UPES
UNIVERSITY OF TOMORROW

IT DATA SECURITY LAB FILE

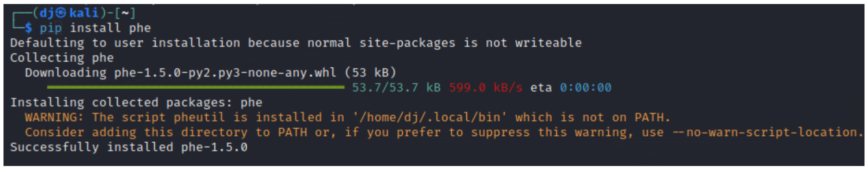
Name- Dhairya Jain
Sap ID- 500105432
Batch- CSF-B4

EXPERIMENT-10

Paet A: Homomorphic Encryption

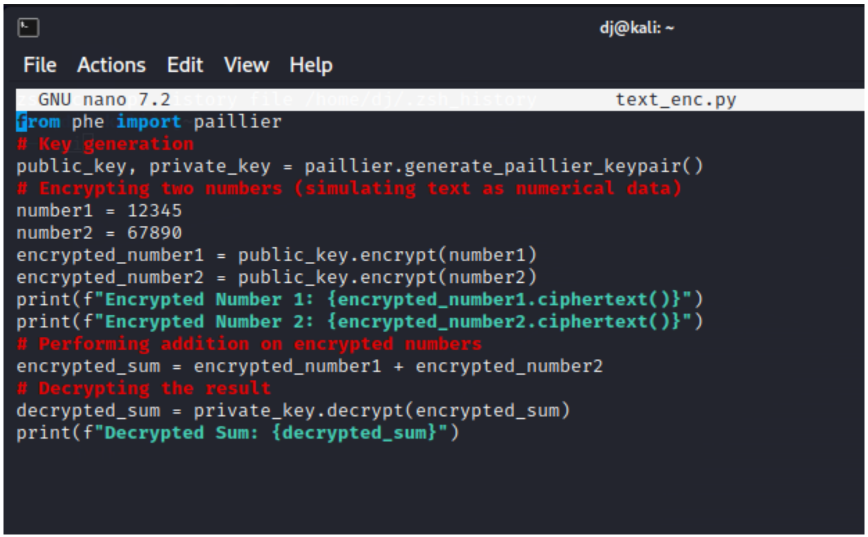
- Overview:
  Homomorphic encryption allows computations to be carried out on encrypted data without needing to decrypt it first. This means that a server or third party can process encrypted data and return results that are still encrypted. The owner of the data can then decrypt the results to obtain the outcome as if the operations were performed on the plaintext.

- Types of Homomorphic Encryption
  There are three main types of homomorphic encryption based on the operations they support:
  1. Partial Homomorphic Encryption (PHE): Supports only one type of operation (either addition or multiplication) on ciphertexts. Example: RSA (supports multiplication), Paillier (supports addition).
  2. Somewhat Homomorphic Encryption (SHE): Supports a limited number of both additions and multiplications on ciphertexts. However, the number of operations is bounded, and once the limit is reached, the ciphertext can no longer be used.
  3. Fully Homomorphic Encryption (FHE): Supports both addition and multiplication on ciphertexts without any limit. FHE allows any arbitrary computation to be performed on encrypted data.
- Applications of Homomorphic Encryption
  Homomorphic encryption is particularly useful in scenarios where sensitive data needs to be processed by a third party without exposing the actual data. Some applications include:
  - Secure Data Analysis: Performing analytics on encrypted data.
  - Cloud Computing: Encrypting data before sending it to the cloud, where computations are performed on the encrypted data.
  - Privacy-Preserving Machine Learning: Training models on encrypted data without revealing the data itself.
- Implementing Homomorphic Encryption in Python
  - Install the Python Paillier library:

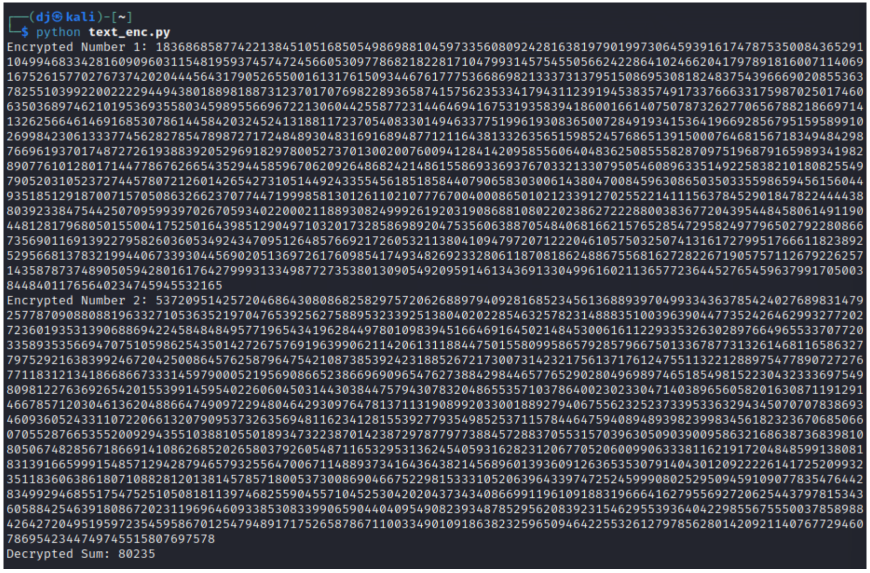


```
┌──(dj㉿kali)-[~]
└─$ pip install phe
Defaulting to user installation because normal site-packages is not writeable
Collecting phe
  Downloading phe-1.5.0-py2.py3-none-any.whl (53 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.7/53.7 kB 599.0 kB/s eta 0:00:00
Installing collected packages: phe
  WARNING: The script pheutil is installed in '/home/dj/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed phe-1.5.0
```

- Text Encryption Example Using Paillier Homomorphic Encryption

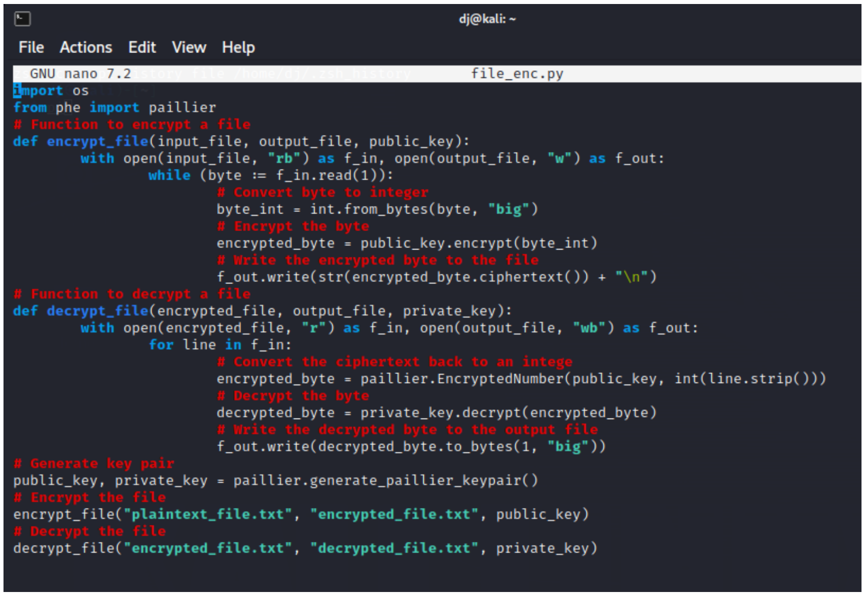


```
from phe import paillier
# Key generation
public_key, private_key = paillier.generate_paillier_keypair()
# Encrypting two numbers (simulating text as numerical data)
number1 = 12345
number2 = 67890
encrypted_number1 = public_key.encrypt(number1)
encrypted_number2 = public_key.encrypt(number2)
print(f"Encrypted Number 1: {encrypted_number1.ciphertext()}")
print(f"Encrypted Number 2: {encrypted_number2.ciphertext()}")
# Performing addition on encrypted numbers
encrypted_sum = encrypted_number1 + encrypted_number2
# Decrypting the result
decrypted_sum = private_key.decrypt(encrypted_sum)
print(f"Decrypted Sum: {decrypted_sum}")
```

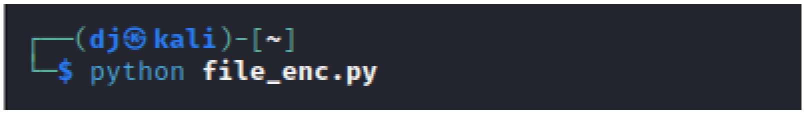


```
┌──(dj㉿kali)-[~]
└─$ python text_enc.py
Encrypted Number 1: 183686858774221384510516850549869881045973356080924281638197901997306459391617478753500843652911
1049946833428160909603115481959374574724566053097786821822817104799314575455056624228641024662041797891816007114069
1675261577027673742020444564317905265500161317615093446761777536686982133373137951508695308182483754396669020855363
7825510399220022229449438018898188731237017076982289365874157562353341794311239194538357491733766633175987025017460
6350368974621019536935580345989556696722130604425587723144646941675319358394186001661407507873262770656788218669714
1326256646146916853078614458420324524131881172370540833014946337751996193083650072849193415364196692856795159589910
2699842306133377456282785478987271724848930483169168948771211643813326356515985245768651391500076468156718349484298
7669619370174872726193883920529691829780052737013002007600941284142095855606404836250855582870975196879165989341982
8907761012801714477867626654352944585967062092648682421486155869336937670332133079505460896335149225838210180825549
7905203105237274457807212601426542731051449243355456185185844079065830300614380470084596308650350335598659456156044
9351851291870071570508632662370774471999858130126110210777670040008650102123391270255221411156378452901847822444438
8039233847544250709599397026705934022000211889308249992619203190868810802202386272228800383677204395448458061491190
4481281796805015500417525016439851290497103201732858698920475356063887054840681662157652854729582497796502792280866
7356901169139227958260360534924347095126485766921726053211380410947972071222046105750325074131617279951766611823892
5295668137832199440673393044569020513697261760985417493482692332806118708186248867556816272822671905757112679226257
1435878737489050594280161764279993133498772735380130905492095914613436913304996160211365772364452765459637991705003
84484011765640234745945532165
Encrypted Number 2: 53720951425720468643080868258297572062688979409281685234561368893970499334363785424027689831479
2577870908808819633271053635219704765392562758895323392513804020228546325782314888351003963904477352426462993277202
7236019353139068869422458484849577196543419628449780109839451664691645021484530061611229335326302897664965533707720
3358935356694707510598625435014272675769196399062114206131188447501558099586579285796675013367877313261468116586327
7975292163839924672042500864576258796475421087385392423188526721730073142321756137176124755113221288975477890727276
7711831213418668667333145979000521956908665238669690965476273884298446577652902804969897465185498152230432333697549
8098122763692654201553991459540226060450314430384475794307832048655357103786400230233047140389656058201630871191291
4667857120304613620488664749097229480464293097647813711319089920330018892794067556232523733953363294345070707838693
4609360524331107220661320790953732635694811623412815539277935498525371157844647594089489398239983456182323670685066
0705528766535520092943551038810550189347322387014238729787797738845728837055315703963050903900958632168638736839810
8050674828567186691410862685202658037926054871165329531362454059316282312067705206009906333811621917204848599138081
8313916659991548571294287946579325564700671148893734164364382145689601393609126365353079140430120922226141725209932
3511836063861807108828120138145785718005373008690466752298153331052063964339747252459990802529509459109077835476442
8349929468551754752510508181139746825590455710452530420204373434086699119610918831966641627955692720625443797815343
6058842546391808672023119696460933853083399065904404095490823934878529562083923154629553936404229855675550037858988
4264272049519597235459586701254794891717526587867110033490109186382325965094642255326127978562801420921140767729460
7869542344749745515807697578
Decrypted Sum: 80235
```

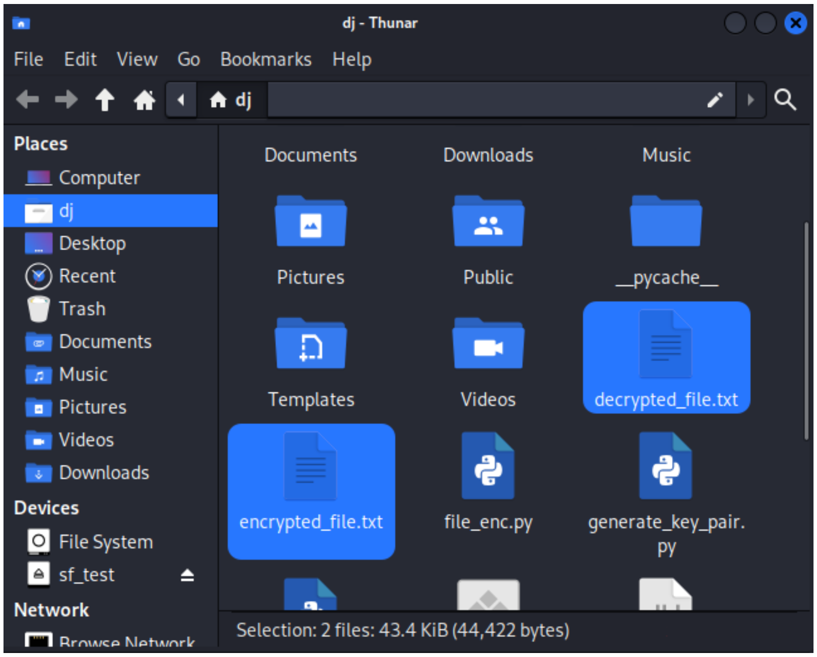
- File Encryption Example Using Paillier Homomorphic Encryption



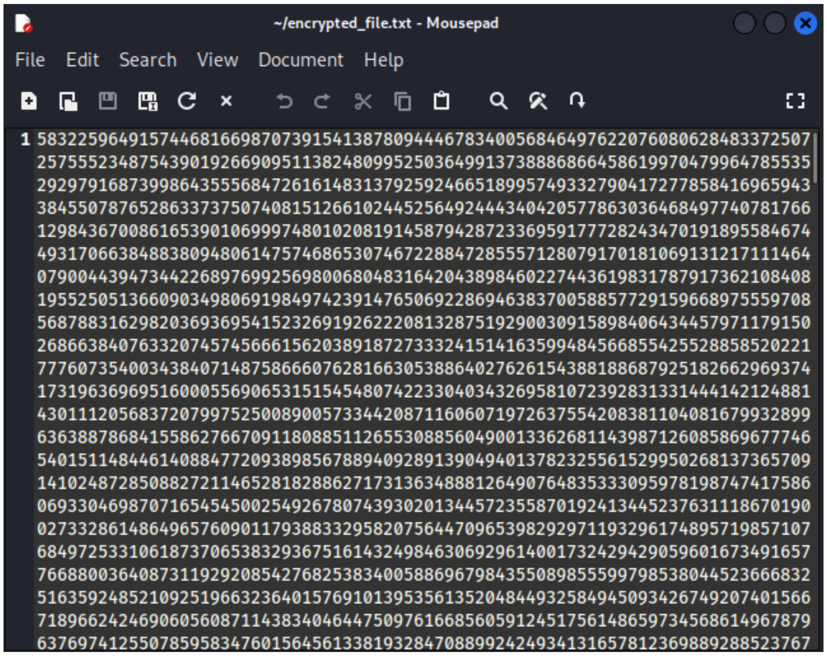
```
import os
from phe import paillier
# Function to encrypt a file
def encrypt_file(input_file, output_file, public_key):
        with open(input_file, "rb") as f_in, open(output_file, "w") as f_out:
                while (byte := f_in.read(1)):
                        # Convert byte to integer
                        byte_int = int.from_bytes(byte, "big")
                        # Encrypt the byte
                        encrypted_byte = public_key.encrypt(byte_int)
                        # Write the encrypted byte to the file
                        f_out.write(str(encrypted_byte.ciphertext()) + "\n")
# Function to decrypt a file
def decrypt_file(encrypted_file, output_file, private_key):
        with open(encrypted_file, "r") as f_in, open(output_file, "wb") as f_out:
                for line in f_in:
                        # Convert the ciphertext back to an intege
                        encrypted_byte = paillier.EncryptedNumber(public_key, int(line.strip()))
                        # Decrypt the byte
                        decrypted_byte = private_key.decrypt(encrypted_byte)
                        # Write the decrypted byte to the output file
                        f_out.write(decrypted_byte.to_bytes(1, "big"))
# Generate key pair
public_key, private_key = paillier.generate_paillier_keypair()
# Encrypt the file
encrypt_file("plaintext_file.txt", "encrypted_file.txt", public_key)
# Decrypt the file
decrypt_file("encrypted_file.txt", "decrypted_file.txt", private_key)
```



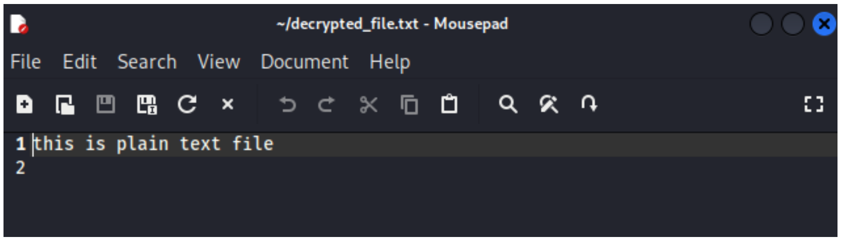
```
┌──(dj㉿kali)-[~]
└─$ python file_enc.py
```



```
Selection: 2 files: 43.4 KiB (44,422 bytes)
```



```
5832259649157446816698707391541387809444678340056846497622076080628483372507
2575552348754390192669095113824809952503649913738886866458619970479964785535
2929791687399864355568472616148313792592466518995749332790417277858416965943
3845507876528633737507408151266102445256492444340420577863036468497740781766
1298436700861653901069997480102081914587942872336959177728243470191895584674
4931706638488380948061475746865307467228847285557128079170181069131217111464
0790044394734422689769925698006804831642043898460227443619831787917362108408
1955250513660903498069198497423914765069228694638370058857729159668975559708
5687883162982036936954152326919262220813287519290030915898406434457971179150
2686638407633207457456661562038918727333241514163599484566855425528858520221
7776073540034384071487586660762816630538864027626154388188687925182662969374
1731963696951600055690653151545480742233040343269581072392831331444142124881
4301112056837207997525008900573344208711606071972637554208381104081679932899
6363887868415586276670911808851126553088560490013362681143987126085869677746
5401511484461408847720938985678894092891390494013782325561529950268137365709
1410248728508827211465281828862717313634888126490764835333095978198747417586
0693304698707165454500254926780743930201344572355870192413445237631118670190
0273328614864965760901179388332958207564470965398292971193296174895719857107
6849725331061873706538329367516143249846306929614001732429429059601673491657
7668800364087311929208542768253834005886967984355089855599798538044523666832
5163592485210925196632364015769101395356135204844932584945093426749207401566
7189662424690605608711438340464475097616685605912451756148659734568614967879
6376974125507859583476015645613381932847088992424934131657812369889288523767
```



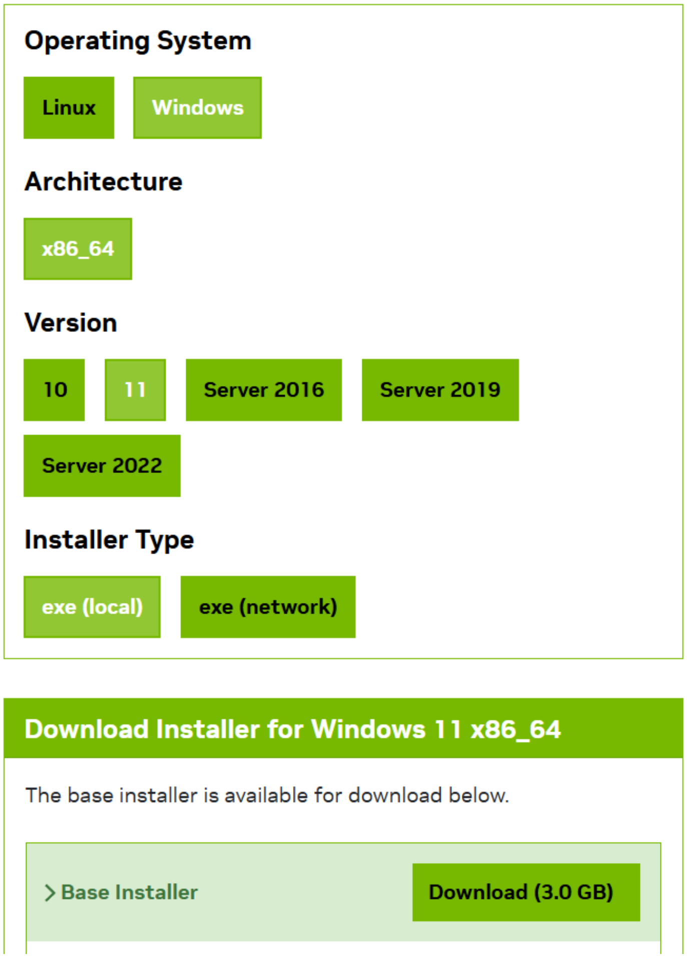
```
this is plain text file
```

Part B- CUDA Installation

Installing CUDA on Windows

- Verify System Requirements
  - Operating System: Windows 10 or 11 (64-bit).
  - Supported NVIDIA GPU: Check the list of supported GPUs on the NVIDIA CUDA website.
  - Visual Studio: The CUDA Toolkit requires Visual Studio. You can install Visual Studio Community Edition for free if you don't already have it.

- Download the CUDA Toolkit
  - Visit the CUDA Toolkit Download Page: Go to the CUDA Toolkit Downloads page.
  - Select Your Operating System: Choose Windows as the operating system, and select the appropriate version (e.g., Windows 10, 64-bit).
  - Download the Installer: Download the installer for the CUDA Toolkit version that matches your environment



Operating System

Linux [✓ Windows]

Architecture

[✓ x86_64]

Version

10 [✓ 11] Server 2016 Server 2019 Server 2022

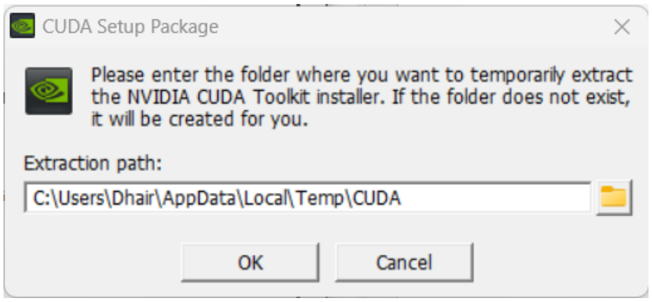
Installer Type

[✓ exe (local)] exe (network)

Download Installer for Windows 11 x86_64

The base installer is available for download below.

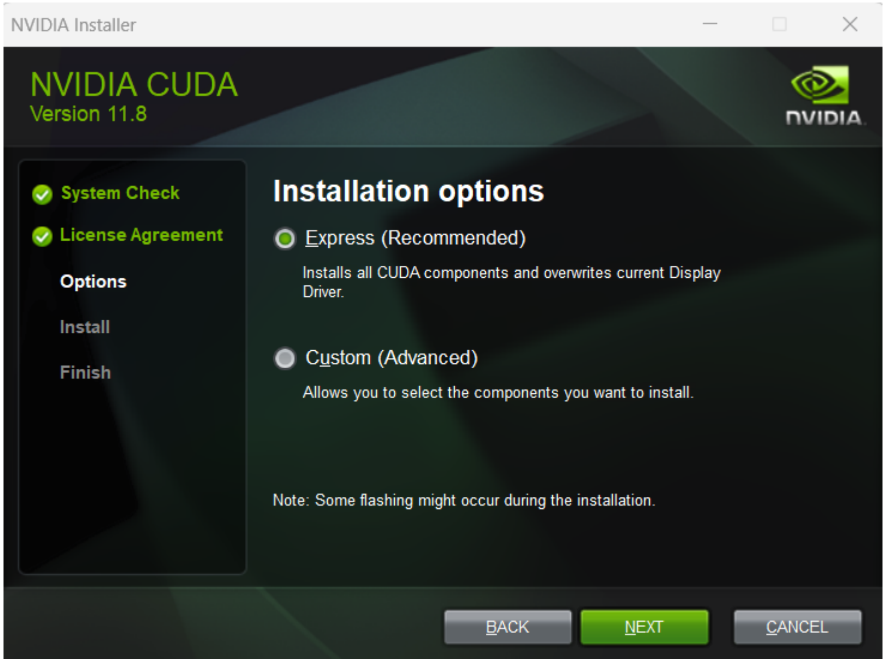
- Install the CUDA Toolkit
  - Run the Installer: Locate the downloaded installer and run it.



Please enter the folder where you want to temporarily extract the NVIDIA CUDA Toolkit installer. If the folder does not exist, it will be created for you.

Extraction path: C:\Users\Dhair\AppData\Local\Temp\CUDA

  - Choose Installation Options: You can choose between an "Express" installation (recommended) or a "Custom" installation if you want to select specific components.
  - The installation includes the CUDA Toolkit, NVIDIA driver, CUDA samples, and more.



Installation options

● Express (Recommended)

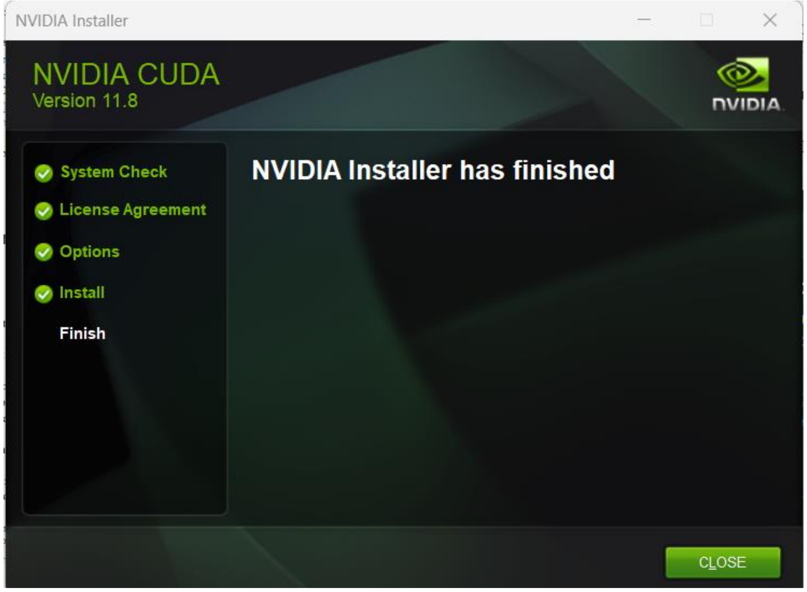
Installs all CUDA components and overwrites current Display Driver.

○ Custom (Advanced)

Allows you to select the components you want to install.

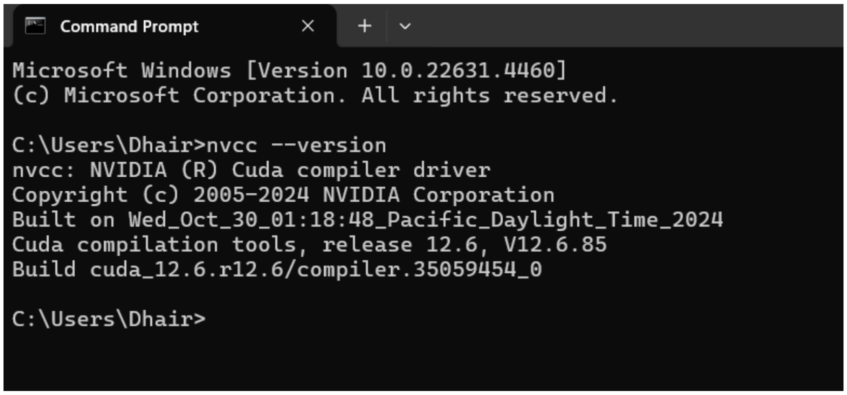
Note: Some flashing might occur during the installation.

  - Complete the Installation: Follow the on-screen instructions to complete the installation.



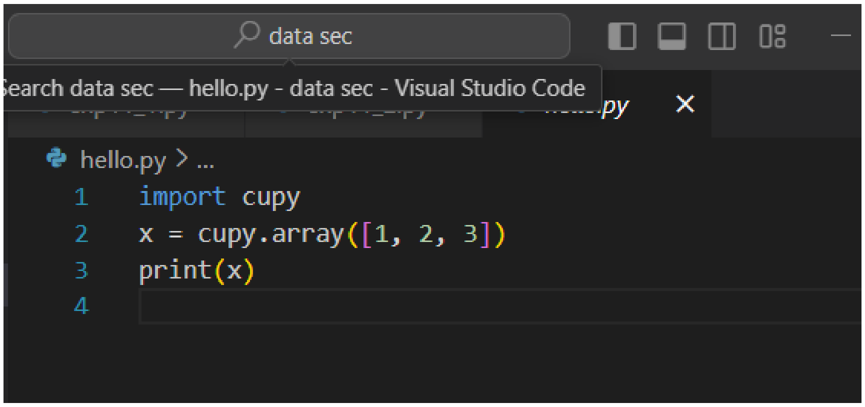
NVIDIA Installer has finished

- Verify the Installation
  - Open a Command Prompt: Open Command Prompt and run the following command to verify that CUDA is installed correctly

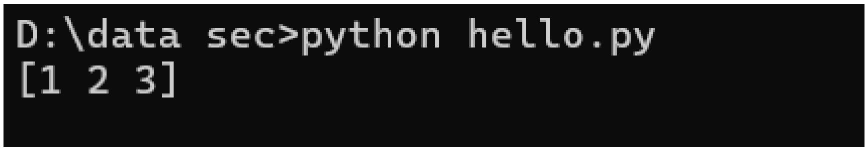


```
Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dhair>nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2024 NVIDIA Corporation
Built on Wed_Oct_30_01:18:48_Pacific_Daylight_Time_2024
Cuda compilation tools, release 12.6, V12.6.85
Build cuda_12.6.r12.6/compiler.35059454_0

C:\Users\Dhair>
```

  - Run CUDA Samples



```python
import cupy
x = cupy.array([1, 2, 3])
print(x)

```



```
D:\data sec>python hello.py
[1 2 3]
```