49VWb+nvi3SeJQpm/gkQpC1D
n1U9l6ECgYEAt/EfW7xDa1F608tWXdwmEuk5WwITiQ0QJW3w8B06P0Bd9j68vKa
UwnkjHwhbEYv+HXA47fJcemJ1+GBPxMCHxIl9XiJaIEG4xHfXl7RJ7LbT5mzeQuG
hpjJfbygMXZLEYdmdxFeJ+8DFPMcZhwGfZq6ti7KVYqvJXnofFcba8=
-----END RSA PRIVATE KEY-----
```