

(19) 대한민국특허청(KR)  
(12) 공개특허공보(A)(11) 공개번호 10-2025-0081096  
(43) 공개일자 2025년06월05일

(51) 국제특허분류(Int. Cl.)  
H04L 9/00 (2022.01) H04L 9/30 (2006.01)  
(52) CPC특허분류  
H04L 9/008 (2013.01)  
H04L 9/3093 (2013.01)  
(21) 출원번호 10-2023-0169063  
(22) 출원일자 2023년11월29일  
심사청구일자 2023년11월29일

(71) 출원인  
인하대학교 산학협력단  
인천광역시 미추홀구 인하로 100(용현동, 인하대학교)  
(72) 발명자  
이용우  
인천광역시 미추홀구 인하로 100 (용현동, 인하대학교)  
홍덕화  
경기도 의정부시 동일로 397, 206동 307호 (장암동, 장암1단지주공·장암2단지주공아파트)  
(74) 대리인  
양성보

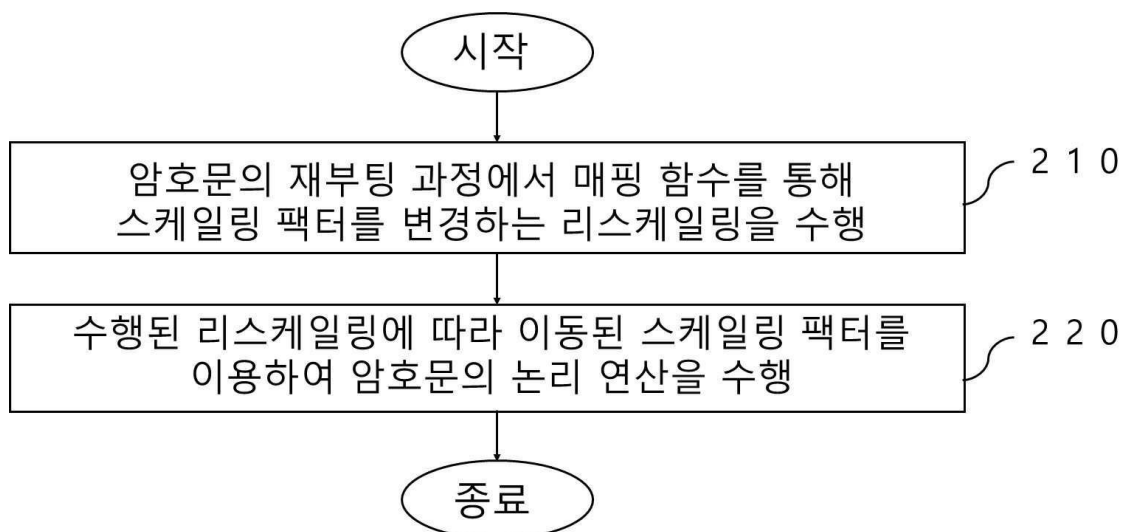
전체 청구항 수 : 총 6 항

(54) 발명의 명칭 양자 내성 동형암호 FHEW 내에서의 효율적인 XOR 연산

## (57) 요약

양자 내성 동형암호 FHEW 내에서의 효율적인 XOR 연산 기술이 개시된다. 일 실시예에 따른 연산 시스템에 의해 수행되는 격자 기반 동형암호에서의 연산 방법은, 암호문의 재부팅 과정에서 매핑 함수를 통해 스케일링 팩터를 변경하는 리스케일링을 수행하는 단계; 및 상기 수행된 리스케일링에 따라 이동된 스케일링 팩터를 이용하여 암호문의 논리 연산을 수행하는 단계를 포함할 수 있다.

대표도 - 도2



이 발명을 지원한 국가연구개발사업

|             |                                    |
|-------------|------------------------------------|
| 과제고유번호      | 1711179347                         |
| 과제번호        | 00155915                           |
| 부처명         | 과학기술정보통신부                          |
| 과제관리(전문)기관명 | 정보통신기획평가원                          |
| 연구사업명       | 정보통신인재양성(교육운영수익)                   |
| 연구과제명       | [Ezbaro][정부+민간] 인공지능융합혁신인재양성(2차년도) |
| 기 여 율       | 1/2                                |
| 과제수행기관명     | 인하대학교 산학협력단                        |
| 연구기간        | 2023.01.01 ~ 2023.12.31            |

이 발명을 지원한 국가연구개발사업

|             |                                       |
|-------------|---------------------------------------|
| 과제고유번호      | 1711193591                            |
| 과제번호        | 2022-0-01047-002                      |
| 부처명         | 과학기술정보통신부                             |
| 과제관리(전문)기관명 | 정보통신기획평가원                             |
| 연구사업명       | 데이터프라이버시글로벌 선도기술연구개발                  |
| 연구과제명       | 실수연산 기반의 동형암호를 활용한 통계 분석 알고리즘 및 모듈 개발 |
| 기 여 율       | 1/2                                   |
| 과제수행기관명     | (주)크립토크                               |
| 연구기간        | 2022.04.01 ~ 2024.12.31               |

---

## 명세서

### 청구범위

#### 청구항 1

연산 시스템에 의해 수행되는 격자 기반 동형암호에서의 연산 방법에 있어서,  
암호문의 재부팅 과정에서 매핑 함수를 통해 스케일링 팩터를 변경하는 리스케일링을 수행하는 단계; 및  
상기 수행된 리스케일링에 따라 이동된 스케일링 팩터를 이용하여 암호문의 논리 연산을 수행하는 단계  
를 포함하는 격자 기반 동형암호에서의 연산 방법.

#### 청구항 2

제1항에 있어서,  
상기 리스케일링을 수행하는 단계는,  
XOR 연산을 수행하기 위한 게이트의 직전 재부팅에서 매핑 함수를 이용하여 평문에 곱해지는 스케일링 팩터를  
기 설정된 값으로 변경하는 단계  
를 포함하는 격자 기반 동형암호에서의 연산 방법.

#### 청구항 3

제2항에 있어서,  
상기 리스케일링을 수행하는 단계는,  
상기 스케일링 팩터를  $-q/8$ 에서  $-q/4$ 로,  $q/8$ 에서  $q/4$ 로 대응시키는 단계  
를 포함하는 격자 기반 동형암호에서의 연산 방법.

#### 청구항 4

제1항에 있어서,  
상기 암호문의 논리 연산은, XOR 연산을 포함하는 것을 특징으로 하는 격자 기반 동형암호에서의 연산 방법.

#### 청구항 5

제1항에 있어서,  
상기 격자 기반의 동형암호는, FHEW인 것을 특징으로 하는 격자 기반 동형암호에서의 연산 방법.

#### 청구항 6

연산 시스템에 있어서,  
암호문의 재부팅 과정에서 매핑 함수를 통해 스케일링 팩터를 변경하는 리스케일링을 수행하는 리스케일링부;  
및  
상기 수행된 리스케일링에 따라 이동된 스케일링 팩터를 이용하여 암호문의 논리 연산을 수행하는 논리 연산부  
를 포함하는 연산 시스템.

## 발명의 설명

## 기술 분야

[0001] 아래의 설명은 동형암호에서의 논리 연산 기술에 관한 것이다.

## 배경 기술

[0003] 양자컴퓨터의 등장으로 기존 암호체계의 붕괴에 대한 우려의 목소리가 나오고 있다. 최근 미국의 국립표준 기술연구소(National Institute of Standards and Technology; NIST)에서 최근에 양자 내성 암호체계에 대한 표준을 제시했다. 그만큼 새로운 암호체계에 대한 필요성이 커지고 있다. 이들이 제시한 표준 중에서는 격자 기반(Lattice based)의 암호체계가 있다. 이 격자 기반 암호는 동형(Homomorphic)의 성질을 갖고 있어 암호문 상태에서의 연산을 지원한다. 양자 내성이 있으며 암호문 상태에서의 연산도 지원하여 매력적으로 느껴지는 암호 체계이다. 그러나 아직 효율성 측면에서 많은 개선이 필요하며, 다양한 연구를 통해 개선되어 왔다.

[0004] 많은 동형암호 시스템이 제안되었으며, 이 중 FHEW는 모든 논리 연산을 지원하여 우수한 활용성을 갖는다. 격자 기반의 동형암호는 암호문 간의 연산을 수행할수록 잡음(noise)이 커지는 특징을 갖는다. FHEW는 이를 해소하기 위해 재부팅(Bootstrapping)이라는 동작을 수행하여 암호문 상태에서 복호화 없이 암호문의 잡음을 제거한다. 그러나, 재부팅 연산은 매우 느리며, 실패할 가능성도 있다. FHEW가 지원하는 논리 연산 중 OpenFHE에서 제공하는 XOR의 경우에는 AND, OR에 비해 잡음에 더욱 취약하다.

## 발명의 내용

### 해결하려는 과제

[0006] 격자 기반(Lattice Based)의 동형암호(Homomorphic Encryption) 중 하나인 FHEW에서 제공하는 논리 연산 중 XOR 연산의 속도와 안전성을 높이는 방법 및 시스템을 제공할 수 있다.

### 과제의 해결 수단

[0008] 연산 시스템에 의해 수행되는 격자 기반 동형암호에서의 연산 방법은, 암호문의 재부팅 과정에서 매핑 함수를 통해 스케일링 팩터를 변경하는 리스케일링을 수행하는 단계; 및 상기 수행된 리스케일링에 따라 이동된 스케일링 팩터를 이용하여 암호문의 논리 연산을 수행하는 단계를 포함할 수 있다.

[0009] 상기 리스케일링을 수행하는 단계는, XOR 연산을 수행하기 위한 게이트의 직전 재부팅에서 매핑 함수를 이용하여 평문에 곱해지는 스케일링 팩터를 기 설정된 값으로 변경하는 단계를 포함할 수 있다.

[0010] 상기 리스케일링을 수행하는 단계는, 상기 스케일링 팩터를  $-q/8$ 에서  $-q/4$ 로,  $q/8$ 에서  $q/4$ 로 대응시키는 단계를 포함할 수 있다.

[0011] 상기 암호문의 논리 연산은, XOR 연산을 포함하는 것일 수 있다.

[0012] 상기 격자 기반의 동형암호는, FHEW인 것일 수 있다.

[0013] 연산 시스템은, 암호문의 재부팅 과정에서 매핑 함수를 통해 스케일링 팩터를 변경하는 리스케일링을 수행하는 리스케일링부; 및 상기 수행된 리스케일링에 따라 이동된 스케일링 팩터를 이용하여 암호문의 논리 연산을 수행하는 논리 연산부를 포함할 수 있다.

## 발명의 효과

[0015] 한 번의 재부팅만으로 잡음에 강한 연산을 수행할 수 있기 때문에 오류 발생 확률을 감소시키고, 연산 속도를 향상시킬 수 있다.

## 도면의 간단한 설명

- [0017] 도 1은 일 실시예에 있어서, 연산 시스템의 구성을 설명하기 위한 블록도이다.
- 도 2는 일 실시예에 있어서, 격자 기반 동형암호에서의 연산 방법을 설명하기 위한 흐름도이다.
- 도 3은 일 실시예에 있어서, OpenFHE 구현과 실시예에서 제안된 기술을 비교한 표이다.
- 도 4는 일 실시예에 있어서, OpenFHE, TFHE 와 실시예에서 제안된 기술의 실패확률을 비교한 표이다.

### 발명을 실시하기 위한 구체적인 내용

- [0018] 이하, 실시예를 첨부한 도면을 참조하여 상세히 설명한다.

- [0020] 실시예에서는 격자 기반(Lattice Based)의 동형암호(Homomorphic Encryption) 중 하나인 FHEW에서 제공하는 논리 연산 중 XOR 연산의 속도와 안전성을 높이는 방법에 대하여 설명하기로 한다. 격자 기반(Lattice Based)의 동형암호(Homomorphic Encryption)란 양자 내성을 가지며 동형의 성질을 갖춘, 즉 암호문 간의 연산을 지원하는 암호체계이다. FHEW에서 제공하는 XOR 연산은 잡음(noise)에 취약하여 비교적 잡음에 강한 AND와 OR 연산의 조합으로 수행하거나 오류(error)가 발생할 확률을 감수하면서 수행되어야 한다. 이에, 빠르고 잡음에서도 강한 새로운 XOR 연산 동작에 대하여 설명하기로 한다.

- [0021] 이러한 XOR 연산 동작의 설명의 이해를 돕기 위해 우선적으로FHEW 동형암호에 대하여 설명하기로 한다. FHEW란 LWE/RLWE에 기반하여 모든 논리 연산을 수행하는 암호체계이다. LWE(Learning with errors)와 RLWE(Ring-Learning with errors)는 격자 기반의 어려운 문제이다. LWE 암호문  $c \in \mathbb{Z}_q^{n+1}$ 는  $(a, b)$ 로 주어지며,  $a \in \mathbb{Z}_q^n$ 는 난수이다.  $b$ 는 다음의 식으로 구할 수 있다.

$$b = \langle a \cdot s \rangle + q/4 \cdot m + e$$

- [0022]
- [0023] 이때,  $s$ 는 비밀키(secret key),  $e$ 는 잡음이다. 복호화 함수  $Dec(c) = b - \langle a \cdot s \rangle = q/4 \cdot m + e$ 로 주어진다. RLWE는 LWE와 달리  $\mathbb{Z}_q$  상에서 정의되지 않고,  $R_q = \mathbb{Z}_q/(x^n + 1)$  상에서 정의되어 키의 크기와 연산량에서 유리하다.

- [0024] FHEW는 LWE/RLWE를 복합적으로 사용하며 논리 연산을 지원한다. 이진 논리 연산을 지원하므로 평문  $m \in \{0, 1\}$ 으로 제한한다. 또한, 암호화를 수행할 때  $m$ 에 곱해지는 값  $q/4$ 를 스케일링 팩터(scaling factor)라 하며, 적절히 큰 값으로 설정해 잡음에 견딜 수 있도록 설정한다.

- [0025] 도 1은 일 실시예에 있어서, 연산 시스템의 구성을 설명하기 위한 블록도이고, 도 2는 일 실시예에 있어서, 격자 기반 동형암호에서의 연산 방법을 설명하기 위한 흐름도이다.

- [0026] 연산 시스템(100)의 프로세서는 리스케일링부(110) 및 논리 연산부(120)를 포함할 수 있다. 이러한 프로세서의 구성요소들은 연산 시스템에 저장된 프로그램 코드가 제공하는 제어 명령에 따라 프로세서에 의해 수행되는 서로 다른 기능들(different functions)의 표현들일 수 있다. 프로세서 및 프로세서의 구성요소들은 도 2의 격자 기반 동형암호에서의 연산 방법이 포함하는 단계들(210 내지 220)을 수행하는 연산 시스템을 제어할 수 있다. 이때, 프로세서 및 프로세서의 구성요소들은 메모리가 포함하는 운영체제의 코드와 적어도 하나의 프로그램의 코드에 따른 명령(instruction)을 실행하도록 구현될 수 있다.

- [0027] 프로세서는 격자 기반 동형암호에서의 연산 방법을 위한 프로그램의 파일에 저장된 프로그램 코드를 메모리에 로딩할 수 있다. 예를 들면, 연산 시스템에서 프로그램이 실행되면, 프로세서는 운영체제의 제어에 따라 프로그램의 파일로부터 프로그램 코드를 메모리에 로딩하도록 연산 시스템을 제어할 수 있다. 이때 프로세서는 리스케일링부(110) 및 논리 연산부(120) 각각에 메모리에 로딩된 프로그램 코드 중 대응하는 부분의 명령을 실행하여 이후 단계들(210 내지 220)을 실행하기 위한 프로세서의 서로 다른 기능적 표현들일 수 있다.

- [0028] 단계(210)에서 리스케일링부(110)는 암호문의 재부팅 과정에서 매핑 함수를 통해 스케일링 팩터를 변경하는 리스케일링을 수행할 수 있다. 리스케일링부(110)는 XOR 연산을 수행하기 위한 게이트의 직전 재부팅에서 매핑 함수를 이용하여 평문에 곱해지는 스케일링 팩터를 기 설정된 값으로 변경할 수 있다. 리스케일링부(110)는 스케일링 팩터를  $-q/8$ 에서  $-q/4$ 로,  $q/8$ 에서  $q/4$ 로 대응시킬 수 있다.

[0029] 재부팅 과정에 대하여 보다 상세히 설명하기로 한다. 암호화 과정에서 잡음을 추가하는 LWE/RLWE의 특성상 암호문 간의 연산을 수행할수록 잡음이 증가하게 된다. 거듭된 연산으로 잡음이 증가해 스케일링 팩터(scaling factor)의 크기를 넘어선다면 이는 연산의 실패로 이어질 것이다. 이를 해결하기 위해 FHEW에서는 재부팅이라는 개념을 도입한다. 누산기(Accumulator) ACC와 수행하고자 하는 게이트(Gate)에 따라 매핑 함수(mapping function)  $f$ 를 정의한다. 이후

$$ACC = RLWE(f = \sum_{i=0}^{q/2-1} f(b-i) \cdot X^i)$$

[0030] 로 초기화한다.

[0031] 이후, 블라인드 로테이션(Blind rotation)이라는 과정을 거쳐  $f \cdot X^u$ 의 RLWE의 암호문을 구한다(이때,  $u = \text{Dec}(c)$ ). 여기서,  $u$ 란  $\text{Dec}(c)$ 로 암호문을 복호화한 값을 의미한다. 암호문 수식은  $b = \langle a, s \rangle + m \cdot q/4 + e$ 로 정의되어 있는 것을 확인할 수 있다. 이때,  $u = b - \langle a, s \rangle = m \cdot q/4 + e$ 로 정의될 수 있다. 다시 말해서,  $s$ 를 갖고있는 사용자밖에 알 수 없는 값이며 지수 상에서 복호화를 진행하니 계수와는 분리되어 있어 잡음을 분리할 수 있는 것이다.  $f \cdot X^u$ 는 다항식(ACC)에서 지수가  $u$ 인 항으로, 지수가  $u$ 인 항의 계수값을 매핑 함수  $f$ 를 통해 새로운 값을 대응시킬 수 있다.

[0032] 그 후, LWE Extraction 과정을 거치면  $f_u$ 의 LWE의 암호문을 구할 수 있으며, 이것이 매핑 함수  $f$ 를 통해 수행하고자 하는 연산의 결과값이다. 여기서, 앞서 정의한  $f \cdot X^u$ 는 다항식에서  $u$ 번째 계수를 추출하겠다는 의미이며(여전히 다항식이다),  $f_u$ 는 추출한 계수값을 의미한다(상수이다). 즉, 다항식의 계수에서 매핑 함수  $f$ 를 이용하여 새로이 값을 정의하고 암호문의 복호화(Decryption)를 지수상에서 수행하여 잡음을 분리해 새로이 암호화할 수 있는 것이다. 재부팅은 연산량이 많아 FHEW의 속도를 저하하는 원인이기도 하다.

[0033] 단계(220)에서 논리 연산부(120)는 수행된 리스케일링에 따라 이동된 스케일링 팩터를 이용하여 암호문의 논리 연산을 수행할 수 있다. 암호문의 논리 연산은 우선 암호문 간의 연산부터 수행될 수 있다. 예시로는  $c1+c2$ ,  $c1-c1$ ,  $2(c1-c2)$  등이 있다. 이러한 덧셈, 뺄셈과 같은 연산이 논리 연산처럼 수행될 수 있는 이유는 매핑 함수가 논리 연산을 수행한 것과 동일하게 값을 바꾸어주기 때문이다. 이때, 암호문에 연산과 스케일링 팩터에 따라 각기 다른 기댓값을 갖게 되는데, 매핑 함수는 기댓값에 따라 각기 다른 대응값을 지정해주게 된다. 이러한 대응값이 연산의 결과값이 된다. 다시 말해서, XOR, AND, OR의 매핑 함수가 전부 다르며, 기댓값이 동일한 경우에도 매핑 함수에 따라 다른 대응값을 가질 수 있다.

[0034] 실시예에서 제안한 새로운 XOR 연산은 이전보다 잡음에 대해서는 강하고 연산 속도의 측면에서는 한 번의 재부팅으로 수행된다. 이러한 결과는 다양한 방면에서 도움이 될 수 있을 것이다. 하나의 예시로 SHA3(Secure Hash Algorithm)을 들 수 있다. SHA3를 구성하는 대부분의 논리 연산이 XOR이다. 또한, 암호학적 Hash 함수의 눈사태 효과 때문에 동작 과정 중의 오류를 허용하면 안 되므로 낮은 오류 발생 확률을 위한 파라미터를 사용해야 한다. 이러한 상황에서 기존의 XOR 연산을 이용해 SHA3를 수행한다면 AND, OR 연산과 동일한 오류 가능성에서 수행되었을 것이다. 그러나 새로이 제시한 XOR 연산을 수행하면 1번의 재부팅만으로 잡음에 더욱 강한 연산을 수행할 수 있기 때문에 오류 발생 확률 측면에서 큰 개선이 있을 것이다. 대부분의 연산이 XOR인 SHA3의 경우 그 효과가 더욱 크게 나타날 것이다.

[0035] 이외에도 XOR 연산이 사용되는 다양한 알고리즘에도 도움이 될 것이며 XOR 연산의 비중이 크면 클수록 그 효과는 클 것이다.

[0036] 도 3은 일 실시예에 있어서, OpenFHE 구현과 실시예에서 제안된 기술을 비교한 표이고, 도 4는 일 실시예에 있어서, OpenFHE, TFHE 와 실시예에서 제안된 기술의 실패확률을 비교한 표이다.

[0037] 기존의 FHEW에서 제공하는 XOR 연산의 경우 두 가지의 방법을 제공한다. 한가지는 두 개의 AND 연산과 하나의 OR 연산으로 총 3개의 논리 연산이 사용되는 방법이다 ( $XOR(m_0, m_1) = OR(AND(m_0, \overline{m_1}), AND(\overline{m_0}, m_1))$ ). 또 다른 하나는  $2(c_1 - c_2)$ 로 암호문 연산을 진행한 뒤  $[q/8, 5q/8]$ 를  $q/8$ 로 대응하고  $[-3q/8, q/8]$ 을  $-q/8$ 로 대응하는 방법이다. FHEW는 매핑 함수가 덧셈(+), 뺄셈(-) 등과 같은 연산을 마치 논리 연산을 수행하듯 만들어주기 때

문에 논리 연산을 지원할 수 있다. 즉, 일반적으로 사용하는 덧셈과 뺄셈같은 연산이 FHEW 상에서는 논리 연산처럼 수행되는 것이다. 이와 같이,  $c_1 - c_2$ ,  $c_1 + c_2$  등의 수식은 (FHEW 상에서 정의되는) 논리 연산을 수행하기 위해 정의된 연산이다. 예를 들어  $2(c_1 - c_2)$  는  $c_1 \text{ XOR } c_2$  와 동일한 것을 의미한다. 또한, 대응한다는 의미는 해당값(사용자의 정보, 메시지)이 0인지 1인지 판별한다는 의미이다. 이는 앞서 정의된 매핑 함수가 수행하는 역할이다. 매핑 함수가 마치 단순한 덧셈, 뺄셈과 같은 연산을 논리 연산처럼 수행하도록 만들어주기 때문에 FHEW가 논리 연산을 수행할 수 있게 된다. 우선, 간단하게  $q/8$ 을 bit 1,  $-q/8$ 을 bit 0으로 생각하여 설명하기로 한다. 또한, 영역을 나누어 비트(bit)가 1인지 0인지 구분하게 된다( $q/8$  인지  $-q/8$  인지). 이는 FHEW 상에서 암호문은 난수처럼 보이기 때문이다. 앞서 설명된 수학적식(ACC)에서  $f(b - i)$ 라는 수식을 확인할 수 있으며, 여기서,  $i$  는 0부터  $q/2-1$ 까지 늘어나는 것을 확인할 수 있다. 즉,  $i$ 가 복호화에 필요한 값인  $\langle a, s \rangle$ 에 한번은 근사한 값으로 가게 된다. 이때, 이 값이 몇 번째 계수에 위치해 있는지는 사용자만 알고 있을 것이며(사용자만  $\langle a, s \rangle$  값을 알고 있음) 해당 계수에 있는 값은 대략  $f(m * q/4 + e)$ 로 메시지와 근사한 값이 된다. 여기서,  $m$  이 0인지 1인지 판별하는 함수가  $f$ 인 것이며, 잡음으로 인해  $m$ 의 값을 정확히 판단할 수 없어 영역을 나누어 특정 영역은  $1(q/8)$ 이고 또 다른 영역은  $0(-q/8)$ 으로 판별하는 것이다. 이렇게 값을 추론해내는 과정을 대응한다고 표현하였다. 추가적으로,  $q/8$ 과  $-q/8$ 은 스케일링 팩터와 직접적인 관련이 있는 값이며, 실시예에서 제안된 리스케일링 과정은 해당 값을  $q/4$ ,  $-q/4$  (동일하게 1과 0을 의미)로 변경하는 과정이다. 전자의 경우에는 논리 연산의 오류가 발생할 확률이 비교적 적지만 3개의 논리 연산을 사용하기 때문에 3번의 재부팅을 진행해야 하므로 시간 소모가 비교적 많다. 후자의 경우 한 번의 재부팅을 사용하여 시간상으로 빠르지만,  $2(c_1 - c_2)$ 의 연산을 수행하기 때문에 암호문에 포함되어 있는 잡음이 2배 증가해 연산의 오류가 발생할 확률을 증가시킨다. 또한, TFHE library에서 제공하는 XOR 연산의 경우  $2(c_1 + c_2)$ 로 연산을 수행한 뒤  $[-q/4, q/4]$ 을  $-q/8$ 로,  $[q/4, 3q/4]$ 를  $q/8$ 로 대응하는 방법을 채택하여 비교적 오류를 발생시키는 확률을 낮추었다.

[0039] 이에 반해, 연산 시스템은 새로운 XOR 연산 방법을 제공할 수 있다. 새로운 XOR 연산 방법은 단 한 번의 재부팅을 필요로 하며 오류가 발생할 확률 또한 기존의 AND, OR, XOR 연산보다 오히려 더 안정적이다. 연산 시스템은 스케일링 팩터를 자유롭게 이동하며 연산을 수행하는 방법을 소개하고자 한다. 매핑 함수  $f$ 의 값을 재부팅에 유리하도록 설정하는 방법이다. XOR 연산을 수행하기 위해, XOR gate의 직전 재부팅에서  $f$ 를 이용하여 기존의  $-q/8$ 근처의 값을  $-q/4$ 로,  $q/8$  근처의 값을  $q/4$ 로 대응하는 것이다. 이를 통해 기존의 XOR 연산에 비해 잡음을 절반으로 줄일 수 있으므로 오류 발생 확률을 크게 낮춘다(도 4 참고). 스케일링 팩터를 자유롭게 이동하기 위해서는 이후에 수행할 논리 연산이 무엇인지 알아야 한다는 조건이 따르지만, 이는 컴파일러의 수준에서 쉽게 해결될 수 있다.

[0040] 다음으로 종래 기술과 실시예에서 제안된 기술을 비교하기로 한다. 종래 AND, OR 연산의 경우 두 연산 모두  $c_1 + c_2$ 로 암호문 연산을 진행한다. FHEW는 매핑 함수가 덧셈(+), 뺄셈(-) 등과 같은 연산을 마치 논리연산을 수행하듯 만들어주기 때문에 논리 연산을 지원할 수 있다. 즉, 일반적으로 사용하는 덧셈과 뺄셈같은 연산이 FHEW 상에서는 논리 연산처럼 수행되는 것이다. 여기서, 나와있는  $c_1 + c_2$ 는 정말 단순히 두 값을 더하는 것이며 매핑 함수에 의해 나오게 되는 결과값이 마치 논리 연산을 수행한 것과 동일한 결과를 갖게 된다. AND의 경우  $[3q/8, 7q/8]$ 를  $q/8$ 로,  $[-q/8, 3q/8]$ 을  $-q/8$ 로 대응한다. OR의 경우  $[q/8, 5q/8]$ 를  $q/8$ 로,  $[-3q/8, q/8]$ 을  $-q/8$ 로 대응한다. XOR 연산과는 달리 암호문 간의 연산을  $c_1 + c_2$ 로만 수행하기 때문에 XOR 연산에 비해 잡음이 적으며, 오류가 발생할 확률 또한 낮다.

[0041] 오류가 발생할 확률이 적은 이유는 다음과 같다. 기존의 AND, OR 연산의 경우  $c_1 + c_2$ 로 암호문 간의 연산을 수행함에 따라 나올 수 있는 메시지는  $(0, q/4, q/2)$ 이며 이 값에 작은 잡음값이 섞여 있다. 이때, AND 연산을 수행하기 위해 지정했던 대응값은  $[3q/8, 7q/8]$ 과  $[-q/8, 3q/8]$ 이다. 이때, 두 암호문의 연산 값이 0과  $q/2$ 일 때 대응값인  $-q/8$ 과  $3q/8$ 에서  $q/8$ 만큼의 차이를 갖고 있다는 것을 알 수 있다. 즉, 잡음의 크기가  $q/8$ 이상으로 차이가 나면 오류가 발생한다는 것이다. OR 연산 또한 비슷한 양상이다. 그러나 OpenFHE에서 제공하는 XOR 연산의 경우에는 암호문 간의 연산을  $2(c_1 - c_2)$ 로 수행하여 AND, OR 연산의 잡음에 비해 약 2배가량 커져 오류발생에 취약하다. TFHE library에서 제공하는 XOR 연산의 경우에는 잡음이 커지는 것에 비례해 수용할 수 있는 잡

음의 크기 또한 커져 AND, OR과 같은 실패 확률을 갖는다.

[0042] 실시예에서 제안된 XOR 연산에서는 기존에  $2(c_1 - c_2)$  혹은  $2(c_1 + c_2)$ 으로 암호문 연산을 수행했던 것과 달리  $c_1 + c_2$ 로 암호문 연산을 수행한다. 이것이 가능한 이유는 이전 재부팅 과정에서  $f$ 를 통해 스케일링 팩터를 변경하는 리스케일링(rescaling) 과정을 거쳤기 때문이다. 이 경우, 0 혹은  $q/2$ 를 기댓값으로 갖는다. 이때, 대응값은  $[q/4, 3q/4]$ 과  $[-q/4, q/4]$ 이다. 기댓값은 암호문 간의 통해 연산으로 나올 수 있는 메시지 값들의 종류이다. 즉,  $m * q/4$  혹은 리스케일링의 과정을 거쳐서 새로 정의된  $m * q/2$ 가 나타낼 수 있는 값의 종류들이다. 앞서 설명한 바와 같이, 암호문 간의 일반적인 덧셈, 뺄셈 등의 연산이 FHEW에서는 마치 논리 연산처럼 수행된다. 덧셈을 예시로 들자면 두개의 암호문을 더했을 때 메시지가 가질 수 있는 값은 3가지 종류이다.  $(0 + 0) * q/4 = 0$ ( $m$ 이 둘다 0일때),  $(0 + 1) * q/4 = q/4$ ,  $(1 + 0) * q/4 = q/4$ (둘 중 하나가 1일 경우),  $(1 + 1) * q/4 = q/2$ (둘 다 1일 경우)이다. 여기서  $q/4$ 는 스케일링 팩터를 의미한다. 여기서, 실시예에서 제안된 리스케일링 과정을 거쳐 기존 스케일링 팩터를  $q/4$ 에서  $q/2$ 로 늘린 것이다. 따라서 이 상황의 기댓값은  $(0 + 0) * q/2 = 0$ ( $m$ 이 둘다 0일때),  $(0 + 1) * q/2 = q/2$ ,  $(1 + 0) * q/2 = q/2$ (둘 중 하나가 1일 경우),  $(1 + 1) * q/2 = q = 0$ (둘 다 1일 경우)가 된다. 여기서,  $q$ 가 0이 되는 이유는 모듈러군이기 때문이다. 간단히 설명하자면,  $q$ 보다 큰 값을 가질 수 없으며 큰 값이 나올 경우 해당 값을  $q$ 로 나눈 나머지를 갖게 된다( $q/q$ 는 몫은 1 나머지는 0 이므로 0이라는 값과 동일한 값으로 취급). 대응값은 앞서 기댓값에는 잡음이 섞여있어 이를 판별하기 위해 특정 영역 안에 위치해 있으면 '0 또는 1이다'라고 판별하게 된다. 다시 말해서, 해당 영역에 있으면 0 또는 1로 대응시킨다는 것을 의미한다.

[0043] 즉, TFHE에서 제공하는 XOR 연산과 동일하게 잡음의 크기를  $q/4$ 까지 허용할 수 있다. 그러나, 암호문 간의 연산을  $c_1 + c_2$ 로 수행하여 기존보다 더욱 낮은 잡음을 갖게 되어 오류가 발생할 확률을 줄일 수 있게 된다.

[0044] 기존에 OpenFHE에서 제공하는 XOR 연산과 우리의 XOR 연산을 비교해 보면 도 3 및 도 4와 같다. XOR 연산을 10번씩 수행하는 시간을 측정한 것이며 파라미터에 따라 비교했다. 시간 측정은 chrono 라이브러리를 이용했다. 기존에 3번 수행되어야 했던 재부팅의 횟수를 1번으로 줄여 기존 수행 시간의 33% 선에서 해결할 수 있었다. 수행된 환경은 Intel(R) Core(TM) i9-11900 @ 2.50GHz, 32GB RAM, Ubuntu 22.04.이며, 컴파일러는 clang++ 14를 사용했다.

[0045] 비특허문헌 1< D. Micciancio and Y. Polyakov, "Bootstrapping in FHEW-like cryptosystems," in WAHC'21, pp 13-15. ACM, 2021>에서 소개하는 AND, OR 연산의 경우  $1 - \text{erf}(\frac{q/8}{\sqrt{2}\beta_{exp}})$ , ( $\beta_{exp}$ 는 오류의 표준편차) XOR (XOR\_FAST) 연산의 경우  $1 - \text{erf}(\frac{q/8}{2\sqrt{2}\beta_{exp}})$ 의 실패 확률을 갖는다. TFHE에서는 XOR 연산의 실패 확률을  $1 - \text{erf}(\frac{q/8}{\sqrt{2}\beta_{exp}})$ 의 수준까지 내린다. 실시예에서 제안된 XOR 연산은 이를  $1 - \text{erf}(\frac{q/4}{\sqrt{2}\beta_{exp}})$ 의 수준까지 내려 잡음에 더욱 강하다. STD128을 기준으로 기존의  $2^{-52}$ 이었던 실패 확률을 약  $2^{-199}$ 까지 내릴 수 있다.

[0046] 이상에서 설명된 장치는 하드웨어 구성요소, 소프트웨어 구성요소, 및/또는 하드웨어 구성요소 및 소프트웨어 구성요소의 조합으로 구현될 수 있다. 예를 들어, 실시예들에서 설명된 장치 및 구성요소는, 예를 들어, 프로세서, 콘트롤러, ALU(arithmetic logic unit), 디지털 신호 프로세서(digital signal processor), 마이크로컴퓨터, FPGA(field programmable gate array), PLU(programmable logic unit), 마이크로프로세서, 또는 명령(instruction)을 실행하고 응답할 수 있는 다른 어떠한 장치와 같이, 하나 이상의 범용 컴퓨터 또는 특수 목적 컴퓨터를 이용하여 구현될 수 있다. 처리 장치는 운영 체제(OS) 및 상기 운영 체제 상에서 수행되는 하나 이상의 소프트웨어 애플리케이션을 수행할 수 있다. 또한, 처리 장치는 소프트웨어의 실행에 응답하여, 데이터를 접근, 저장, 조작, 처리 및 생성할 수도 있다. 이해의 편의를 위하여, 처리 장치는 하나가 사용되는 것으로 설명된 경우도 있지만, 해당 기술분야에서 통상의 지식을 가진 자는, 처리 장치가 복수 개의 처리 요소(processing element) 및/또는 복수 유형의 처리 요소를 포함할 수 있음을 알 수 있다. 예를 들어, 처리 장치는 복수 개의 프로세서 또는 하나의 프로세서 및 하나의 콘트롤러를 포함할 수 있다. 또한, 병렬 프로세서(parallel processor)와 같은, 다른 처리 구성(processing configuration)도 가능하다.

[0047] 소프트웨어는 컴퓨터 프로그램(computer program), 코드(code), 명령(instruction), 또는 이들 중 하나 이상의 조합을 포함할 수 있으며, 원하는 대로 동작하도록 처리 장치를 구성하거나 독립적으로 또는 결합적으로

(collectively) 처리 장치를 명령할 수 있다. 소프트웨어 및/또는 데이터는, 처리 장치에 의하여 해석되거나 처리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성요소(component), 물리적 장치, 가상 장치(virtual equipment), 컴퓨터 저장 매체 또는 장치에 구체화(embody)될 수 있다. 소프트웨어는 네트워크로 연결된 컴퓨터 시스템 상에 분산되어서, 분산된 방법으로 저장되거나 실행될 수도 있다. 소프트웨어 및 데이터는 하나 이상의 컴퓨터 판독 가능 기록 매체에 저장될 수 있다.

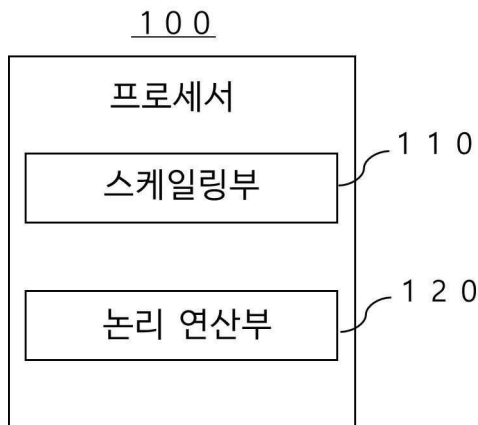
[0048] 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(magnetic media), CD-ROM, DVD와 같은 광기록 매체(optical media), 플롭티컬 디스크(floptical disk)와 같은 자기-광 매체(magneto-optical media), 및 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다.

[0049] 이상과 같이 실시예들이 비록 한정된 실시예와 도면에 의해 설명되었으나, 해당 기술분야에서 통상의 지식을 가진 자라면 상기의 기재로부터 다양한 수정 및 변형이 가능하다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다.

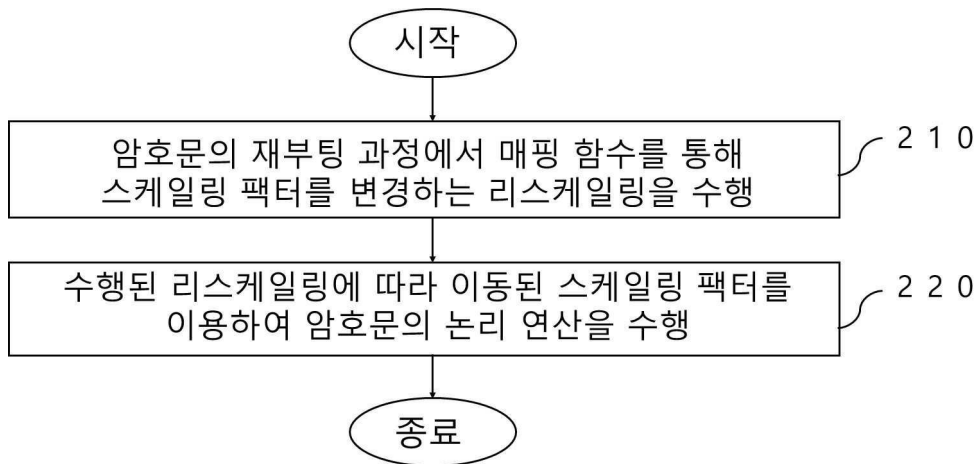
[0050] 그러므로, 다른 구현들, 다른 실시예들 및 특허청구범위와 균등한 것들도 후술하는 특허청구범위의 범위에 속한다.

## 도면

### 도면1



도면2



도면3

| Parameter Set <sup>φ</sup> | XOR[1] <sup>φ</sup>   | Proposed <sup>φ</sup> |
|----------------------------|-----------------------|-----------------------|
| STD 128 <sup>φ</sup>       | 1.81358s <sup>φ</sup> | 0.6042s <sup>φ</sup>  |
| STD 192 <sup>φ</sup>       | 5.81813s <sup>φ</sup> | 1.9416s <sup>φ</sup>  |
| STD 256 <sup>φ</sup>       | 8.6321s <sup>φ</sup>  | 2.8853s <sup>φ</sup>  |

도면4

| Parameter Set <sup>φ</sup> | XOR[1] <sup>φ</sup>    | XOR[6] <sup>φ</sup>    | Proposed <sup>φ</sup>   |
|----------------------------|------------------------|------------------------|-------------------------|
| STD 128 <sup>φ</sup>       | $2^{-14}$ <sup>φ</sup> | $2^{-52}$ <sup>φ</sup> | $2^{-199}$ <sup>φ</sup> |
| STD 192 <sup>φ</sup>       | $2^{-25}$ <sup>φ</sup> | $2^{-96}$ <sup>φ</sup> | $2^{-373}$ <sup>φ</sup> |
| STD 256 <sup>φ</sup>       | $2^{-9}$ <sup>φ</sup>  | $2^{-33}$ <sup>φ</sup> | $2^{-124}$ <sup>φ</sup> |