부분적 동형암호 HW 가속기 설계에 관한 연구

남기빈*, 장지원*, 조명현*, 방인영*, 백윤흥*
*서울대학교 전기정보공학부, 반도체공동연구소
{kvnam, jwchang, mhcho, iybang}@sor.snu.ac.kr, ypaek@snu.ac.kr

Partially Homomorphic Encryption HW accelerator

Kevin Nam*, Jiwon Chang*, Myunghyun Cho*, Inyoung Bang*,
and Yunheung Paek*
*Dept. of Electrical and Computer Engineering and Inter-University
Semiconductor Research Center(ISRC), Seoul National University

요 약

최근 동형암호에 대한 관심이 높아진 가운데, 이를 활용한 Cloud Computing 서비스를 구축하기 위한 시도가 이어지고 있다. 기존 동형암호 HW에 대한 연구는 수학적 기능 구현 자체에 중점을 두고 있다. 본 논문에서는 동형암호 CNN inference 모델 설계 과정에서 HW 구현 한계점과 bottleneck들을 수학적 기법이 아닌 HW 특징을 이용해서 극복하는 과정을 서술하였다.

1. 서론

동형암호는 암호화 전 연산과 암호화한 상태에서의 연산 결과가 복호화할 시 같다는 특징을 지니고 있다. 사용자는 암호화한 데이터를 서버에 전송하여 처리하여 돌려받아, 본인이 복호화하여 값을 확인할 수 있어, 차세대 Cloud Computing Interface의 주요 feature로 주목받고 있다. 이에 최근 FPGA와 GPU를 이용한 동형암호 가속기를 구현하고자 하는 시도가 이어지고 있으나, 수학적 기능 구현에 중점을 두어 구현되어 과도한 resource 사용으로 인한 timing 문제, 지나친 bottleneck의 한계로 성능 제한을 받고 있다.

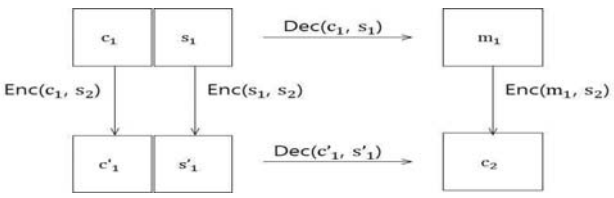
본 연구는 수학적 분석 없이 주어진 동형암호 체계를 부분적으로만 활용하여 데이터 이동, resource, configurability와 같은 HW 특징에 중점을 두고 원하는 기능에 맞게 더 좋은 성능의 HW 체계를 구현하였다. 본 논문은 그 과정에서 이루어지는 선택과정들에 관해 서술하여 가이드라인을 제공하고자 한다.

2. 동형암호의 분류

동형암호 개념이 등장한 것은 1997년이지만[1] 실제 구현 가능한 동형암호는 2009년 Gentry의 BFV scheme을 통해 소개되었다[2]. 본 장에서는 FHE와 그 이전 형태들을 비교하고, 대표적인 최신 scheme들을 간단히 비교하여 설명할 것이다.

2.1 Somewhat/Fully Homomorphic Encryption

곱셈이 이루어지면, 암호문에 noise bits가 증가하게 된다. 이에 허용 최대 수를 벗어나면, overflow가 발생하여 가용하지 않은 상태가 된다. 이러한 이유로 Somewhat이라는 단어가 붙어 SHE라고 부른다. SHE의 한계를 극복한 FHE는 noise bits를 줄이는 bootstrap이라는 방법과 함께 등장했다. (그림 1)에 나타나듯, m_1을 암호화한 암호문 c_1과 s_1을 새로운 암호키 s_2을 통해 각각 암호화한 값들을 c'_1과 s'_1이라 하자. c'_1을 s'_1을 통해 복호화한 문장은, m_1을s_2를 통해 암호화한 새로운 암호문 c_2가 된다. 즉, 암호문을 새로운 암호키를 이용하여 새로운 암호문으로 전환하면서, 아직 어느 곱셈도 하지 않은 상태의 새로운 암호문이 되는 것이다.



(그림 1) Bootstrap procedure

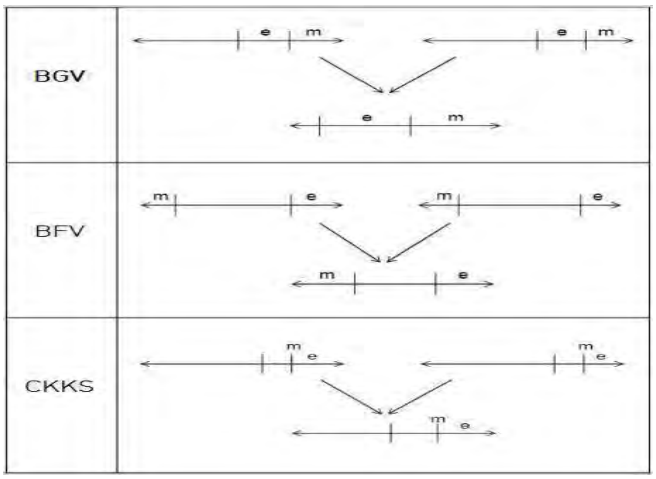
2.2 BGV, BFV, CKKS

같은 원리를 통해 다양한 scheme들이 등장한 가운데, 가장 자주 언급되고 연구되는 세 가지 scheme들이 있다. 각각 BGV[3], BFV[4], CKKS[5]라 불리며, 이들에 대해서 필요한 부분들을 간단히 정리해서 서술하면 다음과 같다.

BGV scheme은 $ct = Enc_{sk}(m) = (a, as + te + m)$의 꼴을 통해 암호화가 이루어진다. (그림 2)에 나타나듯, 문장의 상위비트에 noise를 추가하여 연산이 이루어진다. 곱셈이 이루어질 때마다 noise가 곱해지며 점점 증가하기 때문에, 이를 줄이는 방법이 필요하였기에 modulus switching이라는 기법을 도입하였으나, modulus가 줄어든다는 단점을 지니고 있다.

BFV scheme의 경우 LSB에 noise를, MSB에 문장을 넣어 암호화를 진행한다. $ct = Enc_{st}(m) = (a, as + e + \frac{q}{t}m)$의 꼴을 따라 진행되는데 이 경우 scale-invariant하여 noise를 줄이는 과정 진행시 modulus가 줄지 않는다는 장점을 지니고 있었다. 그럼에도 noise자체는 점점 증가하여 최대 비트 수를 넘길 수 없기에, FHE를 이루기 위해서는 재부팅이 불가피하다.

CKKS는 approximate 형태로 실수를 표현가능한 scheme이다. Noise를 추가로 첨가하는 이전 scheme들과 다르게, 실수 중 오차 허용 범위를 벗어난 값들을 noise로 추가하는 형태로 연산을 진행한다. CKKS의 경우도 마찬가지로 noise와 문장의 길이가 곱할수록 커지는데, 이 오차 허용 범위를 동일하게 하기 위해 rescaling을 활용한다.



(그림 2) BGV, BFV, CKKS

이러한 scheme들을 통해 FHE 구현을 실현할 수 있는 library들이 존재한다. 이후 서술될 내용은 MS의 SEAL[6]을 활용하여 실험하였다.

3. 동형암호 HW 체계 한계점

이러한 동형암호 체계를 실용화하기 위한 단계로써 HW 가속기 설계에 관한 연구가 최근 많이 진행되고 있다. BFV scheme을 활용한 연구[7]와 CKKS를 활용한 HEAX[8]가 full module을 구현한 사례이며, 이 외에도 맞춤형 SHE 체계를 구현한 연구들이 진행됐다. 이러한 연구들은 최고 성능의 FPGA와 GPU를 활용하였음에도 불완전하다는 한계점을 지니고 있는데, 그러한 이유에 관해서 서술하겠다.

3.1 Bootstrapping(rescaling) Complexity

위에서 언급된 재부팅 과정을 다시 살펴보면, 원래 크기의 암호문으로 변경하는 기능을 이루었지만, 이를 해독하기 위해서는 두 번의 해독과정이 필요하다. 즉 재부팅이 이루어질수록 해독과정이 복잡해진다. 이 과정은 FHE 연산 과정에 있어 매우 큰 bottleneck이 되고 있다. 이를 해소하기 위해 재부팅이 필요한 상태를 예측하여 buffer에 미리 추가, 시간을 단축하고자 하는 시도 등이 있었지만, 아직 충분히 해소되지 않았다.

이에 SHE지만 필요한 기능은 수행할 수 있는, 즉 구현하고자 하는 기능이 요구하는 multiplicative depth를 만족하는 SHE 체계를 구현하기 위한 시도들이 있었으며, 이들 시도는 re-scalability를 포기한 시도들이라 정리할 수 있다.

3.2 Parameter Selection

동형암호는 다른 암호체계와 마찬가지로 기반이 되는 parameter들을 설정해야 한다. HW 설계 시 이러한 modulus 기저들의 크기를 고려하여 i/o 비트 수를 정해야 한다. 문제는 HEAX와 같이 높은 차수의 다항식을 다룰 수 있도록 설계할 경우 과한 복잡도로 timing 문제가 발생할 수 있으며, 일부 resource는 거의 쓰지 않게 될 수 있다. 일부 정보는 사용자만 알고 있으므로 노출 위험이 적지만, 많은 사용자가 서로 다른 설정 하에 활용하기 위해서 정보를 저장한다면 error 발생 빈도가 높아질 수 있어 이를 고려하여 설계하여야 한다.

3.3 연산 종류의 한계

동형암호의 장점은 암호화한 상태에서 연산이 이

루어진 후 복호화하더라도 같은 결과를 만들어 준다는 점이다. 덧셈과 곱셈을 할 수 있지만, 다른 연산이 불가능하다. 지수 함수같이 다항식으로 근사가 가능한 방법이 있지만, comparator와 같은 비교 연산은 암호화 상태에서 불가능하다.

직접 비교 연산 없이 같은 기능을 할 방법들에 관한 연구 시도가 있었지만, 단순 부호 판단만 하더라도 zero 값에 대한 암호문이 알려져야 하는 등 조건들이 필요하며, 특정 case들에 대한 암호문을 공개하는 것은 암호체계의 confidelity를 해치기에 위험한 접근으로 여겨진다.

4. Homomorphic CNN HW 구현

2, 3장서 서술한 내용을 바탕으로 HW 구현 솔루션을 설계하는데 고려해야 할 점들이 많다. 본 장은 CNN의 inference 과정을 동형암호 체계로 구현하기 위한 일련의 선택과정들을 서술하고, 이들에 대한 비교 자료를 제시할 것이다. MNIST dataset을 활용하였고, C++과 MS의 SEAL을 통해 CKKS scheme을 구현하였다. SW 구현이지만 HW 설계 시뮬레이션을 이루기 위해 각 component 별 비트 수를 고정하였고, 이를 초과할 경우 error가 발생하도록 만들고 진행하였다.

4.1 SHE vs FHE

SHE와 FHE에 대한 선택은 HW 구현에 있어 큰 trade-off를 일으킨다. FHE는 제한 없는 곱셈 횟수를 이루어 낼 수 있지만 많은 양의 resource가 필요하여 값비싼 솔루션인 만큼 최적화를 목적으로 하는 HW 설계 면에서 한계를 지니고 있다. HEAX의 경우 Stratix 10 보드의 90%가 넘는 resource를 활용하여 연산자를 구현했음에도 bootstrap이 많은 연산을 진행시 speedup이 4.8에 불과한 결과를 보여주었다. 이에 FPGA를 활용하여 parameter를 조절해서 재부팅 없이 사용자 필요에 맞는, 기능을 수행할 수 있도록 설계하여 FHE와 같은 형태를 구현하는 방법이 성능 면에서 효율적일 것이라 판단된다.

4.2 Modulus Switch / Relinearization

곱이 반복되며 noise 증가 폭이 늘어나는데, 이를 줄이기 위한 기법들 역시 HW 구현 시 충분히 포함할 수 있는 기능들이다. <표 1>는 CNN inference과정을 동형암호 체계에서 구현한 시뮬레이션 코드 결과이다. 곱이 지속할수록, 가용 비트 수가 줄어드는데, relinearization을 사용할 경우 가용 비트 수 감소량이 줄어들었다. 한편, 시간 지연이 발생하는데, 이 둘의 trade-off는 병렬화, threading, buffer의 활용 등 다른 설계 특징들에 의해 상호 관계가 많이 변하기 때문에, 직접 실험을 통해 최적의 case를 찾아내는 것이 바람직하다고 판단된다.

<표 1> CNN inference 과정 중 data

No# mult.	no relin.	with relin.
0	338 bits left	338 bits left
1	283 bits left	283 bits left
2	224 bits left	245 bits left
3	194 bits left	211 bits left
total time	22sec/img	48sec/img

4.3 사용자 PC와 서버 간 인터페이스

암호화한 상태에서 불가능한 연산들을 다른 방법으로 구현할 수도 있으나, 사용자 PC로 되돌려 복호화한 후 연산을 진행하는 것 역시 방법이 될 수 있다. 데이터를 전송하는 과정이 오랜 시간을 요구하지만, 필요한 기능에 따라 후자가 더 좋은 성능을 나타내기도 한다. 즉, 모든 과정을 암호화한 상태로 진행할 필요가 없다는 것이다.

CNN의 경우 비교 연산은 ReLu 활성화 함수와 마지막 label 선택할 때 softmax값들 중 가장 큰 값을 고르기 위해 활용한다. 따라서 처음 사용자 PC에서 서버, 마지막 서버에서 PC로의 데이터 전송 외 추가 전송이 발생하지 않아, 추가 지연이 필요하지 않다. 따라서 비교 연산을 암호화 상태에서 구현할 필요 없이 PC로 받아 복호화하여 연산할 수 있다.

활성화 함수의 경우, 다른 함수로 대체하여 진행할 수 있다. 시간 지연을 고려한 시뮬레이션 결과 accuracy는 차이가 없었기에, 따로 기능을 구현하지 않는 것으로 선택하였다. 그 결과는 아래 <표 2>를 통해 확인할 수 있다.

<표 2> 활성화 함수에 따른 결과

	ReLu at PC	Linear Activation
time	36sec/img	22sec/img
accuracy	96%	96%

5. 채택한 HW 구현 방향 (CHCNN)

본 연구를 통해 구현한 Partially Homomorphic CNN(PHCNN)은 MS SEAL을 이용하여 CKKS 체계로 구현했으며 요약하면 <표 3>와 같다. 이전 연

구와 다르게 FHE 구현을 위해 과도한 재부팅과 rescale을 진행하지 않고, parameter를 필요에 따라 조정하였고, 이에 Modulus Switch와 Relinearization도 고려하지 않았다. 필요할 때마다 Normalize를 통해 data 범위를 정정하였고, 사용자 PC와 서버와의 통신을 최소화하기 위해 두 번(시작, 끝)만 data를 주고받을 수 있도록 모델을 구현하였다. 표에 나타나지 않은 값들은 중요하지 않거나 SEAL에서 자동으로 설정해주는 값들로, HW 구현할 시 값들을 확인해서 메모리에 사전 저장해주어야 한다.

<표 3> Parameters for CNN model

Parameter Name	Value
poly modulus degree	16384
plain modulus	5522259017729
Max bits	540
multiplicative depth	11
Normalization	Linearized

6. 결론

이전 연구들의 접근과 본 논문을 통해 구현한 체계와의 비교 분석은 아래 <표 4>와 같다.

<표 4> Performance Comparison

HW Approach	inference (sec/img)
Raw approach	1759
Rescaling CKKS	344
HCNN approach	121
PHCNN	22

실험 결과 수학적 접근에 중점을 둔 완전 구현보다, 필요에 따라 scale을 변경하여 제 기능을 수행할 수 있도록 SHE를 조절하는 것이 훨씬 좋은 성능을 보여주었다.

완전한 체계 구현을 위한 과한 설정은 지나친 resource 남용과 bottleneck으로 이어질 수 있으며, 잘못된 설정은 multi-user 인터페이스 구현에 제한사항을 유발한다. 현재까지 알려진 연구들은 불완전한 구현을 최적화 없이 HW로 구현한 것이 대부분이다. HW 특징을 고려했다는 하나만으로도 본 연구가 이전 결과들보다 좋은 성능을 나타냄을 충분히 설명해줄 수 있다. 물론 재부팅과 같은 동형암호 체계의 기능들에 대한 분석, 이에 따른 효율적인 가속기 구현이 이루어진다면 더 좋은 가속기 설계가 가능하리라 생각한다.

추후 본 논문보다 좋은 결과를 위해 위와 같은 연구가 이루어져야 할 것이다.

7. ACKNOWLEDGEMENT

이 논문은 2020년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구 (NRF-2017R1A2A1A17069478)이며, 2020년도 두뇌한국21플러스사업에 의하여 지원되었음. 또한, 2020년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2018-0-00230, (IoT 총괄/1세부) IoT 디바이스 자율 신뢰보장 기술 및 글로벌 표준기반 IoT 통합보안 오픈 플랫폼 기술개발 [TrusThingz 프로젝트]).

참고문헌

[1] R. Rivest, L. Adleman, and M. Dertouzos, “On data banks and privacy homomorphisms,” Foundations of secure computation, 1978.

[2] C. Gentry, “Fully homomorphic encryption using ideal lattices,” Proceedings of the 41st ACM Symposium on Theory of Computing 2009, pp. 169 - 178, 2009.

[3] Z. Brakerski, C. Gentry, V. Vaikuntanathan, “Fully Homomorphic Encryption without Bootstrapping”, Electronic Colloquium on Computational Complexity Report No.111 (2011)

[4] S. Halevi, Y. Polyakov, & V. Shoup, “An Improved RNS Variant of the BFV Homomorphic Encryption Scheme”, The Cryptographers′ Track at the RSA Conference 2019

[5] J. Cheon, A. Kim, M. Kim, Y. Song, “Homomorphic Encryption for Arithmetic of Approximate Numbers”, ASIACRYPT 2017

[6] N. Dowlin et al. “Manual for Using Homomorphic Encryption for Bioinformatics”, Microsoft Research, 2015

[7] S. Sinha Roy et al. “FPGA-Based High-Performance Parallel Architecture for Homomorphic Computing on Encrypted Data”, 26th International Symposium on high-Performance Coputer Architecture (HPCA, 2019)

[8]M. Sadegh Riazi et al. “HEAX: An Architecture for Computing on Encrypted Data”, 25th The International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS, 2020)