Privacy Preservation with Homomorphic Encryption (HE) Lab

Copyright © 2021 Anyi Liu and Xinyi Li, Oakland University.
The development of this document is funded by the following grants from the US National Science Foundation: No. 1723707 and 1623713, and Michigan Space Grant Consortium (MSGC) Research Seed Grant and Oakland University Research Seed Grant. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. If you remix, transform, or build upon the material, this copyright notice must be left intact, or reproduced in a way that is reasonable to the medium in which the work is being re-published.

1 Overview

This lab will help students build a better understanding of the purpose of HE and the confidentiality issues in a supply chain. In particular, students will learn 1) the basic HE operations; 2) the applications of using HE cryptographic operations in command-line and program level; 3) the real-world problem solving with HE. The learning objectives of this lab are listed below:

1. Be able to use HE cryptographic operations in command-line and program level.
2. Be able to solve real-world problems with HE.

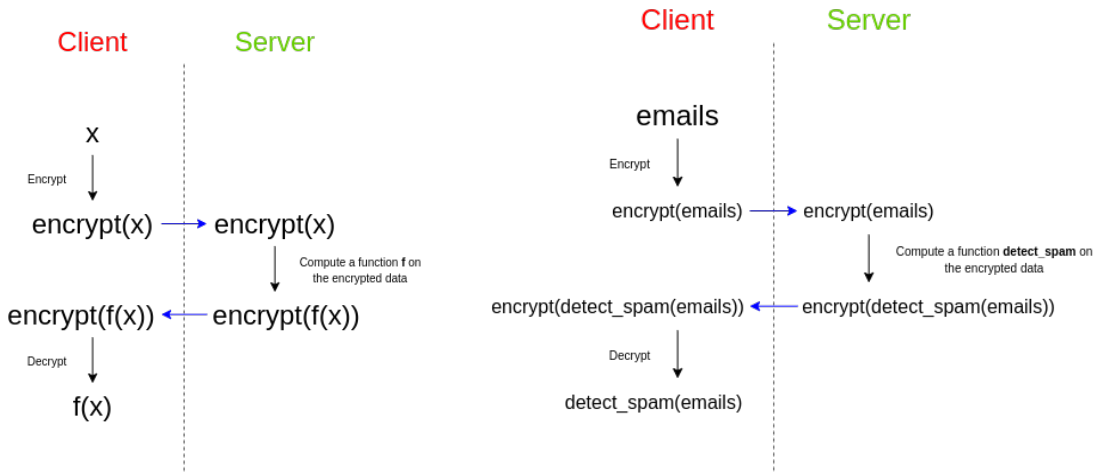
2 Background

Homomorphic Encryption (HE): HE is a special class of encryption technique that allows computations to be conducted on encrypted data (ciphertext) without requiring a key to decrypt it. HE was first envisioned in 1978 [1] and constructed in 2009 [2]. By applying HE to protect the customer's private data in untrustworthy storage, the cloud service can perform the computation directly on the ciphertext, which guarantees data privacy.

Homomorphic Encryption: Partial and Fully **[3]:**

Fully homomorphic encryption (FHE) has for a long time been considered one of the holy grails of cryptography. The promise of FHE is powerful: it is a type of encryption that allows a third party to perform computations on encrypted data, and get an encrypted result that they can hand back to whoever has the decryption key for the original data, without the third party being able to decrypt the data or the result themselves. The idea of FHE is illustrated in Figure 1a.

As a simple example, imagine that you have a set of emails, and want to use a third-party spam filter to check whether or not they are spam. The spam filter has a desire for privacy of their algorithm: either the spam filter provider wants to keep their source code closed, or the spam filter depends on a very large database that they do not want to reveal publicly as that would make attacking easier, or both. However, you care about the privacy of your data, and don't want to upload your unencrypted emails to a third party. So here's how you do it as illustrated in Figure 1b.



(a) The idea of FHE.

(b) An example of the FHE application.

Figure 1: The idea and an example of FHE.

In general, a FHE cryptosystem supports both addition and multiplication operations, while a partially homomorphic encryption (PHE) may only enable one of them (e.g. Paillier cryptosystem is one implementation of partially homomorphic encryption that supports only addition operation).

3 Task 1: Lab Set-up

First, you need to setup the environment for this lab. Please use the following command to download and install `Python` and `pip`:

```
# Install Python3
$ sudo apt install python3-dev

# To avoid any future confusion, please use "python3",
# instead of "python" when you run Python (.py) programs.
```

Figure 2: The command line to install `Python`.

Then, you need to install a toolkit, namely `python-paillier` as follows:

```
# Install Python3 pip (pip3) as a package installer
$ sudo apt install python3-pip

# Install phe[cli], partial HE command line toolkit
$ pip3 install phe[cli]

# Check the version of pheutil
$ pheutil --version
   pheutil, version 1.0-alpha

# Install Pyfhel
$ pip3 install Pyfhel
```

Figure 3: The command lines to install PHE toolkit.

If you can see the output, just like ``pheutil, version x.x-yyyy'', it indicates that the toolkit has been installed successfully.

Now, create a virtual Python environment named he-lab.

```
# Install Python virtual environment
$ pip3 install virtualenv

# Create a virtual environment, namely he-lab
$ virtualenv he-lab

# Create a directory for he-lab
$ source ~/he-lab/bin/activate

# Install phe and Pyfhel packages
$ pip3 install phe
$ pip3 install Pyfhel
```

Figure 4: The command lines to install HE toolkit for Python.

4 HE Encryption and Decryption

4.1 Task 2: A Case Study of HE - Command Line Utility `pheutil`

Let's use the following example to demonstrate the application of HE. First, generate a private key pair, namely private_key.json (the private key) and public_key.json (the public key).

```
# Generate the private key, namely private_key.json
$ pheutil genpkey --keysize 1024 private_key.json

# Generate the public key, namely public_key.json, based on the private key
$ pheutil extract private_key.json public_key.json
```

Figure 5: The command lines to generate a HE private and public key pair.

Then, use the public key to encrypt two numbers, say 4480 and 1420 with the public key and export their ciphertext as `num1.enc` and `num2.enc`, respectively.

```
# Encrytp integers 4480 and 1420.
$ pheutil encrypt --output num1.enc public_key.json 4480
$ pheutil encrypt --output num2.enc public_key.json 1420

# Add two nubmers.
$ pheutil addenc --output sum.enc public_key.json num1.enc num2.enc

# Decrypt the summation.
$ pheutil decrypt private_key.json sum.enc
```

Figure 6: Encrypt and decrypt the summation of two numbers in ciphertext.

Deliverable 1: Please print out the ciphertext content of `num1.enc`, `num2.enc`, and `sum.enc`. You can use Linux's `cat` command to print out the content.

Deliverable 2: Please print out the decryption result.

4.2 Task 3: HE Exercises - Python Programming

First, let's read the following Python code (`fully-basic.py`) with given comments in Listing 1. In this example, you can see the step-by-step process of how to facilitate HE operations, such as addition, subtraction, and multiplication. Remember, don't copy and paste the line numbers of the Python code.

```
1  from Pyfhel import Pyfhel
2
3  # Create an empty Pyfhel object
4  HE = Pyfhel()
5
6  # Initialize a context with plaintext modulo 65537
7  HE.contextGen(p=65537)
8
9  # Generate a (public/private) key pair
10 HE.keyGen()
11
```

```python
# Encrypt two integers
num1 = HE.encryptInt(114)
num2 = HE.encryptInt(514)

# Print the last 16 bytes of ciphertexts
print(f"114 is encrypted as ...{num1.to_bytes()[-16:].hex()}")
print(f"514 is encrypted as ...{num2.to_bytes()[-16:].hex()}")

# Add two ciphertexts
cipher_sum = num1 + num2
plain_sum = HE.decrypt(cipher_sum, decode_value=True)
print(f"Their sum is encrypted as ...{cipher_sum.to_bytes()[-16:].hex()}")
print(f"decrypted sum: {plain_sum}")

# Subtract two ciphertexts
cipher_sub = num2 - num1
plain_sub = HE.decrypt(cipher_sub, decode_value=True)
print(
    f"Their difference is encrypted as ...{cipher_sub.to_bytes()[-16:].hex()}"
)
print(f"decrypted difference: {plain_sub}")

# Multiply two ciphertexts
cipher_mul = num1 * num2
plain_mul = HE.decrypt(cipher_mul, decode_value=True)
print(f"Their product is encrypted as ...{cipher_mul.to_bytes()[-16:].hex()}")
print(f"decrypted product: {plain_mul}")
```

Listing 1: The Python code of `fully-basic.py` for basic HE operations.

You can run the following command line and get the result.

```
$ python3 fully-basic.py
114 is encrypted as ...ae5427aaae5d20000000000000000000
514 is encrypted as ...17d21728c18220000000000000000000
Their sum is encrypted as ...c4263fd36fe00000000000000000000
decrypted sum: 628
Their difference is encrypted as ...697df07d1225000000000000000000
decrypted difference: 400
Their product is encrypted as ...42f550943c273d000000000000000000
decrypted product: 58596
```

Figure 7: The command line to run `fully-basic.py`.

Deliverable 3: Please copy and paste the screenshot that shows you can run `fully-basic.py` successfully.

4.3 Task 4: HE Exercises - Advanced Python Programming

Now, let's walk through a *fictional post-credit scene* of "*God of War (2018)*" [1]. Let's say, Kratos collects four Oracles in his journey [2]. The Oracles are used to compute the Year of *"Ragnarök"* [3]. In order to reveal the secret hidden in Oracles, he needs to pass these Oracles to the *King of Gods*, Odin, and use his magic power to find out the answer. However, Odin is *malicious*, who will kill prophets if he can read the plaintext of Oracles. Thus, Kratos must first encrypts Oracles with his public key. Then, he ask Odin to calculate the following simple formula, of course with Kratos' public key, without tell Odin what does it mean.

$$[(O_1 - O_2) + (O_3 - O_4)] \times 40$$

After calculating, Kratos will get the answer, in ciphertext, and use his private key to find out the secret.

To make your programming easier, we provide the Python code snippet in Appendices.

Deliverable 4 [Extra Credit]: Download *he_gow.zip* from Moodle, which contains the following files:

- **Oracle files**: O_i`.oracle (i = 1..4)`
- **The context for the cryptosystem**: `context`
- **Kratos' public key**: `pub.key`
- **Kratos' private key**: `private.key`

Your job is to write a Python program, namely "**Odin.py**", by playing as Odin. Submit the calculating result. Since Odin don't know the secret Kratos' private, you cannot use Kratos' private key for this job. Please use *"fhe-example1.py"* for your reference.

Deliverable 5 [Extra Credit]: Now, play as Kratos, namely "**Kratos.py**", by playing as Kratos. Use his private key to decypet the secret and tell the Year of *"Ragnarök"*. Please use *"he-example2.py"* for your reference.

[1] https://godofwar.fandom.com/wiki/God_of_War_(2018)

[2] You can consider that Oracles are encrypted as ciphertext, which are denoted as O_i where $i = 1, 2, 3,$ or 4

[3] https://en.wikipedia.org/wiki/Ragnar%C3%B6k

5 Acknowledgement

We thank Frank Wang for proofreading the initial draft of this document and his constructive suggestions.

Appendices

A The Python code for advanced HE operations

The first Python code (`he-example1.py`) includes the basic programming API required for HE operation, such as restoring a context, adding two ciphertexts, and saving the summation into a file. Remember, don't copy and paste the line numbers of the Python code.

```
1
2  from Pyfhel import PyCtxt,Pyfhel
3
4  # Restore the context, which must be specified to conduct operations among
       encrypted numbers
5  HE = Pyfhel()
6  HE.restoreContext("context")
7
8  # Import two ciphertext
9  cipher1 = PyCtxt(fileName="cipher1.text",encoding="int",pyfhel=HE)
10 cipher2 = PyCtxt(fileName="cipher2.text",encoding="int",pyfhel=HE)
11
12
13 # Add them
14 result = (cipher1-cipher2)
15
16 # Save the ciphertext result
17 result.save("sum.text")
```

Listing 2: The Python code example `he-example1.py` for advanced HE operations.

The second Python code (`he-example2.py`) includes the basic programming API required for HE operation, such as restoring a context and the private key, deciphering the ciphertext, and printing out the plaintext result. Remember, don't copy and paste the line numbers of the Python code.

```
1
2  from Pyfhel import PyCtxt, Pyfhel
3
4  # Recover context and private key from files
5  HE = Pyfhel()
6  HE.restoreContext("context")
7  HE.restoresecretKey("private.key")
8
9  # Read the ciphertext and decrypt it
10 res_ctxt = PyCtxt(fileName="sum.text", encoding="int")
11 res_ptxt = HE.decrypt(res_ctxt, decode_value=True)
```

```
12
13 # Print out the result in plaintext
14 print("The result is", res_ptxt)
```

Listing 3: The Python code example `he-example2.py` for advanced HE operations.

References

[1] R. L. Rivest, L. Adleman, M. L. Dertouzos, On data banks and privacy homomorphisms, Foundations of Secure Computation, Academia Press (1978) 169–179.

[2] C. Gentry, Fully homomorphic encryption using ideal lattices, in: M. Mitzenmacher (Ed.), STOC, ACM, 2009, pp. 169–178.

[3] V. Buterin, Exploring fully homomorphic encryption, `https://vitalik.ca/general/2020/07/20/homomorphic.html` ([Accessed: July 10, 2021]).