ASIACRYPT 2019

Numerical Method for Comparison on Homomorphically Encrypted Numbers

Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, Keewoo Lee
Department of Mathematical Sciences, Seoul National University

Introduction

Introduction

❖ HE(Homomorphic Encryption) 개요

- **암호화된 상태**에서 **연산**을 수행할 수 있도록 하는 기술
- 데이터 보안을 유지하면서도 연산을 가능하게 하여 개인정보 보호에 활용됨

❖ HE의 두 가지 방식

- **Bit-wise HE**: 각 **비트**를 **개별적**으로 암호화하여 **논리 연산** 수행
- **Word-wise HE**: 각 **숫자 전체**를 암호화하고 **산술 연산(덧셈, 곱셈)**을 수행
  - ex) 두 개의 4비트 정수 a=5, b=3

$$a = 5\,(0101)_2$$
$$b = 3\,(0011)_2$$

$$Enc(a) = Enc(0), Enc(1), Enc(0), Enc(1)$$
$$Enc(b) = Enc(0), Enc(0), Enc(1), Enc(1)$$

<Bit-wise HE>

$$Enc(a) = Enc(5)$$

$$Enc(b) = Enc(3)$$

<Word-wise HE>

Introduction

❖ 기존 연구의 한계

- 기존 연구에서는 **word-wise HE**에서 **non-polynomial 연산**을 수행하기 위해, **다항식 근사 방법** 사용.
- **기존 방법의 문제점**
  - $2^{-\alpha}$ **오차**를 보장하기 위해 $\Theta(2^{\alpha})$ **차수의 다항식**이 필요하며, 이는 $\Theta(2^{\alpha/2})$ 의 계산 복잡도를 요구함.
  - 다항식의 차수를 높여 오차를 줄일 수 있지만, **계산 복잡도가 지수적으로 증가하는 문제**가 있음.

❖ 연구 목표

- **비교 연산**을 보다 효율적으로 수행하는 새로운 방법 제안.
  - **반복 알고리즘(Iterative Algorithm)** 기반 연산 도입
  - 기존 방법 대비 계산량을 $\Theta(\alpha)$로 줄이면서도 높은 정확도를 유지

Proposed Method: Iterative Algorithms

❖ Iterative Algorithms 소개

- **Square root** 와 **Inverse** 연산을 기반으로 한 **반복 알고리즘**
- Polynomial 연산을 효율적으로 구성

❖ 핵심 아이디어

- **Composite function**을 활용하여 계산 복잡도 감소
  - Composite function: 다항식을 여러 개의 낮은 차수의 함수로 분해하여 계산
- 반복 알고리즘을 통해 min/max 및 비교 연산 근사

Proposed Algorithms

Approximate Min/Max Algorithms 1

❖ Approximate Min/Max Algorithms

- 암호화된 데이터에 대해 **최소값(Min)과 최대값(Max)을 근사적으로 계산**하는 알고리즘.

❖ Min/Max for Two Numbers

- 두 숫자 $a, b \in [0,1)$에 대해 최소값과 최대값을 근사적으로 계산
  - $\boldsymbol{Min(a,b;\ d)}$, $\boldsymbol{Max(a,b;\ d)}$

$$max(a,b) = \frac{a+b}{2} + \frac{\sqrt{(a-b)^2}}{2} \qquad min(a,b) = \frac{a+b}{2} - \frac{\sqrt{(a-b)^2}}{2}$$

Approximate Min/Max Algorithms 1 - Background Concepts

❖ 최대값(max)과 최소값(min)의 수학적 표현

- $max(a,b) = \frac{a+b}{2} + \frac{|a-b|}{2}$, $min(a,b) = \frac{a+b}{2} - \frac{|a-b|}{2}$

  ⇒ 동형 암호 환경에서는 비교 연산이나 조건문과 같은 **non-polynomial 연산을 직접 수행하기 어려움**

$$max(a,b) = \frac{a+b}{2} + \frac{\sqrt{(a-b)^2}}{2} \qquad min(a,b) = \frac{a+b}{2} - \frac{\sqrt{(a-b)^2}}{2}$$

- 제곱근과 같은 다항식 연산을 사용해 비교 연산을 근사적으로 수행
- 동형 암호화에서 지원하는 **기본 연산과 호환**되며, 복잡한 비교 연산을 간단한 산술 연산으로 대체할 수 있음

Approximate Min/Max Algorithms 1 - Background Concepts

❖ Input Scaling

- 입력값이 **[0, 1)** 와 같은 특정 범위에 있도록 **스케일링** 함.
  - 두 입력값 $\bar{a}, \bar{b} \in [0, 2^{\ell})$가 주어지면 다음과 같이 스케일링: $(a, b) \leftarrow (\frac{\bar{a}}{2^{\ell}}, \frac{\bar{b}}{2^{\ell}})$
- 알고리즘이 완료된 후에는 다시 원래의 스케일로 복원하기 위해 2^{ℓ}를 곱하여 원래 숫자 크기로 되돌림.

❖ 제곱근(Square Root) 근사 알고리즘

- Wilkes(1951)의 알고리즘을 사용하여 **실수 $x(0 \leq x \leq 1)$ 에 대해 반복 연산을 이용**하여 $\sqrt{x}$ **를 근사**
- 절댓값(Absolute Value) 계산
  - $|x| = \sqrt{x^2}$를 이용해 $-1 \leq x \leq 1$의 절댓값 계산 가능

Approximate Min/Max Algorithms 1 - Background Concepts

❖ 제곱근(Square Root) 근사 알고리즘

- $\boldsymbol{Sqrt(x;d)}$: 입력값 x, 반복 횟수 d

Algorithm 2 $\mathtt{Sqrt}(x;d)$

Input: $0 \leq x \leq 1$, $d \in \mathbb{N}$
Output: an approximate value of $\sqrt{x}$

1: $a_0 \leftarrow x$
2: $b_0 \leftarrow x - 1$
3: **for** $n \leftarrow 0$ **to** $d - 1$ **do**
4: $\quad a_{n+1} \leftarrow a_n \left(1 - \frac{b_n}{2}\right)$
5: $\quad b_{n+1} \leftarrow b_n^2 \left(\frac{b_n - 3}{4}\right)$
6: **end for**
7: **return** a_d

- $Sqrt(x;d)$ 의 오차는 $\left(\mathbf{1} - \frac{\boldsymbol{x}}{\mathbf{4}}\right)^{\mathbf{2d+1}}$로 표현되며, 반복 횟수를 증가시킬수록 오차는 지수적으로 감소.

Approximate Min/Max Algorithms 1

❖ Error Analysis and Precision

- **Theorem 1: 일반적인 경우**
  - **모든 $a, b \in [0, 1)$** 에 대해 적용
  - $d \geq 2\alpha - 3$ 일 경우 오차는 $2^{-\alpha}$ 이하
  - 반복 횟수가 정확도 α 에 대해 선형적으로 증가해야 함: $\boldsymbol{d = \Theta(\alpha)}$

- **Theorem 2: 제한된 영역에서의 경우**
  - $|a - b| \geq c$ $(0 < c < 1)$에 대해 적용 $\left(c = \left(\frac{1}{2} + \frac{2^{\ell}-1}{2^{\ell}}\right) / \left(\frac{1}{2} + \frac{2^{\ell}-2}{2^{\ell}}\right)\right)$
  - $d \geq \log \alpha + 2 \log\left(\frac{1}{c}\right) + 1$ 일 경우 오차는 $2^{-\alpha}$ 이하
  - 반복 횟수가 로그함수에 따라 증가하므로 $\boldsymbol{d = \Theta(\log \alpha)}$ 로 계산 복잡도를 낮출 수 있음

Approximate Comparison Algorithms 2

❖ Comparison Algorithms

- **암호화된 데이터에 대해 두 숫자 a와 b를 비교하는 문제를 다룸.**

- $\boldsymbol{comp(a,b) := \chi_{(0,\infty)}(a-b)}$ $\qquad$ $\boldsymbol{step\ function}\ \chi_{(0,\infty)}(x) := \begin{cases} 1 & if\ x > 0 \\ 0 & otherwise \end{cases}$

⇒ **불연속 함수**이기 때문에 동형 암호 환경에서 직접 계산하기 어려움

❖ Approximate of Step Function

- **시그모이드 함수**로 근사

  - $\boldsymbol{\sigma(x) = \frac{1}{1+e^{-x}}}$로 정의되며, 연속 함수이므로 동형 암호 환경에서 계산이 가능함

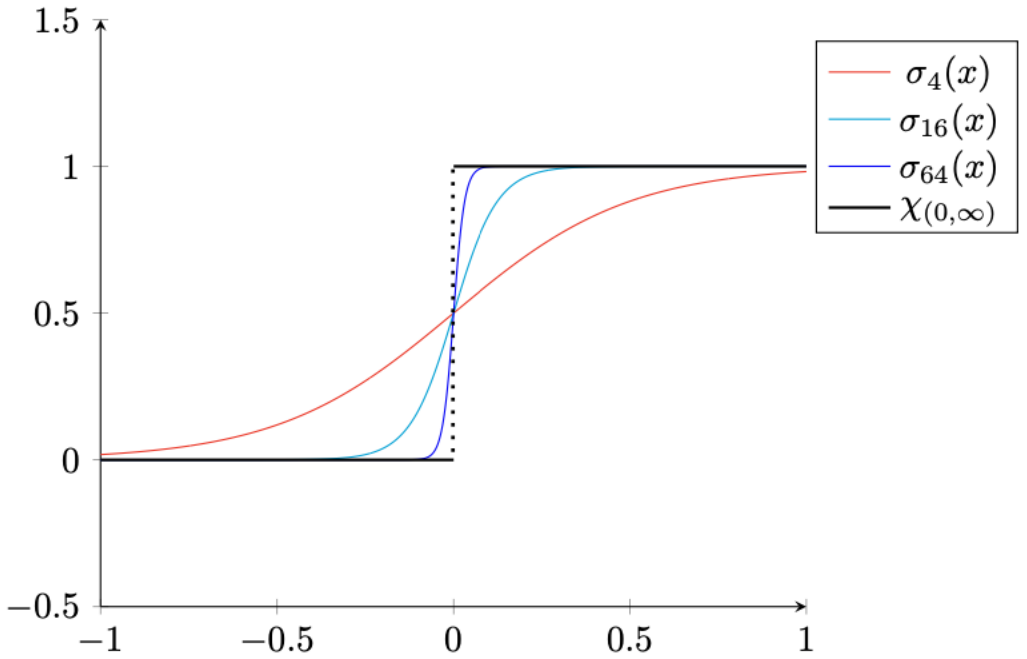
Approximate Comparison Algorithms 2

❖ Approximate of Step Function

- 시그모이드 함수를 스케일링하여 $\boldsymbol{\sigma_k(x)}:=\boldsymbol{\sigma(kx)}$로 정의하면 k가 커질수록 시그모이드 함수는 step function에 가까워짐

  - $\sigma_k(x) = \frac{1}{1+e^{-kx}}$

  - $\lim_{k\to\infty} || \chi_{(0,\infty)} - \sigma_k ||_{\infty,\mathbb{R}-[-\epsilon,\epsilon]} = 0$



[그림 1] 스케일된 시그모이드 함수로 근사된 $\chi_{(0,\infty)}$

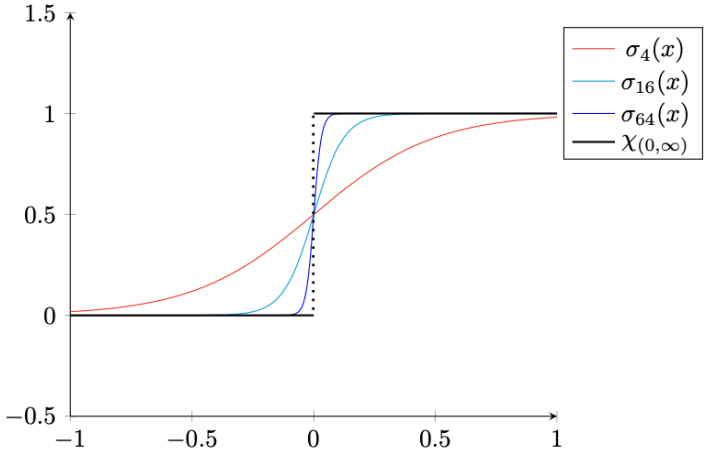
Approximate Comparison Algorithms 2

❖ Approximate of Sigmoid Function

- **지수 함수**는 동형 암호 환경에서 직접 계산하기 어려우므로 **로그를 취해 근사함**

- $$\boldsymbol{comp(a,b)} \approx \sigma_k(\log a - \log b) = \frac{e^{k \log a}}{e^{k \log a} + e^{k \log b}} = \frac{\boldsymbol{a^k}}{\boldsymbol{a^k + b^k}}$$

- $$\lim_{k\to\infty} \frac{\max(a,b)^k}{a^k+b^k} = 1, \ and \ \lim_{k\to\infty} \frac{min(a,b)^k}{a^k+b^k} = 0 \ \ if \ a \neq b$$



[그림 1] 스케일된 시그모이드 함수로 근사된 $\chi_{(0,\infty)}$

Approximate Comparison Algorithms 2

❖ Approximate Comparison Algorithms

- **Input** : $\boldsymbol{a}, \boldsymbol{b}$는 비교할 두 숫자로 $\left[\frac{1}{2}, \frac{3}{2}\right]$ 범위에 있음 (입력값 x에 대해 $x := \frac{1}{2} + \frac{x}{2^{\ell}}$ 로 스케일링한 결과의 범위)
  - $\boldsymbol{d}, \boldsymbol{d'}$: 역수 알고리즘(Inv) 반복 횟수 / $\boldsymbol{t}$: $comp$ 반복 횟수 / $\boldsymbol{m}$: 거듭제곱을 조절하는 값 ($m^t = k$을 만족하도록 선택)
- **Output** : $comp(a, b)$ 의 근사값을 반환 ($a > b$이면 1에 가깝고 $a < b$이면 0에 가까움)

Algorithm 5 $\texttt{Comp}(a, b; d, d', t, m)$

Input: distinct numbers $a, b \in \left[\frac{1}{2}, \frac{3}{2}\right)$, $d, d', t, m \in \mathbb{N}$
Output: an approximate value of $\text{comp}(a, b)$

1: $a_0 \leftarrow \frac{a}{2} \cdot \texttt{Inv}\left(\frac{a+b}{2}; d'\right)$
2: $b_0 \leftarrow 1 - a_0$
3: **for** $n \leftarrow 0$ **to** $t - 1$ **do**
4: $\quad inv \leftarrow \texttt{Inv}(a_n^m + b_n^m; d)$
5: $\quad a_{n+1} \leftarrow a_n^m \cdot inv$
6: $\quad b_{n+1} \leftarrow 1 - a_{n+1}$
7: **end for**
8: **return** a_t

$$\frac{a^k}{a^k + b^k}$$

Asymptotic Optimality of Proposed Methods

Asymptotic Optimality of Proposed Methods

❖ Minimax Polynomial Approximation

- 주어진 함수 $f(x)$를 다항식 $p_k(x)$로 근사할 때, **최대 오차 $\max|f(x) - p_k(x)|$를 최소화하는 다항식**을 찾는 것.
  - $p_k(x)$: 구간 $[-1, 1]$에서의 차수 k를 가지는 minimax 다항식

❖ 기존 접근 방식과의 비교

- 기존 Minimax 다항식 근사 방식보다 **본 연구에서 제안된 재귀적 접근 방식 기반 알고리즘의 계산 복잡도가 낮음**.
  - 기존의 다항식 근사법은 차수가 커질수록 계산량이 급격히 증가.
  - 제안된 방법은 재귀 알고리즘을 활용하여 같은 수준의 정확도를 유지하면서 계산 복잡도를 낮춤.

Min/max from Minimax Approximation

❖ Min/Max 함수의 표현 방법

- $max(a,b) = \frac{a+b}{2} + \frac{|a-b|}{2}$, $min(a,b) = \frac{a+b}{2} - \frac{|a-b|}{2}$

⇒ 절대값 함수 $|x|$를 **minimax 다항식으로 근사**

$$\lim_{k\to\infty} k \cdot || |x| - p_k||_{\infty,[-1,1]} = \beta$$ → 차수 k가 증가할수록 오차가 $O(1/k)$ 비율로 줄어듦

Jackson의 부등식을 기반으로 $|x|$를 Minimax 다항식 $p_k(x)$로 근사할 때의 오차 한계 ($\beta \approx 0.28$)

❖ minimax 다항식 표현의 단점

- minimax 다항식을 이용해 절대값 함수 $|x|$를 근사할 때, 원하는 **오차 수준** $2^{-\alpha}$를 얻기 위해 최소한 $\boldsymbol{\Theta}(2^{\alpha})$차수의 다항식 필요.
  - 하지만 일반적인 차수 n의 다항식을 계산하려면 최소한 $\sqrt{n}$번의 곱셈 필요.

  ⇒ 따라서 minimax 다항식을 이용한 연산은 적어도 $\boldsymbol{\Theta}(2^{\alpha/2})$번의 곱셈을 요구하게 됨.

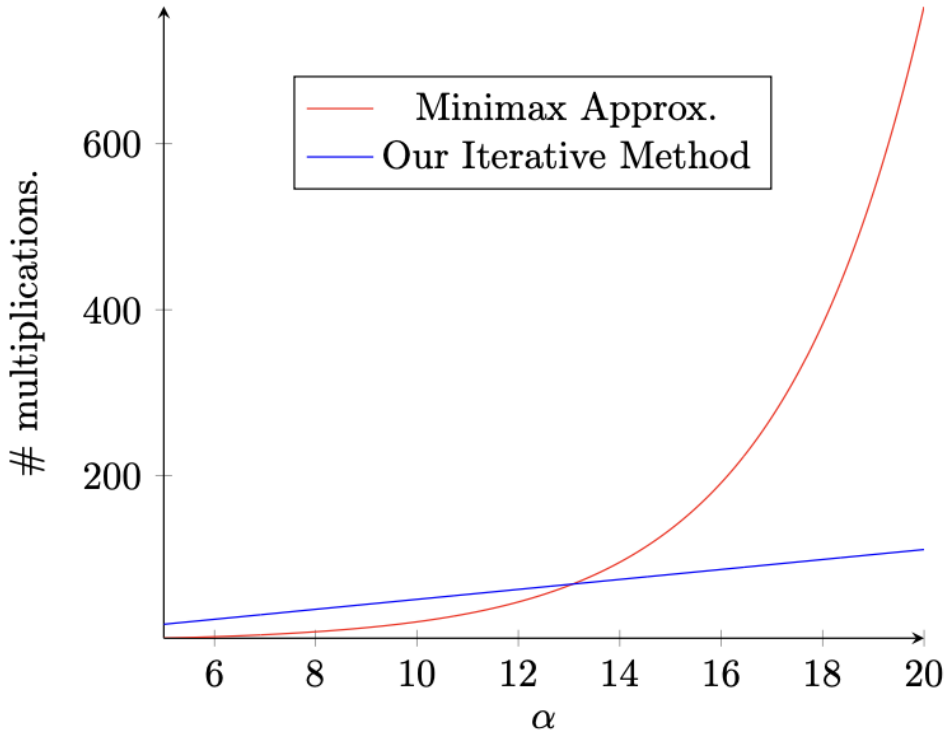
Min/max from Minimax Approximation

Q. 반복 횟수 $2\alpha - 3$에 3 곱하는 이유?

❖ 제안된 방법의 효율성

- **minimax 다항식 근사 vs. 제안된 방법의 연산 횟수 비교**
  - Minimax 다항식 근사: $\sqrt{2\beta} \cdot 2^{\alpha/2} \Rightarrow \boldsymbol{\Theta(2^{\alpha/2})}$
  - 제안된 Iterative 방법: $3 \cdot (2\alpha - 3) = 6\alpha - 9$

    $\Rightarrow \boldsymbol{\Theta(\alpha)}$
  - 제안된 반복적 방법이 정밀도 $\boldsymbol{\alpha \geq 13}$ 에서 더 적은 곱셈 횟수를 필요로 함



[그림 2] Max 함수 계산을 위한 곱셈 횟수 비교 그래프

Comparison from Minimax Approximation

❖ Comparison Function의 다항식 근사

- $comp(a,b) = \chi_{(0,\infty)}(a-b)$ 에서 $a-b$ 의 차이가 **오차 임계값 ϵ보다 큰 경우**에만 근사를 수행하도록 제한함
- Minimax 다항식 근사는 step function과 같은 불연속 함수를 근사할 때 높은 다항식 차수와 계산 복잡성을 요구함. 반면 제안된 Comp 알고리즘은 **계산 복잡성 측면에서 이점**을 가짐.

[표 1] 제안된 방법과 Minimax 근사 방법의 복잡성 비교

		Minimax Approx.	Our Method
min/max		$\Theta(2^{\alpha/2})$	$\Theta(\alpha)$
comparison	$\epsilon = \omega(1)$	$\Theta(\sqrt{\alpha})$	$\Theta\left(\log^2 \alpha\right)$
	$\epsilon = 2^{-\alpha}$	$\Theta\left(\sqrt{\alpha} \cdot 2^{\alpha/2}\right)$	$\Theta\left(\alpha \log \alpha\right)$

Applications of Comparison Algorithms

Applications of Comparison Algorithms

❖ Threshold Counting

- **문제 정의**
  - 주어진 데이터셋 $(a_1, a_2, \dots, a_n)$과 **임계값 b**에 대해 $a_i > b$인 요소의 개수를 찾는 문제
- **해결 방법: Comp 알고리즘을 사용해 각 a_i와 b 비교 →** $\boldsymbol{Threshold(a_1, a_2, \dots, a_n; b; d, d', t, m)}$
  - HE의 **packing** 기법을 사용하여 여러 개의 값을 한 번에 비교 가능

  → packing을 통해 여러 데이터를 하나의 암호문에 저장하고, **SIMD** 방식으로 한 번의 연산으로 여러 데이터를 동시에 비교
  (Single Instruction Multiple Data)
- **출력:** b보다 큰 a_i 값들의 개수를 근사적으로 찾은 결과 반환

> ··· *This organization permits the loan company to utilize the storage facilities of the time—sharing service, but generally makes it difficult to utilize the computational facilities without compromising the privacy of the stored data. The loan company, however, wishes to be able to answer such questions as:*
>
> – *What is the size of the average loan outstanding?*
> – *How much income from loan payments is expected next month?*
> – *How many loans over $5,000 have been granted?*

Experimental Results

Implementations of various Non-Polynomial Operations

❖ HEAAN 기반 구현

- **HEAAN** (Homomorphic Encryption for Arithmetic of Approximate Numbers)
  - 실수/복소수의 근사 계산을 지원하는 동형 암호 스킴
- **실험 환경**
  - Intel Xeon CPU E5-2620 v4 (2.10GHz) processor, Linux 기반 C++
- **보안 수준 및 파라미터**
  - security level $\lambda \geq 128$ / scaling bit $p =$ 약 40 / $\log N = 17$
- **실행 시간 측정**
  - **Actual Running Time**
  - **Amortized Running Time** → Plaintext Batching을 통해 한 번의 동형 연산으로 다수의 데이터를 동시에 처리할 수 있음

Experimental Results 1

❖ HEAAN 기반 Max Algorithm 구현 결과

- 높은 정밀도 설정 시 실행 시간이 증가하며, $\alpha = 20$ 에서 보안 수준이 다소 낮아짐($\lambda > 80$)
- **32비트** 정수 두 개의 최대값을 근사적으로 구하는 데 적용한 결과로, $\alpha = 12$에서 Amortized Running Time이 **1.25ms**
  - *cf) 기존의 bit-wise HE 기반 Max 알고리즘은 두 개의 **8비트** 정수의 최댓값을 계산하는 데 **1ms** 소요..*

⇒ 본 연구에서 Max 알고리즘은 이와 비슷한 실행 시간을 보이면서도 더 큰 입력 크기를 처리할 수 있다.

[표 2] 정밀도에 따른 HEAAN 기반 Max 알고리즘 성능

Algorithm	# precision bits α	# iterations d	Running time	
			Total (s)	Amortized (ms)
Max	8	11	29[a)]	0.44
	12	17	82[c)]	1.25
	16	23	145[d)]	1.66
	20 ($\lambda > 80$)	30	372[e)]	5.68

a) $\log Q = 930$
c) $\log Q = 1410$
d) $\log Q = 2127$
e) $\log Q = 3062$

Experimental Results 2

❖ HEAAN 기반 Compare Algorithm 구현 결과

- 정밀도가 높을수록 더 많은 반복 횟수가 필요하지만, **특정 조건 ($c = 1.01, 1.02$)**에서는 반복 횟수를 줄여도 **충분한 성능**을 보임.
  - *cf) 기존의 bit-wise HE 기반 두 개의 8비트 정수에 대해 비교 연산을 수행에서는 **1.5ms** 소요.*

[표 3] 정밀도에 따른 HEAAN 기반 Comp 알고리즘 성능

Algorithm	# precision bits α	# iterations (d', d, t)	Running time	
			Total (s)	Amortized (ms)
Comp (exact)	8	(5, 5, 6)	238[a)]	3.63
	12	(5, 6, 8)	572[b)]	8.73
	16	(5, 6, 11)	1429[c)]	21.8
	20	(5, 6, 13)	2790[d)]	45.6
Comp ($c = 1.01$)	14	(5, 5, 5)	232[a1)]	3.54
Comp ($c = 1.02$)	20	(5, 4, 5)	189[a2)]	2.88

a) $\log Q = 1870$
b) $\log Q = 3091$
c) $\log Q = 4731$
d) $\log Q = 6221$
a1) $\log Q = 2131$
a2) $\log Q = 1931$

(d, d': *Inv* 반복횟수, t : *Comp* 반복횟수)

Conclusion

❖ 연구 요약

- **문제 제기**: 기존의 **bit-wise** 암호화 방식은 비교 및 최소/최대 연산에 boolean 함수를 사용하여 **계산 비용이 높음**
- **제안 방법**: **word-wise**로 암호화된 데이터의 min/max 및 compare 연산을 **근사적으로 계산하는 반복 알고리즘**을 제안

❖ 성능

- **Min/max 연산: $\Theta(\alpha)$ / Compare 연산: $\Theta(\alpha \log \alpha)$**
  - 기존의 Minimax 다항식 근사 방법 $\Theta\left(2^{\alpha/2}\right)$ 및 $\Theta\left(\sqrt{\alpha} \cdot 2^{\alpha/2}\right)$ 보다 효율적임

❖ 실험 결과

- **HEAAN**을 사용하여 두 개의 ℓ-bit 정수의 근사 최댓값을 오차 $2^{\ell-10}$ 내에서 계산하는 데 걸리는 **Amortized Running Time**은 $\mathbf{1.14}\boldsymbol{ms}$로, bit-wise HE 기반 결과와 비슷한 성능을 보임.

End