Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.3, Jun. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.3.413

완전동형암호로 암호화된 데이터에 적합한 산술 가산기의 구현 및 성능향상에 관한 연구*

서 경 진,† 김 평, 이 윤 호‡
서울과학기술대학교

Implementation and Performance Enhancement of Arithmetic Adder for Fully Homomorphic Encrypted Data*

Kyongjin Seo,† Pyong Kim, Younho Lee‡
SeoulTech

요 약

본 연구에서는 완전동형암호로 암호화된 데이터에 적용할 수 있는 가산기 및 다수개의 데이터를 가산할 때 적용할 수 있는 성능이 향상된 가산 방법을 제안한다. 제안 산술 가산기는 기존의 하드웨어 기반의 산술 가산기 중 최적 회로 단계(level)를 가지는 Kogge-Stone Adder 방법을 기반으로 하며, 완전동형암호가 제공하는 암호학적 SIMD(Single Instruction for Multiple Data) 기법을 적용하기에 적합하게 설계되었다. 제안한 다수 가산 방법은 완벽한 가산 결과를 보장하는 Kogge-Stone Adder를 반복적으로 사용하여 다수개의 데이터를 가산하지 않고, 3개 이상의 수를 더해야 할 경우, Full-Adder를 이용하여 3개의 수를 최종 C(Carry-out)과 논리합의 결과인 S(Sum) 의 두 개로 줄인다. 이러한 과정을 반복하여 최종적으로 두 개의 수를 더할 경우에만 Kogge-Stone Adder를 사용하여 가산하는 방법이다. 제안 방법은 더하고자 하는 데이터의 개수가 많아질수록 성능이 비약적으로 향상되었고, 이를 실험을 통해 검증한다.

ABSTRACT

In this paper, we propose an adder that can be applied to data encrypted with a fully homomorphic encryption scheme and an addition method with improved performance that can be applied when adding multiple data. The proposed arithmetic adder is based on the Kogge-Stone Adder method with the optimal circuit level among the existing hardware-based arithmetic adders and suitable to apply the cryptographic SIMD (Single Instruction for Multiple Data) function on encrypted data. The proposed multiple addition method does not add a large number of data by repeatedly using Kogge-Stone Adder which guarantees perfect addition result. Instead, when three or more numbers are to be added, three numbers are added to C (Carry-out) and S (Sum) using the full-adder circuit implementation. Adding with Kogge-Stone Adder is only when two numbers are finally left to be added. The performance of the proposed method improves dramatically as the number of data increases.

Keywords: Fully Homomorphic Encryption, Adder, Operations Over Encrypted Data, Applied Cryptography, Security

I. 서 론

클라우드 컴퓨팅 기술의 보급은 서버에 저장되는 데이터의 프라이버시(privacy) 보호의 중요성을 증가시켰다[1]. 클라우드 컴퓨팅을 사용하기 위해 기

Received(02. 28. 2017), Modified(04. 13. 2017), Accepted(04. 18. 2017)
* 본 논문은 2016년도 동계학술대회에 발표한 우수논문을 개선 및 확장한 것임.

* 본 논문은 2017년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(2016R1C1B2011022, 2016R1A4A1011761)
† 주저자, skjking86@naver.com
‡ 교신저자, younholee@seoultech.ac.kr(Corresponding author)

업은 자신의 데이터의 일부 또는 모두를 아웃소싱 해야 한다. 따라서 데이터의 소유자와 관리자가 달라지게 되는 것을 의미하며, 결국 보안적인 측면에서 문제점을 야기한다 [13]. 따라서 클라우드 환경에서는 데이터 프라이버시를 보장할 수 있는 추가적인 안전장치가 필요하다.

데이터 프라이버시(privacy)는 클라이언트가 서버에 자신의 데이터를 업로드하기 전에 모든 데이터를 암호화하는 것으로 달성할 수 있다. 그러나 기존의 일반적인 암호화 알고리즘인 AES, RSA 등을 사용할 경우, 암호화된 데이터를 사용하여 임의의 연산을 수행할 수 없기 때문에 서버에서 사용자가 요구하는 데이터 처리가 복호화 없이는 불가능하다. 이러한 상황에서 데이터 프라이버시 보호와 데이터 처리를 동시에 이룰 수 있는 방법으로써 완전동형암호(fully homomorphic encryption)를 대체 암호화 알고리즘으로 고려할 수 있다[2].

기존 암호화 알고리즘은 암호화된 상태에서 연산 기능을 제공하지 않지만 완전동형암호의 경우 암호화된 상태에서 무제한적으로 임의의 연산이 가능하고 연산의 결과도 암호화된 상태로 제공된다. 완전동형암호 환경에서, 사용자는 공개키로 암호화한 데이터를 서버로 전송하고, 서버는 전송받은 데이터를 복호화 없이 사용자가 요구한 연산을 수행할 수 있다. 서버는 사용자의 요구기능 수행 결과인 암호문을 사용자에게 전송하면, 사용자는 개인키로 전송받은 암호문을 복호화 하면 사용자 자신이 원하는 결과를 알 수 있다. 또한 이 과정에서 서버는 사용자가 요구한 연산을 수행하기 위해 복호화를 하여 연산을 수행하는 것이 아닌 암호화된 상태에서 연산을 수행하기 때문에 데이터 프라이버시가 보장된다.

완전동형암호는 기계학습 분야의 응용(암 진단, 질병 진단 등), 단순한 통계(평균, 표준편차) 등의 연산이 필요하면서, 동시에 데이터에 대한 프라이버시가 강하게 요구되는 환경에서 사용될 수 있다. 예를 들어 완전동형암호를 기계학습 분야에 응용한다면 학습 및 예측에 사용되는 데이터를 복호화 하지 않고 학습 및 적용이 가능하므로 데이터 프라이버시가 필요한 암 진단, 질병 진단 등에 유용하게 쓰일 수 있다. 이러한 프라이버시 보호 기계학습 알고리즘 구현은 사용자의 데이터 프라이버시를 지키면서 사용자가 원하는 기계학습을 수행 할 수 있게 해준다.

위와 같은 프라이버시 보존 기계학습이나 통계, 검색 기능을 수행하기 위해서는 완전동형암호로 암호화된 데이터에 대한 가산 연산이 기본적으로 필요하다. 특히 프라이버시 보존 통계의 경우 다수의 데이터를 가산해야 하는 경우가 있다. 이 경우 가장 쉽게 생각할 수 있는 방법은 일반적인 형태의 가산 알고리즘을 완전동형암호 알고리즘에 적용하는 것이다. 그러나 불행히도 가산기 자체의 복잡도로 인하여 수행시간이 오래 걸릴 뿐만 아니라, 필요한 회로 깊이가 깊은 관계로, 일정 수준의 회로 깊이 연산 마다 수행해 주어야 하는 Recryption를 여러 번 수행하여야 한다. 이는 매우 큰 오버헤드를 야기하게 된다. 완전동형암호로 암호화된 데이터를 이용한 가산연구는 다음과 같다. 2013년에 A. Chatterjee 등은 완전동형암호로 암호화된 데이터의 정렬 알고리즘 구현을 위해 산술 가산기를 구현하였다 [3]. 그러나 가산기의 성능개선에 대한 고려가 이루어지지 않아 실제 사용하기는 어려움이 있다. 2015년에 G. S. Cetin, Y. Doroz, B. Sunar, 그리고 E. Savas는 완전동형암호로 암호화된 데이터의 정렬 알고리즘 구현을 위해 산술 가산기를 구현하였지만 Hamming- Weight를 계산하기 위한 1-bit 가산기만을 구현하여 많은 bit의 데이터를 가산하기에는 어려움이 있다[4]. 2016년에 Cheon, J. H., Kim, M., Kim, M.은 데이터베이스의 임의의 질의를 구성할 때 필요한 두 가지 가산기를 제안하였다[5]. 본 연구에서는 평문공간(Plaintext Space)을 1 bit가 아닌 mod 2^{15}로 정의되는 환경에서 연산을 수행하기 때문에 해당 연산 이후 다른 연산 수행시 일반적인 이진 회로 기반의 구현의 적용이 어려워진다는 단점이 있다.

본 연구는 클라우드 컴퓨팅 환경에서 완전동형암호로 암호화된 데이터를 실제 가산 연산을 수행 및 응용할 수 있도록 산술 가산기와 다수의 데이터를 가산할 때 적용할 수 있는 성능향상기법을 제안한다. 하드웨어 기반의 가산기를 완전동형암호로 암호화된 데이터에 적합하게 개선하고, 특히 다수의 데이터를 가산할 때 기존에 알려진 덧셈 알고리즘을 그대로 적용할 경우 많은 수행시간이 소요되기 때문에 이러한 오버헤드를 개선하는 연구를 수행한다.

완전동형암호로 암호화된 데이터를 연산할 때 성능향상을 위해 가장 주목해야 할 점은 최적 회로 단계를 구성해서 가장 오버헤드가 큰 Recryption연산을 최대한 수행하지 않아야 한다는 것이다. 완전동형암호에서는 특정 횟수의 bitwise-AND 연산을 연속적으로 수행할 경우 Recryption 연산을 수행해야 한다. 완전동형암호의 특성상 암호문간의 연산을 수행하면, 그 결과로 암호문 내부에 노이즈(noise)값

이 쌓이게 된다. 이러한 노이즈가 정해진 범위를 벗어나면 복호화시 올바른 값이 나오질 않기 때문에, 특정 횟수 이상의 연속된 bitwise-AND 연산 이후에는 반드시 Recryption을 해 주어야만 추후 계속적인 연산이 가능하다. Recryption 연산으로 노이즈의 양을 줄이는 것에 한계가 있기 때문에, 첫 Recryption 연산 후에는 이전 보다 적은 횟수의 bitwise-AND 연산을 할 때마다 Recryption 연산을 수행해야 한다. 즉, Recryption 연산의 횟수는 구현 회로의 AND 깊이에 의존한다. 일회 Recryption연산의 오버헤드가 다른 연산에 비해 수천배 이상이기 때문에, 큰 오버헤드로 인하여 가산기 성능이 매우 떨어져 실제 사용이 힘들어 질 것이다.

본 연구는 다수개의 데이터를 가산할 때 Full-Adder를 도입하여 bitwise-AND 연산을 최소화 하는 방법을 제안하였다. 이를 통하여 Recryption 연산과 Kogge-Stone Adder [23] 만을 사용할 때 생기는 오버헤드를 개선하였다. 다수개의 데이터를 가산하는 기존의 방법은 Kogge-Stone Adder만을 사용하는 방법이 일반적이었지만, 본 연구에서는 bitwise-AND 연산을 줄이기 위해 Full-Adder를 도입하였다. Kogge-Stone Adder만 사용하는 경우 3개의 데이터를 가산할 때 한 번의 Kogge-Stone Adder 사용에 bitwise-AND 연산의 깊이가 7이 필요하므로 총 14의 bitwise-AND 연산깊이가 필요하다. 하지만 Full-Adder를 적용 할 경우, 3개의 데이터를 가산할 때 Full-Adder를 사용할 수 있다. 이 때 두 개의 데이터는 Full-Adder의 입력으로 설정하고, 나머지 수는 Carry-in 으로 설정할 수 있다. 이 결과로 생기는 논리합의 결과인 S와 Carry의 결과인 C를 얻을 수 있으며, 이를 위해 단지 bitwise-AND 연산 깊이를 1만 필요하다. 최종적으로 두 개의 수를 Kogge-Stone Adder를 사용하여 가산하면 결과적으로 bitwise-AND 연산 깊이를 총 8 사용하여 최종 결과를 얻어낼 수 있다. 이를 확장하여 3개를 초과하는 다수 개의 데이터를 가산할 때에는 데이터의 개수가 많을수록 Kogge-Stone Adder만 사용하는 방법보다 더욱 좋은 성능을 보여준다.

본 연구에서 제안한 가산 메커니즘은 실험 및 성능측정을 통해 3개 이상 1024개 이하의 데이터를 가산할 때 최소 약 2배에서 최대 약 646배 까지 빠른 성능을 보여주었다. i7-6700 3.40GHz, 16.0GB RAM, Ubuntu 14.04 LTS 환경에서 Table 2와 같이 파라미터를 설정한 후 32bit 데이터를 사용하여 실험을 진행한 결과 정수의 개수가 6개일 때 Kogge-Stone Adder만 사용할 경우 연산시간이 약 600초가 필요했지만, Full-Adder를 도입할 경우 약 130초의 연산시간만으로 결과를 얻어내었다. 정수의 개수가 32개일 때에는 Kogge-Stone Adder만 사용할 경우 약 6960초, Full-Adder를 도입할 경우 약 195초의 연산시간이 소요되었다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구 및 필요한 기술을 조사 분석하며, 3장에서 제안하고자 하는 가산기 및 다수 데이터 가산 메커니즘을 설명한다. 4장에서 실험 결과 및 성능을 살펴본다. 마지막으로 5장에서 결론을 내리고 앞으로의 연구방향을 제시한다.

II. 관련연구

본 장에서 완전동형암호, Prefix Adder (Kogge-Stone Adder) 그리고 암호화된 데이터의 가산 연산 관련 연구를 살펴보고, 기존 연구의 문제점을 파악한다.

2.1 완전동형암호

완전동형암호란 복호화 없이 임의의 알고리즘을 암호문을 입출력 데이터로 하여 구현 할 수 있도록 하는 기술이다 [12]. 완전동형암호는 여러 개의 데이터를 한 번의 연산에 처리할 수 있는 암호학적 SIMD(Single Instruction for Multiple Data)를 제공[1]하고 이를 이용하여 암호화 및 복호화에 소요되는 시간을 줄여 주고, 다수의 데이터에 대해 적은 비용으로 연산이 가능하게 되었다. 이러한 완전동형암호는 암호화된 통계 처리를 위해 복호화된 자료의 유출로 인한 피해를 막을 수 있는 기술로 주목받고 있다 [12].

완전동형암호는 암호문에 내재되어 있는 평문들 사이에 bitwise-AND, bitwise-XOR연산을 암호화되어 있는 상태에서 수행가능하게 하고, 또한 이

1) 암호학적 SIMD 는 수학적 성질을 이용하여 다수개의 비트값을 하나의 평문에 집어 넣는 방법을 의미한다. 암호화 후에도 숨겨져 있는 개별적인 평문에 관한 연산 수행이 가능하기 때문에 마치 일반적인 SIMD 연산과 동일한 효과를 암호문에 대해 얻을 수 있다. [5] 참조.

Setup(1^{λ}): Receives the security parameter λ and returns the system params.

KeyGen$_{\varepsilon}$(params): The key generation algorithm returns the secret key sk and the public key pk. (params: Result of set up algorithm, ε:Define an efficient class of functions Fε that can be performed using a fully FHE by receiving ciphertext as input.)

Encrypt$_{\varepsilon}$(params, pk, m): Returns a cryptopraph c that is encrypted the plain text m. The plain text m may have a vector form of a number of bits.

Decrypt$_{\varepsilon}$(params, sk, c): Returns the plain text m decrypted using sk

Encrypted_XOR$_{\varepsilon}$(params, pk, c_1, c_2): The ciphertexts C1 and C2 are input to perform an XOR operation on the plain texts inherent in each ciphertext, and the resulting ciphertext C is returned. When C1 and C2 are represented by bit strings, bitwise-XOR operation is performed. (Hereinafter reffered to as ⊕)

Encrypted_Multiply$_{\varepsilon}$(params, pk, c_1, c_2): The ciphertexts c1 and c2 are input to perform a bitwise-AND operation on the plain texts embedded in each ciphertext (bitwise-AND operation), and the ciphertext c containing the result is returned. (Hereinafter referred to as ·)

Recryption$_{\varepsilon}$(params, pk, c): If c is the result of repeatedly performing an Encrypted_Multiply over a certain number of times, it is necessary to additionally perform the cryptographic operation using c. The number of times is defined by ε. C ′which encrypts plain text such as c.

Pack$_{\varepsilon}$(params, pk, m_0, m_1,…,m_{l-1}): A plain text m is generated by combining plain texts defined in a number of small plain text spaces.

UnPack$_{\varepsilon}$(params, pk, m): Returns the m0, m1,…,ml-1 that is encoded by m defined in the plain text.

Shift$_{\varepsilon}$(params, pk, c, t): Move the plaintexts in the ciphertext to the left by t slots. (-l ≤t≤l)

Fig. 1. FHE algorithm in HE-lib

연산들을 무제한적으로 사용이 가능하다. 위의 두 가지 연산을 제한 없이 사용할 수 있기 때문에 현존하는 모든 알고리즘의 구현 가능하다 [2]. 대부분의 실용적인 완전동형암호 알고리즘은 회로 기반 계산 모델을 제공한다. 이는 비록 완전동형 암호 알고리즘이 소프트웨어로 구현되더라도 마치 AND와 XOR 게이트로 회로를 구현하듯이 구현해야 한다는 것을 의미한다.

평문 m_1, m_2, …, m_n을 완전동형암호를 사용하여 암호화한 암호문 c_1, c_2, …, c_n과 평문에 적용할 수 있는 함수 f가 있다고 가정할 때, 완전동형암호에서 제공하는 두 가지 연산을 바탕으로 f를 수행할 수 있는 회로(Circuit)를 구성할 수 있으며, 이는 m_1, m_2, …, m_n에 해당하는 암호문 c_1, c_2, …, c_n을 이용하여 f(c_1, c_2, …, c_n)을 수행할 수 있다는 것을 의미한다. 수행결과는 암호화 할 때와 같은 공개키로 암호화되어 있는 상태이며 공개키에 대응되는 개인키로 암호문을 복호화하면 평문 f(m_1, m_2, …, m_n)을 얻어 낼 수 있다.

본 연구에서는 완전동형암호[9] 방법을 구현한 HElib(Homomorphic Encryption library) [10]을 사용하였다. Fig. 1.은 HE-lib에 구현된 완전동형암호 알고리즘의 설명이다.

2.2 Prefix Adder

기본적인 하드웨어 기반의 Adder인 Ripple Carry Adder는 Full-Adder를 직렬로 연결한 형태이다 [21]. 이것은 하위비트의 Carry Out이 다음 비트 연산에 사용되므로, 한 비트를 계산 후 그 결과를 이용하여 다음 비트를 순차적으로 계산하는 방식이기 때문에 병렬처리가 불가능하여 비효율적이다. 이 부분을 개선한 것이 Carry Look-Ahead Adder이다 [24]. 이것은 Carry Generate (G), Carry Propagate (P) 개념을 도입하는데 G는 해당 비트에서 Carry를 생성하는 로직이고, P는 해당 비트의 캐리를 전달하는 로직이다. G가 1이면 Carry를 생성하고 P가 1이면 하위 bit Carry를 상위 비트로 전달해 준다. 즉 입력 데이터가 A, B라고 했을 때, $G_i = A_i \cdot B_i$, $P_i = A_i \oplus B_i$이고 $C_{i+1} = G_i \oplus P_i \cdot C_i$ (i는 자리수, C는 Carry)를 만족한다. 이것을 몇 개의 단계로 나누어 각 단계에서 병렬연산 하는 것이 Prefix Adder이다.

2.2.1 Kogge-Stone Adder

Kogge-Stone Adder는 가산 연산을 수행하기 위해 일반적으로 선택되는 고성능 가산기이고 [20,23], Parallel Prefix Adder의 한 종류이다. Brent-Kung Adder [25] 나 Han Carlson Adder [19] 등과 같은 다른 Prefix Adder보다 출력되는 노드의 개수가 많아 데이터의 저장 공간이 많이 필요하다는 단점이 있지만 각 연산 단계마다 일정한 규칙을 갖고 있어 일반화가 쉽다. 또한 최소한의 bitwise-AND 연산을 수행하기 때문에 최적의 연산 단계로 가산 연산 수행이 가능하고 이는 완전동형암호 알고리즘으로 구현 시 빠른 속도로 가산 연산을 수행할 수 있다는 것을 의미한다. Kogge-Stone Adder의 4비트 가산기의 예는 다음 Fig. 2.와 같이 표현 할 수 있다.

Fig. 2의 Third level에서 우측 하단의 두 자리의 결과값 C_1 과 C_0 는 각각 G_1 과 G_0 값이다.

완전동형암호 환경에서 이러한 Adder를 구현할 때 가장 신경써야 하는 것은 가산 회로의 논리적 깊이(logical depth, bitwise-AND 연산단계)를 최대한 줄여 오버헤드가 큰 Recryption 연산을 최소화 하는 것이다.

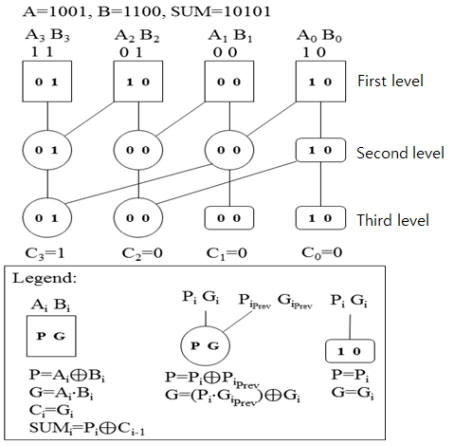

Fig. 2. 4bit Kogge-Stone Adder(q=4bit)

2.3 암호화된 데이터의 가산 연산 관련 연구

암호화된 데이터의 산술 가산 연산을 수행하는 가장 기본적인 방법으로는 암호문 전후로 가산 연산을 보존하는 가산준동형암호(additive homomorphic encryption)를 이용하는 것이다. 암호알고리즘의 준동형성(homomorphism)은 복호화 과정 없이 암호화된 데이터의 연산을 수행할 수 있도록 하는 중요한 개념이며, 가산준동형암호 알고리즘 $E_{homomorphic}$의 경우 암호문만을 가지고 대응되는 평문의 가산 연산만을 수행할 수 있도록 해준다. 가령, 평문 p_1과 p_2의 암호문 $c_1=E_{homomorphic}(p_1)$과 $c_2=E_{homomorphic}(p_2)$가 있을 때, 암호문에서 c_1과 c_2만을 사용하여 $E_{homomorphic}(p_1+p_2)$를 구할 수 있도록 한다. [14[2)], 15, 16]에서는 이와 같은 가산 연산을 보존하는 준동형암호을 다루고 있으며, 실제 이를 적용하여 암호문의 검색(search), 또는 질의(query)를 수행하는 [17, 18]과 같은 연구가 존재한다. 그러나 해당 연구들은 가산 연산만을 보존하며, 이러한 연산만을 사용하여서는 해당 연구가 목표로 하는 기능의 구현이 불가능하기 때문에 복수의 암호 알고리즘 및 프로토콜의 복합구성 기법을 사용하여 원하는 목표를 달성하였다.

완전동형암호(fully homomorphic enryption)는 2가지 연산(bitwise-XOR, bitwise-AND)의 무제한적인 사용을 통해 임의의 효율적인 알고리즘이 모두 구현 가능하다는 성질인 (Universal Computability)를 만족한다. 따라서 암호문만을 사용하여 특정 기능 구현에 제한이 있는 가산연산보존암호의 좋은 대안이라고 할 수 있으며, 완전동형암호를 사용하여 산술 가산기를 구현을 다룬 연구로는 [3-5]가 있다.

[3, 4] 연구는 암호문의 정렬과정에서 필요한 산술 가산기를 제안한 연구들이다. [3]연구의 경우 두 암호문의 크기를 비교 연산(comparison)에 필요한 모듈로 산술 가산기를 사용한다. 이 과정에서 순차캐리가산기(sequential carry adder)를 암호문의 bitwise-XOR와 bitwise-AND 연산을 사용하여 circuit style로 구현하였다. 하지만 Sequential Carry Adder circuit는 자체적으로도 느린 성능을 가지고 있으며, 연구 [3]에서는 1개 암호문에 여러 개의 데이터를 인코딩하는 Cryptographic SIMD 기법의 적용도 고려하고 있지 않기 때문에 병렬처리(parallel processing)를 통한 성능향상 또한 이루어지지 않았다. 연구 [4]는 비교 연산의 1-bit 결과들을 취합하여 Hamming-weight를 구하는 과정에서 산술가산기를 사용한다. N개의 1-bit의 결과를

2) 본 연구는 모듈러 곱셉 연산만을 동형암호로 제공한다.

모두 더하여 log N-bit의 가산결과를 산출하는 과정에서 효율적인 알고리즘을 제안한다. 그러나 [4] 연구 또한 Cryptographic SIMD 기법의 적용이 고려되어 있지 않기 때문에 효율성 측면의 문제가 존재한다.

연구 [5]의 경우 완전동형암호를 사용하여 데이터베이스의 임의의 질의를 구성할 때 필요한 2가지 가산기를 제안하고 있다. 첫 번째 가산기는 1개 암호문에 여러 개의 데이터가 인코딩 된 Cryptographic SIMD 기법의 적용을 가정하고 있으며, 내부의 데이터 이동을 고려하여 연산 오버헤드를 최소화하는 방안을 제안하고 있다. 그러나 내부 데이터 이동을 일반화하여 설명하지 않았다. 두 번째 방법은 암호문에 인코딩 될 평문공간(plaintext space)을 1 bit 기준이 아닌 N-bit로 변경하여 XOR 연산이 가산 연산으로 진행될 수 있도록 하였다. 그러나 평문 공간의 변경으로 인해 가산 연산 이후의 암호문은 회로를 기반으로 하는 연산의 적용이 불가능해진다는 단점이 있다.

III. 가산기 제안 및 성능 개선 방향

본 장에서는 완전동형암호로 암호화된 데이터를 연산할 수 있는 가산기의 세부적인 구현 알고리즘을 설명하고, 다수의 데이터를 가산할 때 성능 개선 방향을 다룬다.

3.1 완전동형암호를 적용한 산술 가산기

본 세부 절에서는 실제 완전동형암호 환경에서 적용시킨 Kogge-Stone Adder의 알고리즘에 대해 설명한다.

3.1.1 완전동형암호 환경에서 적용한 Kogge-Stone Adder의 알고리즘

Kogge-Stone Adder는 산술 가산기 내부에서 발생하는 Carry를 동시에 계산하며 최적 단계의 회로를 구성한다. Kogge-Stone Adder의 경우 중간 연산값(prefix)의 연산량이 많아진다는 단점이 있다. 그러나 완전동형암호를 적용할 때 산술 가산기의 최적화 문제는 중간 캐리를 계산하여 저장되는 노드를 구하는 알고리즘을 얼마나 일반화 시킬 수 있는지에 대한 문제와 동일하다. 또한 완전동형암호 적용 시 bitwise-AND의 연산이 가장 오버헤드가 큰 Recryption 연산의 수행 기준이 되기 때문에, 회로에서 bitwise-AND 연산의 레벨(level)의 최소화가 중요하다. Kogge-Stone Adder는 Fig. 3.과 같이 Prefix($P_{i_{Prev}}$, $G_{i_{Prev}}$, P_i, G_i, G, P)의 연산의 규칙성과 최적 연산 단계를 갖는다는 측면에서 완전동형암호 적용에 유리하다. Fig. 3.는 완전동형암호 환경에서 개발한 Kogge-Stone Adder의 기능별 알고리즘을 의사코드로 표현한 그림이다.

```
A, B: ciphertext with a q-bit integer packed
SUM: ciphertext to store the result of the
addition operation

function K-S Adder(SUM, A, B, q)
  InitialPrefix(P0, G0, A, B)
  for i<-1 to ⌈log q⌉
     //Function for Carry calculation
     LevelEval(Pi-1, Gi-1, Gi, Pi, I)
  end for

  Out(SUM, P0, G⌈log q⌉)
end funtion
```

```
//refer to Fig. 4
InitalPrefix(P0, G0, A, B)
  P0=A⊕B,   G0=A・B
end function
```

```
Prefix(P_iPrev, G_iPrev, Pi-1, Gi-1, P_iCur, G_iCur)
   G_iCur=(G_iPrev・Pi-1)⊕Gi-1,   P_iCur=P_iPrev・Pi-1
end function
```

```
//refer to Fig. 7
Out(SUM, P0, C, q)
   SUM=(C<<(1)⊕P0
end function
```

```
LevelEval(Pi-1, Gi-1, Gi, Pi, i, q)
// (0~2^(i-1)-1) slots of Gi are
// corresponding to confirmed Carrys of Gi-1
// (F0~F2^(i-1)-1, in Fig. 5 ~ Fig. 6), which
// need not to be changed.
// (0~2^(i-1)-1) slots of Pi are similar.

  for j<-0 to 2^(i-1)-1
     Assign(Gi,Gi-1, j, j, q)
     Assign(Pi,Pi-1, j, j, q)
  end for

// P_iPrev and G_iPrev are shifted version of
// Pi-1 and Gi-1 to calculate Pi and Gi for
// this level.
```

```
// (refer to Fig 5~Fig. 6.)

  G_iPrev = G_i-1 << 2^(i-1), P_iPrev = P_i-1 << 2^(i-1)

// Calculation of the prefix for next level,
// using P_iPrev, G_iPrev, P_i-1, and G_i-1.
// (P_iCur, G_iCur are the results)

  Prefix(P_iPrev, G_iPrev, P_i-1, G_i-1, P_iCur, G_iCur)

// P_i and G_i combine the confirmed
// part(e.g., Carrys) and the calculated
// prefix(P_iCur, G_iCur).

  for j<-2^(i-1) to q-1
     Assign(G_i, G_iCur, j, j, q)
     Assign(P_i, P_iCur, j, j, q)
  end for

end function
```

```
Assign(A, B, a, b, q)
  ptxt[b]=1 // ptxt is q size plaintext
  mask = encrypt(ptxt)
  A=A⊕(mask · B<<(b-a))
end function
```

Fig. 3. Pseudocode of Kogge-Stone Adder

본 연구에서 적용한 완전동형암호의 환경설정 파라미터는 IV장의 Table. 2.와 같다. 이것과 같이 파라미터 설정을 할 경우, 암호화 할 수 있는 슬롯의 개수는 1200 slot이고 30 slot 단위로 1 bit 데이터를 삽입할 수 있다. 따라서 최대 40 bit의 데이터를 암호화할 수 있고, Recryption 연산을 하지 않고 최대 24 깊이의 bitwise-AND 연산을 지원한다. 24의 bitwise-AND 연산 깊이를 모두 사용하면 Recryption 연산을 수행해야 하고, 이후 3 bitwise-AND 연산깊이를 사용할 때 마다 Recryption 연산을 수행해야 한다.

Fig. 4~7은 Fig. 3.의 Kogge-Stone 알고리즘을 완전동형암호를 바탕으로 구현한 전체 흐름 및 세부 알고리즘에 대한 설명이다. Fig. 3의 각 단계에서 Carry를 계산하기 필요한 Prefix들은 InitalPrefix, Prefix, LevelEval 함수를 통해 구하며, Assign 함수는 암호문 q-slot 사이의 데이터 사이의 위치이동 및 저장을 위한 함수이다. Carry의 계산이 끝나면 Out 함수를 통해 최종결과를 구한다. 편의상 데이터의 2^k 자리에 해당하는 Carry를 F_k라고 할 때, InitialPrefix와 LevelEval 함수는 모든 F_1~F_q을 구하기 위한 함수이며, K-S Adder에서 수행되는 과정은 다음과 같다.

Fig. 4의 ①,② 과정은 두 개의 입력 A, B를 각각 Half-Adder(bitwise-AND, bitwise-XOR) 연산을 통해 P_0, G_0, 그리고 Carry F_1을 구하는 과정이며 Fig 3의 K-S Adder에서 InitalPrefix 함수를 호출하여 수행된다. ② 과정은 확정 Carry를 구하는 일반화된 회로이다. 그런데 F_0(Carry In)는 일반적으로 0이기 때문에 ② 과정을 통해 구해지는 F_1은 G_0의 2^0 자리의 값과 같은 값을 가지며 ② 과정은 생략 가능하다(Fig 3에서는 생략).

Fig. 5의 ③,④,⑤ 과정에서는 ①,② 과정 다음 레벨의 P_1, G_1 그리고 Carry F_2, F_3 을 구하기 위한 단계이다. 해당 단계는 Fig. 3의 LevelEval 함수를 통해 수행된다. Fig 5에서 알 수 있듯이 P_1과 G_1은 기존 P_0과 G_0을 그대로 사용하는 부분과, 이전단계에서 발생한 Carry 또는 Prefix와의 연산을 통해 계산되는 부분으로 나뉜다. P_0와 G_0를 그대로 사용되는 부분은 각 2^0 자리이며, Fig 3의 Assign 함수를 사용하여 P_1과 G_1의 해당 위치로 할당한다(LevelEval의 첫 번째 for 블록, Fig. 5-⑤).

다른 부분의 연산과정에서 필요한 이전 단계의

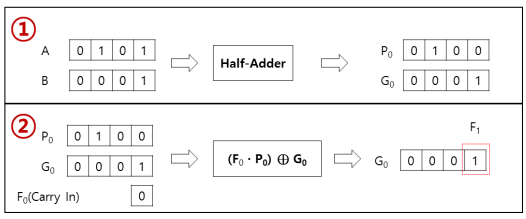

Fig. 4. Kogge-Stone Adder algorithm to add data encrypted with FHE 1

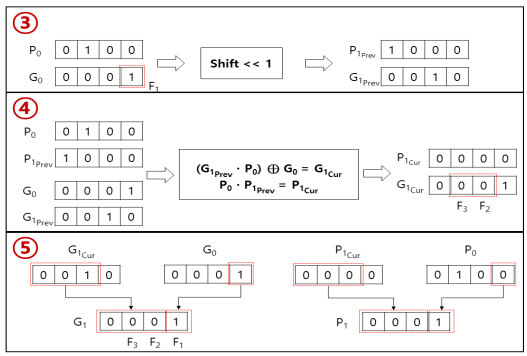

Fig. 5. Kogge-Stone Adder algorithm to add data encrypted with FHE 2

Prefix와 Carry는 ③과 같이 P_0과 G_0를 FHE의 Shift 알고리즘을 사용하여 상위비트로 2^{1-1} bit 씩 이동시킨 $P1_{Prev}$, $G1_{Prev}$가 되며, ④와 같이 $P1_{Prev}$, P_0, $G1_{Prev}$, G_0을 사용하여 다음 레벨에 필요한 Prefix와 Carry를 계산한다. 해당 연산은 Fig 3의 LevelEval에서 Prefix 함수를 호출하여 수행한다. 마지막으로, 이렇게 구해진 값을 역시 Assign 함수를 사용하여 2^1 ~2^3 자리에 할당하며(LevelEval의 두 번째 for 블록, Fig. 5-⑤), 이때, 확정 Carry F_2, F_3은 ⑤와 같이 G_1에 같이 포함된다.

q bit의 데이터 2개를 가산할 때, Fig. 5의 ③, ④,⑤ 과정을 최종 확정 캐리인 F_0 ~ F_q를 구할 때까지 반복한다. 단, ③에서 P_i, G_i를 계산하기 위한 이전 단계의 P_{i-1}, G_{i-1}은 Shift 알고리즘을 통해 상위비트로 2^{i-1} bit 씩 이동시킨다. Fig 6의 ⑥,⑦,⑧은 이와 같은 반복의 예시로 P_2, G_2 그리고 Carry F_4(q〉7일 경우, F_4~ F_7)를 구하기 위한 과정의 설명이다.

확정 캐리인 F_0 ~ F_q를 전부 구하였다면, 최종적으로 Fig. 7의 ⑨번 과정과 같이 Fig. 3의 Out 함수를 호출하여 A와 B의 bitwise_XOR 연산 결과인 P_0와 확정 Carry들을 bitwise-XOR 연산을 수행하여 최종 가산 결과인 Result를 산출한다. 이때, Carry In인 F_0는 일반적으로 0이기 때문에 F_0

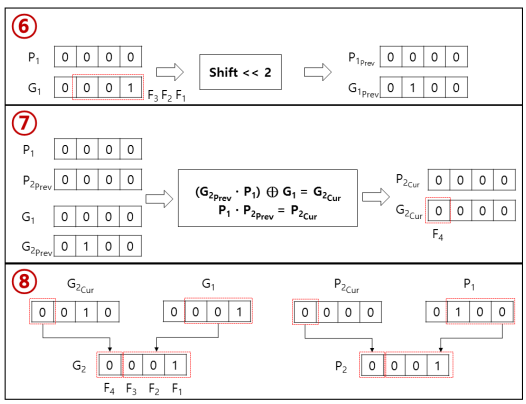

Fig. 6. Kogge-Stone Adder algorithm to add data encrypted with FHE 3

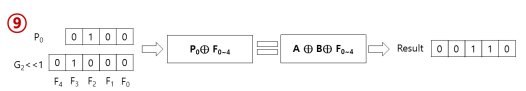

Fig. 7. Kogge-Stone Adder algorithm to add data encrypted with FHE 4

~ F_q가 P_0에 연산되어야하는 형태는 $G\lceil \log q \rceil \langle\langle 1$과 동일하다.

Fig. 3, Fig. 4, Fig. 5, Fig. 6, Fig. 7을 살펴보면 q bit의 데이터 2개를 더할 때, bitwise-AND연산이 $q \le 2^n$ 을 만족하는 가장 작은 n+1번 중첩되어 있는 것을 알 수 있다. 최종 결과를 출력하기 위해 F_i를 정렬하고 의미 없는 데이터를 clear하는 과정이 추가가 되어야 하는데 이때, bitwise-AND연산이 한 번 사용되게 된다. 그러므로 32bit 데이터 기준으로 Kogge-Stone Adder는 bitwise-AND 의 깊이가 7인 회로를 구성하며 이는 1회 수행에 7의 bitwise-AND 연산깊이를 소모하는 것과 같은 의미이다.

3.2 다수개의 가산 연산을 위한 메커니즘 제안

본 연구에서 제안하는 메커니즘은 다수개의 데이터를 가산할 때, 완벽한 가산 결과를 보장하는 Kogge-Stone Adder만을 사용하지 않고 Full-Adder를 도입하는 방법이다. 이를 통하여 Kogge-Stone Adder의 사용횟수를 1회로 줄이고 bitwise-AND 연산깊이를 N개의 데이터를 가산하려 할 때, $\log_2 N + 7$ 만큼의 bitwise-AND 연산깊이만을 사용하여 사용자가 원하는 가산 결과를 얻어낼 수 있다.

3개의 정수의 암호문 X, Y, Z를 가산하기 위해서 Fig. 4.의 알고리즘을 적용시키는 방법은 Kogge-Stone(SUM_1, X, Y, q)를 실행하고 그 결과인 SUM_1과 C를 다시 Kogge-Stone(SUM_2, SUM_1, Z, q)를 적용하여 2회에 나누어 가산기를 적용하는 방법을 생각할 수 있다. 이 때, bitwise-AND 연산깊이는 14를 소모하게 된다. 그런데 X, Y, Z의 각 대응되는 bit를 입력으로 Fig. 3의 bitwise Full-Adder회로를 적용 시킬 경우, 그 출력을 S, C라고 둘 때, X+Y+Z의 가산 결과는 S+C의 가산 결과와 동일하다[22]. 이는 2회의 가산기 수행에 필요한 회로가 1회 가산기 수행과 Full-Adder 연산으로 줄일 수 있고 Full-Adder 연산은 bitwise-AND 연산이 1단계만 사용되기 때문에 bitwise-AND 연산깊이를 1 소모하고, 총 bitwise-AND 연산깊이를 8 소모한다는 것을 의미한다. 또한 전체 n개의 입력에 대해 필요한 Kogge-Stone Adder 연산 횟수는 1회로 줄어든다. C의 결과를 얻기 위해 존재하는 OR-Gate는

(A · B)와 ((A⊕B) · C_{in}))가 모두 1일 경우는 존재할 수 없기 때문에 XOR-Gate로 대체할 수 있다. Fig. 5는 bitwise Full-Adder의 의사코드와 Full-Adder Circuit에 대한 그림이다.

```
Full-Adder(S, Cout, A, B, Cin)
    S=A⊕B⊕Cin,      Cout=((A · B)⊕((A⊕
B) · Cin))〈〈(1)
end function
```

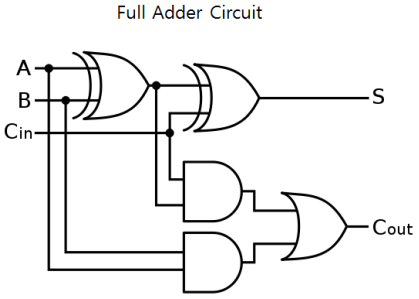

Fig. 8. Pseudocode of Full-Adder and Full-Adder Circuit

Fig. 9은 3개의 정수 암호문 X, Y, Z를 가산하기 위해 Kogge-Stone Adder만 사용한 방법과 제안하는 방법인 Full-Adder회로를 적용시킨 그림이고, Table 1은 Kogge-Stone Adder만 사용하는 방법과 Full-Adder와 함께 사용하는 방법의 연산시간과 bitwise-AND 연산깊이의 결과 비교표이다.

Table. 1.의 결과로 다수 개의 정수 암호문을 가산할 때 제안한 방법인 Kogge-Stone Adder with Full-Adder 메커니즘이 기존의 방법보다 유리하다는 것을 알 수 있다. Recryption을 수행하지 않는 최대의 bitwise-AND 연산깊이가 24일 경우 제안한 메커니즘은 가산하려고 하는 정수 암호문이 N개일 때, $\log_2 N$ 만큼의 bitwise-AND 연산 깊이

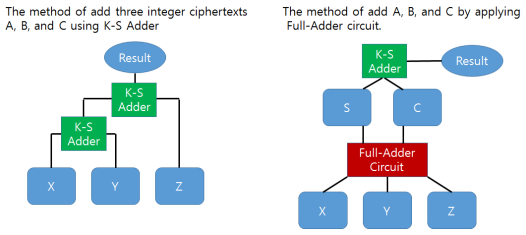

Fig. 9. The method of using Kogge-Stone Adder and applying Full-Adder circuit

Table 1. Comparison table of using Kogge-Stone Adder and applying Full-Adder circuit

	Kogge-Stone Adder	Kogge-Stone Adder with Full-Adder
bitwise-AND depth	14	8
Operation Time	About 120sec + About 120sec ≒ About 240sec	About 120sec + About 1sec(Full-Adder) ≒ About 121sec

만을 소모하여 N개의 정수 암호문을 2개로 줄일 수 있다. 따라서 제안 방법의 실험에서 사용된 환경을 이용하면 가용 깊이가 24 - 7(Kogge-Stone Adder가 소모하는 bitwise-AND 연산깊이) = 17이므로 최대 2^{17}개의 정수 암호문을 가산할 때 Recryption없이 수행 가능하다. 이에 반하여 Kogge-Stone Adder만을 사용하는 기존의 방식은 4번째 단계의 가산부터 Recryption를 수행하기 때문에 8개 이하의 정수 암호문을 가산할 때에만 Recryption없이 수행 가능하다.

Fig. 10, Fig. 11, Fig 12는 제안한 메커니즘인 Kogge-Stone Adder with Full-Adder의 8bit 데이터 6개를 가산할 때 전체 흐름 및 세부 알고리즘에 대한 내용이다.

입력 데이터의 개수가 6개일 때 데이터를 3개씩 묶어서 Full-Adder 연산을 수행하면 bitwise-XOR의 결과인 S와 bitwise-AND의 결과인 C를 얻을 수 있다.

Fig. 10의 ①에서 얻은 S 2개와 C 2개를 묶어서 다시 Full-Adder 연산을 수행하면 ② Full-Adder 연산을 수행하지 못한 나머지 1개의

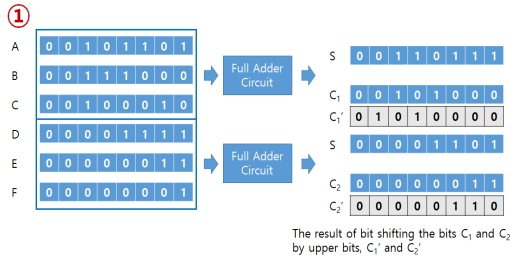

Fig. 10. Kogge-Stone Adder with Full-Adder algorithm 1

데이터와 Full-Adder연산 수행의 결과인 S와 C를 얻을 수 있다. ③에서 S와 C, 나머지 1개의 데이터를 묶어서 다시 Full-Adder 연산을 수행하면 최종적으로 bitwise-AND 연산을 3단계 수행한 (bitwise-AND 연산깊이 3) S와 C를 얻을 수 있다.

Fig. 11의 ③에서 얻은 S와 C를 Kogge-Stone Adder를 이용하여 가산하면 6개의 데이터에 대한 최종 가산 결과를 얻을 수 있다. 이처럼 입력되는 데이터가 다수 개인 경우 반복적으로 Full-Adder Circuit 을 수행하여 데이터의 개수를 줄여나가서 최종 결과인 S와 C만 남을 때 까지 반복한다. Full-Adder의 결과를 Kogge-Stone Adder를 이용하여 가산하는 방법이다. 결과적으로 데이터 6개를 더할 경우 Kogge-Stone Adder만 이용하는 경우 bitwise-AND 연산깊이를 21 소모하지만 제안 방법의 경우 bitwise-AND 연산깊이를 10 소모하고 결과를 얻어낼 수 있다. 즉 1의 bitwise-AND 연산깊이를 소모하여 N개의 데이터를 N/2+1개로 줄일 수 있고, 이는 N개의 데이터를 $\log_2 N$번의 bitwise-AND 연산깊이를 소모하여 데이터를 2개로 줄일 수 있다는 것을 의미한다.

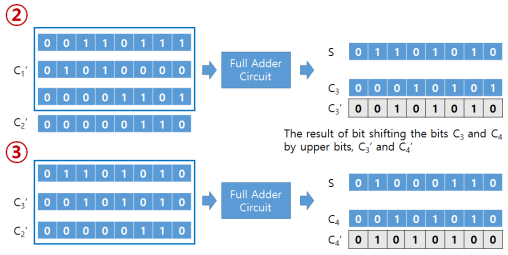

Fig. 11. Kogge-Stone Adder with Full-Adder algorithm 2

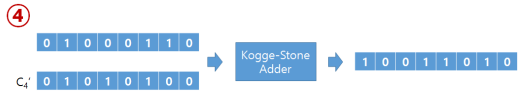

Fig. 12. Kogge-Stone Adder with Full-Adder algorithm 3

IV. 성능분석

본 절에서는 제3장에서 제안된 방법을 실험을 통해 검증한다. 실험 환경은 Intel i7-6700 3.40GHz, 16.0GB RAM, Ubuntu 14.04 LTS 환경에서 수행하였고 실제 구현은 HE-lib[10] 을 사용하였다. Table. 2.와 같이 파라미터를 설정하면 최대 24의 bitwise-AND 연산단계를 제공하며, 이는 bitwise-AND 연산깊이를 Recryption 연산을 수행하지 않고 최대 24까지 사용할 수 있다는 의미이다. 한 번 Recryption을 수행한 뒤에는 bitwise-AND 연산을 3회 수행할 때 마다 Recryption 연산이 필요하다.

Table. 3.은 Kogge-Stone Adder와 Full-Adder Circuit의 16/32bit 입력에 대한 실행시간이다. 3.2에서 제안하는 가산 메커니즘을 적용하여 연산한 다수개의 가산 연산의 결과는 Table 4와 같으며, 데이터의 개수가 많아질수록 매우 큰 효율성을 보여주었다. 이는 Kogge-Stone Adder만 사용할 경우 암호문 연산을 많이 수행하는 과정에서 Recryption 연산의 요구량이 급격하게 증가하는데 반해, 3.2의 가산 메커니즘을 활용할 경우 Kogge-Stone Adder의 사용 횟수가 1회로 줄어들고 전체 연산량도 줄기 때문이다. Fig. 13은 실제

Table 2. Parameter of HElib

Parameters	Values	Remarks
Cyclotomic ring	m = 31755 = 5^2*31*41	
Lattice dimension	phi(m) = 24000	
plaintext space	p = 2, d = 20, GF(2^20)	
number of slots	1200	Total of 40 bits in 30 units available
security level	93	
L, B	L = 25, B = 25	
maximum multiplicative circuit level that can be reached before first Recryptionion: 24(bitwise-AND depth)		

Table 3. Operation time of Kogge-Stone Adder and Full-Adder circuit

	Kogge-Stone Adder		Full-Adder Circuit	
	16 bit	32 bit	16bit	32bit
Operation time(sec)	About 93 s	About 120 s	About 1 s	About 1 s

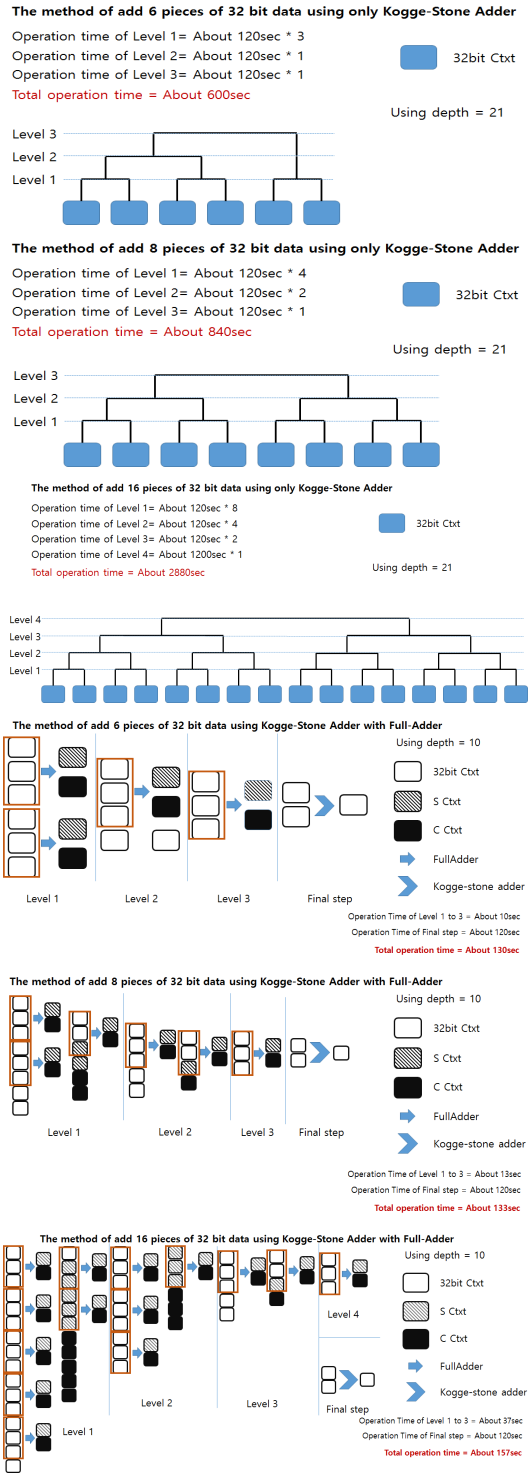

Fig. 13.The operations and results of 6, 8, and 16 pieces of 32 bit with Kogge-Stone Adder and Kogge-Stone Adder with Full-Adder

실험을 통해 32bit 입력이 6개, 8개, 16개일 때 연산 과정과 결과를 보여주고, Table. 4.는 성능이 얼마나 향상되었는지 정리하여 보여준다.

Fig. 13과 Table. 4.의 결과와 같이 다수개의 데이터를 가산할 때 적용 할 수 있는 제안 메커니즘은 정수의 개수가 6개일 경우 약 4.4배 정도 향상된 성능을 보여주었고, 32개일 경우 약 35배 향상된 성능을 보여주었다. 제안 메커니즘은 더하고자 하는 정수의 개수가 많을수록 더욱 향상된 성능을 보여주며, 실험 및 성능분석의 결과 정수의 개수가 1024개일 때 약 646배 향상된 성능을 보여주었다. 다음 Fig. 14는 Kogge-Stone Adder와 Kogge-Stone Adder with Full-Adder의 수행시간을 나타낸 그래프이다.

Table 4. Performance comparison between Kogge-Stone Adder and Kogge-Stone Adder with Full-Adder

Number of Integer	Kogge-Stone Adder		Kogge-Stone Adder with Full-Adder	
	Kogge-Stone Adder usage count	Opration Time	Kogge-Stone Adder usage count	Opration Time
6	5	~10min	1	~130sec
8	7	~14min	1	~133sec
10	9	~36min	1	~138sec
12	11	~40min	1	~144sec
14	13	~44min	1	~149sec
16	15	~48min	1	~157sec
32	31	~116min	1	~195sec

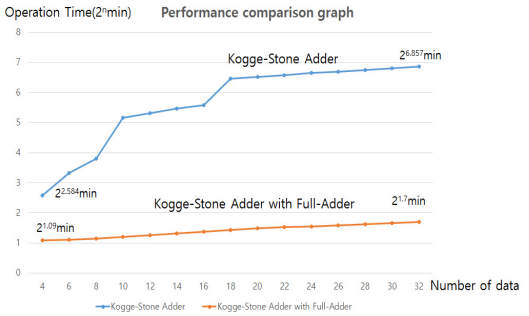

Fig. 14. Performance comparison graph

V. 결 론

본 연구에서는 클라우드 환경에서 완전동형암호를 적용하기 적합한 산술 가산기와 다수개의 데이터를 빠르게 가산하는 향상된 메커니즘을 제안하였고, 이를 실험 및 성능분석을 통해 검증하였다. 제안한 산술 가산기는 SIMD(Single Instruction for Multiple Data)를 적용하기 알맞고, 회로를 기반으로 하는 연산의 적용이 가능하다.

이전의 완전동형암호를 이용한 가산연산에 관련된 연구들은 가산 연산은 가능하지만 'Universal Computability'를 만족하지 않거나, 제한된 조건에서만 수행 가능하거나, 효율적이지 않았다. 그러나 본 연구를 통해 위의 단점들을 개선하였다.

제안한 가산 메커니즘은 완벽한 결과를 도출하는 Kogge-Stone Adder를 사용하지 않고 보다 오버헤드가 적은 Full-Adder를 사용하여 bitwise-AND 연산 깊이를 줄였다. 이 경우 N개의 데이터를 한 연산 단계마다 bitwise-AND 연산 깊이를 1 소모하고 N/2+1개로 줄일 수 있다. 이를 반복하여 최종적으로 $2^{(k-1)} < N < 2^k$ 를 만족하는 k만큼의 bitwise-AND 연산깊이만을 소모하여 2개의 데이터로 줄일 수 있다. 이는 N개의 데이터를 $\log_2 N$의 bitwise-AND 연산깊이를 사용하여 데이터를 2개로 줄일 수 있다는 것을 의미한다. 제안 방법은 가산연산 자체의 수행 시간을 이전 방법보다 크게 감소시키고, 가장 오버헤드가 큰 Recryption연산을 수행하지 않고 이전보다 많은 양의 데이터를 가산할 수 있다. 실험 결과 3개 이상 1024개 이하의 데이터를 가산할 때 최소 약 2배에서 최대 약 646배까지 성능이 향상됨을 확인하였다.

제안한 가산 메커니즘은 가산연산만이 아니라 곱셈연산에도 사용될 수 있다. 두 수의 곱셈연산의 결과는 각 자리수의 부분 곱에 가산 연산을 수행한 결과와 같다. 또한 현재 곱셈기의 속도는 부분 곱을 추가하고 가산하는 연산에 사용되는 덧셈기의 속도에 의해 제한되기 때문에 다수개의 데이터를 가산할 때 유리한 본 연구의 제안방법을 사용하면 보다 빠른 곱셈 연산이 가능하다. 또한 Kogge-Stone Adder 알고리즘을 기반으로 비교기(comparator)의 구현 및 적용이 가능하다.

본 연구에서 제안한 메커니즘은 향후 완전동형암호 환경에서 산술 연산 회로를 소프트웨어로 구현할 경우 기초자료로써 활용이 가능하다. 또한 향후 프라이버시 보존 기계학습이나 프라이버시 보존 통계 그리고 프라이버시 보존 산술연산 등이 필요한 프로토콜 등의 연구 및 개발에 유용하게 사용될 것으로 예상된다.

References

[1] T. Yu, S. Jajodia, Secure data management in decentralized systems, Springer Science and Business Media, vol. 33, pp. 355-380, Springer Press, 2007

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

[3] A. Chatterjee, M. Kaushal, and I. Sengupta, "Accelerating sorting of fully homomorphic encrypted data," the 14th International Conference on Cryptology in India, LNCS 8250, pp.262-273, Dec. 2013.

[4] G.S. Cetin, Y. Doroz, B. Sunar, and E. Savas, "Low depth circuits for efficient homomorphic sorting," In IACR ePrint Arch. 2015/274, Mar. 2015

[5] J.H. Cheon, M. Kim, and M. Kim, "Optimized search-and-compute circuits and their application to query evaluation on encrypted data," IEEE Transactions on Information Forensics and Security, vol. 11, no. 1, pp. 188-199, Jan. 2016.

[6] S. Knowles, "A family of adders," Proceedings of the 14th IEEE Symposium on Computer Arithmetic, pp. 277-281, Jun. 2001.

[7] N.P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," Designs, codes and cryptography, vol. 71, no. 1, pp. 57-81, Apr. 2014.

[8] N. Sureka, R. Porselvi, and K. Kumuthapriya, "An efficient high speed wallace tree multiplier," International Conference on Information

Communication and Embedded Systems, pp. 1023-1026, Feb. 2013.
[9] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," ACM Transactions on Computation Theory, vol. 6, no. 3, pp. 309-325, Jul. 2014.
[10] S. Halevi and V. Shoup, "Design and implementation of a homomorphic encryption library," https://github.com/shaih/HElib, IBM Research (Manuscript) 6, 12-15, 2013.
[11] S. Goldwasser and S. Micali, "Probabilistic encryption and how to play mental poker keeping secret all partial information," Proceedings of the 14th annual ACM symposium on Theory of computing, pp. 365-377, May 1982.
[12] M.I. Jeong "Technical trend of fully homomorphic encryption," The Journal of the Korea Contents Association, vol. 13, no. 8, pp. 36-43, Sep. 2013.
[13] S.J. Jung, Y.M. Bae, "Trend analysis of threats and technologies for cloud security," Journal of Security Engineering, vol. 10, no. 2, pp. 199-212, Apr. 2013.
[14] T.E. Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," IEEE transactions on information theory, vol. 31, no. 4, pp. 469-472, Jul. 1985.
[15] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," In International Conference on the Theory and Applications of Cryptographic Techniques, LNCS 1592, pp. 223-238, 1999.
[16] D. Boneh, E.J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," Proceedings of Theory of Cryptography Conference, LNCS 3378, pp. 325 - 341, 2005.
[17] R.A. Popa, C.M.S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," Proceedings of the 23rd ACM Symposium on Operating Systems Principles, pp. 85 - 100, Oct. 2011.
[18] S. Tu, M.F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," Proceedings of the VLDB Endowment, vol. 6, no. 5, pp. 289 - 300, Mar. 2013.
[19] T. Han and D.A. Carlson, "Fast area-efficient VLSI adders," IEEE 8th Symposium on Computer Arithmetic, pp. 49-56, May 1987.
[20] J. Kaur and P. Kumar "Analysis of 16 & 32 bit kogge stone adder using xilinx tool," Journal of Environmental Science, Computer Science and Engineering & Technology, vol. 3. no. 3, pp. 1639-1644, Jun. 2014.
[21] K. Knauer, "Ripple-carry adder," U.S. Patent No. 4,839,849, Jun. 1989
[22] E.H.J. Persoon and C.J. Vandenbulcke, "Full adder circuit," U.S. Patent No. 5,117,386, May 1992.
[23] P. Kogge, and H. Stone, "A parallel algorithm for the efficient solution of a general class of recurrence equations," IEEE Transactions on Computers, vol. C-22, no. 8, pp. 786-793, May 2009.
[24] C. Babbage, Passages from the life of a philosopher, Cambridge University Press, pp. 59-63 and 114-116, 1864.
[25] R. Brent and H. Kung, "A regular layout for parallel adders", IEEE Transaction on Computers, vol. C-31, no. 3, pp. 260-264, Mar. 1982.

〈저자소개〉

서 경 진(Kyungjin Seo) 학생회원
2013년 8월 : 한성대학교 컴퓨터공학과 졸업
2017년 2월 : 서울과학기술대학교 SW분석설계학과 석사
2017년 3월~현재: 서울과학기술대학교 글로벌융합산업공학과 연구원
관심분야 : 응용암호, 데이터보안, Firmware

김 평(Pyong Kim) 정회원
2007년 2월: KAIST 전산학과 학사
2009년 8월: KAIST 전산학과 석사
2016년 8월: KAIST 전산학과 박사
2016년 9월~현재: 서울과학기술대학교 글로벌융합산업공학과 박사후 연구원
〈관심분야〉 응용암호, 정보보호

이 윤 호 (Younho Lee) 종신회원
2006년 8월:KAIST 전산학과 박사
2007년 10월~2009년 2월: GeorgiaTech Information Security Center 방문 박사후과정
2009년 3월~2013년 8월: 영남대학교 정보통신공학과 조교수
2013년 9월~현재: 서울과학기술대학교 글로벌융합산업공학과 ITM전공 부교수
〈관심분야〉 응용암호, 데이터보안, 시스템보안