(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2016년12월16일
(11) 등록번호 10-1687439
(24) 등록일자 2016년12월12일

(51) 국제특허분류(Int. Cl.)
G06F 21/12 (2013.01) *G06F 9/44* (2006.01)
(21) 출원번호 10-2012-7031855
(22) 출원일자(국제) 2010년07월22일
심사청구일자 2015년06월18일
(85) 번역문제출일자 2012년12월05일
(65) 공개번호 10-2013-0120985
(43) 공개일자 2013년11월05일
(86) 국제출원번호 PCT/EP2010/060603
(87) 국제공개번호 WO 2012/010205
국제공개일자 2012년01월26일
(56) 선행기술조사문헌
JP2009211292 A
JP2002353960 A
KR1020090049857 A
KR1020100069588 A

(73) 특허권자
나그라비젼 에스에이
스위스 체하-1033 세조-쉬르-로잔느 루뜨 드 쥬네브 22-24
(72) 발명자
맥체티, 마르코
스위스 체하-1033 체슈아-주르-로잔 뤼 데 라 메브레 12
쿠델스키, 헨리
스위스 체하-1071 쉐브레 체민 두 다일라드 35
(74) 대리인
김해중, 홍순우

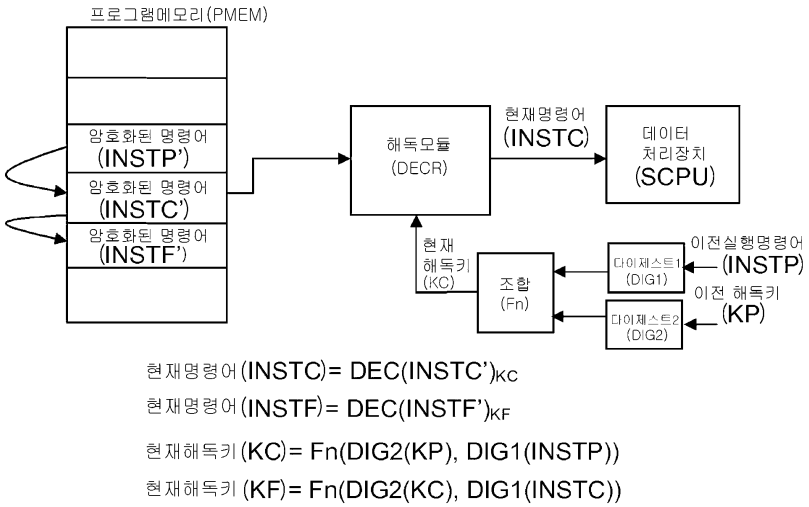
전체 청구항 수 : 총 12 항 심사관 : 구대성

(54) 발명의 명칭 **소프트웨어 무결성을 보장하기위한 프로세서 실행 방법**

(57) 요 약

본 발명은 하나 이상의 전부 또는 일부의 이전 명령어에 기준한 키를 사용하여 프로그램의 각 명령어의 전부 또는 일부를 암호화하여, 명령어마다 다른 암호화 키를 갖도록하여, 소프트웨어의 무결성을 보장하는 문제점에 대한 해결책을 제공한다. 본 발명은 본질적으로 트리형태가 아닌 구조의 소프트웨어 프로그램에 적용할 수 있을 뿐만 아니라, 프로그램이 루프, 점프, 호출 또는 분기 등을 포함하는 경우에도 적용할 수 있다. 명령어 키는 필요에 따라 적절히 초기 설정될 수 있으므로, 최초 명령어가 명확할 필요는 없다. 본 발명은 제3자가 해독된 명령어 또는 해독 키를 가로채지 못하도록, 소프트웨어 또는 완전히 하드웨어로서 구현될 수 있다.

대 표 도 - 도1



현재명령어(INSTC)= $DEC(INSTC')_{KC}$
현재명령어(INSTF)= $DEC(INSTF')_{KF}$
현재해독키(KC)= Fn(DIG2(KP), DIG1(INSTP))
현재해독키(KF)= Fn(DIG2(KC), DIG1(INSTC))

명 세 서

청구범위

청구항 1

초기 설정된 명령어 키를 이용하는, 프로그램 메모리 내에서 소프트웨어의 무결성 (integrity)을 보장하기 위한 프로세서 실행 방법에 있어서 여기서, 상기 소프트웨어는 복수의 암호화된 명령어를 포함하고, 상기 명령어는 적어도 하나의 연산부호를 포함함-;

현재 암호화된 명령어를 판독하는 단계,

명령어 키를 이용하여 현재 암호화된 명령어의 적어도 일부분을 해독하는 단계,

판독된 차회 암호화된 명령어가 갱신된 명령어 키로 해독되도록, 명령어 키의 현재 값의 다이제스트 (digest)와 현재 명령어의 다이제스트 (digest)에 근거한 계산을 이용하여 명령어 키를 갱신하는 단계, 및

현재 명령어를 실행하는 단계

를 포함하는 것을 특징으로 하는 프로그램 메모리 내에서 소프트웨어 무결성을 보장하기 위한 프로세서 실행 방법

청구항 2

제1항에 있어서,

프로그램 메모리에 있는 제1 명령어는 암호화되지 않은 것을 특징으로하는 프로그램 메모리 내에서 소프트웨어 무결성을 보장하기 위한 프로세서 실행 방법

청구항 3

제1항 또는 제2항에 있어서,

현재 명령어는 인증태그를 더 포함하며, 상기 인증태그는 실행되기 전에 상기 명령어를 인증하는데 사용되는 것을 특징으로 하는 프로그램 메모리 내에서 소프트웨어 무결성을 보장하기 위한 프로세서 실행 방법

청구항 4

제1항 또는 제2항에 있어서,

변경이 명령어 키에 만들어지며, 상기 변경은 실행 명령을 주는 상기 변경된 명령어 키를 사용하여 차회 암호화된 명령어의 해독을 허락하는 것을 특징으로 하는 프로그램 메모리 내에서 소프트웨어 무결성을 보장하기 위한 프로세서 실행 방법

청구항 5

제4항에 있어서,

현재 명령어는 변경을 행하는데 사용되는 변경 값을 더 포함하며, 상기 변경 값은 명령어 값에서 추출되고, 차회 암호화 키를 결정하는 동안 갱신 단계에서 작동하는 것을 특징으로하는 프로그램 메모리 내에서 소프트웨어 무결성을 보장하기 위한 프로세서 실행 방법

청구항 6

제1항에 있어서,

암호화된 명령어를 해독하거나, 명령어 키를 갱신하거나, 현재 명령어를 인증하거나, 또는 현재 명령어를 실행하는 어떤 또는 모든 프로세스는 보안 모듈 내에서 이루어지는 것을 특징으로 하는 프로그램 메모리 내에서 소프트웨어 무결성을 보장하기 위한 프로세서 실행 방법

청구항 7

제1항 또는 제2항에 있어서,

상기 다이제스트는 상기 현재 명령어의 전부 또는 일부에 적용되는 함수의 결과이며, 상기 함수는 논리 함수, 산술 함수, 암호 함수 또는 단 방향 함수로부터 선택되는 것을 특징으로하는 프로그램 메모리 내에서 소프트웨어 무결성을 보장하기 위한 프로세서 실행 방법

청구항 8

제1항 또는 제2항에 있어서,

명령어 키를 갱신하는 단계는 변경 값에 기준하는 것을 더 포함하며; 상기 변경 값은 명령어 키를 공지 값으로 가져가는데 사용되는 것을 특징으로 하는 프로그램 메모리 내에서 소프트웨어 무결성을 보장하기 위한 프로세서 실행 방법

청구항 9

제1항 또는 제2항에 있어서,

마스터 키는 명령어 키를 초기 설정하는데 이용되는 것을 특징으로 하는 프로그램 메모리 내에서 소프트웨어 무결성을 보장하기 위한 프로세서 실행 방법

청구항 10

암호화된 프로그램을 저장하기 위한 프로그램 메모리 (PMEM)와 프로그램 카운터 (PC)를 포함하는 장치에 있어서,

상기의 암호화된 프로그램은 복수의 암호화된 명령어들 (INST')을 포함하고,

상기 명령어들은 적어도 하나의 연산 부호를 포함하고,

상기 장치는 해독 모듈 (DECR)과 데이터 처리유닛(SCPU)을 더 포함하며,

상기 장치는 초기 설정된 명령어 키 (KI) 에 접근하고,

상기 장치는 적어도 하나의 이전에 실행된 명령어의 다이제스트와 상기 명령어 키의 전부 또는 일부분에 기준한 명령어 키 (KI)를 반복적으로 갱신하는 수단을 더 포함하는 것을 특징으로 하는 장치

청구항 11

제10항에 있어서,

명령어 키를 반복적으로 갱신하는 수단이 하드웨어에서 실현되는 것을 특징으로 하는 장치

청구항 12

제10항 또는 제 11항에 있어서,

명령어 키의 갱신은 변경 값에 기준하며, 상기 변경 값은 명령어 키를 공지 값으로 가져가는데 이용되는 것을 특징으로 하는 장치

발명의 설명

기 술 분 야

[0001] 본 발명은 소프트웨어 보호영역에 관련된 것이며, 더욱 구체적으로는 소프트웨어의 변조 방지를 통하여, 소프트웨어의 무결성을 보장하기 위한 장치와 수단에 관한 것이다

배 경 기 술

[0002] 안전한 데이터처리 영역에 있어서, 데이터 처리가 안전하게 이루어 질 수 있는 변조 방지 환경을 마련할 필요가 있다. 애플리케이션 보안문제를 다루기 위한 제1 접근방법은 관련 소프트웨어가 설치된 하드웨어를 가능한 한 안전하게 유지하는 데에 초점이 맞추어 졌었다. 과거에는 변조방지의 개념은, 하드웨어를 열기 어렵게 하거나, 또는 일단 오픈이 되면, 안전한 소프트웨어가 내장된 칩(chip)을 파괴하는 것이었다. 그러나, 애프리케이션 보안을 위한 소프트웨어 기술이 점차 다양해지고, 저 비용이 소요되고, 실제로 좋은 애프리케이션 보안은 소프트웨어가 변조되지 않도록 하는 것이라고 인식됨에 따라, 오늘날에는 소프트웨어 접근과 하드웨어 접근이 결합되어 사용되고 있다.

[0003] 애플리케이션이 구동되는 전형적인 시스템은 통상적으로 처리 유닛, 다수의 주변장치와 메모리로 구성되어 있다. 보안이 필요한 대부분의 경우에, 암호화 방식이 사용된다. 암호화 방식에 있어서, 안전을 필요로 하는 정보, 예를 들면, 작업 데이터 또는 실행 코드와 같은 정보는 암호화 된다. 암호화는 통상적으로 시스템의 일부를 이루고 있는 보안 모듈에서 이루어진다. 보안 모듈은, 마이크로프로세서 카드, 스마트 카드 또는 배지나 키의 형상으로 된 전자 모듈 등의 다양한 방법으로 사용될 수 있다. 상기의 모듈들은 일반적으로 운반하기 쉽고 리시버로부터 분리가능하며 변조가 방지되도록 디자인되어 있다. 가장 통상적으로 사용되는 형태는 전기적 접촉부를 가지나, ISO 14443 타입의 무접촉 버전도 존재한다. 또 다른 실시예로서, 보안 모듈은 리시버 내부에 직접 납땜되거나, 변형된 형태로 SIM 모듈과 같이 소켓이나 컨넥터 상의 회로로도 존재한다. 그러나, 또 다른 실시 예는 보안 모듈을, 예를 들면 디코더의 디스크램블 모듈 또는 마이크로프로세서 모듈에서 다른 기능을 갖는 칩에 통합시키는 것이다. 보안 모듈은 또한 소프트웨어로 실행될 수 있다.

[0004] 오늘날의 안전한 프로세싱 시스템에서, 보안 모듈과 첨단의 암호화 기술이 사용되고 있음에도, 시스템들의 보안을 깨트리기 위한 시도가 계속되고 있다. 그러한 시스템에서 보안을 깨트리기 위하여 사용되는 기술들에는, 예를 들면, 관련된 하드웨어의 리버스 엔지니어링 또는 그 시스템에 사용되는 소프트웨어의 동적 분석 또는 정적 분석과 그 소프트웨어를 이용한 변조기술을 포함한다. 정적 분석이란 비실행코드의 분해 또는 역컴파일(decompilation) 형태를 의미한다. 동적 분석은 코드가 실행되는 동안 즉, 소프트웨어가 실행되는 동안 어떤 신호들을 관찰함으로써, 분석하는 것을 의미한다. 그러한 분석들은, 예를 들면 현재 조건들이 그러한 실행을 규정하지 않았을 때 무조건 건너뜀 (unconditional jump)이 조건부 건너뜀 (conditional jump) 대신에 유도되어 분기가 실행되는 분기 끼워넣기 공격이 실행되어, 소프트웨어가 변경되어 변조될 수 있다. 특히, 그러한 공격은 프로그램으로 하여금, 예를 들면, 일련번호 또는 비밀번호 조사와 같은 인증 단계를 회피하도록 한다.

[0005] 공학석사학위 논문으로 2005년 제출된, Tamper-Resistance for Software Protection 이라는 제목의 논문에서. Ping Wang 은 소프트웨어 프로그램이 프로그램의 흐름에 따라서 다수의 독립 블록으로 나누어지는 다중 블록 암호화 기술을 기재하였다. 그 프로그램의 각각의 블록은 암호화되고, 각각의 블록은 서로 다른 암호화 키를 가진다. 각 블록의 암호화 키는 프로그램의 흐름에 따라 이전 블록의 해시 값(hash value)이다. 상기 기술은 하나의 블록이 다른 블록을 이끄는 계층구조 형태로 블록이 배열된 트리같은 구조를 갖는 프로그램에서 사용된다. 상기 기술에서, 실행되는 제1 블록은 명확해야 한다. 복호화 루틴을 호출하는 코드가 각각의 블록 내부에 위치하며, 동적 무결성 검사를 실행하는 프로그램 컨트롤러는 프로그램의 끝에 부가된다. 만약 허가받지 않은 사람이 프로그램의 일부를 변경하려고 시도하면, 그 프로그램의 변경된 부분을 포함하는 블록의 해시값 (hash value) 이 상이하여 그 다음 불럭은 규정에 맞도록 해독되지 않아, 그 프로그램은 파괴된다.

[0006] 이러한 구조는 각각의 블록이 두 번 판독되므로 손해이다. 또한 한 개의 해독 키는 전체 블록에서 유효하므로,

암호화가 명령어에서 순차적으로 이루어지는 것이 아니라, 블록에서 순차적으로 이루어진다는 점에서 손해이다. 이것은 한 개의 키 발견이 소프트웨어의 전체 블록을 공격에 취약하도록 하는 것을 의미한다. 가장 적은 가능한 블록의 크기는 루프를 포함하는 가장 작은 블록에 의하여 결정된다. 왜냐하면 이 디자인에 있어서, 정의에 의하여 한 개의 블록은 루프의 전체를 포함하여야 하기 때문이다. 루프가 없는 경우에 프로그램이 블록당 한 개의 명령어로 될 수 있다 하여도, 그 방법을 실행하는데 발생하는 오버헤드 시간은 크기와 실행 속도에 따라 최종 결정된다. 더욱이, 프로그램의 무결성을 유지하기 위하여, 블록에서 만들어지는 변화에 따라 적절한 해시값을 계산하는 방법으로, 블록에서 변경이 이루어지고, 그에 해당하는 변화가 프로그램 컨트롤러에서 이루어진다면, 사람들은 가능한 공격을 상상할 수 있다.

[0007] 본 발명에서는 실행 코드가 암호화된 포맷으로 존재하도록 하며, 암호화가 명령어에서 차례로 (instruction by instruction basis) 이루어지고, 명령어들이 두 번 판독되지 않는다. 그 방식은 암호화 키가 쉽게 인터셉트될 수 있는 곳에서는 결코 나타나지 않는, 고유의 장점을 가진 하드웨어에서 완벽하게 실현된다. 소프트웨어 오버헤드가 없으므로 실행속도가 굉장히 빨라진다. 종래기술에서는 다음 블록을 위한 암호화 키가 단지 이전 블럭의 내용에만 의존하였으나, 본 발명에서는 암호화 키가 다수의 이전 암호화 키 값에 의존할 수 있다. 예를 들면, 다음 명령어를 해독하는 키는 두 개의 이전 명령을 위한 축적된 키들과 결합된 현재 명령에 근거할 수 있다.

발명의 내용

해결하려는 과제

[0008] 본 발명의 목적은, 해결책을 찾기 위하여 오버헤드를 최소화하고, 다양하게 하며, 많은 다른 구조 형태의 소프트웨어를 사용하는 시스템에서도 적용할 수 있으며, 소프트웨어의 분석과 그에 따른 변조에 의하여 발생하는 보안의 문제를 해결하고자 하는 것이다.

과제의 해결 수단

[0009] 본 발명은 프로그램 메모리에서 소프트웨어의 무결성을 확보하기 위한 프로세서 구현 방법을 사용하여 이루어지며, 상기 소프트웨어는 다수의 암호화된 명령어들을 포함하며, 명령어는 적어도 한 개의 연산 부호를 포함하며, 상기의 방법은 초기 설정된 명령어 키를 사용하며, 아래의 단계들을 포함한다:

[0010] - 현재 암호화된 명령어 판독 단계

[0011] - 명령어 키를 사용하여 현재 암호화된 명령어를 해독하는 단계

[0012] - 판독되는 차회 암호화 명령어가 갱신된 명령어 키로 해독되기 위하여, 명령어 키의 현재값과 현재 명령어의 다이제스트 (a digest)에 따라 계산하여 명령어 키를 갱신하는 단계,

[0013] - 현재 명령어 실행 단계

발명의 효과

[0014] 본 발명은 본래 트리형태가 아닌 구조의 프로그램에 적용 될 수 있으며, 소프트웨어 또는 완전하게 하드웨어로 구현될 수 있고, 그것에 의하여 해독된 명령어 또는 해독 키를 제3자가 가로챌 수 있는 가능성을 차단한다.

도면의 간단한 설명

[0015] 본 발명은 아래에 기재된 구체적 실시예의 상세한 설명을 첨부된 도면들과 연관하여 읽을 때, 가장 이해가 잘 될 것이다.

도 1은 본 발명의 일실시예의 간략한 블록 다이어그램이다.

도 2는 본 발명의 일실시예의 흐름도이다.

도 3은 본 발명의 일 실시예에 따라 소프트웨어 점프(jump) 또는 분기들(branches)이 어떻게 다루어지는 지를 보여주는 간략한 블록 다이어그램이다.

발명을 실시하기 위한 구체적인 내용

[0016] 상기한 바와 같이, 본 발명의 목적은, 명령어가 순차적으로, 안전한 프로세서에서 소프트웨어가 암호화 형태로 메모리에 저장되고, 해독되고, 실행되는 안전한 방법으로, 소프트웨어를 구동하는 수단을 제공하는 것이다. 현재 명령어의 해독 키는 해독된 적어도 한 개의 선행 명령어에 의존하는 반면, 후행 명령어의 해독은 현재 명령어의 정확한 해독에 의존한다. 이러한 방법으로, 소프트웨어의 무결성을 보장하기 위한 자기 검사 수단이 얻어진다. 소프트웨어의 단순한 성공적 실행은 흐름이나 내용이 변경되지 않도록 하는 것이다. 왜냐하면, 명령어의 변경은 후행 명령어를 해독하는 능력을 무효화시켜, 프로그램을 조기 종료시키거나, 또는 적어도 프로그램 실행 추적을 방해하기 때문이다. 본 발명에 사용되는 방법은 제3자가 명령어들을 인터셉트하거나 또는 관련된 해독 키들이 인터셉트되지 않도록 하기 위하여, 소프트웨어에 의하여 구현될 수 있으나, 순전히 하드웨어에 의하여 구현될 수도 있다. 본 발명은 종래기술의 해결방법과 비교하여 거의 오버헤드가 없다. 이 방법은 점프나 브레이크를 포함하는 다양한 다른 아키텍쳐 또는 구조들의 소프트웨어에 적용될 수 있으며, 트리 구조로 알려진 구조들에 한정되지 아니한다.

[0017] 그러므로, 본 발명은 시스템에서 변조 방지 소프트웨어 실행을 보장하기 위한 방법을 제공하는 것이며, 상기의 시스템은 적어도 암호화된 프로그램 명령어 (INSTP, INSTC, INSTF)를 가지는 프로그램 메모리 장치, 상기의 프로그램 명령어들을 해독하기 위한 해독 모듈(DECR), 해독된 프로그램 명령어들 (INSTP, INSTC, INSTF)을 실행하기 위한 데이터 처리유닛 (SCPU), 암호화된 프로그램 명령어들을 해독하기 위하여 명령어 키 (KP, KC, KF)로 알려진 해독 키를 만드는 수단을 포함한다. 물론, 명령어 키들을 만드는 수단은 데이터 처리장치 내부에 위치할 수 있다. 해독(복호화) 모듈과 데이터 처리장치는 바람직하게는 종래기술에서 잘 알려진 타입의 보안 모듈에 위치한다.

[0018] 암호화된 프로그램을 실행하는 동안에, 현재 암호화된 명령어 (INSTC)는 프로그램 메모리 (PMEM)으로부터 판독되고, 도 1에서 보여지는 바와 같이, 한편으로는 이전 복호화 (KP)의 다이제스트 (digest)와 다른 한편으로는 이전 실행명령어 (DIG(INSTRP))의 다이제스트 (digest)의 조합 (Fn)으로 만들어진 현재 해독 키 (KP)을 사용한 현재 명령어를 주기 위하여 해독된다. 다이제스트(digest)는 피연산자의 전부 또는 일부에 적용하여 출력을 발생시키는 연산을 의미한다. 다이제스트는, 피연산자에서 수행될 때, 피연산자와 동일한 출력을 낼 수 있다. 본 발명의 일 실시예에 따라, 다이제스트는 피연산자에서 단방향 함수를 포함한다. 이것은 제3자가 되돌아가서 이전 키 또는 이전 명령어를 추론하지 못하도록 한다. 해시함수는 그러한 단 방향 함수 (예를 들면 SHA2, MD5) 의 한 예이다. 조합 (combination)이란 논리적 또는 산술적 또는 암호적인지에 관계없이 이미 언급된 피연산자의 모든 형태의 조합을 의미한다. 이러한 방식으로,프로그램의 흐름과 내용이 보장된다, 왜냐하면, 만약 현재 암호화된 명령어들이 프로그램 제작자에 의하여 의도된 명령어가 아니면, 현재 암호화된 명령어를 해독할 때 사용되는 현재 해독 키 (FC)는 어떤 다른 의도하지 않은 값을 산출할 것이기 때문이다. 이러한 방법으로, 소프트웨어의 무결성은 단지 성공적인 실행에 의하여 보장되므로, 우리는 자체 확인을 하는 소프트웨어를 얻을 수 있다. 만약 소프트웨어가 변조되었다면, 그것은 실행되지 않을 것이다

[0019] 도 2는 상기의 본 발명의 일실시예를 보여주는 흐름도로서, 현재 해독 키를 갖는 현재 명령어과 이전 키를 갖는 이전 명령어 등을 말하기 보다는, 시간에 따른 스냅샷 (snapshot) 관점에서 발명을 설명하며, 각각의 명령어가 실행될 때 갱신되는 명령어 키 (KI)만에 관한 것이다. 처리유닛에서 프로그램 카운터 (PC)는 실행되는 차회 명령어의 위치를 가리킨다. 프로그램 카운터 (PC)는 명령어 실행에 따라 증가하거나, 만일 상기 명령어가 단순한 증가보다는 다른 갱신 형태를 지시한다면 달리 갱신된다. 예를 들어, 만약 명령어가 레지스터로부터 어떤 값을 로딩하라는 명령 (command)을 포함하면, 프로그램 카운터(PC)는 다음의 위치를 지시하기 위하여 단순히 증가할 것이다. 그러나, 만약 명령어가 어떤 위치의 점프를 포함한다면, 점프에 의하여 지시되는 위치값으로 갱신될 것이다.

[0020] 프로그램 카운터 (PC)와 명령어 키 (KI)는 먼저 초기 설정된다(INI PC, INI KI). 암호화된 명령어는 프로그램카운터 (RD INST c.f. PC)에 의하여 지시되는 위치에 있는 프로그램 메모리로부터 판독되어, 명령어 키 (DCPT INST, KI)를 사용하여 해독된다. 명령어는 실행되고 (EX INST), 프로그램 카운터는 단순 증가하거나, 명령어가 지시한대로 새로운 값으로 치환되어 갱신된다. 명령어 키는 실행 명령의 다이제스트를 사용하여 갱신된다 (UPD KI, INST). 그러므로 명령어 키의 갱신은 방금 실행된 명령어뿐만 아니라 명령어를 해독하기위하여 사용된 키의 값도 고려한다. 차례로, 먼저 이전 명령어를 해독하기 위하여 이전에 사용된 명령어 키는 이전 명령어와 그 이

전에 명령어를 해독하는데 사용된 명령어 키로부터 만들어진다. 이 방법으로 명령어 키의 값은 지난 실행 명령뿐만 아니라 결합된 이전의 모든 실행 명령에 의존한다. 실제로, 본 발명의 일 실시예에 있어서, 명령어 키의 갱신은 지난 실행 명령의 값과 적어도 2개의 이전 실행 명령어 값들을 참작한다. 예를 들어, 해독 명령어 4의 키는 명령어 3의 다이체스트 , 명령어 2의 다이제스트와 명령어 1의 다이제스트의 조합일 수 있다

[0021] 도 2에 도시된 바와 같이, 본 발명의 방법은 명령어 키가 이전의 실행 명령어를 사용하여 갱신되는 루프를 포함한다. 이것은 프로그램에서 제1 명령어를 해독하는 방법에 대한 질문으로 안내한다. 만약 이전의 실행 명령어가 없었다면, 어떻게 제1 명령어 키가 계산될 수 있겠는가? 본 발명의 일 실시예에서, 다른 모든 명령어는 암호화되지만, 프로그램의 제1 명령어는 명확히 남는다. 제1 명령어는 직접 실행되며, 그것에 의하여 루프을 시작하고, 제2 명령어는 제1 명령어에 따라 명령어 키를 사용하여 해독된다. 본 발명의 다른 실시예에서, 전체 프로그램은, 제1 명령어를 포함하여, 암호화되며, 명령어 키는 제1 명령어를 해독하는 값을 사용하여 초기 설정된다. 이 값은 보안모듈에 구축되거나, 또는 달리 외부 보안모듈에 전달되는 마스터 키가 될 수 있다.

[0022] 프로그램을 실행하는 동안, 현재 메모리 위치(C)에 있는 현재 명령어(INSTC)가 하나 이상의 이전 명령어들(INSTP1, INSTP2)에 의하여 참조되어 여러 가지 상황이 발생한다. 즉, 현재 명령어, 또는 피호출자(callee)는, 예를 들면, 분기형 명령어 (예를 들어, 점프, 분기, 호출을 포함)을 만날 때, 하나 이상의 호출자(caller)에 의하여 참조될 수 있다. 도 3은 두 개의 호출자 (INSTP1, INSTP2)가 하나의 피호출자 (INSTC)를 언급하는 상황을 설명한다. 이 경우에 명령어 키의 2개의 다른 값이 다른 가능성있는 히스토리의 관점에서 가능하므로, 어느 키가 암호화된 피호출자를 해독하는데 사용되느냐에 따라 2개의 다른 결과가 유도된다. 이것은 물론 바람직한 현상은 아니다 왜냐하면, 암호화된 피호출자는 하나의 키에 의하여만 암호화될 수 있기 때문이다. 이러한 문제점을 피하기 위하여, 피호출자를 정확히 해독하는 필요한 값으로 결과 명령어 키를 강제하기 위하여. 변경(CORR1, CORR2)이 계산된다. 예를 들면, 위치 C에 있는 피호출자 레지던트는 P1과 P2위치에 있는 2개의 다른 호출자 레지던트 (callers resident)에 의하여 참조된다. 위치 C (INSTC) 에 있는 암호화 명령어를 적절히 해독하는 데 필요한 명령어 키는 KCin. 이다. 그러나, P1 (INSTP1) 에서 명령어 레지던트 (instruction resident)의 실행에 따른 명령어 키의 값은 KP1out 이며, P2 (INSTP2)에서 명령어 레지던트의 실행에 따른 명령어 키의 값은 KP2out 이다. 더욱이, KP1out 은 KP2out과 동일하지 않고, KP1out 또는 KP2out 어느 것도 KCin과 동일하지 않다. 그러므로, 분기 형태의 명령어가 실행될 때는 언제나, 명령어 키 값이 필요한 값을 갖도록, 변경 (CORR1, CORR2)이 필요하다. 피호출자를 해독하는데 필요한 키의 값이 알려지고 (즉 KCin), 호출자의 실행에 따른 키의 값이 알려졌기 때문에, 계산에 사용될 때 변경 값은 명령어 키를 필요한 값으로 가져갈 것이라는 점에서, 각각의 호출자에 대한 변경 값을 예견하는 것이 가능하다. 이러한 명령어가 사용될 때마다 명령어 키에 필요한 변경을 하기 위하여 각각의 분기 형태의 명령어에서 적절한 변경 값이 실행된다- 호출자마다 다른 변경이 만들어 진다. 본 발명의 일 실시예에 따르면, 상기에서 설명한 바와 같이 이전 해독 키와 이전 명령어의 다이제스트의 조합에서 변경 값이 피연산자로서 된다.

[0023] 상기에서 설명한 명령어 키의 변경이 어떻게 이루어지는지의 한 예로서, 점프 명령어를 고려한다. 본 발명의 구체적인 실시예에서, 점프 명령어는 목적지 매개변수(destination parameter)를 포함하고, 또한 JMP C, #CORR1 같은 변경 매개변수 (modification parameter)를 더 포함한다. 이전 명령어 키와 전부 또는 부분의 이전 명령어의 조합에서 변경 값 (#CORR1)은 추가 매개변수 (additional parameter)로 사용된다. 명령어로부터 변경 값을 추출하여, 그것을 조합단계에서 여분의 매개변수 (extra parameter)로 사용하는 것보다, 점프 명령어의 다이제스트가 변경 값을 먼저 고려할 수 있다는 것을 지적할 필요가 있다. 상기에 설명한 형태의 변경된 점프 명령어를 통하여 프로그램이 실행될 때, 명령어 키의 상태를 아래의 표 [T1]이 설명한다. 상기 표는 명령어를 해독하는 데 필요한 키의 값과, 명령어 실행에 따른 키의 값과 새로운 키의 계산을 포함한다. label 1에서 명령어를 해독하는 데 필요한 키의 값이 공지이므로, 변경되지 않은 값인 K4 또는 K14를 필요한 값 K91으로 가져오기 위하여, 적절한 정정 값인 CORR1 또는 CORR2이 계산될 수 있다.

[0024] [T 1]

필요한 키	라벨 (Label)	명령어	결과 키
K1		명령어1	K2
K2		명령어2	K3
K3		점프 라벨1, 변경값1	K91=fn(K4,변경값1)
K11		명령어11	K12
K12		명령어12	K13
K13		점프 라벨1, 변경값2	K91=(Fn(K14, 변경값2)
K91	라벨1 (Label1)	명령어91	K92
K92		명령어92	K93

[0025]

[0026] 본 발명의 다른 실시예에서, 예를 들면 변경된 점프 명령어 대신에, 표준 점프 명령어 (standard jump instruction)가 사용되고, 상기에서 설명한 명령어 키에 대한 변경은 변경 값을 매개변수로 갖는 전용 변경 명령어 (dedicated modifying instruction)에 의하여 이루어진다. 그러한 변경 명령어의 함수는 변경 값에 의한 명령어 키에 직접 작용한다. 변경 명령어는 분기 또는 점프형 명령어 바로 앞에 놓여져, 피호출자를 적절히 해독하기 위하여 명령어 키가 적절하게 갱신되도록 한다. 상기에서 상술한 바와 같이 변경 함수는 사실상 명령어 키의 값에 원하는 변경 연산을 수행하도록 만들어진 복수의 명령어들이다. 예를 들면, 만약 피호출자를 적절히 해독하는데 필요한 명령어 키의 값이 #39 이라면, 점프 명령어를 말하기 직전에, 변경 값 (CORR1)을 발견하는 것은 #39를 가진 명령어 키 (K1)의 XOR 이고, CORR 과 K1의 합산기가 새로운 (정정된) K1 값을 준다.

[0027] 다른 실시 예에서, 피호출자 " 라벨1(label1)"에서 명령어 키는 K91 값을 갖는다. 프로그램 흐름이 다른 경로로부터 도달할 수 있다는 사실 때문에, 정정 명령어 Inst_ CORR가 점프 직전 가산되어, 명령어 키는 미리 정해진 값 K90으로 갱신된다. 이 경우에 점프인 분기형 명령어의 실행은 명령어 키를 K90에서 K91로 변경한다. 아래의 표 [T2]에서 명백한 바와 같이, 정정 명령어(Inst_ CORR)와 관련된 정정 값 (C1,C2)은 현재 명령어 키 (K3, K13)를 미리 정해진 값 K90으로 변경하고자 한다. 그 결과로 점프의 실행은 명령어 키를 K90에서 피호출자에서 명령어를 해독하는데 사용한 값인 K91로 갱신될 것이다(label 1).

[0028] 분기형 명령어 (branch-type instruction)가 다른 값을 가질 경우, 즉 명령어가 짧은 분기 BRA일 때, 이 명령어에 의하여 만들어진 다이제스트는 점프 명령어에 의하여 만들어진 다이제스트와는 다를 것이다. 그 결과로, 정정 명령어 Inst_ CORR에 부착된 정정 값 C3는 차이를 고려해야 하며, 분기 명령어가 실행하는 동안 명령어 키는 점프 명령어와 동일하지 않을 것이다. 그러나, 정정 값 C3 때문에, 짧은 분기 명령어가 실행된 후 최종 값은 그래도 K91일 것이다.

[0029] [T 2]

필요한 키	라 벨	명령어	결과 키
K1		명령어1	K2
K2		명령어2	K3
K3		정정명령어, 정정값 C1	K90=Fn (C1,K4)
K90		점프 라벨1	K91
K11		명령어11	K12
K12		명령어12	K13
K13		정정명령어. 정정값 C2	K90=Fn (C2,K14)
K90		점프 라벨1	K91
K20		명령어1	K2
K21		명령어2	K3
K22		정정명령어, 정정값 C3	K80=Fn (C3,K23)
K80		BRA 라벨1	K91
K91	라벨1	명령어91	K92
K92		명령어92	K93

[0030]

[0031] 아래의 표 [T3]는 프로그램 실행이, 2개의 다른 목적지 label1 과 label2 가 조건부 점프의 실행에 뒤이어 가능한, 조건부 점프 명령어를 통하여 진행될 때 명령어 키들의 상태를 설명하고 있다. 이 경우에 2개의 목적지에서 명령어들을 해독하는데 필요한 키는 동일해야 한다. 표는 명령어를 해독하는데 필요한 키의 값과, 명령어의 실행에 따른 키의 값과 새로운 키의 계산을 포함한다.

[0032] [T 3]

필요한 키	라 벨	명령어	결과 키
K91	라벨1 (L1)	명령어1	K2
K2		명령어2	K3
K3		변경= 정정값 C1	K90
K90		점프 COND L1, L2	K91
K11		명령어11	K12
K12		명령어12	K13
K13		점프 L2, 변경값2	K91
K91	라벨2 (L2)	명령어91	K92
K92		명령어92	K93

[0033]

[0034] 하나의 피호출자가 다수의 호출자에 의하여 참조되어 피호출자를 적절히 해독하기 위하여 명령어 키에 변경이 필요한 또 다른 상황은 함수 호출 또는 서브루틴 호출이다. 통상적으로, 이러한 호출 동안에, 함수와 서브루틴 안에서 가능한 다른 흐름을 증가시키고 그 결과로 함수 또는 서브루틴의 실행에 따른 가능한 결과의 수를 증가시켜, 호출동안 매개변수들은 통과할 수 있다. 그러한 호출이 마주칠 때, 명령어 키에 변경이 발생이 되어, 그 상태는 함수 또는 서브루틴의 초기에 알려지고, 호출로부터의 회신, 즉, 함수 또는 서브루틴으로부터 오기 바로 전에 더 많은 변경이 이루어진다

[0035] 본 발명의 내용에서, 상기에서 설명한대로 변경은 하나의 키를 다른 키로 단순히 대체할 수 있다는 것은 지적할 만한 가치가 있다.

[0036] 데이터 처리기술에서, 통상의 지식을 가진 자에게 잘 알려진 바와 같이, 명령어는 실행되는 연산을 정의하는 적어도 하나의 연산부호{opcode)를 포함한다. 명령어는 이것 이상 더 포함할 수 없거나, 또는 연산이 실행되는 하나 또는 다수의 피연산자{operand)를 더 포함할 수 있다. 연산 부호와 피연산자 또는 피연산자들에 더하여, 명령어는 명령어의 유효성을 체크하는 방법으로 사용되는 무결성 형상 (integrity figure)로 알려진 인증 태그을 포함할 수 있다. 결과적으로, 본 발명의 다른 실시예에서, 상기에서 설명된 바와 같이, 명령어를 실행하기 전에, 명령어는 인증 태그를 사용하여 먼저 확인될 수 있다. 인증 태그는 체크섬 (checksum)의 형태 또는 연산부호와 피연산자의 전부 또는 일부의 해시값의 형태를 가질 수 있다. 대부분의 경우에, 인증태그는 연산부호의 사인으로 간주될 수 있다. 그러므로, 전부 또는 일부의 명령어를 암호화하는데 있어서, 우리는 단지 연산부호를 암호화 할 것이지, 또는 인증 태그를 가진 조합 코드를 암호화 할 것인지 또는 피연산자도 포함을 할 것인지의 선택에 직면한다. 그러나, 제3자에게 본 발명의 내용을 감출 필요가 있는 경우에, 상기의 어떤 조합도 효과를 발휘한다. 인증 태그는 연산부호가 무엇인가에 대한 실마리를 잠재적 공격자에게 제공할 수 있기 때문에 본 발명은 연산부호와 인증 태그를 암호화시키는 데 도움이 된다. 그러므로 본 발명의 실시예에서 사용하는 방법은, 현재 암호화된 명령어를 판독하는 단계; 현재 암호화된 명령어와 인증 태그를 해독하기 위하여 명령어 키를 사용하는 단계; 추출된 인증 태그를 확인하는 단계; 판독된 다음의 암호화된 명령어가 갱신된 명령어 키로 해독되기 위하여, 명령어 키의 현재 값 (또는 다이제스트의 현재값) 과 현재 명령어의 다이제스트에 근거한 계산을 행하여 명령어 키를 갱신시키는 단계, 인증 태그가 유효한 경우에 현재 명령어를 실행하는 단계를 포함한다. 만일 인증 태그가 유효하다고 판정되지 아니하면, 프로그램은 적절한 경고음을 발생하면서 종료할 수 있다.

[0037] 본 발명의 목적은 제3자가 소프트웨어를 복사하는 것을 방지 할 뿐만 아니라, 제3자가 소프트웨어를 변경하고자 하는 시도가 감지되지 않고 소프트웨어를 변경하는 것을 방지하고자 하므로, 명령들의 연산 부호가 명확히 있으며, 인증 태그가 암호화되어 있는 경우에만 본 발명의 실시예가 존재한다. 이것은 본 발명에 의하여 소프트웨어 무결성을 보장하고자하는 목적을 달성하는데 필요하다. 또한 또 다른 실시 예에서, 만일 피연산자들이 있다면, 피연산자만 암호화하는 것이 가능하다. 마찬가지로, 어떠한 연산부호, 피연산자 또는 인증 태그 또는 그것들의 어떠한 조합의 암호화도 가능하다.

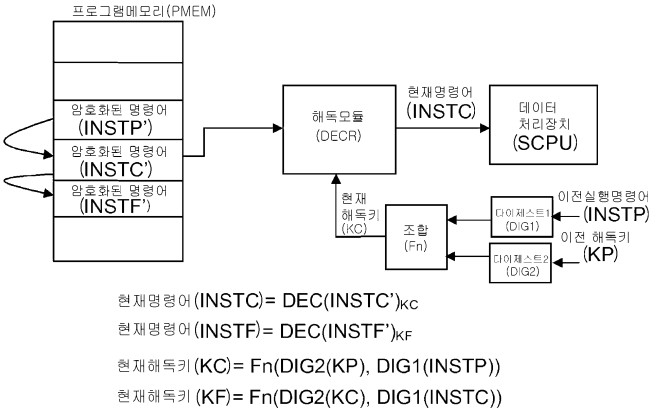
[0038] 마찬가지로, 본 발명의 실시 예에서 연산부호와 피연산자를 명확히 유지하고, 인증 태그만을 암호화하는 것이 가능하다. 점프의 경우에, 상기에서 설명한대로 변경 값을 사용하는 대신에, 점프 명령어 이후에 인증 태그의 검사를 불활성하게 할 수 있다. 이러한 솔루션의 이점은 점프 명령어가 변경 값을 필요로 하지 않는 다는 것이다.

[0039] 그러므로, 본 발명은 하나 또는 다수의 이전 명령어에 근거한 키를 사용하여, 명령어마다 다른 암호 키를 사용하는 프로그램의 각각의 명령어의 전부 또는 일부를 암호화하여, 소프트웨어 프로그램의 무결성을 보장하는 문제에 대한 해결책을 마련해 준다. 본 발명은 본질적으로 트리형태가 아닌 구조의 소프트웨어 프로그램에 적용할 수 있을 뿐만 아니라, 프로그램이 루프, 점프, 호출 또는 분기 등을 포함하는 경우에도 적용할 수 있다. 본 발명은 암호화 명령어가 잘못 해독되는 경우에도 그 예외를 허용한다. 명령어 키는 요청하는 대로 적절히 초기 설정될 수 있으므로, 최초 명령어가 명확할 필요는 없다. 본 발명은 제3자가 해독된 명령어 또는 해독 키를 가로채지 못하도록, 소프트웨어 또는 완전히 하드웨어로서 구현될 수 있다.

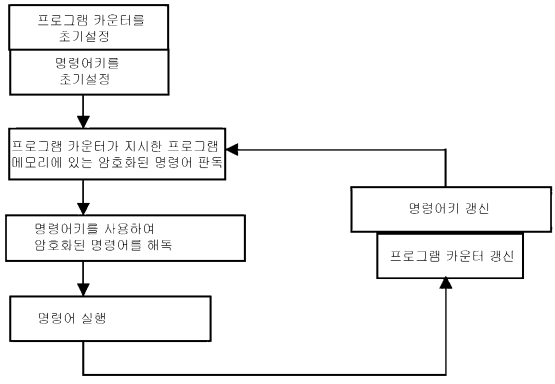
[0040] 명령어의 암호화는 비트 역전, 비트 이동, 비트 교환, 패리티 알고리즘, 순환중복코드와 같은 변환기, 흐름 암호와 블록 암호 같은 다양한 범주의 암호 알고리즘을 사용할 수 있다.

도면

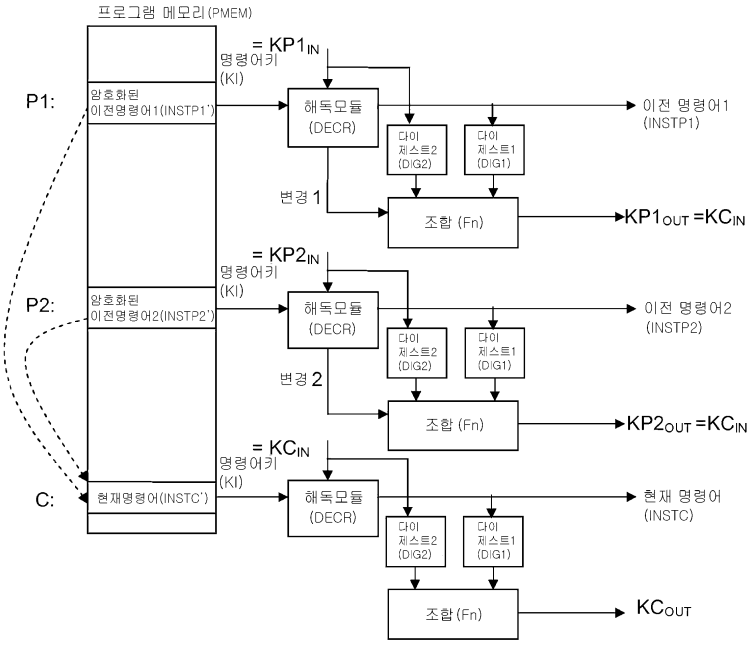
도면1



도면2



도면3



KC_{IN} : 위치C에 있는 암호화된 명령어 (INSTC')를 해독하도록 요구하는 명령어키

$KP1_{OUT}$: P1에서 명령어 레지던트의 실행에 따른 명령어키의 값

$KP2_{OUT}$: P2에서 명령어 레지던트의 실행에 따른 명령어키의 값