

(19) 대한민국특허청(KR)
(12) 공개특허공보(A)(11) 공개번호 10-2022-0096223
(43) 공개일자 2022년07월07일

(51) 국제특허분류(Int. Cl.)
H04L 9/00 (2022.01) G06F 17/16 (2006.01)
(52) CPC특허분류
H04L 9/008 (2013.01)
G06F 17/16 (2013.01)
(21) 출원번호 10-2020-0188480
(22) 출원일자 2020년12월30일
심사청구일자 2020년12월30일

(71) 출원인
고려대학교 산학협력단
서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)
(72) 발명자
윤지원
서울특별시 용산구 한강대로43길 8, 102동 1401호
홍미연
경기도 성남시 분당구 내정로 152, 130동 901호
유준수
서울특별시 송파구 백제고분로36나길 28-1, 3층
(74) 대리인
김홍석

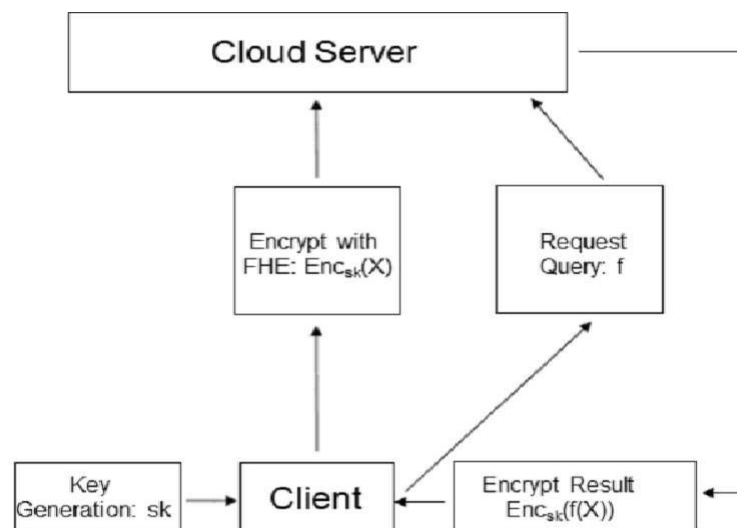
전체 청구항 수 : 총 11 항

(54) 발명의 명칭 동형암호로 암호화된 데이터 연산 방법

(57) 요약

동형암호로 암호화된 데이터 연산 방법이 개시된다. 방법은, 동형암호를 이용하여 암호화된 암호문 데이터를 클라이언트로부터 수신하고, 암호문 데이터들에서 실수 기반의 사칙연산을 이용하여 동형암호 역행렬 연산을 생성하며, 동형암호 역행렬을 클라이언트에게 전송하는 것을 포함한다.

대표도 - 도1



(52) CPC특허분류

H04L 2209/046 (2013.01)

이 발명을 지원한 국가연구개발사업

과제고유번호	1711103092
과제번호	2017-0-00545-004
부처명	과학기술정보통신부
과제관리(전문)기관명	정보통신기획평가원
연구사업명	정보보호핵심원천기술개발(R&D, 정보화)
연구과제명	암호화된 데이터상의 기계학습 라이브러리 설계 및 활용기법 연구
기 여 율	1/1
과제수행기관명	고려대학교산학협력단
연구기간	2020.01.01 ~ 2020.12.31

명세서

청구범위

청구항 1

동형암호를 이용하여 암호화된 암호문 데이터를 클라이언트로부터 수신하는 단계;

상기 암호문 데이터들에서 실수 기반의 사칙연산을 이용하여 동형암호 역행렬 연산을 생성하는 단계; 및

상기 동호암호 역행렬을 상기 클라이언트에게 전송하는 단계를 포함하는 암호화된 데이터의 연산 방법.

청구항 2

제1항에 있어서,

상기 암호문 데이터들 각각은 비트 단위 동형암호 스키마를 사용하는 암호화된 데이터의 연산 방법.

청구항 3

제1항에 있어서,

상기 암호문 데이터들 각각은 실수로 표현하는 단계를 더 포함하되,

인덱스 0부터 $\frac{r}{2}-2$ 까지의 $(\frac{r}{2}-1)$ 만큼을 상기 실수 데이터의 소수 부분으로 하고, 인덱스 $\frac{r}{2}-1$ 부터 $r-1$ 까지 $\frac{r}{2}$ 만큼을 상기 실수 데이터의 정수 부분으로 하며, 여기에서, r은 비트의 길이를 의미하는 암호화된 데이터의 연산 방법.

청구항 4

제1항에 있어서,

하기의 알고리즘을 이용하여 상기 암호문 데이터들(ct.a, ct.b)을 동형암호의 대소비교하는 단계를 더 포함하는 암호화된 데이터 연산 방법.

Algorithm HomCompL(ct.a, ct.b): Homomorphic compare large

```

1: ct.s1 ← ENC[0]
2: for i = 0 to r - 1 do
3:   ct.s2 ← bootsXNOR(ct.a[i], ct.b[i])
4:   ct.s1 ← bootsMUX(ct.s2, ct.s1[i], ct.a[i])
5: end for
6: Return ct.s1

```

청구항 5

제1항에 있어서,

하기의 알고리즘을 이용하여 상기 암호문 데이터들(ct.a, ct.b)을 동형암호의 대소비교하는 단계를 더 포함하는 암호화된 데이터 연산 방법.

Algorithm HomCompS(ct.a, ct.b): Homomorphic compare small

```

1: ct.s1 ← ENC[0]
2: for i = 0 to r - 1 do
3:   ct.s2 ← bootsXNOR(ct.a[i], ct.b[i])
4:   ct.s1 ← bootsMUX(ct.s2, ct.s1[i], ct.b[i])
5: end for
6: Return ct.s1

```

청구항 6

제1항에 있어서,

하기의 알고리즘을 이용하여 상기 암호문 데이터들(ct.a, ct.b) 간 동형암호 덧셈을 수행하는 단계를 더 포함하는 암호화된 데이터 연산 방법.

Algorithm HomAdd(ct.a, ct.b, r): Homomorphic addition

```

1: ct.a[0] ← bootsCONSTANT[0]
2: for i = 0 to r - 1 do
3:   ct.m0 ← bootsXOR(ct.a[i], ct.b[i])
4:   ct.m1 ← bootsAND(ct.m0, ct.b[i])
5:   ct.c[i] ← bootsXOR(ct.m0, ct.t[i])
6:   ct.m2 ← bootsAND(ct.m0, ct.t[i])
7:   ct.t[i + 1] ← bootsOR(ct.m2, ct.m1)
8: end for
9: ct.m3 ← bootsXOR(ct.a[r - 1], ct.b[r - 1])
10: ct.c[r - 1] ← bootsXOR(ct.m0, ct.t[r - 1])
11: Return ct.c

```

청구항 7

제1항에 있어서,

하기의 알고리즘을 이용하여 상기 암호문 데이터들(ct.a, ct.b) 간 동형암호 뺄셈을 수행하는 단계를 더 포함하는 암호화된 데이터 연산 방법.

Algorithm HomSubt(ct.a, ct.b, r): Homomorphic subtraction

```

1: ct.b ← HomTwosComp(ct.b)
2: c[0] ← Enc[0]
3: for i = 0 to r - 1 do
4:   ct.m0 ← bootsXOR(ct.a[i], ct.b[i])
5:   ct.m1 ← bootsAND(ct.m1, ct.m0, ct.b[i])
6:   ct.c[i] ← bootsXOR(ct.m0, ct.t[i])
7:   ct.m0 ← bootsAND(ct.m0, ct.t[i])
8:   ct.c[i + 1] ← bootsOR(ct.m0, ct.m1)
9: end for
10: ct.m0 ← bootsXOR(ct.a[r - 1], ct.b[r - 1])
11: ct.c[r - 1] ← bootsXOR(ct.m0, ct.t[r - 1])
12: Return ct.c

```

청구항 8

제1항에 있어서,

하기의 알고리즘을 이용하여 상기 암호문 데이터들(ct.a, ct.b) 간 동형암호 곱셈을 수행하는 단계를 더 포함하는 암호화된 데이터 연산 방법.

Algorithm HomMulti(ct.a, ct.b, r): Homomorphic multiplication

```

1: ct.â ← HomAbs(ct.a),   ct.b̂ ← HomAbs(ct.b)
2: for i = 1 to r do
3:   ct.d[i] ← bootsAND(ct.â[i], ct.b̂[i])
4: end for
5: for i = 1 to r - 2 do
6:   if i <  $\frac{r}{2} - 1$  then
7:     ct.e ← HomRShift(ct.â,  $\frac{r}{2} - i - 1$ )
8:     for j = 1 to r do
9:       ct.f[j] ← bootsAND(ct.d[j], ct.b̂[i + 1])
10:    end for
11:    ct.d ← HomAdd(ct.d[j], ct.f[i + 1])
12:   else if i =  $\frac{r}{2} - 1$  then
13:     for j = 1 to r do
14:       ct.f[j] ← bootsAND(ct.â[j], ct.b̂[i + 1])
15:     end for
16:     ct.d ← HomAdd(ct.d, ct.f)
17:   else
18:     ct.e ← HomRShift(ct.â, i - ( $\frac{r}{2} + 1$ ))
19:     for j = 1 to r do
20:       ct.f[j] ← bootsAND(ct.e[j], ct.b̂[i + 1])
21:     end for
22:     ct.d ← HomAdd(ct.d, ct.f)
23:   end if
24: end for
25: ct.f ← HomTwosComp(ct.d)
26: ct.g ← bootsXOR(ct.a[r - 1], ct.b[r - 1])
27: ct.h ← bootsNOT(ct.g)
28: for i = 1 to r do
29:   ct.o ← bootsAND(ct.d, ct.h)
30:   ct.p ← bootsAND(ct.f, ct.g)
31: end for
32: ct.q ← HomAdd(ct.o, ct.p)
33: Return ct.q

```

청구항 9

제1항에 있어서,

하기의 알고리즘을 이용하여 상기 암호문 데이터들(ct.a, ct.b) 간 동형암호 나눗셈을 수행하는 단계를 더 포함하는 암호화된 데이터 연산 방법.

Algorithm HomDiv(ct.q, ct.m, r): Homomorphic division

```

1: ct.q̂ ← HomAbs(ct.q),   ct.m̂ ← HomAbs(ct.m)
2: repeat
3:   ct.c ← (ct.q || ct.a)
4:   ct.c ← HomRShift(ct.c, 1)
5:   ct.a ← (ct.c[r], ct.c[r + 1], ..., ct.c[2r - 1])
6:   ct.q ← (ct.c[0], ct.c[1], ..., ct.c[r - 1])
7:   ct.d ← HomCompS(ct.a, ct.m)
8:   ct.q̂[0] ← bootsNOT(ct.d)
9:   ct.aM ← HomSubt(ct.a, ct.m)
10:  ct.aL ← bootsAND(ct.q̂[0], ct.aM)
11:  ct.aR ← bootsAND(ct.d, ct.a)
12:  ct.a ← bootsAND(ct.aL, ct.aR)
13: until All fractional bits of ct.q̂ are extracted( $r + \frac{r}{2} - 1$  times)
14: ct.f ← HomTwosComp(ct.q̂)
15: ct.g ← bootsXOR(ct.q[r - 1], ct.m[r - 1])
16: ct.h ← bootsNOT(ct.g)
17: for i = 1 to r do
18:   ct.qL ← bootsAND(ct.h, ct.q̂)
19:   ct.qR ← bootsAND(ct.g, ct.f)
20: end for
21: ct.q ← HomAdd(ct.qL, ct.qR)
22: Return ct.q

```

청구항 10

제1항에 있어서,

하기의 알고리즘을 이용하여 상기 암호문 데이터들(ct.A, ct.B) 간 동형암호 행렬곱셈을 수행하는 단계를 더 포함하는 암호화된 데이터 연산 방법.

Algorithm HomMatMulti(ct.A, ct.B)

```

1: ct.C  $\leftarrow$  Enc[0]
2: for  $i = 1$  to  $n$  do
3:   for  $j = 1$  to  $n$  do
4:     for  $k = 1$  to  $n$  do
5:       ct.cij  $\leftarrow$  HomMulti(ct.Aik, ct.Bkj)
6:       ct.C  $\leftarrow$  HomAdd(ct.C, ct.cij)
7:     end for
8:   end for
9: end for
10: Return ct.C

```

청구항 11

제1항에 있어서,

하기의 알고리즘을 이용하여 상기 암호문 데이터들을 가우스-조던 역행렬 연산을 수행하는 단계를 더 포함하는 암호화된 데이터 연산 방법.

Algorithm ct.W⁻¹=HomGJ(ct.W): Homomorphic Matrix Inversion using Gauss-Jordan Elimination

Require: Encrypted Matrix ct.W_{nxn}

Ensure: Encrypted Inverse Matrix ct.W_{nxn}⁻¹

```

1: for each row  $k$  do
2:   ct.i*  $\leftarrow$  argmax $_{k \leq i \leq n}$  HomAbs(ct.Wik)
3:   if ct.Wi*,k=0 then
4:     Matrix is not invertible
5:   end if
6:   Swap rows  $k$  and  $i^*$   $\leftarrow$  HomMux(ct.k, ct.i*, ct.temp)
7:   for each row  $j$  below  $k$  (i.e.  $j = k + 1, \dots, n$ ) do
8:     ct. $\tau$   $\leftarrow$  HomDiv(ct.Wjk, ct.Wkk)
9:     ct.Wk  $\leftarrow$  HomMulti(ct. $\tau$ , ct.Wk)
10:    ct.Wj  $\leftarrow$  HomSubt(ct.Wj, ct.Wk)
11:   end for
12: end for
13: for each row  $k = n, \dots, 1$  (i.e. in reverse) do
14:   ct.Wk  $\leftarrow$  HomDiv(ct.Wk, ct.Wkk)
15:   for each row  $j$  above  $k$  (i.e.  $j = k - 1, \dots, 1$ ) do
16:     ct. $\tau$   $\leftarrow$  HomDiv(ct.Wjk, ct.Wkk)
17:     ct.Wk  $\leftarrow$  HomMulti(ct. $\tau$ , ct.Wk)
18:     ct.Wj  $\leftarrow$  HomSubt(ct.Wj, ct.Wk)
19:   end for
20: end for
21: Return ct.W-1

```

발명의 설명

기술 분야

[0001] 본 발명은 동형암호로 암호화된 데이터 연산 방법에 관한 것이다.

배경 기술

[0002] 데이터 프라이버시를 보장하는 가장 좋은 방법 중 하나는 데이터를 클라우드 서버로 보내기 전에 암호화하는 것이다. 그러나, 클라우드 서버가 클라이언트의 요청에 응답하기 위해 암호화된 데이터를 계산하는 문제는 여전히 존재한다. 클라이언트는 계산을 실행하기 전에 데이터를 해독하기 위해 서버에 비밀 키를 제공해야 하므로 기밀성 위반 또는 개인 정보 침해로 이어질 수 있다. 동형암호화는 암호화된 데이터를 해독하지 않고 작업을 구현할 수 있으므로 이러한 문제에 적합한 솔루션이다. 머신 러닝은 프로그래밍 없이도 경험을 통해 자동으로 학습하고 개선할 수 있는 기능을 시스템에 제공한다. 동형암호화는 개인 정보를 보호하는데 사용할 수 있으며, 의료 산업에서 환자의 질병을 예측하거나 은행 산업에서 개인의 신용도를 평가하기 위한 모델을 만드는 동시에 개인 정보 유출을 방지하는 학습이 될 수도 있다.

발명의 내용

해결하려는 과제

[0003] 본원 발명이 해결하고자 하는 과제는 클라이언트의 개인 정보를 보호하면서 동형암호로 암호화된 데이터를 연산하는 방법에 관한 것이다.

과제의 해결 수단

[0004] 해결하고자 하는 과제를 달성하기 위하여 본 발명의 실시 예들에 따른 동형암호로 암호화된 데이터 연산 방법은, 동형암호를 이용하여 암호화된 암호문 데이터를 클라이언트로부터 수신하는 단계, 상기 암호문 데이터들에서 실수 기반의 사칙연산을 이용하여 동형암호 역행렬 연산을 생성하는 단계 및 상기 동호암호 역행렬을 상기 클라이언트에게 전송하는 단계를 포함한다.

발명의 효과

[0005] 본 발명의 실시 예들에 따르면, 동형암호로 암호화된 데이터를 별도의 복호화 없이, 암호화된 상태에서 데이터를 처리할 수 있다. 따라서, 클라이언트의 데이터가 노출될 위험 없이, 동형암호는 데이터의 프라이버시가 강하게 요구되는 환경에서 기계학습, 통계 등의 분야에 사용할 수 있다. 동형암호를 이용하면 기밀성을 보장하면서 데이터를 활용하여 분석이 가능하다.

도면의 간단한 설명

[0006] 도 1은 클라이언트와 클라우드 간의 암호화된 데이터 처리를 위한 프로토콜이다.

도 2는 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터 연산 방법을 이용하여 소수 표현을 나타내는 일 예이다.

도 3은 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터 연산 방법을 이용하여 대소비교하는 일 알고리즘이다.

도 4는 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터 연산 방법을 이용하여 대소비교하는 다른 알고리즘이다.

도 5는 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 덧셈 알고리즘이다.

도 6은 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 뺄셈을 나타내는 알고리즘이다.

도 7은 평문에서의 이진 곱셈에 대한 예시이다.

도 8은 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 곱셈의 알고리즘이다.

도 9는 평문 이진 나눗셈 순서도이다.

도 10은 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 나눗셈의 알고리즘이다.

도 11은 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 행렬 곱셈의 알고리즘이다.

도 12는 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 가우스 조던 역행렬 알고리즘을 나타낸다.

도 13은 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 가우스 조던 역행렬 알고리즘을 클라우드에서 수행하는 방법을 나타낸 흐름도이다.

발명을 실시하기 위한 구체적인 내용

[0007] 본 발명의 구성 및 효과를 충분히 이해하기 위하여, 첨부한 도면을 참조하여 본 발명의 바람직한 실시 예들을 설명한다. 그러나 본 발명은, 이하에서 개시되는 실시 예들에 한정되는 것이 아니라, 여러 가지 형태로 구현될 수 있고 다양한 변경을 가할 수 있다.

[0008] 또한, 본 발명의 실시 예들에서 사용되는 용어들은 다르게 정의되지 않는 한, 해당 기술 분야에서 통상의 지식을 가진 자에게 통상적으로 알려진 의미로 해석될 수 있다.

- [0009] 이하에서, 본 발명의 실시예들에 따른 동형암호로 암호화된 데이터 연산 방법을 설명하기로 한다. 이하에서 설명되는 동형암호는 완전동형암호 완전동형암호(fully homomorphic encryption)이다.
- [0010] 도 1은 클라이언트와 클라우드 간의 암호화된 데이터 처리를 위한 프로토콜이다. 도 1의 프로토콜을 이용하여 본 기법을 수행하면, 클라우드 암호화된 결과값(역행렬)을 클라이언트에게 반환한다.
- [0011] 도 1을 참조하면, 클라이언트는 쿼리(f), 사용자의 암호문 데이터($Enc_{sk}(X)$)를 클라우드에게 전달한다. 여기서 $Enc_{sk}(X)$ 는 클라이언트 데이터 X 를 클라이언트의 개인키(sk)로 암호화하는 것을 의미하고, f 는 본 발명에서 역행렬 의뢰를 의미한다.
- [0012] 다음으로, 클라우드는 클라이언트의 개인키로부터 파생되는 클라우드 키($cloud\ key$)와 역행렬을 수행하는 알고리즘(f_{HE})을 바탕으로, 사용자로부터 받은 암호화된 데이터($Enc_{sk}(X)$)를 처리한다. 이어서, 행렬 연산이 완료된 $Enc_{sk}(f(X))$ 를 클라이언트에게 반환하면, 클라이언트는 이를 복호화하여 결과값(역행렬)을 얻게 된다.
- [0013] 일 실시예에 따르면, 비트 단위 동형암호 스키마를 사용하여, 평문에서 실시하는 역행렬 연산과 정확도에서 거의 차이가 나지 않는다. 일반적으로, 대부분 동형암호 기법에서는 정수 기반을 바탕으로 동형암호 성질 두 가지(덧셈과 곱셈)만 이용 가능하기 때문에 오차가 생기게 된다. 하지만, 본 발명의 일 실시예에서는 비트 단위를 동형암호 스키마를 이용하여 새롭게 사칙연산과 행렬 곱셈, 역행렬을 실시할 수 있다. 이에 대한 설명은 하기에서 상세하게 하기로 한다.
- [0014] 본 발명은 클라이언트의 개인 정보를 보호하면서 암호화된 데이터 상에서의 역행렬을 연산하는 기술을 제시한다. 결과값은 앞서 언급한 것과 같이 사용자에게 역행렬 값을 반환한다.
- [0016] 이하에서, 본 발명의 실시예들에서 사용되는 기호 및 동형암호 함수를 정리한다.
- [0017] (1) 기호 정의
- [0018] X 는 데이터 행렬을, $ct.x$ 는 벡터를, x 는 스칼라를 의미한다. r 은 벡터(배열)의 길이(length)로 나타내며, 각각의 인덱스에 해당하는 요소들은 $ct.x[i]$ 로 i 는 0부터 r 사이의 값을 갖는다. 또한, Enc 는 암호화를 의미하고 Dec 는 복호화를 의미한다. 용어 *bitwise*는 특정 연산을 각 배열의 요소들 모두에게 각각 적용하는 것을 의미한다. 마지막으로, *boot*는 동형암호 게이트들과 일반 게이트들과의 차이를 나타내기 위해 표시한다. 일반적으로 완전동형암호 게이트는 연산 이후 잡음(noise)을 제거하기 위한 부트스트래핑(bootstrapping)이 필요하다.
- [0019] 예컨대, $ct.a[0] \leftarrow boot_{XOR}(ct.b[0], ct.c[0])$ 가 의미하는 것은 두 입력 암호비트 $b[0], c[0]$ 에 대해서 XOR 연산을 수행하고, 연산 도중 발생하는 잡음이 bootstrapping에 의해 제거되어 반환되는 암호비트 값을 $ct.a$ 배열의 0번째 요소에 저장하는 것을 의미한다.
- [0020] (2) 동형암호의 함수
- [0021] 표 1은 동형암호 가우스-조던 역행렬 연산에 필요한 동형암호 함수 테이블이다. 표 1을 참조하면, 첫 번째로 HomTwosComp($ct.a$) 함수는 동형암호 2의 보수 연산으로 입력 암호문 a 에 대해 2의 보수를 취한 값을 반환하는 알고리즘이다. HomAbs(a) 연산은 암호문 a 를 받아 이에 대해 절대값을 결과값으로 반환하는 함수이다. HomMux($ct.s, ct.a, ct.b$) 연산은 평문에서의 MUX 게이트와 같은 역할로, 입력 암호문 $ct.s$ 이 1이면 $ct.a$ 값을, 그렇지 않으면 $ct.b$ 값을 반환하는 알고리즘이다. HomRShift($ct.x, l$)는 입력 암호문 $ct.x$ 에 대해서 오른쪽으로 l 만큼 이동연산을 수행한 결과값을 반환해주는 함수이다. 마찬가지로 HomLShift($ct.x, l$)는 왼쪽으로 l 만큼 이동연산을 수행한 결과값을 반환해주는 함수이다.

표 1

번호	분류	동형암호(함수)	기호
1	-	2의 보수	HomTwosComp

2		절대값	HomAbs
3		멀티플렉서	HomMux
4	이동연산	왼쪽 이동연산	HomRShift
5		오른쪽 이동연산	HomLShift
6	대소비교	크다(>)	HomCompB
7		작다(<)	HomCompS
8	사칙연산	덧셈	HomAdd
9		뺄셈	HomSubt
10		곱셈	HomMulti
11		나눗셈	HomDiv
12	행렬연산	역행렬	HomGJ

[0023] 표 2의 대소비교, 사칙연산, 및 행렬 연산을 후속하여 설명된다.

[0024] 동형암호 실수 표현 및 기초 연산

[0025] (1) 암호화된 데이터의 실수 표현

[0026] 기존의 정수 기반 동형암호 스키마를 이용한 방법은 실수 데이터에 대해 특정값을 곱해줘 정수로 만들어 준 후 암호 알고리즘을 진행한다. 이를 통해 오차가 발생할 수 있다.

[0027] 그러나, 본 발명에서는 비트 단위 기반 동형암호 스키마를 사용하여, 실수 유효 숫자 범위를 유동적으로 정할 수 있다. 도 2는 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터 연산 방법을 이용하여 소수 표현을 나타내는 일 예이다. 도 2에서는 실수 3.25를 8비트 고정 소수점 표현의 예를 보여준다. 도 2를 참조하면, 먼저 암호문 배열의 인덱스는 왼쪽이 최상위 비트가 되게 한다. 일반적으로 입력한 비트의 길이를 r , 암호문의 배열

을 $ct.a$, i 번째 배열의 성분을 $ct.a[i]$ 라고 두면, 인덱스 0부터 $\frac{r}{2}-2$ 까지의 $(\frac{r}{2}-1)$ 만큼을 실수 데이터의 소수 부분으로 두고, 인덱스 $\frac{r}{2}-1$ 부터 $r-1$ 까지 $\frac{r}{2}$ 만큼을 실수 데이터의 정수 부분으로 둔다. 마지막, 최상위 비트, r 번째 비트를 부호 비트로 두고 실수를 표현한다.

[0028] 이하에서는, 암호화된 데이터 간에 복호화 없이, 대소비교 및 사칙 연산에 대하여 살펴본다.

[0029] (2) 동형암호 대소비교

[0030] 암호문 간에 대소비교 알고리즘을 두 가지로 나누어 구축한다. 첫 번째는 두 값을 비교하여 먼저 입력한 값이 큰 값을 알리는 HE.CompareLarge 함수, 두 번째는 먼저 입력한 값이 작은 값을 알리는 HE.CompareSmall 함수로 나뉜다.

[0031] 도 3은 본 발명의 일 실시예에 따른 방법을 이용하여 대소비교하는 일 알고리즘이다.

[0032] 도 3을 참조하면, 암호문 $ct.a$ 와 $ct.b$ 를 동형비교하여 $ct.a$ 가 $ct.b$ 보다 크면 $Enc(1)$ 을 반환하고, 작으면 $Enc(0)$ 을 반환하는 알고리즘이다. 즉, $Dec(ct.a) > Dec(ct.b)$ 이면 $Enc(1)$ 반대의 경우는 $Enc(0)$ 이다.

[0033] 도 4는 본 발명의 일 실시예에 따른 방법을 이용하여 대소비교하는 다른 알고리즘이다.

[0034] 도 4를 참조하면, 암호문 $ct.a$ 와 $ct.b$ 를 비교하여 $ct.a$ 가 $ct.b$ 보다 작으면 $Enc(1)$ 을 반환하고, 크면 $Enc(0)$ 을 반환하는 알고리즘이다. 즉, $ct.a < ct.b$ 이면 $Enc(1)$, 반대의 경우는 $Enc(0)$ 이다.

[0035] (3) 동형암호 덧셈

[0036] 평문 상에서는 전가산기(full-adder)를 통해 덧셈을 구성할 수 있다. 마찬가지로, 이를 활용하면 암호문 간의 덧셈도 전가산기의 게이트와 같은 원리로 설계할 수 있다. 도 5는 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 덧셈 알고리즘이다. 도 5의 알고리즘을 통해 동형암호 덧셈 결과값이 반환된다. $bootAND$, $bootOR$, $bootXOR$ 의 게이트들은 두 입력값에 대해서 연산 도중 발생한 잡음(noise)을 감소시키는 부트스트래핑

(bootstrapping) 과정을 포함하는 게이트들이며, 평문에서 사용하는 AND, OR, XOR과 같은 역할을 한다.

[0037] (4) 동형암호 뺄셈

[0038] 도 6은 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 뺄셈을 나타내는 알고리즘이다. 뺄셈은 도 5의 동형암호 덧셈, 즉, $ct.a + ct.b$ 에서 $ct.b$ 를 2의 보수로 바꿔주면 간단하게 얻을 수 있다. 즉, 입력값 $ct.a, ct.b$ 중에서 $ct.b$ 에 HomTwosComp(2의 보수)를 취해주면(첫 번째 줄), 덧셈 게이트를 이용해 뺄셈을 진행할 수 있다. 즉, $ct.a + \text{HomTwosComp}(ct.b) = ct.a - ct.b$ 로 요약할 수 있다.

[0039] (5) 동형암호 곱셈

[0040] 바이너리 간의 곱셈 방식에는 다양한 기법이 있다. 본 발명에서는 평문 상에서 존재하는 여러 곱셈 방식 중에서 이진 곱셈 방식을 이용한다. 이 방법의 장점은 직관적으로 이해할 수 있다는 것이다. 일상생활 안에서 곱셈을 하는 방식과 마찬가지로 행해지기 때문이다.

[0041] 도 7은 평문에서의 이진 곱셈에 대한 예시이다. 도 7을 참조하면, 위 숫자 7 ($= 0111_{(2)}$)과 아래 숫자 5 ($= 0101_{(2)}$)의 곱 결과로 35 ($= 100011_{(2)}$)가 나옴을 보여준다. 위 이진 숫자 $0111_{(2)}$ 가 아래 이진 숫자 $0101_{(2)}$ 의 한 비트마다 곱해지면서 나오는 결과들(r_1, r_2, r_3)의 합이 최종적인 결과가 된다.

[0042] 이와 마찬가지로, 암호문 간의 곱셈도 비슷한 방식을 따른다. 다만, 곱셈을 하는 두 입력값의 부호를 고려해줘야 하는 문제와 마지막으로 한정된 비트 수만을 추출해서 결과값으로 반환하는 과정이 더 필요하게 된다.

[0043] 도 8은 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 곱셈의 알고리즘이다. 도 8을 참조하면, 동형암호 스키마를 이용하여 두 암호문 간에 곱셈 결과값을 반환하는 알고리즘을 나타낸다. 도 8에서, 25-32 번째 줄이 암호문 간에 부호를 모르기 때문에 해결하기 위한 과정이다. 이를 위해, 두 암호문 $ct.a, ct.b$ 의 부호 비트 $r-1$ 번째 비트들 간에 $boot \oplus XOR$ 을 해준다. 두 입력값이 같은 부호라면, $boot \oplus XOR$ 결과로 $Enc(0)$ 이 나오고, 반대 부호라면 $Enc(1)$ 이 나오게 된다. 이를 활용하여, 두 부호가 만약 같은 부호라면 $Enc(0)$ 이 나오기 때문에, 이것에 $boot \oplus NOT$ 을 취하게 되면 $Enc(1)$ 이 된다. 이에 대해 양의 곱셈 결과 과정으로 나온 $ct.d$ 에 대해 비트마다 $boot \oplus AND$ 를 적용해주면 양의 곱셈 결과가 나오게 된다.

[0044] 마찬가지로, 부호가 반대일 경우는 각 부호 비트의 $boot \oplus XOR$ 결과가 1이 된다. 양의 곱셈 결과값에 반대 부호를 취해 나온 결과값, 즉, 음의 값에 대해 비트마다 $boot \oplus AND$ 를 적용해주게 되면 음의 결과값이 나오게 된다. 이 두 가지 결과에 대해서 더해지면 최종적으로 부호를 고려한 암호문 간 곱셈이 된다.

[0045] (6) 동형암호 나눗셈

[0046] 동형암호 나눗셈은 평문에서 진행되는 도 9의 순서도와 유사하다. 도 9는 평문 이진 나눗셈 순서도이다. 도 9를 참조하면, 먼저, Q와 M을 각각 배열의 길이가 $|$ 인 피제수와 제수, 즉 Q를 M으로 나눈다고 가정하자. 또한, A라는 길이가 $|$ 이고 모든 요소가 0인 0 벡터를 만든다.

[0047] $|$ 이 0이 될 때까지 다음을 반복한다. 우선, 두 배열, Q와 A를 합치고($Q \oplus A$), 1비트 오른쪽으로 이동연산을 진행한다. 길이가 $2|$ 인 $Q \oplus A$ 에서 왼쪽 $|$ 비트를 Q라 하고, 오른쪽 $|$ 비트만큼을 A라 저장한다. A를 M과 대소비교를 하여, A가 M보다 작으면 $Q[0]$ 에 0을, 반대의 경우는 1을 저장하고 A 배열에 $A - M$ 의 값을 저장한다. 그리고, $|$ 의 값을 1만큼 빼준다.

[0048] 그리고, Q에 저장된 값이 Q를 M으로 나눈 몫이고 이를 반환한다.

[0049] 도 10은 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 나눗셈의 알고리즘이다. 도 10을 참조하면, 암호문은 값을 알 수 없어서 평문에서처럼 단순 비교 또는 if 가정을 진행할 수 없다. 도 10에서의 $ct.q$, $ct.m$, $ct.a$ 는 도 9에서의 Q, M, A에 해당하고 암호문이다. 우선으로, 단순 비교 과정은 위에서 언급한 동형암

호 대소비교 함수를 통해 해결할 수 있다(도 10의 7번째 줄). 두 번째 if 가정은 $\text{HomCompS}(ct.a, ct.m)$ 의 값의 반환이 $Enc(0)$ 또는 $Enc(1)$ 인 것에 착안하여, $ct.a < ct.m$ 이면 $Enc(1)$ 이고 이 값에 $boot\text{NOT}$ 을 취한 값(d)은 도 9에서 $Q[0]$ 값이 된다. 마찬가지로, $ct.a \geq ct.m$ 이면 $\text{HomCompS}(ct.a, ct.m)$ 의 값은 $Enc(0)$ 이고 이는 도 9에서 $Q[0]$ 값이 된다. $ct.a > ct.m$ 인 상황에서 $Enc(0)$ 에 $boot\text{NOT}$ 을 취한 값, 즉 $Enc(1)$ 과 $ct.a - ct.m$ 을 비트별로 $boot\text{AND}$ 연산을 적용하면, $ct.a - ct.m$ 값이 나오게 되고, 반대의 상황($ct.a < ct.m$)에서는 $Enc(1)$ 이므로 $boot\text{NOT}$ 을 취한 값, $Enc(0)$ 과 $ct.a - ct.m$ 을 비트별로 $boot\text{AND}$ 연산을 적용하면 $Enc(0)$ 이 나오게 된다. 이를 통해 암호문 $ct.a$ 의 값이 정해지고, 이를 $r + \frac{r}{2} - 1$ 만큼 반복해주면, 몫을 실수 범위까지 확장하여 구할 수 있다.

[0050] 또한, 암호문에서는 평문에서와 달리 부호를 알 수 없어 추가적인 연산을 진행해주어야 한다. 도 10의 15번째 줄에서 제수($ct.q$)와 피제수($ct.m$)의 부호비트 간에 $boot\text{XOR}$ 연산을 해주어 같은 부호면 $Enc(0)$, 반대 부호면 $Enc(1)$ 을 $ct.q$ 에 저장한다. 1번째 줄에서는 제수와 피제수 모두 양으로 전환하여, 양의 값을 갖는 상황에서 나눗셈을 쫓 진행하여, $ct.q$ 값을 얻는다. 이 값에 대해 2의 보수를 취해주어 $ct.f$ 값을 저장하고, 앞서 말한 것처럼 두 입력값의 부호가 같은 경우 $ct.q = Enc(0)$ 이기 때문에, 이를 $boot\text{NOT}$ 취한 $Enc(1)$ 과 $ct.q$ 의 비트단위별로 $boot\text{AND}$ 를 취해 $ct.q_L$ 을 얻는다.

[0051] 마찬가지로, 반대 부호를 갖는 경우 $Enc(1)$ 이 $ct.f$ 의 비트별로 $boot\text{AND}$ 를 취해 $ct.q_R$ 을 얻는다. 마지막으로, 이 두 값을 더하게 되면 최종적으로 결과값 $ct.q$ 가 된다.

[0053] 동형암호 가우스 조던 역행렬

[0054] (1) 동형암호 역행렬의 개관

[0055] 다양한 동형암호 기법들은 정수 기반의 덧셈과 곱셈 위주의 계산을 하기 때문에 결과값이 근사치로 표현되기 때문에 연산을 반복할수록 오류가 증가하여 큰 오차가 생길 수 있다. 하지만 본 기법은 실수 기반의 사칙 연산을 구현하기 때문에 입력 값의 비트 수를 늘리면, 평문에서의 결과값과의 오차를 줄일 수 있다.

[0056] 역행렬 연산에서도 마찬가지로, 솔즈 알고리즘을 적용해(나눗셈 연산이 불가능 하기 때문에) 근사값을 구하는 것과 달리 본 가우스 조던 역행렬에서는 위에서 설명한 연산들을 이용해 암호화된 데이터상에서 역행렬 값을 구한다.

[0057] 역행렬을 구하는 다양한 방법 중, 가우스 조던 소거법을 이용하는 이유는 이 소거법이 갖는 기약 행사다리꼴(reduced row echelon form(rref)) 형태가 고유성을 보장하며, 역행렬을 직관적으로 구성할 수 있기 때문이다.

[0058] (2) 암호문의 행렬 연산

[0059] 본 알고리즘의 주 단계를 시작하기 전, 입력 값이 1차원이 아닌 행렬일 때의 연산이 가능해야 한다. 암호문 상에서의 행렬 곱셈 기술은 평문에서의 행렬 곱셈과 접근법이 동일하다. 도 11은 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 행렬 곱셈의 알고리즘이다.

[0060] (3) 암호문의 가우스 조던 소거법 연산

[0061] 가우스 조던 소거법은 다른 행을 배수 연산을 통해 연속적으로 빼거나, 스케일링을 통해 행렬을 기약 행사다리꼴로 만드는 과정이다. 소거 과정은 크게 세 가지 단계- 두 행을 바꾸는 단계, 행 크기를 조절하는 단계, 다른 행에서 여러 행을 빼는 단계로 구성된다.

[0062] 도 12는 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 가우스 조던 역행렬 알고리즘을 나타낸다. 도 12를 참조하면, 우선, 피벗 행렬을 결정하는 단계로써, 가우스 조던 소거법에서 각 행렬의 가장 앞에 0이 아닌 계수를 피벗으로 결정할 수 있다. 가장 큰 피벗 요소가 있는 행을 찾기 위해 암호문의 값을 HomAbs 를 이용해

절댓값이 가장 큰 행을 찾고, 두 행렬의 순서를 바꿔 가장 위 행렬로 둔다(1 내지 6번째줄).

[0063] 다음으로 포워드(Forward) 단계로써, 대각선 아래의 방정식에서 모든 $ct.W_{jk}$ 를 소거하여 $ct.W$ 를 행 사다리꼴(ref) 형식으로 변환한다. 이 소거 단계에 의해 대각선 아래의 모든 요소가 제거되고, $ct.W$ 는 상위 삼각 행렬이 된다. 각 행에 다른 행의 변수와 계수가 반대가 되도록 상수 $ct.\tau$ 로 나누어 두 식을 곱한 뒤, 해당하는 변수들을 차례로 소거한다(7 내지 12번째줄).

[0064] 마지막으로, 백워드(Backward) 단계로, 대각선 위의 방정식 $ct.W_j$ 에서 모든 $ct.W_{jk}$ 를 소거하여 얻은 행 사다리꼴을 기약 행사다리꼴 형태로 변환한다. 이 소거 단계에서는 대각선 위의 모든 요소가 제거되고, $ct.W$ 는 대각 행렬이 되어 단위행렬이 된다. 새로운 행으로부터 하나의 변수 값을 구한 후 남은 행에 차례로 대입하여 전체 역행렬을 구한다(13 내지 20번째줄).

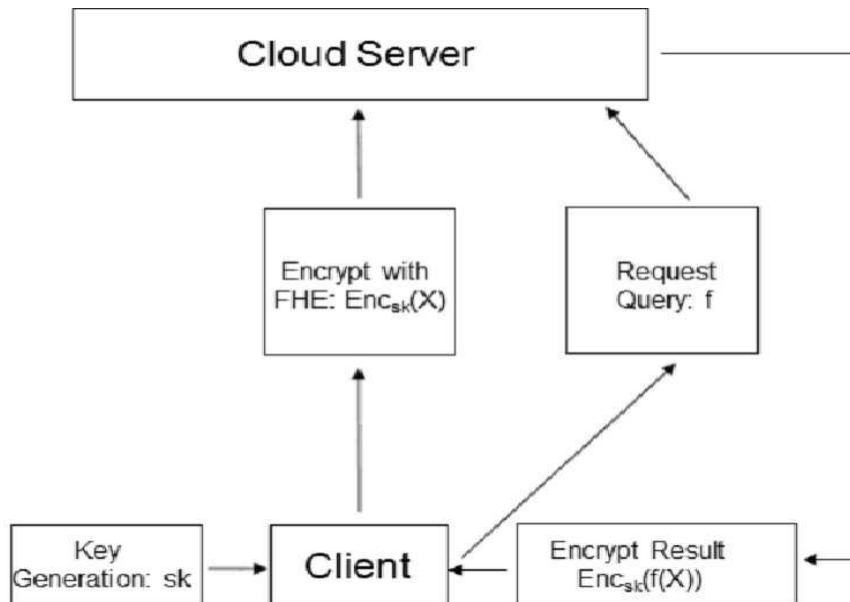
[0065] 도 13은 본 발명의 일 실시예에 따른 동형암호로 암호화된 데이터의 가우스 조던 역행렬 알고리즘을 클라우드에서 수행하는 방법을 나타낸 흐름도이다.

[0066] 도 13을 참조하면, 사용자의 개인키로 암호화한 데이터에 대해서 클라우드는 가우스 조던 역행렬 연산을 동형암호 스키마 내에서 수행한다. 마지막으로, 사용자에게 암호화된 결과값을 보내고 사용자는 이를 복호화하여 확인할 수 있다.

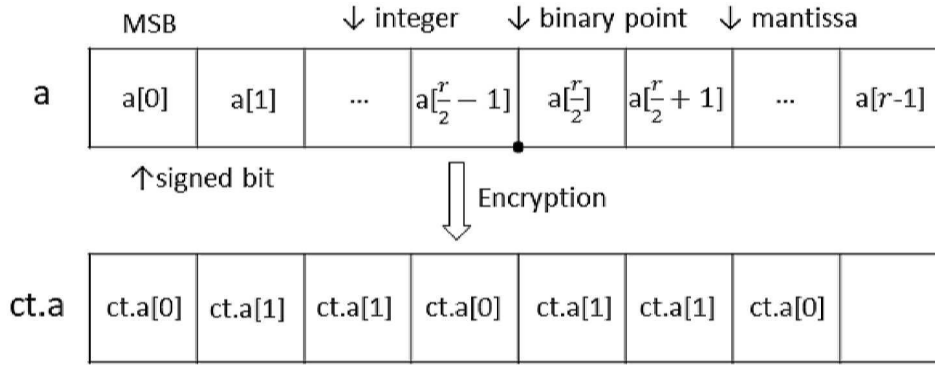
[0067] 이상, 첨부된 도면을 참조하여 본 발명의 실시 예를 설명하였지만, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자는 본 발명이 그 기술적 사상이나 필수적인 특징을 변경하지 않고서 다른 구체적인 형태로 실시될 수 있다는 것을 이해할 수 있을 것이다. 그러므로 이상에서 기술한 실시 예에는 모든 면에서 예시적인 것이며 한정적이 아닌 것으로 이해해야만 한다.

도면

도면1



도면2



도면3

Algorithm HomCompL(ct.a, ct.b): Homomorphic compare large

```

1: ct.s1 ← ENC[0]
2: for i = 0 to r - 1 do
3:   ct.s2 ← bootsXNOR(ct.a[i], ct.b[i])
4:   ct.s1 ← bootsMUX(ct.s2, ct.s1[i], ct.a[i])
5: end for
6: Return ct.s1

```

도면4

Algorithm HomCompS(ct.a, ct.b): Homomorphic compare small

```

1: ct.s1 ← ENC[0]
2: for i = 0 to r - 1 do
3:   ct.s2 ← bootsXNOR(ct.a[i], ct.b[i])
4:   ct.s1 ← bootsMUX(ct.s2, ct.s1[i], ct.b[i])
5: end for
6: Return ct.s1

```

도면5

Algorithm HomAdd(ct.a, ct.b, r): Homomorphic addition

```

1: ct.a[0] ← bootsCONSTANT[0]
2: for i = 0 to r - 1 do
3:   ct.m0 ← bootsXOR(ct.a[i], ct.b[i])
4:   ct.m1 ← bootsAND(ct.m0, ct.b[i])
5:   ct.c[i] ← bootsXOR(ct.m0, ct.t[i])
6:   ct.m2 ← bootsAND(ct.m0, ct.t[i])
7:   ct.t[i + 1] ← bootsOR(ct.m2, ct.m1)
8: end for
9: ct.m3 ← bootsXOR(ct.a[r - 1], ct.b[r - 1])
10: ct.c[r - 1] ← bootsXOR(ct.m0, ct.t[r - 1])
11: Return ct.c

```

도면6

Algorithm HomSubt(ct.a, ct.b, r): Homomorphic subtraction

```

1: ct.b ← HomTwosComp(ct.b)
2: c[0] ← Enc[0]
3: for i = 0 to r - 1 do
4:   ct.m0 ← bootsXOR(ct.a[i], ct.b[i])
5:   ct.m1 ← bootsAND(ct.m1, ct.m0, ct.b[i])
6:   ct.c[i] ← bootsXOR(ct.m0, ct.t[i])
7:   ct.m0 ← bootsAND(ct.m0, ct.t[i])
8:   ct.c[i + 1] ← bootsOR(ct.m0, ct.m1)
9: end for
10: ct.m0 ← bootsXOR(ct.a[r - 1], ct.b[r - 1])
11: ct.c[r - 1] ← bootsXOR(ct.m0, ct.t[r - 1])
12: Return ct.c

```

도면7

				0	1	1	1
×				0	1	0	1
	(r ₁)	0	0	0	1	1	1
	(r ₂)	0	0	0	0	0	0
+	(r ₃)	0	1	1	1	0	0
	(result) :	1	0	0	0	1	1

도면8

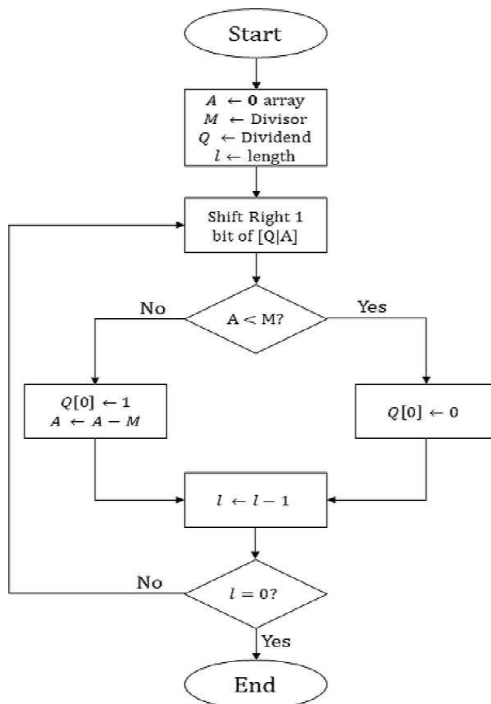
Algorithm HomMulti(ct.a, ct.b, r): Homomorphic multiplication

```

1: ct.â ← HomAbs(ct.a), ct.b̂ ← HomAbs(ct.b)
2: for i = 1 to r do
3:   ct.d[i] ← bootsAND(ct.â[i], ct.b̂[i])
4: end for
5: for i = 1 to r - 2 do
6:   if i <  $\frac{r}{2} - 1$  then
7:     ct.e ← HomRShift(ct.â,  $\frac{r}{2} - i - 1$ )
8:     for j = 1 to r do
9:       ct.f[j] ← bootsAND(ct.d[j], ct.b̂[i + 1])
10:    end for
11:    ct.d ← HomAdd(ct.d[j], ct.f[i + 1])
12:   else if i =  $\frac{r}{2} - 1$  then
13:     for j = 1 to r do
14:       ct.f[j] ← bootsAND(ct.â[j], ct.b̂[i + 1])
15:     end for
16:     ct.d ← HomAdd(ct.d, ct.f)
17:   else
18:     ct.e ← HomRShift(ct.â, i - ( $\frac{r}{2} + 1$ ))
19:     for j = 1 to r do
20:       ct.f[j] ← bootsAND(ct.e[j], ct.b̂[i + 1])
21:     end for
22:     ct.d ← HomAdd(ct.d, ct.f)
23:   end if
24: end for
25: ct.f ← HomTwosComp(ct.d)
26: ct.g ← bootsXOR(ct.a[r - 1], ct.b[r - 1])
27: ct.h ← bootsNOT(ct.g)
28: for i = 1 to r do
29:   ct.o ← bootsAND(ct.d, ct.h)
30:   ct.p ← bootsAND(ct.f, ct.g)
31: end for
32: ct.q ← HomAdd(ct.o, ct.p)
33: Return ct.q

```

도면9



도면10

Algorithm HomDiv(ct.q, ct.m, r): Homomorphic division

```

1: ct.q̇ ← HomAbs(ct.q),   ct.ṁ ← HomAbs(ct.m)
2: repeat
3:   ct.c ← (ct.q || ct.a)
4:   ct.c ← HomRShift(ct.c, 1)
5:   ct.a ← (ct.c[r], ct.c[r + 1], ..., ct.c[2r - 1])
6:   ct.q ← (ct.c[0], ct.c[1], ..., ct.c[r - 1])
7:   ct.d ← HomCompS(ct.a, ct.m)
8:   ct.q̇[0] ← bootsNOT(ct.d)
9:   ct.aM ← HomSubt(ct.a, ct.m)
10:  ct.aL ← bootsAND(ct.q̇[0], ct.aM)
11:  ct.aR ← bootsAND(ct.d, ct.a)
12:  ct.a ← bootsAND(ct.aL, ct.aR)
13: until All fractional bits of ct.q̇ are extracted( $r + \frac{r}{2} - 1$  times)
14: ct.f ← HomTwosComp(ct.q̇)
15: ct.g ← bootsXOR(ct.q[r - 1], ct.m[r - 1])
16: ct.h ← bootsNOT(ct.g)
17: for i = 1 to r do
18:   ct.qL ← bootsAND(ct.h, ct.q̇)
19:   ct.qR ← bootsAND(ct.g, ct.f)
20: end for
21: ct.q ← HomAdd(ct.qL, ct.qR)
22: Return ct.q

```

도면11

Algorithm HomMatMulti(ct.A, ct.B)

```

1: ct.C ← Enc[0]
2: for i = 1 to n do
3:   for j = 1 to n do
4:     for k = 1 to n do
5:       ct.cij ← HomMulti(ct.Aik, ct.Bkj)
6:       ct.C ← HomAdd(ct.C, ct.cij)
7:     end for
8:   end for
9: end for
10: Return ct.C

```

도면12

Algorithm ct.W⁻¹ ← HomGJ(ct.W): Homomorphic Matrix Inversion using Gauss-Jordan Elimination

Require: Encrypted Matrix ct.W_{n×n}

Ensure: Encrypted Inverse Matrix ct.W_{n×n}⁻¹

```

1: for each row k do
2:   ct.i* ← argmaxk ≤ i ≤ n HomAbs(ct.Wik)
3:   if ct.Wi*,k = 0 then
4:     Matrix is not invertible
5:   end if
6:   Swap rows k and i* ← HomMux(ct.k, ct.i*, ct.temp)
7:   for each row j below k (i.e. j = k + 1, ..., n) do
8:     ct.τ ← HomDiv(ct.Wjk, ct.Wkk)
9:     ct.Wk ← HomMulti(ct.τ, ct.Wk)
10:    ct.Wj ← HomSubt(ct.Wj, ct.Wk)
11:   end for
12: end for
13: for each row k = n, ..., 1 (i.e. in reverse) do
14:   ct.Wk ← HomDiv(ct.Wk, ct.Wkk)
15:   for each row j above k (i.e. j = k - 1, ..., 1) do
16:     ct.τ ← HomDiv(ct.Wjk, ct.Wkk)
17:     ct.Wk ← HomMulti(ct.τ, ct.Wk)
18:     ct.Wj ← HomSubt(ct.Wj, ct.Wk)
19:   end for
20: end for
21: Return ct.W-1

```

도면13

