2019 국가암호공모전 Ⅱ 분야

문제 05

2009년에 처음 개발된 암호문간의 연산이 가능한 동형암호 기법은 암호화된 데이터 간의 연산이 가능하다는 특징을 가지는 기법이다. 암호화된 데이터의 처리를 위한 복호화가 없기 때문에 동형암호 기법은 데이터의 완전한 보호를 위한 중요한 도구로 떠오르고 있다. 본 문제에서는 주어진 암호화된 실수값에 대해서 복호화 과정 없이 Sigmoid 함수를 계산하는 과정을 근사계산 동형암호[1] 라이브러리를 이용해서 구현하는 것을 목표로 한다.

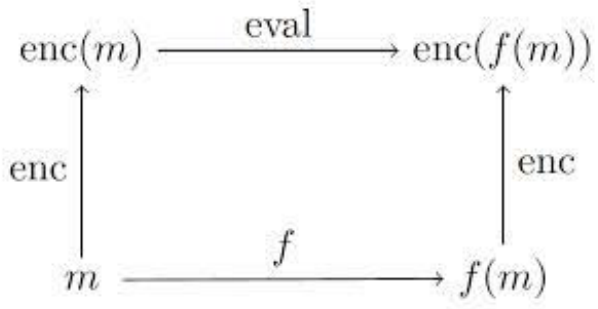

그림 1 동형암호의 개념도

동형암호는 그림 1에서 보여주는 것처럼 암호화된 데이터에 대한 연산이 가능한 기능을 가진 암호화 방법이다. 다양한 동형암호 기법 중에서 특히 근사계산 동형암호는 암호화된 실수형 데이터에 대해서 복호화 없이 연산을 수행할 수 있는 암호화 방법으로 아래 6개의 알고리즘으로 구성되어 있다. 아래에서 볼 수 있는 동형암호의 중요한 성질은 암호문 간의 연산을 수행하지만, 세 가지 연산인 덧셈, 뺄셈, 그리고 곱셈에 한정되어 있다는 것이다. (근사계산 동형암호화가 제공하는 추가적인 기능과 사용법에 대해서는 "동형암호.pdf"파일을 참고한다.)

- **KEYGEN(파라미터)**: 주어진 동형암호 파라미터에 대해서, 암호화, 복호화, 암호문 간의 연산에 필요한 키를 생성하는 과정으로 비밀키, 암호화키, 곱셈연산키를 생성한다.
- **ENC(msg, 암호화키)**: 실수값을 암호화하는 과정으로 평문 msg와 KEYGEN에서 생성된 암호화키를 이용한다.
- **DEC(ctxt, 비밀키)**: 암호문을 복호화하는 과정으로 암호문 ctxt와 KEYGEN에서 생성된 비밀키를 이용해서 복호화를 수행한다.
- **ADD(ctxt1, ctxt2)**: 암호문 간의 덧셈 연산을 수행하는 과정으로 출력되는 암호문인 ctxt_out은 m1 + m2 의 암호문이다. (여기서 ctxt1은 m1의 암호문, ctxt2는 m2의 암호문이다.)
- **SUB(ctxt1, ctxt2)**: 암호문 간의 뺄셈 연산을 수행하는 과정으로 출력되는 암호문인 ctxt_out은 m1 − m2 의 암호문이다. (여기서 ctxt1은 m1의 암호문, ctxt2는 m2의 암호문이다.)
- **MUL(ctxt1, ctxt2, 곱셈연산키)**: 암호문 간의 곱셈 연산을 수행하는 과정으로 출력되는 암호문인 ctxt_out은 m1 * m2 의 암호문이다. (여기서 ctxt1은 m1의 암호문, ctxt2는 m2의 암호문이다.)

동형암호 기법이 +, −, 그리고 × 연산만 지원한다는 것은 임의의 함수가 아닌 다항식을 암호화된 상태에서 계산할 수 있다고 생각할 수 있다. 따라서 Sigmoid와 같은 함수의 경우에는 해당 함수와 유사한 값을 주는 다항식을 찾고 이를 계산하는 방법으로 대체하는 것이 필요하다. 본 문제는 3개의 문제로 구성되어 있으며, 1번을 제외한 나머지 2개의 문제는 구현을 동반하는 문제이다. 1번 문제의 경우에는 해답으로 제시할 근사다항식을 구하는 과정과 그 이론에 대해서 상세하게 서술해야한다.

(1) 주어진 함수 $f(x)=\frac{e^x}{1+e^x}$에 대해서 $[-4,4]$ 범위 상에서의 근사다항식 $p(x)$을 구하고 해당 근사다항식을 얻은 방법론에 대해서 구체적으로 서술하시오. (단, 근사다항식의 차수는 31차 이하)

(2) 주어진 랜덤한 실수값 $x\in[-4,4]$에 대해서, 다음의 과정을 구현하시오.
- **실수값 x를 동형암호화**
- **암호화된 x에 대해서 $p(x)$의 암호문을 계산**
- **계산된 암호문을 복호화**

(3) (2)번 문제를 확장하여 주어진 다수의 랜덤한 실수값 $x_i\in[-4,4]$ ($i=1,2,...,128$)에 대해서 다음의 과정을 구현하시오.
- **실수값 $x_i\in[-4,4]$를 동형암호화**
- **암호화된 $x_i\in[-4,4]$에 대해서 $p(x_i)$의 암호문을 계산**
- **계산된 암호문을 복호화**

■ 구현 및 테스트 환경 상세

- 해당 문제의 구현물은 링크를 통해서 다운받은 Ubuntu.vdi를 가상머신 (e.g., Virtual Box 6.0)에서 구동하여 “Home/Documents/Problem2” 와 “Home/Documents/Problem3” 폴더 안의 Problem2.cpp와 Problem3.cpp 파일을 수정해서 구현한다(가상머신 설정: 메모리 2048MB, CPU 2개).

- 파일을 수정한 뒤에 터미널을 통해서 $cmake CMakeLists.txt 와 $ make 명령어를 통해서 새롭게 run 이라는 실행파일을 얻을 수 있다. 해당 실행 파일은 $ ./run 이라는 명령어를 통해서 실행가능하다.

- 해당 문제의 구현물에 대한 성능 측정은 아래의 성능을 가지는 PC에서 측정한다.
  - ✓ MacBook Pro (Retina, 13-inch, Late 2013), 2.6 GHz Intel Core i5, 8GB 1600 MHz DDR3

- 해당 문제에서 정확도는 평문 상태에서 계산한 $f(x)$와의 비교

- 결과물은 다음을 포함한다.
  - ✓ 결과 다항식 및 문서 (근사 방법 상세) (1번)
  - ✓ c++로 작성된 코드 (2, 3번)
  - ✓ 문서 (구현 기법 상세) (2, 3번)

- 평가방법
  - ✓ 암호화되지 않은 상태에서의 계산과의 비교 (절대평가)
  - ✓ 문서화 (절대평가)
  - ✓ 벤치마크 결과 (상대평가)

- 주의사항
  - ✓ 벤치마크 결과는 상대평가이며 연산속도가 빠른 순서대로 높은 점수를 받는다. (2, 3번)
  - ✓ 암호화되지 않은 상태에서 계산한 $f(x)$와 0.001 보다 큰 차이가 나는 경우에는 0점 처리된다. (2, 3번)
  - ✓ 암호화되지 않은 상태에서 계산한 $f(x)$와 0.001 이하의 차이가 나는 경우에는 정확도는 더 이상 채점 기준에 들어가지 않는다. (2, 3번)
  - ✓ (1), (2), (3)번 문제 중에 일부만 풀어서 제출해도 부분점수로 인정한다.

1) "Homomorphic Encryption for Arithmetic of Approximate Numbers"