CS 546 Computer Security (Spring 2018)

Lab Assignment #1, Due on 4/02/2018, Monday (11PM)

Introduction:

Fully Homomorphic Encryption (FHE) has been studied for a long time. Until recently, this research became a hot topic because such algorithms can be specifically applied in cloud computing for data privacy protection. In 2009, Craig Gentry using lattice-based cryptography showed the first FHE algorithm. Two variants of his algorithm were later proposed in the literature. However, these algorithms are not practical due to their prohibitively expensive computing cost.

To be a FHE algorithm, its ciphertext can be directly applied to arithmetic operations $(+, \times)$ without messing up the results. That is, $E(m_1) + E(m_2)$ "=" $E(m_1 + m_2)$ and $E(m_1) \times E(m_2)$ "=" $E(m_1 \times m_2)$. The quotes around the equal sign above indicate that the two sides don't have to be quantitatively equal. Two ciphers are said equal if they can be tested equal, provided a testing procedure.

Description:

This assignment asks you to implement an FHE algorithm (name it FHEv1) and its variant (FHEv2) that we discussed in classes. You need to download a C package that can handle big digits. The package can be copied from the directory in onyx

```
~jhyeh/cs546/labs/lab1/files/BigDigits.2.2.0.zip
```

For FHEv1, your program should provide the following options:

- FHEv1 -k <key size> <KeyFileName>: Generating the encryption/decryption keys (P_1, N) and the operational keys (N, g_1, g_2, T), and write them to a key file. The <key size> is the size (number of bits) of P_1, P_2 and P_3, where $N = P_1 \cdot P_2$ and $T = (2P_1 + 1) \cdot P_3$.
- FHEv1 -e <m> <KeyFileName>: Read keys from the key file and then encrypting a plain integer m (must be less than the decryption key P_1). Your program should output C_m (the cipher of m).
- FHEv1 -d $<C_m>$ <KeyFileName>: Read keys from the key file and then decrypting the cipher C_m. Your program should output the plaintext m.
- FHEv1 -b <m> <KeyFileName>: Read keys from the key file and then encrypting and decrypting an integer m. The output of this option will be printing the input m, the cipher C_m, and then the decrypted m (each in a different line).
- FHEv1 -a <-e $<m_1>$ | C_{m_1}>, <-e $<m_2>$ | C_{m_2}> <KeyFileName>: Read keys from the key file and then add two ciphers C_{m_1} and C_{m_2}. With -e option, encrypt m before the add operation. Output the add result.
- FHEv1 -m <-e $<m_1>$ | C_{m_1}>, <-e $<m_2>$ | C_{m_2}> <KeyFileName>: Read keys from the key file and then multiply two ciphers C_{m_1} and C_{m_2}. With -e option, encrypt m before the multiply operation. Output the multiply result.

- FHEv1 -t <-e <m_1> | C_{m_1}>, <-e <m_2> | C_{m_2}> <KeyFileName>: Read keys from the key file and then test the equality of two ciphers C_{m_1} and C_{m_2}. With -e option, encrypt m before the equality testing operation. Output the equality test result.

For FHEv2, the program requires two command-line arguments w (bit-size for the maximal application data) and z (number of random padding bits). **Your implementation must ensure at least 5 consecutive homomorphic multiplications.** Your program should provide the following options:

- FHEv2 -k <key size> <w> <z> <KeyFileName>: Generating the encryption/decryption keys $(P_1, N, 2^w)$, the operational key (N), the user-defined constants (w, z), and write them to a key file. Again, the <key size> is the size of P_1.
- FHEv2 -p <m> <KeyFileName>: Given a plain integer, this option reads the constants from the key file and then perform a random padding to m. The padded integer is $m'(< P_1)$.
- FHEv2 -e <m> <KeyFileName>: Read keys from the key file and then encrypting a plain integer m (must be less than the decryption key 2^w). Your program should output C_m (the cipher of m).
- FHEv2 -d <C_m> <KeyFileName>: Read keys from the key file and then decrypting the cipher C_m. Your program should output the plaintext m.
- FHEv2 -b <m> <KeyFileName>: Read keys from the key file and then encrypting and decrypting an integer m. The output of this option will be printing the input m, the cipher C_m, and then the decrypted m (each in a different line).
- FHEv2 -a <-e <m_1> | C_{m_1}>, <-e <m_2> | C_{m_2}> <KeyFileName>: Read keys from the key file and then add two ciphers C_{m_1} and C_{m_2}. With -e option, encrypt m before the add operation. Output the add result.
- FHEv2 -m <-e <m_1> | C_{m_1}>, <-e <m_2> | C_{m_2}> <KeyFileName>: Read keys from the key file and then multiply two ciphers C_{m_1} and C_{m_2}. With -e option, encrypt m before the multiply operation. Output the multiply result.

Submission

Please provide a README file describing how to run your C program(s). In the README file, you should also briefly describe your thought about the algorithm, that may includes the security weakness of the algorithm and how you can improve the algorithm. Submit your programs, along with the README file, from onyx by copying all the files to an empty directory (with no subdirectories) and typing the following FROM WITHIN this directory:

```
submit jhyeh cs546 p1
```