Python 기반 동형암호 라이브러리 비교 및 분석 연구

서지완[1], 정헌희[1], 이동주[1], 백윤흥[1]
[1]서울대학교 전기정보공학부, 서울대학교 반도체 공동연구소
{jwseo, hhjung, djlee}@sor.snu.ac.kr, ypaek@snu.ac.kr

A Comparative Study of Python-Based Homomorphic Encryption Libraries

Jiwan Seo[1], Heonhui Jung[1], Dongju Lee[1], Yunheung Paek[1]
[1]Dept. of Electrical and Computer Engineering and Inter-University Semiconductor Research Center(ISRC), Seoul National University

요 약

동형암호는 암호화된 상태로 연산이 가능하여 개인정보 보호에 효과적인 기술로 주목받고 있다. 본 연구는 Python 환경에서 활용 가능한 TenSEAL, Pyfhel, openfhe-python을 소개하고, Inner Product, Matrix Multiplication 그리고 Convolution 연산을 중심으로 특징과 성능을 비교하였다. 이를 통해 연구자와 개발자가 목적에 맞는 라이브러리를 선택할 수 있는 가이드를 제공한다.

1. 서론

정보 기술의 급격한 발전으로 우리는 대규모 데이터를 수집, 분석, 활용하는 데이터 경제 시대에 살고 있다. 이러한 데이터는 개인 맞춤형 서비스부터 인공지능(AI) 모델 학습에 이르기까지 광범위하게 활용되지만, 이와 동시에 개인정보 유출 및 오용의 위험성 또한 심각한 사회적 문제로 대두되고 있다. 특히 클라우드 컴퓨팅 환경에서 사용자의 민감 데이터를 처리할 때, 데이터는 전송 계층에서 암호화를 통해 보호되지만, 실제 연산을 위해서는 서버에서 복호화되어야 한다. 이 과정에서 데이터가 평문 상태로 메모리에 노출되면서 악의적인 내부 관리자나 외부 시스템 공격자에 의한 정보 유출 위협이 상존하게 된다.

동형암호(Homomorphic Encryption, HE)는 데이터를 암호화된 상태로 연산할 수 있다는 특성을 가지고 있으므로 이러한 문제를 해결할 수 있다. 즉, 사용자는 데이터를 암호화하여 서버에 전송하고, 서버는 암호문 상태에서 연산을 수행한 후 그 결과를 사용자에게 반환한다. 사용자는 이를 복호화하여 최종 결과를 얻을 수 있으므로, 연산 과정에서 사용자의 데이터가 외부에 유출되지 않는다. 이러한 특성 덕분에 동형암호는 프라이버시 보존이 필수적인 의료, 금융, 그리고 서비스형 AI(AI-as-a-Service)와 같은 분야에서 높은 잠재력을 지닌 기술로 평가받고 있다.

그러나 동형암호를 실제 활용하기 위해서는 HE 라이브러리를 사용해야 하며, 대부분 C/C++ 기반으로 작성되어 있다. 이는 프로그래밍 경험이 부족한 초보자나 Python 환경에 익숙한 연구자들에게는 높은 접근 장벽으로서 동작한다.

Python은 인터프리터 언어로서 상대적으로 학습이 용이하고, 특히 최근 딥러닝 연구와 AI 서비스 개발의 표준 언어로 자리 잡았다. 이는 Python이 간결한 문법과 방대한 라이브러리 생태계를 바탕으로 TensorFlow, PyTorch, 등 주요 AI 프레임워크에서 중심적으로 사용되고 있기 때문이다. 뿐만 아니라, Python은 교육용으로도 널리 보급되어 있어 초보자와 연구자 모두에게 접근성이 높다. 이러한 환경적 특성은 동형암호 기술 역시 Python 환경에서의 구현과 실험 가능성을 요구하게 되었다.

이에 본 연구에서는 Python에서 사용할 수 있는 대표적인 동형암호 라이브러리들을 소개하고, 각 라이브러리의 특징과 성능을 비교 및 분석함으로써 초보자 및 연구자들에게 동형암호 사용의 가이드를 제공하고자 한다.

2. 배경

2.1 동형암호

동형암호는 평문 연산과 암호문 연산의 결과가 일치한다는 동형성(Homomorphism) 속성을 기반으로 하

며, 대표적으로 덧셈 동형성을 갖는 Paillier 암호, 곱셈 동형성을 갖는 RSA, 그리고 완전 동형성을 지원하 완전동형암호(Fully Homomorphic Encryption, FHE) 체계가 존재한다.. 그 중에서 FHE는 덧셈과 곱셈을 모두 지원한다.

동형암호 암호문은 메시지 m, 비밀키 s, 노이즈 e, 랜덤 a를 통해 생성할 수 있다.

$$c = (a \cdot s + m + e, a)$$

암호문 요소들은 다항식 구조로 생성되고, 다항식 차수로는 N을 사용한다. 더 큰 차수의 다항식을 사용할수록 보안성이 높아진다. 그러나 동시에 연산량이 증가하므로 속도가 느려진다는 Trade-off가 존재한다.[1] 동형암호는 암호문 생성에서 난수 노이즈 e를 사용하므로, 암호문 연산을 반복하는 경우 노이즈가 누적된다. 누적된 노이즈가 일정 크기를 넘어서게 되면 정확한 복호화에 어려움이 있다는 한계가 존재한다.

2.2 AI 모델과 동형암호 재부팅

딥러닝 모델은 다수의 계층(layer)으로 구성되어 있으며, 이는 추론 과정에서 대규모의 덧셈 및 곱셈 연산을 수반한다. 예를 들어, 널리 사용되는 ResNet이나 LeNet-5와 같은 모델은 CIFAR-10 데이터셋을 기준으로 수십만 회 이상의 산술 연산을 필요로 한다

동형암호는 연산이 중첩될수록 암호문에 포함된 노이즈(noise)가 점차 증가하는 특성을 가진다. 이 노이즈가 특정 임계치를 초과하면 데이터의 복호화가 불가능해지므로, 연산 횟수가 많은 딥러닝 모델에 동형암호를 적용하기 위해서는 효과적인 노이즈 관리가 필수적이다. 재부팅(Bootstrapping)[2]은 이러한 문제를 해결하기 위한 핵심 기법으로, 암호문을 복호화하지 않은 상태에서 노이즈를 초기화하여 추가적인 연산을 가능하게 하는 과정이다.

상대적으로 연산 깊이(computational depth)가 얕은 모델의 경우, 전체 연산 과정에서 누적되는 노이즈가 임계치를 넘지 않도록 파라미터를 설계하여 부트스트래핑의 사용을 최소화하거나 배제할 수 있다.

그러나 오늘날 활용되는 대부분의 CNN(Convolutional Neural Network) 모델과 같이 깊은 구조를 가진 모델에서는 부트스트래핑의 주기적인 수행이 선택이 아닌 필수적인 요소로 간주된다.

3. 기존 Python 라이브러리 소개

현재 연구 및 개발에 활용되는 Python 동형암호 라이브러리는 대부분 성능 최적화를 위해 C++로 구현된 핵심 엔진을 Python에서 호출할 수 있도록 바인딩(binding)또는 래핑(wrapping)한 구조를 가진다. 본 장에서는 대표적인 라이브러리인 TenSEAL[3], pyFHEL[4] 그리고 OpenFHE-Python[5]의 특징을 소개한다.

3.1 TenSEAL

TenSEAL은 암호화된 상태에서도 텐서(Tensor) 연산이 가능하도록 설계된 Python 기반 동형암호(FHE) 라이브러리로, 주로 머신러닝 추론(inference) 단계의 보호를 목적으로 한다. 이 라이브러리는 Microsoft SEAL의 CKKS 스킴을 기반으로 하며, 실수형 벡터 또는 행렬에 대한 근사 연산을 지원한다. TenSEAL의 가장 큰 특징은 암호학적 복잡성을 사용자로부터 감추고, PyTorch 또는 TensorFlow와 유사한 방식으로 텐서를 암호화하여 연산할 수 있도록 추상화된 API를 제공한다는 점이다.

사용자는 암호화 키나 파라미터 세팅에 대한 깊은 이해 없이도 CKKSVector 객체를 활용하여 암호화된 상태의 벡터 덧셈, 곱셈, 내적(dot product), 행렬곱(matrix multiplication), 다항식 평가(polynomial evaluation) 등을 수행할 수 있다. 이러한 추상화는 동형암호의 실용화를 위한 높은 수준의 인터페이스를 제공하며, 딥러닝 모델의 암호화 추론에 적합한 환경을 조성한다.

다만, TenSEAL은 Bootstrapping을 지원하지 않는다. 따라서 연산 깊이가 깊어질수록 암호문 내 노이즈가 누적되어 복호화 실패 가능성이 증가하며, 복잡한 다층 신경망 계층(예: 다중 Convolution layer)에는 적용이 제한될 수 있다.

3.2 Pyfhel

Pyfhel은 Microsoft SEAL 라이브러리를 Python 환경에서 직접적으로 사용할 수 있도록 설계된 래퍼(wrapper) 라이브러리이다. 내부적으로는 Cython을 사용하여 C++ 클래스와 함수들을 Python 인터페이스에 매핑하며, 사용자는 SEAL의 다양한 기능을 Python 객체 기반으로 호출할 수 있다. Pyfhel은 CKKS 및 BFV 스킴을 모두 지원하며, 사용자는 컨텍스트 파라미터, 스케일, 암호문 레벨, 노이즈 예산 등 핵심 암호 파라미터를 세밀하게 설정하고 제어할 수 있다.

그러나 TenSEAL과 마찬가지로, Pyfhel 또한 Bootstrapping 기능을 지원하지 않으며, 이는 연산 깊이(multiplicative depth) 및 곱셈 횟수에 따라 암호문이 복호화 불가능해질 수 있음을 의미한다. 따라서 Pyfhel 역시 복잡한 모델, 특히 깊은 계층 구조를 요구하는 신경망의 동형암호 연산에는 한계가 존재한다.

3.3 OpenFHE-Python

OpenFHE-Python은 C++로 구현된 고성능 동형암호 프레임워크인 OpenFHE의 기능을 Python 환경에서 활용할 수 있도록 구성된 래퍼 라이브러리이다. 본 라이브러리는 Python 개발자에게도 FHE 기술의 실험과 적용을 가능하게 하는 사용자 친화적인 인터페이스를 제공하며, 암호 파라미터 설정부터 암호화 연산까지 다양한 기능을 Python 객체 단위로 사용할 수 있도록 구성되어 있다.

OpenFHE-Python은 대표적으로 BFV, BGV, CKKS 스킴을 지원하며, 사용자는 각 스킴의 보안 수준, 다항식 차수, 스케일링 인자 등 핵심 암호 파라미터를 명시적으로 설정할 수 있다.

연산 측면에서는 암호문 간 덧셈, 곱셈, 스칼라 곱, 슬롯 회전등의 기능을 암호화된 상태에서 수행할 수 있으며, 암호화/복호화, 키 생성, 컨텍스트 직렬화/역직렬화 등의 고급 기능 또한 Python 코드로 직접 실행 가능하다.

OpenFHE-Python은 부트스트래핑 기능을 공식적으로 지원하며, 암호문 내 노이즈를 줄이고 연산 가능 깊이를 확장하는 데 활용할 수 있다. 이로 인해, 다른 Python 기반 FHE 라이브러리들이 가지는 연산 깊이 제한 문제를 효과적으로 극복할 수 있으며, 복잡한 회로나 다층 신경망 모델의 암호화 추론 환경에서도 보다 안정적인 실행이 가능하다.

항목	TenSEAL	Pyfhel	OpenFHE-Python
코드 기반	C++ (Microsoft SEAL) + Python 래퍼	C++ (SEAL) + Cython	C++ (OpenFHE) + pybind11
지원 Scheme	CKKS	CKKS, BFV	BFV, BGV, CKKS, TFHE
부트스트래핑 지원	X	X	O
내부 파라미터 제어 수준	낮음 (자동 설정 중심)	높음 (context 직접 설정)	매우 높음 (모든 파라미터 명시 가능)

표 1 라이브러리 별 특징 비교

4. 라이브러리 성능 평가 방법

Python 기반 동형암호 라이브러리에 대해 내적 연산(dot product), 행렬-벡터 곱(matrix-vector multiplication), 그리고 컨볼루션 연산(convolution)의 성능을 비교하고자 한다. 이들 연산은 신경망 추론에서 핵심적으로 사용되는 연산으로, 각 라이브러리의 실질적인 적용 가능성을 판단하는 데 있어 중요한 지표가 된다.

4.1 평가항목 정의

평가는 다음의 세 가지 연산으로 수행한다.

내적 (Dot Product)

1차원 벡터 간 연산, 기본적인 연산 단위로서 최적화를 측정한다.

행렬-벡터 곱 (Matrix-vector Multiplication)

2차원 텐서 연산으로, 암호문 패킹/슬롯 활용 및 회전 연산의 최적화 수준을 반영한다.

컨볼루션 (Convolution)

다차원 텐서 연산을 기반으로 하며, 라이브러리의 회전 키 활용, 덧셈/곱셈 조합 최적화 능력을 평가한다.

4.2 성능 측정 기준

각 연산에 대해 다음의 성능 지표를 측정한다:

실행 시간 (Latency)

암호문 입력에 대해 연산이 수행되는 총 소요 시간 (ms 단위 측정)

오차율 (Error Rate)

복호화 결과와 평문 연산 결과 간의 평균 제곱 오차(MSE)를 기반으로 측정

라이브러리별 정량적 비교를 위해, 각 연산에 대해 아래와 같은 평가 점수를 산출한다. 여기서 w는 연산별 가중치로, 시간 복잡도를 기준으로 설정하였으며, 내적: 1.0, 행렬-벡터 곱: 300, 컨볼루션: 400으로 설정하였다.

$$score = \frac{w}{Latency}(1 - ErrorRate)$$

5. 실험 결과

성능을 확인하기 위해 AMD EPYC 7262 와 DRAM 64GB를 탑재한 서버에서 성능을 측정했다.

실험 결과, OpenFHE가 전반적으로 가장 우수한 성능을 보였다. 합성곱 및 행렬 곱 연산에서 가장 낮은 지연 시간을 기록하였으며, 평균제곱오차

(MSE)는 약 10^{-14}수준으로 측정되어 정확도 측면에서도 탁월하였다.

이름	값
N	16384
Poly Modulus Degree	40
Dot Product Input	1024
Matrix-Vec Mult Input	128x128
Convolution Input	14x14
Convolution Kernel Size	5x5

표 2 실험 Parameter

Libary	내적	행렬-벡터 곱	컨볼루션	총점
TenSEAL	1.1	3.97	8.11	13.2
Pyfhel	0.61	1.53	2.24	4.38
OpenFHE-Python	1.13	3.86	8.54	**13.5**

표 3 연산 별 평가 점수 및 총합

Pyfhel은 연산 속도 가장 느린 결과를 보였으나, 오차는 OpenFHE와 동일하게 10^{-14} 수준을 유지하였다. 따라서 성능 면에서는 한계가 있으나, 계산 결과의 신뢰도 자체는 매우 높음을 확인할 수 있었다.

반면, TenSEAL은 실행 속도가 OpenFHE과 유사하였으나 오차가 10^{-9} 수준으로 상대적으로 크게 나타났다. 다만, PyTorch와의 통합 편의성과 직관적인 API 제공 측면에서는 장점이 확인되었다.

6. 토론 및 고찰

실험 결과는 동형암호 라이브러리 선택 시 연구 목적과 활용 환경에 따라 명확한 차별점을 고려해야 함을 보여준다. OpenFHE는 Bootstrapping을 지원하여 연산 깊이를 요구하는 복잡한 구조에서도 안정적으로 동작하므로, 대규모 데이터셋이나 서비스 환경에 적합하다. TenSEAL은 Bootstrapping을 지원하지 않아 깊은 연산에는 한계가 있으나 PyTorch와의 친화성과 사용 편의성 덕분에 얕은 수준의 DNN 구현이나 프로토타입 개발에 유리하다. 마지막으로, Pyfhel은 연산 속도는 느리지만 정확도가 높아 학습 목적이나 소규모 검증 환경에서 여전히 활용 가능하다.

7. 결론

본 연구에서는 Python 기반 동형암호 라이브러리인 TenSEAL, Pyfhel, OpenFHE-Python을 비교 분석하였다. 각 라이브러리는 설계 목적과 지원 기능, 연산 성능에서 뚜렷한 차이를 보였으며, 활용자의 수준과 적용 시나리오에 따라 적합성이 달라진다. 초보자나 파라미터 설정에 익숙하지 않은 사용자에게는 TenSEAL이, 정확도 중심의 실험에는 Pyfhel이, 복잡한 연산이나 깊은 신경망 모델 구현에는 OpenFHE-Python이 적절하다.

결론적으로, 본 비교 결과는 각 라이브러리의 특징과 한계를 명확히 이해하고, 연구 목적이나 개발 환경에 맞는 FHE 도구를 선택하는 데 실질적인 기준을 제공할 수 있을 것으로 기대된다.

8. ACKNOWLEDGEMENT

이 논문은 연구 수행에 있어 2025년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 (RS-2023-00277326) 받았으며 2025년도 BK21 FOUR 정보기술 미래인재 교육연구단, 반도체 공동연구소 지원의 결과물이다. 또한, 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드린다.

참고문헌

[1] Lyubashevsky, Vadim, Chris Peikert, and Oded Regev. "A toolkit for ring-LWE cryptography." Annual international conference on the theory and applications of cryptographic techniques. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[2] Cheon, Jung Hee, et al. "Bootstrapping for approximate homomorphic encryption." Annual International Conference on the Theory and Applications of Cryptographic Techniques. Cham: Springer International Publishing, 2018.

[3] Benaissa, Ayoub, et al. "Tenseal: A library for encrypted tensor operations using homomorphic encryption." arXiv preprint arXiv:2104.03152 (2021).

[4] Ibarrondo, Alberto, and Alexander Viand. "Pyfhel: Python for homomorphic encryption libraries." Proceedings of the 9th on workshop on encrypted computing & applied homomorphic cryptography. 2021.

[5] OpenFHE-Python: Python wrapper for the OpenFHE library, Version 1.4.0.1, 2025. [Online]. https://github.com/openfheorg/openfhe-python