(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2022-0068006
(43) 공개일자 2022년05월25일

(51) 국제특허분류(Int. Cl.)
G06F 8/41 (2018.01) *H04L 9/00* (2022.01)
(52) CPC특허분류
G06F 8/41 (2013.01)
H04L 9/008 (2013.01)
(21) 출원번호 10-2020-0154879
(22) 출원일자 2020년11월18일
심사청구일자 없음

(71) 출원인
삼성에스디에스 주식회사
서울특별시 송파구 올림픽로35길 125 (신천동)
연세대학교 산학협력단
서울특별시 서대문구 연세로 50 (신촌동, 연세대학교)
(72) 발명자
손정훈
서울특별시 송파구 올림픽로35길 125 (신천동, 삼성SDS West Campus)
윤효진
서울특별시 송파구 올림픽로35길 125 (신천동, 삼성SDS West Campus)
(뒷면에 계속)
(74) 대리인
두호특허법인

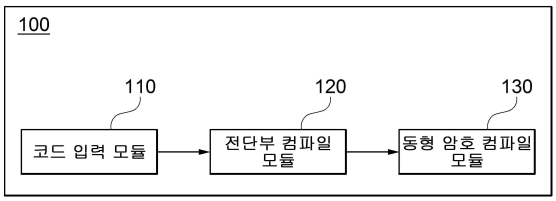
전체 청구항 수 : 총 18 항

(54) 발명의 명칭 **동형 암호 활용을 위한 소스 코드 컴파일 장치 및 방법**

(57) 요 약

동형 암호 활용을 위한 소스 코드 컴파일 장치 및 방법이 개시된다. 일 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 장치는, 동적 언어로 기술된, 주석(Annotation)을 포함하는 소스 코드를 입력 받는 코드 입력 모듈; 상기 소스 코드를 파싱하고, 상기 파싱된 소스 코드와 대응되되 상이한 형식에 따라 기술된 제1 중간 코드를 생성하는 전단부 컴파일 모듈; 및 상기 제1 중간 코드를 파싱하고, 상기 파싱된 제1 중간 코드에 포함된 적어도 일부 연산을 동형 암호 연산으로 대체하여 제2 중간 코드를 생성하는 동형 암호 컴파일 모듈을 포함한다.

대 표 도 - 도1



(72) 발명자

문덕재

서울특별시 송파구 올림픽로35길 125 (신천동, 삼성SDS West Campus)

유헌희

서울특별시 송파구 올림픽로35길 125 (신천동, 삼성SDS West Campus)

김한준

서울특별시 서대문구 연세로 50 연세대학교 제3공학관 415호

이용우

서울특별시 서대문구 연세로 50 연세대학교

김봉준

서울특별시 서대문구 연세로 50 연세대학교

명 세 서

청구범위

청구항 1

동적 언어로 기술된, 주석(Annotation)을 포함하는 소스 코드를 입력 받는 코드 입력 모듈;

상기 소스 코드를 파싱하고, 상기 파싱된 소스 코드와 대응되되 상이한 형식에 따라 기술된 제1 중간 코드를 생성하는 전단부 컴파일 모듈; 및

상기 제1 중간 코드를 파싱하고, 상기 파싱된 제1 중간 코드에 포함된 적어도 일부 연산을 동형 암호 연산으로 대체하여 제2 중간 코드를 생성하는 동형 암호 컴파일 모듈을 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 장치.

청구항 2

청구항 1항에 있어서,

상기 코드 입력 모듈은,

함수 또는 연산자를 래핑(Wrapping)한 하나 이상의 래퍼 함수에 주석이 부가된 소스 코드를 입력 받는, 동형 암호 활용을 위한 소스 코드 컴파일 장치.

청구항 3

청구항 1항에 있어서,

상기 제1 중간 코드는,

상기 소스 코드에 포함된 함수, 연산자 및 주석에 각각 대응되는 함수, 연산자 및 주석을 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 장치.

청구항 4

청구항 1항에 있어서,

상기 제1 중간 코드는,

상기 소스 코드에 포함된 데이터 중 동형 암호화될 데이터와 나머지 데이터를 구별하는 기 설정된 기호를 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 장치.

청구항 5

청구항 1항에 있어서,

상기 제2 중간 코드는,

상기 제1 중간 코드에 포함된 연산 중 동형 암호화될 입력 값을 갖는 연산을 대체하는 하나 이상의 동형 암호 연산을 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 장치.

청구항 6

청구항 5항에 있어서,

상기 하나 이상의 동형 암호 연산은,

입력되는 매개 변수가 암호화된 값인지 여부에 기초하여 선언되는 동형 암호 연산자의 종류를 달리하는, 동형 암호 활용을 위한 소스 코드 컴파일 장치.

청구항 7

청구항 1항에 있어서,

상기 파싱된 제1 중간 코드에 포함된 데이터 중 동형 암호화되지 않을 데이터 사이의 연산을 최적화하여 제3 중간 코드를 생성하는 연산 최적화 컴파일 모듈을 더 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 장치.

청구항 8

청구항 7항에 있어서,

상기 제2 중간 코드 및 상기 제3 중간 코드를 저레벨 가상 머신(LLVM; Low-Level Virtual Machine)에 기초하여 컴파일함으로써 LLVM 중간 코드를 생성하는 LLVM 컴파일 모듈을 더 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 장치.

청구항 9

청구항 8항에 있어서,

상기 LLVM 중간 코드를 기 정의된 동형 암호 라이브러리와 연동하여 동형 암호화된 입출력을 갖는 실행 프로그램을 생성하는 실행 프로그램 생성 모듈을 더 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 장치.

청구항 10

하나 이상의 프로세서들, 및

상기 하나 이상의 프로세서들에 의해 실행되는 하나 이상의 프로그램들을 저장하는 메모리를 구비한 컴퓨팅 장치에서 수행되는 방법으로서,

동적 언어로 기술된, 주석(Annotation)을 포함하는 소스 코드를 입력 받는 단계;

상기 소스 코드를 파싱하는 단계;

상기 파싱된 소스 코드와 대응되되 상이한 형식에 따라 기술된 제1 중간 코드를 생성하는 단계;

상기 제1 중간 코드를 파싱하는 단계; 및

상기 파싱된 제1 중간 코드에 포함된 적어도 일부 연산을 동형 암호 연산으로 대체하여 제2 중간 코드를 생성하는 단계를 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 방법.

청구항 11

청구항 10항에 있어서,

상기 소스 코드를 입력 받는 단계는,

함수 또는 연산자를 래핑(Wrapping)한 하나 이상의 래퍼 함수에 주석이 부가된 소스 코드를 입력 받는, 동형 암호 활용을 위한 소스 코드 컴파일 방법.

청구항 12

청구항 10항에 있어서,

상기 제1 중간 코드는,

상기 소스 코드에 포함된 함수, 연산자 및 주석에 각각 대응되는 함수, 연산자 및 주석을 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 방법.

청구항 13

청구항 10항에 있어서,

상기 제1 중간 코드는,

상기 소스 코드에 포함된 데이터 중 동형 암호화될 데이터와 나머지 데이터를 구별하는 기 설정된 기호를 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 방법.

청구항 14

청구항 10항에 있어서,

상기 제2 중간 코드는,

상기 제1 중간 코드에 포함된 연산 중 동형 암호화될 입력 값을 갖는 연산을 대체하는 하나 이상의 동형 암호 연산을 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 방법.

청구항 15

청구항 14항에 있어서,

상기 하나 이상의 동형 암호 연산은,

입력되는 매개 변수가 암호화된 값인지 여부에 기초하여 선언되는 동형 암호 연산자의 종류를 달리하는, 동형 암호 활용을 위한 소스 코드 컴파일 방법.

청구항 16

청구항 10항에 있어서,

상기 파싱된 제1 중간 코드에 포함된 데이터 중 동형 암호화되지 않을 데이터 사이의 연산을 최적화하여 제3 중간 코드를 생성하는 단계를 더 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 방법.

청구항 17

청구항 16항에 있어서,

상기 제2 중간 코드 및 상기 제3 중간 코드를 저레벨 가상 머신(LLVM; Low-Level Virtual Machine)에 기초하여 컴파일함으로써 LLVM 중간 코드를 생성하는 단계를 더 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 방법.

청구항 18

청구항 17항에 있어서,

상기 LLVM 중간 코드를 기 정의된 동형 암호 라이브러리와 연동하여 동형 암호화된 입출력을 갖는 실행 프로그램을 생성하는 단계를 더 포함하는, 동형 암호 활용을 위한 소스 코드 컴파일 방법.

발명의 설명

기 술 분 야

[0001] 개시되는 실시예들은 동형 암호의 활용을 위해 소스 코드를 컴파일하기 위한 기술과 관련된다.

배 경 기 술

[0002] 동형 암호(Homomorphic encryption)는 데이터를 암호화된 상태에서 연산할 수 있는 암호화 방법으로, 동형 암호화된 암호문들을 이용한 연산의 결과는 새로운 암호문이 되고, 이를 복호화하여 얻은 평문은 암호화하기 전의 데이터의 연산 결과와 같다는 특징이 있다.

[0003] 이에 따라, 동형 암호 기술은 프라이버시 보존이 필요한 다양한 분야에서 각광받고 있으며, 최근에는 동형 암호 기술을 소스 코드에 편리하게 적용할 수 있도록 몇몇 복잡한 알고리즘을 추상화한 동형 암호 라이브러리가 등장하기도 했다.

[0004] 그러나, 이러한 동형 암호 라이브러리를 사용하기 위해서는 여전히 보안 및 동형 암호에 대한 심도 있는 이해가 선행되어야 하기 때문에, 관련 지식이 없는 사용자가 소스 코드에 동형 암호 기술을 적용하기에는 적잖은 불편함이 있다.

선행기술문헌

특허문헌

[0005] (특허문헌 0001) 대한민국 공개특허공보 제10-2020-0046499호(2020.05.07. 공개)

발명의 내용

해결하려는 과제

[0006] 개시되는 실시예들은 보안 및 동형 암호에 대한 사전 지식 없이도 동형 암호 기술을 활용할 수 있도록 소스 코드를 효과적으로 컴파일하는 기술적인 수단을 제공하기 위한 것이다.

과제의 해결 수단

[0007] 개시되는 일 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 장치는, 동적 언어로 기술된, 주석(Annotation)을 포함하는 소스 코드를 입력 받는 코드 입력 모듈; 상기 소스 코드를 파싱하고, 상기 파싱된 소스 코드와 대응되되 상이한 형식에 따라 기술된 제1 중간 코드를 생성하는 전단부 컴파일 모듈; 및 상기 제1 중간 코드를 파싱하고, 상기 파싱된 제1 중간 코드에 포함된 적어도 일부 연산을 동형 암호 연산으로 대체하여 제2 중간 코드를 생성하는 동형 암호 컴파일 모듈을 포함한다.

[0008] 상기 코드 입력 모듈은, 함수 또는 연산자를 래핑(Wrapping)한 하나 이상의 래퍼 함수에 주석이 부가된 소스 코드를 입력 받을 수 있다.

[0009] 상기 제1 중간 코드는, 상기 소스 코드에 포함된 함수, 연산자 및 주석에 각각 대응되는 함수, 연산자 및 주석을 포함할 수 있다.

[0010] 상기 제1 중간 코드는, 상기 소스 코드에 포함된 데이터 중 동형 암호화될 데이터와 나머지 데이터를 구별하는 기 설정된 기호를 포함할 수 있다.

[0011] 상기 제2 중간 코드는, 상기 제1 중간 코드에 포함된 연산 중 동형 암호화될 입력 값을 갖는 연산을 대체하는 하나 이상의 동형 암호 연산을 포함할 수 있다.

[0012] 상기 하나 이상의 동형 암호 연산은, 입력되는 매개 변수가 암호화된 값인지 여부에 기초하여 선언되는 동형 암호 연산자의 종류를 달리할 수 있다.

[0013] 다른 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 장치는, 상기 파싱된 제1 중간 코드에 포함된 데이터 중 동형 암호화되지 않을 데이터 사이의 연산을 최적화하여 제3 중간 코드를 생성하는 연산 최적화 컴파일 모듈을 더 포함할 수 있다.

[0014] 다른 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 장치는, 상기 제2 중간 코드 및 상기 제3 중간 코드를 저레벨 가상 머신(LLVM; Low-Level Virtual Machine)에 기초하여 컴파일함으로써 LLVM 중간 코드를 생성하는 LLVM 컴파일 모듈을 더 포함할 수 있다.

[0015] 다른 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 장치는, 상기 LLVM 중간 코드를 기 정의된 동형 암호 라이브러리와 연동하여 동형 암호화된 입출력을 갖는 실행 프로그램을 생성하는 실행 프로그램 생성 모듈을 더 포함할 수 있다.

[0016] 개시되는 일 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 방법은, 하나 이상의 프로세서들, 및 상기 하나 이상의 프로세서들에 의해 실행되는 하나 이상의 프로그램들을 저장하는 메모리를 구비한 컴퓨팅 장치에서 수행되는 방법으로서, 동적 언어로 기술된, 주석(Annotation)을 포함하는 소스 코드를 입력 받는 단계; 상기 소스 코드를 파싱하는 단계; 상기 파싱된 소스 코드와 대응되되 상이한 형식에 따라 기술된 제1 중간 코드를 생성하는 단계; 상기 제1 중간 코드를 파싱하는 단계; 및 상기 파싱된 제1 중간 코드에 포함된 적어도 일부 연산을 동형 암호 연산으로 대체하여 제2 중간 코드를 생성하는 단계를 포함할 수 있다.

[0017] 상기 소스 코드를 입력 받는 단계는, 함수 또는 연산자를 래핑(Wrapping)한 하나 이상의 래퍼 함수에 주석이 부가된 소스 코드를 입력 받을 수 있다.

[0018] 상기 제1 중간 코드는, 상기 소스 코드에 포함된 함수, 연산자 및 주석에 각각 대응되는 함수, 연산자 및 주석을 포함할 수 있다.

[0019] 상기 제1 중간 코드는, 상기 소스 코드에 포함된 데이터 중 동형 암호화될 데이터와 나머지 데이터를 구별하는 기 설정된 기호를 포함할 수 있다.

[0020] 상기 제2 중간 코드는, 상기 제1 중간 코드에 포함된 연산 중 동형 암호화될 입력 값을 갖는 연산을 대체하는 하나 이상의 동형 암호 연산을 포함할 수 있다.

[0021] 상기 하나 이상의 동형 암호 연산은, 입력되는 매개 변수가 암호화된 값인지 여부에 기초하여 선언되는 동형 암호 연산자의 종류를 달리할 수 있다.

[0022] 다른 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 방법은, 상기 파싱된 제1 중간 코드에 포함된 데이터 중 동형 암호화되지 않을 데이터 사이의 연산을 최적화하여 제3 중간 코드를 생성하는 단계를 더 포함할 수 있다.

[0023] 다른 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 방법은, 상기 제2 중간 코드 및 상기 제3 중간 코드를 저레벨 가상 머신(LLVM; Low-Level Virtual Machine)에 기초하여 컴파일함으로써 LLVM 중간 코드를 생성하는 단계를 더 포함할 수 있다.

[0024] 다른 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 방법은, 상기 LLVM 중간 코드를 기 정의된 동형 암호 라이브러리와 연동하여 동형 암호화된 입출력을 갖는 실행 프로그램을 생성하는 단계를 더 포함할 수 있다.

발명의 효과

[0025] 개시되는 실시예들에 따르면, 동적 언어로 기술된 소스 코드를 입력으로 하여 컴파일 장치 내부적으로 동형 암호 기술을 적용함으로써, 동적 언어를 이용하여 전단부 개발을 수행한 개발자가 동형 암호에 대한 사전 지식을 가지고 있지 않아도 손쉽게 동형 암호화된 입출력을 갖는 애플리케이션을 개발하도록 할 수 있다.

[0026] 또한 개시되는 실시예들에 따르면, 중간 코드 내부의 연산을 최적화하는 컴파일 모듈과 동형 암호 컴파일 모듈을 복합적으로 사용함으로써, 사용자가 추후 추가적인 동형 암호 연산자 또는 최적화 기법을 용이하게 도입할 수 있다.

도면의 간단한 설명

[0027] 도 1은 일 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 장치를 설명하기 위한 블록도

도 2는 추가적인 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 장치를 설명하기 위한 블록도

도 3은 일 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 방법을 설명하기 위한 흐름도

도 4는 추가적인 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 방법을 설명하기 위한 흐름도

도 5는 단계 330을 보다 상세히 설명하기 위한 흐름도

도 6은 단계 350을 보다 상세히 설명하기 위한 흐름도

도 7은 일 실시예에 따른 소스 코드의 예시도

도 8은 일 실시예에 따른 제1 중간 코드의 예시도

도 9는 일 실시예에 따른 제2 중간 코드의 예시도

도 10은 일 실시예에 따른 컴퓨팅 장치를 포함하는 컴퓨팅 환경을 예시하여 설명하기 위한 블록도

발명을 실시하기 위한 구체적인 내용

[0028] 이하, 도면을 참조하여 구체적인 실시형태를 설명하기로 한다. 이하의 상세한 설명은 본 명세서에서 기술된 방법, 장치 및/또는 시스템에 대한 포괄적인 이해를 돕기 위해 제공된다. 그러나 이는 예시에 불과하며 개시되는 실시예들은 이에 제한되지 않는다.

[0029] 실시예들을 설명함에 있어서, 관련된 공지기술에 대한 구체적인 설명이 개시되는 실시예들의 요지를 불필요하게

흐릴 수 있다고 판단되는 경우에는 그 상세한 설명을 생략하기로 한다. 그리고, 후술되는 용어들은 개시되는 실시예들에서의 기능을 고려하여 정의된 용어들로서 이는 사용자, 운용자의 의도 또는 관례 등에 따라 달라질 수 있다. 그러므로 그 정의는 본 명세서 전반에 걸친 내용을 토대로 내려져야 할 것이다. 상세한 설명에서 사용되는 용어는 단지 실시예들을 기술하기 위한 것이며, 결코 제한적이어서는 안 된다. 명확하게 달리 사용되지 않는 한, 단수 형태의 표현은 복수 형태의 의미를 포함한다. 본 설명에서, "포함" 또는 "구비"와 같은 표현은 어떤 특성들, 숫자들, 단계들, 동작들, 요소들, 이들의 일부 또는 조합을 가리키기 위한 것이며, 기술된 것 이외에 하나 또는 그 이상의 다른 특성, 숫자, 단계, 동작, 요소, 이들의 일부 또는 조합의 존재 또는 가능성을 배제하도록 해석되어서는 안 된다.

[0030] 도 1은 일 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 장치(100)를 설명하기 위한 블록도이다.

[0031] 이하의 실시예들에서, '소스 코드'는 컴퓨터 프로그램을 사람이 읽을 수 있는 프로그래밍 언어로 기술한 텍스트 파일을 의미한다. 또한, '컴파일(Compile)'은 특정 프로그래밍 언어로 쓰여 있는 문서를 다른 프로그래밍 언어로 번역하는 일련의 작업을 의미하며, 컴파일 결과 출력되는 파일을 '목적 코드'라고 지칭한다.

[0032] 도시된 바와 같이, 일 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 장치(100)는 코드 입력 모듈(110), 전단부 컴파일 모듈(120) 및 동형 암호 컴파일 모듈(130)을 포함한다.

[0033] 코드 입력 모듈(110)은 동적 언어로 기술된, 주석(Annotation)을 포함하는 소스 코드를 입력 받는다.

[0034] 일 실시예에 따르면, 소스 코드는 임의의 동적 언어로 기술될 수 있으며, 이때 동적 언어는 컴파일 과정 중 수행하는 작업들을 실행 도중(Runtime)에 수행하는 고급 언어를 통칭한다. 동적 언어는 예를 들어, 파이썬(Python), R, 자바(Java), 자바스크립트(JavaScript), 루비(Ruby), PHP, Objective-C 등을 포함할 수 있으나, 반드시 이에 한정되는 것은 아니다.

[0035] 일 실시예에 따르면, 코드 입력 모듈(110)에 입력되는 소스 코드는 함수 또는 연산자를 래핑(Wrapping)한 하나 이상의 래퍼 함수에 부가된 주석을 포함할 수 있다.

[0036] 이하의 실시예들에서, '래퍼 함수'란 함수 내에 실행하고자 하는 구현 소스가 위치하지 않고, 함수가 호출되는 부분만 내부적인 소스로 구현된 함수를 의미한다.

[0037] 일 실시예에 따르면, 래핑된 함수가 if 함수인 경우, 래퍼 함수에 부가되는 주석은 if 함수임을 식별할 수 있는 "모듈 호출 이름.if" 형태의 주석일 수 있다. 예를 들어, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100)를 구현한 모듈의 이름이 "hecate"이고, 파이썬(Python) 언어로 작성된 소스 코드 상에서 "hecate"를 import 명령어를 통해 "hc"로 호출한 경우, if 함수를 래핑한 래퍼 함수에 부가되는 주석은 "hc.if" 형태의 주석일 수 있다.

[0038] 한편 일 실시예에 따르면, 래핑된 함수가 for 함수인 경우, 래퍼 함수에 부가되는 주석은 for 함수임을 식별할 수 있는 "모듈 호출 이름.for" 형태의 주석일 수 있다.

[0039] 한편 일 실시예에 따르면, 래핑된 함수가 if 함수 및 for 함수 이외의 별도로 정의된 함수인 경우, 래퍼 함수에 부가되는 주석은 기 정의된 함수임을 식별할 수 있는 "모듈 호출 이름.func" 형태의 주석일 수 있다.

[0040] 전단부 컴파일 모듈(120)은 입력된 소스 코드를 파싱하고, 파싱된 소스 코드와 대응되되 상이한 형식에 따라 기술된 제1 중간 코드를 생성한다.

[0041] 일 실시예에 따르면, 전단부 컴파일 모듈(120)은 입력된 소스 코드를 파싱하되, 입력된 소스 코드의 추상 구문 구조(AST; Abstract Syntax Tree)를 따라 순차적으로 파싱 및 제1 중간 코드의 생성을 수행할 수 있다.

[0042] 일 실시예에 따르면, 제1 중간 코드는 소스 코드에 포함된 함수, 연산자 및 주석에 각각 대응되는 함수, 연산자 및 주석을 포함할 수 있다.

[0043] 구체적으로, 전단부 컴파일 모듈(120)은 소스 코드 내 if 함수, for 함수 및 별도로 기 정의된 함수 각각에 부가된 주석을 식별하여, 각 주석 및 각 주석이 부가된 함수를 기 정의된 별도의 라이브러리에서 지원하는 연산자 및 함수로 기술함으로써 제1 중간 코드를 생성할 수 있다.

[0044] 예를 들어, 전단부 컴파일 모듈(120)은 소스 코드 내 기 정의된 함수 및 이에 부가된 주석을 멀티 레벨 중간 표현(MLIR; Multi-Level Intermediate Representation) 표준 다이어렉트(Dialect)에서 지원하는 연산자 및 함수로 기술할 수 있으며, 소스 코드 내 if 함수, for 함수 및 이에 부가된 주석을 구조 제어 흐름(SCF; Structured

Control Flow) 다이어렉트에서 지원하는 연산자 및 함수로 기술할 수 있다.

[0045] 일 실시예에 따르면, 제1 중간 코드는 소스 코드에 포함된 데이터 중 동형 암호화될 데이터와 나머지 데이터를 구별하는 기 설정된 기호를 포함할 수 있다.

[0046] 예를 들어, 제1 중간 코드는 동형 암호화될 데이터 앞에 별표(asterisk)를 표기함으로써 동형 암호화될 데이터와 나머지 데이터를 구별할 수 있으며, 상기 별표는 코드 입력 모듈(110)에 소스 코드가 입력될 당시 기 표기된 기호일 수 있다. 그러나, 동형 암호화될 데이터를 식별하기 위한 기호는 별표에 한정되는 것은 아니며, 사용자의 선택에 따라 다양한 기호가 사용될 수 있음에 유의해야 할 것이다.

[0047] 동형 암호 컴파일 모듈(130)은 생성된 제1 중간 코드를 파싱하고, 파싱된 제1 중간 코드에 포함된 적어도 일부 연산을 동형 암호 연산으로 대체하여 제2 중간 코드를 생성한다.

[0048] 일 실시예에 따르면, 제2 중간 코드는 제1 중간 코드에 포함된 연산 중 동형 암호화될 입력 값을 갖는 연산을 대체하는 하나 이상의 동형 암호 연산을 포함할 수 있다.

[0049] 예를 들어, 동형 암호 컴파일 모듈(130)은 제1 중간 코드 내 동형 암호화될 데이터 및 이 데이터의 연산에 필요한 연산자를 기 정의된 동형 암호 라이브러리에서 지원하는 동형 암호 연산자 및 함수로 기술할 수 있다.

[0050] 일 실시예에 따르면, 하나 이상의 동형 암호 연산은 입력되는 매개 변수가 암호화된 값인지 여부에 기초하여 선언되는 동형 암호 연산자의 종류를 달리할 수 있다.

[0051] 예를 들어, 입력되는 매개 변수가 평문인 경우, 동형 암호 컴파일 모듈(130)은 매개 변수를 암호화하기 위한 함수를 선언할 수 있다. 반면, 입력되는 매개 변수가 암호화된 값인 경우, 동형 암호 컴파일 모듈(130)은 입력된 하나 이상의 매개 변수에 기초한 연산을 수행하기 위해 재선형화(relinearization), 리스케일(rescale), 합산(sum) 함수 등을 선언할 수 있다. 다만, 동형 암호 컴파일 모듈(130)이 선언 가능한 함수들은 이에 한정되는 것은 아니며, 동형 암호 라이브러리에서 기 정의된 사항에 따라 추가적인 함수를 선언할 수 있음은 자명하다.

[0052] 도 2는 추가적인 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 장치(200)를 설명하기 위한 블록도이다.

[0053] 도시된 바와 같이, 추가적인 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 장치(200)는 도 1에 도시된 구성 외에도, 연산 최적화 컴파일 모듈(210), LLVM 컴파일 모듈(220) 및 실행 프로그램 생성 모듈(230)을 더 포함할 수 있다.

[0054] 연산 최적화 컴파일 모듈(210)은 파싱된 제1 중간 코드에 포함된 데이터 중 동형 암호화되지 않을 데이터 사이의 연산을 최적화하여 제3 중간 코드를 생성할 수 있다.

[0055] 일 실시예에 따르면, 연산 최적화 컴파일 모듈(210)은 제1 중간 코드에 포함된 데이터 사이의 연산을 최적화하기 위하여, 제1 중간 코드의 적어도 일부를 어파인(Affine) 다이어렉트, 구조 제어 흐름 다이어렉트, 벡터 다이어렉트 중 적어도 하나에서 지원하는 연산자 및 함수로 기술함으로써 제3 중간 코드를 생성할 수 있다.

[0056] LLVM 컴파일 모듈(220)은 제2 중간 코드 및 제3 중간 코드를 저레벨 가상 머신(LLVM; Low-Level Virtual Machine)에 기초하여 컴파일함으로써 LLVM 중간 코드를 생성할 수 있다.

[0057] 이하의 실시예들에서, 'LLVM'은 컴파일러에 입력되는 프로그램을 프로그램이 작성된 언어에 무관하게 최적화하는 컴파일러 내 기반구조를 의미한다.

[0058] 구체적으로, 제2 중간 코드와 제3 중간 코드는 상이한 다이어렉트에서 지원하는 연산자 및 함수로 기술된 중간 코드이기 때문에, LLVM 컴파일 모듈(220)은 이를 결합하여 컴파일함으로써 소스 코드에 다양한 다이어렉트에서 지원하는 연산자 및 함수가 적용되도록 할 수 있다.

[0059] 일 실시예에 따르면, LLVM 컴파일 모듈(220)은 제2 중간 코드의 경우, 동형 암호 라이브러리에서 지원하는 함수를 호출함으로써 컴파일을 수행할 수 있다.

[0060] 한편 일 실시예에 따르면, LLVM 컴파일 모듈(220)은 제3 중간 코드의 경우, 기 정의된 어파인 다이어렉트, 구조 제어 흐름 다이어렉트, 벡터 다이어렉트 등에서 기 정의된 하강 루트(lowering pass)에 따라 컴파일을 수행할 수 있다.

[0061] 실행 프로그램 생성 모듈(230)은 LLVM 컴파일 모듈(220)에서 생성된 LLVM 중간 코드를 기 정의된 동형 암호 라

이브러리(Library)와 연동하여 동형 암호화된 입출력을 갖는 실행 프로그램을 생성할 수 있다.

[0062] 일 실시예에 따르면, 실행 프로그램 생성 모듈(230)은 비트코드(bitcode) 형태의 LLVM 중간 코드를 기 정의된 동형 암호 라이브러리와 연동함으로써 실행 프로그램을 생성할 수 있다.

[0063] 일 실시예에 따르면, 실행 프로그램 생성 모듈(230)은 LLVM 중간 코드를 기 정의된 동형 암호 라이브러리와 연동하고, 추가적으로 SCARF(SCAlable Realtime Forensics) API를 호출함으로써 실행 프로그램을 생성할 수 있다.

[0064] 상기 도 1 및 도 2로 도시된 실시예에서, 각 구성들은 이하에 기술된 것 이외에 상이한 기능 및 능력을 가질 수 있고, 이하에 기술된 것 이외에도 추가적인 구성을 포함할 수 있다.

[0065] 또한, 일 실시예에서, 코드 입력 모듈(110), 전단부 컴파일 모듈(120), 동형 암호 컴파일 모듈(130), 연산 최적화 컴파일 모듈(210), LLVM 컴파일 모듈(220) 및 실행 프로그램 생성 모듈(230)은 물리적으로 구분된 하나 이상의 장치를 이용하여 구현되거나, 하나 이상의 프로세서 또는 하나 이상의 프로세서 및 소프트웨어의 결합에 의해 구현될 수 있으며, 도시된 예와 달리 구체적 동작에 있어 명확히 구분되지 않을 수 있다.

[0066] 도 3은 일 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 방법을 설명하기 위한 흐름도이다.

[0067] 도 3에 도시된 방법은 예를 들어, 상술한 동형 암호 활용을 위한 소스 코드 컴파일 장치(100)에 의해 수행될 수 있다.

[0068] 310 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100)는 동적 언어로 기술된, 주석을 포함하는 소스 코드를 입력 받는다.

[0069] 320 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100)는 입력된 소스 코드를 파싱한다.

[0070] 330 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100)는 파싱된 소스 코드와 대응되되, 상이한 형식에 따라 기술된 제1 중간 코드를 생성한다.

[0071] 340 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100)는 생성된 제1 중간 코드를 파싱한다.

[0072] 350 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100)는 파싱된 제1 중간 코드에 포함된 적어도 일부 연산을 동형 암호 연산으로 대체하여 제2 중간 코드를 생성한다.

[0073] 도 4는 추가적인 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 방법을 설명하기 위한 흐름도이다.

[0074] 도 4에 도시된 방법은 예를 들어, 상술한 동형 암호 활용을 위한 소스 코드 컴파일 장치(200)에 의해 수행될 수 있다.

[0075] 도시된 단계 410 내지 440은 도 3을 참조하여 설명한 단계 310 내지 340과 대응되므로, 이에 대한 중복되는 설명은 생략하기로 한다.

[0076] 450 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(200)는 제1 중간 코드가 실행됨에 따라 수행되는 연산이 동형 암호화 대상이 되는 연산인지 판단한다.

[0077] 460 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(200)는 판단 대상인 연산이 동형 암호화 대상인 것으로 판단된 경우, 파싱된 제1 중간 코드에 포함된 적어도 일부 연산을 동형 암호 연산으로 대체하여 제2 중간 코드를 생성한다.

[0078] 한편 470 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(200)는 판단 대상인 연산이 동형 암호화 대상이 아닌 것으로 판단된 경우, 해당 연산을 최적화하여 제3 중간 코드를 생성한다.

[0079] 480 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(200)는 제2 중간 코드 및 제3 중간 코드를 저레벨 가상 머신에 기초하여 컴파일함으로써 LLVM 중간 코드를 생성한다.

[0080] 490 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(200)는 LLVM 중간 코드를 기 정의된 동형 암호 라이브러리와 연동하여 동형 암호화된 입출력을 갖는 실행 프로그램을 생성한다.

[0081] 도 5는 단계 330을 보다 상세히 설명하기 위한 흐름도이다.

[0082] 도 5에 도시된 방법은 예를 들어, 상술한 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)에 의해 수행될 수 있다.

[0083] 510 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 파싱된 소스 코드를 라인 단위로 탐색한다.

[0084] 520 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 탐색 대상 라인에 위치한 소스 코드가 기 정의된 함수형 주석"hc.func"인지 판단한다.

[0085] 530 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 탐색 대상 라인에 위치한 소스 코드가 기 정의된 함수형 주석 "hc.func"인 경우, 해당 주석 및 해당 주석에 대응되는 기 정의된 함수를 상이한 형식의 함수 "func"로 번역한다.

[0086] 540 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 탐색 대상 라인에 위치한 소스 코드가 기 정의된 함수형 주석 "hc.func"이 아닌 경우, 해당 소스 코드가 기 정의된 for 함수에 대한 주석 "hc.for"인지 판단한다.

[0087] 550 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 탐색 대상 라인에 위치한 소스 코드가 기 정의된 for 함수에 대한 주석 "hc.for"인 경우, 해당 주석 및 해당 주석에 대응되는 기 정의된 for 함수를 상이한 형식의 for 함수로 번역한다.

[0088] 560 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 탐색 대상 라인에 위치한 소스 코드가 기 정의된 for 함수에 대한 주석 "hc.for"가 아닌 경우, 해당 소스 코드가 기 정의된 if 함수에 대한 주석 "hc.if"인지 판단한다.

[0089] 570 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 탐색 대상 라인에 위치한 소스 코드가 기 정의된 if 함수에 대한 주석 "hc.if"인 경우, 해당 주석 및 해당 주석에 대응되는 기 정의된 if 함수를 상이한 형식의 if 함수로 번역한다.

[0090] 580 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 탐색 대상 라인에 위치한 소스 코드가 기 정의된 주석 "hc.func", "hc.for" 및 "hc.if"가 아닌 경우, 해당 라인에 위치한 소스 코드를 기 설정된 형식에 따라 번역한다.

[0091] 590 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 소스 코드의 모든 라인이 탐색되었는지 판단한다.

[0092] 이후, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 모든 라인이 탐색되었다고 판단되는 경우, 소스 코드에 대한 탐색 과정을 종료하고 이 시점까지의 번역의 결과를 제1 중간 코드로 생성한다.

[0093] 한편, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 탐색하지 않은 라인이 있다고 판단되는 경우, 최근에 탐색한 라인부터 기산하여 소스 코드를 다시 라인 단위로 탐색한다.

[0094] 도 6은 단계 350을 보다 상세히 설명하기 위한 흐름도이다.

[0095] 도 6에 도시된 방법은 예를 들어, 상술한 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)에 의해 수행될 수 있다.

[0096] 610 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 파싱된 제1 중간 코드를 라인 단위로 탐색한다.

[0097] 620 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 탐색 대상 라인에 위치한 제1 중간 코드가 동형 암호화될 데이터 사이의 연산을 위해 필요한 연산자인지 판단한다.

[0098] 630 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 탐색 대상 라인에 위치한 제1 중간 코드가 동형 암호화될 데이터 사이의 연산을 위해 필요한 연산자라 판단된 경우, 해당 연산자를 기 정의된 동형 암호 라이브러리에서 지원하는 동형 암호 연산자로 번역한다.

[0099] 640 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 탐색 대상 라인에 위치한 제1 중간 코드가 동형 암호화될 데이터 사이의 연산을 위해 필요한 연산자가 아닌 것으로 판단된 경우, 탐색 대상 라인에 위치한 제1 중간 코드를 기 설정된 형식에 따라 번역한다.

[0100] 650 단계에서, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 제1 중간 코드의 모든 라인이 탐색되었는지 판단한다.

[0101] 이후, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 모든 라인이 탐색되었다고 판단되는 경우, 제1 중간 코드에 대한 탐색 과정을 종료하고 이 시점까지의 번역의 결과를 제2 중간 코드로 생성한다.

[0102] 한편, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100, 200)는 탐색하지 않은 라인이 있다고 판단되는 경우, 최근에 탐색한 라인부터 기산하여 제1 중간 코드를 다시 라인 단위로 탐색한다.

[0103] 상기 도시된 도 3 내지 도 6에서는 상기 방법을 복수 개의 단계로 나누어 기재하였으나, 적어도 일부의 단계들은 순서를 바꾸어 수행되거나, 다른 단계와 결합되어 함께 수행되거나, 생략되거나, 세부 단계들로 나뉘어 수행되거나, 또는 도시되지 않은 하나 이상의 단계가 부가되어 수행될 수 있다.

[0104] 도 7은 일 실시예에 따른 소스 코드의 예시도(700)이다.

[0105] 도 7을 참조하면, 동형 암호 활용을 위한 소스 코드 컴파일 장치(100)를 구현한 모듈 "hecate"가 "hc"로 호출된다.

[0106] "def" 명령어에 의해 정의되는 "example" 함수는 매개 변수 a, b를 입력으로 받으며, 해당 함수 앞에는 "hc.func"주석이 부가된다. 구체적으로, "hc.func"주석의 내용을 보면, 매개 변수 a에 *5가 대응되고, b에 1이 대응되며, "example"함수의 리턴 값인 a**[1,2,3,4,5]+b에 *1이 대응된다.

[0107] 즉 다시 말하면, 매개 변수 a와 리턴 값 a**[1,2,3,4,5]+b는 동형 암호화 대상이며, 각 대응되는 숫자는 매개 변수나 리턴 값의 원소(element) 수를 의미한다.

[0108] 또한, "abs"함수는 래퍼 함수로서, 내부의 if 함수를 래핑하는데, "abs"함수 앞에는 if 함수를 식별하기 위한 "hc.if"주석이 부가된다.

[0109] 이때, "hc.if"주석이 부가되는 "abs"함수는 불리안(Boolean) 매개 변수 하나를 전달받아 if 함수를 평가하는 바디(body)를 갖는 함수임을 알 수 있다.

[0110] 또한, "hc.setMain" 함수는 메인 함수로 호출될 "hecate" 상의 함수를 선택하는 함수이며, "hc.compile" 함수는 일련의 선언된 함수를 컴파일하고, 컴파일된 모듈의 경로를 포함하는 이름을 리턴 값으로 출력하는 함수이다.

[0111] 도 8은 일 실시예에 따른 제1 중간 코드의 예시도(800)이다.

[0112] 도 8을 참조하면, 소스 코드의 "example" 함수는 "func" 함수로 변환되고, "example" 함수의 매개 변수였던 a와 b는 각각 %arg, %arg1으로 변환된다.

[0113] 또한, 소스 코드에서는 주석을 "@"를 이용하여 표기하였으나, 제1 중간 코드에서는 주석을 콜론(:)을 이용하여 표기하며, 동형 암호화될 데이터는 소스 코드와 마찬가지로 별표로 구별함을 알 수 있다.

[0114] 한편, "constant", "cmph", "negation", "innerprod", "add" 등의 연산자 또는 함수는 기 정의된 전단부 라이브러리(도 8에서 "fe"로 호출됨)에서 지원하는 연산자 또는 함수이며, "if", "yield" 등의 함수는 기 정의된 구조 제어 흐름 다이어렉트에서 지원하는 함수임을 알 수 있다.

[0115] 도 9는 일 실시예에 따른 제2 중간 코드의 예시도(900)이다.

[0116] 도 9를 참조하면, 동형 암호화될 데이터인 %arg와 리턴 값을 연산하기 위해, 기 정의된 동형 암호 라이브러리(도 9에서 "he"로 호출됨)에서 지원하는 연산자 및 함수인 "multcp", "rescale", "sum", "modswitch", "addcp" 등이 선언됨을 알 수 있다.

[0117] 또한, 제1 중간 코드에 기술된 기 정의된 전단부 라이브러리에서 지원하는 연산자 또는 함수들은 제2 중간 코드에서 MLIR 표준 다이어렉트(도 9에서 "std"로 호출됨)에서 지원하는 연산자 또는 함수 "constant", "cmp", "negf" 등으로 기술됨을 알 수 있다.

[0118] 또한, 동형 암호화 대상을 구별하기 위해 표기되었던 별표 대신, 동형 암호화 대상은 "cipher"로, 평문은 "plain"으로 표기함을 알 수 있다.

[0119] 이때, 평문 또한 동형 암호화 대상과 함께 연산되어야 하는 관계로, 평문을 암호화하기 위해 "encode" 함수 역시 선언됨을 알 수 있다.

[0120] 결과적으로, 도 7에서 별표로 표기된 매개 변수는, 도 9에서 동형 암호 라이브러리에서 지원하는 연산자 및 함수에 기초하여 %6으로 동형 암호화되고, 도 7에서 별표로 표기된 리턴 값은, 도 9에서 %10으로 동형 암호화됨을

알 수 있다.

[0121] 도 10은 일 실시예에 따른 컴퓨팅 장치를 포함하는 컴퓨팅 환경(10)을 예시하여 설명하기 위한 블록도이다. 도시된 실시예에서, 각 컴포넌트들은 이하에 기술된 것 이외에 상이한 기능 및 능력을 가질 수 있고, 이하에 기술된 것 이외에도 추가적인 컴포넌트를 포함할 수 있다.

[0122] 도시된 컴퓨팅 환경(10)은 컴퓨팅 장치(12)를 포함한다. 일 실시예에서, 컴퓨팅 장치(12)는 동형 암호 활용을 위한 소스 코드 컴파일 장치(100)일 수 있다. 또한, 컴퓨팅 장치(12)는 추가적인 실시예에 따른 동형 암호 활용을 위한 소스 코드 컴파일 장치(200)일 수 있다.

[0123] 컴퓨팅 장치(12)는 적어도 하나의 프로세서(14), 컴퓨터 판독 가능 저장 매체(16) 및 통신 버스(18)를 포함한다. 프로세서(14)는 컴퓨팅 장치(12)로 하여금 앞서 언급된 예시적인 실시예에 따라 동작하도록 할 수 있다. 예컨대, 프로세서(14)는 컴퓨터 판독 가능 저장 매체(16)에 저장된 하나 이상의 프로그램들을 실행할 수 있다. 상기 하나 이상의 프로그램들은 하나 이상의 컴퓨터 실행 가능 명령어를 포함할 수 있으며, 상기 컴퓨터 실행 가능 명령어는 프로세서(14)에 의해 실행되는 경우 컴퓨팅 장치(12)로 하여금 예시적인 실시예에 따른 동작들을 수행하도록 구성될 수 있다.

[0124] 컴퓨터 판독 가능 저장 매체(16)는 컴퓨터 실행 가능 명령어 내지 프로그램 코드, 프로그램 데이터 및/또는 다른 적합한 형태의 정보를 저장하도록 구성된다. 컴퓨터 판독 가능 저장 매체(16)에 저장된 프로그램(20)은 프로세서(14)에 의해 실행 가능한 명령어의 집합을 포함한다. 일 실시예에서, 컴퓨터 판독 가능 저장 매체(16)는 메모리(랜덤 액세스 메모리와 같은 휘발성 메모리, 비휘발성 메모리, 또는 이들의 적절한 조합), 하나 이상의 자기 디스크 저장 디바이스들, 광학 디스크 저장 디바이스들, 플래시 메모리 디바이스들, 그 밖에 컴퓨팅 장치(12)에 의해 액세스되고 원하는 정보를 저장할 수 있는 다른 형태의 저장 매체, 또는 이들의 적합한 조합일 수 있다.

[0125] 통신 버스(18)는 프로세서(14), 컴퓨터 판독 가능 저장 매체(16)를 포함하여 컴퓨팅 장치(12)의 다른 다양한 컴포넌트들을 상호 연결한다.

[0126] 컴퓨팅 장치(12)는 또한 하나 이상의 입출력 장치(24)를 위한 인터페이스를 제공하는 하나 이상의 입출력 인터페이스(22) 및 하나 이상의 네트워크 통신 인터페이스(26)를 포함할 수 있다. 입출력 인터페이스(22) 및 네트워크 통신 인터페이스(26)는 통신 버스(18)에 연결된다. 입출력 장치(24)는 입출력 인터페이스(22)를 통해 컴퓨팅 장치(12)의 다른 컴포넌트들에 연결될 수 있다. 예시적인 입출력 장치(24)는 포인팅 장치(마우스 또는 트랙패드 등), 키보드, 터치 입력 장치(터치패드 또는 터치스크린 등), 음성 또는 소리 입력 장치, 다양한 종류의 센서 장치 및/또는 촬영 장치와 같은 입력 장치, 및/또는 디스플레이 장치, 프린터, 스피커 및/또는 네트워크 카드와 같은 출력 장치를 포함할 수 있다. 예시적인 입출력 장치(24)는 컴퓨팅 장치(12)를 구성하는 일 컴포넌트로서 컴퓨팅 장치(12)의 내부에 포함될 수도 있고, 컴퓨팅 장치(12)와는 구별되는 별개의 장치로 컴퓨팅 장치(12)와 연결될 수도 있다.

[0127] 한편, 본 발명의 실시예는 본 명세서에서 기술한 방법들을 컴퓨터상에서 수행하기 위한 프로그램, 및 상기 프로그램을 포함하는 컴퓨터 판독 가능 기록매체를 포함할 수 있다. 상기 컴퓨터 판독 가능 기록매체는 프로그램 명령, 로컬 데이터 파일, 로컬 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체는 본 발명을 위하여 특별히 설계되고 구성된 것들이거나, 또는 컴퓨터 소프트웨어 분야에서 통상적으로 사용 가능한 것일 수 있다. 컴퓨터 판독 가능 기록매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체, CD-ROM, DVD와 같은 광 기록 매체, 및 롬, 램, 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 상기 프로그램의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함할 수 있다.

[0128] 이상에서 본 발명의 대표적인 실시예들을 상세하게 설명하였으나, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자는 상술한 실시예에 대하여 본 발명의 범주에서 벗어나지 않는 한도 내에서 다양한 변형이 가능함을 이해할 것이다. 그러므로 본 발명의 권리범위는 설명된 실시예에 국한되어 정해져서는 안 되며, 후술하는 청구범위뿐만 아니라 이 청구범위와 균등한 것들에 의해 정해져야 한다.

부호의 설명

[0129] 10: 컴퓨팅 환경

12: 컴퓨팅 장치

14: 프로세서

16: 컴퓨터 판독 가능 저장 매체

18: 통신 버스

20: 프로그램

22: 입출력 인터페이스

24: 입출력 장치

26: 네트워크 통신 인터페이스

100, 200: 동형 암호 활용을 위한 소스 코드 컴파일 장치

110: 코드 입력 모듈

120: 전단부 컴파일 모듈

130: 동형 암호 컴파일 모듈

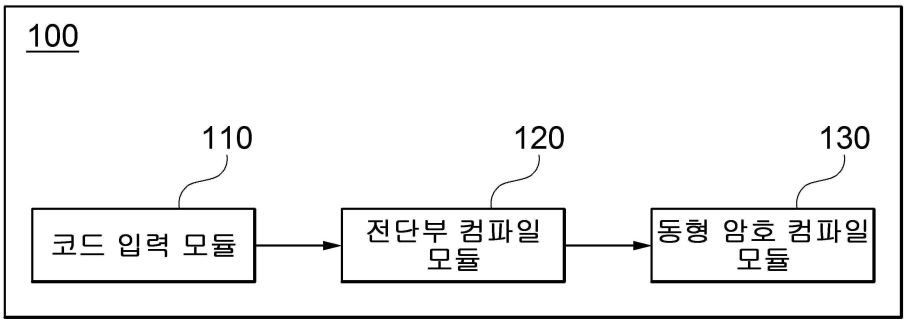
210: 연산 최적화 컴파일 모듈

220: LLVM 컴파일 모듈

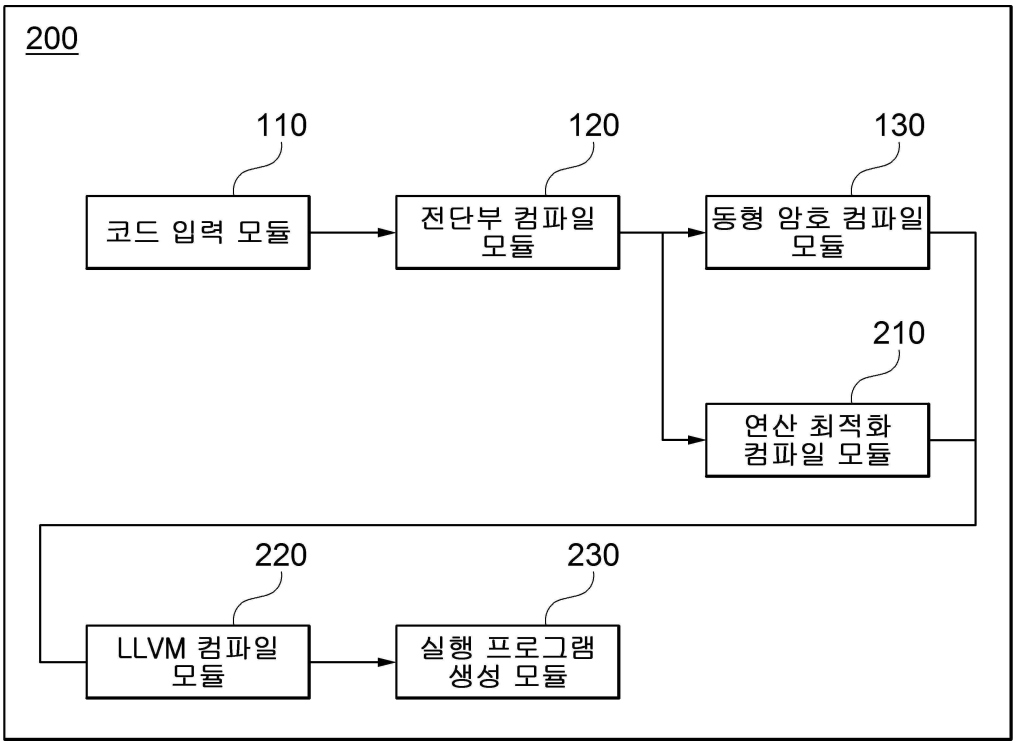
230: 실행 프로그램 생성 모듈

도면

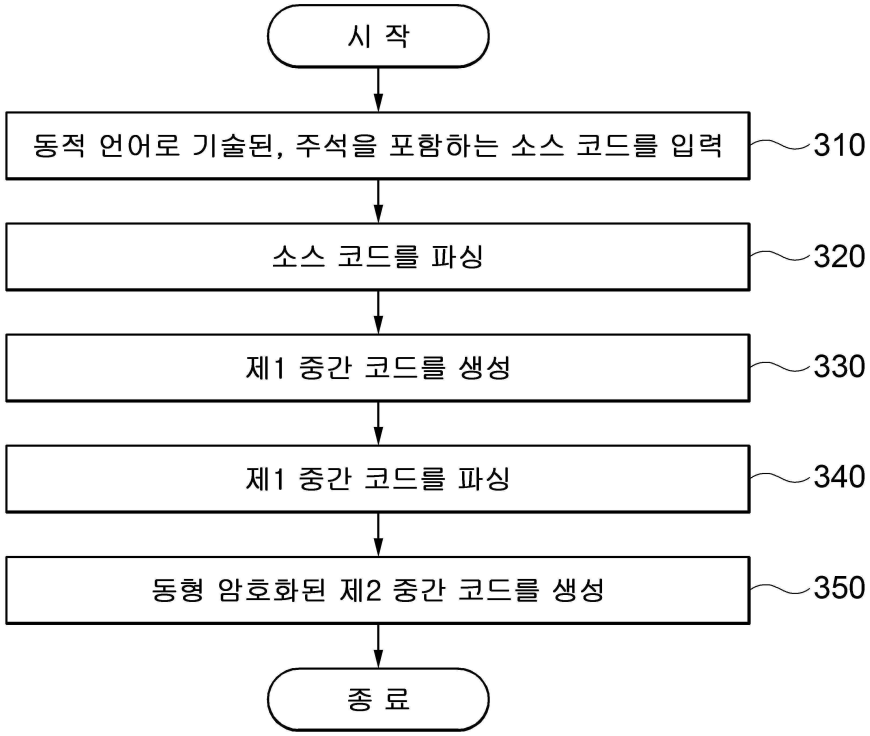
도면1



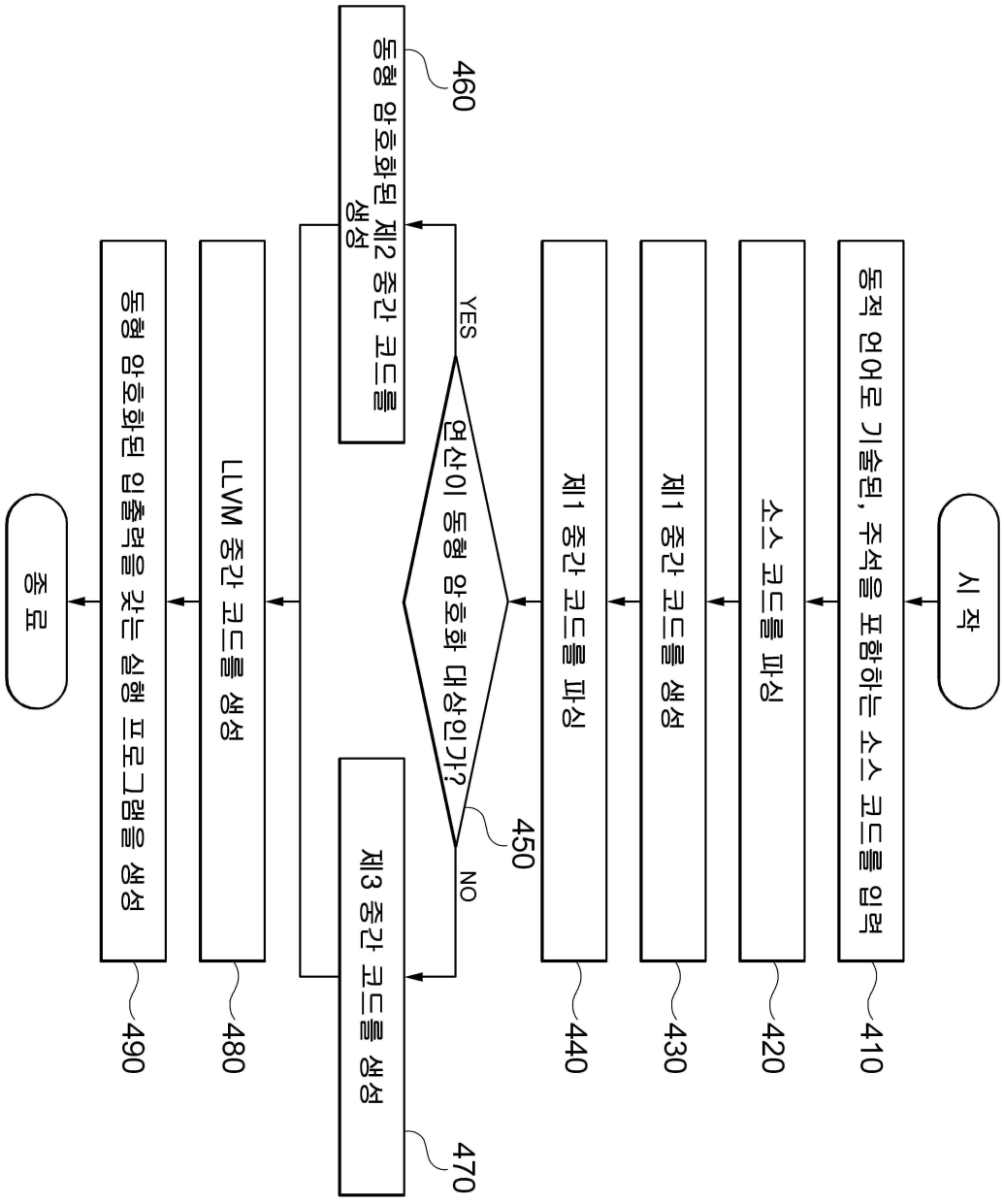
도면2


200
110
120
130
코드 입력 모듈
전단부 컴파일 모듈
동형 암호 컴파일 모듈
210
연산 최적화 컴파일 모듈
220
230
LLVM 컴파일 모듈
실행 프로그램 생성 모듈

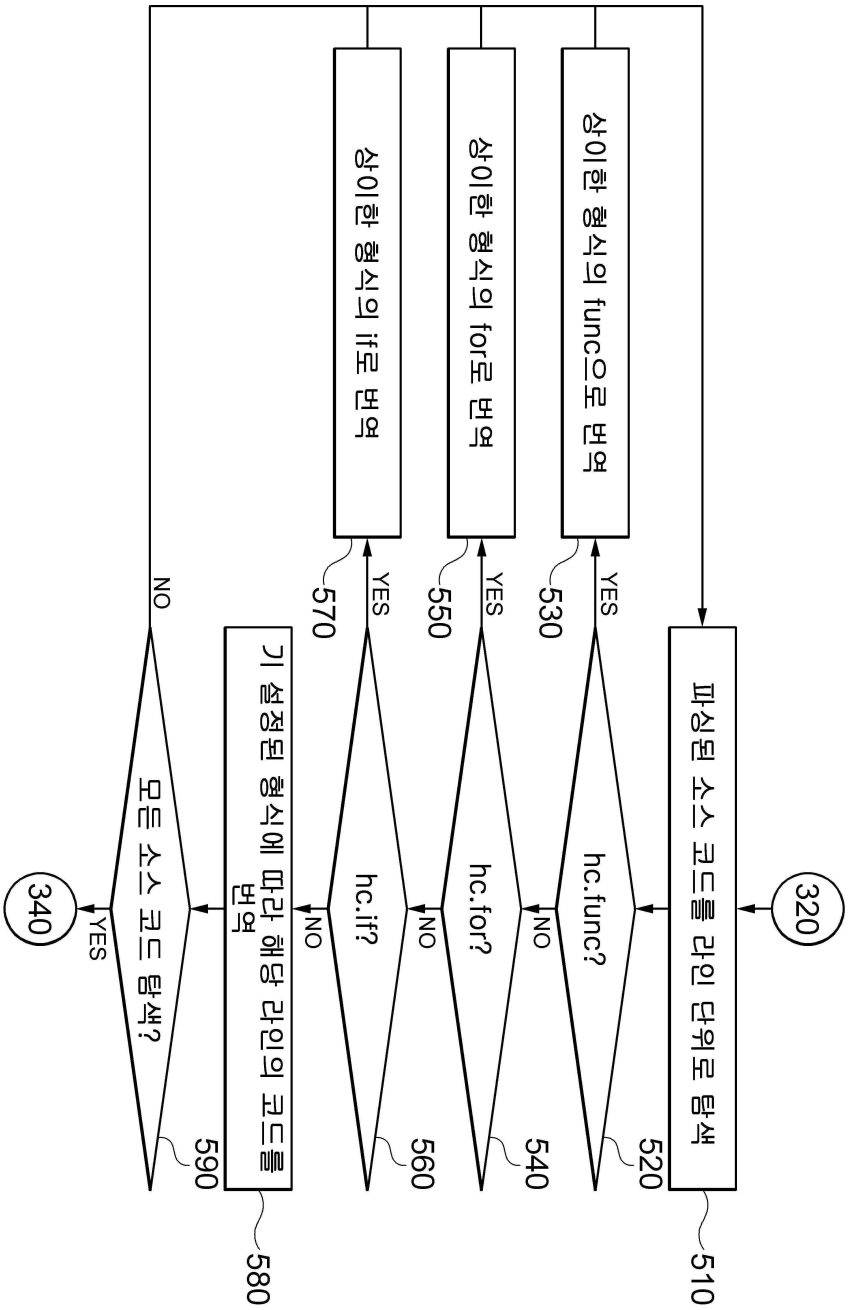
도면3


시 작
동적 언어로 기술된, 주석을 포함하는 소스 코드를 입력
310
소스 코드를 파싱
320
제1 중간 코드를 생성
330
제1 중간 코드를 파싱
340
동형 암호화된 제2 중간 코드를 생성
350
종 료

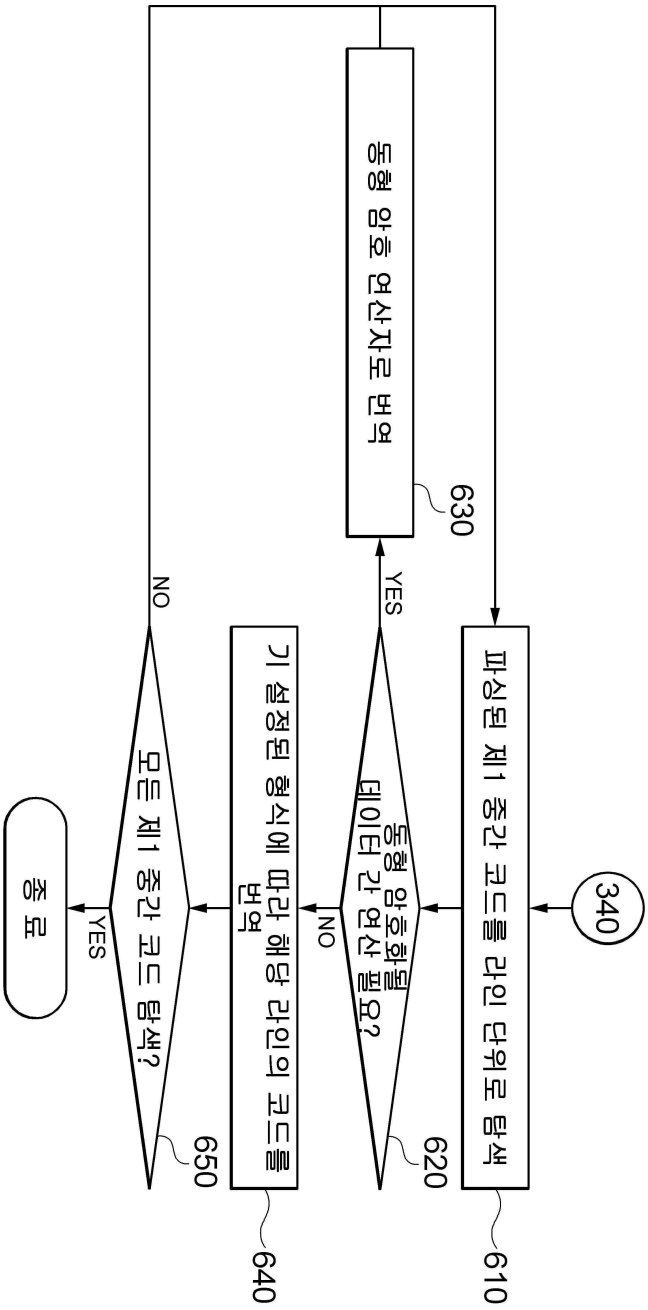
도면4


시 작
동적 언어로 기술된, 주석을 포함하는 소스 코드를 입력
410
소스 코드를 파싱
420
제1 중간 코드를 생성
430
제1 중간 코드를 파싱
440
연산이 동형 암호화 대상인가?
450
YES
NO
동형 암호화된 제2 중간 코드를 생성
460
제3 중간 코드를 생성
470
LLVM 중간 코드를 생성
480
동형 암호화된 입출력을 갖는 실행 프로그램을 생성
490
종 료

도면5


320
파싱된 소스 코드를 라인 단위로 탐색
510
hc.func?
520
상이한 형식의 func으로 번역
530
hc.for?
540
상이한 형식의 for로 번역
550
hc.if?
560
상이한 형식의 if로 번역
570
기 설정된 형식에 따라 해당 라인의 코드를 변환
580
모든 소스 코드 탐색?
590
340
YES
NO

도면6


340
파싱된 제1 중간 코드를 라인 단위로 탐색
610
620
동형 암호화를 필요?
데이터 간 연산
YES
NO
630
동형 암호 연산자로 변경
640
기 설정된 형식에 따라 해당 라인의 코드를 변경
650
모든 제1 중간 코드 탐색?
YES
NO
종료

도면7

700

```
import hecate as hc
@hc.func(“*5, 1 → *1”)
def example( a,b ) :
@hc.If (b >0 , “1”)
def abs ( condition ) :
  if condition :
    return b
  else :
    return -b
[b] = abs
return a ** [1, 2, 3, 4, 5] + b
hc.setMain(example)
module_name = hc.compile()
```

도면8

800

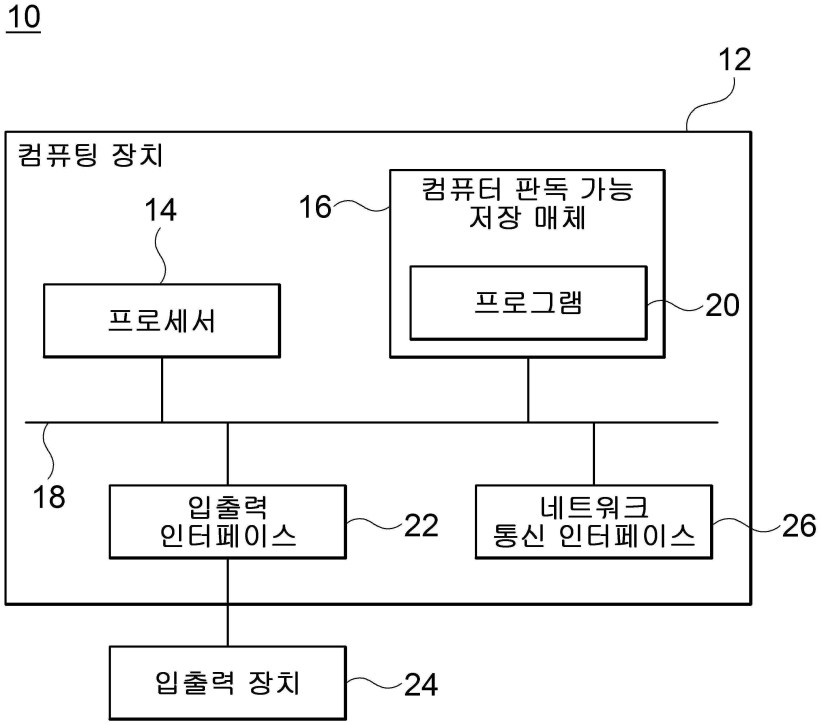
```
module {
func @example(%arg: !fe<"*5">, %arg1: !fe<"1">) → !fe<"*1"> {
%cst = "fe.constant" [0.0000e+00] : (!fe<"1">)
%0 = "fe.cmph" (%arg1, %cst) {predicate=5} : (!fe<"1">)
%2 = scf.if (%0) {
  scf.yield (%arg1)
} {
  %10 = "fe.negation" (%arg1) : (!fe<"1">)
  scf.yield (%10)
} : !fe <"1">
%c1 = "fe.constant" [1.000, 2.000, 3.000, 4.000, 5.000] : (!fe<"5">)
%3 = "fe.innerprod" %arg, %c1 : (!fe<"*1">)
%4 = "fe.add" %3, %2
return %4 : !fe<"*1">
} }
```

도면9

900

```
module {
#0 = he.level<2>
#2 = he.level<1>
func @example(%arg: !he.cipher<"5">, %arg1: f64) → !he.cipher<"1"> {
%cst = std.constant [0.0000e+00] : f64
%0 = std.cmp (%arg1, %cst) {predicate=5} : i1
%2 = scf.if (%0) {
  scf.yield (%arg1)
} {
  %10 = std.negf (%arg1) : f64
  scf.yield (%10)
} : f64
%c1 = std.constant [1.000, 2.000, 3.000, 4.000, 5.000] : 5xf64
%3 = he.encode (%c1) : (5xf64 → !he.plain<"5">)
%4 = he.multcp %arg, %3 : (!he.cipher<"5">)
%6 = he.rescale %4, 1 : (!he.cipher<"5">)
%7 = he.sum %6 : (!he.cipher<"1">)
%8 = he.encode(%2) : (!he.plain<"1">)
%9 = he.modswitch %8, 1: (!he.plain<"1">)
%10 = he.addcp %7, %9 : (!he.cipher<"1">)
return %10 : !he.cipher<"1">
} }
```

도면10


10
12
컴퓨팅 장치
14
프로세서
16
컴퓨터 판독 가능
저장 매체
프로그램
20
18
입출력
인터페이스
22
네트워크
통신 인터페이스
26
입출력 장치
24