



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2014-0148153

(43) 공개일자 2014년12월31일

(51) 국제특허분류(Int. Cl.)

G06F 21/14 (2013.01)

(21) 출원번호 10-2013-0071695

(22) 출원일자 2013년06월21일

심사청구일자 2013년06월21일

(71) 출원인

고려대학교 산학협력단

서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)

(72) 발명자

이동훈

서울 종로구 사직로8길 34, 1404호 (내수동, 경희궁의아침3단지)

김성훈

서울 성북구 안암로 145, 정보보호대학원 (안암동5가, 고려대학교)

이광재

서울 성북구 안암로 145, 정보보호대학원 (안암동5가, 고려대학교)

(74) 대리인

특허법인충현

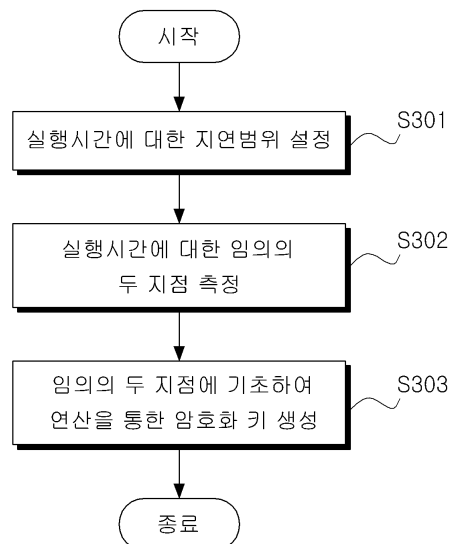
전체 청구항 수 : 총 11 항

(54) 발명의 명칭 역공학을 방지하기 위한 암호화 방법

(57) 요약

역공학을 방지하기 위한 암호화 방법에 관한 것으로, 프로세서의 실행시간에 대한 지연범위를 설정하고, 프로세서의 실행시간에 대한 임의의 두 지점을 측정하며, 측정한 임의의 두 지점에 기초하여 지연범위를 기반으로 하는 연산을 통해 역공학을 방지할 수 있는 암호화 키를 생성함으로써, 역공학에 의한 공격이 발생할 경우 역공학 대상 프로그램의 실행시간이 증가함으로써 연산에 의해 도출된 값이 설정된 지연범위를 초과하게 되어 비정상적인 암호화 키가 생성된다.

대표도 - 도3



이 발명을 지원한 국가연구개발사업

과제고유번호 KI002113

부처명 산업통상자원부

연구관리전문기관 한국산업기술평가관리원

연구사업명 산업융합원천기술개발사업

연구과제명 Car-헬스케어 보안 기술개발

기 여 율 1/1

주관기관 고려대학교 산학협력단

연구기간 2009.03.01 ~ 2014.02.28

특허청구의 범위

청구항 1

역공학(reverse engineering)을 방지하기 위한 암호화 방법에 있어서,
프로세서(Processor)의 실행시간에 대한 지연범위를 설정하는 단계;
상기 프로세서의 실행시간에 대한 임의의 두 지점을 측정하는 단계; 및
상기 측정된 임의의 두 지점에 기초하여 상기 지연범위를 기반으로 하는 연산을 통해 상기 역공학을 방지할 수 있는 암호화 키를 생성하는 단계를 포함하되,
상기 역공학에 의한 소정 프로그램이 상기 역공학 대상 프로그램에 추가될 경우 상기 역공학 대상 프로그램의 실행시간이 증가함으로써 상기 연산에 의해 도출된 값이 상기 설정된 지연범위를 초과하게 되는 것을 특징으로 하는 방법.

청구항 2

제 1 항에 있어서,
상기 지연범위는,
상기 프로세서의 실행환경에 따른 상기 프로세서의 실행시간 변화를 고려하여 설정되는 것을 특징으로 하는 방법.

청구항 3

제 1 항에 있어서,
상기 연산은,
상기 측정된 두 지점에 대한 오차범위만큼 시프트(Shift) 연산하며,
상기 시프트 연산된 두 지점의 값을 뺄셈한 값이 0이 되도록 함으로써 각기 다른 실행환경의 프로세서에 적용 가능한 것을 특징으로 하는 방법.

청구항 4

제 1 항에 있어서,
상기 키를 생성하는 단계는,
상기 측정된 두 지점에 기초하여 시프트 연산하고, 상기 시프트 연산을 통해 도출된 값을 다시 뺄셈하고,
상기 뺄셈을 통해 도출된 값과 매개 변수 값을 덧셈하고, 상기 덧셈을 통해 도출된 값을 해시 함수를 이용하여 상기 키를 생성하며,
상기 생성된 키를 이용하여 보호하고자 하는 영역을 암호화하여 저장하는 단계를 더 포함하는 방법.

청구항 5

제 4 항에 있어서,
상기 뺄셈을 통해 도출된 값이 프로세서의 실행시간에 대한 지연범위에 포함될 경우,
정상적인 암호화에 사용 가능한 키가 생성되고,
상기 암호화된 영역을 상기 생성된 키를 사용하여 복호화 하며,
상기 복호화 과정을 통해 정상적인 데이터를 획득할 수 있는 것을 특징으로 하는 방법.

청구항 6

제 4 항에 있어서,

상기 뺄셈을 통해 도출된 값이 프로세서의 실행시간에 대한 지연범위를 초과하는 경우,

정상적인 암호화에 사용 불가능한 키가 생성되고,

상기 암호화된 영역을 상기 생성된 키를 사용하여 복호화 하며,

상기 복호화 과정을 통해 비정상적인 데이터를 획득하거나 상기 복호화 과정에서 오류가 발생하는 것을 특징으로 하는 방법.

청구항 7

제 4 항에 있어서,

상기 매개 변수 값은,

상기 프로세서의 실행코드 중 소정의 코드영역을 지정할 수 있는 것을 특징으로 하는 방법.

청구항 8

제 7 항에 있어서,

상기 지정된 코드영역에 변조가 일어날 경우,

상기 매개 변수 값이 변조되어 정상적인 복호화에 사용 불가능한 키가 생성되는 것을 특징으로 하는 방법.

청구항 9

제 1 항에 있어서,

상기 키를 생성하는 단계는,

상기 측정된 두 지점에 기초하여 시프트 연산하고, 상기 시프트 연산을 통해 도출된 값을 다시 뺄셈하고,

상기 뺄셈을 통해 도출된 값에 기초하여 상기 키를 생성하며,

상기 생성된 키를 원본 명령어 세트를 보호하는 위장 명령어 세트에 첨부하여 상기 위장 명령어 세트를 재생성하는 단계를 더 포함하는 방법.

청구항 10

제 9 항에 있어서,

상기 뺄셈을 통해 도출된 값이 프로세서의 실행시간에 대한 지연범위에 포함되는 경우,

정상적인 암호화에 사용 가능한 키가 생성되고,

상기 생성된 키는 상기 위장 명령어 세트에 첨부하여 상기 위장 명령어 세트를 재생성하며,

상기 원본 명령어 실행 요청 시 상기 위장 명령어 세트는 상기 원본 명령어 세트로 복원되어 원본 명령어를 실행할 수 있는 것을 특징으로 하는 방법.

청구항 11

제 9 항에 있어서,

상기 뺄셈을 통해 도출된 값이 프로세서의 실행시간에 대한 지연범위를 초과하는 경우,

정상적인 암호화에 사용 불가능한 키가 생성되고,

상기 생성된 키는 상기 위장 명령어 세트에 첨부하여 상기 위장 명령어 세트를 재생성하며,

상기 원본 명령어 실행 요청 시 상기 위장 명령어 세트는 상기 원본 명령어 세트로 복원이 되지 않고 오류가 발생하는 것을 특징으로 하는 방법.

명세서

기술 분야

[0001] 본 발명은 역공학을 방지하기 위한 암호화 방법에 관한 것으로, 특히 암호화에 필요한 키를 생성함에 있어서, 프로세서의 실행시간을 기반으로 하는 연산에 기초하여 역공학을 방지할 수 있는 암호화 키를 생성하는 방법에 관한 것이다.

배경 기술

[0002] 소프트웨어(SW, Software)는 보이지 않는 여러 방면에 산재하여 인간과 상호작용을 하며 다양한 서비스를 제공한다. 최근 지식정보화 사회에서 국가 및 군사의 기밀 데이터, 기업 데이터, 병원 기록, 개인 사생활 정보 등의 대부분은 SW에 의해 처리된다. 사람들의 일상생활과 관련한 민감하고 가치 있는 데이터를 다루는 공공 기관의 많은 기능(함수)들과 대부분의 국가 안보체계도 복잡하고 상호 연동된 소프트웨어 집약 정보 시스템(software-intensive information system)에 의존한다. 현재 소프트웨어 집약 시스템은 전력 시스템, 상하수도처리 및 공급 시스템, 자동차와 항공 교통제어 및 수송 신호 시스템, 핵 발전소, 생화학 실험실, 은행, 교도소 등의 국가나 기업의 주요 시스템 및 기반시설을 움직인다.

[0003] 이에, 소프트웨어 관련 시장 규모가 증가함으로써, 소프트웨어를 공격할 수 있는 악성코드의 종류가 다양해지고 악성코드에 의한 피해도 커지고 있다. 악성코드는 바이러스, 웜, 트로이 목마, 스파이웨어 등으로 구분되며 감염대상 코드의 실행 구조를 변경하거나 내부구조를 바꿔서 대상 코드를 수행하기 전후에 특정 기능을 수행하는데 보통 자기복제 능력을 갖는다. 또한, 인터넷에의 의존도가 커짐에 따라 악성코드의 전파 속도도 증가하는 추세에 있다.

[0004] 한편, 기존의 소프트웨어 보호기술은 암호화(encryption) 기술에 기반을 둔 디지털 저작권 관리(DRM, Digital Rights Management), 디지털 워터마킹(digital watermarking), 디지털 핑거프린팅(digital fingerprinting), 코드 암호화 기법, 명령어 위장 기법 등의 기술들이 개발되어 왔다. 특히, 코드 암호화 기법은 코드를 암호화함으로써 정적 분석을 어렵게 하며, 명령어 위장 기법은 원본 명령어 셋을 위장 명령어 셋으로 변환함으로써 정적 분석을 어렵게 하였다. 하지만, 상기 코드 암호화 기법은 디버거(debugger)를 이용한 동적 분석에 취약하며, 상기 명령어 위장 기법은 리턴 값 조작이나 무조건 점프문으로 변화하여 우회 공격이 가능하였다.

[0005] 이상과 같은 관점에서, 기존 암호화 기술은 역공학에 취약하며, 역공학을 방지할 수 있는 기술적 수단이 필요하다는 사실을 알 수 있다.

선행기술문헌

비특허문헌

[0006] (비특허문헌 0001) 응용프로그램 역분석 방지를 위한 코드블록 암호화 방법, 정보보호학회논문지, 정동우, 2008. 4, 공개

발명의 내용

해결하려는 과제

[0007] 본 발명이 해결하고자 하는 기술적 과제는, 종래의 암호화 기술에 기반을 둔 코드 암호화 기법이 디버거 등의 역공학에 의한 동적 분석에 취약하다는 단점을 해결하고, 명령어 위장 기법 역시 조건 구문의 우회 공격 회피의 어려움을 극복하고자 한다.

과제의 해결 수단

[0008] 상기 기술적 과제를 해결하기 위하여, 본 발명의 일 실시예에 따른 역공학(reverse engineering)을 방지하기 위한 암호화 방법에 있어서, 프로세서(Processor)의 실행시간에 대한 지연범위를 설정하는 단계; 상기 프로세서의 실행시간에 대한 임의의 두 지점을 측정하는 단계; 및 상기 측정한 임의의 두 지점에 기초하여 상기 지연범위를 기반으로 하는 연산을 통해 상기 역공학을 방지할 수 있는 암호화 키를 생성하는 단계를 포함하되, 상기 역공학에 의한 소정 프로그램이 상기 역공학 대상 프로그램에 추가될 경우 상기 역공학 대상 프로그램의 실행시간이 증가함으로써 상기 연산에 의해 도출된 값이 상기 설정된 지연범위를 초과할 수 있다.

- [0009] 일 실시예에 따른 상기 지연범위는, 상기 프로세서의 실행환경에 따른 상기 프로세서의 실행시간 변화를 고려하여 설정할 수 있다.
- [0010] 일 실시예에 따른 상기 연산은, 상기 측정된 두 지점에 대한 오차범위만큼 시프트(Shift) 연산하며, 상기 시프트 연산된 두 지점의 값을 뺀 값이 0이 되도록 함으로써 각기 다른 실행환경의 프로세서에 적용 가능할 수 있다.
- [0011] 일 실시예에 따른 상기 키를 생성하는 단계는, 상기 측정된 두 지점에 기초하여 시프트 연산하고, 상기 시프트 연산을 통해 도출된 값을 다시 뺀 값하고, 상기 뺀 값을 통해 도출된 값과 매개 변수 값을 덧셈하고, 상기 덧셈을 통해 도출된 값을 해시 함수를 이용하여 상기 키를 생성하며, 상기 생성된 키를 이용하여 보호하고자 하는 영역을 암호화하여 저장하는 단계를 더 포함할 수 있다.
- [0012] 일 실시예에 따른 상기 뺀 값을 통해 도출된 값이 프로세서의 실행시간에 대한 지연범위에 포함될 경우, 정상적인 암호화에 사용 가능한 키가 생성되고, 상기 암호화된 영역을 상기 생성된 키를 사용하여 복호화 하며, 상기 복호화 과정을 통해 정상적인 데이터를 획득할 수 있다.
- [0013] 일 실시예에 따른 상기 뺀 값을 통해 도출된 값이 프로세서의 실행시간에 대한 지연범위를 초과하는 경우, 정상적인 암호화에 사용 불가능한 키가 생성되고, 상기 암호화된 영역을 상기 생성된 키를 사용하여 복호화 하며, 상기 복호화 과정을 통해 비정상적인 데이터를 획득하거나 상기 복호화 과정에서 오류가 발생할 수 있다.
- [0014] 일 실시예에 따른 상기 매개 변수 값은, 상기 프로세서의 실행코드 중 소정의 코드영역을 지정할 수 있다.
- [0015] 일 실시예에 따른 상기 지정된 코드영역에 변조가 일어날 경우, 상기 매개 변수 값이 변조되어 정상적인 복호화에 사용 불가능한 키가 생성될 수 있다.
- [0016] 일 실시예에 따른 상기 키를 생성하는 단계는, 상기 측정된 두 지점에 기초하여 시프트 연산하고, 상기 시프트 연산을 통해 도출된 값을 다시 뺀 값하고, 상기 뺀 값을 통해 도출된 값에 기초하여 상기 키를 생성하며, 상기 생성된 키를 원본 명령어 세트를 보호하는 위장 명령어 세트에 첨부하여 상기 위장 명령어 세트를 재생성 하는 단계를 더 포함할 수 있다.

발명의 효과

- [0017] 본 발명의 실시예들은, 프로세서의 실행시간에 대한 지연범위를 설정하고, 실행시간에 대한 임의의 두 지점을 측정하며, 측정된 임의의 두 지점에 기초하여 상기 지연범위를 기반으로 하는 연산을 통해 암호화 키를 생성함으로써, 역공학을 방지할 수 있기 때문에 소프트웨어 보호에 효과적으로 활용될 수 있다.

도면의 간단한 설명

- [0018] 도 1은 코드 암호화 기법의 실행 과정을 설명하기 위한 도면이다.
- 도 2는 명령어 위장 기법의 실행 과정을 설명하기 위한 도면이다.
- 도 3은 본 발명의 실시예들이 채택하고 있는 프로세서의 실행시간을 기반으로 하는 암호화 키 생성 방법을 설명하기 위한 흐름도이다.
- 도 4는 본 발명의 일 실시예에 따른 측정된 두 지점에 대한 오차범위만큼 시프트(Shift) 연산하는 과정을 설명하기 위한 도면이다.
- 도 5는 본 발명의 일 실시예에 따른 실행시간을 기반으로 하는 암호화 키를 코드 암호화 기법에 적용하여 실행하는 과정을 설명하기 위한 도면이다.
- 도 6은 본 발명의 일 실시예에 따른 실행시간을 기반으로 하는 암호화 키를 명령어 위장 기법에 적용하여 실행하는 과정을 설명하기 위한 도면이다.

발명을 실시하기 위한 구체적인 내용

- [0019] 본 발명의 실시예들을 설명하기에 앞서 본 발명의 실시예들이 구현, 활용되는 환경에서 발생하고 있는 문제점을 제시하고, 이에 기초하여 안출된 본 발명의 기본 아이디어를 제시하도록 한다.
- [0020] 일반적인 암호화 방식은 프로그램 전체를 암호화하여 저장하고, 복호화 루틴(routine)은 프로그램을 실행하고 있을 때 다른 프로그램 기능이 필요하게 되어 호출된 프로그램이 최초로 실행하는 명령이 저장되어 있는 어드레

스(address)인 엔트리 포인트(entry point)처럼 암호화된 바디(body)와 세트(set)에 보통 추가된다. 따라서, 암호화된 프로그램은 한 비트(bit)를 변조하면 한 비트 이상이 변하게 되므로 정적 분석을 통한 취약점을 찾기 어려웠다. 하지만, 복호화 시 코드 전체가 동시에 복호화 되고, 공격자는 복호화 되는 순간 기억장치 영역의 내용만 습득하면 언제든지 복호화가 가능하다는 취약점이 있었다.

[0021] 따라서, 프로그램 실행 시 프로그램 전체가 노출되는 것을 막기 위해, 프로그램 실행 시 필요한 부분만 복호화를 통해 다시 암호화하는 요구 시(On-demand) 암호화 방식이 고안되었다.

[0022] 도 1은 코드 암호화 기법의 실행 과정을 설명하기 위한 도면이다.

[0023] 요구 시 암호화 방식을 이용한 코드 암호화 기법은 b영역(12)을 암호화하기 위해서 b영역(12)이 아닌 a영역(11)을 해시함수에 기초하여 키(Key) 값(13)을 생성하며, 키 값(13)으로 b영역(12)을 암호화하는 방식이다.

[0024] 따라서, 암호화된 b영역(14)을 복호화 하기 위해서는 다른 영역인 a영역(11)의 해시함수에 기초하여 복호화 키(13)를 생성하고, 생성된 복호화 키(13)를 사용하여 암호화된 b영역(14)을 복호화 한다. 여기서, 복호화 키(13) 생성에 사용된 a영역(11)이 변조될 경우 복호화 키(13)의 해시 값이 달라져 비정상적인 키 값이 생성되며, 비정상적인 키 값으로 암호화된 b영역(14)을 복호화 하였을 경우 잘못된 데이터로 변조되고 오류가 발생한다. 반면, 정상적인 복호화 키(13)가 생성되어 암호화된 b영역(14)을 복호화 할 경우 암호화 이전 정상적인 b영역(12)을 획득하고, 획득한 b영역(12)을 실행하며, 암호화 키(13)를 사용하여 다시 b영역(12)을 암호화한다.

[0025] 상기와 같이, 코드 암호화 기법은 코드를 암호화함으로써, 정적 분석을 어렵게 한다. 하지만, 프로세서에 디버거(debugger)가 첨부되면 동적 분석을 통한 복호화 키를 획득할 수 있다는 취약점은 여전히 존재한다.

[0026] 도 2는 명령어 위장 기법의 실행 과정을 설명하기 위한 도면이다.

[0027] 코드가 기계 언어를 입력 받고 상기 입력 받은 기계언어를 어셈블리(assembler) 언어에 의한 프로그램을 작성하는 원시 프로그램인 역어셈블리(deassembler) 되더라도 상기 코드의 분석을 어렵게 하기 위해 원본 명령어 세트(21)를 위장 명령어 세트(22)로 변환하는 기법이다.

[0028] 보다 구체적으로, 프로그램의 명령어 중 보호하려는 원본 명령어 세트(21)를 지정하고, 원본 명령어 세트(21)를 위장 명령어 세트(22)로 변환한다. 이제, 원본 명령어 세트(21) 실행 요청 시 목표 블록(Target Block)(25)의 시간을 측정하여 실행 시간이 유효 범위 이내일 경우 복원 루틴(Restoring Routine)(23)을 사용하여 위장 명령어 세트(22)를 원본 명령어 세트(21)로 복원하고, 복원된 원본 명령어 세트(21)를 실행하며, 메모리에 원본 명령어 세트(21)가 남아 있는 것을 방지하기 위해 은닉 루틴(Hiding Routing)(24)을 사용하여 복원된 원본 명령어 세트(21)를 위장 명령어 세트(22)로 변환시킨다.

[0029] 반면, 원본 명령어 세트(21) 실행 요청 시 목표 블록(25)의 시간을 측정하여 실행 시간이 유효 범위를 초과할 경우 복원 루틴(23)은 작동되지 않고, 위장 명령어 세트(22)의 명령어가 실행되며, 잘못된 명령어로 인한 오류가 발생한다.

[0030] 상기와 같이, 조건문 응답 방식인 명령어 위장 기법은 원본 명령어 세트(21)를 위장 명령어 세트(22)로 변환함으로써, 정적 분석을 어렵게 한다. 하지만, 프로그램의 리턴(return) 값 조작, 무조건 점프문으로 변환 및 함수 삭제로 우회 공격이 가능하다는 취약점이 여전히 존재한다.

[0031] 따라서, 본 발명의 실시예들은 프로세서의 실행시간에 대한 지연범위를 설정하고, 실행시간에 대한 임의의 두 지점을 측정하며, 측정한 임의의 두 지점에 기초하여 상기 지연범위를 기반으로 하는 연산을 통해 암호화 키를 생성함으로써, 역공학을 방지할 수 있는 기술적 수단을 제안하고자 한다.

[0032] 이하에서는 첨부된 도면을 참조하여 본 발명의 실시예들을 보다 구체적으로 설명한다. 다만, 하기의 설명 및 첨부된 도면에서 본 발명의 요지를 흐릴 수 있는 공지 기능 또는 구성에 대한 상세한 설명은 생략한다. 또한, 도면 전체에 걸쳐 동일한 구성 요소들은 가능한 한 동일한 도면 부호로 나타내고 있음에 유의하여야 한다.

[0033] 도 3은 본 발명의 실시예들이 채택하고 있는 프로세서의 실행시간을 기반으로 하는 암호화 키 생성 방법을 설명하기 위한 흐름도로서, 적어도 하나의 프로세서를 포함하는 암호화 시스템에 의해 수행될 수 있다. 또한, 암호화 시스템은 구현상의 필요에 따라 이하의 일련의 연산을 수행하기 위한 메모리 내지 연산을 정의하는 소프트웨어 코드를 포함할 수 있다.

[0034] S301 단계에서, 암호화 시스템이 프로세서의 실행시간에 대한 지연범위를 설정한다. 여기서, 상기 지연범위는 상기 프로세서의 실행환경에 따른 상기 프로세서의 실행시간 변화를 고려하여 설정할 수 있다. 상기 프로세서의

실행시간은 클락(Clock)에 의해 구현되며, 상기 지연범위는 클락에 기초하여 구현된 상기 프로세서의 실행시간을 기반으로 설정할 수 있다.

[0035] S302 단계에서, 암호화 시스템은 프로세서의 실행시간에 대한 임의의 두 지점을 측정한다.

[0036] S303 단계에서, 암호화 시스템은 S302 단계에서 측정한 임의의 두 지점에 기초하여 상기 지연범위를 기반으로 하는 연산을 통해 역공학을 방지할 수 있는 암호화 키를 생성할 수 있다.

[0037] 여기서, 역공학에 의한 디버깅 도구와 같은 프로그램이 역공학 대상 프로그램에 추가될 경우 역공학 대상 프로그램의 실행시간이 증가함으로써 상기 연산에 의해 도출된 값이 S301 단계에서 설정된 지연범위를 초과하게 되어 비정상적인 키가 생성될 수 있다. 또한, 상기 연산은 상기 측정된 두 지점에 대한 오차범위만큼 시프트(Shift) 연산하며, 상기 시프트 연산된 두 지점의 값을 뺄셈한 값이 0이 되도록 함으로써 각기 다른 실행환경의 프로세서에 적용할 수 있다. 상기 시프트 연산은 이하에서 도 4를 참조하여 구체적으로 설명하도록 한다.

[0038] 도 4는 본 발명의 일 실시예에 따른 측정된 두 지점에 대한 오차범위만큼 시프트(Shift) 연산하는 과정을 설명하기 위한 도면이다.

[0039] 보다 구체적으로, 프로세서의 실행시간에 대한 임의의 두 지점을 측정한 것을 각각 $t_1(41)$ 과 $t_2(42)$ 라고 할 때, 측정된 $t_1(41)$ 과 $t_2(42)$ 에 기초하여 오차 범위를 설정하고, 상기 오차 범위만큼 $t_1(41)$ 과 $t_2(42)$ 를 각각 시프트 연산하며, 상기 시프트 연산을 통해 각각 $t'_1(43)$ 과 $t'_2(44)$ 를 획득할 수 있다. 이제, 시프트 연산을 통하여 획득한 $t'_2(44)$ 를 시프트 연산을 통해 획득한 $t'_1(43)$ 으로 뺄셈하면, 상기 뺄셈의 값은 상기 설정한 오차 범위 이내일 경우 프로세서의 실행시간과 무관하게 0 의 값이 도출된다.

[0040] 따라서, 프로세서의 실행 환경에 따른 상기 프로세서의 실행시간이 변화함으로써, 상기 프로세서의 실행시간을 키 생성에 적용할 경우, 키 생성에 필요한 연산을 통해 도출된 값이 상기 프로세서의 실행시간에 따라 변화되어 키 값 생성에 바로 적용 불가능한 문제점이 있었다. 하지만, 상기 시프트 연산을 통하여 상기 프로세서의 실행시간과 무관하게 0의 값이 도출됨으로써, 실행시간을 키 생성에 바로 사용할 수 없는 문제를 해결할 수 있다.

[0041] 도 5는 본 발명의 일 실시예에 따른 실행시간을 기반으로 하는 암호화 키를 코드 암호화 기법에 적용하여 실행하는 과정을 설명하기 위한 도면이다.

[0042] 보다 구체적으로, 처음 단계에서 프로세서의 실행시간에 대한 지연범위(501)를 설정한다. 여기서, 지연범위(501)는 상기 프로세서의 실행환경에 따른 상기 프로세서의 실행시간 변화를 고려하여 설정할 수 있다. 또한, 지연범위(501)는 정상적인 암호화에 사용 가능한 키를 생성하는 기준점으로 사용할 수 있다. 즉, 키를 생성하기 위한 연산을 통하여 도출된 값이 지연범위(501) 이내일 경우 정상적인 키가 생성되며, 상기 도출된 값이 지연범위(501)를 초과할 경우 비정상적인 키가 생성된다.

[0043] 다음 단계에서, 프로세서의 실행시간에 대한 임의의 두 지점인 $t_1(502)$ 과 $t_2(503)$ 를 각각 측정한다. 여기서, 상기 측정된 $t_1(502)$ 과 $t_2(503)$ 는 암호화에 사용되는 키를 생성하기 위한 연산에 사용할 수 있다.

[0044] 다음 단계에서, 상기 측정된 $t_1(502)$ 과 $t_2(503)$ 에 기초하여 오차 범위를 설정하고, 상기 오차 범위만큼 $t_1(502)$ 과 $t_2(503)$ 를 각각 시프트 연산(504)한다. 여기서, 상기 오차 범위만큼 시프트 연산하여 도출된 값을 뺄셈할 경우 상기 프로세서의 실행시간과 무관하게 0의 값이 도출됨으로써, 실행시간을 키 생성에 바로 사용할 수 있다.

[0045] 다음 단계에서, 시프트 연산(504)을 통해 각각 도출된 $t'_1(505)$ 과 $t'_2(506)$ 를 뺄셈하고, 상기 뺄셈을 통해 도출된 값과 매개 변수(507) 값을 덧셈하고, 상기 덧셈을 통해 도출된 값을 해쉬 함수를 이용하여 키(508)를 생성한다. 상기 시간을 기반으로 하는 키 생성은 다음과 같이 표현될 수 있다.

수학식 1

$$K=h\{t'_2-t'_1 \parallel IB\}$$

[0046]

[0047] 여기서, 상기 뺄셈을 통해 도출된 값이 상기 프로세서의 실행시간에 대한 지연범위에 포함될 경우, 정상적인 암호화 키를 생성할 수 있다.

호화에 사용 가능한 키(508)가 생성되고, 암호화된 영역을 상기 생성된 키(508)를 사용하여 복호화 하며, 상기 복호화 과정을 통해 정상적인 데이터를 획득할 수 있다.

[0048] 반면, 상기 뺄셈을 통해 도출된 값이 상기 프로세서의 실행시간에 대한 지연범위를 초과할 경우, 정상적인 암호화에 사용 불가능한 키가 생성되고, 암호화된 영역을 상기 암호화에 사용 불가능한 키를 사용하여 복호화 하며, 상기 복호화 과정을 통해 비정상적인 데이터를 획득하거나 상기 복호화 과정에서 오류가 발생할 수 있다. 여기서, 프로세서의 실행시간에 대한 지연범위를 초과할 경우는, 상기 프로세서에 대한 동적 분석을 하기 위하여 상기 프로세서에 디버거가 첨부되면, 상기 뺄셈을 통해 도출된 값이 상기 프로세서의 실행시간에 대한 지연범위를 초과하는 경우이다.

[0049] 또한, 상기 연산에 사용된 매개 변수(507)는 사용자가 프로세서의 실행코드 중 임의의 코드 영역을 지정할 수 있으며, 매개 변수(507)로 지정된 상기 코드 영역에 변조가 일어날 경우 정상적인 복호화에 사용 불가능한 키가 생성되어 데이터의 무결성을 보장할 수 있다.

[0050] 다음 단계에서, 상기 생성된 키(508)를 사용하여 보호하고자 하는 영역(509)을 암호화(509)하여 저장한다. 여기서, 암호화(509)된 보호하고자 하는 영역(509)을 복호화(510)하기 위해서는 상기 생성된 키(508)를 사용하며, 상기 생성된 키(508)를 사용하여 복호화(510)할 경우 암호화(509) 이전의 보호하고자 하는 영역(509)의 데이터를 획득할 수 있음으로써, 데이터의 기밀성을 보장할 수 있다.

[0051] 다음 단계에서, 복호화(510)된 보호하고자 하는 영역(509)을 다시 상기 생성된 키(508)를 사용하여 암호화(509)함으로써, 보호하고자 하는 영역(509)을 안전하게 보호할 수 있다.

[0052] 이와 같이, 시간을 기반으로 하는 키 생성 방법을 적용한 개선된 코드 암호화 기법은 동적 분석을 위한 디버거가 첨부될 경우, 시간 기반 키 생성을 위한 연산에 의해 도출된 값이 프로세서의 실행시간에 대한 지연범위를 초과함으로써, 비정상적인 키가 생성되며 복호화 과정에서 오류가 발생한다.

[0053] 도 6은 본 발명의 일 실시예에 따른 실행시간을 기반으로 하는 암호화 키를 명령어 위장 기법에 적용하여 실행하는 과정을 설명하기 위한 도면이다.

[0054] 보다 구체적으로, 처음 단계에서 프로그램의 명령어 중 보호하려는 원본 명령어 세트(61)를 지정하고, 원본 명령어 세트(61)를 위장 명령어 세트(62)로 변환할 수 있다.

[0055] 다음 단계에서, 상기 프로그램의 실행시간에 대한 임의의 한 지점인 t_1 을 측정하고, 상기 측정한 t_1 을 상기 프로그램 실행시간에 대한 지연범위(63)만큼 시프트 연산하여 t'_1 을 획득할 수 있다.

[0056] 다음 단계에서, 상기 프로그램의 목표 블록을 실행한 후 실행이 끝나는 지점인 t_2 를 측정하고, 상기 측정한 t_2 를 상기 프로그램의 실행시간에 대한 지연범위(63)만큼 시프트 연산하여 t'_2 를 획득한다. 이제, 획득한 t'_2 를 t'_1 에 뺄셈함으로써 암호화 키를 생성할 수 있다.

[0057] 다음 단계에서, 상기 생성된 키를 위장 명령어 세트(62)에 첨부하여 위장 명령어 세트를 재생성 한다. 여기서, 상기 뺄셈을 통해 도출된 값이 프로그램 실행시간에 대한 지연범위(63)에 포함할 경우, 정상적인 암호화에 사용 가능한 키가 생성되고, 상기 생성된 키는 위장 명령어 세트(62)에 첨부하여 위장 명령어 세트(62)를 재생성하며, 원본 명령어 세트(61) 실행 요청 시 복구루틴(64)을 사용하여 위장 명령어 세트(62)를 원본 명령어 세트(61)로 복원할 수 있다.

[0058] 반면, 상기 뺄셈을 통해 도출된 값이 프로그램 실행시간에 대한 지연범위(63)를 초과할 경우, 정상적인 암호화에 사용 불가능한 키가 생성되고, 상기 생성된 키는 위장 명령어 세트(62)에 첨부하여 위장 명령어 세트(62)를 재생성하며, 원본 명령어 세트(61) 실행 요청 시 복구루틴(64)을 사용하여 위장 명령어 세트(62)를 원본 명령어 세트(61)로 복원되지 않고 복원 과정에서 오류가 발생할 수 있다.

[0059] 다음 단계에서, 상기 복원된 원본 명령어 세트(61)는 은닉루틴(65)을 사용하여 위장 명령어 세트(22)로 다시 변환할 수 있다.

[0060] 이와 같이, 시간을 기반으로 하는 키 생성 방법을 적용한 개선된 명령어 위장 기법은, 조건문 응답 방식을 통한 실행 시간 측정값이 유효 범위에 있을 때만 복원 루틴을 실행하는 방식이 아닌, 실행 시간을 기반으로 하는 키를 생성하여 상기 생성된 키를 위장 명령어 세트(22)에 첨부하여 재생성함으로써, 정상적인 키가 생성되어 위장 명령어 세트(22)에 첨부될 경우만 복구루틴(64)을 사용하여 원본 명령어 세트(61)를 획득할 수 있다. 따라서,

조건 구문의 우회 공격으로 복구 루틴(64)을 실행한다 하여도, 복원에 필요한 키가 생성되지 않을 경우 복구루틴(64) 실행과정에서 오류가 발생한다. 또한, 디버거를 첨부하여 키를 생성할 경우 실행 시간의 증가로 인해 비정상적인 키가 생성됨으로써, 복구루틴(64) 실행과정에서 오류가 발생한다.

[0061] 본 발명의 일 실시예에 따른 시간 기반 키 생성 방법을 적용한 암호화 기법은 상기 코드 암호화 기법 및 상기 명령어 위장 기법에 국한되지 않으며, 상기 코드 암호화 기법 및 상기 명령어 위장 기법을 제외한 암호화 키를 생성하는 다른 암호화 기법으로 대체될 수 있다.

[0062] 한편, 본 발명은 컴퓨터로 읽을 수 있는 기록 매체에 컴퓨터가 읽을 수 있는 코드로 구현하는 것이 가능하다. 컴퓨터가 읽을 수 있는 기록 매체는 컴퓨터 시스템에 의하여 읽혀질 수 있는 데이터가 저장되는 모든 종류의 기록 장치를 포함한다.

[0063] 컴퓨터가 읽을 수 있는 기록 매체의 예로는 ROM, RAM, CD-ROM, 자기 테이프, 플로피디스크, 광 데이터 저장장치 등이 있으며, 또한 캐리어 웨이브(예를 들어 인터넷을 통한 전송)의 형태로 구현하는 것을 포함한다. 또한, 컴퓨터가 읽을 수 있는 기록 매체는 네트워크로 연결된 컴퓨터 시스템에 분산되어, 분산 방식으로 컴퓨터가 읽을 수 있는 코드가 저장되고 실행될 수 있다. 그리고 본 발명을 구현하기 위한 기능적인(functional) 프로그램, 코드 및 코드 세그먼트들은 본 발명이 속하는 기술 분야의 프로그래머들에 의하여 용이하게 추론될 수 있다.

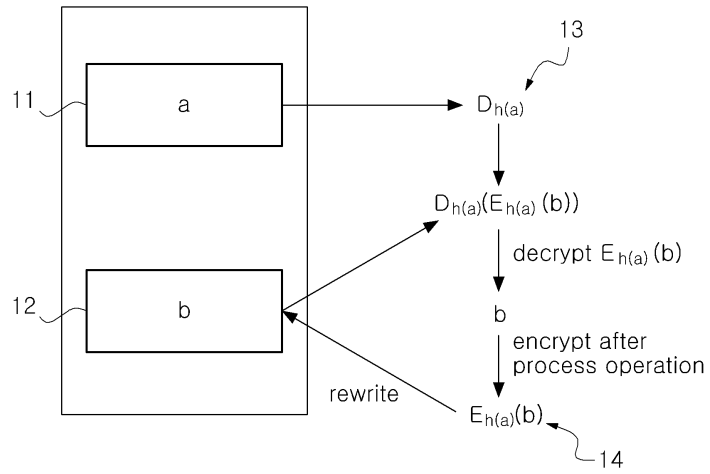
[0064] 이상에서 본 발명에 대하여 그 다양한 실시예들을 중심으로 살펴보았다. 본 발명에 속하는 기술 분야에서 통상의 지식을 가진 자는 본 발명이 본 발명의 본질적인 특성에서 벗어나지 않는 범위에서 변형된 형태로 구현될 수 있음을 이해할 수 있을 것이다. 그러므로 개시된 실시예들은 한정적인 관점이 아니라 설명적인 관점에서 고려되어야 한다. 본 발명의 범위는 전술한 설명이 아니라 특허청구범위에 나타나 있으며, 그와 동등한 범위 내에 있는 모든 차이점은 본 발명에 포함된 것으로 해석되어야 할 것이다.

부호의 설명

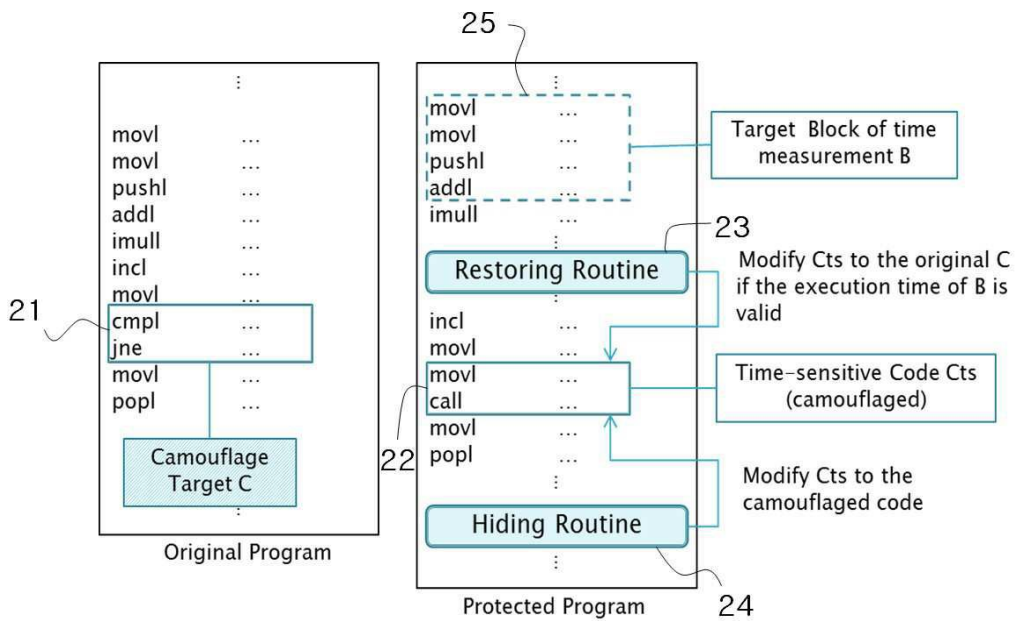
[0065] 501 : 프로세서 실행시간에 대한 지연범위
502 : 프로세서 실행시간에 대한 임의의 한 지점인 t_1
503 : 프로세서 실행시간에 대한 임의의 한 지점인 t_2
504 : 시프트(Shift) 연산
505 : t_1 을 오차범위만큼 시프트 연산하여 도출된 값인 t'_1
506 : t_2 을 오차범위만큼 시프트 연산하여 도출된 값인 t'_2
507 : 매개 변수
508 : 시간 기반 키
509 : 암호화
510 : 복호화
61 : 원본 명령어 세트
62 : 위장 명령어 세트
63 : 프로그램 실행시간에 대한 지연범위
64 : 복구 루틴
65 : 은닉 루틴

도면

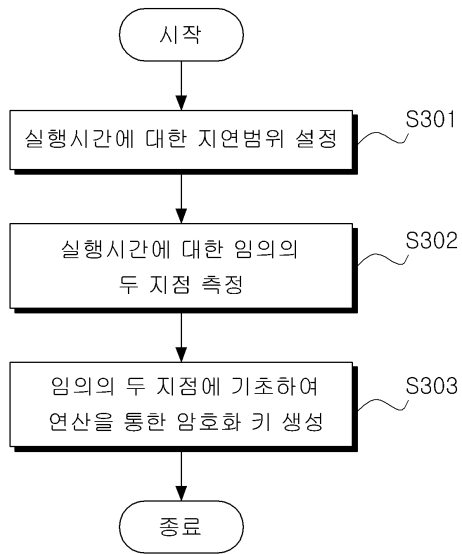
도면1



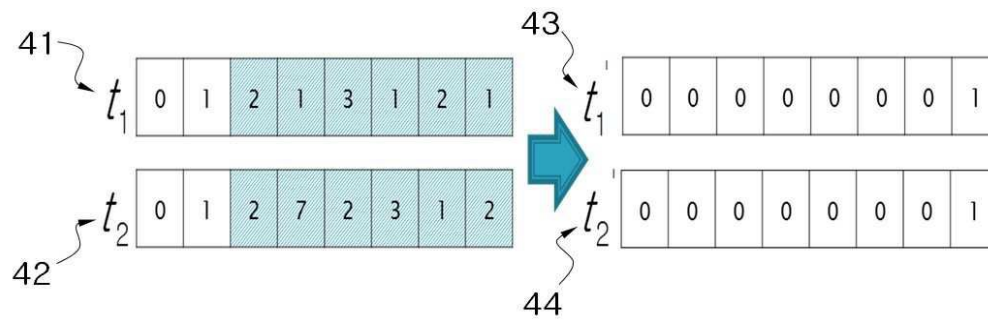
도면2



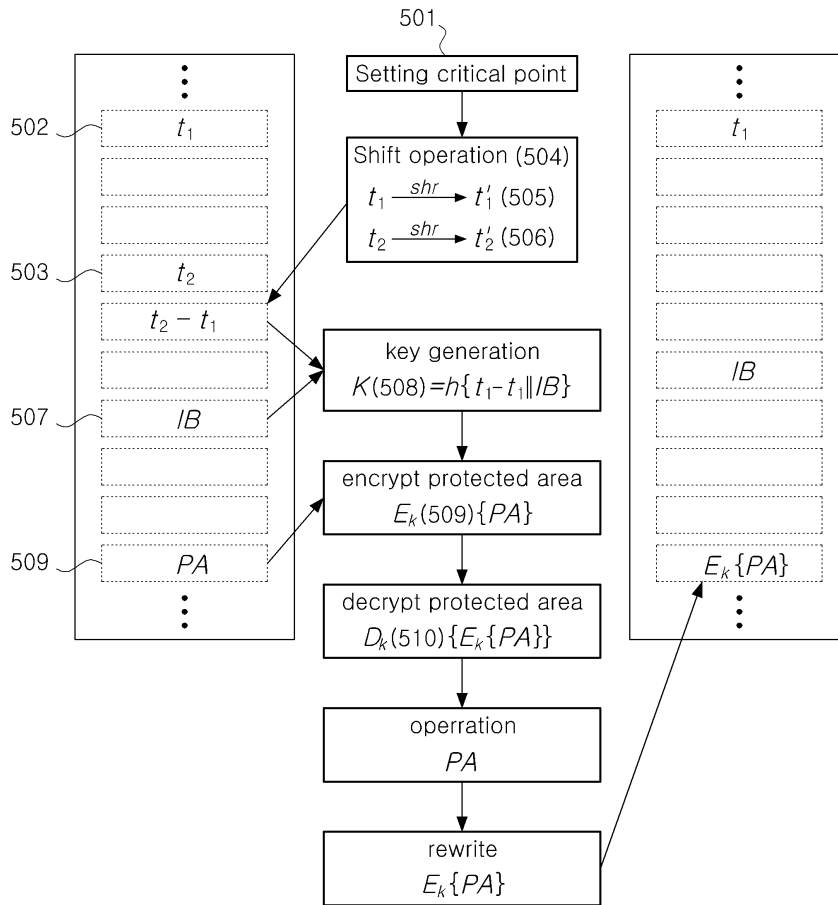
도면3



도면4



도면5



도면6

