(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2011-0106849
(43) 공개일자 2011년09월29일

(51) Int. Cl.
H04L 9/32 (2006.01) *H04L 9/14* (2006.01)
H04L 29/06 (2006.01) *G06F 21/22* (2006.01)
(21) 출원번호 10-2011-7013422
(22) 출원일자(국제출원일자) 2009년11월10일
심사청구일자 없음
(85) 번역문제출일자 2011년06월10일
(86) 국제출원번호 PCT/US2009/063861
(87) 국제공개번호 WO 2010/054369
국제공개일자 2010년05월14일
(30) 우선권주장
61/113,111 2008년11월10일 미국(US)

(71) 출원인
옥스포드 윌리엄 브이
미국 텍사스주 78755 오스틴 피오 박스 28161
(72) 발명자
옥스포드 윌리엄 브이
미국 텍사스주 78755 오스틴 피오 박스 28161
(74) 대리인
신정건, 김태홍

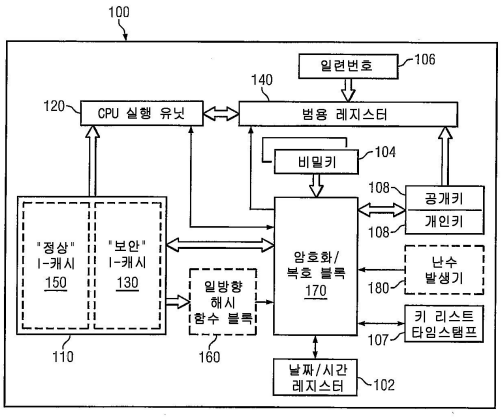
전체 청구항 수 : 총 24 항

(54) 회귀형 보안 프로토콜을 이용하여 컴퓨팅 장치에서 코드 실행을 제어하는 방법 및 시스템

(57) 요 약

범용 코드 블록의 실행에 대한 고도의 특수 제어를 제공하는 시스템 및 방법의 실시예가 개시된다. 이 실시예들은 주어진 코드 블록이 실행될 수 있는 정확한 환경을 특이성있게 결정할 수 있게 한다. 이러한 제어 메카니즘은 예를 들면 회귀적 실행을 통하여 구현되는 코드 세그멘트 집합의 순서정해진 실행에 기초하여 데이터 은닉 시스템 및 방법의 실시예와 결합된다. 이러한 시스템 및 방법의 실시예가 방해없는 일반성 및 공격에 대한 소정의 보호 레벨과 함께 활용될 때 많은 다른 보안 시스템을 능가하는 시스템 및 방법이 얻어질 수 있다.

대 표 도 - 도2



특허청구의 범위

청구항 1

엔드포인트 장치에서 코드의 실행을 제어하는 방법에 있어서,

장치에서 제1 비트스트림을 수신하는 단계와;

제1 비트스트림에 대응하고, 제1 비트스트림을 해싱하고 해시된 제1 비트스트림을 암호화함으로써 생성된 제1 키를 획득하는 단계와;

장치에 특정되고 장치의 하드웨어에 저장되며 장치가 보안 모드에서 실행할 때 액세스가능한 제1 비밀키를 액세스하도록 동작하는 장치의 하드웨어를 이용하여 제1 비트스트림을 인증하는 단계를 포함하고, 이 제1 비트스트림 인증 단계는,

제1 비트스트림을 해싱하는 단계와;

해시된 제1 비트스트림을 암호화 -해시된 제1 비트스트림의 암호화는 장치의 하드웨어에서 행하여지고 하드웨어는 장치에 특정된 제1 비밀키를 액세스하려고 시도하며 액세스의 결과를 암호화에 사용하는 것임- 함으로써 제2 키를 발생하는 단계와;

발생된 제2 키를 제1 키와 비교하는 단계와;

제2 키와 제1 키가 일치하는 경우 제1 비트스트림을 프로세서에서 보안 모드로 실행하는 단계를 포함하는 것인 코드 실행 제어 방법.

청구항 2

제1항에 있어서, 제1 비트스트림은 제1 암호화 엔진을 포함하고, 제1 비트스트림의 실행 단계는 제1 비트스트림과 관련된 암호화 디지털 콘텐트를 제1 암호화 엔진 및 장치에 특정된 제1 비밀키를 이용하여 복호하는 단계를 포함하며, 제1 비트스트림의 실행은 보안 모드에서 행하여지는 것인 코드 실행 제어 방법.

청구항 3

제2항에 있어서, 제2 키와 제1 키가 일치하지 않은 경우,

제1 비트스트림이 암호화되었는지 결정하는 단계를 포함하고, 제1 비트스트림이 암호화되었으면,

제2 비트스트림을 획득하는 단계와;

장치에 특정되고 하드웨어에 저장된 제1 비밀키를 액세스하도록 동작하는 장치의 하드웨어를 이용하여 제2 비트스트림을 인증하는 단계를 포함하며, 이 제2 비트스트림 인증 단계는,

제2 비트스트림에 대응하고, 제2 비트스트림을 해싱하고 해시된 제2 비트스트림을 암호화함으로써 생성된 제3 키를 획득하는 단계와;

제2 비트스트림을 해싱하는 단계와;

해시된 제2 비트스트림을 암호화 -해시된 제2 비트스트림의 암호화는 장치의 하드웨어에서 행하여지고 하드웨어는 장치에 특정된 제1 비밀키를 액세스하려고 시도하며 이 액세스의 결과를 암호화에 사용하는 것임- 함으로써 제4 키를 발생하는 단계와;

발생된 제4 키를 제3 키와 비교하는 단계와;

제4 키와 제3 키가 일치하는 경우 제2 비트스트림을 프로세서에서 보안 모드로 실행하는 단계를 포함하는 것인 코드 실행 제어 방법.

청구항 4

제3항에 있어서, 제2 비트스트림의 해싱은 제1 비트스트림으로부터 발생된 제2 키를 종자 값으로 활용하는 것인 코드 실행 제어 방법.

청구항 5

제3항에 있어서, 제2 비트스트림은 제2 암호화 엔진을 포함하고, 제2 비트스트림의 실행 단계는 제1 비트스트림 및 암호화 디지털 콘텐트를 장치에 특정된 제1 비밀키를 이용하여 제2 암호화 엔진으로 복호하는 단계를 포함하며, 제2 비트스트림의 실행은 보안 모드에서 행하여지는 것인 코드 실행 제어 방법.

청구항 6

제5항에 있어서, 제2 비트스트림의 인증 및 제2 비트스트림의 실행은 제1 비트스트림의 실행 전에 행하여지는 것인 코드 실행 제어 방법.

청구항 7

제6항에 있어서, 제2 비트스트림의 실행 후 및 제1 비트스트림의 실행 전에 제1 비트스트림을 인증하는 단계를 더 포함한 코드 실행 제어 방법.

청구항 8

제7항에 있어서, 제1 비트스트림, 제2 비트스트림, 암호화 디지털 콘텐트, 제1 키 및 제3 키는 메시지로 수신되고, 상기 메시지는,

디지털 콘텐트를 제1 비트스트림의 제1 암호화 엔진으로 암호화하는 단계와;

제1 비트스트림을 해싱하고 해시된 제1 비트스트림을 장치에 특정된 제1 비밀키로 암호화함으로써 제1 키를 발생하는 단계와;

제1 키, 제1 비트스트림 및 암호화 디지털 콘텐트를 관련시키는 단계와;

제1 키, 제1 비트스트림 및 암호화 디지털 콘텐트를 제2 비트스트림의 제2 암호화 엔진으로 암호화하는 단계와;

제2 비트스트림을 해싱하고 해시된 제2 비트스트림을 장치에 특정된 제1 비밀키로 암호화함으로써 제3 키를 발생하는 단계와;

제1 복호 알고리즘을 제1 암호화 비트스트림과 관련시키는 단계와;

제3 키, 제2 비트스트림 및 암호화 관련 제1 키, 제1 비트스트림 및 암호화 디지털 콘텐트를 관련시키는 단계에 의해 발생된 것인 코드 실행 제어 방법.

청구항 9

코드의 실행을 제어하는 시스템에 있어서,

장치를 포함하고, 이 장치는,

프로세서와;

제1 비밀키를 저장하는 제1 하드웨어와;

프로세서가 보안 모드에서 실행할 때 제1 비밀키를 액세스하고 제1 비밀키를 이용하여 암호화 알고리즘을 구현하도록 동작하는 제2 하드웨어와;

프로세서에 의해 실행가능한 명령어를 포함하는 컴퓨터 판독가능 매체를 포함하며, 상기 명령어는 프로세서에 의해 실행될 때,

장치에서 제1 비트스트림을 수신하는 단계와;

제1 비트스트림에 대응하고, 제1 비트스트림을 해싱하고 해시된 제1 비트스트림을 암호화함으로써 생성된 제1 키를 획득하는 단계와;

장치에서 제2 하드웨어를 이용하여 제1 비트스트림을 인증하는 단계를 수행하는 것이고, 상기 제1 비트스트림 인증 단계는,

제1 비트스트림을 해싱하는 단계와;

해시된 제1 비트스트림을 암호화 -해시된 제1 비트스트림의 암호화는 장치의 제2 하드웨어에서 행하여지고 제2 하드웨어는 장치에 특정된 제1 비밀키를 액세스하려고 시도하며 액세스의 결과를 암호화에 사용하는 것임- 함으로써 제2 키를 발생하는 단계와;

발생된 제2 키를 제1 키와 비교하는 단계와;

제2 키와 제1 키가 일치하는 경우 제1 비트스트림을 프로세서에서 보안 모드로 실행하는 단계를 포함하는 것인 코드 실행 제어 시스템.

청구항 10

제9항에 있어서, 제1 비트스트림은 제1 암호화 엔진을 포함하고, 제1 비트스트림의 실행 단계는 제1 비트스트림과 관련된 암호화 디지털 콘텐트를 제1 암호화 엔진 및 장치에 특정된 제1 비밀키를 이용하여 복호하는 단계를 포함하며, 제1 비트스트림의 실행은 보안 모드에서 행하여지는 것인 코드 실행 제어 시스템.

청구항 11

제10항에 있어서, 제2 키와 제1 키가 일치하지 않은 경우, 상기 명령어는,

제1 비트스트림이 암호화되었는지 결정하는 단계를 수행하고, 제1 비트스트림이 암호화되었으면,

제2 비트스트림을 획득하는 단계와;

장치에 특정되고 제1 하드웨어에 저장된 제1 비밀키를 액세스하도록 동작하는 장치의 하드웨어를 이용하여 제2 비트스트림을 인증하는 단계를 수행하며, 상기 제2 비트스트림 인증 단계는,

제2 비트스트림에 대응하고, 제2 비트스트림을 해싱하고 해시된 제2 비트스트림을 암호화함으로써 생성된 제3 키를 획득하는 단계와;

제2 비트스트림을 해싱하는 단계와;

해시된 제2 비트스트림을 암호화 -해시된 제2 비트스트림의 암호화는 장치의 제2 하드웨어에서 행하여지고 제2 하드웨어는 장치에 특정된 제1 비밀키를 액세스하려고 시도하며 이 액세스의 결과를 암호화에 사용하는 것임- 함으로써 제4 키를 발생하는 단계와;

발생된 제4 키를 제3 키와 비교하는 단계와;

제4 키와 제3 키가 일치하는 경우 제2 비트스트림을 프로세서에서 보안 모드로 실행하는 단계를 포함하는 것인 코드 실행 제어 시스템.

청구항 12

제11항에 있어서, 제2 비트스트림의 해싱은 제1 비트스트림으로부터 발생된 제2 키를 종자 값으로 활용하는 것인 코드 실행 제어 시스템.

청구항 13

제11항에 있어서, 제2 비트스트림은 제2 암호화 엔진을 포함하고, 제2 비트스트림의 실행 단계는 제1 비트스트림 및 암호화 디지털 콘텐트를 장치에 특정된 제1 비밀키를 이용하여 제2 암호화 엔진으로 복호하는 단계를 포함하며, 제2 비트스트림의 실행은 보안 모드에서 행하여지는 것인 코드 실행 제어 시스템.

청구항 14

제13항에 있어서, 제2 비트스트림의 인증 및 제2 비트스트림의 실행은 제1 비트스트림의 실행 전에 행하여지는 것인 코드 실행 제어 시스템.

청구항 15

제14항에 있어서, 상기 명령어는 제2 비트스트림의 실행 후 및 제1 비트스트림의 실행 전에 제1 비트스트림을 인증하는 단계를 더 수행하는 것인 코드 실행 제어 시스템.

청구항 16

제15항에 있어서, 제1 비트스트림, 제2 비트스트림, 암호화 디지털 콘텐트, 제1 키 및 제3 키는 메시지로 수신되고, 상기 메시지는,

디지털 콘텐트를 제1 비트스트림의 제1 암호화 엔진으로 암호화하는 단계와;

제1 비트스트림을 해싱하고 해시된 제1 비트스트림을 장치에 특정된 제1 비밀키로 암호화함으로써 제1 키를 발생하는 단계와;

제1 키, 제1 비트스트림 및 암호화 디지털 콘텐트를 관련시키는 단계와;

제1 키, 제1 비트스트림 및 암호화 디지털 콘텐트를 제2 비트스트림의 제2 암호화 엔진으로 암호화하는 단계와;

제2 비트스트림을 해싱하고 해시된 제2 비트스트림을 장치에 특정된 제1 비밀키로 암호화함으로써 제3 키를 발생하는 단계와;

제1 복호 알고리즘을 제1 암호화 비트스트림과 관련시키는 단계와;

제3 키, 제2 비트스트림 및 암호화 관련 제1 키, 제1 비트스트림 및 암호화 디지털 콘텐트를 관련시키는 단계에 의해 발생된 것인 코드 실행 제어 시스템.

청구항 17

프로세서에 의해 실행되어 엔드포인트 장치에서 코드의 실행을 제어하는 명령어를 포함한 컴퓨터 판독가능 매체에 있어서, 상기 명령어가,

장치에서 제1 비트스트림을 수신하는 단계와;

제1 비트스트림에 대응하고, 제1 비트스트림을 해싱하고 해시된 제1 비트스트림을 암호화함으로써 생성된 제1 키를 획득하는 단계와;

장치에 특정되고 장치의 하드웨어에 저장되며 장치가 보안 모드에서 실행할 때 액세스가능한 제1 비밀키를 액세스하도록 동작하는 장치의 하드웨어를 이용하여 제1 비트스트림을 인증하는 단계를 수행하는 것이고, 상기 제1 비트스트림 인증 단계는,

제1 비트스트림을 해싱하는 단계와;

해시된 제1 비트스트림을 암호화 -해시된 제1 비트스트림의 암호화는 장치의 하드웨어에서 행하여지고 하드웨어는 장치에 특정된 제1 비밀키를 액세스하려고 시도하며 액세스의 결과를 암호화에 사용하는 것임- 함으로써 제2 키를 발생하는 단계와;

발생된 제2 키를 제1 키와 비교하는 단계와;

제2 키와 제1 키가 일치하는 경우 제1 비트스트림을 프로세서에서 보안 모드로 실행하는 단계를 포함하는 것인 컴퓨터 판독가능 매체.

청구항 18

제17항에 있어서, 제1 비트스트림은 제1 암호화 엔진을 포함하고, 제1 비트스트림의 실행 단계는 제1 비트스트림과 관련된 암호화 디지털 콘텐트를 제1 암호화 엔진 및 장치에 특정된 제1 비밀키를 이용하여 복호하는 단계를 포함하며, 제1 비트스트림의 실행은 보안 모드에서 행하여지는 것인 컴퓨터 판독가능 매체.

청구항 19

제18항에 있어서, 제2 키와 제1 키가 일치하지 않은 경우, 상기 명령어가,

제1 비트스트림이 암호화되었는지 결정하는 단계를 더 수행하고, 제1 비트스트림이 암호화되었으면,

제2 비트스트림을 획득하는 단계와;

장치에 특정되고 하드웨어에 저장된 제1 비밀키를 액세스하도록 동작하는 장치의 하드웨어를 이용하여 제2 비트스트림을 인증하는 단계를 더 수행하며, 상기 제2 비트스트림 인증 단계는,

제2 비트스트림에 대응하고, 제2 비트스트림을 해싱하고 해시된 제2 비트스트림을 암호화함으로써 생성된 제3

키를 획득하는 단계와;

제2 비트스트림을 해싱하는 단계와;

해시된 제2 비트스트림을 암호화 -해시된 제2 비트스트림의 암호화는 장치의 하드웨어에서 행하여지고 하드웨어는 장치에 특정된 제1 비밀키를 액세스하려고 시도하며 이 액세스의 결과를 암호화에 사용하는 것임- 함으로써 제4 키를 발생하는 단계와;

발생된 제4 키를 제3 키와 비교하는 단계와;

제4 키와 제3 키가 일치하는 경우 제2 비트스트림을 프로세서에서 보안 모드로 실행하는 단계를 포함하는 것인 컴퓨터 판독가능 매체.

청구항 20

제19항에 있어서, 제2 비트스트림의 해싱은 제1 비트스트림으로부터 발생된 제2 키를 종자 값으로 활용하는 것인 컴퓨터 판독가능 매체.

청구항 21

제19항에 있어서, 제2 비트스트림은 제2 암호화 엔진을 포함하고, 제2 비트스트림의 실행 단계는 제1 비트스트림 및 암호화 디지털 콘텐트를 장치에 특정된 제1 비밀키를 이용하여 제2 암호화 엔진으로 복호하는 단계를 포함하며, 제2 비트스트림의 실행은 보안 모드에서 행하여지는 것인 컴퓨터 판독가능 매체.

청구항 22

제21항에 있어서, 제2 비트스트림의 인증 및 제2 비트스트림의 실행은 제1 비트스트림의 실행 전에 행하여지는 것인 컴퓨터 판독가능 매체.

청구항 23

제22항에 있어서, 상기 명령어가 제2 비트스트림의 실행 후 및 제1 비트스트림의 실행 전에 제1 비트스트림을 인증하는 단계를 더 수행하는 컴퓨터 판독가능 매체.

청구항 24

제23항에 있어서, 제1 비트스트림, 제2 비트스트림, 암호화 디지털 콘텐트, 제1 키 및 제3 키는 메시지로 수신되고, 상기 메시지는,

디지털 콘텐트를 제1 비트스트림의 제1 암호화 엔진으로 암호화하는 단계와;

제1 비트스트림을 해싱하고 해시된 제1 비트스트림을 장치에 특정된 제1 비밀키로 암호화함으로써 제1 키를 발생하는 단계와;

제1 키, 제1 비트스트림 및 암호화 디지털 콘텐트를 관련시키는 단계와;

제1 키, 제1 비트스트림 및 암호화 디지털 콘텐트를 제2 비트스트림의 제2 암호화 엔진으로 암호화하는 단계와;

제2 비트스트림을 해싱하고 해시된 제2 비트스트림을 장치에 특정된 제1 비밀키로 암호화함으로써 제3 키를 발생하는 단계와;

제1 복호 알고리즘을 제1 암호화 비트스트림과 관련시키는 단계와;

제3 키, 제2 비트스트림 및 암호화 관련 제1 키, 제1 비트스트림 및 암호화 디지털 콘텐트를 관련시키는 단계에 의해 발생된 것인 컴퓨터 판독가능 매체.

명 세 서

기 술 분 야

[0001] 본 발명은 범용 컴퓨터에서의 임의 코드(arbitrary code) 실행을 제어하는 방법에 관한 것이다. 특히, 본 발명은 코드의 실행을 제어하기 위해 코드의 실행을 컴퓨팅 장치에 연결하는 효과적인 방법에 관한 것이다. 더 구체적으로, 본 발명은 회귀형 보안 프로토콜(recursive security protocol)의 구현과 함께 코드의 실행을 제어하는

것에 관한 것이다.

배 경 기 술

[0002] 컴퓨터 바이러스 및 기타의 유해 소프트웨어는 정보 기술 산업에서 많은 문제를 일으킨다. 범용 컴퓨터는, 정의상으로, 임의 코드를 동작시킬 수 있기 때문에, 주어진 범용 컴퓨터 플랫폼에서 부분적으로 또는 전체적으로 소프트웨어의 동작을 허용하는 제어를 정확히 유지하는 것이 매우 어려울 수 있고, 따라서 멀웨어(malware) 또는 다른 유형의 바람직하지 않은 소프트웨어의 실행을 금지하기가 어려울 수 있다. 이러한 수준의 제어를 현재 시도하고 있는 많은 방법이 있지만, 프로세서를 공격으로부터 격리시키는 대부분의 노력은 2가지의 기본적인 문제점, 즉 프로세서 플랫폼에서의 일반성 상실 또는 성능 상실에 봉착한다. 이러한 상실은 인증된 사용 모드와 인증되지 않은 사용 모드를 어떻게 신속하고 명확하게 구별할 것인지에 대한 기본적인 문제로부터 발생한다.

발명의 내용

해결하려는 과제

[0003] 부차적이지만 관련이 있는 문제점은 저작권 제어(copyright control)의 문제이다. 오늘날 생성된 대부분의 기록된 음향적 및 시각적 예술 작품은 디지털 형식으로 시작하거나 종료하고 있다. 디지털 데이터의 특성들 중 하나는 쉽고 실질적으로 정확하게 복제가 가능하다는 것이다. 이러한 속성은 매우 다양한 저가의 분배 메카니즘을 촉진하고, 그 대부분은 용이하게 제어되지 않는다. 디지털 콘텐트의 분배를 제한하는 것에 대한 고유의 무능력은 최근 수 십년간 저작권법 분야에서 중요한 암시(far-reaching implication)를 갖고 있었다. 그러한 복제 데이터의 복사 및 분배를 제어하기 위한 특수한 시스템 및 방법이 개발되었지만, 이러한 시스템 및 방법이 갖는 하나의 문제점은 그 시스템 및 방법과 함께 특수 유형의 소프트웨어, 예를 들면 시스템 및 방법을 수정하거나 그러한 시스템 및 방법이 사용한 데이터를 획득하는 코드의 실행을 통하여 교묘히 회피될 수 있다는 점이다.

[0004] 따라서, 범용 컴퓨팅 장치에서의 코드 실행을 어느 정도 제어할 수 있는 시스템 및 방법을 찾을 필요가 있고, 보안 프로토콜과 함께 이러한 시스템 및 방법을 이용함으로써 그러한 시스템의 유효성이 개선될 수 있다.

과제의 해결 수단

[0005] 범용 코드 블록의 실행에 대한 고도의 특수 제어를 제공하는 시스템 및 방법의 실시예가 개시된다. 이 실시예들은 주어진 코드 블록이 실행될 수 있는 정확한 환경을 특이성있게 결정할 수 있게 한다. 이러한 제어 메카니즘은 예를 들면 회귀적 실행을 통하여 구현되는 코드 세그멘트 집합의 순서정해진 실행에 기초하여 데이터 은닉 시스템 및 방법의 실시예와 결합된다. 이러한 시스템 및 방법의 실시예가 방해없는 일반성(unencumbered generality) 및 공격에 대한 소정의 보호 레벨과 함께 활용될 때 많은 다른 보안 시스템을 능가하는 시스템 및 방법이 얻어질 수 있다.

[0006] 특히, 일 실시예에 있어서, 코드 실행을 조건부 제어하는 시스템 및 방법은 특수한 연산에서 사용하는 데이터를 불명료하게 하는 시스템 및 방법과 함께 개시되고, 그럼에도 불구하고 그러한 데이터의 사용이 여전히 허용된다. 상기 제어 또는 머리를 어지럽게 하는 시스템 및 방법은, 비제한적인 예를 들자면, 디지털 보안, 저작권 제어, 조건부 액세스, 바람직하지 않은 컴퓨터 바이러스에 대한 보호 등을 포함한 보안 영역 등의 다수의 잠재적 애플리케이션 영역에서 사용될 수 있다. 특히, 본 발명의 실시예들은 이러한 보안 프로토콜을 증가시키는 회귀형 보안 프로토콜과 함께 사용될 수 있다.

[0007] 추가적으로, 상기 유형의 방법을 컴퓨터 시스템, 하드웨어 및 소프트웨어로 구체화하는 시스템의 실시예들이 제시된다. 정확히 동일한 하드웨어 구현이 소프트웨어의 필요조건에 따라서 전체 솔루션의 범위의 어느 하나 또는 조합을 잠재적으로 구현하기 위해 사용될 수 있다는 것에 주목하여야 한다.

[0008] 본 발명의 상기 및 기타의 태양들은 하기의 설명 및 첨부 도면과 함께 고려할 때 더 잘 인식되고 이해될 것이다. 그러나, 이하의 설명은, 비록 발명의 각종 실시예 및 그에 대한 많은 특수한 세부를 표시하지만, 단지 예시를 위하여 주어진 것이고 발명을 제한하는 의도는 없다. 본 발명의 정신으로부터 벗어나지 않고 발명의 범위 내에서 각종의 치환, 수정, 추가 및/또는 재구성이 이루어질 수 있고, 본 발명은 이러한 치환, 수정, 추가 및/또는 재구성을 모두 포함한다.

발명의 효과

[0009] 범용 코드 블록의 실행에 대한 고도의 특수 제어를 제공하는 시스템 및 방법이 제공될 수 있다.

도면의 간단한 설명

[0010] 이 명세서에 첨부되어 이 명세서의 일부를 구성하는 도면은 본 발명의 특정 태양을 묘사하기 위해 포함된다. 본 발명 및 컴포넌트들의 명확한 개념 및 본 발명으로 제공되는 시스템의 동작은 도면에 도시된 예시적이고 따라서 비제한적인 실시예를 참조함으로써 더 쉽게 인식될 것이고, 첨부 도면에 있어서 동일한 참조 번호는 동일한 컴포넌트를 표시한다. 본 발명은 첨부 도면을 여기에서 제공하는 설명과 함께 참조함으로써 더 잘 이해할 수 있다. 도면에 도시된 특징들은 반드시 정확한 축척으로 도시된 것이 아님에 주목하여야 한다.

도 1은 콘텐트 분배를 위한 아키텍처의 일 실시예를 보인 도이다.

도 2는 목표 유닛 범용 컴퓨팅 장치의 일 실시예를 보인 도이다.

도 3은 합성 키의 생성을 위한 일 실시예를 보인 도이다.

도 4a 및 도 4b는 디지털 서명 또는 그 등가물의 생성에 대한 실시예를 보인 도이다.

도 5a 및 도 5b는 보안 코드 블록과 함께 합성 키 구조물의 사용에 대한 실시예를 보인 도이다.

도 6은 합성 메시지 다이제스트의 사용에 대한 일 실시예를 보인 도이다.

도 7a, 7b 및 7c는 보안 코드 블록 메시지의 실시예를 보인 도이다.

도 8은 복호 동작의 실시예를 보인 도이다.

도 9는 합성 키의 사용에 대한 일 실시예를 보인 도이다.

도 10은 후보 코드 블록의 인증에 있어서 합성 키의 사용에 대한 일 실시예를 보인 도이다.

도 11은 복호 동작의 일 실시예를 보인 도이다.

도 12는 코드 블록의 유효화의 일 실시예를 보인 도이다.

도 13은 회귀형 보안 프로토콜을 이용하여 수행되는 복호 동작의 일 실시예를 보인 도이다.

도 14는 코드 유효화 동작의 일 실시예를 보인 도이다.

도 15는 디지털 서명 기능 블록의 일 실시예를 보인 도이다.

발명을 실시하기 위한 구체적인 내용

[0011] 본 발명 및 그 각종 특징과 유리한 세부는 첨부 도면에 도시되고 이하의 설명에서 상세화되는 비제한적인 실시예를 참조하여 더욱 완전하게 설명된다. 잘 알려져 있는 시작 물질, 처리 기술, 컴포넌트 및 설비에 관한 설명은 본 발명의 세부를 불필요하게 불명료하게 하는 것을 방지하기 위해 생략한다. 그러나, 본 발명의 양호한 실시예를 표시하는 것이지만, 그 실시예에 대한 상세한 설명 및 특수한 예는 제한하는 의도가 없이 단지 예시하는 용도로만 주어진다는 것을 이해하여야 한다. 근원적인 발명 개념의 정신 및/또는 범위 내에서의 각종 치환, 수정, 추가 및/또는 재구성이 이 기술에 숙련된 사람에게는 이 명세서의 설명으로부터 명백할 것이다. 여기에서 설명하는 실시예들은 컴퓨터 판독가능 매체(예를 들면, HD)에 존재할 수 있는 적당한 컴퓨터 실행가능 명령어로, 하드웨어 회로 등으로, 또는 이들의 임의의 조합으로 구현될 수 있다.

[0012] 특정 실시예를 설명하기 전에, 특정 실시예를 구현하기 위한 하드웨어 아키텍처의 실시예를 여기에서 설명한다. 일 실시예는 네트워크에 통신적으로 결합된 하나 이상의 컴퓨터를 포함할 수 있다. 이 기술에 숙련된 사람이라면 알고 있는 바와 같이, 컴퓨터는 중앙 처리 장치("CPU"), 적어도 하나의 읽기 전용 메모리("ROM"), 적어도 하나의 랜덤 액세스 메모리("RAM"), 적어도 하나의 하드 드라이브("HD") 및 하나 이상의 입력/출력("I/O") 장치를 포함할 수 있다. I/O 장치는 키보드, 모니터, 프린터, 전자식 포인팅 장치(마우스, 트랙볼, 스타일리스트 등) 등을 포함할 수 있다. 각종 실시예에 있어서, 컴퓨터는 네트워크를 통해 적어도 하나의 데이터베이스에 액세스한다.

[0013] ROM, RAM 및 HD는 CPU에 의해 (예를 들면, 컴파일, 번역 등에 의해 직접 실행되거나 실행가능으로 되는 다른 것에서) 실행가능한 컴퓨터 명령어를 저장하기 위한 컴퓨터 메모리이다. 이 명세서에서, 용어 "컴퓨터 판독가능 매체"는 ROM, RAM 및 HD로 제한되는 것이 아니고, 프로세서에 의해 판독가능한 임의 유형의 데이터 기억 매체를

포함할 수 있다. 일부 실시예에서, 컴퓨터 판독가능 매체는 데이터 카트리지, 데이터 백업용 자기 테이프, 플로피 디스켓, 플래시 메모리 드라이브, 광학 데이터 기억장치 드라이브, CD-ROM, ROM, RAM, HD 등을 인용할 수 있다.

[0014] 여기에서 설명하는 기능 또는 처리의 적어도 일부는 적당한 컴퓨터 실행가능 명령어로 구현될 수 있다. 컴퓨터 실행가능 명령어는 하나 이상의 컴퓨터 판독가능 매체(비휘발성 메모리, 휘발성 메모리, DASD 어레이, 자기 테이프, 플로피 디스켓, 하드 드라이브, 광학 기억 장치 등, 또는 임의의 다른 적당한 컴퓨터 판독가능 매체 또는 기억 장치 등)에 소프트웨어 코드 성분 또는 모듈로서 저장될 수 있다. 일 실시예에 있어서, 컴퓨터 실행가능 명령어는 컴파일드 C++의 행(lines), 자바, HTML, 또는 임의의 다른 프로그래밍 또는 스크립팅 코드를 포함할 수 있다.

[0015] 게다가, 개시된 실시예의 기능들은 하나의 컴퓨터에서 구현되거나, 네트워크 내의 또는 네트워크에 걸친 2개 이상의 컴퓨터 사이에서 공유/분산될 수 있다. 실시예를 구현하는 컴퓨터들 간의 통신은 임의의 전자 신호, 광학 신호, 고주파수 신호를 이용하여, 또는 공지의 네트워크 프로토콜과 호환하는 다른 적당한 방법 및 통신 도구를 이용하여 달성될 수 있다.

[0016] 여기에서 사용되는 용어 "포함한다", "포함하는", "구비한다", "구비하는", "갖는다", "갖는" 또는 이들의 임의의 다른 변형 용어는 비배타적 내포를 포괄하는 것으로 의도된다. 예를 들면, 요소들의 리스트를 포함하는 프로세스, 처리, 물품 또는 장치는 단지 그 요소로만 제한될 필요가 없고, 명시적으로 리스트되지 않은 다른 요소 또는 그러한 프로세스, 처리, 물품 또는 장치에 고유한 다른 요소들을 포함할 수 있다. 또한, 명시적으로 다르게 설명되지 않는 한, "또는"은 총괄적 또는(inclusive or)을 인용하는 것이고 배타적 또는(exclusive or)을 의미하는 것이 아니다. 예를 들면, 조건 A 또는 B는 하기의 것 중 임의의 하나에 의해 만족된다; A는 참(또는 존재)이고 B는 거짓(또는 부재)이다; A는 거짓(또는 부재)이고 B는 참(또는 존재)이다; 및 A와 B가 둘 다 참(또는 존재)이다.

[0017] 추가적으로, 여기에서 주어진 임의의 예 또는 설명은 어떻게든 이들이 사용하는 임의의 용어 또는 용어들로 구속되는 것으로서, 제한되는 것으로서, 또는 용어들의 정의를 표현하는 것으로서 간주되지 않는다. 그 대신에, 이들 예 및 설명은 하나의 특수한 실시예와 관련하여 설명하고 단지 예시하는 것으로서 간주되어야 한다. 이 기술에 숙련된 사람이라면, 이들 예 및 설명에서 사용하는 임의의 용어 또는 용어들이 이 명세서의 어딘가에서 제시하거나 제시하지 않을 수 있는 다른 실시예에도 적용된다는 것을 알 것이며, 그러한 모든 실시예는 그 용어 또는 용어들의 범위에 내포된 것으로 의도된다. 그러한 비제한적인 예 및 설명을 지시하는 용어는, 비제한적인 예를 들자면, "예를 들면", "예로서", "예컨대", "일 실시예에 있어서" 등이 있다.

[0018] 전술한 바와 같이, 프로세서가 미리 규정된 방식으로 임의의 코드 세그멘트를 실행하게 하는 것이 바람직하다. 이 제어 문제는 적법한 소프트웨어까지도 조작하여 의도하지 않은 또는 오히려 유해한 결과를 생성할 수 있는 많은 변형 방법에 의해 조성될 수 있다. 이러한 공격 방법은 입력 데이터 코너 케이스 또는 다른 알고리즘적 결함을 이용하기 위해 루틴에 대한 입력으로서 가짜 아규멘트(bogus argument)를 공급함으로써 빈약하게 기록되었지만 유효한 코드를 이용하는 것을 포함할 수 있다. 다른 가능한 공격 방법은 각종 프로세서 레지스터(스택 포인터 등), 또는 부적당하거나 의도적으로 잘못된 결과를 산출시키는 방법으로 다른 합법적 코드에 의해 참조되는 외부 메모리 위치에 저장된 데이터를 독립적으로 수정하는 것을 포함한다.

[0019] 이러한 종류의 제어에 영향을 줄 수 있는 다수의 메카니즘이 있다. 이러한 시스템들은 프로세서를 이러한 종류의 비의도적 사용으로부터 보호하는 단순한 방식 및 복잡한 방식 둘 다를 포함한다. 한가지 상당히 안전하지만 복잡한 메카니즘은 코드 세그멘트를 그 실행 전에 선암호화(pre-encryption)하는 것이다. 일단 코드 세그멘트가 메모리로부터 프로세서로 로드되면, 코드 세그멘트는 신중하게 제어된 환경하에서 복호되고 안전한 방식으로 실행되어야 한다(다시 말해서, 코드 세그멘트는 복호 동작과 후속 실행 동작 사이에서 수정되거나 고쳐지지 않아야 한다). 이 메카니즘은 당해 코드 세그멘트가 실행될 수 있기 전에 복호될 때까지 프로세서가 대기해야 하기 때문에 발생할 수 있는 성능 비효율성이 있다. 실행 대상의 특수 코드 세그멘트의 선택과 복호후 실제 실행 사이의 이러한 대기시간(latency)은, 비제한적인 예를 들자면, 프로세서 파이프라인 스톨(stall) 및 데이터 경로 비효율성을 포함한 많은 문제점을 야기할 뿐만 아니라 (복호 동작과 실행 동작 사이에 어떻게든지 코드를 강탈(hijacking)함으로써) 잠재적 공격을 위한 다른 수단을 제공할 수 있다.

[0020] 상기 암호화 코드 메카니즘은 명시적으로 보호된 암호화 코드 세그멘트를 적절히 복호하도록 관리하는(또는 상기 암호화 코드 세그멘트의 복호된 복사본을 획득한) 해커의 우발성(eventuality)으로부터 프로세서를 또한 보호할 수 없다. 그 경우, 해커는 보호되지 않은 코드를 목표 프로세서에서 또는 어떤 다른 인증되지 않은 프로세

서에서 비제어 방식으로 동작시킬 수 있다. 따라서, 코드가 명문(clear)으로(즉 평문 형태로) 또는 암호화 형태로 분배되는지 여부에 관계없이, 코드 세그멘트가 특수한 프로세서 또는 프로세서들에서 동작할 수 있도록 정확히 제어하는 방법을 찾는 것이 바람직하다. 반면에, 만일 특수한 프로세서에서 동작할 수 있는 코드가 어떤 미리 선택된 부분 집합으로 제한되면, 프로세서 자체의 범용성이 위배될 수 있다. 이것은 아마도 프로세서가 덜 범용적으로 되고 따라서 그 잠재적 응용 공간에서 훨씬 덜 융통성있게 하는 방법으로 아키텍쳐를 속박하는 효과를 가질 수 있다. 최대로 유통성있는 범용 프로세서 아키텍쳐에 대한 강한 희망이 항상 있지만, 그러한 프로세서는 엄밀히 말해서 대부분 멀웨어 공격에 취약하다.

[0021] 따라서, 임의의 특수 프로세서 아키텍쳐에 의존하지 않을 정도로 충분한 범용인 코드 실행을 제어하는 방법이 필요하다. 그러한 방법이 목적 코드(object code) 밀도 또는 목표 프로세서에 악영향을 주지 않는다면 또한 유용할 것이다. 동시에, 그러한 시스템 및 방법이 원래의 목표 프로세서 또는 어떤 다른 의도하지 않은 목표 프로세서에서 다른 적법한 코드 세그멘트의 허가되지 않은 사용에 대한 보호를 제공하는 것이 바람직하다. 그러한 방법은 소프트웨어 바이러스 및 멀웨어를 제어하기 위한 싸움에서 가치있는 도구일 뿐만 아니라 디지털 콘텐트의 세계에서 저작권을 보호하는 유일하게 강력한 메카니즘이다.

[0022] 이 때문에, 이제 범용 코드 블록의 실행에 대한 고도의 특수한 제어를 제공하여 프로그래머가 주어진 코드 블록이 실행될 수 있게 하는 정확한 환경을 결정할 수 있게 하는 시스템 및 방법의 실시예에 주의가 기울여지고 있다. 이 조건들은 코드 블록이 실행하려고 하는 개별 장치, 코드 블록이 호출되는 환경, 실행 시간, 실행 순서, 및 코드 블록이 특수한 실행 스레드(execution thread)에서 호출된 횟수와 같은 제약들(그러나 상기의 것들로 제한되는 것은 아님)을 포함할 수 있다.

[0023] 이러한 제어 메카니즘은 예를 들면 회귀적 실행을 통하여 일 실시예로서 구현되는 호출된 코드 세그멘트 집합의 명시적으로 순서정해진 실행에 기초하여 데이터 은닉 시스템 및 방법의 실시예와 결합될 수 있다. 이러한 시스템 및 방법의 실시예가 방해없는 일반성 및 공격에 대한 보호 레벨과 함께 사용될 때 많은 다른 보안 시스템을 능가하는 시스템 및 방법이 얻어질 수 있다.

[0024] 실시예를 상세히 설명하기 전에, 본 발명의 실시예가 효과적으로 활용될 수 있는 아키텍쳐의 일반적 개관을 제공하는 것이 도움이 될 수 있다. 도 1은 그러한 토폴로지의 일 실시예를 묘사한 것이다. 여기에서, 콘텐트 분배 시스템(101)은 프로토콜 엔진을 포함하는 하나 이상의 목표 유닛(100)(엔드포인트 장치라고도 부름)에 디지털 콘텐트(예를 들면, 오디오 또는 비디오 데이터를 포함한 비트 스트림, 소프트웨어 애플리케이션 등일 수 있음)를 분배하도록 동작한다. 상기 목표 유닛은 예를 들면 유선 또는 무선 네트워크상의 컴퓨팅 장치 또는 네트워크가 없는 컴퓨터 장치의 일부일 수 있고, 상기 컴퓨팅 장치는 예를 들면 네트워크를 통해 비트스트림으로서 전달된 콘텐트 또는 예컨대 메일을 통해 전달될 수 있는 컴퓨터 판독가능 기억 매체상의 콘텐트를 재생할 수 있는 퍼스널 컴퓨터, 셀룰러 폰, 개인용 정보 단말기, 미디어 플레이어 등을 포함한다. 뒤에서 설명하는 바와 같이, 상기 디지털 콘텐트는 디지털 콘텐트의 실행에 대한 제어가 조절될 수 있고 보안이 디지털 콘텐트와 관련하여 구현될 수 있는 그러한 방법으로 구성 또는 분배될 수 있다.

[0025] 도 2는 디지털 콘텐트의 실행을 제어하거나 수신된 디지털 콘텐트와 함께 보안 프로토콜을 구현할 수 있는 목표 유닛의 일 실시예의 구조를 묘사하고 있다. 목표 유닛의 요소들은 목표 유닛(100)의 프로토콜 엔진에서 보안 방식으로 프로토콜을 구현하는 블록들의 집합을 포함할 수 있다. 이 블록들은 이 실시예에서 하드웨어로서 묘사되어 있지만, 소프트웨어를 이용하여 동일한 효력을 가진 유사한 기능을 달성할 수 있다는 것을 알 것이다. 또한, 어떤 실시예는 여기에서 설명한 블록들을 모두 포함할 수 있지만, 다른 실시예는 더 적거나 더 많은 블록들을 이용할 수 있다는 것을 알 것이다.

[0026] 이 블록들 중 첫번째는 실시간 클럭 또는 날짜/시간 레지스터(102)이다. 이것은 자유 동작 타이머(free-running timer)이고 중앙 서버와의 보안적 상호작용에 의해 세트 또는 리세트될 수 있다. 시간이 보안 시간 표준에 대한 질의를 행함으로써 확립될 수 있기 때문에, 이 기능을 온칩(on-chip)으로 하는 것이 더 편리하다. 목표 유닛(100)은 난수 발생기(180)를 또한 포함할 수 있으며, 난수 발생기(180)는 충분한 난수의 열(sequence)을 생성하도록 구성되고, 또는 의사 난수 발생 시스템에 종자값(seed value)을 공급하기 위해 사용될 수 있다. 상기 의사 난수 발생기는 잠재적으로 하드웨어, 소프트웨어 또는 "보안" 소프트웨어로 또한 구현될 수 있다.

[0027] 일방향(one-way) 해싱 함수 블록(160)은 실질적으로 하드웨어로 해싱 함수를 구현할 수 있다. 목표 유닛(100)의 다른 부분은 하드웨어 조력 암호화/복호 시스템(170)이고, 이것은 목표 유닛(100)의 비밀키 또는 공개/개인키(뒤에서 설명함)를 이용하여 키들을 실행가능 코드 블록으로 변환하기 위해 암호화 메시지에서 동작하거나 키들을 암호화 메시지로 변환하기 위해 비암호화 데이터에서 동작한다. 상기 복호 시스템(170)은 다양한 방법으로

구현될 수 있다. 이러한 일방향 해싱 함수 및 후속하는 암호화/복호 시스템의 조합은 임의의 디지털 데이터의 유효화를 위해 사용되는 디지털 서명 발생기를 구성할 수 있고, 데이터는 암호화 형태로 또는 평문 형태로 분배된다. 전체 프로토콜의 속도 및 보안은 이 블록의 구성에 따라 변할 수 있고, 따라서 보안 시스템 갱신을 수용하도록 충분한 융통성이 있고 시스템이 시간 임계(time-critical) 메시지의 실시간 복호를 수행하도록 충분히 빠르게 구성할 수 있다.

[0028] 이것을 염두에 두고 있으면 어떤 암호화 알고리즘이 이 하드웨어 블록(170)에 대하여 사용되는지에 관한 프로토콜의 정확한 구체화는 중요하지 않다. 최대의 융통성을 촉진하기 위하여, 실제 하드웨어는 비알고리즘적으로 특수한 방법으로 사용되도록 충분히 범용이라고 추정되지만, 이 메카니즘을 구현할 수 있는 많은 다른 수단이 있다. 이 점에서, 용어 암호화 및 복호는 암호화/복호를 수행하는 엔진(알고리즘, 하드웨어, 소프트웨어 등)을 지칭할 때 여기에서 상호 교환적으로 사용된다는 점에 주목하여야 한다. 알 수 있는 바와 같이, 특정 실시예에서 대칭 암호화가 사용되는 경우, 동일하거나 유사한 암호화 또는 복호 엔진이 암호화 및 복호 둘 다를 위해 사용될 수 있다.

[0029] 다른 블록은 실행될 코드를 저장할 수 있는 메모리(110)이다. 이것은 전형적으로 명령어 캐시(I-캐시)라고 알려져 있다. 일부 실시예에 있어서, 상기 I-캐시(110) 부분들의 중요한 특성은 특정 블록에 내포된 데이터가 CPU 실행 유닛(120)에 의해서만 판독될 수 있다는 것이다. 다시 말해서, I-캐시 메모리(130)의 상기 특수 블록은 실행만 가능하고 임의의 소프트웨어에 의해 판독 또는 기록될 수 없다. 이 I-캐시 블록은 여기에서 "보안 I-캐시"(130)라고도 부른다. 실행 대상 코드를 상기 보안 I-캐시 블록(130)에 저장하는 방법은 묘사되거나 묘사되지 않은 다른 블록에 의해서 수행될 수 있다. 정상 I-캐시(150)는 잘 알려져 있는 바와 같이 정상적으로 실행되는 코드를 저장하기 위해 사용된다.

[0030] 게다가, 일부 실시예에서, 특정 블록을 사용하여 보안 코드 블록의 동작을 가속화할 수 있다. 따라서, CPU(120)가 보안 코드를 실행하는 동안 액세스만 가능한 CPU 레지스터(140)의 집합이 지정될 수 있고 보안 코드 블록이 실행 종료시에 CPU 레지스터가 클리어되며(보안 코드 블록(130) 내의 명령어는 보안 모드라고 부른다), 또는 보안 I-캐시(130)에 저장된 코드의 실행 중에 어떤 이유로 비보안 즉 "정상" I-캐시 또는 다른 영역에 위치된 임의의 코드 섹션으로의 점프가 발생한다.

[0031] 일 실시예에 있어서, CPU 실행 유닛(120)은 보안 코드 블록(130)에 저장된 코드를 실행하는 동안 어떤 레지스터(140)가 판독 또는 기록되는지를 추적하고, 그 다음에 "보안 실행" 모드를 빠져나갈 때 상기 레지스터를 자동으로 클리어하도록 구성될 수 있다. 이것은 보안 코드가 그 자체 후에 신속히 "청소(clean-up)"되어 2종류의 코드 블록들 간에 공유되도록 허용된 데이터만이 본래대로 유지되게 한다. 다른 하나의 가능성은 보안 코드 블록(130) 내에서 실행되는 코드의 창조자(author)가 어떤 레지스터(140)가 클리어되어야 하는지를 명시적으로 식별할 수 있다는 것이다.

[0032] 보안 코드 세그멘트와 비보안 코드 세그멘트 사이에서 레지스터(140)에 저장된 데이터의 "누설"을 취급하는 다른 잠재적인 방법은 CPU(120)가 보안 코드를 실행할 때만 사용되는 레지스터(140)의 집합을 식별하는 것이다. 일 실시예에 있어서, 이것은 많은 현대의 CPU 설계에서 실시되고 있는 레지스터 리네이밍(renaming) 및 스코어보딩(scoreboarding) 메카니즘의 버전을 이용하여 달성될 수 있다. 만일 보안 코드 블록의 실행이 원자 동작(atomic action)으로 취급되면(즉, 중단할 수 없으면), 이것은 상기 리네이밍 및 스코어보딩의 구현을 더 쉽게 할 수 있다.

[0033] CPU(120)가 "보안" 코드(보안 코드 블록(130) 내의 코드)와 "비보안 코드"(정상 I- 캐시(150)와 같은 다른 위치 또는 메모리의 다른 위치 내의 코드)의 혼합물을 실행할 가능성이 거의 없어보인다 할지라도, 그러한 상황은 인터럽트 루틴으로 점프할 때와 같이 콘텍스트를 스위칭하는 처리에서, 또는 CPU(120) 콘텍스트가 어디에 저장되어 있는가에 따라서 발생할 수 있다(대부분의 CPU는 메인 메모리에 콘텍스트를 저장하지만, 메인 메모리는 잠재적으로 비보안 코드 블록에 의해 발견 및 조작되기 쉽다).

[0034] 이러한 우발성에 대한 보호를 돕기 위해, 일 실시예에 있어서, 중간 실행(mid-execution)이 시스템 내의 다른 실행 스레드에 노출되는 것이 금지되고 있는 보안 코드 블록의 실행 중에 얻어진 결과를 보호하기 위해 사용될 수 있는 다른 방법은 프로세서가 보안 실행 모드에서 동작중인 동안 스택 푸시(stack push)를 디스에이블 하는 것이다. 이러한 스택 푸시의 디스에이블은 만일 보안 코드 블록이 그 정상적인 종료 전에 인터럽트되면 보안 코드 블록은 재개시(resume)될 수 없고 따라서 처음부터 다시 시작되어야 한다는 점에서 보안 코드 블록이 인터럽트 불능이라는 것을 의미한다. 특정 실시예에서, 만일 "보안 실행" 모드가 프로세서 인터럽트 중에 디스에이블되면 보안 코드 블록은 전체 호출 사슬(calling chain)이 재시작되지 않는 한 잠재적으로 재시작될 수 없다.

[0035] 각 목표 유닛(100)은 2세트의 비밀키 상수(104)를 또한 가질 수 있고, 그들의 값은 어느 것도 소프트웨어에 의해 판독될 수 없다. 이들 키의 첫번째(1차 비밀키)는 비밀키의 집합으로서 구성될 수 있고, 그들 중에서 하나만이 임의의 특정 시간에 판독될 수 있다. 만일 유닛의 "소유권"이 바뀌면(예를 들면, 프로토콜 엔진을 내포한 설비가 판매되거나 그 소유권이 다른 방식으로 이전되면), 현재 활성인 1차 비밀키는 "클리어"되거나 다른 값으로 덮어쓰기 될 수 있다. 이 값은 보안 방식으로 유닛에 전달될 수도 있고, 또는 상기 제1 키가 클리어된 때에만 사용되는 방식으로 유닛에 이미 저장되어 있을 수도 있다. 사실, 이것은 유닛의 소유권이 변경되었을 때 또는 그러한 변경에 관한 어떤 다른 이유가 있는 경우(절충키(compromised key)와 같이) 새로운 1차 비밀키를 상기 특수 유닛에 발행하는 것과 등가이다. 2차 비밀키는 목표 유닛(100) 자체와 함께 사용될 수 있다. 목표 유닛(100)의 CPU(120)는 1차 또는 2차 비밀키의 값에 액세스할 수 없으므로, 어떤 면에서 목표 유닛(100)은 그 자신의 비밀키(104)까지도 알지 못한다. 이 키들은 목표 유닛 CPU(120)의 보안 블록 내에 저장되어 사용될 뿐이다.

[0036] 다른 키 집합은 임시 공개/개인키 시스템(비대칭 키 시스템 또는 PKI 시스템이라고도 알려져 있다)의 일부로서 동작할 수 있다. 이 쌍의 키들은 급하게(on the fly) 생성되고, 중앙 서버의 개입없이 유사한 유닛들 간의 보안 통신 링크를 확립하기 위해 사용될 수 있다. 이러한 시스템의 보안성이 등가 키 길이 대칭 키 암호화 시스템의 보안성보다 전형적으로 더 낮기 때문에, 이 키들은 전술한 비밀키 집합보다 사이즈가 더 크다. 이 키들은 다른 무엇보다도 "재전송 공격"에 대하여 보호하기 위해 온칩 타이머 블록에서 제시되는 값과 함께 사용될 수 있다. 이 키들은 급하게 생성되기 때문에, 키들이 생성되는 방법은 난수 발생 시스템(180)에 의존할 수 있다.

[0037] 일 실시예에 있어서, 특수 목표 유닛의 "소유권" 변경에 영향을 줄 수 있는 하나의 방법은 우리가 타임스탬프 또는 타임스탬프 값이라고 부르는 다른 키(107)와 함께하는 합성키(compound key)로서 1차 비밀키를 항상 사용하는 것이다. 왜냐하면, 상기 다른 키의 값은 변경될 수 있고(다시 말해서 다른 시간에 다른 값을 가질 수 있음), 반드시 현재 일시를 반영하지 않기 때문이다. 이 타임스탬프 값 자체는 구조적으로 가시적일 수도 가시적이 아닐 수도 있지만(즉, 반드시 비밀키일 필요가 없다), 그럼에도 불구하고 목표 유닛이 보안 실행 모드에서 동작하지 않는 한 수정될 수 없다. 그 경우에, 1차 비밀키가 사용될 때마다 합성 키의 성분으로서 타임스탬프 값의 일관된 사용은 본질적으로 마치 1차 비밀키가 별도의 값으로 전환된 것과 동일한 효과를 만들 수 있고, 따라서 1차 비밀키 자체를 수정할 필요없이 특수 목표 엔드포인트 유닛의 "소유권 변경"을 효과적으로 가능하게 한다.

[0038] 이제, 목표 유닛의 일방향 해싱 함수 하드웨어에 대하여 더 상세히 설명한다. 이제 도 15를 참조하면, 후속되는 반복에서 일방향 해싱 함수를 위한 종자값으로서 회귀형 보안 프로토콜의 1회 반복시에 생성된 디지털 서명 동작의 결과를 이용할 수 있는 일방향 해싱 함수 블록의 일 실시예가 도시되어 있다. 일 실시예에 있어서, 목표 유닛의 상태는, 목표 유닛이 보안 실행 모드에서 동작하는지 아닌지에 관계가 있는 한, 여기에서 "보안 모드 인에이블" 비트라고 부르는 하드웨어 비트(1570)의 값에 의해 반영될 수 있다.

[0039] 여기에서, 이 하드웨어 비트의 디폴트 상태는 클리어될 수 있다(즉, 목표 프로세서의 디폴트 상태는 보안 실행 모드에서 동작하지 못하게 한다). 특정 실시예에서 이 비트와 일방향 해싱 함수 하드웨어 블록(1561)과의 상호작용은 2개의 부분으로 나누어 설명할 수 있다. 첫번째(비보안) 경우에, 하드웨어 비트가 세트(예를 들면 "1"로 되는 것, 그러나 특정의 아키텍처에서는 이 비트가 "0"의 값을 가질 때 "세트"로 간주될 수 있다)일 때에만 비밀 하드웨어 키의 사용을 허용하도록 "보안 모드 인에이블" 비트가 문지기로서 작용하기 때문에, 비밀 하드웨어 키(1540)에 대한 모든 액세스는 차단된다. 또한, 이 경우에, 디지털 서명 레지스터(1564)의 출력은 피드백되어 일방향 해싱 함수(1561)의 입력 "종자"(1510)를 형성한다. 따라서, 프로세서가 "비보안 실행" 모드에서 동작하는 동안 임의의 일방향 해싱 함수 동작의 중간 결과가 피드백되어 임의의 후속 일방향 해싱 함수 동작을 위한 종자를 형성한다. 이것은 네스트된(nested) 또는 연쇄 기능(concatenated function)의 집합의 전체 호출 사슬(calling chain)과 동일한 가동 체크섬(running checksum)이 유지될 수 있게 한다. 실행을 시도하는 각 코드 블록의 실행이 허용되기 전에 상기 일방향 해싱 함수로 먼저 평가된 경우에, 임의의 주어진 코드 블록의 전체 호출 사슬은 이 단일 메카니즘으로 실질적으로 명확하게 결정될 수 있다.

[0040] 마찬가지로, "보안 모드 인에이블" 비트가 세트인 경우(즉, 프로세서가 "보안 실행 모드"에서 동작하는 경우)에, 비밀 하드웨어 키는 액세스 가능하다(다시 말해서, 그 값이 프로세서 자체에 의해 직접 액세스할 수 없는 경우에도 직접 액세스가능하거나, 적어도 그 값이 계산 동작에서 사용될 수 있다). 게다가, 보안 실행 모드에서 동작할 때, 디지털 서명 레지스터의 출력은 피드백되지 않고 일방향 해싱 함수의 후속 평가를 위한 종자값을 형성한다. 이 디지털 서명 발생기 블록의 정확한 구현은 뒤에서 더 자세히 설명한다. 이제 알 수 있는 바와 같이, 특정 실시예에서, 특수 코드 블록의 전체 호출 사슬은 시스템 와이드 소프트웨어 또는 하드웨어 유효화(또는 인증) 동작과 같은 수단을 사용할 필요없이 그 보안 실행 전에 유효화될 수 있다. 타임스탬프 레지스터

와 관련하여 위에서 설명한 경우에서처럼, 특정 실시예에 있어서, 이 "보안 모드 인에이블" 비트는 프로세서에 대해 구조적으로 가시적이거나 가시적이 아닐 수 있지만, 그 상태는 프로세서에 의해 명시적으로 세트되지 않을 수 있다는 점에 주목한다. 이 하드웨어 비트는 비보안 코드 세그멘트를 호출함으로써 디폴트 값으로 리세트될 수 있지만, 일 실시예에 있어서, 이 비트가 세트될 수 있는 유일한 방법은 하드웨어 부분에서의 직접 동작을 통하는 것이다. 비트가 구조적으로 가시적인 경우에, 프로세서가 보안 실행 모드에서 동작중인지 아닌지는 명시적으로 결정될 수 있다. 비트가 구조적으로 가시적이지 않은 경우에, 결정은 그럼에도 불구하고 그 값이 어쨌든 하드웨어 비밀키에 의존한다는 어떤 표현을 평가함으로써 암시적으로 행하여질 수 있다.

[0041] 이제, 코드 실행의 제어 및 보안 프로토콜의 구현에 밀접한 관계가 있는 주제(subject) 하의 기본적인 문제점에 대하여 구체적으로 설명한다. 그래서 전술한 하드웨어의 실시예를 이용하여 임의의 범용 프로세서에서 임의 코드의 실행을 제어하는 법 및 이들 시스템 및 방법의 실시예가 보안 프로토콜 및 시스템과 함께 어떻게 효과적으로 사용되어 효과적인 전체 보안 시스템을 구성하는지를 알 수 있다.

[0042] 비밀키 은닉

[0043] 상업적 디지털 콘텐트 전송 시스템의 대부분은 디지털 미디어 데이터가 자유롭게 복제 및 분배되는 것으로부터 보호하기 위해 소정 형태의 암호화 또는 데이터 은닉(data hiding)을 포함한다. 대부분의 경우에, 데이터 은닉 전략은 콘텐트 보호와 관련하여 완전히 비효과적인 수단이라는 것이 결국 입증되었다. 이러한 은닉이 비성공적인 것으로 입증된 중요 이유중의 하나는 노출로부터 보호되어야 할 정확한 데이터가 그럼에도 불구하고 임의의 인증된 당사자에게는 자유롭게 이용할 수 있어야 한다는 것이다. 따라서, 디지털 콘텐트의 분배에 있어서 외관상 모순이되는 일련의 필요조건이 존재한다.

[0044] 모든 의도된 수령자에 대하여 원래의 디지털 콘텐트가 별도로 암호화될 수 있는 경우 및 의도된 수령자만이 분배된 디지털 콘텐트를 실제로 사용할 수 있는 경우에, 시스템의 보안성은 잠재적으로 아주 양호할 수 있다. 그러나, 다수의 특수한 조건들이 부합되지 않으면, 이러한 종류의 시스템의 보안성은 몇 가지 점에서 불충분한 것으로 보여질 수 있다. 첫째로, 이러한 시스템은 분배된 데이터 집합 전체가 각각의 의도된 수령자에 대하여 별도로 재암호화될 것을 필요로 한다는 점에서 덜 효율적이다. 둘째로, 분배자는 범용 프로세서에서 인증되지 않은 복호가 불가능하다는 것을 보장할 필요가 있다. 셋째로, 각각의 개별 수신 장치는 일부 다른 엔드포인트 장치에서 쉽게 복제(또는 범용 프로세서에서 에뮬레이트)될 수 없는 일부 속성을 가져야 한다는 점에서 유일한 것이어야 한다. 만일 상기 마지막 2가지 조건 중의 어느 하나가 위배되면, 이 시스템은 개별적으로 암호화된 데이터 뿐만 아니라 그 데이터와 관련된 장치 지정 키 둘 다를 단순히 가로챔(intercepting)으로써 공격을 받기 쉽다.

[0045] 사실, 이러한 시스템의 보안성은 각 수신 장치의 유일한 속성(unique attribute)의 보안성에 기초를 두는 것으로 보여질 수 있다. 이 유일한 속성은 전형적으로 분배자 및 인증된 수령자에게만 공지된 비밀키를 이용하여 구현된다. 비록, 원칙적으로, 이러한 종류의 셋업은 유효한 보안 시스템이 될 수 있지만, 각 수령자에 대하여 원래의 디지털 콘텐트를 별도로 암호화하는 필요조건은 대부분의 용도에서의 실제 구현을 비실용적으로 만든다. 최초의 디지털 콘텐트를 1회만 암호화하고 모든 잠재적인 인증된 당사자에게 동일하게 분배되는 것이 바람직하다면, 문제점은 데이터 은닉의 문제로 되돌아간다. 이러한 종류의 문제점은 방송 암호화 분야에서 공지되어 있다.

[0046] 거의 모든 종류의 분배형 비밀 데이터 시스템과 관련된 기본적인 문제점들 중의 하나는, 대부분의 경우에, 보안 시스템의 별도의 엔티티들 사이에서 이리저리로 흐르는 모든 메시지 및 데이터가 일반적으로 개방 상태로 전송되고 따라서 도청자(eavesdropper)에 의해 포착될 수 있다는 것이다. 따라서, 그러한 시스템의 개별 컴포넌트 사이에서 전송되는 임의의 메시지 및 데이터는 인증되지 않은 당사자에 의해 가로채는 것으로부터 보호하기 위해 암호화되어야 한다. 이러한 시스템에서 고려해야 할 다른 하나의 문제점은 임의의 이러한 비밀 데이터 전송시에 전송자 뿐만 아니라 수신자 둘 다의 신원을 검증하는 것이다. 두 당사자가 서로 알지 못하는 경우에, 전형적으로 상호 신뢰되는 매개 전략이 사용된다.

[0047] 그러나, 비밀 데이터가 그 목적지에 도달하면, 취급되어야 할 동일하게 어려운 문제점은 절충되지 않은 방식으로 그 비밀 데이터를 어떻게 안전하게 사용할 것인가이다. 이러한 예방책은 적법한 엔드포인트마저도 잘못된 정보가 제공됨으로써 절충되는 보안성을 가질 수 있기 때문에 일반적으로 필요하다. 그래서, 분배중에 인증되지 않은 발견에 대한 보호 외에도, 가끔은 비밀 데이터가 그 비밀 데이터에 대해 다른 인증되지 않은 사용자에 의해 발견되는 것으로부터 보호하는 것이 요구된다.

[0048] 일 실시예에 있어서, 이러한 요구되는 제어는 구조적으로 은닉된 비밀키 또는 실행 전에 실시간으로 복호되어야 하는 암호화 목적 코드 블록의 단순한 시간 의존성 사용을 이용하여 구현될 수 있다. 첫번째 경우에, 코드 블록 실행은 제어 메카니즘에 완전하게 투명할 수 있고, 이것은 실행 속도가 최소로 영향을 받아야 한다는 것을 의미한다. 후자의 경우에, 동작 대상의 코드 블록은 실행 전에 복호되어 복호 처리의 대기 시간에 기인하는 동시 성능 손실을 받을 가능성이 높다. 그러나, 이 후자의 경우에, 목적 코드는 분해(disassembly)로부터 비교적 안전하고, 따라서 예비(would-be) 공격자에 의해 파괴되기가 잠재적으로 더 어렵다. 나중에 여기에서 설명하는 실시예는 고도로 안전한 암호화 목적 코드 방법으로부터 비교적 높은 성능이지만 여전히 매우 안전하고 선택적으로 이용가능한 비밀키 방법까지에 걸치는 가능한 솔루션의 큰 연속체로 구현될 수 있는 시스템 및 방법을 개시한다.

[0049] 일 실시예에 있어서, 비밀키의 사용자로부터 비밀키를 은닉하는 것은 하버드 아키텍쳐 메모리 스페이스 분기(Harvard Architecture memory space bifurcation)와 유사한 방법으로 달성될 수 있다. 그러나, 이 실시예에서, 비밀키가 암호화/복호 계산에서 사용될 수 있지만 실제로는 프로세서에 의해 직접 판독되지 않는 차이점이 있다. 이러한 차이는 하드웨어의 설계시에 고정된, 하드웨어로 구현되거나 소프트웨어 매크로(마이크로 코드라고도 알려져 있음)를 미리 결정한 것에 대한 암호화/복호 동작을 제한함으로써 시행될 수 있다. 예를 들면, 하드웨어 비밀키가 임의 코드에 의해 사용되는 경우, 비밀키가 프로세서에 의해 직접 판독될 수 없다 하더라도 비밀키는 단순한 계산에 의해 쉽게 결정될 수 있다. 따라서, 단지 보안 관련 계산이 하드웨어 비밀키에 액세스하여 코드 세그멘트를 더 범용이지만 덜 안전한 코드 블록으로부터 구별하는 것이 요구될 수 있다.

[0050] 이러한 구별은 특정 실시예에서 여기에서 설명한 것과 실질적으로 유사한 유효화 방법을 이용하여 달성될 수 있다. 위에서 설명한 적응 디지털 서명 방법의 실시예가 하드웨어 비밀키의 액세스 가능성을 결정하기 위해 사용된 경우 목표 프로세서가 보안 관련 계산(즉, 목표 프로세서가 "보안 실행" 모드에서 동작할 때 수행하는 계산) 및 보안되지 않은 계산을 실행하는지를 쉽고 신뢰성있게 결정할 수 있다. 또한, 위에서 언급한 것과 실질적으로 유사한 회귀적 방법을 이용하여 최종 계산이 완료되고 완전히 디코드된 결과가 보고될 때까지 복구로부터 숨겨진 임의의 중간 키 결과를 유지할 수 있다. 따라서, 여기에서 설명한 실시예들은 동일한 비트스트림을 생성하기 위해 사용된 비밀 글로벌 키를 노출시키지 않고 암호화 디지털 비트스트림을 디코드하는 능력이 있다.

[0051] 코드 실행 제어

[0052] 주어진 프로세서에서 특수 코드 세그멘트가 안전하게 실행되는 것을 보장하는 방법은 수년 동안 폭넓게 연구되어 왔다. 보안 코드 실행 보호를 생성하기 위한 일부 초기의 시도는 "특권(privileged)" 명령어 집합을 확립하기 위해 프로세서 구조를 변경하는 것을 포함하였다. 이 특권 명령어는 프로세서가 "수퍼바이저(supervisor)" 또는 "커넬(kernel)" 모드라고 알려져 있는 특수 모드에서 동작할 때에만 상기 특권 명령어가 실행될 수 있도록 구조를 설계함으로써 안전하게 되었다. 이러한 종류의 프로세서 구조의 분기는 프로세서 일반성의 잠재적 손실 및 잠재적 성능 감퇴를 포함한 많은 단점이 있다. 이러한 단점 외에도, 상기 보호 수단은 프로세서가 수퍼바이저 모드에서 실행하는 동안 예기치않은 실행 경로의 잇점을 취하는 방식으로 표준 시스템 루틴에 대한 특수 설계된 소프트웨어 콜을 사용함으로써 가끔 우회될 수 있다. 이러한 특수 설계된 멀웨어 공격의 예로는 소위 "스택 오버플로우", "스택 오버런" 및 "코드 주입" 공격이 있다.

[0053] 체크섬 검증 또는 아규멘트 바운즈(argument bounds) 체킹의 각종 수단에 주로 기초해서 상기 종류의 익스플로잇(exploit)에 대한 보호를 돕는 시도에서 많은 전략이 고안되었다. 이러한 종류의 보호 수단에도 불구하고, 다형 바이러스(polymorphic viruses)(즉, 자기 수정 코드)를 포함한 다양한 카운터-카운터-방책이 안출되었다. 바운즈-체킹에도 불구하고 프로세서 약점을 이용하는 다른 전략은 바운즈 체킹 "수퍼바이저" 루틴 자체를 단순히 우회하는 것을 포함한다. 이러한 종류의 익스플로잇은 우회하는 각종 복사 보호 시스템에서 아주 가끔 사용된다. 이것이 발표됨으로써, 수퍼바이저 루틴을 강탈하는 전략은 컴퓨터 보안 세계에서 유일한 것이 아니고 전혀 새로운 개념이 아니다. 사실, 이 정확한 문제는 다양한 애플리케이션에서 유사체(analog)를 가지며 플라토 시절까지 그의 저서 "더 리퍼블릭(The Republic)"에서 참조되었다. 기본적인 문제점은 임의의 주어진 시스템에서 글로벌 수퍼바이저의 일부 종류를 항상 식별할 수 있고, 이것으로 궁극적인 보안 또는 구조의 안정성이 신뢰된다는 것이다. 후속하는 모든 보안 기능에 대한 이러한 총체적 토대의 개념은 보안 시스템에 관한 현대의 연구에서 "신뢰의 근간(Root-of-Trust)"이라고 알려져 있다.

[0054] 더 최근에는 프로세서가 본래 읽기만 되는 명령어를 인출하는 메모리 세그멘트를 제한함으로써 수퍼바이저 루틴 공격에 대해 프로세서를 보호하려는 일부 시도가 있었다(이것은 소위 W^X 또는 "기록-XOR-실행" 방법을 포함한다). 다른 범용 컴퓨터의 메모리 스페이스를 데이터 기반 및 코드 기반 파티션으로 분할하는 개념은 소위 "하버

드 아키텍처"의 변형체로 보여질 수 있다. 이 방법은 보호 메카니즘과 관련된 특정의 성능 페널티뿐만 아니라 메모리 활용에서의 동시 증가를 갖는다. 마지막으로, 최근에는 이러한 종류의 방어조차도 소위 "복귀 기반(return-based)" 프로그래밍 익스플로잇의 사용에 의해 또는 단순한 메모리 앨리어싱 익스플로잇에 의해 교묘히 피할 수 있는 것으로 또한 보여지고, 2개의 별도의 실행 스레드는 다른 모드에 있는 동일한 메모리 블록(예를 들면, 하나는 "데이터 모드"에 있고 다른 하나는 "실행 모드"에 있는 것)을 참조할 수 있다.

[0055] 프로세서의 실행 스레드가 강탈되는 것으로부터 보호하는 다른 제안된 수단은 암호화 코드 블록의 사용을 포함한다. 이 방법에서, 실행되는 코드 세그멘트는 미리 암호화되고, 따라서 프로세서에 로딩되기 전에는 판독할 수 없다(아마도 더 중요한 것은 변경할 수 없다는 것이다). 이 방법은 또한 몇 가지 약점을 갖는다. 첫째로, 코드 세그멘트 자체는 보호될 수 있지만, 아규멘트에는 반드시 동일 수준의 보안이 제공되지 않는다. 따라서, 완전히 적법하고 안전한 코드 세그멘트는 그럼에도 불구하고 코드 세그멘트가 예기치 않은 형태로 행동하게 할 수 있는 호출 루틴으로부터 가짜 아규멘트가 제공될 수 있다. 둘째로, 일부 경우에, 실행 스레드는 전술한 복귀 기반 프로그래밍 공격에 대하여 반드시 보호되지 않는다. 또한, 프로세서 버스가 공격자에게 쉽게 관찰될 수 있는 경우에는 올바르게 실행된(암호화되었다 하더라도) 코드 세그멘트의 장기 관찰 및 실행가능 스트림에 주입된 부적절하게 암호화된 코드 세그멘트에 의해 야기되는 실행 오류의 관찰이 수정된 사전 공격(dictionary attack) 방법을 이용하여 암호화 키의 누설을 도울 수 있다. 마지막으로, 이러한 시스템에서의 프로세서 성능은 유사하지만 암호화되지 않은 코드 시스템을 통해 반드시 심각하게 감퇴된다. 이러한 성능 페널티는 많은 이슈에 의해 야기될 수 있는데, 그 중 가장 중요한 것은 코드 블록이 메모리로부터 인출될 때와 코드 블록이 실행 가능으로 될 때 사이에 코드 블록의 필수적인 복호에 의해 초래되는 대기시간(latency)이다. 비록 대부분의 현대 프로세서가 (각종 수단에 의해) 병렬로 실행될 수 있는 명령어의 수를 증가시키기 위해 파이프라인 메카니즘을 이용하지만, 암호화 코드의 블록은 먼저 복호되기 전에 파이프라인에 읽어들일 수 없다. 코드가 빈번하게 분기하는 경우에는 복호 처리가 하드웨어 조력 복호를 이용하는 경우에도 코드 실행 자체보다 훨씬 더 오래 걸릴 수 있다.

[0056] 본 발명에 따른 시스템 및 방법의 실시예는 비암호화 코드 블록의 활용을 가능하게 하고, 그래서 암호화 실행가능과 관련된 성능 페널티가 더 적은 이슈로 된다. 그러나, 실행 효율이 실질적인 관심사가 아닌 경우에는 암호화 코드 블록이 여전히 활용될 수 있다. 따라서, 여기에서 설명하는 실시예는 평문 실행가능의 효율뿐만 아니라 동일하거나 유사한 방법 및 시스템을 활용한 암호화 코드 세그멘트의 추가된 보안성을 둘 다 가질 수 있다. 또한, 여기에서 설명하는 보안 시스템 및 방법의 실시예는 새로 발견된 보안 관심사를 취급하기 위해서 및 실시예가 이미 전개된 후에 새로운 기능을 추가하기 위해서 원위치(in-situ)로 갱신될 수 있다.

[0057] 본 발명의 실시예는 암호적 해싱 함수에 의한 실행 전에 "보안 코드 세그멘트"가 유효화되는 것을 보장함으로써 상기 장점을 획득할 수 있다. 이 유효화는 예를 들면 보안 코드 세그멘트용으로 생성된 메시지 다이제스트 또는 디지털 서명을 인증함으로써 달성될 수 있다. 이러한 암호적 해싱 함수의 평가가 위에서 설명한 합성키 구조를 이용한 결과적인 메시지 다이제스트의 암호화와 함께 발생하여 디지털 서명을 형성하는 경우에는 특수 코드 블록이 특정 목표 유닛 또는 프로세서와 관련하여 유일하게 될 수 있다. 이 프로세스는 특정 실시예에서 보안 코드 블록이 합성키 기반 디지털 서명을 이용하여 특정 목표 유닛에 암호적으로 결합될 수 있다는 사실에 기초하여, "보안 코드 결합"이라고 여기에서 인용된다.

[0058] 비록 이러한 해싱 함수의 실행이 리소스 없이 될 수 있지만, 이 방법의 장점은 보안 코드 세그멘트가 그 암호적 유효화를 완료하기 전에 실행 파이프라인에 도입될 수 있다는 점이다. 따라서, 해싱 함수는 (투기적 분기 실행과 유사한 방법으로) 보안 코드 세그멘트 자체의 실행과 나란하게 평가될 수 있다. 이 실시예에서, 보안 코드 세그멘트의 결과는 결과적인 메시지 다이제스트가 진정한 것으로 판정된 경우에만 활용될 수 있다. 그러나, 다른 실시예에서, 보안 코드 세그멘트의 결과가 후속 동작에서 활용될 수 있지만, 그 결과들은 프로세서가 보안 실행 모드에서 동작하는지 아닌지에 따라서 달라질 수 있다. 이 실시예는 디지털 서명의 사용을 위해 합성키의 평가에 대하여 위에서 설명한 프로세스와 실질적으로 유사하고, 도 15에 도시한 것과 같은 하드웨어를 사용함으로써 생성될 수 있다.

[0059] 그러나, 암호적 유효화의 사용은 암호화 코드 세그멘트의 사용을 배제하지 않는다. 사실, 올바르게 복호된 코드(임의 유형의 암호화를 적용하기 전의 최초 상태의 보안 코드 세그멘트)의 메시지 다이제스트 또는 디지털 서명의 사용은 추가적인 수준의 보호를 제공할 수 있다. 이것은 장래의 공격자가 모조 메시지 다이제스트를 생성하기 위해 올바르게 복호된 코드 블록에 대한 선험적 지식(a-priori knowledge)을 갖고 있어야 한다는 사실에 기인한다. 따라서, 코드 세그멘트 유효화뿐만 아니라 암호화 코드 메서드가 동시에 사용될 수 있다면, 공격에 대한 더 높은 강함(robustness)을 실현할 수 있다.

[0060] 그러나, 역시 실현될 수 있는 것처럼, 상기 해싱 유효화를 우회할 수 있는 몇 가지 방법이 있고, 그 중 가장 간단한 것은 해싱 함수 자체를 파괴하는 것이다. 이 전략이 특정 실시예에서는 가능하지 않다 하더라도(예를 들면, 하드웨어 해싱 함수를 활용함으로써), 적절히 유효화된 메시지 다이제스트와 함께 사기(impostor) 코드 세그멘트를 제공함으로써 상기 실시예의 보안성을 공격하는 것이 여전히 가능하다. 많은 메시지 다이제스트가 실제로 암호화되어 디지털 서명을 형성하기 때문에, 외관상으로 이 공격 전략은 어려워보인다. 그러나, 디지털 서명 메카니즘조차도 디지털 서명의 공개키 조사 부분을 눈속임하고 그에 따라서 사기 디지털 서명의 인위적인 유효화를 제공함으로써, 또는 대안적으로 서명 유효화 루틴 자체의 악의적인 파괴에 의해 잠재적으로 공격받을 수 있다.

[0061] 이러한 제한은 보안 코드 세그멘트와 관련된 메시지 다이제스트를 이중 암호화, 즉 한번은 (총체적인) "창조자"의 비밀키를 이용해서 암호화하고 그 다음에 엔드포인트 "제조자"(실제로 최초의 칩 제조자는 아닐지라도 2차 레벨 분배자, 시스템 통합자, 서비스 공급자 등일 수 있음) 및 당해 코드 세그멘트가 실행되는 특수 엔드포인트 장치에만 공지된 비밀키를 이용하여 다시 암호화함으로써, 여기에서 설명하는 시스템 및 방법의 실시예로 극복된다. 이 실시예의 장점은 상기 디지털 서명이 유사한 엔드포인트 장치들 간에 공유된다 하더라도 상이한 목표 유닛의 비밀키가 서로 다르기 때문에 의도된 목표 유닛에서 올바르게 기능한다는 점이다. 따라서, 임의의 이러한 디지털 서명은 명문으로 전송 및 저장될 수 있다.

[0062] 비밀을 이중으로 암호화하는 기술의 실시예(이것은 소위 "층을 이룬 키(layered key)" 시스템에서뿐만 아니라 회귀형 보안 시스템에서 사용될 수 있다)는 만일 올바르게 사용되지 않는 경우 특정의 이슈를 가질 수 있다. 첫째로, 만일 상기 층을 이룬 암호화 처리의 중간 결과가 인터셉트되면, 이 중간키는 임의의 또는 모든 시스템에서 더 높은 수준의 보안을 우회하도록 사용될 수 있다. 또한, 층을 이룬 시스템의 "최저층"에서 상기 중간 결과는 실제로 "글로벌" 복호키이고, 이 복호키는, 만일 발견되면, 모든 실질적으로 유사한 엔드포인트 장치용의 전체 보안 시스템을 우회하도록 사용될 수 있다. 이러한 종류의 "인터셉트" 공격은 복호 처리중에 임의의 메모리 트랜잭션을 단순히 주시하고 그 다음에 잠재적인 글로벌 복호키의 모든 메모리 위치를 시험함으로써 1회 이상 발생된다. 복호 처리 중에 모든 메모리 액세스를 주시하는 처리는 먼저 귀찮아 보일 수 있지만, 이것은 비밀키의 가치에 대한 무차별적 추측(brute-force guessing)보다 거의 확실히 더 효율적인 공격 전략이다. 층을 이룬 키 시스템의 제2의 잠재적 약점은 재전송 공격(replay attack)의 변형체에 의해 이용될 수 있다. 층을 이룬 시스템의 보안이 절충되고 그 키가 갱신되어야 하는 경우에, 오래된 (절충된) 키는 만일 최초 시스템이 그 이전 상태로 다시 리세트되거나 그 이전 상태가 사기 유닛에 의해 복제(clone)되었으면 여전히 사용될 수 있다.

[0063] 이러한 약점은 "층을 이룬 키" 구조와 대조적으로 우리가 "합성키"라고 부르는 것을 이용하여 여기에서 설명하는 실시예에서 취급될 수 있다. 합성키와 층을 이룬 키의 중요한 차이점 중의 하나는 전자의 모든 세그멘트가 단일의 모노리식 패스(monolithic pass)로 평가될 수 있다는 점이다. 대조적으로, 층을 이룬 키 시스템에서는 "최외측" 층 키가 먼저 평가되고 그 다음에 최내측 키(이것은 다음 층 키를 생성하기 위한 아규멘트로서 사용되는 등으로 전체 키 스택이 관통될 때까지 계속된다)로 복귀한다. 이 시스템과 관련된 문제점은 하위 레벨의 키가 인터셉트되고 나중에 사용되어 최외측 보안층을 효과적으로 우회할 수 있다는 점이다. 따라서, 이러한 층을 이룬 키 실시예에서는 가장 중요한(이 경우에는 글로벌) 키가 사슬에서 마지막으로 생성 및 사용되는 것이고, 이때 임의의 추가적인(또는 더 최근의) 보안층은 완전히 없다.

[0064] 이 때문에, 이러한 보안 스택을 관통하는 더 강력한 방법은 스택이 "인사이드 아웃(inside out)"으로부터 관통되도록 활용될 수 있다. 이것은 보안 사슬에 대한 가장 최근의 추가가 시퀀스에서 최초로 실행된 것이지만 사실상 보안 시스템의 최내측 층에 위치된 것임을 의미한다. 따라서, 실시예는 상기 "인사이드 아웃" 실행 순서화(ordering)를 시행하도록 사용될 수 있다. 코드 스택 관통의 이러한 특수 순서화는 단순한 반복 방법을 이용하여 달성될 수 있고, 이때 코드 루프는 먼저 현재 보안 수준을 평가하고 그 다음에 그에 따라서 분기한다. 그러나, 반복적 방법에 있어서, 보안 시스템 관통의 중간 결과는 위에서 설명한 것처럼 공격자가 적법한 보안 시스템 관통에서 다음 하위 레벨 키가 노출되기를 단순히 기다리고 그 인터셉트한 키를 시스템의 가짜 "초기" 버전을 복제하기 위해 사용하기 때문에 잠재적으로 우회될 수 있다. 따라서, 시스템 및 방법의 실시예는 이러한 "인사이드 아웃" 실행 순서화를 시행할 수 있을 뿐만 아니라 중간 결과가 악성 코드 또는 다른 잘 알려진 보안 시스템 익스플로잇에 의해 인터셉트되는 것으로부터 보호할 수 있는 것을 설명한다.

[0065] "인사이드 아웃" 보안 방법을 이용할 때의 다른 중요한 장점은 호출 아규멘트의 전체 시퀀스가 보안 시스템의 최내측 층(및 대부분의 최근 버전)에 보여질 수 있다는 점이다. 만일 이 "인사이드 아웃" 실행 시퀀스가 적절히 구현되면, 그 시스템에서 사용되는 올바르게 구성된 바운즈-체킹 메카니즘은 전체 호출 사슬에 대하여 가시성을 갖는다는 것을 알 수 있다. 따라서, 실시예는 시스템 인증 기능의 상당량을 일반적으로 그 기능과 관련된 임의

의 추가적인 성능 페널티의 발생없이 수행하기 위한 내장(built-in) 메카니즘을 구비할 수 있다.

[0066] 따라서, 특정 실시예는 중간 키를 상위 레벨(및 그에 따라서 보안성이 약한) 보안 시스템 루틴에 노출되는 것으로부터 지키고 적절한 보안 스택 관통 방법을 보장하는 수단을 활용할 수 있다. 이것을 달성하는 한가지 방법은 회귀형 보안 구조를 이용하는 것이고, 그 일 실시예가 "디지털 저작권 제어를 위한 회귀형 보안 프로토콜의 방법 및 시스템(Method and System for a Recursive Security Protocol for Digital Copyright Control)"이라는 명칭으로 윌리엄 브이. 옥스포트가 2003년 6월 19일에 출원한 미국 특허 출원 제10/465,274호에 개시되어 있고, 이 특허 출원은 그 후 2007년 4월 10일에 미국 특허 제7,203,844호로 특허되었으며, 모든 목적으로 인용에 의해 여기에 통합된다.

[0067] 이러한 회귀형 보안 프로토콜의 실시예를 활용하면, 특정의 장점들이 실현될 수 있다. 첫째로, 스택 순서 관통을 "인사이드 아웃"으로부터 평가되도록 설계할 수 있다. 이것은 최근 보안 시스템 추가가 먼저 실행되고 시스템이 "중간에서 시작"될 수 없다는 것을 의미한하(예를 들면, "복귀 기반" 프로그래밍 익스플로잇에서 사용된 것처럼). 회귀형 시스템의 두번째 장점은 보안 시스템에 대한 임의 갱신의 추가가 보안 시스템 자체에서 최초의 호출 아규멘트를 변경시키지 않는다는 점이다. 따라서, 전통적인 재생 기반 공격 메카니즘을 이용하여 보안 시스템을 눈속임하기가 더 어렵다. 여기에서 설명하는 실시예가 역순서의 반복 형식으로 인라인 실행 스택을 사용하는 것이 가능하지만, 반복 메카니즘은 인터럽트를 받기 쉽고, 따라서 보안 스택의 부분 평가가 수행되는 상황을 생성하는 것이 또한 가능하다. 이것은 하나 이상의 중간 결과가 외부 관측자에 의해 인터셉트되는 것을 잠재적으로 허용한다. 여기에서의 실시예에 의해 활용될 수 있는 것처럼 회귀를 통하여 구현되는 인사이드 아웃 보안 시스템에 있어서, 중간 결과는 임의의 지점에서 다음 레벨 루틴에 아규멘트로서 통과될 수 없고, 현재 실행되고 있는 보안 시스템의 최종 결과만이 보안 시스템 스택의 다음 레벨로 통과할 수 있다.

[0068] 합성키 구조는 특정 실시예에서 목표 유닛의 비밀키에 대한 액세스를 엄격하게 제어함으로써 부분적 평가로부터 또한 보호될 수 있다. 예를 들어서, 비밀키가 구조적으로 가시적이지 않은 액세스불능의 메모리 위치에 저장되어 있으면, 비밀키는 특수한 보안 관련 명령어 또는 기능의 일부로서만 액세스될 수 있다. 특정 실시예에서 이 기능 또는 명령어는 평범하지 않은 일방향 변환과 같이 쉽게 역으로 되지 않는 것이다. 이 방법은 모조 보안 시스템의 경우에도 비밀키의 값을 드러낼 수 없어야 한다. 결국, 비밀키를 일방향 함수의 일부로서 간접적으로 참조되게 함으로써 비밀키는 비밀키가 수학적 연산의 일부로서 그것만으로 절대 사용될 수 없지만 단독으로 또는 어떤 다른 데이터와 함께 해싱 동작의 일부로서만 사용할 수 있을 때 보호될 수 있고, 이때 해싱 함수의 결과만이 관측될 수 있다.

[0069] 비밀키를 보호하는 추가적인 메카니즘은 특정 실시예에서 유용한 것으로 또한 입증할 수 있다. 이러한 한가지 잠재적 메카니즘은 합성키를 사용하는 것이고, 이때 목표 유닛의 비밀키는 어떤 다른 제약 또는 추가적인 오퍼랜드의 집합에 단단하게 결합된다. 이러한 2차 오퍼랜드의 예로는 별도의 비밀키, 총체적 가시성 레지스터(타임스탬프 또는 시스템 버전 번호 등), 비밀키에 액세스하는 코드의 메시지 다이제스트 등이 있다. 이러한 시스템의 실시예에서, 이 최종 예는 비밀키가 동일한 키를 사용하도록 인증된 코드의 세그멘트에 의해서만 액세스되는 것을 보장할 수 있다. 또한, 메시지 다이제스트가 암호화되어 디지털 서명을 형성하는 경우 및 상기 메시지 다이제스트를 암호화하기 위해 사용한 키가 비밀키 자체인 경우, 비밀키에 액세스하는 유일한 방법이 그 비밀키가 무엇이었는지를 이미 알고 있는 누군가에 의해 인증된 코드 세그멘트를 이용하는 것임을 보장할 수 있는 의존성의 사이클(circle of dependencies)이 생성될 수 있다.

[0070] 이 경우에, 합성키 구조의 사용은 목표 유닛의 비밀키 사용이 허용되기 전에 그 비밀키의 사용을 요청하는 코드 부분의 "권한(authority)"을 유효화하는 것을 허용한다. "층을 이룬 키" 구조와 "합성키" 구조 간의 차이는, 특정 실시예에서, 후자가 원자 형태로 평가되고 그 자체는 회귀 수단에 의해 달성될 수 있는 것임을 상기하자. 회귀형 구조와 대조적으로 반복 방법을 이용한 유사한 구조의 어셈블(assemble)이 시도되면, 키 평가 처리의 중간 결과를 노출시킬 위험성이 있다(따라서 비밀키를 잠재적으로 노출시킨다). 이 "노출"은 인터럽트가 발생한 때(또는 메모리 자체에서 직접) 메인 메모리에 밀어내어지는 범용 레지스터와 같은 공개적으로 이용가능한 위치에 비밀키(또는 그 원본)가 저장된 때 발생할 수 있다.

[0071] 특정 실시예에서 취급될 수 있는 잠재적 보안 파괴는 보안 스택 동작이 중간 평가(mid-evaluation)에서 중단된 때 부분 결과의 보호이고, 목표 유닛 프로세서의 상태는 그 다음에 외부 관측자에 의한 시험을 위해 개방되어 있는 메인 메모리에 기록된다. 일 실시예에 있어서, 이 메모리 "노출"을 방지하기 위해, 프로세서가 보안 실행 모드에 있는 동안 힙 푸시(heap push)는 허용되지 않는다. 만일 그 조건이 시행되면, 회귀형 보안 프로토콜은 그 현재 상태를 상실하지 않고 중단될 수 없다(중간 아규멘트가 없기 때문에). 회귀형 보안 프로토콜의 실시예

에 있어서, 전체 보안 프로토콜은 회귀가 종료되고 프로세서가 보안 실행 모드에서 동작할 때 관통되고, 따라서 중단이 아닌 다른 임의의 경우에 임의의 아규멘트를 힙에 밀어내는 것은 더 이상 없다. 따라서, 만일 합성키 평가가 임의의 지점에서 중단되면, 및 힙 푸시가 보안 실행 모드에서 금지되면, 보안 시스템 스택 관통은 중간 실행에서 재시작된다(즉, 계산이 처음부터 재시작하여야 한다). 따라서, 회귀형 보안 프로토콜을 이 방법으로 사용하여 임의의 중간 결과가 시스템 힙에 저장되는 것(및 따라서 인증되지 않은 관측자에게 잠재적으로 노출되는 것)을 방지할 수 있다. 물론, 특정 실시예에서, 반복적인 보안 시스템 평가 중에 힙 동작을 디스에이블하고 그러한 중단된 보안 시스템 동작이 처음부터 재시작되어야 한다는 것을 효과적으로 요구할 수 있다. 그러나, 이러한 반복적인 방법은 "복귀 기반" 프로그래밍 익스플로잇에 대한 보호를 순서정하는 "인사이드 아웃" 실행, 호출 아규멘트를 원래 함수로 변경하지 않는 방식으로 후속 보안층을 추가하는 능력, 및 중간 결과와 최종 함수 출력 결과의 격리와 같이, 회귀형 구조가 제공하는 모든 조건에 대하여 시행할 수 없다. 보안 시스템 회귀가 종료하고 프로세서가 보안 실행 모드로 진입하게 하는 메카니즘에 대해서는 뒤에서 더 자세하게 설명될 것이다.

[0072] 회귀 종료

[0073] 일 실시예에 있어서, 회귀는 주어진 코드 세그멘트의 메시지 다이제스트가 코드 세그멘트 자체와 함께 공급된 것과 일치할 때 종료하도록 신호될 수 있다. 이 방법은 만일 메시지 다이제스트가 하드웨어 해싱 함수에 의해 계산되면 공격에 대하여 더 강하게 될 수 있다. 특정 실시예에서는 디지털 서명이 또한 활용될 수 있다. 디지털 서명 메카니즘은 적어도 2가지의 주요 속성, 즉 1) 코드 세그멘트가 부정하게 변경되지 않았다는 보증 및 2) 코드 세그멘트 창조자의 즉시 식별을 제공하는 잠재성을 구비하고 있다. 그러나, 이러한 디지털 서명이 공개적으로 가시성이거나 수정가능한 위치에 캐시되는 실시예의 경우에는 디지털 서명 자체가 언제든지 수정될 수 있고 따라서 반드시 진정한 것이라고 할 수 없기 때문에 추가적인 보안이 요구될 수 있다. 따라서, 이러한 유형의 실시예에서는 공개키 시스템을 이용하여 디지털 서명을 유효화할 수 있고, 또는 합성키 구조(위에서 설명함)를 이용하여 당해 코드 세그멘트가 제공된 디지털 서명이 목표 유닛의 비밀키를 소유하고 있는 어떤 당사자에 의해 생성되었음을 보증할 수 있다. 후자의 경우에, 합성키의 사용은 단일 엔드포인트에 대하여 또는 일부 엔드포인트 집합에 대하여 또한 제한될 수 있다. 게다가, 공개키 방법과 합성키 방법을 함께 활용할 수 있다. 이러한 방법으로 코드 세그멘트의 진정성 및 코드 세그멘트가 합성 디지털 서명의 수령자용으로 의도된 것을 보증할 수 있다.

[0074] 특정 실시예에 있어서, 목표 유닛에서 보안 메카니즘을 유효화하는 것이 또한 요구될 수 있다. 목표 장치에서 보안 시스템의 임의의 하나의 세그멘트에 대한 하드웨어 생성 디지털 서명이 활용될 수 있지만, 보안 시스템이 회귀형인 경우에, 보안 시스템 자체가 관통될 때 별개의 디지털 서명이 실질적으로 자동으로 생성될 수 있다. 위에서 언급한 바와 같이, 회귀형 보안 프로토콜이 종료되면 전체 호출 사슬이 노출되었다. 따라서, 보안 프로토콜의 최내측(따라서 최근) 부분이 스택에 저장된 호출 아규멘트뿐만 아니라 시스템 힙에(또는 메모리의 어디에든) 저장된 다른 환경 변수를 비롯해서 그것이 호출된 전체 환경에 액세스한다. 이 내장 시스템 인증 메카니즘은 보안 프로토콜 자체의 관통과 동시에 평가되기 때문에 공격에 대하여 특히 효율적일 뿐만 아니라 강력하다.

[0075] 일 실시예에 있어서, 그 다음에, 보안 시스템 스택 관통의 "실행 단계" 전에 적소에 있어야 하는 조건들의 집합이 특정될 수 있다. 일 실시예에 있어서, 이 조건들은 개별적으로 요구되는 모든 보안 조건의 "교집합"이라고 표현할 수 있다. 즉, 새로운 보안 리스크가 발견된 때, 그러한 리스크를 책임지는 추가의 조건들이 쉽게 적소에 놓여질 수 있다. 이 조건들은 신조건 및 구조건의 모든 조건들이 충족될 때까지 실행이 허용되는 보안 시스템 부분은 없다는 것을 확실히 한다. 각종 보안 시스템 조건들의 이러한 "교집합"은 위에서 설명한 것처럼 합성키 구조 메카니즘을 이용함으로써 달성될 수 있다. 예를 들어서, 만일 그러한 합성키 구조의 컴포넌트들 중 하나가 부분적으로 목표 유닛의 비밀키에 기초하고 있으면, 이 비밀키는 전체 보안 시스템의 "신뢰의 근간(Roots-of-Trust)" 중의 하나로서 생각할 수 있다. 더 나아가, 만일 하드웨어 기반 타임스탬프 메카니즘이 합성키의 다른 컴포넌트들 중의 하나로서 활용되면, 시스템은 재전송 공격에 대하여 더 잘 보호될 수 있다. 다른 조건들을 시행하기 위해 특정 실시예에서 사용될 수 있는 다수의 컴포넌트들이 상기의 것 외에도 있다. 이러한 컴포넌트들은 코드가 부정하게 변경된 경우 키가 적절히 평가되는 것을 금지하기 위하여 합성키의 일부로서 코드 블록의 메시지 다이제스트의 하드웨어 기반 해시 계산을 이용하는 것을 포함한다. 일 실시예에 있어서, 다른 하나의 컴포넌트는 실행 대상 코드 블록의 호출 아규멘트의 어떤 선택된 부분집합의 디지털 서명일 수 있고, 이것은 스택 오버플로우 스타일 공격에 대하여 보호할 수 있다.

[0076] 코드 세그멘트가 타임스탬프 또는 사용량 관련 제한과 같은 그 실행상의 다른 제약을 갖는 경우, 특정 실시예에서, 추가의 조건들이 합성 디지털 서명에 추가되어 상기 제약들이 또한 적절히 시행되는 것을 보장할 수 있다.

이러한 동일한 메카니즘을 이용해서, 중간 보안 토큰의 정확한 평가에 기초하여 시스템의 각 층 내에서의 적당한 코드 분기를 시행함으로써 각종 보안 스택 층을 통한 특정의 복수의 반복을 강제할 수 있다.

[0077] 위에서 설명한 것처럼, 회귀형 보안 시스템은 모든 조건들이 보안 토큰의 평가를 시작하기 전에 적소에 있는 것을 보증하도록 요구되는 특정 실시예에서 유리하다. 인사이드 아웃 보안 스택 관통의 시행 능력 및 중간 결과의 가시도(visibility)에 대한 제한이 있는 회귀형 시스템은 따라서 최소 방해 형식으로 보안 시스템 평가에 대한 더 많은 제약을 추가하도록 요구된 때 외부 공격에 대한 향상된 강력함 및 융통성을 제공할 수 있다.

[0078] 여기에서, 보안 시스템 스택의 회귀형 관통은 모든 연산 흐름에 대한 회귀 형태와 반드시 동일할 필요는 없다는 것에 주목하여야 한다. 따라서, 보안 시스템의 논리 흐름 및 시스템 보안 시스템의 사용을 가능하게 하는 코드 스레드의 논리 흐름은 완전히 별개이다.

[0079] 게다가, 특정 실시예에서, 특수 코드 세그멘트가 분석됨에 따라 디지털 서명이 어떻게 수정되는지를 특정하는 함수들의 집합을 포함함으로써, 디지털 서명이 어떻게 사용되는지에 관한 융통성이 증가될 수 있다. 예를 들면, 코드 세그멘트가 최초의 반복 후에 변경되지 않은 분석 처리를 통하여 디지털 서명을 통과하도록 허용되는 경우, 그 코드 세그멘트는 보안 시스템이 그 특수 코드 블록을 통하여 몇 번이나 순환하는지를 먼저 특정할 필요없이 유효화될 수 있다. 유사하게, 특수 코드 세그멘트가 조우될 때 디지털 서명이 공지의 "종자" 상태로 리세트되는 것을 특정할 수 있다. 따라서, 단일의 유일한 번호(이것은 명문으로 저장될 수 있다)를 단순히 공급함으로써, 보안 시스템의 각종 부분이 몇 번 및 어떤 순서로 관통되는지에 관한 유일한 변형체가 특정될 수 있다. 사실, 이러한 코드 유효화 처리를 이용하여 각종 유용한 기능을 구현할 수 있고, 따라서 이 기술은 보안 시스템 자체의 배타적 사용에 대하여 반드시 제한할 필요가 없다.

[0080] 적당한 디지털 서명이 일반 코드(generic code)(보안 구현과 관련되거나 관련되지 않은 코드)와 함께 공급되는 경우에, 특수 코드 블록이 특정의 목표 유닛에서 실행하는 방법이 매우 특별하게 제어될 수 있다. 이것은 일반 코드를 목표 장치의 집합에 안전하게 분배하기 위해 사용될 수 있는 매우 강력한 메카니즘이다. 이 분배 방법은 예를 들면 애플리케이션에 대한 무료 또는 유료 업그레이드에 대하여 또는 소프트웨어 바이러스 및 다른 바람직하지 않은 멀웨어의 확산을 관리하기 위해 효과적으로 적용될 수 있다. 상기 후자의 실시예에서, 회귀형 보안 시스템을 이용하여 목표 장치에서의 실행 후보인 각각의 모든 코드 블록을 유효화할 수 있다. 따라서, 멀웨어 애플리케이션 또는 미리 인증되어 있는 코드 세그멘트에 대한 다형태 바이러스성 공격조차도 실행이 금지된다.

[0081] 하드웨어 의존성을 보안 시스템 평가에 통합하는 능력을 제공하기 위해, 특정 실시예에서, 하드웨어 구현 버전 번호가 디지털 서명 평가의 합성 컴포넌트 중의 하나로서 활용될 수 있다. 만일 하드웨어 버전 번호가 보안 시스템이 수정될 때마다 갱신되면(및 그 갱신이 안전하면), 보안 시스템은 보안 시스템이 실행하는 목표 유닛에 정합되는 것으로 보장될 수 있다. 이것은 위에서 설명한 타임스탬핑 메카니즘과 다른 것이지만, 이 둘을 합성키 평가에서 함께 사용하여 재전송 공격 시나리오 또는 다른 위배에 대하여 보호할 수 있다는 점에 주목한다.

[0082] 예를 들어서 합성키 구조의 일부로서 JTAG 서명과 같은 하드웨어 유도형 체크섬을 사용하면, 하드웨어 구현 자체가 인증될 수 있다. 이런 종류의 메카니즘은 소프트웨어 및 하드웨어가 정합 쌍(matched pair)이고 하드웨어가 믿을만한 것(즉, 부정하게 변경되지 않았다는 것)임을 보증할 수 있다. 더 나아가, 합성키 평가의 일부로서 사용되는 JTAG 서명이 직접 관찰될 수 없으면(예를 들어서, 그 상태가 외부적으로도 보여질 수 없고 구조적으로도 보여질 수 없는 스캔 사슬의 한 지점으로부터 취해진 것이면), 하드웨어 복제에 기초하여 잠재적 공격을 탑재시키는 어려움은 수 배 증가될 수 있다. 이 전략은, 예를 들어서 장치의 개별 일련 번호가 상기 스캔 사슬에 포함되어 있으면, 효과적으로 될 수 있다.

[0083] 여기에서, 프로세서의 시각에서, 본질적으로, 암호화 코드 블록(직접 실행될 수 없는 것)과 "구식" 코드 블록 간의 논리적 차이는 없을 수 있고, 이것은 아마도 한번에 실행되어 정확한 디지털 서명 정합을 제공할 수 있지만, 그 디지털 서명이 더 이상 유효하지 않기 때문에 더 이상 실행될 수 없다는 점에 주목하여야 한다. 이 시나리오는 예컨대 목표 장치의 타임스탬프 레지스터가 변경되었기 때문에, 또는 대안적으로 목표 장치의 하드웨어가 어떤 인증되지 않은 방식으로 수정된 경우에 발생할 수 있다.

[0084] 따라서, 특수 코드 블록이 갱신 버전으로 교체된 경우에(비록 둘 다 잠재적으로 실행가능이라 하더라도), 일 실시예에 있어서, 이것을 달성하는 간단하지만 효과적인 방법은 당해 코드 블록의 복호 키 포인터를 구 버전의 코드 블록을 갱신 버전으로 교체하는 수단을 유도하는 새로운 포인터로 먼저 교체하고 그 다음에 목표 엔드포인트 장치의 타임스탬프 레지스터를 갱신하는 것일 수 있다. 여기에서, 갱신된 타임스탬프 레지스터 값은 오래된 값을 이용하여 생성된 이전의 모든 디지털 서명을 무효화하고, 그에 따라서 보안 시스템을 최신식으로 하기 위해

서 및 오래된 디지털 서명(또는 키)을 새로운 키/디지털 서명 값 및 갱신된 기능으로 교체하기 위해 보안 시스템 전체를 개조(이상적으로는 안전한 방법으로)하는 것을 수반할 수 있다. 이것은 엔드포인트 장치의 타임스탬프 레지스터에 저장된 값에 대하여 단일 변경으로 쉽게 영향을 줄 수 있는 매우 강력한(및 잠재적으로 매우 중요한) 메카니즘이다. 이 경우에, 엔드포인트 타임스탬프 레지스터 값은 안전하지 않거나 무모한 방식으로 변경되지 않도록 주의해야 한다. 이러한 강제 갱신 시나리오의 일 실시예는 (단순히 단일 디지털 서명을 불일치하게 함으로써) 다른 직접 실행가능한 코드 블록에 암호화 층을 추가하는 것과 논리적 등가물로서 인용될 수 있다.

[0085] 보안 모드 및 보안 코드 결합

[0086] 시스템이 전술한 구조적으로 비가시성인 비밀키 중의 하나를 활용하는 실시예에 있어서, 그러한 키를 사용하게 하는 코드는 비밀키가 절충되는 것을 금지하는 방식으로 설계되어야 한다. 위에서 언급한 바와 같이, 특수 엔드포인트 장치의 다른 적법한 코드 블록이 인증되지 않은 방식으로 사용되는 경우 그 코드 블록의 정확한 실행을 금지하는 보안 코드 결합 메카니즘을 사용할 수 있다.

[0087] 일 실시예에 있어서, 이 보안 코드 결합은 특수 종류의 해싱 함수를 후보 코드 세그멘트에 적용한 결과가 그 코드 세그멘트의 실행이 허용되기 전에 그 코드 세그멘트가 제공된 특수하게 미리 결정된 메시지 다이제스트와 정합하는 필요조건에 기초를 둔다. 이 해싱 함수는 후보 코드 세그멘트가 호출된 후이지만 실행이 허용되기 전에 적용될 수 있다. 이 해싱 함수가 개시되면 후보 코드 세그멘트를 포함하는 특수 메모리 스페이스에 대한 임의의 기록이 디스에이블 또는 무시될 수 있다. 후보 코드 세그멘트가 CPU의 명령어 캐시에서와 같이 CPU 실행 유닛과 동일한 칩에 위치되어 있으면, 이것은 쉽게 구현될 수 있다. 그러나, 예를 들어서 I-캐시가 동일 칩상에 있는 하나 이상의 프로세서 간에 공유되는 멀티프로세서 시스템에서, 이것은 표면상으로 보이는 것처럼 구현하기가 간단하지 않을 수 있다. 메시지 다이제스트가 계산된 후에 코드가 수정되는 것을 금지하는 다른 잠재적인 방법은 해싱 함수가 설치된 후에 발생하는 메모리 스페이스에 대한 임의의 시도된 기록이 프로세서 인터럽트를 야기하도록 시스템을 구성하는 것이다. 위에서 설명한 것처럼, 이것은 프로세서의 보안 실행 모드를 그 디폴트 초기 "보안 아님" 모드로 리세트할 수 있다. 이러한 명령어에 대한 다른 응답은 보안 실행 스레드가 예컨대 메모리 세그멘트화 오류를 개시함으로써 에러를 가지고 종료되게 한다.

[0088] 후보 코드 세그멘트의 계산된 메시지 다이제스트가 후보 코드 세그멘트와 함께 수신된 미리 결정된 메시지 다이제스트와 일치하면, 후보 코드 세그멘트는 "보안 모드" 또는 "보안 실행 모드"라고 명명된 곳에서 실행이 허용된다. 위에서 설명한 것처럼, 보안 모드에서 동작하는 코드만이 구조적 비가시성 비밀키를 활용할 수 있다. 특수 코드 세그멘트가 보안 모드에서 동작하지 않으면, 비밀키는 디스에이블되고, 그들 중 임의의 하나에 대한 임의의 참조는 어떤 다른 값(예를 들면, 제로)으로 복귀할 것이다.

[0089] 특정 실시예에 있어서, 후보 코드 세그멘트의 메시지 다이제스트를 계산할 때 사용한 해싱 함수는 소정의 특성을 가질 수 있다. 제1 특성은 해싱 함수가 목표 유닛의 하드웨어에서 구현되는 것이다. 이것은 이 함수가 최초 하드웨어 해싱 함수의 어떤 다른(아마도 파괴된) 버전으로 완전히 교체될 수 없다는 것을 의미한다. 이 해싱 함수는 필요할 때 최초 함수의 정제된 버전(또는 조건부 완전 교체)에 의해 보충될 수 있다는 점에 주목하여야 한다. 일 실시예에 있어서, 하드웨어 해싱 함수를 정제된 버전으로 교체하는 방법은 보안 시스템 구조의 회귀적 정의를 통하여 보안 시스템에 새로운 층을 삽입하기 위해 사용되는 위에서 설명한 절차와 실질적으로 유사하다. 그러나, 이 경우에, 비록 새로운 해싱 함수가 모든 후속하는 보안 시스템 동작의 용도로 최초 함수를 교체할 수 있다 하더라도, 이 새로운 해싱 함수 자체는 그 신뢰의 근간의 토대로서 최초의 하드웨어 해싱 함수에 여전히 의존할 수 있다는 점에 주목하여야 한다. 따라서, "조건부 완전 교체"의 용어를 사용한다. 일 실시예에 있어서, 최초의 하드웨어 기반인 신뢰의 근간은 일정하게 유지될 수 있다. 이것은 하드웨어 기반 보안 시스템을 손상시키기가 매우 어렵다는 점에서 바람직하다. 그러나, 최초 하드웨어 해싱 함수의 단점이 목표 장치가 현장에서 전개된 후에 발견되면, 그러한 단점은 호출 아규멘트를 효과적으로 제한할 수 있는 단일 애플리케이션에서 최초 해싱 함수를 이용함으로써 잠재적으로 억제될 수 있다.

[0090] 하드웨어 해싱 함수의 제2 특성은 해싱 함수가 그 종자 값으로서 구조적으로 비가시성인 비밀키를 이용할 수 있다는 것이다. 따라서, 동일한 입력 아규멘트가 주어진다 하더라도, 그러한 하드웨어 해싱 함수의 메시지 다이제스트 결과는 유닛마다 다를 수 있다. 이 차이는 모든 목표 유닛 각각에 대해 유일한 메시지 다이제스트를 야기할 수 있다는 점에서 이용될 수 있다. 이 특성은 디지털 서명과 개념상으로 유사하지만, 하드웨어 해싱 함수에 대한 별도의 암호화/복호 블록의 추가를 반드시 요구하는 것은 아니다. 이때, 후보 코드 세그멘트는 하드웨어 생성 메시지 다이제스트가 후보 코드 세그멘트와 함께 분배된 것과 일치하는 유닛에서만 실행하도록(적어도 보안 모드에서) 구속되기 때문에, 순환 의존성이 생성되었다. 이 순환 의존성은 그 메시지 다이제스트가 올바른

목표 유닛의 비밀키로 생성된 코드만이 실제로 동일한 비밀키를 사용할 수 있게 한다는 의미이다. 이 특성은 목표 장치에서 보안 모드로 실행이 허용된 코드 세그멘트를 예비 공격자가 생성하는 능력을 실질적으로 손상시킨다.

[0091] 전술한 메카니즘은 코드 세그멘트가 특수한 목표 장치에(또는 특수한 엔드포인트 장치의 집합에) "결합"될 수 있기 때문에 "보안 코드 결합"이라고 부른다. 위에서 언급한 바와 같이, 보안 코드의 실행 블록이 인터럽트된 경우에, 콘텍스트는 세이브되지 않고, 따라서 이 코드 세그멘트의 실행은 포기되거나 처음부터 재시작되어야 한다. 또한, 코드 세그멘트의 보안 모드에서의 실행이 인터럽트되면 프로세서는 더 이상 보안 모드에서 동작할 수 없고, 구조적으로 비가시성인 비밀키에 대한 임의의 액세스는 프로세서가 보안 모드로 복귀할 때까지 차단된다. 특정 실시예에서, 임의의 오프칩(off-chip) 저장 동작은 프로세서가 보안 모드에서 동작하는 동안 통제되거나 금지된다.

[0092] 전술한 바와 같이, 특정 실시예에서, 각 목표 유닛은 구조적 비가시성 비밀키의 유일한 집합을 가질 수 있다. 그러나, 다른 실시예에서, 이러한 키의 일부 부분집합은 다수의 동일한 장치에 공통일 수 있다. 따라서, 특수한 코드 세그멘트는 키들의 공통 부분집합을 가진 특수한 엔드포인트 장치 부류에 결합될 수 있고, 그러한 장치들 간에 공통으로 유지되는 구조적 비가시성 비밀키의 상기 집합을 여전히 보호한다. 따라서, 하드웨어 해싱 함수와 하나 이상의 구조적 비가시성 비밀키의 조합은 고도로 효과적이고 강한 회귀형 보안 프로토콜에 신뢰 사슬의 기초를 제공한다.

[0093] 이제, 각종 실시예의 구현 세부에 대하여 첨부 도면을 참조하면서 설명한다. 모든 예에 있어서 용어 "디지털 비트스트림"은 디지털 데이터의 포괄적 집합을 말하고, 따라서 이 용어는 디지털 콘텐트, 코드 블록 또는 디지털 데이터 집합과 상호교환적으로 사용될 수 있다. 코드 블록 용어의 경우에, 인용된 데이터는 실행가능 파일, 실행가능 스크립트 또는 의사코드의 알고리즘 설명 또는 블록을 나타내는 것으로 또한 추정할 수 있다.

[0094] 도 3은 디지털 콘텐트의 합성키의 생성에 대한 일 실시예를 보인 도이다. 이 합성키(310)는 전술한 바와 같이 특수 엔드포인트 장치(목표 유닛)와 관련된 구조적 비가시성 비밀키일 수 있는 엔드포인트 지정 하드웨어 키(340)를 활용하는 글로벌 콘텐트 키(330)(이것은 디지털 콘텐트의 소유자 또는 창조자에 의해 제공되거나 결정될 수 있음)에 암호화 엔진(320)을 적용함으로써 생성될 수 있다. 특수 엔드포인트 및 디지털 콘텐트 둘 다에 특수한 결과적인 합성키는 합성키가 제공된 엔드포인트 장치에 전송되어 저장되고, 명문으로 저장된다.

[0095] 도 4a는 보안된 디지털 데이터 블록 구조의 생성에 대한 일 실시예를 보인 도이다. 이 실시예에서, 디지털 데이터 블록(410)은 암호화되지 않을 수 있지만, 디지털 서명(420)은 하나 이상의 토큰(440 또는 450)을 가진 디지털 데이터 블록으로부터 해싱 함수(430)에 의해 계산된 메시지 다이제스트를 암호화함으로써 형성된다. 상기 토큰은 비밀키 또는 타임스탬프와 같이 공개적으로 이용가능한 데이터일 수 있다. 암호화 엔진(460, 461)을 통과한 데이터를 암호화하기 위해 사용하는 방법은 동일할 수도 있고 동일하지 않을 수도 있다. 비밀키를 암호화 키 중의 하나로서 사용하는 경우에는 그 비밀키의 값을 모른 채 디지털 서명을 위조하는 것이 더 어렵다. 또한, 암호화 동작(460, 461)의 순서는 결과의 전체 보안성에 관련이 없지만, 결과적인 디지털 서명(420)은 동작의 순서가 변경되면 달라진다는 것을 아는 것도 교훈적이다.

[0096] 도 4b는 보안 코드 블록 데이터 구조의 생성에 대한 다른 실시예를 보인 도이다. 이 경우에, 비밀키(470)는 디지털 데이터 블록(471)에 첨부되어 전체 메시지(480)를 형성한다. 앞에서처럼, 상기 첨부 동작이 비밀키(470)를 최초 디지털 데이터 집합(471)의 앞에 두는지 또는 뒤에 두는지는 결과적인 보안성의 강함에 반드시 관련이 있는 것은 아니지만, 최종 결과는 만일 순서가 변경되면 달라질 것이다. 보안성을 보장하기 위해, 비밀키(470)는 최초 디지털 데이터 집합(471)과 함께 공개되어서는 안된다는 점에 또한 주목한다. 그러므로, 공개된 데이터 집합은 전체 데이터 구조(480)보다는 디지털 데이터 집합(471)으로 한정될 것이다. 그 다음에, 상기 전체 데이터 구조(480)는 도 4a와 관련하여 위에서 설명한 것과 본질적으로 동일한 방법으로 해싱 함수를 통과한다. 그러나, 이 실시예에 있어서, 최종 출력(490)은 도 4a에 도시한 디지털 서명(420)의 특성들 중 많은 것을 갖지만, 암호화 엔진(460 또는 461)의 사용을 요구하지 않을 수 있다. 따라서, 이 동작의 결과(490)는 디지털 서명 등가물이라고 인용될 것이다. 이 디지털 서명 등가물(490)은 각각의 유일한 전체 데이터 구조(480)에 대하여 유일하다(해싱 함수(430)가 적절히 구성되었다는 가정하에)는 점에 주목해야 한다. 따라서, 만일 비밀키(470)가 디지털 데이터 집합(471)의 창조자 및 그 디지털 데이터 집합의 소비자(엔드포인트 장치 또는 목표 장치)에 의해서만 공유되면, 이들 두 당사자만이 동일한 올바른 디지털 서명 등가물(490)을 재생성할 수 있을 것이다. 이 경우에, 디지털 데이터 블록(471)은 그 비밀키(470)에(및 그에 따라서 목표 장치에) 결합된 것으로 생각할 수 있다.

[0097] 도 5a는 암호화 데이터 블록(510)을 특정의 복호 엔진 코드 블록(562)에 암호적으로 결합하고 그 다음에 그 조

합(530)을 해싱 함수(540) 및 암호화 엔진(561)에 의해 계산된 디지털 서명(524)을 이용하여 특정 엔드포인트의 하드웨어 비밀키(523)에 결합하기 위해, 여기에서 설명한 바와 같은 보안 시스템이 어떻게 이용될 수 있는지에 대한 일 실시예를 보인 도이다. 이 예에서, 공개키(522)(글로벌 콘텐트 비밀키(520)로 복호 엔진 코드 블록(562)의 메시지 다이제스트(521)를 암호화함으로써 구성된 것)는 최초의 암호화 데이터 블록(510)과 함께 단일 연쇄 데이터 집합(530)으로서 공개적으로 분배된다. 결합된 메시지(530)(공개키(522)와 결합된 최초 암호화 데이터 블록(510)을 포함함)의 메시지 다이제스트로부터 디지털 서명(524)을 생성하는 동작은 적절하게 인증된 엔드포인트 장치만이 최초 암호화 데이터 블록(510)을 복호할 수 있게 하고, 이 복호 처리는 복호 엔진(562)의 미리 규정된 방법을 이용하여 달성될 수 있을 뿐이다. 암호화 엔진 사슬(560)(예를 들면, 멀티텀(multi-term) 합성 암호화 등)에 더 많은 컴포넌트를 추가함으로써 더 많은 제약이 복호 인증 절차에 쉽게 추가될 수 있다는 점에 주목한다.

[0098] 도 5b는 도 5a에 도시한 실시예의 변형예를 보인 도이다. 이 실시예에서, 특수한 암호화 메시지(511)의 창조자는 명백하게 인증될 수 있지만 특정의 엔드포인트 장치에서만 그렇다. 여기에서, 최초 암호화 데이터 블록(511)은 위에서 설명한 것처럼 특정 복호 루틴(562)에 암호적으로 결합된다. 이 점에서, 복호 루틴(562)은 비대칭 암호화 엔진이고, 입력은 창조자의 비밀 개인키(525)이며, 출력은 창조자 개인키를 이용하는 경우에만 정확하게 복호된다는 것을 추가로 특정할 수 있다. 비대칭 암호화 루틴(562)의 메시지 다이제스트(527)는 디지털 서명(526)과 함께 최초 암호화 디지털 데이터(511)에 첨부되어 전체 데이터 구조(531)를 형성한다. 데이터 구조(531)는 그 다음에 엔드포인트 장치의 비밀키(523), 해싱 함수(544) 및 암호화 엔진(561)을 이용하여 특정 엔드포인트 장치에 암호적으로 결합되어 디지털 구조(528)를 형성할 수 있다. 이 실시예에 의해, 암호화 메시지(511)가 진정한 것임이 보증될 수 있고, 그 창조자는 창조자가 하드웨어 비밀키(523)를 소유하고 있다는 사실을 알게 된다. 여기에서 주목할 것은 여기에서 사용하는 용어 창조자(author)는 반드시 데이터의 창시자(originator)를 의미하는 것이 아니고, 허가자, 분배자, 또는 이러한 데이터를 분배하거나 다른 방식으로 통신하기 원하는 다른 유형의 엔티티를 또한 인용할 수 있다. 이러한 특수한 신뢰 사슬이 중요한 용도로 될 수 있는 일 예는 엔드포인트 장치의 보안 시스템이 보안 코드 블록(최초 데이터 블록(511)에 암호화 형태로 내포되어 있는 것)을 이용하여 갱신되어야 하는 경우이다.

[0099] 도 6은 보안 코드 블록(620)의 실행을 제어하기 위해 직렬 해시 방법을 활용하는 일 실시예를 보인 도이다. 이 경우에는 2개의 독립 코드 블록(610, 620)이 있다. 이 예에서, 제1 코드 블록(보안 코드 블록(610))은 제2 코드 블록(보안 코드 블록(620))에 대한 매립(embedded) 서브루틴 호출을 포함한다. 따라서, 보안 코드 블록(610)에 대하여 해싱 함수(640)에 의해 계산된 메시지 다이제스트(630)는 보안 코드 블록(610) 내에 포함된 보안 코드 블록(620)에 대한 참조에 의존한다. 그 다음에, 이 메시지 다이제스트(630)는 보안 코드 블록(610)의 시각으로부터 2개의 보안 코드 블록을 함께 연결한다. 다음에, 메시지 다이제스트(650)는 해싱 함수(670)를 이용하여 코드 블록(620)에 대하여 구성될 수 있다. 그러나, 메시지 다이제스트(650)를 보안 코드 블록(620)뿐만 아니라 그 호출하는 부모 루틴(이 경우에는 보안 코드 블록(610)) 둘 다에 결합하기 위하여, 최초 메시지 다이제스트(630)는 해싱 함수(670)에 의해 계산된 메시지 다이제스트(650)에 대한 종자로서 사용될 수 있다. 이러한 종자 값은 여러 가지 방법으로 구현될 수 있고, 그러한 방법 중의 하나는 최초의 메시지 다이제스트(630)를 제2 디지털 데이터 집합(예를 들면, 이 경우에 보안 코드 블록(620))에 단순히 연결하여 전체 메시지(660)를 형성하는 것임을 상기하자. 그 다음에, 전체 메시지(660)는 해싱 함수(670)(이것은 해싱 함수(640)와 동일한 것일 수도 있고 또는 어떤 다른 독립 해싱 함수일 수도 있다)를 통과하여 제2 메시지 다이제스트(650)를 형성하고, 이것은 따라서 보안 코드 블록(620)뿐만 아니라 최초 메시지 다이제스트(630)(이것 자체는 2개의 보안 코드 블록(610, 620)에 의존하는 것임) 둘 다에 의존한다. 도 4b와 관련하여 위에서 설명한 바와 같이, 상기 연결된 요소들(620, 630)의 순서는 결과적인 메시지 다이제스트(650)에 중요할 수 있지만, 해싱 함수(670)의 경우에는 전체 메시지(660)를 구성하는 요소들의 순서가 해싱 함수(670)의 보안에 실질적으로 영향을 주지 않는다.

[0100] 상기 제2 메시지 다이제스트(650)는 그 다음에 위에서 설명한 것과 실질적으로 유사한 방법으로 사용되어 보안 코드 블록(620)이 보안 코드 블록(610)으로부터 호출된 경우에만 정확하게 실행되는 것을 보증할 수 있다. 코드 블록(620)은 실제로 코드 블록(610)의 정확한 복제품(또는 등가 참조)일 수 있고, 이것이 회귀 시스템의 실시예를 구성한다. 동일한 코드 블록의 2개의 사례(instantiation) 간의 유일한 차이점은 보안 코드 블록 메시지 다이제스트를 형성하기 위해 코드 블록에 첨부되는 특수한 메시지 다이제스트일 것이다.

[0101] 이 특수한 실시예에 있어서, 우리는 임의의 비밀키를 사용하지 않았고, 그래서 이런 종류의 구조는 여기에서 설명하는 동일한 전체 보안 시스템을 이용하는 임의의 엔드포인트 장치에서 적당한 실행 순서를 시행하는 특이성 없이 사용될 수 있다는 점에 주목한다. 또한, 앞에서와 같이, 유사한 예를 구성할 수 있고, 그 경우 보안 코드

블록 중의 어느 하나(610 또는 620)의 실행은 메시지 다이제스트(630 또는 650) 대신에 합성키 기반 디지털 서명 구조 또는 그 등가물을 각각 활용함으로써 특정 엔드포인트 장치 또는 장치들의 집합에 추가로 구속된다.

[0102] 도 7a는 보안 코드 블록 메시지의 구성에 관한 실시예를 보인 도이다. 일 실시예에 있어서, 암호화 디지털 데이터 집합(711)은 포인터(720)에 의해 식별된 암호화 알고리즘을 이용하여 암호화되었다. 데이터 구조(730)는 디지털 데이터 집합(711)과 포인터(720)의 연결에 의해 형성된다. 데이터 구조(730)의 메시지 다이제스트(750)는 해싱 함수(740)에 의해 생성된다. 이 구성은 암호화 데이터 집합과 그 관련 복호 루틴의 암호적 결합을 가능하게 한다.

[0103] 제2 실시예에 있어서, 추가의 텀이 연쇄 데이터 구조(731)에 추가된다. 즉 포인터(721)가 복호키(760)에 추가된다. 이 키(760)는 반드시 이 특수 실시예에서 묘사하는 하드웨어 기반 비밀키일 필요는 없다는 점에 주목하여야 한다. 사실, 포인터(721)에 의해 지적된 키(760)는 뒤에서 도 7c와 관련하여 설명하는 바와 같이 데이터 구조 자체일 수 있다. 그렇지 않으면, 이 실시예는 위에서 설명한 실시예와 실질적으로 유사하다. 암호화 디지털 데이터 집합(711)은 최초의 비암호화 데이터 집합(710)에 대하여 동작하는 암호화 엔진(770) 및 하나 이상의 키(760)를 사용한 결과로서 생성된다. 메시지 다이제스트(751)는 연쇄 데이터 구조(731)에서 해싱 함수(741)를 이용하여 생성된다. 이 경우에, 이제 암호화 데이터 집합(711)으로부터 비암호화 데이터 집합(710)을 재생성하기 위해 사용될 수 있는 유일키(760) 및 암호화 엔진(770) 둘 다에 비암호화 데이터 집합(710)을 암호적으로 연관시키는 메카니즘이 있다. 위에서의 실시예와 같이, 추가의 텀은 주어진 엔드포인트 장치 및 그 유일한 하드웨어 비밀키(760)에서 만족되어야 하는 특수한 조건들의 집합에 전체 구조를 암호적으로 결합하기 위해 암호화 사슬에 추가될 수 있다. 디지털 데이터 집합(710, 711)의 포맷 및 암호화 상태(즉, 암호화되었는지 아닌지)는 그 세부가 포인터(720, 721)로부터 추론될 수 있기 때문에 이 처리에 관련되지 않을 수 있다는 점에 주목한다.

[0104] 이것을 염두에 두고, 도 7b는 회귀형 보안 시스템에서 사용될 수 있는 범용 암호 데이터 구조의 기본 일반화 포맷의 한가지 가능한 실시예를 도시한 것이다. 이 구조의 실시예는 단순하고 강력하며, 3개의 기본 요소, 즉 일반 데이터 블록(712), 복호 포인터(720) 및 복호 키 포인터(721)의 단순한 연결 리스트로서 구현될 수 있다. 전체적인 연결 리스트는 데이터 구조(732)에 함께 묶여진다. 연결 리스트를 사용함으로써 연쇄 데이터 구조(732) 내의 요소들의 순서화는, 비록 데이터 구조의 동작 또는 평가에 영향을 줄 수 있다 하더라도, 그 기능에 관계가 없다는 것을 쉽게 알 수 있다. 일반(예를 들면, 미리 규정되지 않은) 데이터 블록 구조 및 연결 리스트 포맷을 이용하는 다른 중요한 태양은 3개의 요소(712, 720, 721)가 반드시 선형으로 또는 연속적으로 순서정해질 필요가 없다는 것이다. 따라서, 일 실시예는 전체적 데이터 구조(732)와 함께 저장되는 어떤 다른 독립적이지만 아마도 관계가 있는 데이터를 포함한 보조 데이터 구조(713)를 포함할 수 있다. 이 개념의 일 실시예는 도 7의 보조 데이터 구조(713) 내측에 있는 포인터(720)에 의해 지적된 것과 같이, 실제 복호 엔진 코드 블록(771)을 위치시키는 것이다. 다른 예는 이 보조 데이터 블록 내측에 있는 포인터(721)에 의해 특정된 실제 키 값을 저장하는 것일 수 있다.

[0105] 상기 2가지 경우에, 보조 데이터 블록에 포함된 실제 데이터는 도 4a, 도 4b, 도 5a, 도 5b, 도 6 및 도 7a의 실시예에서 다양하게 묘사된 메시지 다이제스트 또는 디지털 서명의 생성 과정에서 사용될 수 있다. 이 명세서 설명에서 주지된 바와 같이, 연쇄 데이터 집합에 저장된 각종 데이터 필드는 해싱 함수를 적절히 설계하면 결과적인 메시지 다이제스트(또는 디지털 서명)에 영향을 줄 수 있다.

[0106] 특정 실시예에서 활용되는 키를 보호하기 위해 유사한 블록 구조가 또한 사용될 수 있다는 것이 명백하다. 도 7c는 키들만을 포함하는 보안 코드 블록(733)의 일 실시예를 도시한 것이다. 여기에서, 데이터 블록은 장치 지정 키(714, 715, 716)(또는 필요하다면 다른 것)의 리스트를 포함할 수 있다. 이 예에서, 이들 키 중 임의의 키는 암호화 엔진(771, 772)을 이용하여 각각 암호화된 엔드포인트 장치(760) 및 엔드포인트 장치 타임스탬프 레지스터(790)의 비밀키(예를 들면)를 이용하여 생성될 수 있다. 보안 시스템의 강력함의 시각에서 이러한 동작에 대해 위에서 설명한 경우처럼, 암호화 엔진(771, 772)이 별개의 것이거나 달라야 한다는 필요조건은 없고, 암호화 사슬에서 이러한 암호화 동작의 특정 수에 대한 기본적인 제한이 없으며, 이러한 동작의 순서가 결과적인 합성키에 문제가 될 수 있지만 암호화 동작에 대한 특정 순서에 대한 필요조건은 없다. 이 경우에 주지되는 다른 하나의 특징은 연쇄 키 리스트 데이터 구조(733)의 키 리스트 포인터 요소(721)가 또다른 연쇄 키 리스트 데이터 구조(734)를 지적할 수 있다는 것이다. 이들 데이터 구조는 둘 다 도 7b에 도시한 것과 동일한 범용 암호 형식을 갖기 때문에, 키 리스트 데이터 구조는 회귀적 방법으로 형성될 수 있다. 따라서, 이러한 회귀형 보안 시스템의 실시예에서 사용하는 키(733)는 보안 프로토콜의 실시예가 적용될 수 있는 임의의 다른 데이터와 동일한 구조를 이용하여 동일한 방법으로 보호될 수 있으며, 마찬가지로, 이러한 보호된 키는 여기에서 설명하는 시스템 및 방법의 실시예에 의해 보호된 다른 데이터와 같은 방법으로 엔드포인트 장치에서 또한 복호 및 인증될 수

있다.

[0107] 이제 도 8을 참조하면, 암호화 콘텐트를 복호하기 위해 합성키가 어떻게 활용될 수 있는지에 대한 일 실시예가 도시되어 있다. 이 복호 동작은 위에서 설명한 것처럼 "보안 모드"에서 발생할 수 있다. 여기에서, 콘텐트(810)는 합성키(830)와 함께 엔드포인트 장치에 제공되고, 이때 콘텐트는 초기에 글로벌 콘텐트 키를 이용하여 암호화된 것이다. 합성키(830)는 도 3을 참조하여 위에서 설명한 것처럼 생성될 수 있다. 따라서, 암호화 콘텐트(810)가 엔드포인트 장치에 수신된 때, 암호화 콘텐트는 관련된 합성키(830)와 함께 수신될 수 있다. 장치의 비밀키(840)가 액세스될 수 있도록 보안 모드에서 실행하면, 합성키(830)는 보안 코드 블록(860)의 내측에서 복호되어 글로벌 콘텐트 키를 산출할 수 있다. 글로벌 콘텐트 키는 그 다음에 최초 암호화 콘텐트(810)를 복호하도록 보안 코드 블록(860) 내에서 사용되어 복호 콘텐트(880)를 산출할 수 있다.

[0108] 도 9는 코드 블록이 실행 전에 특수 엔드포인트 장치에서 동작하도록 인증되는 것을 검증하기 위해 비밀키가 어떻게 활용될 수 있는지에 대한 일 실시예를 보인 도이다. 실행을 위한 후보 코드 블록(910)이 엔드포인트 장치에 제공될 수 있고, 또는 수신된(예를 들면, 도 8을 참조하여 위에서 설명한 것처럼) 암호화 디지털 콘텐트를 복호함으로써 얻어질 수 있다. 게다가, 엔드포인트 장치는 후보 코드 블록(910)에 대응하는 대응 디지털 서명(920)을 수신할 수 있다. 이 디지털 서명(920)은 엔드포인트 장치 하드웨어 지정 비밀키(930)를 이용하여 암호화된 코드 블록으로부터 (예를 들면, 이 코드 블록을 해싱함으로써) 생성된 메시지 다이제스트를 포함할 수 있다. 따라서, 후보 코드 블록(910)이 실행될 수 있는지를 검증하기 위해, 인증 동작이 보안 모드에서 구현되고, 이것에 의해 후보 코드 블록(910)이 해시되어 메시지 다이제스트(912)를 생성한다. 이 메시지 다이제스트(912)는 그 다음에 엔드포인트 장치(검증이 보안 모드에서 이루어졌기 때문에 액세스 가능함)의 엔드포인트 장치 하드웨어 지정 비밀키(930)를 이용하여 암호화되어 디지털 서명을 생성할 수 있고, 이 디지털 서명은 단계 914에서 최초 공급된 디지털 서명(920)과 비교된다. 만일 이 디지털 하드웨어 발생 디지털 서명이 후보 코드 블록(910)에 대응하는 수신 디지털 서명(920)과 일치하면, 후보 코드 블록(910)은 검증된 것으로 보아서 실행가능으로 간주할 수 있고, 그렇지 않으면 예외 에러(exception error)가 발생한다(단계 916).

[0109] 도 10은 코드 블록이 (미리 규정된 환경하에서) 특수 엔드포인트 프로세서에서 "보안 실행" 모드로 동작하도록 어떻게 허용되는지에 대한 일 실시예를 보인 블록도이다. 이 특수 경우에, 코드 블록(1011)의 미리 계산된 디지털 서명(1030)(이것은 엔드포인트 지정 복호키라고도 불리운다)은 코드 블록의 메시지 다이제스트를 이용하여 구성되고, 인증된 목표 엔드포인트 장치의 비밀키(1040), 인증된 목표 엔드포인트 장치의 최근 타임스탬프 값(1041), 및 위에서 설명한 하나 이상 임의 갯수의 제약 조건들(이 특수 실시예에서는 도시 생략함) 중에서 하나 이상을 이용하여 암호화된다.

[0110] 이 텀들 중의 임의의 하나는 텀 자체의 부분집합에 마스킹 함수를 적용함으로써 미리 조건지어질 수 있다는 점에 또한 주목해야 한다. 예를 들어서, 타임스탬프 필드의 다수의 최하위 비트가 마스크 오프되면(따라서 디지털 서명의 계산에서 고려될 수 없으면), 그 타임스탬프 값의 유효 입상(effective granularity)이 하드웨어의 어떠한 변경도 없이 코드 세그멘트마다 쉽게 제어될 수 있다. 이러한 동일한 원리는 특정 실시예에서 디지털 서명의 계산에 사용되는 임의 갯수의 텀에 적용될 수 있다.

[0111] 도 7에서 설명한 키 리스트 데이터 구조에서와 같이, 코드 블록(1011)을 내포하는 연쇄 디지털 데이터 집합(1010)은 적어도 하나의 복호 포인터(1012) 및 적어도 하나의 복호 키 또는 키 리스트 포인터(1013)를 또한 포함한다. 또한, 위에서 설명한 것처럼, 이들 중의 임의의 하나는 외부 데이터 구조(엔드포인트 지정 디지털 키 또는 디지털 서명(1030)) 또는 연쇄 데이터 집합(1010)에 전체적으로 내포된 내장 데이터 구조를 인용할 수 있다.

[0112] 도 10에 도시한 실시예를 설명하기 위하여, 코드 블록(1011)은 암호화되지 않았다고 가정한다(따라서 엔드포인트 장치 목표 프로세서에서 잠재적으로 실행가능하다). 이 경우에, 복호 포인터는 사용 전에 코드 블록(1011)에 대해 요구되는 추가의 복호가 없기 때문에 무효(null)이다. 이 경우에서처럼 코드 블록이 암호화되지 않으면, 그 대응하는 복호키(포인터)(1013)는 관련 엔드포인트 또는 하드웨어 지정 디지털 서명(1030)을 지적할 수 있다. 따라서, 도 4a, 도 4b, 도 5a 및 도 5b를 참조하여 위에서 설명한 것과 같은 데이터 구조 및 방법의 실시예를 사용하여 블록 1011에 묘사된 것과 같은 비암호화 데이터 집합의 사용에 대한 매우 다양한 인증, 암호적 결합 또는 기타 제약들을 시행할 수 있다.

[0113] 엔드포인트 지정 디지털 서명(또는 복호키)(1030)이 하드웨어 비밀키(1040)만을 또는 대안적으로 하드웨어 비밀키(1040)와 엔드포인트 장치 타임스탬프 레지스터(1041)만을 지적하는 경우에, 우리는 보안 시스템 관련 호출이 호출 사슬의 "바닥"에 도달하는 것 및 이 특수한 호출 사슬에서 보안 시스템의 추가적인 층에 대한 추가의 호출

이 없을 것이라는 것을 결정할 수 있다. 따라서, 보안 시스템 회귀는 이 지점에서 "종료"된다. 이 회귀 종료 조건은 엔드포인트 지정 하드웨어 비밀키(1040)의 값에 대한 액세스를 선택적으로 허용 또는 거부하는 "문지기"로서, 및 하드웨어 해싱 함수 블록(1061)의 출력을 이용하는 암호 함수에 대한 입력 성분으로서만 작용하는 하드웨어 블록(1050)에 의해 검출된다. 도 10에 도시한 예에서, 하드웨어 지정 비밀키(1040) 및 하드웨어 해싱 함수 블록(1061)의 (메시지 다이제스트) 출력은 암호화 엔진(1062, 1063)에 대한 입력 아규멘트로서 사용된다.

[0114] 마지막으로, 암호화 엔진(1063)의 출력(이것은 최초 연쇄 데이터 구조(1010)의 디지털 서명임)이 전에 공급된 디지털 서명(1030)의 값과 일치하면, "보안 모드 인에이블" 하드웨어 비트(1070)가 세트된다. 이 조건은 엔드포인트 하드웨어 I-캐시(1020)에 로드된 후보 코드 블록(1011)이 이제 "보안" 모드에서 실행하도록 인증되었음을 표시한다. I-캐시(1020)에 상주하는 후보 코드 블록(1011)에 물리적인 변화가 없고 I-캐시(1020) 자체에 어떠한 변화도 없다는 점에 주목한다. 이 지점에서 변경이 있는 유일한 것은 "보안 모드 인에이블" 하드웨어 비트(1070)의 값이다.

[0115] 도 11은 회귀형 보안 시스템에 의해 수행되는 복호 동작의 일 실시예를 보인 도이다. 이 복호 동작은 분배된 콘텐트의 복호 또는 다른 방식의 조작 및 사용하는 처리에서 사용할 보안 코드 블록을 유효화하기 위해 합성키를 사용할 수 있다. 위에서 설명한 것처럼, 엔드포인트 장치는 암호화 콘텐트(1111), 복호 엔진(1120)에 대한 포인터(1112)(또는 복호 엔진 자체) 및 엔드포인트 지정 합성키(1130)에 대한 포인터(1113)(도 9를 참조하여 위에서 설명함)를 내포한 데이터 구조(1110)를 수신한다. 보안 모드에서의 실행을 허용하기 전에, 지적된 또는 수신된 합성 복호 엔진(1140)은 인증될 것이다. 이 인증은 엔드포인트 장치에 상주하는 해싱 함수(1121)를 이용하여 합성 복호 엔진(1140) 코드 블록의 메시지 다이제스트(1122)를 계산함으로써 달성될 수 있다. 비록 이 예에서 해싱 함수(1121)가 하드웨어 블록으로서 묘사되고 있지만, 이 해싱 함수(1121)는 예를 들면 위에서 설명한 것처럼 엔드포인트 장치의 빌트인(built-in) 하드웨어 해싱 함수 대신에 사용될 수 있는 보안 소프트웨어 코드 블록일 수 있다. 그러나, 이 경우, 해싱 함수의 소프트웨어 버전은 궁극적으로 인증 또는 권한 부여 목적의 빌트인 하드웨어 해싱 함수에 여전히 의존하고, 그래서 이 경우의 종국적인 신뢰의 근간은 엔드포인트의 빌트인 하드웨어 해싱 함수 블록(1121)에 여전히 존재한다.

[0116] 이 해싱 블록(1121)에 의해 발생된 메시지 다이제스트(1122)는 그 다음에, 단계 1123에서, 복호 엔진(1140)에 대응하는 미리 계산된 메시지 다이제스트(1150)와 비교된다. 상기 미리 계산된 메시지 다이제스트(1150)는 예를 들면 안전한 방식으로 엔드포인트 장치에 제공되거나, 미리 계산되어 엔드포인트 장치 자체에 저장된다. 만일 메시지 다이제스트가 일치하면, 합성 복호 엔진(1140)은 엔드포인트 장치에서의 실행이 허용된다(단계 1125). 만일 메시지 다이제스트가 실질적으로 동일하지 않으면, 무효 코드 예외가 발생한다(단계 1126).

[0117] 그러나, 만일 메시지 다이제스트가 실질적으로 동일하면, 엔드포인트 장치의 프로세서는 이제 보안 실행 모드로 진입하여 합성 복호 엔진(1140)에 내포된 코드를 실행한다. 이 합성 복호 엔진(1140)의 제1 부분은 엔드포인트 장치의 하드웨어 지정 비밀키(1131)를 활용하여 달성되어 합성키로부터 글로벌 콘텐트 지정 키를 발생한다(단계 1132). 제2 복호 동작(1142)은 획득한 글로벌 콘텐트 지정 키를 이용하여 암호화 콘텐트(1110)로부터 복호 콘텐트(1152)를 발생하기 위해 복호 동작(1141)에 의해 발생된 중간 결과를 이용할 수 있다. 여기에서 주목할 것은 복호 엔진(1140)이 복호 알고리즘의 쌍(1141, 1142)으로서 묘사되어 있지만, 최초 암호화 데이터 집합(1110)에 적용된 보안 코드 블록(1140)의 각종 개별 컴포넌트(1141, 1142 등)의 동작의 최종 결과가 필요한 복호 콘텐트 결과(1152)를 생성하도록 임의의 더 적거나 더 많은 수의 직렬 복호 스테이지를 포함할 수 있다는 것이다. 이러한 각종 개별 복호 컴포넌트 중의 임의의 2개는 동일하거나 다른 알고리즘일 수 있다는 점에 또한 주목하여야 한다.

[0118] 특정 실시예에 있어서, 층에 대한 추가의 보안성을 추가로 요구할 수 있고, 따라서, 일부 실시예에서, 합성키는 도 4a, 도 7c 및 도 10을 참조하여 위에서 설명한 것과 실질적으로 동일한 방법으로 엔드포인트 장치 지정 하드웨어 키 및 엔드포인트 지정 타임스탬프 값을 이용하여 미리 계산된 메시지 다이제스트로부터 형성될 수 있다.

[0119] 도 12는 엔드포인트 장치에서 회귀형 보안 프로토콜을 구현하는 일 실시예를 보인 도이다. 구체적으로, 보안 코드 블록의 유효화를 위해서 및 분배된 디지털 비트스트림의 실제 복호 및/또는 다른 사용을 위해서 합성키 집합을 사용하는 일 실시예를 묘사한다. 이 실시예는 많은 점에서 도 11에 도시한 실시예와 유사하고, 따라서 실시예의 상이한 태양에 대해서만 도 12와 관련하여 설명한다. 암호화 콘텐트(1211)를 포함한 메시지(1210)는 복호 엔진(1240)에 대한 포인터(1212)(또는 복호 엔진 자체), 콘텐트 지정 합성키(1231)(도 8과 관련하여 설명한 것처럼), 및 엔드포인트 및 타임스탬프 지정 합성키(1232)와 함께 수신될 수 있다. 암호화 콘텐트(1211)는 엔드포인트 장치의 메모리에 로드될 수 있고, 복호 엔진(1240)에 대한 포인터(1212)는 메모리(예를 들면, 엔드포인트

장치의 명령어 캐시 또는 명령어 캐시의 보안된 부분)에 또한 로드될 수 있다. 그 다음에, 지적된 복호 엔진(1240)이 인증될 것이다. 이 인증은 도 11을 참조하여 설명한 것과 실질적으로 유사한 방법으로, 엔드포인트 장치에 존재하는 해싱 함수(1221)를 이용하여 암호화 엔진의 메시지 다이제스트를 계산함으로서 달성될 수 있다.

[0120] 이 예에서, 하드웨어 발생 메시지 다이제스트는 암호화 엔진을 이용하여 암호화된다. 상기 암호화 엔진은 엔드포인트 장치에서 하드웨어 또는 소프트웨어로 구현될 수 있고, 엔드포인트 장치 하드웨어 지정 비밀키(1270) 또는 엔드포인트 장치 타임스탬프 레지스터(1260)의 값과 같이, 계산된 메시지 다이제스트 및 하나 이상의 하드웨어 지정 키 또는 레지스터에서 동작하는 하나 이상의 직렬 접속된 합성 암호화 엔진 스테이지(1224, 1225 등)를 포함한다. 발생된 결과적인 합성 디지털 서명(1226)은 복호 엔진 코드 블록(1240)에 정확히 대응하고, 따라서 특정 엔드포인트 장치에 (하나 이상의 암호화 스테이지(1224, 1225) 및 각종 비밀 또는 공개 변수 또는 상수(예를 들면 1260, 1270)를 이용하여) 암호적으로 결합될 수 있다. 위에서 설명한 바와 같이, 상기 발생된 디지털 서명은 (동일하거나 상이한 암호화 엔진을 이용해서 및 상기 합성 디지털 서명의 응용성을 추가로 제한하기 위해 다른 제약 변수 또는 상수를 이용해서) 선택적으로 추가로 암호화될 수 있다. 또한, 이 디지털 서명(1232)과 관련된 코드 블록(1240)의 응용을 단일의 유일한 엔드포인트 장치를 넘어서까지 연장하기 원하는 경우에, 예를 들면, 잠재적인 발생된 합성 디지털 서명 정합의 분야를 넓히기 위해 하나 이상의 암호화 스테이지가 선택적으로 제한될 수 있다.

[0121] 발생된 합성 디지털 서명(1226)은 그 다음에, 단계 1223에서, (예를 들면, 이전 지점에서 엔드포인트 코드 허가 처리의 일부로서 허가 기관에 의해) 엔드포인트 장치에 최초로 제공된 암호화 엔진(1240)에 대응하는 엔드포인트 및 타임스탬프 지정 합성 디지털 서명(1232)과 비교된다. 이 토큰(1232)이 디지털 서명이든 키든 상관없이 데이터 구조는 동일하며, 따라서, 용어 "키"와 "디지털 서명"은 이 경우에 상호 교환적으로 사용될 수 있다는 점에 주목한다.

[0122] 합성 디지털 서명(1226, 1232)이 실질적으로 동일한 경우, 엔드포인트 장치의 프로세서는 복호 엔진 코드 블록(1240)에 내포된 코드를 보안 실행 모드로 동작시키도록 허용될 수 있다. 보안 실행 모드에서 동작할 때, 복호 엔진(1240)은 엔드포인트 장치의 하드웨어 키(1270)를 사용할 수 있고, 복호 엔진(1241 또는 1242)을 이용하여 장치 지정 합성키(1231)로부터 글로벌 콘텐트 지정 키를 발생할 수 있다. 따라서, 글로벌 콘텐트 지정 키는 중간 결과일 수 있고, 따라서 캐시되거나 또는 다른 방식으로 합성 복호 엔진 코드 블록(1240)이 아닌 다른 임의의 소프트웨어 또는 하드웨어 엔티티에게 보여질 수 없다. 이 글로벌 콘텐트 지정 키는 그 다음에 복호 엔진(1243)에 의해 사용되어 최초 암호화 콘텐트(1211)로부터 최종 복호 콘텐트(1250)를 발생한다.

[0123] 그러나, 만일 발생된 디지털 서명(1226)이 공급된 디지털 서명(1232)과 실질적으로 일치하지 않으면, 복호 엔진 코드 블록(1240)의 사용 시도가 비인증 당사자에 의해 이루어진 경우를 포함해서 불일치가 발생하는 몇 가지 가능한 이유가 있다. 그러나, 불일치에 대한 다른 가능한 이유는 복호 엔진의 소프트웨어가 갱신된 경우(및 엔드포인트의 타임스탬프 레지스터가 마찬가지로 증분되거나 다른 방식으로 변경된 경우)일 수 있다. 이 경우에, 2개의 디지털 서명은 일치하지 않을 수 있고, 암호화 엔진 코드(1240)가 (예를 들면) 자기 암호화되었는지 다른 방식으로 교체의 필요성이 있는지를 단계 1281에서 체크할 수 있다. 여기에서 설명하는 실시예는 회귀형 보안 시스템에서 효과적으로 활용될 수 있고, 따라서 많은 경우에 암호화 알고리즘(암호화 콘텐트로 지적된 것 또는 포함된 것)은 그 자신이 암호화될 수 있고 이 암호화된 암호화 알고리즘 자체가 암호화될 수 있다는 것을 상기하자. 그래서, 만일 암호화 알고리즘의 발생된 엔드포인트 및 타임스탬프 지정 합성키(1226)와 수신된 엔드포인트 및 타임스탬프 지정 합성키(1232)가 일치하지 않으면, 이것은 부정 수단(indirection) 또는 암호화의 적어도 하나 이상의 층이 활용된 경우이다.

[0124] 위에서 언급한 바와 같이, 특수 실행가능 코드 블록에 암호화 층을 추가하는 개념은 특수 코드 블록의 구식 버전을 그 코드 블록의 새로운 버전으로 교체하는 동작과 논리적으로 등가일 수 있다. 따라서, 복호 엔진(1240) 자체가 암호화되었는지 또는 (단계 1282에서 표시한 것처럼) 다른 방식으로 교체의 필요성이 있는지를, 그 코드 블록과 관련된 하기의 토큰, 즉 엔드포인트 및 타임스탬프 지정 합성 디지털 서명(1232), 코드 블록의 복호 포인터(도시 생략) 또는 코드 블록의 복호키 포인터(도시 생략) 중 하나 이상을 시험함으로써 결정할 수 있다. 일예로서, 만일 코드 블록(1240)의 관련 복호 포인터가 무효값(null value)을 지적하면, 이것은 암호화 엔진(1240)이 암호화되지 않았거나 또는 다른 방식으로 구식으로 되지 않았음을 표시하고, 따라서 발생된 디지털 서명(1226)과 공급된 디지털 서명(1232)이 실질적으로 동일하지 않지만, 코드 블록을 정확한 디지털 서명을 생성할 수 있는 다른 버전으로 교체하기 위한 다른 의지(recourse)가 없기 때문에 예외 에러가 발생할 수 있다(단계 1283). 그러나, 만일 복호 엔진 코드 블록(1240)의 복호 포인터가 다른 코드 블록, 즉 다른(아마도 갱신된) 암호화 엔진(도시 생략) 또는 어떤 다른 코드 블록을 지적하면, 이 새로운 코드 블록이 로드되고 위의 인증 단계

들이 다음 암호화 엔진에 적용될 수 있다(다시 말하면, 다른 하나의 회귀층이 도입될 수 있다). 이 회귀형 실행 메카니즘은 발생된 엔드포인트 및 타임스탬프 지정 합성 디지털 서명(1226)과 공급된 엔드포인트 및 타임스탬프 지정 합성 디지털 서명(1232) 간의 일치가 발생(단계 1227에서)하였다고 결정되거나 또는 일치가 없고 복호 엔진(1240) 자체가 암호화되지 않았다고 결정될 때까지 계속되고, 그 지점에서 예외 에러가 발생할 수 있다(단계 1283).

[0125] 만일 발생된 엔드포인트 및 타임스탬프 지정 합성 디지털 서명(1226)과 공급된 엔드포인트 및 타임스탬프 지정 합성 디지털 서명(1232)이 일치한다고 결정되면, 회귀가 종료되고 언운드(unwound)될 수 있다. 이것은 전체적인 회귀형 호출 사슬을 통한 초기의 순방향 통과 중에 스택에서 조우되어 세이브된 각 코드 블록의 인증 및 실행을 수반한다. 이들 코드 블록의 일부 또는 아마도 모두는 반드시 암호화 엔진 또는 복호 엔진일 필요가 없다는 것에 주목하여야 한다. 어느 경우이든, 이들 각 코드 블록은 목표 엔드포인트 장치가 보안 실행 모드에서 동작하는 동안 인증될 수 있다.

[0126] 이 실행은 회귀형 보안 시스템에 의해 수행될 수 있는 복호 동작의 일 실시예를 도시하는 도 13을 참조하여 더 잘 설명될 수 있다. 도시된 바와 같이, 엔드포인트 장치는 메시지(1310)를 수신할 수 있고, 이 메시지(1310)는 다른 무엇보다도 콘텐트 지정 합성키(1316)(도 8과 관련하여 설명함)와 함께 암호화 콘텐트(1312), 복호 엔진 데이터 구조(1320)에 대한 포인터(1313) 또는 복호 엔진 자체를 내포할 수 있으며, 최초 메시지(1310) 및 키 리스트 포인터(1314)에 매립되어 있으면 키 또는 키 리스트 데이터 구조(1318)를 지적할 수 있다. 위에서 설명한 바와 같이, 이 데이터 구조는 키 또는 키 리스트(1316) 또는 디지털 서명(1317)을 포함할 수 있다. 복호 엔진 데이터 구조(1320)는 암호화 코드 블록(1321), 암호화(또는 대안적으로 퇴화하여 교체의 필요성이 있는) 복호 코드 블록(1321)과 관련된 후속 복호 포인터(1322) 및 관련 복호 키 리스트 포인터(1323)를 내포할 수 있다. 후속 복호 포인터(1322)는 최종 복호 코드 블록 데이터 구조(1330)를 지적할 수 있고, 상기 최종 복호 코드 블록 데이터 구조는 데이터 구조(1330)의 경우에 복호 코드 블록(1331)이 암호화 형태로 있지 않은 것을 제외하면 복호 코드 블록 데이터 구조(1320)의 구조와 실질적으로 유사한 구조를 갖는다.

[0127] 도 13에 도시한 실시예의 동작은 다음과 같이 설명할 수 있다. 암호화 콘텐트 데이터 구조(1310)는 내포된 암호화 콘텐트(1312)의 복호를 예상하여 엔드포인트 프로세서의 메모리 스페이스에 로드된다. 데이터 구조(1310)는 복호 포인터(1313)를 내포하기 때문에, 관련 복호 엔진 코드 블록 데이터 구조(1320)는 메모리에 위치정해지고 읽어 들여진다. 이 후속 데이터 구조(1320)가 또한 복호 포인터(1322)를 내포하기 때문에, 포인터(1322)와 관련된 복호 엔진 코드 블록 데이터 구조(1330)는 메모리에 위치정해지고 로드된다. 데이터 구조(1330)에 있어서, 이 예에서 매립 복호 포인터(1332)는 무효 포인터로 결정되고, 그래서 목표 엔드포인트 장치의 보안 시스템이 현재 복호 회귀 사슬이 종료되었다고 결정할 수 있고(예를 들면 도 10에서 설명한 것처럼), 따라서 데이터 구조(1330)의 일부로서 메모리에 막 읽어 들여진 복호 엔진(1331)은 비암호화(및 따라서 잠재적으로 실행가능한) 코드 블록을 내포할 수 있다.

[0128] 디지털 콘텐트(1331)가 코드 블록이고 데이터가 아니라고 결정될 수 있기 때문에(디지털 콘텐트가 호출된 방법으로), 복호 키 리스트 포인터(1333)(데이터 구조(1330)의 일부로서 메모리에 읽어 들여진 것)에 의해 지적된 키 리스트 데이터 구조(1338)가 디지털 서명(1337)(합성키(1336)에 추가해서)을 내포하고 있다고 또한 결정될 수 있다. 이 예에서의 키 리스트 데이터 구조(1318, 1328, 1338)는 도 7b와 관련하여 위에서 설명한 범용 암호 데이터구조를 이용하여 구현될 수 있다는 점에 또한 주목하여야 한다. 따라서, 이들 키 리스트 데이터 구조(1318, 1328, 1338)에서의 아규멘트의 순서는 반드시 고정될 필요는 없고, 따라서 이들은 런타임에서 데이터 구조 자체가 관통된 것으로 해석될 수 있다. 사실, 이들 키 리스트 데이터 구조(1318, 1328, 1338)는 키 리스트 데이터 구조(1318, 1328, 1338)의 일부 또는 전부에 보충적 복호 포인터 및 키 리스트 포인터를 통합함으로써 추가의 복호 또는 후속되는 해석을 위한 참조(reference)를 포함할 수 있지만, 이러한 특수한 옵션은 간단성을 위해 도 13의 실시예에서 도시하지 않았다.

[0129] 키 리스트 데이터 구조(1338) 내의 적어도 하나의 키 포인터(1336)는 엔드포인트의 하드웨어 비밀키(1392)에 대한 참조에 대응한다고 또한 결정될 수 있다. 엔드포인트의 하드웨어 비밀키(1392)에 대한 이러한 참조는 적절히 보존된 메모리 위치(프로세서에 의해 직접 판독될 수 없고 따라서 구조적으로 직접 보여지지 않더라도 프로세서 아키텍쳐에 특정될 수 있는 위치)를 지적함으로써 명시적으로, 또는 포인터에 대해 어떤 특별하게 보존된 값을 이용함으로써 암시적으로 달성될 수 있다. 어느 경우이든, 이 참조는 다양한 수단을 이용하여 구현될 수 있지만, 그러한 실시예 중의 한가지 예는 키 리스트 데이터 구조의 "0"의 값("무효"의 값과는 다른 것임)을 엔드포인트의 하드웨어 비밀키(1392)에 대한 참조와 같게 하는 것이다. 키 리스트 데이터 구조의 적어도 일부분이 엔드포인트의 하드웨어 비밀키(1392)를 인용한다는 사실은 복호 엔진 코드 블록(1331)이 목표 엔드포인트 장치

의 프로세서에서 보안 실행 모드로 동작하려고 의도한다는 것을 또한 표시할 수 있다. 따라서, 하드웨어 기반 디지털 서명 발생기 블록(1390)의 출력은 데이터 구조(1337)에 저장된 값과 비교된다. 2개의 값이 실질적으로 일치하는 경우에, 프로세서는 보안 실행 모드로의 진입이 허용된다.

[0130] 여기에서 주목해야 할 것은 하드웨어 기반 디지털 서명 발생기 블록(1390)(이것에 대한 일 실시예의 상세한 사항은 도 15를 참조하여 더 포괄적으로 제시될 것이다)은, 일 실시예에 있어서, 하나 이상의 소프트웨어 기반 요소를 포함할 수 있지만, 위에서 설명한 것처럼 적어도 하나의 하드웨어 기반 보안 컴포넌트를 직접적으로 또는 간접적으로 또한 포함할 수 있다. 이 하드웨어 컴포넌트는 위에서의 많은 설명에서 인용된 하드웨어 기반 해싱 함수이고, 이것은 전체 목표 엔드포인트 유닛 보안 시스템의 신뢰의 근간을 포함한다.

[0131] 이 점에서, 복호 엔진 코드 블록(1331)은 보안 실행 모드에서 동작하는 것이 허용되고, 이것은 엔드포인트 프로세서가 엔드포인트의 하드웨어 장치 지정 비밀키(1392)를 보안 관련 계산(위에서 설명됨)의 일부로서 잠재적으로 사용하는 것을 허용한다. 프로세서가 보안 실행 모드에서 동작하지 않는 경우에, 비밀키(1392)의 값은 그러한 보안 관련 계산에서 사용될 수 없다. 이 개념은 하드웨어 액세스 제어 블록(1343)으로서 도 13과 관련하여 설명되고, 이것은 프로세서가 보안 실행 모드에서 동작하는 경우 비밀키(1392)의 값만이 (예를 들면 복호 엔진 코드 블록(1331)에서) 후속 사용을 위해 통과하도록 허용할 것이다.

[0132] 또한, 하드웨어 액세스 제어 블록(1343)에 대한 입력 파라미터 중의 하나는 액세스 제어 블록(1341)의 출력임을 알 수 있다. 이 방법으로, 하드웨어 액세스 제어 블록(1343)의 상태(이것은 실제상으로는 복호 코드 블록(1321)의 "보안 실행 모드 인에이블" 표시자이다)는 복호 코드 블록(1331)이 보안 실행 모드에서 또한 동작했다는 사실에 의존한다. 이것은 복호 코드 블록(1331)의 "보안 실행 모드 인에이블" 표시자의 상태(예를 들면, 하드웨어 액세스 제어 블록(1341)의 출력)에 의해 표시될 수 있다. 이 의존성은 복호 코드 블록(1331)이 보안 실행 모드에서 동작한 경우에만 보안 실행 모드에서 동작할 수 있는 복호 엔진 코드 블록(1321)의 능력을 억제한다. 본질적으로 동일한 방법으로, 하드웨어 액세스 제어 블록(1343)의 출력은 하드웨어 액세스 제어 블록(1345)에 대한 입력들 중 하나로서 사용되고, 이것은 복호 코드 블록(1311)의 "보안 실행 모드 인에이블" 표시자이다. 따라서, "보안 실행 모드 인에이블" 비트가 전파되게 하는 메카니즘은 선행하는 부모 코드 블록이 적절히 인증된 경우(도 14를 참조하여 더 자세히 설명함) 및 선행하는 부모 코드 블록이 회귀형 호출 사슬의 하부로부터 보안 사슬의 적절히 권한이 부여된 부분으로부터의 인증 복호 결과와 함께 공급된 경우에만 선행하는 부모 코드 블록이 보안 실행 모드에서 동작하도록 권한을 부여할 할 목적으로 호출 사슬을 역방향으로 백업한다. 위에서 설명한 것처럼, 몇 가지 조건들 중 임의의 조건은 임의의 "보안 실행 모드 인에이블" 비트가 "안전하지 않은" 디폴트 상태로 리세트시킬 수 있다는 점에 주목한다(따라서, 전체 보안 사슬이 재시작될 것을 잠재적으로 요구한다). 그러한 조건은 프로세서 인터럽트 또는 후속하는 디지털 서명 비교 불일치를 포함할 수 있다. 비록 이들 하드웨어 액세스 제어 블록(1341, 1343, 1345)이 도 13에서 명확성을 위해 별도의 엔티티로 도시되어 있지만, 이들은 사실상 출력이 그 자신의 입력 텀 중의 하나로서 피드백되는 단일 하드웨어 유닛으로 (도 15와 관련하여 설명하는 바와 같이) 구체화될 수 있다는 것을 알 수 있다. 궁극적으로, 전체 사슬에서 최고 레벨 또는 최종 "보안 실행 모드 인에이블" 비트의 출력은 목표 장치의 일부 외부적으로 보여지는 출력을 (예를 들면, 오디오 또는 비디오 출력 인에이블과 같이) 인에이블 또는 디스에이블하기 위한 제어 메카즘의 일부로서 사용될 수 있다.

[0133] 단계 1370에서 복호 엔진 코드 블록(1331)의 동작은 데이터 구조(1320)의 복호 엔진 코드 블록부(1321)에 저장된 데이터 집합을 최초 데이터의 갱신된 및/또는 적절히 실행가능한 버전으로 교체 또는 다른 방식으로 보충하는 것이다. 이 동작은 복호 코드 블록(1321)에 저장된 최초 데이터를 이용하고 키 리스트 데이터 구조(1328)에 의해 저장되거나 지적된 하나 이상의 복호 키로 최초 데이터를 복호함으로써 달성될 수 있다. 대안적으로, 위에서 설명한 바와 같이, 복호 엔진 코드 블록(1331)의 동작(1370)은 복호 코드 블록(1321)을 갱신 버전으로 교체하거나 복호 엔진 코드 블록(1321) 대신에 직접 실행하게 할 수 있다. 어느 경우이든, 복호 엔진 코드 블록(1331)은 (이 실시예에서) 목표 엔드포인트 장치의 타임스탬프 레지스터(1394)에 내포된 값, 목표 엔드포인트 장치의 하드웨어 지정 비밀키(1392)(하드웨어 액세스 제어(1342)를 통과함으로써 수정됨) 및 엔드포인트 및 타임스탬프 지정 합성 디지털 키(1326)를 포함하는 각종 입력 데이터를 이용하여 최초로 동작할 수 있다. 복호 엔진 코드 블록(1331)이 복호 엔진 코드 블록(1321)의 직접 교체로서 후속적으로 동작하는 경우에, 복호 엔진 코드 블록(1331)은 예를 들면, 이 실시예에서 목표 엔드포인트 장치의 타임스탬프 레지스터(1394)에 내포된 값, 목표 엔드포인트 장치의 하드웨어 지정 비밀키(1392)(하드웨어 액세스 제어(1344)를 통과함으로써 수정됨) 및 엔드포인트 및 타임스탬프 지정 합성 디지털 키(1316)를 포함하는 입력 데이터의 제2 집합을 활용할 수 있다.

[0134] 단계 1371에서 갱신 복호 엔진 코드 블록(1321)의 추가의 동작은 소망의 출력 데이터(1380)를 생성하기 위해 최초의 암호화 콘텐트 데이터(1312)를 교체 또는 다른 방식으로 해석하는 것이다. 이 동작은 복호 코드 블록

(1321)에 저장된 최초 데이터를 이용하고 키 리스트 데이터 구조(1318)에 의해 저장되거나 지적된 하나 이상의 복호 키로 최초 데이터를 복호함으로써 달성될 수 있다. 양측의 복호 엔진 코드 블록(1321, 1331)의 동작이 사실상 유사하기 때문에, 복호 엔진 코드 블록(1331)의 동작과 관련하여 위해서 구체적으로 설명한 임의의 옵션이 복호 엔진 코드 블록(1321)의 갱신 버전의 동작에도 동일하게 적용할 수 있음을 명백히 하여야 한다. 또한, 복호 엔진 코드 블록(1321)의 동작의 경우에, 일부 실시예에서, 관련 하드웨어 액세스 제어 블록(1344)은 하드웨어 액세스 제어 블록(1342)과 구별된다는 점에 주목하여야 한다. 그러나, 상기 2개의 하드웨어 액세스 제어 블록(1342, 1344)의 동작은 그들의 목적이 그들의 관련 복호 엔진(1331 또는 1321)에 의한 목표 엔드포인트 장치의 하드웨어 지정 비밀키(1392)의 사용을 가능 또는 불가능하게 하는 것이라는 점에서 사실상 유사하고, 따라서 다른 실시예에서는 구별되지 않는다.

[0135] 마지막으로, 위에서 설명한 도 13의 실시예로 묘사한 모든 동작에 있어서, 목표 엔드포인트 장치의 타임스탬프 레지스터(1394)의 사용은 다른 실시예로 위에서 설명한 예들과 본질적으로 유사하다. 따라서, 레지스터(1394)에 저장된 값은 도 13에 도시한 특수 실시예에서 설명한 다른 인증 및 복호 동작에서 사용한 각종 합성키 및/또는 디지털 서명의 생성시에 추가적인 요소로서 사용될 수 있다.

[0136] 도 14는 회귀형 호출 사슬이 어떻게 관통 및 종료되는지 및 프로세서가 하나 이상의 매립 코드 블록의 메시지 다이제스트 기반 인증을 이용하여 어떻게 보안 실행 모드로의 진입이 허용되는지에 대한 일 실시예를 도시한 것이다. 이 실시예에서는 도 7b와 관련하여 위에서 설명한 것처럼 범용 암호 데이터 구조(1411, 1421)에 각각 내포될 수 있는 2개의 후보 코드 블록(1412, 1422)의 동작이 설명된다.

[0137] 코드 블록 데이터 구조(1421)는 도 14에서 2회 표시된다는 점에 주목한다. 이 중복은 명확히 할 목적으로 별도의 반복을 표시하기 위해 도시한 것이지만, 이것은 두 가지 예에서 정확히 동일한 데이터 구조임을 알아야 한다. 그러나 인지될 수 있는 한가지 차이점은 키 리스트 포인터(1421)의 사례에 의해 지적되는 키 리스트 데이터 구조(1428, 1438)에 있다. 비록 키 리스트 포인터(1421)의 값이 이 도면에 도시한 2개의 사례 사이에서 변하지 않지만, 키 리스트 데이터 구조(1428)에 내포된(또는 키 리스트 데이터 구조(1428)에 의해 지적된) 값은 2개의 반복 사이에서 변할 수 있고, 따라서 이것의 세부는 데이터 구조(및 그 각종 컴포넌트)의 참조 번호를 1426, 1427 및 1428로부터 1436, 1437 및 1438로 각각 다시 번호 붙임으로써 표시된다. 이 구조가 다시 번호 붙임되었다는 사실은 데이터 구조의 실제 위치가 이동되었다는 것 및 그 콘텐트가 변경되었다는 것을 표시하는 것이 아니다. 마찬가지로, 명확성을 높이기 위해, 하드웨어 해싱 함수(1480)가 이 도면에서 또한 복수 회 도시되었다. 마지막으로, 2개의 후보 코드 블록(1421, 1422)의 어느 것도 암호화되지 않고, 따라서 그 관련 복호 포인터(1416, 1426, 1436)는 모두 무효(null) 포인터일 수 있다.

[0138] 이 실시예에 있어서, 후보 코드 블록(1412)에 대한 호출이 개시될 수 있다. 위에서 설명한 것과 동일한 방식으로, 코드 블록 데이터 구조(1411)는 메모리에 읽어 들일 수 있고, 그 메시지 다이제스트(1441)는 해싱 함수(1480)(이것은 위에서 설명한 것처럼 전체적으로 또는 부분적으로 하드웨어로 실현될 수 있다)에 의해 계산될 수 있다. 그러나, 이 실시예에서, 해싱 함수는 초기 종자값(1440)(모두 0으로 세트될 수도 있고 아닐 수도 있다)이 주어진다. 위에서 설명한 것처럼, 이 해싱 함수 종자값 특징은 다수의 방법들 중 하나를 이용하여 구현될 수 있지만, 이 실시예에서는 종자값(1440)이 공지되어 있고, 종자값이 해싱 함수 블록(1480)의 메시지 다이제스트 출력(1441)에 영향을 주는 방법은 반복적이며 결정론적이다.

[0139] 해싱 함수의 결과(1441)가 생성되면, 프로세서는 코드 블록(1412)에 내포된 코드의 실행을 시작할 수 있다. 도 14에 도시한 실시예에 있어서, 키 리스트 포인터(1414)에 의해 지적된 2개의 위치(1416, 1417)(이것은 키 리스트 데이터 구조(1418) 내측에 포함되어 있다)의 값 및 복호 포인터(1413)는 모두 무효(null)이고, 코드 블록(1412)은 보안 실행 모드에서 동작하도록 설계되지 않을 수 있으며, 따라서 임의의 목표 엔드포인트 장치 보안 하드웨어 특징의 사용을 요구하지 않는다. 따라서, 프로세서는 코드 블록(1422)을 지적하는 매립 서브루틴 호출에 도달할 때까지 코드 블록(1412)에 내포된 명령어들의 실행을 시작한다.

[0140] 이 점에서, 코드 블록 데이터 구조(1421)는 메모리에 로드되고, 다음 메시지 다이제스트(1442)를 생성하는 처리가 해싱 함수 블록(1480)에 의해 반복된다. 그러나, 이 특수한 사례에 있어서, 해싱 함수 종자값은 더 이상 초기 종자값(1440)이 아니고 오히려 미리 생성된 결과(1441)이다. 따라서, 메시지 다이제스트(1442)의 값은 양쪽 코드 블록(1411, 1421)의 메시지 다이제스트에 결정론적으로 의존한다는 것을 알 수 있다. 그러나, 이전 경우에서처럼, 복호 포인터(1423)의 값 및 키 리스트 포인터(1424)에 의해 지적된 키 리스트 데이터 구조(1428)에 내포된 값들은 여전히 무효(null)이고, 따라서 프로세서는 앞에서처럼 비보안 실행 모드에서의 동작을 계속한다.

[0141] 어떤 나중의 지점에서, 프로세서는 다른 서브루틴 호출을 만나지만, 이 예에서 코드 블록(1422)은 회귀형 호출

(예를 들면, 자신에 대한 서브루틴 호출)을 내포한다. 특정 실시예에 있어서, 그러한 회귀형 호출 구조는 단지 예시적인 것이고, 목표 엔드포인트 장치 보안 시스템의 정확한 동작은 예를 들면 보안 시스템에 대한 임의의 호출이 단일 코드 층에 내포된 것을 보증하는 다른 수단에 의해 달성될 수 있다. 그러나, 보안 시스템의 복수의 레벨이 관통되자마자 회귀형 호출 형태가 위에서 상세히 설명한 것처럼 상대적으로 더 안전하게 되고, 도시된 실시예와 함께 보안 시스템을 구현하기 위해 효과적으로 활용될 수 있다.

[0142] 임의의 경우에, 프로세서가 코드 블록(1422)(자신을 참조함) 내에 매립된 서브루틴 호출과 만나면, 코드 블록 데이터 구조(1421)는 메모리에 다시 한번 로드되고(예를 들면, 대부분의 현대 시스템에서 데이터 구조(1421)는 두번째 인출된 때 물리적으로 다른 위치에 로드된다), 해싱 함수(1480)는 새로운 메시지 다이제스트(1443)를 계산한다. 이 새로운 메시지 다이제스트(1443)는 초기 메시지 다이제스트 종자값(1440), (코드 블록(1421)의) 메시지 다이제스트(1441) 및 코드 블록(1422)의 2개의 별도의 반복의 메시지 다이제스트에 의존한다는 점에 주목한다.

[0143] 이 두번째 시간에 키 리스트 포인터는 무효가 아닌(non-null) 디지털 서명값(1437)을 내포하는 새로운 데이터 구조(1438)를 지적한다. 이 무효가 아닌 값은 코드 블록(1422)의 반복이 목표 엔드포인트 하드웨어 지정 보안 시스템에 대한 참조를 내포한다는 것을 보안 시스템에 표시하는 표시자이다. 따라서, 이 실시예에서, 이러한 참조가 적절히 동작하기 위하여, 프로세서는 어떤 지점에서 보안 실행 모드로 진입하여야 한다. 따라서, 코드 블록 데이터 구조(1421)가 가장 최근에 메모리에 로드된 때 생성된 디지털 서명(1443)은 키 리스트 데이터 구조(1438)에 내포된 디지털 서명(1437)과 비교된다. 단계 1491에서 상기 2개의 값이 실질적으로 유사한 것으로 판정된 경우에, 목표 엔드포인트 프로세서는 보안 실행 모드로의 진입이 허용된다. 그러나, 만일 2개의 디지털 서명 값(1437, 1443)이 일치하지 않으면(및 디지털 서명(1437)이 이 지점에서 무효가 아닌 것으로 알려지면), 단계 1492의 결과가 프로세서에 제공되어 보안 시스템의 적당한 예외 에러 핸들러 부분(1470)을 실행시킨다.

[0144] 도 15는 위에서 설명한 특징들을 지원하기 위해 디지털 서명 발생기 블록(1560)이 어떻게 하드웨어에서 구현될 수 있는지에 관한 일 실시예를 보인 도이다. 도 15에 도시한 실시예는 도 10에 도시한 디지털 서명 발생기 블록의 기능과 유사한 기능의 하드웨어 구현을 보여주고 있고, 이것은 예를 들면 도 11, 12, 13 및 14와 관련하여 동작상의 세부로 설명한 기능적 특징들을 지원할 것이다.

[0145] 해싱 함수 종자 레지스터(1510)는 도 14에 블록(1440)으로 표시된 기능과 유사한 기능을 포함할 수 있고, 해싱 함수 블록(1561)에 공급된 초기값을 유지하도록 동작할 수 있다. 해싱 함수 블록(1561)의 출력은 합성 암호화 엔진의 제1 단계(1562)에 대한 입력들 중 하나로서 공급된다. 암호화 엔진(1562)의 다른 입력은 목표 엔드포인트 장치 타임스탬프 레지스터(1541)의 출력이다. 제1 단계 암호화 엔진(1562)의 결과적인 출력은 다음에 제2 단계 암호화 엔진(1563)의 입력들 중 하나로서 공급된다. 제2단계 암호화 엔진(1563)의 다른 입력은 보안 실행 모드 액세스 포인트(1566)의 출력이다.

[0146] 액세스 포인트(1566)는 도 14를 참조하여 위에서 설명한 것처럼, 목표 엔드포인트 장치가 보안 실행 모드에서 동작할 때 또는 "회귀 종료" 조건이 검출된 때에만 목표 엔드포인트의 하드웨어 지정 비밀키(1540)의 값을 통과하도록 동작한다. 제2 단계 암호화 엔진(1563)으로부터의 결과적인 출력 값은 디지털 서명 레지스터(1564)에 저장되고, 상기 발생된 디지털 서명을 후보 코드 블록에 공급된 디지털 서명과 비교할 때 사용된다(예를 들면, 도 9, 도 10, 도 11, 도 12, 도 13 및 도 14의 설명에서 인용됨).

[0147] 디지털 서명 레지스터(1564)의 출력은 액세스 포인트(1565)에 의해 제어되고, 그 동작은 목표 엔드포인트 장치가 보안 실행 모드에서 동작하지 않을 때 디지털 서명 레지스터(1564)의 값을 통과하는 것이다. 그 다음에, 액세스 포인트(1565)의 출력은 해싱 함수 종자 레지스터(1510)의 입력으로 피드백되어 도 14와 관련한 설명에서 구체적으로 설명한 직렬 메시지 다이제스트 특징을 생성한다. 목표 엔드포인트 장치가 보안 실행 모드에서 동작하면, 해싱 함수 종자 레지스터(1510)의 입력은 디지털 서명 레지스터(1564)의 값에 의존하지 않고, 따라서 임의의 초기값으로 설정될 수 있고(도 14와 관련한 설명에서 설명됨) 또는 어떤 다른 수단에 의해 설정될 수 있다(예를 들면, 프로세서가 특정 메모리 위치에 기록한다).

[0148] 지금까지 특정 실시예를 참조하여 본 발명을 설명하였다. 그러나, 이 기술에 통상의 지식을 가진 자라면 청구범위에서 규정하는 본 발명의 범위로부터 벗어나지 않고 여러 가지로 수정 및 변경이 가능하다는 것을 알 것이다. 따라서, 명세서, 부록 및 도면은 제한하는 의도라기보다는 예시적인 것으로 간주되어야 하고, 그러한 수정들은 모두 임의의 제한적인 용어의 사용에도 불구하고 본 발명의 범위에 포함된 것으로 의도된다.

[0149] 잇점, 다른 장점 및 문제점에 대한 해법을 특정 실시예와 관련하여 위에서 설명하였다. 그러나, 잇점, 장점, 문

제점에 대한 해법, 및 임의의 잇점, 장점 또는 해법을 발생시키거나 더 명백하게 하는 임의의 컴포넌트는 임의의 또는 모든 청구항의 중요한, 요구된, 또는 본질적인 특징 또는 컴포넌트로서 해석되어서는 안된다.

[0150] **부 록 A**

[0151] 본 발명 및 그 각종 특징과 유리한 세부는 첨부 도면에 도시되고 이하의 설명에서 상세화되는 비제한적인 실시예를 참조하여 더욱 완전하게 설명된다. 잘 알려져 있는 시작 물질, 처리 기술, 컴포넌트 및 설비에 관한 설명은 본 발명의 세부를 불필요하게 불명료하게 하는 것을 방지하기 위해 생략한다. 그러나, 본 발명의 양호한 실시예를 표시하는 것이지만, 그 실시예에 대한 상세한 설명 및 특수한 예는 제한하는 의도가 없이 단지 예시하는 용도로만 주어진다는 것을 이해하여야 한다. 근원적인 발명 개념의 정신 및/또는 범위 내에서의 각종 치환, 수정, 추가 및/또는 재구성이 이 기술에 숙련된 사람에게는 이 명세서의 설명으로부터 명백할 것이다.

[0152] 이제, 디지털 콘텐트를 보호하기 위한 보안 프로토콜의 시스템 및 방법에 대하여 살펴본다. 이 보안 프로토콜은 임의의 디지털 콘텐트에 대해 사용가능하고, 실제 디지털 콘텐트의 변경을 요구하지 않고 일반적으로 전통적인 워터마킹 방식과 관련된 아이덴티티 트레이싱의 개념을 또한 지원할 수 있다. 이 프로토콜은 모든 디지털 비트 스트림이 동일하다는 전제에 기초를 두기 때문에, 프로토콜 자체의 갱신을 위한 액세스를 제어하기 위해 회귀 형식으로 사용될 수 있다. 다시 말해서, 프로토콜은 데이터가 보호대상의 미디어 스트림이든지, 그러한 스트림을 재생하기 위해 필요한 실행가능 코드이든지, 그러한 스트림을 재생하기 위해 필요한 암호화 실행가능 코드이든지, 그러한 스트림을 재생하기 위해 필요한 암호화 코드를 복호하기 위해 필요한 실행가능 코드이든지, 복호 코드와 함께 사용되는 키이든지, 등등과 상관없이 디지털 데이터의 종류들 간에 구별을 만들지 않는다. 이러한 데이터의 디지털 속성은 모두 프로토콜에 중요하다. 따라서, 디지털 데이터의 속성 및/또는 용도가 보안 프로토콜에 관심이 없기 때문에, 프로토콜은 그 자체를 보호할 수 있다.

[0153] 이 능력은 보안 프로토콜이, 실행중이라 하더라도, 프로토콜이 동작하는 하드웨어의 어떠한 변경을 요구하지 않고 (예를 들면, 최근에 발견된 보안 취약점(security hole)을 수리하기 위해) 갱신될 수 있음을 의미한다. "오래된" 보안 시스템은 새로운 보안 시스템이 일부로서 포함된다(즉, 새롭고 잠재적으로 더 안전한 보안 레벨을 전체 시스템에 추가하기 위해 오래된 보호 "래퍼(wrapper)"를 벗겨낼 필요가 없다). 따라서, 전체 시스템이 최근의 가장 안전한 암호화 및/또는 액세스 제어 시스템으로 보호된다. 새로운 키가 추가될 뿐만 아니라, 완전히 새로운 보안 및/또는 암호화 알고리즘이 현재의 시스템에 또한 추가될 수 있다.

[0154] 이 융통성은 시간 제한이 있는 대여, 페이 퍼 뷰, 다중 버전화(multiple versioning), 머신 의존형 허가 취소 및 하나의 사용자로부터 다른 사용자로 소유권의 영구 양도를 포함한 다수의 사업 모델을 프로토콜이 지원할 수 있게 한다.

[0155] 저작권이 있는 소프트웨어 애플리케이션이 예시적인 실시예에서 활용되지만, 이 기술에 숙련된 사람이라면 동일한 방법 및 시스템을 사용하여 텍스트, 비디오 및 오디오 데이터, 소스 및 목적 코드 등을 포함한 모든 비트스트림에 대해 보안을 제공할 수 있다는 것을 이해할 것이다.

[0156] 보안 프로토콜을 구체화하는 기본 기능은 하기의 것을 제공하도록(그러나 제한하는 것은 아님) 설계된다.

[0157] 공정한 사용("시간 이동", "공간 이동" 및 보관소 백업)

[0158] 증진적 업그레이드

[0159] 소유권의 임시 양도

[0160] 소유권의 영구 양도

[0161] 시간 제한 액세스

[0162] 사용량 제한 액세스(사용 횟수)

[0163] 장치 지정 허가 취소

[0164] 데이터 또는 스트림 지정 허가 취소

[0165] 많은 보안 시스템에 있어서, 저작권이 있는 작품에 포함된 지적 재산의 보호를 위한 기본 메카니즘 중의 하나는 단순한 액세스 제어이다. 그러나, 이러한 메카니즘이 우회되면, 가장 복잡한 액세스 제어 메카니즘에 의해 제공된 보호는 매우 작은 가치를 갖는다. 이것은 액세스 제어가 무용 메카니즘이고 본질적으로 및 자연히 전체 보안 시스템이 아니라고 말할 수 있다. 다수의 저작권있는 미디어 스트림이 인터넷에서 공개 소비에 자유롭게 이용가

능하다는 사실은 그러한 보안 시스템이 거의 항상 우회될 수 있다는 사실의 증거이다. 이러한 종류의 액세스 제어는 합법적으로 구매한 저작권 작품의 카피를 백업하도록 메카니즘을 구성하는 것이 더욱 어렵게 하고, 이것은 원본이 파괴될 위험에 처해있는 경우 필요하다. 따라서, 여기에서 설명하는 보안 프로토콜은 보안 프로토콜을 유용하게 하기 위해 어떠한 종류의 액세스 제어 시스템도 요구하지 않는다.

[0166] 여기에서 설명하는 보안 프로토콜은 저작권 작품의 표현을 제어하는 것에 집중되고, 작품 자체를 구성하는 디지털 데이터에 집중되는 것이 아니다. 그래서, 프로토콜은 저작권 작품을 보호하기 위해 사용하는 디지털 데이터 또는 그 작품이 어떻게 해석되어야 하는지를 설명하기 위해 사용하는 다른 디지털 데이터들 사이에 구별을 만들지 않는다. 그 결과, 프로토콜은 다른 보안 프로토콜을 보호하기 위해 사용될 수 있다.

[0167] **기본 동작 설명:**

[0168] 보안 프로토콜의 실시예는 소프트웨어 부분의 창조자가 그들의 코드가 그 알고리즘을 복사 또는 다른 방식으로 악용하고자 하는 사람에 의해 분해되는 것으로부터 보호되는 높은 신용 정도를 가질 수 있도록 설계된다. 실시예는 또한 코드의 기능을 변경하려고 하는 사람에 의해 코드가 수정되는 것으로부터 보호하도록 설계된다. 이러한 기본 특징이 다른 범용 컴퓨팅 시스템에서 구현될 수 있게 하는 방법들 중의 하나는 다음 섹션에서 설명된다. 이러한 2가지 기본 기능의 부산물로서 발생하는 추가의 특성은 소프트웨어가 동작할 수 있는 조건(즉, 언제 어떻게 어느 머신에서 코드의 실행이 허용되는지)을 제어하는 능력이다. 이 기능들 중 첫번째의 것은 변경 방지(tamper-resistant) 타이머 요소를 시스템에 추가함으로써 달성될 수 있다. 다른 기능은 당해 코드 블록을 실행하기 위해 부합되어야 하는 소정의 조건들을 표시하기 위해 사용되는 보안 데이터 구조를 구현함으로써 달성된다. 이 데이터 구조는 하드웨어 지정이 아니기 때문에, 다양한 방법으로 사용될 수 있고, 그것을 해석하기 위해 사용한 소프트웨어를 갱신함으로써 수정될 수 있다. 프로토콜을 더 효율적으로 구현하기 위해 활용되는 하드웨어 지정 특징들이 설명되고, 프로토콜을 지원하기 위해 상기 특징들을 어떻게 사용할 수 있는지에 관한 예들이 주어진다. 마지막으로, 저작권 작품을 보호하기 위해 프로토콜을 어떻게 사용할 수 있는지를 설명한다.

[0169] 보안 프로토콜의 실시예는 디지털 비트스트림을 그 의도된 수령자에 의해서만 복호할 수 있게 하는 방법으로 디지털 비트스트림을 암호화하는 능력에 의존한다. 이것은 잘 이해되는 문제이고, 수많은 산업 표준 암호화 알고리즘의 기초가 된다. 그러나, 보안 프로토콜의 실시예와 함께 사용하도록 고려해야 하는 2가지의 추가적인 요소가 있다. 즉, 프로토콜의 핵심(core)이 전형적인 온칩 명령어 캐시(I-캐시)의 (비교적) 작은 한계에 맞추어질 수 있다면 도움이 된다는 사실과 반독립 방식으로 동작이 가능하다는 사실이다. 다시 말해서, 프로토콜이 작고 평소의 매일같은 동작(day-to-day operation)을 위해 중앙의 보안 서버의 사용을 요구하지 않는다면 유용하다.

[0170] **하드웨어:**

[0171] 이제 도 161로 돌아가서, 이 보안 프로토콜을 실행할 수 있는 장치의 예시적인 전체 블록도가 도시되어 있다. 보안 프로토콜 시스템의 예는 프로토콜을 프로토콜 엔진(100)("목표 유닛"이라고도 부름)에서 안전한 방식으로 구현하는 하드웨어 블록의 집합을 포함한다. 이 블록들은 프로토콜이 정확하게 동작하게 하기 위해 하드웨어에서 캐스트될 필요가 없지만, 뒤에서 설명하는 하드웨어 요소를 모두 포함하는 장치는 최소의 경상비(overhead)로 프로토콜을 구현할 수 있을 것이다.

[0172] 이러한 하드웨어 블록의 첫번째는 실시간 클럭(102)이다. 이것은 중앙 서버와의 안전한 상호 작용에 의해 세트 또는 리세트될 수 있는 자유 가동(free-running) 타이머이다. 비록 이것이 완전히 본질적인 블록은 아니지만, 안전 시간 표준의 질의를 행함으로써 시간이 확립될 수 있기 때문에 이 기능을 온칩으로 하는 것이 더 편리하다. 이것은 시간 의존성 소프트웨어 라이센스와 함께 행하여져야 하고, 그 예는 이 문서의 뒤의 섹션에서 주어질 것이다.

[0173] 다른 하나의 하드웨어 요소는 실행될 코드를 온칩으로 저장할 수 있는 메모리(110)의 블록이다. 이것은 전형적으로 명령어 캐시(I-캐시)라고 알려져 있고, 일부 실시예에 있어서, 상기 I-캐시(110)의 부분들의 중요한 특성은 특정 블록에 내포된 데이터가 CPU 실행 유닛(120)에 의해서만 판독될 수 있다는 것이다. 다시 말해서, I-캐시 메모리(130)의 상기 특수 블록은 실행만 가능하고 임의의 소프트웨어에 의해 판독 또는 기록될 수 없다. 이 I-캐시의 특수 섹션을 우리는 "보안 코드 블록"(130)이라고도 부른다. 실행 대상 코드가 이 보안 I-캐시 블록(130)에 실제로 저장되는 방법은 다른 하드웨어 요소에 의해서 수행될 수 있다.

[0174] 게다가, 보안 코드 블록의 동작을 가속화하기 위해 사용할 수 있는 가능한 "증대(enhancement)"의 다른 카테고리가 있다. 이 카테고리 중의 하나는 CPU(120)가 보안 코드를 실행하는 동안 액세스만 가능한 CPU 레지스터

(140)의 (부분)집합을 지정하는 능력이고 및/또는 이것은 보안 코드 블록의 실행 종료시 또는 어떤 이유로 실행 유닛이 비보안 즉 "정상" I-캐시에 위치된 임의의 코드 섹션으로 점프하면 클리어된다. CPU(120)가 "보안" 코드와 "비보안" 코드의 혼합물을 실행할 가능성은 없어 보이지만, 인터럽트 루틴으로 점프할 때 콘텍스트를 스위칭하는 과정에서 무슨 일이 발생할 수 있는지 및 CPU(120) 콘텍스트가 어디에 저장되는지는 항상 염두에 두어야 한다(대부분의 CPU는 콘텍스트를 메인 메모리에 저장하는데, 이 메모리는 나중에 비보안 코드 블록에 의해 발견될 가능성이 있다).

[0175] 다른 가능성(어떤 레지스터(140)가 클리어되어야 하는지를 보안 코드 블록의 창조자가 명백히 식별하는 것을 요구하는 것이 아닌 것)은 그것이 자동으로 행하여져야 한다는 것이다. 이것은 보안 코드 블록 내에서 실행하는 동안 어떤 레지스터(140)가 판독 또는 기록되는지를 CPU 실행 유닛(120)이 계속하여 추적하고, 그 다음에 "보안" 모드를 빠져나갈 때 상기 레지스터를 자동으로 클리어하는 것일 수 있다. 이것은 보안 코드가 그 자체 후에 신속히 "청소(clean-up)"되어 2종류의 코들 블록들 간에 공유되도록 허용된 데이터만이 본래대로 유지되게 한다. "자동" 처리는 "명시적" 절차보다 잠재적으로 더 안전할 수 있지만, 코드 창조자가 보안 코드 블록과 비보안 코드 블록 사이에서 정보를 공유하고자 하는 경우에 더욱 복잡하게 될 수 있다.

[0176] 보안 코드 세그멘트와 비보안 코드 세그멘트 사이에서 레지스터 저장 데이터의 "누설"을 취급하는 다른 잠재적인 방법은 CPU(120)가 보안 코드를 실행할 때만 사용되는 레지스터의 유일한 집합을 식별하는 것이다. 대형 범용 레지스터 집합(140)을 가진 일부 CPU 아키텍처에 있어서, 이것은 최초로 금지적으로 비싼 것으로 보인다. 그러나, 많은 현대의 CPU 설계에서 실시되는 레지스터 리네이밍 및 스코어보딩 메카니즘의 수정 버전을 이용함으로써 과도한 경상비를 요구하지 않고(즉, "보안" 레지스터의 물리적으로 다른 집합을 구현할 때 수반되는 실리콘 오버헤드없이) 동일한 효과가 달성될 수 있다. 만일 보안 코드 블록의 실행을 원자 동작(atomic action)으로 취급하면(즉, 중단할 수 없으면), 이러한 이슈는 다루기가 더 쉽지만, 이 편리성은 성능 및 잠재적인 전체 코드 복잡성을 가져올 수 있다. I-캐시의 "보안" 부분(130)은 CPU에 대하여 I-캐시의 "정상" 부분(150)과는 다른 데이터 경로를 반드시 요구하지 않는다는 점에 주목한다. 사실, 상기 둘은 완전히 동의어일 수 있다.

[0177] 일방향 해시 함수 블록(160)이 또한 도시되어 있다. 이 기능을 하드웨어로 구현할 필요없이 보안 프로토콜의 실시예를 실행할 수 있는 엔진을 구성하는 것이 가능하다. 그러나, 해싱 알고리즘의 특정 부분에 대한 하드웨어 가속기는 확실히 바람직한 특징이다. 이 기능 블록의 하드웨어와 소프트웨어 구현 간의 교환조건(tradeoff)은 뒤에서 설명한다.

[0178] 목표 유닛(100)의 다른 부분은 하드웨어 조력 복호 시스템(170)이고, 이것은 암호화 메시지에서 동작하는 목표 유닛(100)의 비밀키 및 공개/개인키(뒤에서 설명함)를 사용하여 이들을 실행가능 코드 블록으로 변환한다. 이 복호 시스템(170)은 여러 가지 방법으로 구현될 수 있다. 전체 프로토콜의 속도 및 보안성은 이 블록의 구성에 의존하고, 따라서 보안 시스템 갱신을 수용하도록 충분히 융통성이 있을 뿐만 아니라 시스템이 시간 임계 메시지의 실시간 복호를 수행하도록 충분히 고속이어야 한다.

[0179] 이 두 가지 제약을 염두에 두고 있으면 프로토콜에 있어서 정확히 어떤 암호화 알고리즘이 이 하드웨어 블록(170)에 대하여 사용되는지는 중요하지 않다. 최대의 융통성을 촉진하기 위하여, 실제 하드웨어는 비알고리즘적으로 특수한 방법으로 사용되도록 충분히 범용이라고 추정되지만, 이 메카니즘을 구현할 수 있는 많은 다른 수단이 있다.

[0180] 온칩 난수 발생기(180)가 또한 블록도에 점선으로 표시되어 있다. 이 블록은 선택적이다. 또한, 이 난수 발생기는 소프트웨어 기반 의사 난수 발생 시스템에 종자 값을 공급할 수 있는 충분히 무작위인 수의 열을 생성하는 적당한 오프칩 방법으로 교체될 수 있다. 이 의사 난수 발생기는 잠재적으로 하드웨어로 또는 "보안" 소프트웨어로 또한 구현될 수 있다. 물론, 소프트웨어 기반 시스템의 융통성과 하드웨어 구현 간의 동일 원리의 교환조건이 이 경우에 또한 적용될 수 있다. 그러나, 목표 장치(100)가 난수를 발생해야 하는 경우는 이 프로토콜에서 빈번하게 발생하지 않기 때문에, 이 특수한 기능이 하드웨어 가속화가 아닐지라도 전체 성능에 영향을 주지는 않을 것이다.

[0181] **비밀키:**

[0182] 각 프로토콜 엔진(100)은 온칩으로 저장된 2세트의 비밀키 상수(104)를 가질 수 있고, 그들의 값은 소프트웨어에 의해 판독될 수 없다. 이들 키의 첫번째(1차 비밀키)는 비밀키의 집합으로서 구성될 수 있고, 그들 중에서 하나만이 임의의 특정 시간에 판독될 수 있다. 만일 유닛의 "소유권"이 바뀌면(예를 들면, 프로토콜 엔진을 내포한 설비가 판매되거나 그 소유권이 다른 방식으로 이전되면), 현재 활성인 1차 비밀키는 "클리어"되거나 다른

값으로 덮어쓰기 될 수 있다. 이 값은 안전한 방법으로 유닛에 전달될 수도 있고, 또는 상기 제1 키가 클리어된 때에만 사용되는 방식으로 유닛에 이미 저장되어 있을 수도 있다. 사실, 이것은 유닛의 소유권이 변경되었을 때 또는 그러한 변경에 관한 어떤 다른 이유가 있는 경우(절충키(compromised key)와 같이) 새로운 1차 비밀키를 상기 특수 유닛에 발행하는 것과 등가이다. 이 1차 비밀키 값(또는 값들의 집합)이 저장되는 유일한 다른 장소는 허가 기관에 있는 중앙 서버이다.

[0183] 1차 비밀키는 중앙 서버 데이터베이스에 있는 특수한 목표 유닛(100)의 일련번호(106)와 관련될 수 있다. 일련번호(106)는 목표 유닛(100)의 어디에든 저장될 수 있고, 소프트웨어 액세스가능이며 1차 비밀키에 대한 다른 관계를 갖지 않는다. 유닛의 동작 태양에 대한 임의의 갱신(보안 시스템의 갱신 등)은 1차 비밀키를 이용하여 달성될 수 있다. 만일 이 키의 값이 목표 유닛(100) 및 허가 기관 이외의 다른 당사자에게 알려져 있지 않으면, 안전한 중앙 서버를 통한 링크를 수반하지 않은 임의의 보안 트랜잭션을 위하여 사용될 수 없다. 그러나, 이 1차 키의 보안성은 최고의 중요성을 갖기 때문에, 절대적으로 필요할 때만 사용되어야 한다. 따라서, 예를 들면, 허가 기관의 중앙 서버와 목표 유닛 간의 안전한 트랜잭션을 위한 통신 링크를 암호화하기 위해 사용되어서는 안된다. 이 링크는 현재 허용되는 표준 실시에 따라서 급히 발생되는 임의 키를 이용하는 표준 키 교환 프로토콜을 이용하여 안전하게 될 수 있다.

[0184] 2차 비밀키는 목표 유닛(100) 자체에만 알려질 수 있다(따라서 허가 기관에도 알려지지 않는다). 목표 유닛(100)의 CPU(120)는 1차 또는 2차 비밀키의 값에 액세스할 수 없으므로, 어떤 면에서 목표 유닛(100)은 그 자신의 비밀키(104)까지도 알지 못한다. 이 키들은 목표 유닛 CPU(120)의 보안 블록 내에 저장되어 사용될 뿐이다. 이들 두 가지 비밀키의 조합은 목표 유닛의 전체 보안성을 증대시킨다. 이들이 어떻게 사용되는 지는 뒤에서 설명한다.

[0185] 다른 키 집합은 임시 공개/개인키 시스템(비대칭 키 시스템 또는 PKI 시스템이라고도 알려져 있다)의 일부로서 동작할 수 있다. 이 쌍의 키들은 급하게 생성되고, 중앙 서버의 개입없이 유사한 유닛들 간의 보안 통신 링크를 확립하기 위해 사용될 수 있다. 이러한 시스템의 보안성이 등가 키 길이 대칭 키 암호화 시스템의 보안성보다 전형적으로 더 낮기 때문에, 이 키들은 전술한 비밀키 집합보다 사이즈가 더 커야 한다. 이 키들은 다른 무엇보다도 "재전송 공격"에 대하여 보호하기 위해 온칩 타이머 블록에서 제시되는 값과 함께 사용될 수 있다. 이 키들은 급하게 생성되기 때문에, 키들이 생성되는 방법은 일부 종류의 난수 발생 시스템(180)에 의존한다. 마지막으로, 상기 생성된 키들은 소위 "취약(weak)" 키의 부류에 포함되지 않도록 주의를 기울여야 한다. "취약"이라고 생각되는 특정 키 집합은 사용되는 특정 암호화 알고리즘에 의존한다.

[0186] **동작 세부:**

[0187] 보안 프로토콜의 실시예가 동작하는 방식은 몇 개의 별도의 처리, 즉 시스템 초기화, 보안 코드 발생 및 대량 분배, 보안 코드 로딩 및 실행, 키 리스트 데이터 구조 구성, 임시 라이센스 이전, 영구 라이센스 이전, 시스템 소유권 양도, 허가 취소 및 보안 시스템 갱신으로 나누어 질 수 있다. 상기 각 처리는 차례로 설명한다. 그러나, 이하에서 설명하는 예들은 설명의 간편성을 위하여 선택된 것이고, 이 프로토콜이 구현될 수 있는 가장 효과적인(유일한 것이 아님) 방법이라고 할 수는 없다.

[0188] **시스템 초기화**

[0189] 이것은 목표 유닛의 비밀키(104)가 소정의 초기값으로 설정되는 단계이다. 이 절차는 2개의 비밀키 중의 어느 하나에 대하여 몇 개의 위치 중의 하나에서 달성될 수 있지만, 논리적인 이유 때문에, 일련 번호 또는 비밀키가 변경될 가능성이 있는 조립 과정의 최종 단계이어야 한다. 유닛(100)의 일련 번호가 오프칩으로 저장되는 경우에, 이 절차는 최종 조립 시점에서 수행될 가능성이 가장 높다. 유닛의 일련 번호(106)가 온칩으로 저장되면, 칩 제조 공정의 최종 지점(즉, 칩이 패키지된 후)에서 이 절차를 실행하는 것이 일반적이고, 따라서 임의의 후반작업(postproduction) 또는 번인 폴아웃(burn-in fall out)은 비기능성 부분을 골라내는 기회를 갖는다. 이 방법으로 안전하게 유지해야 할 데이터 양을 최호화한다. 전체 프로토콜의 보안은 유닛의 비밀키(104)의 보안에 기초하므로, 초기화 절차는 물리적인 보안이 가능한 시점에서 수행되어야 한다.

[0190] 1차 비밀키는 2차 비밀키를 공급하기 위해 사용되는 절차와 다른 절차에서 초기화(또는 장치에 "번(burn)"됨)되어야 한다. 비록, 실제상으로, 이 2차 키는 소정 시점에서 공지되겠지만(2차 키가 제조 공정 중의 소정 시점에서 유닛에 프로그램되기 때문에) 관련된 유닛은 2차 키가 목표 장치(100)에 저장되었으면 어디에도 기록되어서는 안된다. 회계 감사의 목적으로, 2차 비밀키 값의 전체 집합에 대하여 어느 부분이 어떤 키를 보유하고 있는지를 아는 것과 무관하게 시험되는 것이 바람직하다(분배의 무작위성을 테스트하기 위해 또는 어떤 다른 이유

로). 그러나, 시스템의 보안성을 유지하기 위해, 상기 2차 비밀키를 유닛에 프로그램하는 장치는 2차 비밀키를 제1 비밀키에 또는 목표 장치 일련 번호(106)에 관련시키는 어떠한 수단도 갖지 않는 것이 바람직하다. 또한, 상기 두 가지 비밀키는 뒤에서 설명하는 이유 때문에 변경 방지(tamper-proof) 방식으로 구현되어야 한다. 상기 2개의 비밀키가 어떤 순서로 초기화되는지는 문제가 되지 않는다. 예시적인 실시예에서 설명한 초기화 절차를 따르면, 목표 장치의 일련 번호(106) 및 그 관련 1차 비밀키가 공동으로 위치되는 (실제 칩이 아닌) 유일한 위치는 허가 기관의 보안 서버이어야 한다.

[0191] **보안 코드 발생 및 대량 분배**

[0192] 도 165를 참조하면, 예시적인 실시예에 있어서, 개발자(520)가 이 프로토콜 하에서 동작하는 애플리케이션을 생성하고자 한다고 가정하자. 상기 애플리케이션은 분해(disassembly)로부터 합리적으로 면역성을 갖는 것이고 특수한 장치에서만 실행될 수 있는 것이다. 각 등록된 개발자(520)는 허가 기관의 서버와 통신하기 위해, 및 임의의 공개된 코드 블록 또는 다른 비트스트림을 인증하기 위해 사용될 수 있는 부호화 메시지 인증 코드(MAC)(전형적으로 디지털 서명이라고 함)를 생성하기 위해 개발자가 사용하는 임의의 메시지를 인증하기 위해 사용되는 공개키/개인키 쌍을 갖고 있을 것이다.

[0193] 애플리케이션이 디버그(debug)된 후에, 애플리케이션은 최초 개발자에게만 알려져 있는 애플리케이션 지정 암호화 알고리즘 및 키를 이용하여 인코드된다. 상기 애플리케이션 지정 알고리즘 및 키는 대칭(비밀) 키 시스템 또는 비대칭(PKI) 키 기반 시스템일 수 있다. 암호화 코드 블록의 끝에는 MAC가 첨부되고, 이것은 개발자(520)에 의해 그들의 공개된 공개키/개인키 쌍의 개인키를 이용하여 부호화된다(따라서 암호화 코드 블록에 대하여 불명료하지 않은 디지털 서명을 형성한다). 디지털 서명 또는 최초 MAC와 대응하는 코드 지정 ID 번호는 허가 기관에 제공될 수 있다. 애플리케이션 개발자(520)는 또한 적당한 디코딩 키를 제공하도록 선택할 수 있다(이 결정의 교환조건에 대해서는 이 문서의 뒷부분에서 설명한다).

[0194] 만일 애플리케이션 지정 알고리즘이 비대칭 암호화 시스템이면, 부호화 메시지 인증 코드(디지털 서명)를 생성하기 위해 사용한 동일한 공개 PKI 키 쌍을 사용하여 암호화할 필요가 없다. 그러나, 코드 블록의 끝에 저장된 MAC는 공지된 해싱 알고리즘을 이용하여 생성되어야 하고 또한 개발자의 공개된 공개키 중의 하나를 이용하여 부호화되어야 한다(따라서 디지털 서명을 형성한다). 이것은 목표가 공지의 해싱 함수 및 공지의 공개키를 이용하여 MAC의 인증을 검증할 수 있게 한다.

[0195] 다시 도 162로 되돌아가서, 모든 애플리케이션 지정 암호화 키 데이터 구조(210)는 다수의 여분의 필드(복호키 자체(220)에 추가해서)를 포함할 수 있다. 이 필드 중의 하나는 타임스탬프(230) 및 관련 마스크 값(240)을 포함할 수 있다. 다른 필드는 "카운트다운 값"(250)을 포함할 수 있다. 마스크 값(240)은 다른 2개의 필드(230, 250)와 함께 사용되어 키가 유효로 된 때를 결정한다. 정확히 얼마나 많은 비트가 각 필드에 할당되었는지는 프로토콜과 관계가 없다는 점에 또한 주목하여야 한다.

[0196] 타임스탬프 값(230)은 타임스탬프 마스크(240) 필드에 저장된 비트 패턴에 따라서 몇 가지 방법으로 사용될 수 있다는 점에 주목한다. 타임스탬프 마스크(240) 값은 목표 유닛(100)의 현재 시간과 비교를 수행할 때 무시된 타임스탬프 도형(figure)의 일부 부분집합을 개발자(520)가 선택할 수 있게 한다. 그러나, 예로서, 타임스탬프 필드(230)에 의해 지원된 최소 해상도(resolution)가 1초라고 가정하면, 타임스탬프 데이터(230)의 하위 5비트를 제거함으로써 타임스탬프 필드(230)에 저장된 때로부터 시작하여 약 32초의 과정에서 사용될 때만 유효로 되는 특수한 키 데이터 구조(210)가 발생될 수 있다. 보안 프로토콜의 전체 기능은 타임스탬프 필드(230)의 최하위 순서 비트의 실제 해상도에 의존하지 않는다.

[0197] 마스크 필드(240)와 관련된 다른 비트들이 있을 수 있고, 그 중 일부는 값이 타임스탬프(230)에서 지정되기 전에 키가 유효화되었는지 후에 유효화되었는지를 표시하기 위해 사용될 수 있다. 또다른 마스크 필드(240) 비트는 타임스탬프(230)와 "카운트다운" 값(250)이 어떻게 관련되는지를 표시하기 위해 사용될 수 있다. 예를 들면, 이것은 애플리케이션 개발자(520)의 의도가 특정 데이터 및 시간창(time window)에 단순히 결합하기 보다는, 특정 날짜 이전에 또는 후에 특정의 반복 횟수로 소프트웨어의 사용을 제한하는 것인 경우에 유용할 것이다. 물론, 이들 조건의 임의 조합이 구성될 수 있고, 따라서 프로토콜은 이 점에서 매우 융통성있는 것이다. 또한, 얼마나 많은 합법적 키 복사본이 최초 목표 유닛(100)으로부터 다른 유닛으로 동시에 분배될 수 있는지와 같은 다른 특성들을 표시하기 위한 추가의 플래그가 이 데이터 구조에 포함될 수 있다. 이것은 예를 들면 디지털 도서관에서 볼 수 있는 것처럼 다중 복사 허가가 필요한 경우에 유용할 것이다.

[0198] 암호화 처리의 일 실시예를 나타내는 흐름도가 도 163에 도시되어 있다. 디지털 미디어 스트림 또는 소프트웨어

애플리케이션(미디어 스트림을 해석하기 위해 사용된 복호 명령어 등)을 분배하기 위해 사용된 처리들 간에 실질적인 차이는 없다는 점에 주목한다. 어느 경우이든, 온라인 서버를 통하여 또는 직렬화 디스크(표준 DVD 등)를 이용하여 암호화 코드 블록(310, 320)을 분배하기 위한 몇 가지 다른 옵션들이 있다. 후자의 경우에, 개발자(520)는 대량 생산된 디스크의 개별 일련 번호를 허가 기관(510)에 미리 등록하도록(또는 등록하지 않도록) 선택할 수 있다. 만일 등록하면, 일련 번호는 버스트 커팅 영역에 버닝(burning)함으로써(DVD의 경우) 또는 잉크젯 인쇄함으로써(표준 CD의 경우) 디스크에 영구적으로 첨부될 수 있다. 동일한 일련 번호가 대량 생산된 모든 디스크에 복제될 것이기 때문에, 개발자(520)는 일련 번호를 CD 또는 DVD의 데이터 영역에 매립(embed)할 수 없다는 점에 주목한다. 디스크의 일부가 대량 생산되고 다른 부분에 1회 기록되는 소정의 하이브리드 포맷이 사용되는 경우, 이것은 개별 일련 번호와 함께 디스크를 분배하는 다른 잠재적 방법이다. 어느 경우이든, 등록 과정에서 에러가 발생할 가능성이 적기 때문에 기계 판독가능 일련 번호가 확실히 바람직하다.

[0199] 만일 개발자(520)가 미디어 일련 번호를 허가 기관에 등록하지 않는 것을 선택하면, 적당한 암호화 키를 애플리케이션 또는 미디어 스트림 필드에 관련시킬 수 있는 소정의 다른 방법이 있다. 따라서, 애플리케이션 개발자(520)는 코드 지정 ID 또는 관련 미디어 일련 번호를 등록할 수 있다. 전자의 경우에, 애플리케이션은 자유롭게 분배될 수 있다(즉, 특수한 해제 포맷 및 미디어에 결합되지 않는다).

[0200] 개별 일련 번호 메카니즘의 경우에는 허가 기관(510)이 어떤 애플리케이션(또는 미디어 스트림)이 어떤 일련 번호와 관련되는지 알 필요가 없기 때문에(잠재적으로 표시가 없음), 최종 사용자의 프라이버시가 유지된다. 개발자(520)가 애플리케이션 ID(또는 미디어 스트림 ID)를 그 관련 키와 함께 등록하는 경우에, 허가 기관(510)은 어떤 애플리케이션 또는 미디어 스트림이 특수한 최종 사용자에 의해 "소유"되었다는 것을 알 수 있다. 반면에, 이 잠재적 프라이버시 결핍은 개발자(520)가 물리적 매체를 제조하여 분배할 것을 요구하지 않는 추가적인 편리성 및 비용 절감으로 상쇄된다. 용어 "물리적 매체"는 반드시 디스크를 의미하는 것이 아님에 주목한다. 이 기능은 매체에 첨부되는 개별 일련 번호 스티커와 함께 인쇄된 매뉴얼(또는 단순한 등록 양식)을 이용함으로써 달성될 수 있다. 필요한 것은 개발자(520)가 최종 사용자에게 공급되는 유일한 일련 번호를 가진 어떤 물체(object)를 생성해야 하는 것뿐이다. 이 일련 번호의 목적은 비트스트림 등록 번호로 사용하기 위한 것이다. 이 일련 번호가 프로토콜에서 어떻게 사용되는지는 다음 섹션에서 설명한다.

[0201] 도 163에 도시한 예에서, 암호화 소프트웨어 애플리케이션(또는 미디어 스트림)(310)과 머신 의존 복호 소프트웨어(330)는 둘 다 동일한 메카니즘을 이용하여 분배된다. 이것이 그 경우이고 암호화 코드 블록(310, 330)의 어느 하나 또는 둘 다가 온라인으로 또는 디스크를 인쇄함으로써 분배될 수 있다는 것은 프로토콜의 필요조건이 아니다. 그러나, 디지털 미디어 스트림의 경우에, 미디어 스트림 자체는 몇 개의 크기 등급만큼 2개의 블록(310, 330) 중에서 더 큰 것일 가능성이 높다는 것에 주목하여여 한다. 따라서, 그 경우에, 적어도 이 블록의 분배를 대량 생산 디스크 포맷으로 시행하는 것이 가장 사리에 맞는 일이다. 많은 경우에, 동반자 암호화 코드 블록(제1 블록을 디코드하는 방법에 대한 명령어를 포함하고 있는 것)뿐만 아니라 1차 암호화 코드 블록을 맞추기 위한 룸이 디스크에 충분히 있을 수 있다. 2개의 데이터 집합의 어느 것도 공개 후에 변경될 가능성은 없고, 따라서 그들이 온라인으로 분배되어야 한다는 기본적인 필요조건도 없다는 점에 또한 주목하여야 한다. 그래서 이들은 대량 생산 디스크 기반 분배 메카니즘에 적합하다. 2개의 코드 블록을 동일한 디스크에 배치하면 명백한 방식으로 하나를 다른 하나에 관련시키는 것이 더 쉬워진다.

[0202] **보안 코드 로딩 및 실행**

[0203] 분배 메카니즘이 실제 디스크를 통해 달성되는 경우에, 소비자는 전통적인 소프트웨어 구매와 정확히 동일한 방식으로 애플리케이션을 내포한 디스크를 구매할 수 있다. 물론, 최종 사용자는 "목표" 유닛의 프로세서에서 수정되지 않은 암호화 코드 블록을 동작시키지 못할 수 있다. 사용자가 애플리케이션을 그들의 머신에서 동작시키려고 시도할 때, CPU(120)는 암호화 소프트웨어 블록을 로드하고, 소프트웨어 개발자의 공개키와 함께 코드 블록의 끝에 저장된 디지털 서명("부호화" MAC)을 이용하여 당해 코드 블록이 진정한 것인지 검증한다. 이것은 다른 범용 CPU(120)에 대한 제1 하드웨어 수정이 동작(play)으로 되는 경우이다. 상기 보안 코드의 블록을 로드하고 복호하는 처리는 도 164에 도시하였다.

[0204] 해싱 함수가 정확하게 계산되는 것(및 더 나아가 일반화 메시지 다이제스트와 "실제" 메시지 다이제스트의 비교가 유효인 것)을 보증하기 위해, CPU(120)는 이 해싱 함수를 보안 방식으로 수행하여야 한다. 따라서, 해싱 함수는 디코더 유닛의 하드웨어에 의해 직접 발생되거나 해싱 함수 자체가 "보안" 코드 블록을 이용하여 계산되어야 하고, 그 동작은 다른 "비보안" 프로그램에 의해 부정하게 변경될 수 없다.

[0205] 소프트웨어 기반 해시의 경우에, 이 보안 코드 블록은 유닛(100)의 보안 시스템의 일부로서 고려되어야 하고,

그래서 유닛(100)과 허가 기관(510) 간의 안전한 트랜잭션을 통해 플레이어에 다운로드될 수 있다. 충분히 흥미롭게, "보안" 해싱 함수의 확립은 여기에서 설명한 동일한 보안 프로토콜을 통해 달성될 수 있다. 보안 시스템의 모든 태양에 대한 이러한 회귀적 행동은 이 프로토콜의 소프트웨어 기반 버전이 그 암호화/복호 아키텍처에서 극히 유연하게(따라서 갱신가능하게) 할 수 있다.

[0206] 메시지 다이제스트 계산이 하드웨어로 고정되면, 우리는 잠재적으로 어느 정도의 보안성을 얻을 수 있지만, 이것은 융통성의 희생을 요구한다. 전용 하드웨어 블록을 이용하여 해시 값을 생성하고, 그 다음에 칩이 제조된 후의 소정 시점에서 해싱 알고리즘의 어떤 약점이 발견되면(또는 그 구현시에 어떤 버그가 있으면), 그 사실을 안 후에 문제를 처리할 기회가 없다. 이것은 처리를 가속화하기 위해 소프트웨어 기반 해싱 함수(프로그램가능 S-박스 구조 등)에 대한 어떤 종류의 하드웨어 가속화를 사용할 수 없다는 것을 말하는 것이 아니다. 그러나, 그 경우에, 하드웨어는 이상적으로 매우 다양한 일방향 해싱 함수를 지원하기 위해 충분히 범용이어야 한다.

[0207] 그러나, 이 프로토콜의 보안은 궁극적으로 이 보안 코드 로딩 절차의 일부로서 제공된 최저 수준 기능에 의존한다는 것에 주목하여야 한다. 저수준 특징(해싱 함수에서 사용된 비밀키 또는 원시 연산 등)은 다른 방법으로 함께 결합되어 부호화 메시지 다이제스트와 같은 고수준 기능을 생성한다. 다음에, 상기 고수준 기능 블록을 사용하여 아이덴티티 검증과 같은 고수준 유틸리티를 제공한다. 보다 원시층의 위에 고수준 기능을 구축하는 이러한 처리는 "신뢰 사슬(Chain of Trust)" 구축이라고 알려져 있다. 시스템의 융통성은 보안 관련 기능이 이 계층구조 내에서 가능한 한 낮게 수정될 수 있는 지점의 배치에 있다. 그러나, 소정 지점에서, 상기 사슬이 기초하는 기본적인 원시 연산은 본질적으로 원자이어야 한다(즉, 이것은 하드웨어에서 구현되어야 하는 기능의 최저 수준이다). 이 하드웨어 입상 지점의 정확한 선택은 대부분 구현예의 세부이고, 이 프로토콜의 전체 동작은 위에서 주어진 조건의 이러한 태양에 의존하지 않는다.

[0208] 암호화 코드 블록(310)이 목표의 메모리 스페이스(110)에 로드되고 메시지 다이제스트가 계산되면, 그 결과는 개발자의 공개키를 가진 암호화 코드(310)와 함께 저장된 디지털 서명(340)을 복호함으로써 계산된 메시지 다이제스트와 비교된다. 2개가 일치하면 목표 유닛(100)은 암호화 코드 블록(310)이 진정한 것(또는 적어도 코드가 디지털 서명을 복호하기 위해 자신의 공개키를 사용한 개발자(520)에 의해 분배된 것)임을 확신할 수 있다.

[0209] 이 시점에서, 목표(100)는 복호키의 복사본을 요구하는 허가 기관(510)에 보안 메시지를 전송하고, 보안 메시지는 최근에 검증한 암호화 코드 블록과 제휴하여 사용될 것이다. 허가 기관과의 보안 접속을 설정하는 일환으로서, 목표 유닛(100)은 임시 공개키/개인키 쌍을 발생한다(그 중 공개 부분은 허가 기관(510) 서버에 제공된다). 키 교환 절차의 세부는 잘 알려져 있으므로 이것을 달성하는 정확한 메카니즘까지 여기에서 설명할 필요는 없을 것이다. 어느 경우이든 목표 유닛(100)과 허가 기관(510)의 중앙 서버 간의 전체 네트워크 트래픽은 상당히 작은 데이터 집합으로 제한되는데, 그 이유는 데이터 집합이 함께 저장된 몇 개의 키 전송, 코드 지정 ID 및 MAC로 구성되기 때문이다.

[0210] 코드 지정 ID(260)가 허가 기관(510)에서 인식하고 있는 것이라고 가정하면, 애플리케이션 창조자가 피요청 복호키의 "명문(clear)" 복사본을 이미 허가 기관(510)에 제공하였는지 여부에 따라서 2가지의 가능한 동작 과정이 있을 수 있다. 개발자(520)가 허가 기관(510)에 그러한 정보를 제공하지 않은 경우에는 중앙 서버가 목표 장치의 임시 공개키의 복사본(및 당해 코드 지정 ID(260))을 애플리케이션 개발자의 서버에 전송한다. 그 시점에서, 개발자의 서버는 피요청 복호키(목표 장치의 임시 공개키로 암호화한 것)를 내포한 메시지 및 적절히 복호된 코드로부터 생성한 메시지 다이제스트로 허가 기관(510)에 응답한다. 이러한 방법으로, 목표 장치(100)는 메시지를 복호하여 애플리케이션 지정 복호키를 얻을 수 있고, 허가 기관(510)은 명문 형식의 복호키에 액세스하지 않을 것이다.

[0211] 비록 메시지 다이제스트가 미리 계산되어 허가 기관 서버에 저장되지만, 메시지 다이제스트가 트랜잭션 중에 개발자(520)에 의해 제공될 수 있다는 사실은 해싱 함수(메시지 다이제스트를 생성하기 위해 사용한 것)가 변경되어야 하는 경우에 잠재적으로 사용된다. 이러한 일이 발생하면, 개발자(520)는 복호된 코드 메시지 다이제스트의 갱신된 버전을 목표 장치(100)와의 실제 트랜잭션 전에 또는 트랜잭션 중에 허가 기관(510)에 제공할 필요가 있다. 허가 기관(510)이 최초의 (복호된) 코드에 액세스하지 않기 때문에 개발자(520)는 이 정보를 제공해야 한다. 앞에서처럼, 허가 기관 서버와 개발자 서버 간의 네트워크 트래픽 양은 여전히 매우 작다. 개발자(520)로부터 수신한 암호화 키는 허가 기관(510)으로부터 목표 장치로 전송되기 전에 목표 장치의 1차 비밀키를 이용하여 다시 암호화된다. 이 제2 암호화는 다른 트랜잭션의 일부로서 개발자측에서 실제로 실행될 수 있지만, 그 경우 개발자는 잠재적으로 목표 장치의 1차 키에 액세스하는데(양측의 키가 어떤식으로든 결합되어 있지 않는 한), 이것은 최종 사용자에 대하여 프라이버시 문제의 잠재적 손실을 가져온다.

[0212] 애플리케이션 개발자(520)가 허가 기관(510)과 목표 장치(100) 간의 트랜잭션을 위하여 "루프외(out of the loop)"에 머무르기를 원하는 경우, 명문(암호화되지 않은) 형식의 관련 복호키의 복사본 및 복호된 코드 블록의 관련 MAC(그 값은 해싱 알고리즘이 변경된 때마다 갱신되어야 한다)를 허가 기관(510)에 간단히 제공할 수 있다. 따라서, 허가 기관(510)의 중앙 서버는 독립적으로 동작할 수 있고 목표 장치(100)로부터의 키 요청을 충족시키기 위하여 개발자 서버에 대한 통신 링크를 확립할 필요가 없다. 그러나, 이것은 상기 "명문 키" 정보가 허가 기관에 의해 의도적으로 또는 비의도적으로 절충되는 경우 개발자에게 잠재적인 보안 위험을 야기한다.

[0213] 전체적인 키 암호화/복호 처리의 흐름도는 도 165에 도시되어 있다. 이 경우에, 명문 키는 (위에서처럼) 전송 전에 목표 장치의 임시 공개키 및 다시 목표 장치의 1차 비밀키를 이용하여 암호화된다. 이 시점에서, 목표 장치(100)는 두번 암호화된 포맷으로 적당한 복호키를 갖는다. 허가 기관(510)이 명문의 애플리케이션 지정 키(550) 정보에 액세스하지 않는 경우에, 의도된 목표 장치(100)가 아닌 누군가가 이 키 데이터를 명문 형식으로 재생산하는 것은 불가능하다. 왜냐하면, 각 유닛(100)의 비밀키는 허가 기관(510)에만 알려져 있고, 전송을 위한 개인키는 목표 장치(100)에 의해서만 공지되기 때문이다.

[0214] 그러나, 이 시점에서, 목표 장치(100)가 애플리케이션 개발자(520)로부터 수신한 인코드된 복호키는 목표 장치(100)의 개방시에 안전하게 저장될 수 없다(예를 들면, 플래시 ROM에 또는 하드드라이브에 백업됨). 목표 장치(100)는 허가 기관(510)으로부터 전송된 인코드된 복호키와 함께 임시 비밀키의 복사본을 또한 저장해야 한다는 점에 문제점이 있다. 만일 허가 기관(510)의 누군가가 어떤 수단에 의해 상기 2편의 데이터에 액세스하였으면, 그들은 잠재적으로 복호 애플리케이션 지정 키(550)를 재구성할 수 있다(그들은 목표 장치(100)의 1차 비밀키에도 또한 액세스할 것이다).

[0215] 이것은 목표 장치의 2차 비밀키가 사용으로 되는 지점이다. 이 2차 비밀키는 목표 장치의 복호 유닛 외의 누구에게도 알려져 있지 않다는 것을 상기하자. 따라서, 키를 복호하기 위해 허가 기관으로부터 목표 장치(100)에 공급된 임시 개인키를 사용하였으면, 2차 비밀키가 애플리케이션 지정 키를 그 사용(및/또는 보관) 전에 재암호화하기 위해 사용된다.

[0216] 그 다음에, 목표 장치는 애플리케이션 지정 (명문) 키(550)를 이용하여 코드 블록(또는 미디어 스트림)을 복호한다. 따라서, 애플리케이션 코드가 명문 형태로 존재하는 단지 2개의 장소가 최초 개발자(520) 자체에 및 목표 장치의 I-캐시(110)의 "보안" 부분 내측에 있다(이곳에서 애플리케이션 코드가 실행될 수 있고 메모리에 명문 형태로 다시 기록되지 않는다). 이것은 사용자와 허가 기관(510) 간의 프라이버시를 허용한다. 다시 말해서, 허가 기관(510)은 사용자가 허가받은 것이 무엇인지 알지 못하지만(거대한 프라이버시 장점), 목표 장치(100)가 손상되거나 분실되거나 또는 다른 방법으로 동작불능으로 된 경우에 사용자 키 리스트에 대한 보관소(또는 백업)으로서 여전히 동작할 수 있다.

[0217] 복호 처리가 정확하게 수행되었는지 검증하기 위한 체크로서, 적절히 복호된 코드의 메시지 다이제스트는 그 다음에, 최초 개발자(520)로부터 허가 기관(510)을 통하여 목표 장치(100)에 전송된, 디지털 서명을 복호함으로써 생성된 메시지 다이제스트와 비교된다. 위에서 언급한 바와 같이, 이 디지털 서명은 애플리케이션 개발자의 개인키로 비암호화 코드 블록의 메시지 다이제스트를 암호화함으로써 생성된다. 대안적으로, 이 디지털 서명은 접속이 확립되었을 때 허가 기관(510)에 공급하였던 다른 임시 공개키(530)를 이용하여 개발자(520)에 의해 다시 암호화될 수 있다. 어쨌든, 올바른 메시지 다이제스트는 그 다음에 개발자의 공개키로 디지털 서명을 복호함으로써 목표 장치(100)에 의해 디코드될 수 있다. 만일 이 메시지 다이제스트가 복호 코드 블록의 MAC와 일치하면, 코드는 진정한 것으로 고려되고 목표 장치(100)에서의 동작이 허용된다. 이 메시지 다이제스트는 재암호화된 애플리케이션 지정키(550)와 함께 보관하기 위하여 목표 장치의 2차 키(540)로 재암호화될 수 있다.

[0218] 이 절차에서의 마지막 단계는 애플리케이션 지정키(560)의 (목표 장치의 2차 키(540)로) 새로 암호화된 버전이 보관 목적으로 허가 기관(510) 서버에 다시 재전송된다. 이 전송 서버는 몇 가지 목적으로 사용된다. 첫째, 목표 장치(100)가 코드 블록을 적절히 복호할 수 있었음을 통지하는 것이다. 둘째, 최종 사용자가 어떤 종류의 파국적인 데이터 장애를 받고 그들 자신의 액세스 키에 대한 백업 카피를 만드는 것을 게을리한 경우를 취급하기 위해 허가 기관(510)이 상기 암호화 키(560)의 복사본을 가질 필요가 있다. 그 다음에, 허가 기관(510)은 임의의 특정 사용자에 대해 백업 스토리지 설비로서 동작할 수 있다. 상기 절차의 또다른 이유는 특수한 목표 장치(100)가 하나의 사용자로부터 다른 사용자로 그 소유권을 변경한 경우 또는 사용자가 자신의 목표 장치(100)를 업그레이드하고자 하는 경우를 취급하기 위해서이다. 이러한 종류의 소유권 영구 양도는 그 유닛(100)에 대한 모든 허가된 애플리케이션 키의 양도를 수반할 수 있다(이 경우에, 그 유닛을 새로운 소유자 이름으로 재등록하는 것 외에 할 것은 아무것도 없다). 그러나, 사용자가 자신의 키 데이터의 영구적인 소유권을 제1 장치로부터

제2 장치로 이전하기 원하면, 이것은 허가 기관(510)과 2개의 목표 장치 간의 보안 트랜잭션에 의해 달성될 수 있다.

[0219] 목표 장치(100)가 허가 기관(510) 서버에 다시 전송하는 다른 정보는 목표 장치의 새로 갱신된 키 리스트 데이터 구조(610)(도 166에 도시됨)의 메시지 다이제스트이다. 이것은 새로 갱신된 키 리스트 데이터 구조(610)의 승인(acknowledgement)이고, 허가 기관(510) 서버 및 목표 장치(100)에서 상기 특수한 목표 장치(100)와 관련된 키 리스트 데이터 구조(610)의 동일성을 검증하기 위해 또한 사용된다. 이 데이터 구조의 정확한 구성은 다음 섹션에서 설명한다. 우리는 또한 특수 키 또는 키 집합의 소유권의 영구 양도를 달성하는 방법에 대하여 뒤의 섹션에서 설명할 것이다.

[0220] 이 시점에서, 위에서 설명한 프로세스는 애플리케이션 지정키(550)를 개발자(520)로부터 목표 장치(100)로 이전하기 위해 프로토콜을 사용하는 유일한 방법이 아니라는 점에 주목하여야 한다. 예를 들면, 실제 키 전송 트랜잭션은 목표 장치(100)와 애플리케이션 개발자(520) 사이에서만의 직접 접속을 수반할 수 있다. 그러나, 이 경우에 장치 지정 암호화 정보를 트랜잭션에 제공하기 위해 개발자 서버와 허가 기관 서버 간에 접속이 확립되어야 한다. 보안 방식으로 작업하기 위해 이 프로토콜이 만들어질 수 있는 다수의 메카니즘이 있고, 위에서 설명한 예는 이 메카니즘들 중의 하나일 뿐이다. 그러나, 공통의 스레드(thread)는 목표 장치(100)에 전송된 키 데이터가 그 목표 장치(100)에 대해서만 사용할 수 있다는 것을 보증하기 위해 3개의 당사자 모두가 함께 작용해야 한다는 것이다.

[0221] 키의 구조는 2개의 부분, 즉 하드웨어 지정 부분과 애플리케이션 지정 부분을 갖도록 설정될 수 있다는 것에 주목한다. 상기 2개의 부분이 완전히 불가분성인 것은 필요조건이 아니다. 만일 이들이 불가분성이면, 우리는 위에서 설명한 특성들을 정확히 취하게 된다. 그러나, 키 부분들이 독립적으로 동작하게 하는 방법이 있으면, 우리는 총체적인 복사본 집합을 얻을 수 있고, 실제 코드 또는 실제 목표 장치(100)와 무관한 제약들을 사용할 수 있다. 다시 말해서, 임의의 개발자(520)는 분배에 제약이 없지만 판독할 수 없고 실행만 가능한 애플리케이션 또는 미디어 스트림을 공개할 수 있다. 이것은 허가 기관(510)이 제조자에 관계없이 모든 장치에서 동작하는 보안 시스템 갱신을 송출하기 원하는 경우에 유용할 수 있다. 이것의 다른 예는 스트림의 저작권에 대한 제어를 여전히 유지하면서 공개적으로 이용가능한 미디어 스트림의 방송일 것이다. 유사하게, 공개자는 누군가가 판독 및/또는 복사할 수 있지만 하나의 특수한 목표 장치(100) 또는 장치들의 집합에서 실행만 가능한 애플리케이션을 분배할 수 있다. 이것은 예를 들면 "이 특수 종류의 장치 갱신" 메시지를 송출할 때 유용하다. 다른 가능한 응용은 어디에서든 동작할 수 있고 분배에 제약이 없는 애플리케이션을 송출하는 것이다. 이것은 특수한 응용(즉, 개방 소스 분배)을 위해 소스 코드를 공개하는 것과 본질적으로 유사하다. 분리가능한 H/W 지정 및 S/W 지정 키 구조에 의해 가능하게 되는 다른 종류의 보안은 표 1에 도시하였다.

[0222] [표 1]

소프트웨어 또는 애플리케이션 지정 키 세그멘트

하드웨어 지정 키 세그멘트		
*"잠김" **	제한된 동작 및 제한된 분배	제한없는 분배이지만 지정된 유닛에서만 실행가능함(예를 들면, 특정 유닛에 목표된 코드)
*"풀림" ***	제한없는 분배이지만 실행만 가능함. 즉 코드는 "판독가능"이 아님	동작 또는 분배에 제약이 없음(즉, 공개 및 개방임)

* 즉, 특수 일련번호 (또는 소정 범위의 번호)로 잠금됨

** 즉, 어디에서든 동작가능함

[0223]

[0224] <분리 가능한 하드웨어 지정 및 애플리케이션 지정 키 구조>

[0225] **키 리스트 데이터 구조 구성**

[0226] 이제 도 166을 살펴보면, 특수 목표 장치(100)에 허가된 애플리케이션 또는 미디어 지정 키를 내포한 데이터 구조(610)는 가치있는 상품(commodity)이고, 그래서 소유자에 의해 백업될 수 있어야 한다. 개별 키들은 목표 장치의 2차 비밀키로 (위에서 설명한 것처럼) 암호화되기 때문에, 그 리스트는 키들이 허가된 유닛에만 유용하다. 그러나, 우리는 이 데이터 구조(610)가 부정한 변경, 훼손 및/또는 완전한 손실로부터 안전하다는 것을 확실하게 할 필요가 있다. 손실된 키 리스트 데이터 구조의 경우에, 전체 데이터 구조(610)는 그 특수 목표 장치(100)에 대한 키 리스트의 새로운 복사본을 위에서 설명한 것처럼 허가 기관(510)으로부터 요청함으로써 복구될 수 있다. 키 리스트 데이터 구조에 대하여 임의 변경이 이루어진 경우에(그러한 시나리오의 이유에 대해서는 이 섹션에 이어지는 섹션에서 설명할 것이다), 프로토콜은 그러한 변경을 임시인 것으로 식별하는 수단을 수용할 수 있다. 마지막으로, 우리는 키 리스트 데이터 구조(610)의 진정성, 적시성 및 유효성을 유효화하기 위한 소정의 위조 방지 메카니즘을 포함한다.

[0227] 이러한 필요조건을 염두에 두고서, 도 166에 도시한 것과 유사한 방법으로 상기의 모든 품질을 나타내는 안전한 키 리스트 데이터 구조(610)를 구성할 수 있다. 평소와 같이, 도시된 예는 모든 소망하는 특성들을 그러한 데이터 구조에 포함시킬 수 있는 유일한 방법이 아니다. 그럼에도 불구하고, 도 166에 도시한 특수한 데이터 구조는 사실상 프로토콜의 모든 기본적인 필요조건을 충족시킨다.

[0228] 위의 도표에서 주목해야 할 몇 가지 법칙이 있다. 첫번째는 키 리스트 데이터 구조(610)의 최상위 암호화는 목표 장치의 1차 비밀키로 수행되어야 한다는 것이다. 이러한 특수한 키를 이용하는 데는 몇 가지 이유가 있지만, 그 중 중요한 것은 데이터 구조의 국부적 복사본(local copy)이 복원되어야 하는 경우에 허가 기관(510)이 이 데이터 구조의 암호화 형태를 목표 장치(100)와 무관하게 재생성할 수 있어야 한다는 것이다. 이 데이터 구조를 암호화하기 위해 임의의 다른 키(예를 들면, 목표 장치의 2차 비밀키)가 사용되면, 목표 장치가 데이터 구조를 변경할 필요가 있을 때(키가 리스트에 추가되는 경우처럼) 전체 리스트가 백업 목적으로 허가 기관(510)에 전송되어야 한다. 이것은 허가 기관(510)에 다시 전송해야 할 네트워크 트래픽 양을 잠재적으로 크게 증가시킬 수 있으며, 이것은 반드시 채널 대역폭의 가장 효율적인 사용이라고 볼 수 없다.

[0229] 또한, 이 키 리스트 데이터 구조(610)는 표준 애플리케이션 또는 미디어 스트림 지정 허가 키의 저장용으로 사용되는 것 외에 보안 시스템 관련 키의 저장용으로 사용되는 것이 바람직하다. 이 데이터 구조는 허가 기관(510)에 의해 재생성될 수 있기 때문에, 목표 장치에서 동작하는 보안 소프트웨어를 갱신하는 것이 바람직한 경우에, 이것은 동일한 키 리스트 데이터 구조(610)가 2가지 기능으로 사용될 수 있는 경우 (목표 장치(100)의 코드 저장 필요조건의 입장에서) 더 안전하고 더 효율적일 것이다.

[0230] 두번째 이슈는 키 리스트 데이터 구조(610)의 암호화 버전이 최초 키 리스트 데이터 구조(610)의 메시지 다이제스트를 포함한다는 것이다. 비록 각각의 개별적인 키들이 암호화되지만, 리스트 자체의 다른 부분은 메시지 다이제스트가 계산되는 시점에서 별도로 암호화되지 않는다는 점에 주목하여야 한다. 메시지 다이제스트 계산에 이어서, 전체 키 리스트 데이터 구조(610)(메시지 다이제스트를 포함함)는 최상위(또는 마스터) 키에 의해 식별된 키 값 및 알고리즘으로 암호화된다. 이것은 악의있는 제3자가 리스트를 부정하게 변경하고 새로운 메시지 다이제스트를 계산하며 진정한 리스트를 수정된 리스트로 대체하는 것을 금지하기 위해 수행된다. 키 리스트 데이터 구조(610)가 목표 장치(100)의 메모리 스페이스에 읽어 들여질 때, 이 (복호된) 메시지 다이제스트는 MAC가 임의의 다른 보안 암호화 코드 블록에 대하여 사용된 방법과 동일한 방법으로 키 리스트 자체의 진정성 및 유효성을 검증하기 위해 사용된다. 개별 키 외의 모든 요소들이 마스터 키로만 암호화된다는 사실은 리스트가 최상위 키 외의 임의의 키에 액세스할 필요가 없이 관통될 수 있음(및 리스트가 유지됨)을 의미한다. 또한, 키 리스트 목록(inventory)은 복호 블록의 단일 통과만으로 편집될 수 있다.

[0231] 중요한 제3 원칙은 개별 애플리케이션 코드 또는 미디어 스트림 지정 키가 각 목표 장치(100)의 개별화 키를 수용하도록 충분히 크게 될 수 있다는 것이다. 코드 또는 미디어 스트림이 대량 생산 디스크에 의해 분배된 경우에, 이것은 애플리케이션 개발자(520)가 개별 복호키와 함께 새로운 코드 지정 ID를 발행할 필요가 있음을 의미한다. 비록 이것이 허가 처리에 수반된 모든 당사자들 간에 전송되어야 하는 데이터 양을 최소화한다는 관점에서 덜 효과적일 수 있지만, 이것은 절충된 복호키를 추적하는 능력을 포함한(제한되는 것은 아님) 기능을 프로토콜에 추가한다. 우리는 이것을 키 취소를 취급하는 뒤의 섹션에서 설명할 것이다.

[0232] 주목해야 할 다음 이슈는 키 리스트 데이터 구조(610) 헤더가 리스트의 나머지를 구성하는 애플리케이션 지정 키와 동일한 특성들의 집합을 공유한다는 것이다. 사실, 헤더는 키 리스트 데이터 구조(610) 자체의 나머지에 대한 마스터 키(620)로서 생각할 수 있다. 따라서, 나머지 리스트의 관리를 결정하기 위해 이 키를 사용할 수 있는 한 동일한 동작 원칙이 적용될 수 있다. 이것은 목표 장치(100)의 보안 시스템의 시간 의존 관리를 포함한

다. 따라서, 목표 장치(100)는 미리 정해진 간격으로 그 보안 시스템을 갱신하도록 강요될 수 있으며, 이것은 그 자체로 극히 강력한 개념이다.

[0233] 키 리스트가 자신의 마스터 키(620)(리스트 헤더) 및 자신의 독립적 암호화 메카니즘을 각각 갖는 다수의 섹션을 포함할 가능성이 또한 존재한다. 임의의 다른 키와 같이, 리스트 헤더는 키 리스트 데이터 구조(610)를 해석하기 위해 사용되는 암호화 코드 블록을 지적할 수 있는 코드 지정 ID 필드(620)를 포함한다. 전체 리스트는 그 다음에 자신의 마스터 키(이것은 또다른 리스트 헤더이다)를 포함하는 또다른 마스터 리스트 내에 포함될 수 있다. 따라서, 전체 키 리스트 데이터 구조(610)는 회귀적으로 정의될 수 있다. 앞에서처럼, 이 회귀적 특성은 동일한 데이터 구조의 이전 버전의 단점을 취급하기 위해 새로운 키 리스트 데이터 구조를 생성함으로써 보안 시스템을 갱신하기 위해 사용될 수 있다. 전체 리스트의 보안성은 "최외측"(또는 최근) 보안층에 내포되기 때문에, 전체 키 리스트 데이터 구조(610)의 보안성은 항상 보안 소프트웨어의 최종 반복에 기초를 둔다.

[0234] 따라서, 키 리스트 데이터 구조(610)의 회귀적 특성은 강력한 특징이다. 앞의 섹션에서 설명한 데이터 구조의 정확한 구현이 큰 중요도가 없는 것도 또한 이유이다. 위에서 제공한 설명은 프로토콜 작업의 회귀적 성질을 만드는데 필요한 기능의 최소 부분집합인 특징들을 포함한 단순한 예이었다.

[0235] 키 리스트(160)는, 어떻게 구성되는가에 관계없이, 몇 가지 공통 환경 하에서 유지 및/또는 갱신될 수 있다. 이 환경은 리스트에 내포된 하나 이상의 키의 상태가 수정되는 경우를 포함한다(제한되는 것은 아님). 특수 키(210)의 소유권이 하나의 유닛으로부터 다른 유닛으로 이전될 수 있는 몇 가지 기본 메카니즘이 있고, 우리는 이것을 뒤의 섹션에서 설명할 것이다. 그러나, 어느 경우이든, 개정된 키 리스트가 유지되는 메카니즘은 2가지 분류, 즉 허가 기관(510)의 개입을 요구하는 것과 독립적으로 실행될 수 있는 것으로 나누어질 수 있다.

[0236] 이 프로토콜이 기초하고 있는 주요 동작 개념 중의 하나는 허가 기관(510)의 중앙 서버와 개별 목표 유닛 간의 필요한 네트워크 트래픽의 양을 최소로 감소시키는 것이다. 따라서, 키 리스트 데이터 구조(610)에 대한 임의의 임시 변경(그 이유는 뒤에서 설명함)은 목표 유닛(100)에 의해 독립적으로 유지될 수 있어야 한다. 이것의 주요 이유는 이러한 변경이 장치 보안 시스템에 대한 영구적인 변경보다 더 빈번하게 명시적으로 발생한다는 것이다(이것은 항상 목표 장치(100)와 허가 기관(510) 간의 상호작용에 의해서만 달성된다).

[0237] 어쨌든, 목표 장치(100)가 명료한 방법으로 마스터 키 리스트 데이터 구조의 현재 상태를 계속하여 추적할 수 있는 소정의 메카니즘이 있어야 한다. 이것은 2개의 "마스터" 리스트를 가짐으로써 달성될 수 있다. 이들 2개의 리스트 중 첫번째(우리는 이것을 영구 키 리스트라고 부른다)는 허가 기관(510)에 의해 유지된다. 이 리스트는 당해 목표 유닛(100)과 관련된 애플리케이션 지정 키의 "영구" 소유권과 관계된다. 두번째 리스트는 동일한 중요도를 갖지만, 이것은 "영구" 키 리스트 데이터 구조의 임시 수정과 관계된다. 이러한 수정은 리스트에의 추가이거나 리스트로부터 삭제하는 것일 수 있다. 2개의 리스트 자체의 데이터 구조의 구현에 있어서의 필요한 차이는 없고, 중요한 차이는 그들이 어떻게 유지되는가에서 발생한다. 목표 유닛(100)에 대하여 상기 리스트 중의 하나 또는 다른 것으로부터 데이터가 손실되는 경우로부터 복구하는 어떤 방법이 있다는 것은 바람직하다. 이러한 손실은 어떤 파국적 장애에 기인하거나 또는 리스트 중의 하나에 내포된 정보가 어떤 이유로(단순하게 또는 악의적으로) 훼손된 경우에 기인할 수 있다. 우리는 이러한 "키 리스트 훼손"의 경우의 영향에 대하여 뒤의 섹션에서 설명한다. 비록 영구 리스트가 허가 기관과의 접속에 의해 복구되는 것이 필요하지만, 허가 기관(510)이 특수 목표 장치의 임시 키 리스트를 복구할 수 있는 것은 필요하지 않다(또는 바람직하지 않다). 이러한 입장에 대한 많은 이유가 있지만, 주요 이유는 임시 키 리스트가 영구 키 리스트보다 훨씬 더 빈번하게 갱신될 가능성이 있고 중앙의 허가 기관(510)과 목표 유닛 간의 필요한 네트워크 트래픽의 양을 절대 최소치로 유지하고 싶기 때문이다. 그럼에도 불구하고, 허가 기관(510)이 몇 가지 이유(이들 중 일부에 대해서는 뒤에서 설명한다)로 특수 목표 장치의 임시 키 리스트에 대한 수정을 행할 수 있는 것이 잠재적으로 바람직하다. 이 경우에, 이 리스트는 목표 장치의 1차 비밀키(이것은 허가 기관(510)에 알려져 있음)를 이용하여 암호화되는 것이 바람직할 것이다.

[0238] 위에서 언급한 바와 같이, 양측 키 리스트 데이터 구조의 완전성은 리스트 자체와 함께 저장된 부호화 메시지 다이제스트(디지털 서명)를 이용하여 검증될 수 있다. 이 메시지 다이제스트를 생성하기 위해 사용되는 보안 코드 메카니즘의 구현은 위의 섹션에서 설명하였고, 그 절차를 다시 설명할 필요는 없을 것이다. 우리는 또한 손실 및/또는 훼손이 있는 경우에 영구 키 리스트 데이터 구조(610)를 복구하는 절차에 대하여 이미 설명하였다. 취급되어야 할 유일한 나머지 이슈는 임시 키 리스트 데이터 구조의 시간 의존 부분을 어떻게 해석하고 임시 키 리스트가 사용불능으로 된 경우를 어떻게 다룰 것인가이다.

[0239] **임시 라이센스 이전**

[0240] 이것은 타임스탬프 필드(230)의 사용이 가장 중요한 보안 프로토콜의 섹션들 중 하나이다. 위에서 설명한 것처럼, 임시 키 리스트 데이터 구조는 목표 장치의 영구 키 리스트와 정확히 동일한 방법으로 구성된다. 그러나, 양자간에는 몇 가지 차이가 있다. 첫번째 차이는 임시 키 리스트가 목표 장치의 비밀키(104) 중의 하나를 이용하여 잠재적으로 암호화될 수 있다는 것이다. 허가 기관(510)이 정상 환경 하에서 이 데이터 구조를 재구성할 필요가 없기 때문에, 목표 장치의 어떤 키를 이용하여 데이터 구조를 암호화할지는 명시적으로 관계가 없다. 그러나, 이 리스트가 목표 장치의 1차 비밀키를 이용하여 또한 암호화되었다면 허가 기관(510)에 대하여 잠재적으로 사용될 것이다. 이것의 이유는 허가 취소와 함께 이루어져야 하고, 그 상황은 뒤의 섹션에서 설명할 것이다.

[0241] 임시 키 리스트와 영구 키 리스트 간의 두번째(및 더 중요한) 차이는 가장 최근의 임시 키 리스트 데이터 구조와 관련된 타임스탬프 값(230)의 복사본이 또한 목표 장치(100) 내에 저장(즉 온칩)된다는 것이다. 이 레지스터는 소프트웨어 판독가능이 아니고, 보안 블록의 일부이기 때문에 보안 코드에 의한 덮어쓰기만 가능하다. 이 레지스터의 값은 임시 키 리스트 데이터 구조가 어떻게든 손실되고 및/또는 훼손된 경우에 무엇을 할 것인지를 결정하기 위해 사용된다. 우리는 그 절차를 이 섹션의 뒷부분에서 설명할 것이다.

[0242] 임시 키 리스트와 영구 키 리스트 간의 또다른 차이는 목표 유닛(100)이 특수 키의 소유권을 그 영구 리스트로부터 다른 유닛(100)의 임시 리스트로 (임시로) 이전할 수 있지만, (올바르게 동작하는) 장치는 특수 키의 소유권을 그 임시 키 리스트로부터 임의의 다른 키 리스트로 이전할 수 없다는 것이다. 이것은 물론 다른 유닛의 임시 키 리스트뿐만 아니라 목표 장치(100) 자신의 영구 키 리스트도 또한 포함한다. 이것은 영구 소유자만이 어떤 장치가 언제 임의의 특수 키를 "차용"하도록 허용되는지 결정할 수 있다는 의미이다. 그러나, 이 "대여" 기간은 막연하게 될 수 있다는 점에 주목한다(이 트랜잭션은 허가 기관에 접촉할 필요없이 실행될 수 있다). 이 "영구 대여" 특징은 현대의 디지털 저작권 제어 정보(Copyright Control Information; CCI) 시스템의 일부인 표준 "1회 복사(Copy Once)" 기능 필요조건과 등가이다.

[0243] 이제 도 167을 참조하면, 임시 "키 체크아웃" 절차를 나타내는 상세한 흐름도가 도시되어 있다. "키 소유권" 이전 절차는 도서관으로부터 서적 복사본을 체크아웃하는 절차와 어느 정도 유사하다. "차용자"(720)가 영구 소유자("대여자"(710))로부터 특수한 애플리케이션 지정 키(550)의 임시 사용을 요청하면, 대여자(710)는 먼저 상기 특수 키의 사용을 키 체크아웃 협상 처리 동안에 금지하는 갱신된 임시 키 리스트를 단독으로 발생한다. 이 동작은 하나 이상의 차용자(720) 유닛이 동일한 키를 요청하는 것을 금지한다. 대여자 유닛(710)의 임시 키 리스트에 존재하는 "체크아웃 키"는 임의의 특수 키에 대한 액세스를 제어하는 신호기(semaphore)로서 유효하게 사용된다. 그러나, 키가 "구속 상태"로 있는 초기 시간량은 비교적 짧은 기간으로 제한되어야 한다. 이것은 차용자(720) 장치가 특수 키에 대한 액세스를 장기간 동안 요청하고 그래서 특수 키의 사용을 불공정하게 독점하는 것으로부터 어떤 이유로 트랜잭션을 완성할 수 없는 경우를 방지하기 위한 것이다. 상기 비교적 짧은 체크아웃 협상 단계 타임아웃은 대여자 유닛(710)에 대항하여 "서비스 거부" 공격의 등가물을 설치하려고 하는 악의적 장치에 대한 싸움에서 또한 도움이 된다. 사실, 대여자 유닛(710)은 자신의 "승인된 차용자" 리스트에 없는 장치로부터의 요청이 있는 경우 또는 그러한 유닛들의 임의의 하나가 소정 시간 내에 너무 많은 요청을 하려고 시도하는 경우 그 요청을 선택적으로 무시할 수 있다. 이 임시 블록이 키에 존재하는 정확한 시간 길이는 중요하지 않지만, 임의의 주어진 체크아웃 절차가 완성될 수 있도록 충분히 길어야 한다. 네트워크 트래픽 또는 대기시간이 높은 시간대에서는 이 기간을 연장할 수 있다.

[0244] 주어진 키의 하나 이상의 복사본이 동시에 체크아웃될 수 있는 경우에, 대여자 장치(710)의 임시 키 리스트 내의 적당한 필드를 사용하여 얼마나 많은 키 복사본이 임의의 한 시점에서 체크아웃되는지를 표시할 수 있다는 점에 주목한다. 차용자(720)와 대여자(710)가 주어진 키에 대한 특정 체크아웃 기간을 협상하였으면, 대여자(710)는 키의 암호화한 복사본(740)을 차용자(720)에게 보낸다. 이 암호화는 대여자 장치(710)에게만 알려져 있는 임시 비밀키(730)를 이용하여 실행된다. 차용자(720)가 암호화 키의 정확한 수령을 (암호화 메시지로부터 계산된 메시지 다이제스트에 의해) 승인한 경우, 대여자(710)는 체크아웃 키의 "대여 기간"을 연장하고 임시 비밀키(730)를 차용자 장치(720)에 보낸다. 이 대여 처리의 최대 기간은 프로토콜의 동작에 중요하지 않고 이 값의 선택에 있어서 행하여져야 할 몇 가지 교환조건이 있다. 우리는 그러한 특수한 이슈를 이 섹션의 뒷부분에서 설명할 것이다. 위에서 설명한 예에서, 우리는 "차용자(720)"와 "대여자(710)" 장치가 키마다 체크아웃 기간의 실제 길이를 협상할 수 있는 것으로 가정하지만, 이것은 프로토콜의 필요조건이 아니다.

[0245] 차용자(720) 또는 대여자(710)의 임시 키 리스트가 갱신되는 바로 앞의 시점에서, 이 새로운 임시 리스트와 관련된 타임스탬프 값(230)의 복사본이 목표 장치(100)에 비휘발성 형태로 저장된다. 그 시점에서, 임시 키 리스트 데이터 구조의 암호화 버전이 온보드 NVRAM, 플래시 ROM 또는 일부 하드 디스크(750)의 백업 파일 외부와 같은 메모리에 안전하게 기록될 수 있다(또는 소정의 다른 더 영구적인 위치에 저장될 수 있다). 임시 키 리스트

는 잠재적으로 판독가능하고 영구 키 리스트보다 훨씬 더 빈번하게 갱신되기 때문에, 이 리스트는 목표 유닛에 신속히 액세스되는 것이 바람직하고, 그래서 이 리스트는 액세스 대기시간이 비교적 짧은 적어도 하나의 위치에 저장되는 것이 권장된다(그러나 이것은 프로토콜의 실제 필요조건이 아니다). 반면에, 전력 고장에 의해 중간 시간량 동안 목표 장치(100)의 기능 상실을 잠재적으로 야기할 수 있기 때문에, 상기 리스트를 저장하는 장소는 어떤 휘발성 기억 매체가 아닌 것(DRAM 등)이 권장된다. 우리는 이 이슈에 대한 세부를 이 섹션의 뒤에서 설명할 것이다.

[0246] 특수 키에 대한 체크아웃 기간이 만료되면, 차용자(720) 및 대여자(710) 장치는 그들 각각의 임시 키 리스트 데이터베이스를 독립적으로 갱신할 수 있다. 따라서, 차용자(720)가 계산 대상 특수 키를 돌려주기 위해 대여자(710) 장치와 접촉할 필요는 없다. 이것은 차용자(720)와 대여자(710) 장치가 멀리 떨어져 있는 경우에 중요한 편리 요소이다. 물론, 이 동작의 보안성은 키 타임스탬프 기록을 생성 및 제어하기 위해 사용하는 온칩 클럭들 간의 매우 단단한 상관성에 의존할 수 있다. 따라서, 시간/날짜 클럭이 보안 시스템의 통합 부분으로 되어야 하고, 그래서 중앙 서버와의 트랜잭션에 의해 덮어쓰기 될 수 있어야 한다. 또한, 클럭들은 악의있는 사용자가 내부 타임스탬프 값(230)을 수정하려고 하는 경우에 부정한 변경에 저항하도록 및 평소에 시스템 전력 장애가 발생한 경우에 살아남을 수 있도록 충분히 강하게 설계되어야 한다. 이 클럭이 배터리에 의해 전력이 공급되는 경우 배터리가 제거되거나 시간 경과에 따라 배터리 전력이 없어지는 것을 상상할 수 있기 때문에, 시스템은 허가 기관과의 안전한 상호작용에 의해 클럭이 잠재적으로 재시작되거나 리세트될 수 있도록 하는 방법으로 설계되어야 한다.

[0247] 따라서, 우리는 특수 애플리케이션 지정키(550)의 소유권이 하나의 유닛으로부터 다른 유닛으로 임시로 이전될 수 있는 상황을 설명하였다. "대여 기간"의 끝에서, 차용자(720) 유닛과 대여자(710) 유닛은 최초 소유자로의 키의 "복귀"를 반영하도록 그들의 임시 키 리스트 데이터 구조를 갱신할 수 있다. 이 절차는 양측 유닛에서 독립적으로 수행될 수 있고, 따라서 두 장치 간에 어떤 상호작용을 필요로 하지 않는다는 점에 주목한다.

[0248] 이제, 하나 이상의 키가 "체크아웃" 하는 동안 또는 "대여중"인 동안 하나 또는 다른 임시 키 리스트 데이터 구조가 훼손 및/또는 상실된 경우를 설명한다. "대여자(710)" 유닛 측에서, 임의의 키가 체크아웃될 때, 해야 할 첫번째 일은 "대여" 기간의 끝을 결정하는 것이다. 이 값은 대여 기간의 지속기간을 현재 시간/날짜 필드의 값에 가산함으로써 명백하게 구성된다. 이 시간/날짜 값은 그 다음에 장치의 임시 키 리스트가 마지막으로 갱신된 결과로서 저장된 값과 비교된다. 새로운 값이 오래된 값보다 더 크면(더 뒤에 있으면) 새로운 값이 오래된 값 대신으로 덮어쓰기 된다. "차용자(720)" 측에서, 상기와 동일한 처리가 사용되고, 그래서 그 결과는 임의의 주어진 목표 유닛에 있어서, 임시 키 리스트 타임스탬프가 항상 특수 유닛(100)의 임시 키 리스트의 일부로서 저장된 임의의 타임스탬프 중 최신의 것으로 된다.

[0249] 만일 유닛(100)의 임시 키 리스트가 상실되거나 또는 다른 방식으로 부적절하게 수정되면, 임시 키 리스트와 영구 리스트 양자는 상기 "최신 타임스탬프" 값이 만료되는 시점(실제로는 "타임아웃" 기간)까지 디스에이블된다. 그 시점에서, 유닛은 영구 키 리스트를 사용하는 것으로 되돌아갈 수 있고 새로운 임시 키 리스트를 재구성하는 처리를 시작할 수 있다.

[0250] 따라서, 만일 장치의 임시 리스트가 부정하게 변경되거나 삭제되면, 유닛은 타임아웃 기간이 만료될 때까지 효과적으로 비동작 상태로 된다. 이 타임아웃 절차는 불필요하게 구속하는 것으로 보이지만, 이것은 어떤 악의적인 동작의 결과로서 또는 키를 하나의 유닛으로부터 다른 유닛으로 이전하는 동안에 발생할 수 있는 어떤 돌연한 고장(glitch)(전력이 나가거나 네트워크 접속이 다운되는 것 등) 때문에 있을 수 있는 임의의 특수 애플리케이션 지정 키의 다수의 복사본에서의 잠재적인 문제점을 피할 수 있게 한다. 또한, 임시 키 리스트 데이터 구조를 부정하게 변경한 결과로서 심각한 영향을 받을 잠재성은 모든 더 정교한 공격자에 의한 실행을 저지하는데 도움이 되어야 한다.

[0251] 이 점에서 프로토콜의 동작을 강화하기 위해 사용될 수 있는 다수의 선택적인 추가의 특징들이 있다. 그러한 가능한 옵션 중의 하나는 암호화 키 리스트 데이터 구조 중의 어느 하나(또는 양자)로부터 발생된 부호화 메시지 다이제스트(디지털 서명)를 목표 장치의 온칩 보안 섹션에 저장된 값들에 추가하는 것이다. 디지털 서명의 복호로부터 발생하는 MAC 값은 전체 복호 처리를 통과할 필요없이 임의의 특수 키 리스트의 유효성을 신속히 검증하기 위해 사용될 수 있다. 그러나, 네스티드 키 리스트를 증가시키는 문제는 이 복호 처리가 비암호화 키를 최종적으로 생성하기 위해 소정 시점에서 복수 회 수행되어야 한다는 것을 의미하고, 따라서 디지털 서명이 온칩으로 저장되는 것은 프로토콜의 동작에 중요한 것이 아니다.

[0252] 증대(enhancement)를 위한 다른 하나의 가능성은 온칩 타임스탬프 값을 하나씩 저장하기보다는 쌍으로 저장하는

것이다. 추가적인 타임스탬프는 임시 키 리스트가 갱신되어야 할 때 가장 이른 (다음) 시간을 표시하기 위해 사용할 수 있다. 이것은 목표 장치(100)가 리스트를 지속적으로 체크할 수 없기 때문에(이것은 복호 처리의 수행을 수반한다) 그 임시 키 리스트를 개정할 필요가 있을 때를 더 쉽게 결정할 수 있게 한다. 비록 이 특징이 매우 유용하지만, 유닛이 이 프로토콜을 실행할 수 있게 하는 기본적인 필요조건은 아니다. 만일 상기 제2 타임스탬프를 내포한 시스템이 구현되면, 2개의 타임스탬프가 어떤 이유로 "동기 파괴(out of sync)"로 되는 경우에 혼란을 일으킬 가능성이 있다. 그러한 예 중의 하나는 하나의 타임스탬프가 기록된 직후이고 두번째 타임스탬프가 갱신되기 전의 시점에서 돌연한 전력 고장이 발생한 경우이다.

[0253] 다루어야 할 최종 이슈는 상기 임시 키 리스트 타임스탬프의 값에 대한 최소 및 최대 한계가 무엇이냐 하는 문제이다. 한편으로, 최대 "임시 대여 기간"의 더 큰 한계는 특수 데이터 애플리케이션(또는 미디어 스트림)의 사용권이 상당히 긴 기간동안 하나의 유닛으로부터 다른 유닛으로 이전할 수 있게 한다. 이것은 사용자가 미디어 스트림의 소유권을 자신의 "홈 유닛"으로부터 휴대용 장치로 이전하고자 하는 경우에 잠재적으로 사용할 수 있다. 긴 "체크아웃 기간"을 가지면, 사용자가 최초의 "대여자" 유닛(710)과 접촉할 필요없이 상기 체크아웃 기간(그 관련 임시 키와 함께)을 가진 휴대용 장치를 가지고 긴 여행을 할 수 있다. 긴 "체크아웃 기간"의 불리한 점은 최초 유닛에서 임시 키 리스트 데이터 구조에 어떤 일이 발생하면 그 유닛이 장시간 동안 잠재적으로 디스에이블된다는 것이다.

[0254] 상기 최종 이슈는 일부 악성 코드가 온칩 타임스탬프 레지스터의 값을 어떤 불확실한 값으로 설정할 수 있는 경우에 목표 유닛(100)에서 잠재적인 위험성을 또한 나타낸다. 이것은 공격의 목표를 디스에이블시키는 것과 잠재적으로 동등한 것이고, 따라서 이 타임스탬프 레지스터의 값은 "보안" 코드 블록에 의해서만 기록될 수 있어야 한다. 또한, 각 유닛이 별개의 비밀키 집합을 갖기 때문에, 하나의 특수 유닛의 비밀키(104) 데이터의 발견은 부당한 장치가 합법적 장치로 효과적으로 가장할 수 있는 경우를 제외하고 임의의 다른 장치의 관심을 일으키지 않을 것이다. 이 공격 모드는 뒤에서 신원 검증과 관련된 이슈를 다루면서 설명한다.

[0255] **영구적 라이센스 이전**

[0256] 이 절차와 관련한 많은 요소들은 이 문서의 앞의 섹션에서 설명하였다. 특정 키를 하나의 유닛으로부터 다른 유닛으로 영구적으로 이전하는 기본 처리는 앞에서 도 5와 관련하여 설명하였다. 많은 점에서 이 절차는 바로 앞의 섹션에서 설명한 것처럼 키 소유권을 임시 이전하는 절차와 본질적으로 유사하다.

[0257] 2개의 절차 간의 중요한 차이점은 영구 이전이 임시 이전보다 처리가 더 간단하고 영구 키 소유권 이전 절차는 허가 기관(510)과 목표 유닛(100)의 상호작용을 활용해야 한다는 것이다. 영구 이전 처리가 더 간단한 이유는 임시 키 체크아웃 절차에서의 선행조건인 체크아웃 시간 기간 협상을 영구 이전 처리에서 요구하지 않는다는 사실에 있다. 영구 이전 기능이 허가 기관(510)과 목표 유닛(100) 간의 상호작용을 활용하는 이유는 갱신된 키 리스트 데이터 구조가 트랜잭션의 양 끝에서 재구성될 수 있어야 한다는 사실에 기인한다.

[0258] 영구적 라이센스 이전이 일반적으로 허가 기관(510)과의 상호작용에 의해 발생하기 때문에, 어떤 애플리케이션 또는 미디어 스트림 지정 키가 어떤 목표 유닛에 속하는지에 관한 기록이 있다. 앞에서 설명한 바와 같이, 이것은 목표 유닛(100)의 키 리스트가 어떤 파국적 데이터 상실 상황 후에 복원되어야 하는 경우에, 또는 특수 목표 유닛(100)의 소유권이 다른 엔티티로 이전된 경우에 필요하다. 허가 기관(510) 부분에서의 이러한 개입은 특수 키의 영구 소유권이 하나의 목표 유닛(100)으로부터 다른 목표 유닛으로 이전된 경우에 또한 필요하다. 소유자가 다른 엔티티로부터 최초로 구매한 자산을 되파는 이러한 능력은 "최초 판매 권한(right of first sale)"이라고 알려져 있고, 여기에서 설명한 프로토콜이 상기 특수 기능을 지원하는 능력은 중요한 것이다.

[0259] 목표 유닛(100)의 영구 키 리스트가 허가 기관(510)에 의해 유지되는 사실의 다른 중요한 태양은 목표 유닛(100)이 어떻게든지 절충되었음이 입증된 경우에 또는 키들 중의 하나가 절충된 것으로 확인된 경우에, 이 보디(body)가 개별적인 목표 유닛(100)의 라이센스 키의 일부 또는 전부를 취소할 능력이 있다는 점이다. 각각의 목표 유닛(100)에 (위에서 설명한 것처럼) 유일한 키 리스트를 제공할 가능성이 있기 때문에, 허가 기관(510)이 임의의 절충된 키의 소스를 추적할 기회가 또한 제공될 수 있다. 그러한 상황에서, 이 프로토콜은 전통적인 워터마크 처리의 단점(워터마크가 미디어 스트림의 품질에 악영향을 줄 가능성 등) 없이 일반적으로 "워터마크" 특징의 기능과 관련된 기능을 충족시킬 수 있다.

[0260] 그러한 경우로 보이지는 않지만, 디지털 콘텐트 소유자의 프라이버시는, 애플리케이션 코드 또는 미디어 스트림 지정 ID 정보가 애플리케이션 개발자(520)에게서 발원하고 허가 기관(510)이 임의의 특수한 애플리케이션 또는 미디어 스트림과 그 허가된 소유자 간에 연합(association)을 만들 수 있는 충분한 정보를 가질 필요가 없기 때

문에, 이 처리에 의해 여전히 유지된다. 소유자의 프라이버시를 보호하기 위한 이러한 능력은 역시 이 프로토콜의 중요한 태양이다.

[0261] 영구 키 이전 처리와 관련하여 주목해야 할 마지막 이슈는 사실상 영구적 키 이전이 수행하는 모든 동일한 기능이 임시 키 라이센스 이전과 함께 달성될 수 있다는 점이다. 그러나, 목표 유닛의 보안 시스템의 유지는 이상적으로 중앙 보안 서버에 의해 제어되어야 하는 기능이고, 따라서 사슬의 어딘가의 적소에 그러한 메카니즘을 가질 필요가 있다. 또한, 사용자가 자신의 프라이버시를 유지하는 것에 관심이 있는 경우에, 중앙 서버가 저작권 보유자와 목표 유닛(100) 간의 완충기(buffer)로서 작용할 수 있다는 사실은 크게 유익한 것이다. 마지막으로, 허가 기관(510)이 상기 기능을 임시 키 이전 메카니즘을 제외하고 설정하는 특수 목표 유닛(100)의 영구 키 리스트에 대한 중앙 백업 스토리지 메카니즘으로서 작용할 수 있는 점도 또한 흥미로운 점이다.

[0262] **시스템 소유권 양도, 허가 취소 및 보안 시스템 갱신**

[0263] 하나 이상의 목표 장치(100)의 라이센스(또는 키)를 취소하는 몇 가지 상이한 수단들이 있다. 가장 간단한 방법은 목표 장치(100)의 1차 비밀키를 단순히 갱신하는 것이다. 이 시점에서, 목표 장치(100)는 자신의 영구 키 리스트에 액세스할 수 없고, 따라서 새로운 키 리스트를 생성하는 처리를 시작하여야 한다. 1차 비밀키가 임시 키 리스트 데이터 구조의 암호화 처리에 사용될 수 없는 경우에 상기 임시 키 리스트는 영구 키 리스트가 다른 방식으로 액세스불능으로 된다 하더라도 잠재적으로 여전히 액세스될 수 있다는 점에 주목한다.

[0264] 이 점에 대해서는 임시 키 리스트의 암호화 처리를 설명하면서 앞에서 설명하였다. 이 때문에, 영구 및 임시 키 리스트 데이터 구조 양자에 대한 암호화 키로서 목표 장치(100)의 1차 비밀키를 사용하는 것이 아마도 최상의 아이디어일 것이다.

[0265] 목표 장치(100)의 소유권이 일 개인으로부터 다른 개인으로 변경되는 경우에, 이 소유권 변경을 시행하는 가장 간단한 방법은 목표 장치(100)의 1차 비밀키를 어떤 새로운 값으로 설정하는 것이다. 그러나, 이러한 변경이 최초 소유자가 자신의 모든 영구 키를 목표로부터 복구하는 기회를 갖기 전에 발생하면, 그들은 자신의 라이센스를 잃을 것이다. 최초 소유자가 관련 영구 키 리스트의 소유권을 목표 장치와 함께 이전하기를 원하는 경우, 그 특수 장치와 관련된 소유권 정보(허가 기관(510)에 저장되어 있음)를 변경하는 것 외에 목표 장치(100)가 해야 할 것은 아무것도 없다.

[0266] 라이센스 취소가 발생할 수 있는 다른 경우는 특수 목표 장치(100)의 영구 키 리스트의 마스터 키가 "만료"되는 경우이다. 목표 장치(100) 보안 시스템의 갱신이 영구 키 리스트의 일부로서 저장되기 때문에, 이 상황은 잠재적으로 비참한 반향을 가질 수 있다.

[0267] 잠재적으로 이러한 곤경으로부터 회복될 수는 있지만, 목표 장치(100)는 처음부터 구축되는 완전히 새로운 "신뢰 사슬(chain of trust)"을 가질 필요가 있다. 이 상황에서, 새로 개시되는 보안 시스템의 핵심은 목표 장치(100)의 일부에서 원자식으로 동작하는 것처럼 검증될 수 있는 그러한 계산에만 기초를 두어야 한다. 따라서, 다른 방식의 범용 코드(이것은 잠재적으로 의심을 받을 수 있다)의 최소량만을 요구하는 임의의 해싱 함수의 사용을 배제한다. 다행히도, 이 상황은 만료가 되지 않은 영구 키 리스트 데이터 구조의 일부로서 검증가능하게 안전한 코드 파편(fragment)의 영구적인 핵심을 항상 유지하는 간단한 방법으로 피할 수 있다. 그러나, 이것은 위에서 설명한 이유 때문에 보안상 위험이 있고, 따라서 이 영구 코드 핵심의 콘텐트는 가능한 한 많이 제한되어야 한다.

[0268] 라이센스 취소가 발생할 수 있는 또다른 경우는 허가 기관(510)이 목표 장치(100)의 영구 또는 임시 키 리스트의 특수 키 엔트리를 무효화(override) 하기로 선택한 경우이다. 이것은 보안 시스템 업그레이드가 필요한 경우 또는 특수 목표 장치(100)가 특수 애플리케이션 또는 미디어 스트림의 허가되지 않은 복사본을 갖고 있는 것으로 확인된 경우에 사용될 수 있다. 목표 장치(100)는 평소에 자신의 키 리스트 데이터 구조를 유지하고 있기 때문에, 이 절차는 허가 기관(510)과 목표 장치 간의 정상적인 네트워크 트래픽 양보다 더 큰 트래픽을 수반할 것이다. 따라서, 이 동작 과정은 극한의 경우를 위하여 보존되어야 한다.

[0269] 그럼에도 불구하고, 이러한 절차는 당해 목표 장치(100)가 경쟁 키(disputed key)를 조사하여 디스에이블시키도록 설계된 목표 지정 커스텀 버전으로 자신의 보안 시스템 소프트웨어를 개정하게 하고 및/또는 오래된 소프트웨어를 갱신 버전으로 교체하게 함으로써 달성될 수 있다. 물론, 이 절차는 목표 장치(100)가 허가 기관 중앙 서버와의 접속을 개시한 때의 시점에서의 동작으로만 설정할 수 있다. 정상적인 상황에서는 임의의 특수 목표 장치(100)가 임의의 특수한 계획으로 허가 기관(510)과 접촉을 개시하는 것을 보장할 수 없다. 다행히도, 당해 목표 장치(100)는 자신의 영구 키 리스트에 대한 임의의 새로운 추가를 인증하기 위해 허가 기관(510)에 (직접

적으로 또는 간접적으로) 접속하여야 하고, 그래서 임의의 키 취소 동작이 새로운 키 허가 절차의 일부로서 달성될 수 있다. 위에서 설명한 "보안 시스템 타임아웃" 메카니즘을 사용하여 이 "리스트 폴리싱(list policing)" 동작을 지원하는 것도 또한 가능하다. 그러나, 이것이 그 경우인 것 및 그러한 시스템이 사용자 프라이버시 권리의 훼손을 초래할 가능성이 있다는 것은 이 프로토콜에 대한 필요조건이 아니다.

[0270] **기타 관심사:**

[0271] 반드시 프로토콜 자체의 일부는 아니지만 그럼에도 불구하고 여기에서 설명하는 프로토콜을 적절히 실행할 수 있는 실제적인 시스템을 생성하는 처리에서 다루어야 할 다수의 이슈가 있다. 이러한 이슈들 중의 일부는 실제 프로세서 장치의 구현에 의존하고, 다른 것들은 대부분 애플리케이션에 특수한 것이다. 이 정보는 적합한 목표 장치(100)의 적절한 구성에 밀접한 관계가 있기 때문에, 우리는 이 이슈의 일부에 대하여 다음 섹션에서 설명할 것이다.

[0272] **상호 동작이 가능한 장치 수 제한**

[0273] 1차 목표 장치가 소유권을 임시로 이전할 수 있는 장치의 총 수를 저작권 보유자가 제한하고자 하는 경우에, 이것은 임의의 하나의 시간에 동작할 수 있는 제한된 수의 공개/개인키 쌍을 확립하는 수단에 의해 달성될 수 있다. 이것은 동일한 애플리케이션 지정 키의 복수의 복사본이 위의 섹션에서 설명한 동시 "대여중"인 경우와 다르다는 점에 주목한다. 특수 목표 장치(100)로부터 임의의 애플리케이션 지정 키를 "체크아웃"할 수 있는 장치들의 리스트가 특정 일련 번호 집합으로 제한될 수 있는 다른 시나리오가 가능하다. 허가 기관(510)은 목표 장치(100)의 보안 시스템이 관리되는 것과 정확히 동일한 방법으로 상기 "승인된 차용자" 리스트를 관리할 수 있다. 따라서, 허가 기관(510)은, 예를 들면, "승인된 차용자" 리스트의 일련 번호 집합을 최초 목표 장치(100)와 동일한 소유권 정보를 갖는 것으로 제한할 수 있다. 이 문제에 대한 다른 가능한 해법은 임의의 "차용자" 장치(720)가 중앙 허가 기관(510)의 공개키로 증명서(certificate)를 복호함으로써만 검증될 수 있는 신임장(credential)(서명된 증명서 등)을 대여자에게 제공함으로써 "인증"된 차용자로서 유효화될 것을 요구하는 것이다. 이 시나리오는 물론 특수 유닛이 절충된 것으로 결정된 경우에 허가 기관(510)이 상기 증명서를 취소하는 능력을 수반한다. 이 증명서 취소 처리가 달성될 수 있는 한가지 잘 알려져 있는 방법은 정기적으로 발행되는 "취소 리스트"를 이용하는 것이다.

[0274] **비밀키 발견 및 신원 검증 문제**

[0275] 특수 플레이어에 대한 1차 비밀키가 물리적인 분해 및 칩 다이 시험에 의해 발견되면, 각 장치가 별도의 비밀키(104) 집합을 가질 것이기 때문에 임의의 다른 장치의 보안성을 희생시켜서는 안된다. 그러나, 특수 플레이어의 1차 키가 어떻게든 희생되면 인증되지 않은 장치가 합법적 목표 장치로서 가장할 잠재적인 가능성이 있다. 이 문제가 검지되지 않은 경우에, 이 지식으로 무장한 허가받지 않은 장치가 그 특수 목표 장치에 발행된 임의의 애플리케이션 지정 복호키를 손상시킬 가능성이 있다. 목표 장치(100)의 일련 번호(106)는 허가 기관(510)이 제1 장소에 있는 장치에 복호키를 발행하기 위해 등록되어야 하기 때문에, 이점에 대한 문제점은 허가되지 않은 장치에 의한 다른 방식의 합법적인 목표 장치(100)의 제한으로 명시적으로 제한된다.

[0276] 그러나, 목표 장치(100)의 비밀키가 이러한 처리에 의해 발견되면, 암호화 키 리스트 다이제스트의 복사본을 이전에 백업한 시험에 기초하여 그 유닛에 허가를 준 모든 애플리케이션 지정키를 손상시킬 가능성이 있다. 이 때문에, 1차 비밀키와 2차 비밀키는 이 키들의 값을 찾아내고자 하는 임의의 시도가 키 데이터의 상실을 야기하는 "변경 방지" 방식으로 목표 칩에서 구현되어야 한다.

[0277] 목표 장치(100)에서 변경 방지 특징을 구현하는 다수의 수단이 있지만, 그 정확한 구현은 이 문서에서 설명한 프로토콜에 중요한 것이 아니다. 만일 "비밀키 상실" 상황이 사용자 부분에서 부주의(또는 남용)에 따른 어떤 악의없는 행동에 의해 발생하였으면, 합법적인 사용자는 손상된 유닛의 애플리케이션 지정키를 새로운 장치에 이전시키도록 배열할 수 있는 허가 기관(510)에 자신의 손상된 유닛을 되돌려보낼 수 있어야 한다. 그러나, 최초의 목표 장치(100)가 기능할 수 없는 경우에, 새로운 목표 장치(100)로의 키 이전은 (적어도 제1 장소에서 명문으로 허가 기관(510)에 공급되지 않은 키에 대해서) 애플리케이션 개발자(520)와의 트랜잭션을 수반하여야 한다.

[0278] 그러나, 다른 진정한 목표 장치(100)를 흉내낼 수 있었던 장치는 표면상으로, 신용하는 합법적으로 허가된 장치를 하나 이상의 애플리케이션 지정 키의 임시 포기하는 소유권으로 또는 일시 정지한 동작으로 속일 수 있다는 점에 주목하여야 한다(위에서 설명하였음). 후자의 경우가 발생하면, 키를 차용하려고 시도한 모든 장치들을 디스에이블시키는 "부정 장치(rogue unit)"를 가질 가능성이 존재한다. 전자의 경우가 발생하면, 임의 갯수의 애

플리케이션 또는 미디어 지정 키들이 잠재적으로 절충될 수 있다.

[0279] 따라서, 특수 목표 장치(100)에 대해 잠재적인 "허가된 차용자"의 수를 허가 기관(510) 서버로부터의 안전한 갱신에 의해 합법적인 장치에 공급될 수 있는 리스트로 제한하는 것에 관한 위에서 설명한 개념은 세심한 것이다. 전자의 경우에, 이것은 유닛이 제1 장소에서 그들에게 실제로 속하지 않는 한 다른 신용하는 장치의 소유자가 그 비밀키에 액세스하기 위해 기능 유닛을 분해하는 해커에 의해 디스에이블된 합법적 장치를 갖는 것을 방지할 것이다. 후자의 경우에, 이것은 하나의 지점에서 허가 기관에 적절히 등록된 허가된 장치이었던 그러한 장치들에게 애플리케이션 또는 미디어 지정 키가 이전하는 것을 제한할 것이다. 그럼에도 불구하고, 단호한 해커는 여전히 합법적 장치를 구매하고, 그것을 분해하여 어떻게든 그 비밀키 데이터에 액세스하며, 그 다음에 합법적 장치로 가장하기 위해 이 정보를 이용할 수 있다.

[0280] 그래서, 이러한 종류의 흉내 사건(impersonation event)을 탐지하기 위하여 어떻게 할 것인지에 대한 이슈가 남아있다. 이러한 성질에 극히 잘 대처하는 반대자(opponent)를 좌절시키기 위한 유일한 성공적인 전략은 적어도 비용 교환조건 관점에서 잠재적 이득이 요구되는 노력을 무가치하게 하도록 시스템을 설계하는 것이다.

[0281] 통신을 행하는 다른 알 수 없는 장치의 진정성을 증명하기 위한 몇 가지 수단이 있다. 그러나, 장치가 사실상 청구하는 것임을 입증하는 가장 성공적인 방법은 이 장치를 다른 장치와 구별되게 하는 특성에 촛점을 맞추는 것이다. 이 문서에서 설명하는 디지털 복호 메카니즘과 같은 특수 용도 장치의 경우에는 장치가 보안 프로토콜을 적절히 실행하고 주어진 입력 변수 집합에 기초하여 정확한 결과를 계산하는 능력이다. 그러나, 여기에서 설명하는 보안 프로토콜은 공개적으로 알려진 알고리즘에 기초를 두기 때문에, 이것은 표면상 계산 완료까지 충분한 시간을 갖는 임의의 범용 컴퓨팅 장치에 의해 달성될 수 있다. 사실, 이 이슈는 장치를 유일하게 하는 비밀키 정보가 어떻게든 절충되면 공개적으로 이용가능한 기술에 기초한 임의의 장치에 대하여 잠재적인 문제가 될 것이다. 따라서, 분해 및 칩 다이 검사에도 불구하고 모든 합법적 목표 장치를 위하여 온칩으로 저장된 비밀키 정보가 비밀로 유지되어야 하는 교훈에 궁극적으로 의존해야 한다.

[0282] 우리는 소정량의 시간 내에서 검증가능한 MAC 값을 정확하게 찾아내는 능력과 같은 목표 식별 및 검증 처리에 대한 필요조건을 확실하게 추가할 수 있다. 우리는 최종 MAC 값이 복수회 암호화되는 것을 요구함으로써 절차를 더 어렵게 할 수 있다. 따라서, 우리는 라이센스 자체의 합법적 복사본을 단순히 구매하는 비용보다 일반적으로 훨씬 더 비싼 (더 일반적인) 연산 리소스에 액세스할 것이 요구된다는 점에서 합법적 장치를 위조하는 공격자의 능력을 잠재적으로 제한할 수 있다. 미디어 스트림 플레이어의 경우, 플레이어가 표면상 수용하도록 설계되는 하나 이상의 미디어 스트림의 일부를 정확히 디코드하는 능력을 또한 포함할 수 있다.

[0283] 그러나, 어쨌든, 디지털 저작권 보호의 전체 과정은 튜링 문제(Turing problem)이다. 따라서, 충분한 시간 및 리소스가 주어지면, 임의의 디지털 저작권 보호 방식이 단호한 반대자에 의해 좌절될 수 있다. 이것은 물론 비밀키 정보에 대한 액세스가 명확하게 예비 공격자에 대해 큰 장점으로 된다는 사실과 무관하다. 그러므로, 유닛의 비밀키가 절충되지 않도록 지키는 능력은 이 보안 프로토콜의 중요한 부분이다.

[0284] **결론:**

[0285] 위에서 설명한 저작권 보호 프로토콜은 몇 가지 방법에서 유일한 것이다. 첫번째는 사용자가 합법적으로 구매한 애플리케이션 또는 미디어 지정 키 데이터의 백업 복사본을 만들 능력을 갖지 못하도록 시도하지 않는다는 사실이다. 둘째로, 이 프로토콜은 임의 종류의 디지털 데이터 간에 차이를 만들지 않으며, 따라서 보안 프로토콜이 보호하도록 설계된 데이터 스트림처럼 쉽게 갱신되게 한다. 셋째로, 이 프로토콜은 사용자가 자신의 애플리케이션 또는 미디어 지정 키의 소유권을 프로토콜을 실행할 수 있는 다른 유닛에 임시로 이전할 수 있게 한다. 또한, 이 프로토콜은 허가받은 자가 소유권을 하나의 목표 장치(100)로부터 다른 목표 장치로 영구 이전하는 능력을 제공한다. 상기 마지막 특성은 이 프로토콜 하에서 소비자의 법적 "최초 판매 권한"의 구현을 가능하게 한다.

[0286] 사실, 이 문서에서 설명한 프로토콜과 다른 복사 보호 방식 간의 기본적인 차이 중의 하나는 이 시스템의 보안이 특수 데이터 집합에 액세스하는 능력을 제어하는 것에 의존하는 것이 아니라 오히려 데이터 집합 내에 포함된 아이디어를 표시하는 동작을 제어하는 능력에 의존한다는 것이다.

[0287] 앞의 명세서에서, 본 발명은 특정 실시예를 참조하여 설명하였다. 그러나, 이 기술에 통상의 지식을 가진 자라면 아래의 청구범위에서 규정하는 본 발명의 범위로부터 벗어나지 않고 여러 가지로 수정 및 변경이 가능하다는 것을 알 것이다. 따라서, 명세서 및 도면은 제한하는 의도라기보다는 예시적인 것으로 간주되어야 하고, 그러한 수정들은 모두 본 발명의 범위에 포함된 것으로 의도된다.

[0288] 잇점, 다른 장점 및 문제점에 대한 해법을 특정 실시예와 관련하여 위에서 설명하였다. 그러나, 잇점, 장점, 문제점에 대한 해법, 및 임의의 잇점, 장점 또는 해법을 발생시키거나 더 명백하게 하는 임의의 컴포넌트는 임의의 또는 모든 청구항의 중요한, 요구된, 또는 본질적인 특징 또는 컴포넌트로 해석되어서는 안된다.

부호의 설명

[0289] 101: 클라이언트 분배 시스템,

120: CPU 실행 유닛

140: 범용 레지스터,

104: 비밀키

170: 암호화/복호 블록,

102: 날짜/시간 레지스터

108: 공개키,

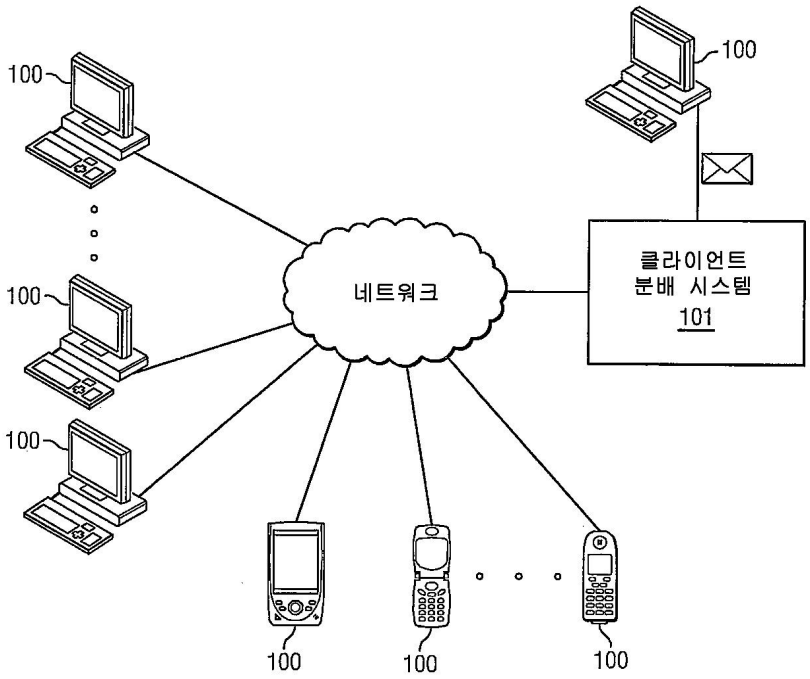
108: 개인키

180: 난수 발생기,

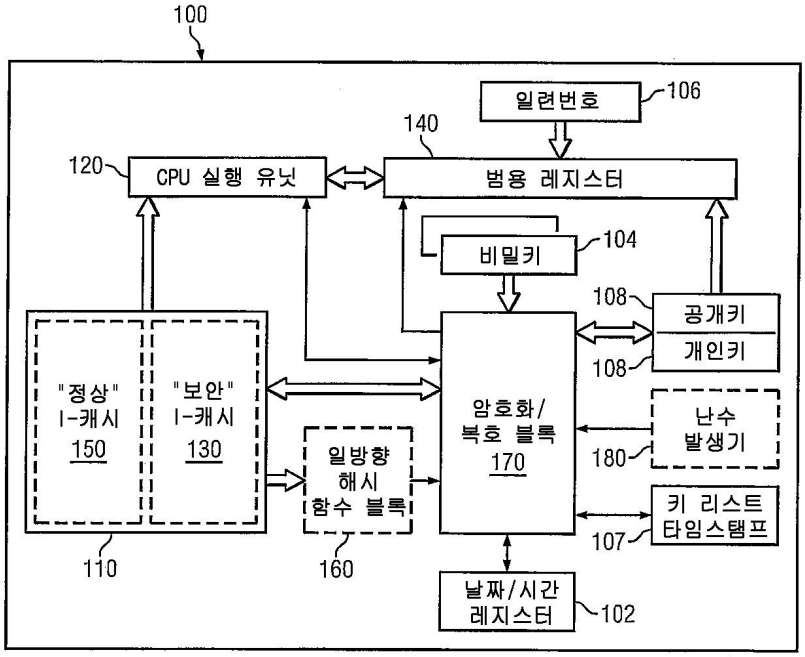
107: 키 리스트 타임스탬프

도면

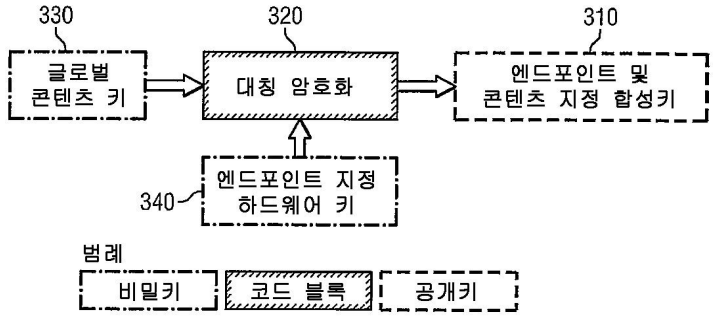
도면1



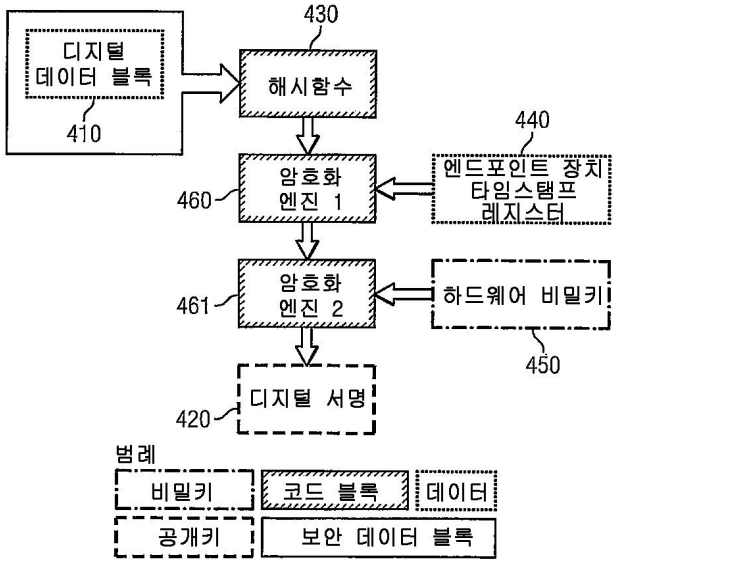
도면2


100
일련번호
106
140
120
CPU 실행 유닛
범용 레지스터
비밀키
104
108
공개키
개인키
108
"정상"
I-캐시
150
"보안"
I-캐시
130
암호화/
복호 블록
170
난수
발생기
180
일방향
해시
함수 블록
키 리스트
타임스탬프
107
110
160
날짜/시간
레지스터
102

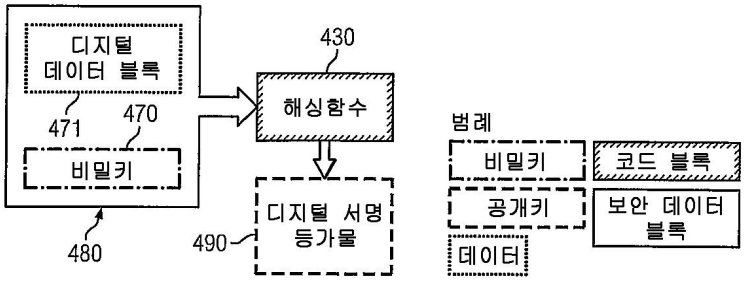
도면3


330
320
310
글로벌
콘텐츠 키
대칭 암호화
엔드포인트 및
콘텐츠 지정 합성키
엔드포인트 지정
하드웨어 키
340
범례
비밀키
코드 블록
공개키

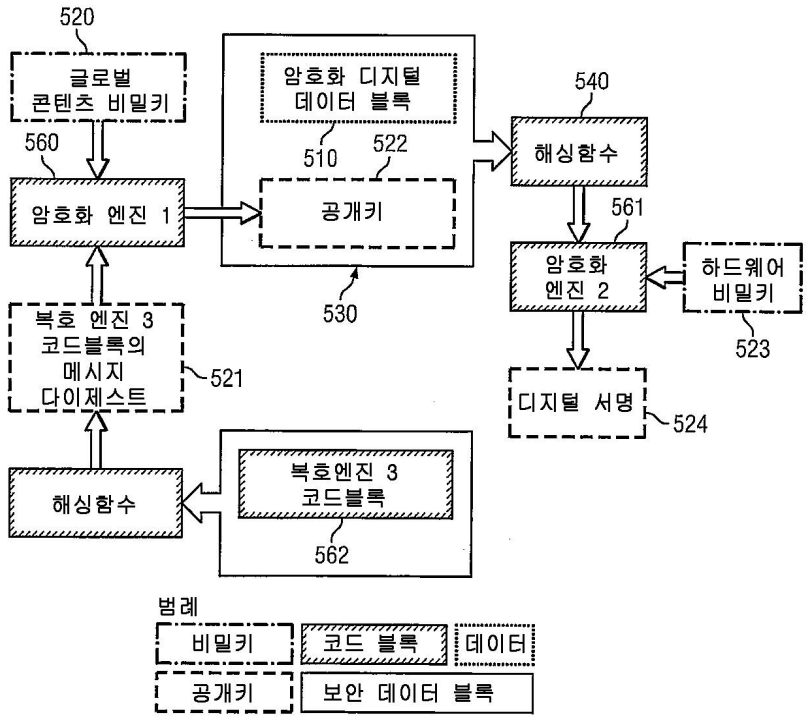
도면4a


디지털
데이터 블록
410
430
해시함수
440
엔드포인트 장치
타임스탬프
레지스터
460
암호화
엔진 1
461
암호화
엔진 2
하드웨어 비밀키
450
420
디지털 서명
범례
비밀키
코드 블록
데이터
공개키
보안 데이터 블록

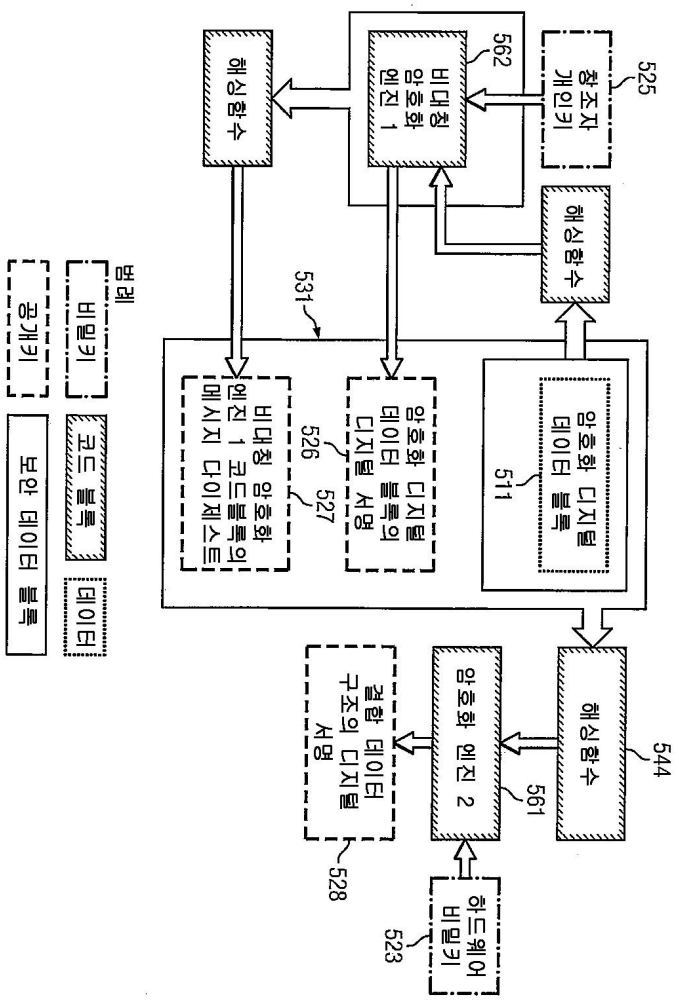
도면4b


디지털
데이터 블록
471
470
비밀키
480
430
해싱함수
490
디지털 서명
등가물
범례
비밀키
코드 블록
공개키
보안 데이터
블록
데이터

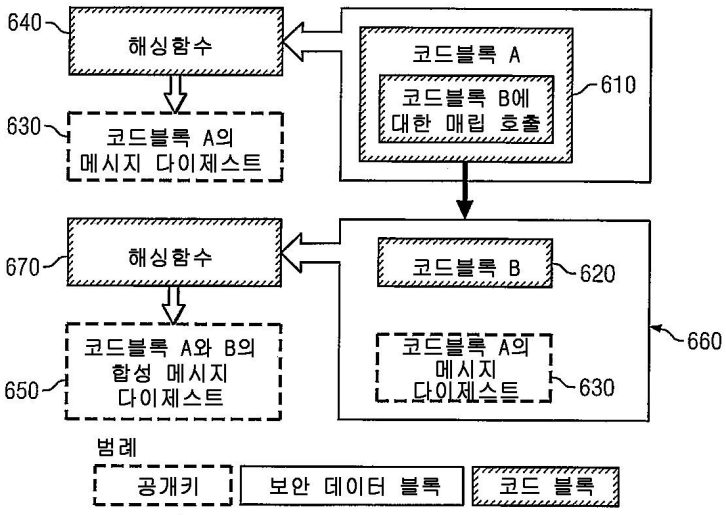
도면5a


520
글로벌
콘텐츠 비밀키
560
암호화 엔진 1
암호화 디지털
데이터 블록
510
522
공개키
530
540
해싱함수
561
암호화
엔진 2
하드웨어
비밀키
523
디지털 서명
524
복호 엔진 3
코드블록의
메시지
다이제스트
521
해싱함수
복호엔진 3
코드블록
562
범례
비밀키
코드 블록
데이터
공개키
보안 데이터 블록

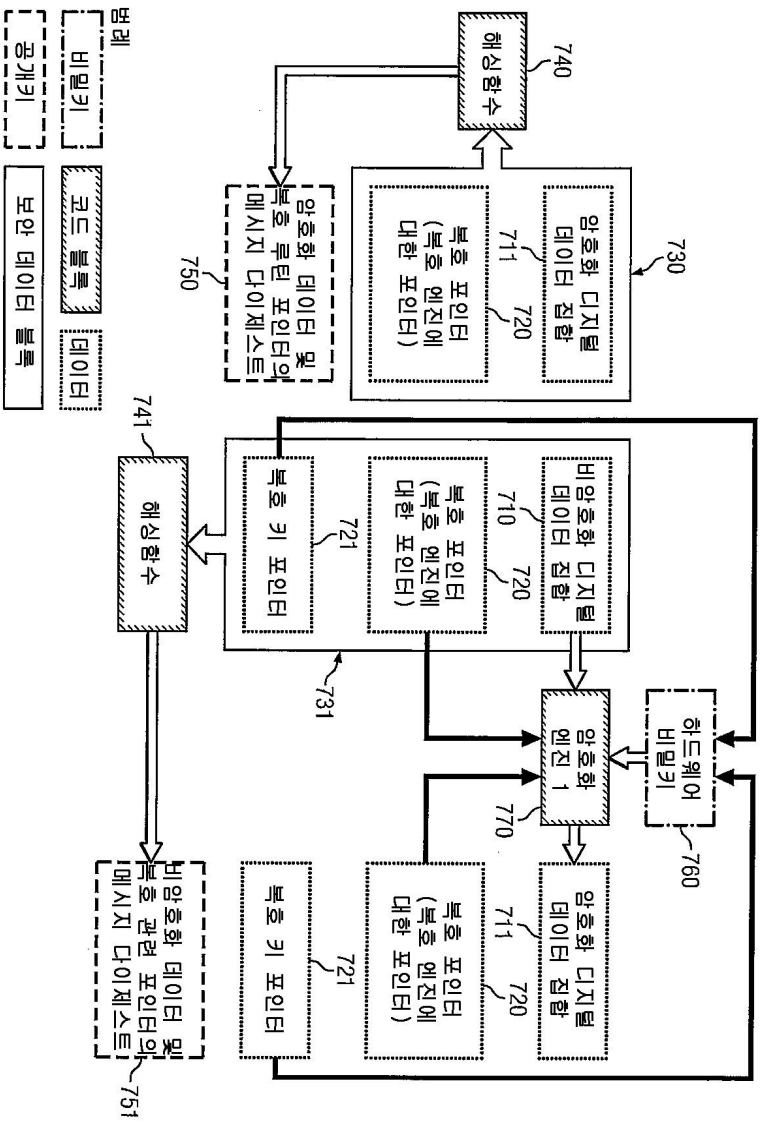
도면5b


해싱함수
비대칭 암호화 엔진 1
562
참조자 개인키
525
해싱함수
531
비대칭 암호화 엔진 1 코드블록의 메시지 다이제스트
526
암호화 디지털 데이터 블록의 디지털 서명
527
암호화 디지털 데이터 블록
511
범례
공개키
비밀키
코드 블록
보안 데이터 블록
데이터
결합 데이터 구조의 디지털 서명
528
암호화 엔진 2
561
해싱함수
544
하드웨어 비밀키
523

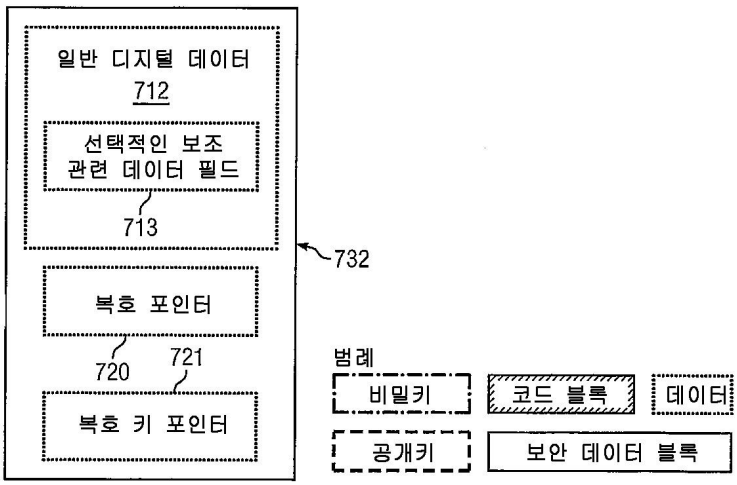
도면6


640
해싱함수
630
코드블록 A의 메시지 다이제스트
코드블록 A
코드블록 B에 대한 매립 호출
610
670
해싱함수
650
코드블록 A와 B의 합성 메시지 다이제스트
코드블록 B
620
코드블록 A의 메시지 다이제스트
630
660
범례
공개키
보안 데이터 블록
코드 블록

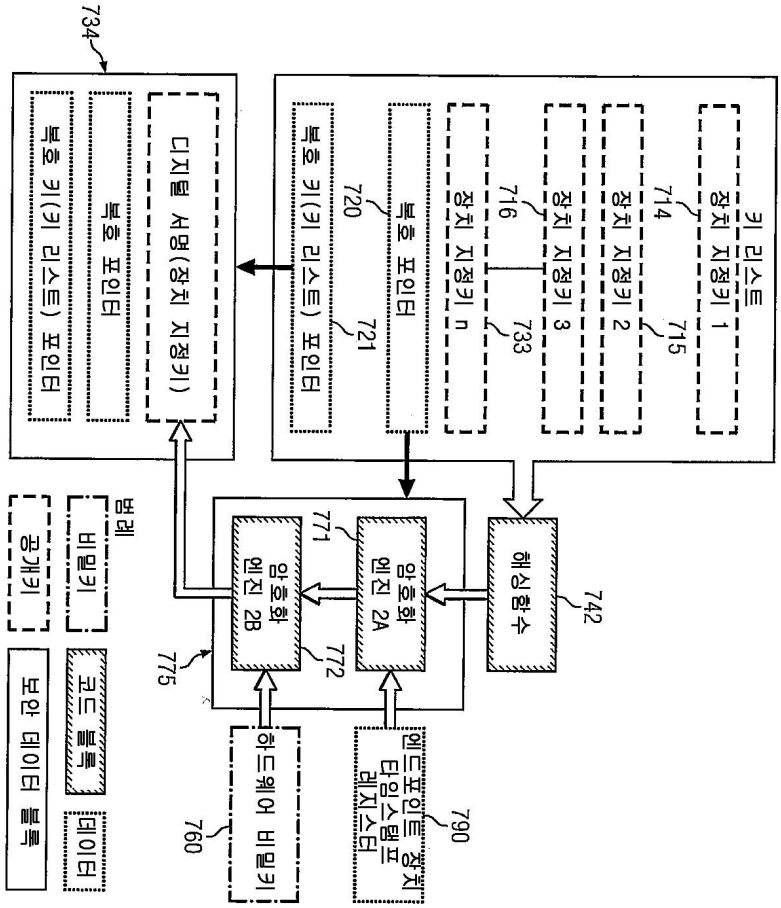
도면7a


범례
공개키
비밀키
보안 데이터 블록
코드 블록
데이터
해싱함수
740
암호화 데이터 및 복호 루틴 포인터의 메시지 다이제스트
750
암호화 디지털 데이터 집합
730
복호 포인터 (복호 엔진에 대한 포인터)
711
720
해싱함수
741
복호 키 포인터
721
비암호화 디지털 데이터 집합
710
731
암호화 엔진 1
770
하드웨어 비밀키
760
비암호화 데이터 및 복호 관련 포인터의 메시지 다이제스트
751

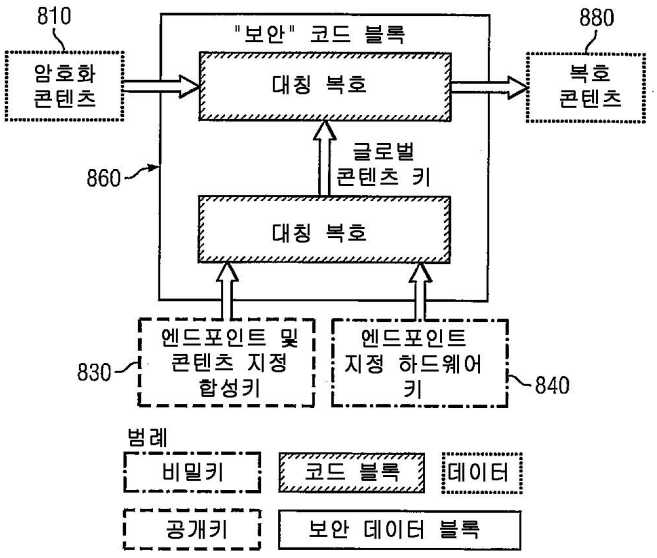
도면7b


일반 디지털 데이터
712
선택적인 보조 관련 데이터 필드
713
732
복호 포인터
720
721
복호 키 포인터
범례
비밀키
코드 블록
데이터
공개키
보안 데이터 블록

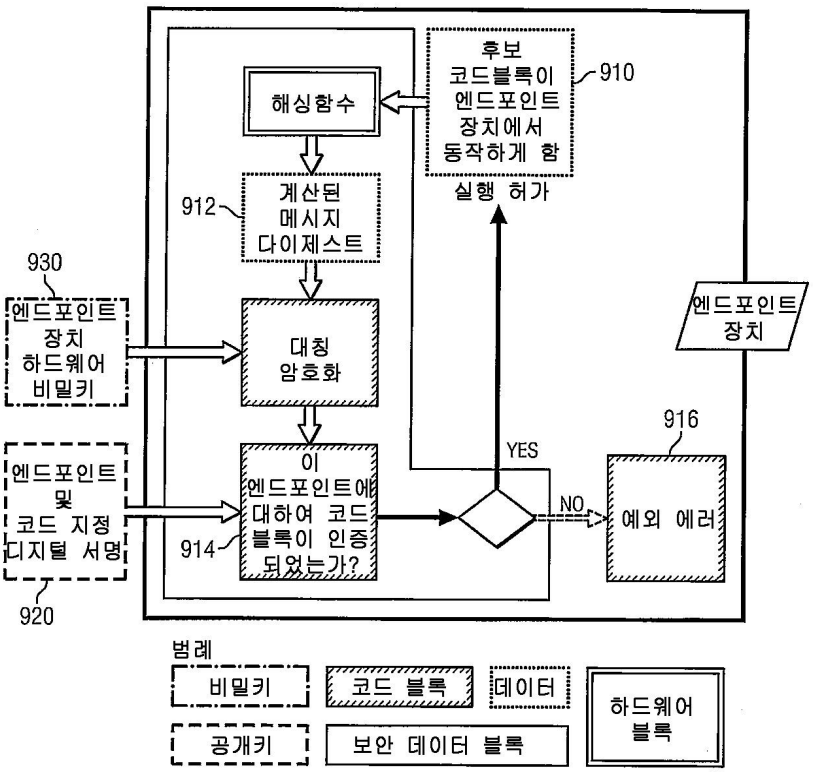
도면7c


734
디지털 서명(장치 지정키)
복호 포인터
복호 키(키 리스트) 포인터
720
721
복호 포인터
복호 키(키 리스트) 포인터
716
733
장치 지정키 n
장치 지정키 3
714
715
장치 지정키 2
장치 지정키 1
키 리스트
771
암호화 엔진 2B
암호화 엔진 2A
772
775
해싱함수
742
범례
비밀키
공개키
코드 블록
데이터
보안 데이터 블록
하드웨어 비밀키
760
엔드포인트 장치 타임스탬프 레지스터
790

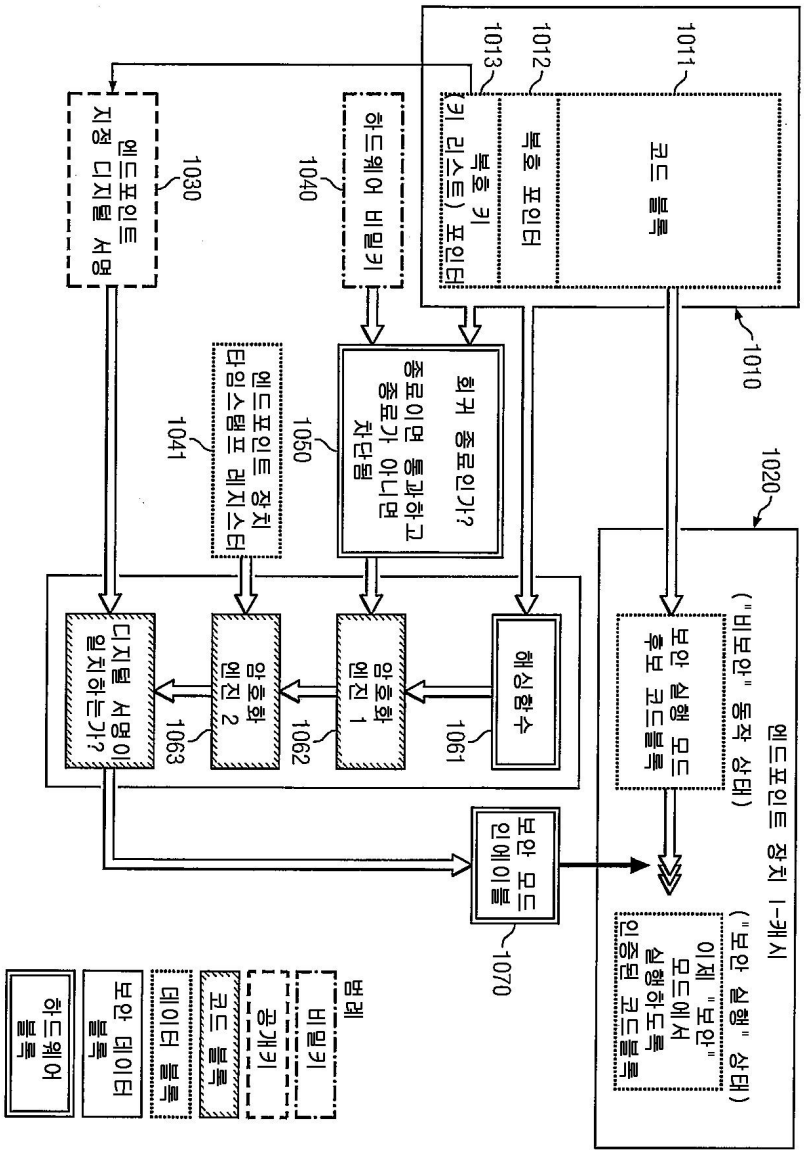
도면8


810
암호화 콘텐츠
"보안" 코드 블록
대칭 복호
880
복호 콘텐츠
860
글로벌 콘텐츠 키
대칭 복호
830
엔드포인트 및 콘텐츠 지정 합성키
엔드포인트 지정 하드웨어 키
840
범례
비밀키
코드 블록
데이터
공개키
보안 데이터 블록

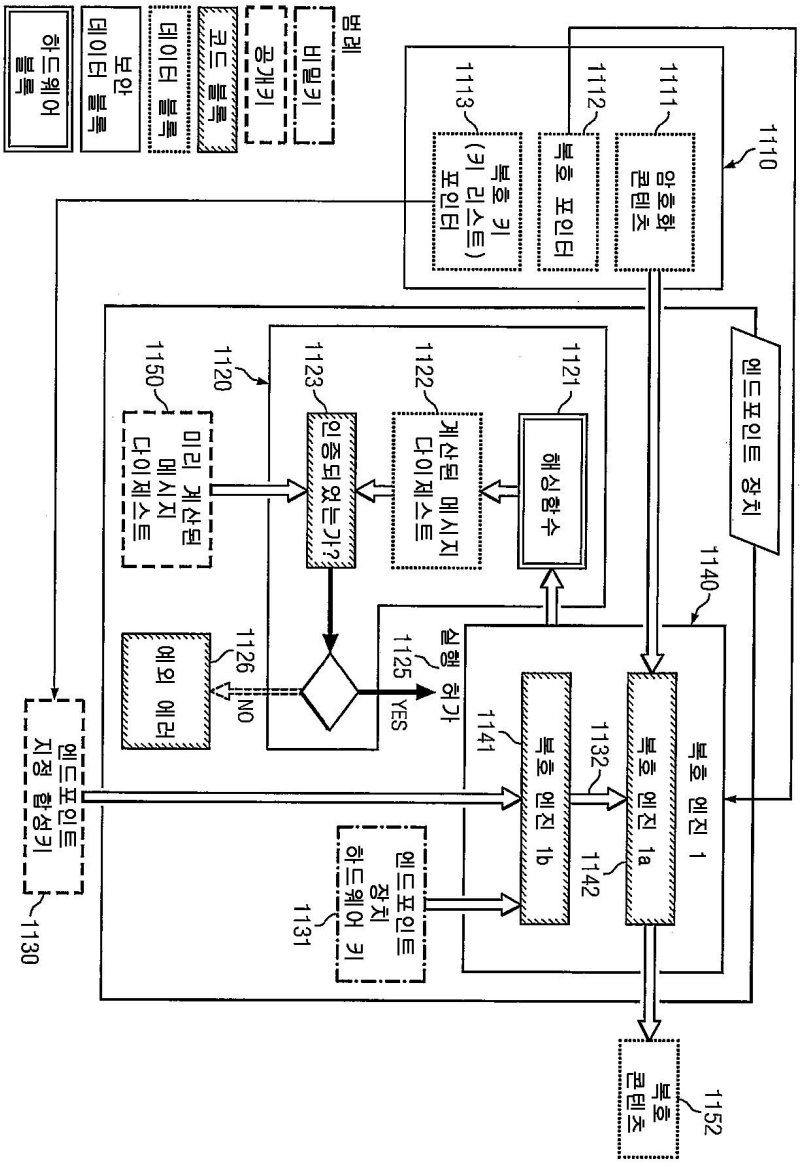
도면9


후보
코드블록이
엔드포인트
장치에서
동작하게 함
910
실행 허가
해싱함수
912
계산된
메시지
다이제스트
930
엔드포인트
장치
하드웨어
비밀키
대칭
암호화
엔드포인트
장치
916
엔드포인트
및
코드 지정
디지털 서명
914
이
엔드포인트에
대하여 코드
블록이 인증
되었는가?
YES
NO
예외 에러
920
범례
비밀키
코드 블록
데이터
하드웨어
블록
공개키
보안 데이터 블록

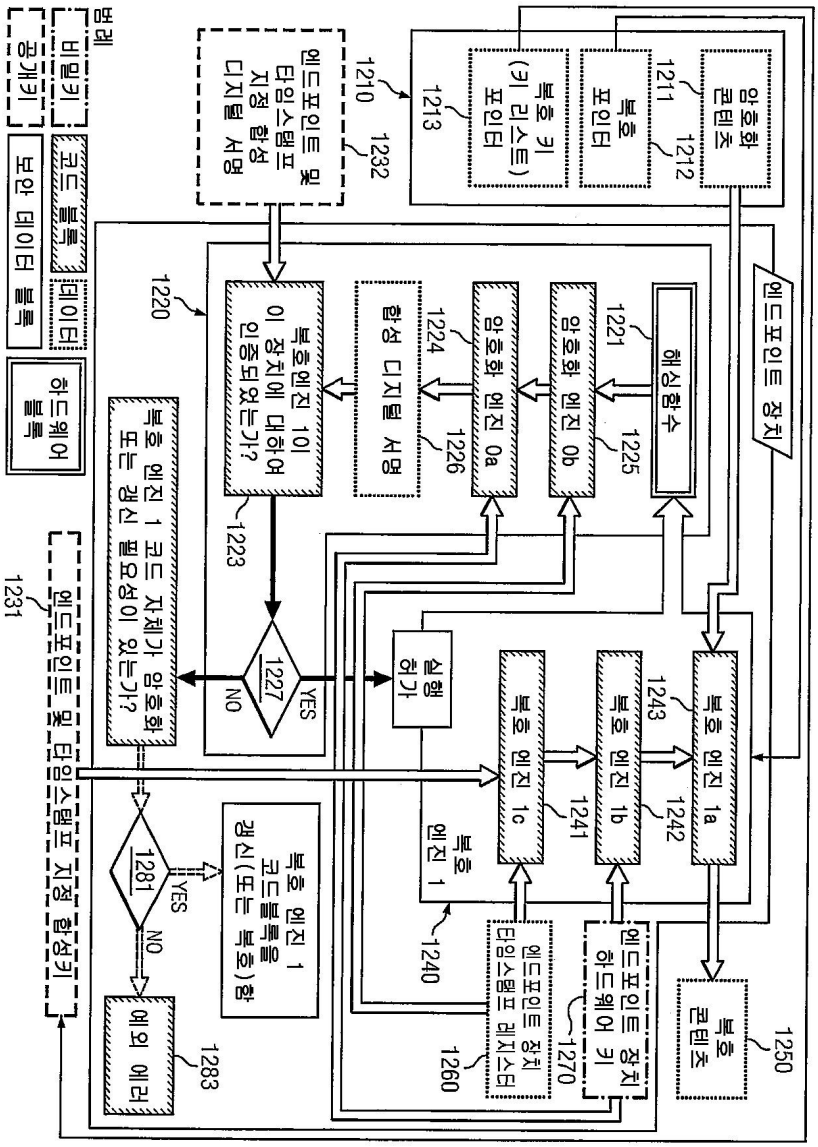
도면10


1011
코드 블록
1012
복호 포인터
1013
복호 키 (키 리스트) 포인터
1010
1020
엔드포인트 장치 I-캐시
("비보안" 동작 상태)
보안 실행 모드 후보 코드블록
("보안 실행" 상태)
이제 "보안" 모드에서 실행하도록 인증된 코드블록
1030
엔드포인트 지정 디지털 서명
1040
하드웨어 비밀키
1041
엔드포인트 장치 타임스탬프 레지스터
1050
회귀 종료인가? 종료이면 통과하고 종료가 아니면 차단됨
1061
해싱함수
1062
암호화 엔진 1
1063
암호화 엔진 2
디지털 서명이 일치하는가?
1070
보안 모드 인에이블
범례
비밀키
공개키
코드 블록
데이터 블록
보안 데이터 블록
하드웨어 블록

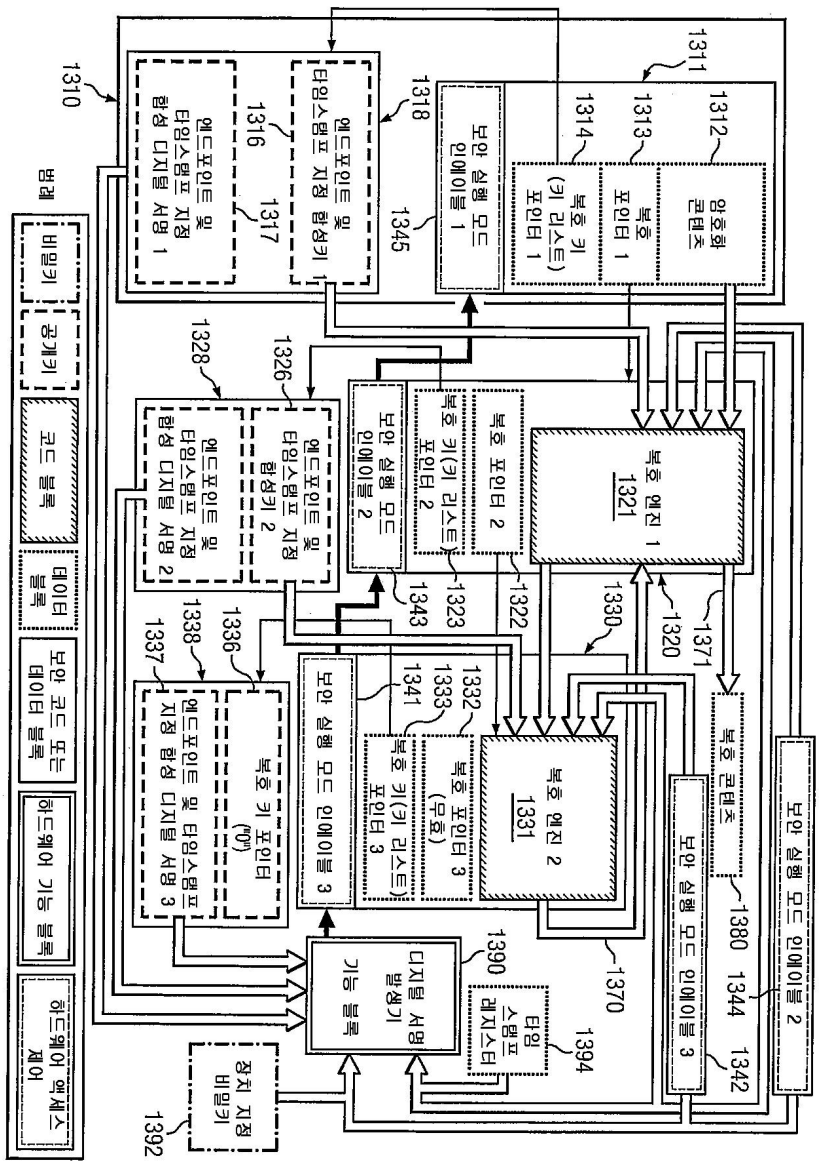
도면11


범례
비밀키
공개키
코드 블록
데이터 블록
보안 데이터 블록
하드웨어 블록
1110
1111
암호화 콘텐츠
1112
복호 포인터
1113
복호 키 (키 리스트) 포인터
엔드포인트 장치
1140
복호 엔진 1
1132
복호 엔진 1a
1142
복호 엔진 1b
1141
1120
1121
해싱 함수
1122
계산된 메시지 다이제스트
1123
인증되었는가?
1150
미리 계산된 메시지 다이제스트
1126
NO
예외 에러
1125
YES
실행 허가
1130
엔드포인트 지정 합성키
1131
엔드포인트 장치 하드웨어 키
1152
복호 콘텐츠

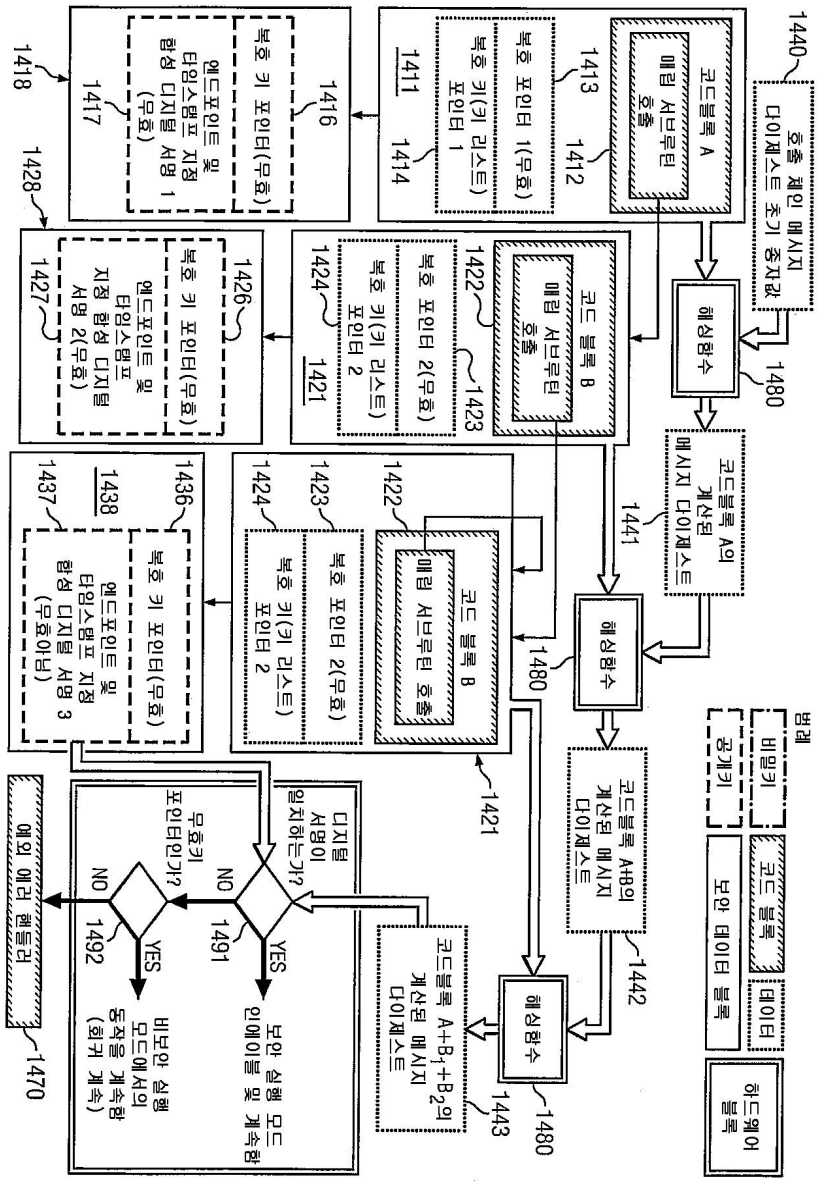
도면12


범례
비밀키
코드 블록
공개키
데이터 블록
보안 데이터 블록
하드웨어 블록
엔드포인트 장치
암호화 콘텐츠
1211
복호 포인터
1212
복호 키 (키 리스트) 포인터
1213
엔드포인트 및 타임스탬프 지정 합성 디지털 서명
1210
1232
1220
복호엔진 1이 이 장치에 대하여 인증되었는가?
1223
합성 디지털 서명
1224
암호화 엔진 0a
1226
암호화 엔진 0b
1225
해싱함수
1221
복호 엔진 1a
1243
복호 엔진 1b
1242
복호 엔진 1c
1241
복호 엔진 1
1240
실행 허가
1227
YES
NO
복호 엔진 1 코드 자체가 암호화 또는 갱신 필요성이 있는가?
엔드포인트 및 타임스탬프 지정 합성키
1231
1281
YES
NO
복호 엔진 1 코드블록을 갱신(또는 복호)함
예외 에러
1283
엔드포인트 장치 타임스탬프 레지스터
1260
엔드포인트 장치 하드웨어 키
1270
복호 콘텐츠
1250

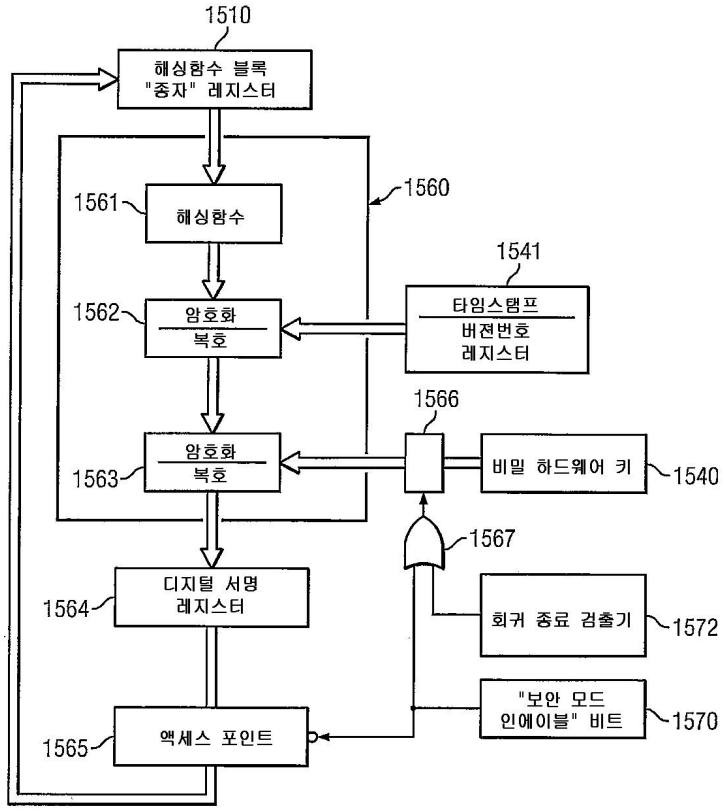
도면13


1310
범례
비밀키
공개키
코드 블록
데이터 블록
보안 코드 또는 데이터 블록
하드웨어 기능 블록
하드웨어 엑세스 제어
1311
1312
암호화 콘텐츠
1313
복호 포인터 1
1314
복호 키 (키 리스트) 포인터 1
보안 실행 모드 인에이블 1
1318
1345
엔드포인트 및 타임스탬프 지정 합성키 1
1316
1317
엔드포인트 및 타임스탬프 지정 합성 디지털 서명 1
1328
1326
엔드포인트 및 타임스탬프 지정 합성키 2
엔드포인트 및 타임스탬프 지정 합성 디지털 서명 2
보안 실행 모드 인에이블 2
복호 키(키 리스트) 포인터 2
복호 포인터 2
복호 엔진 1
1321
1323
1322
1343
1330
1320
1371
1337
1338
1336
1332
1333
1341
보안 실행 모드 인에이블 3
복호 키 포인터 (PP)
엔드포인트 및 타임스탬프 지정 합성 디지털 서명 3
복호 키(키 리스트) 포인터 3 (무효)
복호 엔진 2
1331
복호 콘텐츠
보안 실행 모드 인에이블 2
1380
1344
1342
보안 실행 모드 인에이블 3
1390
디지털 서명 발생기 기능 블록
1370
타임스탬프 메지스터
1394
장치 지정 비밀키
1392

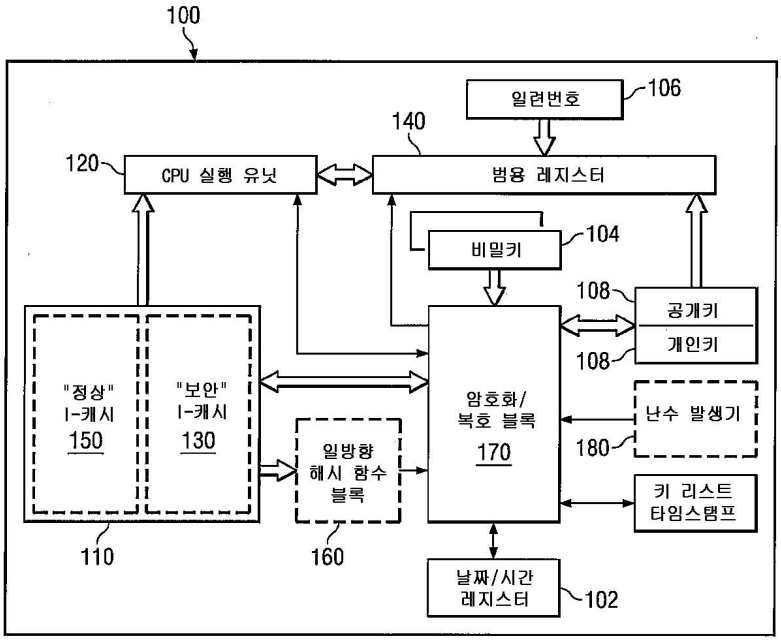
도면14


범례
비밀키
공개키
코드 블록
보안 데이터 블록
데이터
하드웨어 블록
1440
호출 체인 메시지 다이제스트 초기 중자값
코드블록 A
메립 서브루틴 호출
1413
1412
복호 포인터 1(무효)
복호 키(키 리스트) 포인터 1
1411
1414
1416
복호 키 포인터(무효)
엔드포인트 및 타임스탬프 지정 합성 디지털 서명 1 (무효)
1417
1418
1428
해싱함수
1480
코드블록 A의 계산된 메시지 다이제스트
1441
코드 블록 B
메립 서브루틴 호출
1422
1423
복호 포인터 2(무효)
복호 키(키 리스트) 포인터 2
1424
1421
1426
복호 키 포인터(무효)
엔드포인트 및 타임스탬프 지정 합성 디지털 서명 2(무효)
1427
해싱함수
1480
코드블록 A+B의 계산된 메시지 다이제스트
1442
코드 블록 B
메립 서브루틴 호출
1422
1423
복호 포인터 2(무효)
복호 키(키 리스트) 포인터 2
1424
1421
1436
복호 키 포인터(무효)
엔드포인트 및 타임스탬프 지정 합성 디지털 서명 3 (무효아님)
1438
1437
해싱함수
1480
코드블록 A+B1+B2의 계산된 메시지 다이제스트
1443
디지털 서명이 일치하는가?
YES
보안 실행 모드 인에이블 및 계속함
NO
1491
무효키 포인터인가?
YES
비보안 실행 모드에서의 동작을 계속함 (회귀 계속)
NO
1492
예외 에러 핸들러
1470

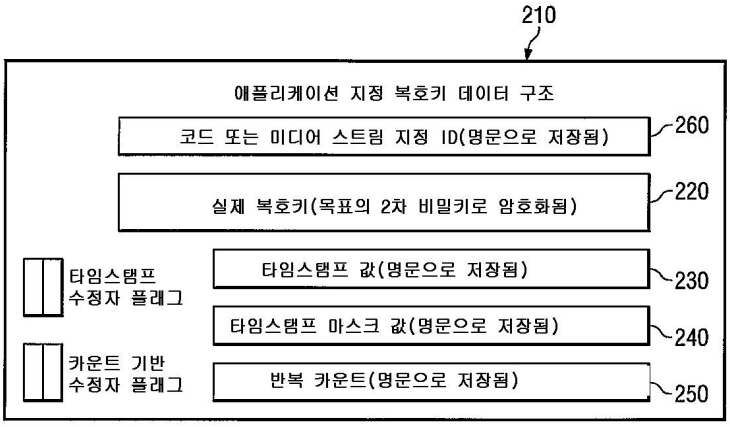
도면15


1510
해싱함수 블록
"종자" 레지스터
1560
1561
해싱함수
1541
타임스탬프
버전번호
레지스터
1562
암호화
복호
1566
암호화
복호
1563
비밀 하드웨어 키
1540
1567
디지털 서명
레지스터
1564
회귀 종료 검출기
1572
"보안 모드
인에이블" 비트
1570
액세스 포인트
1565

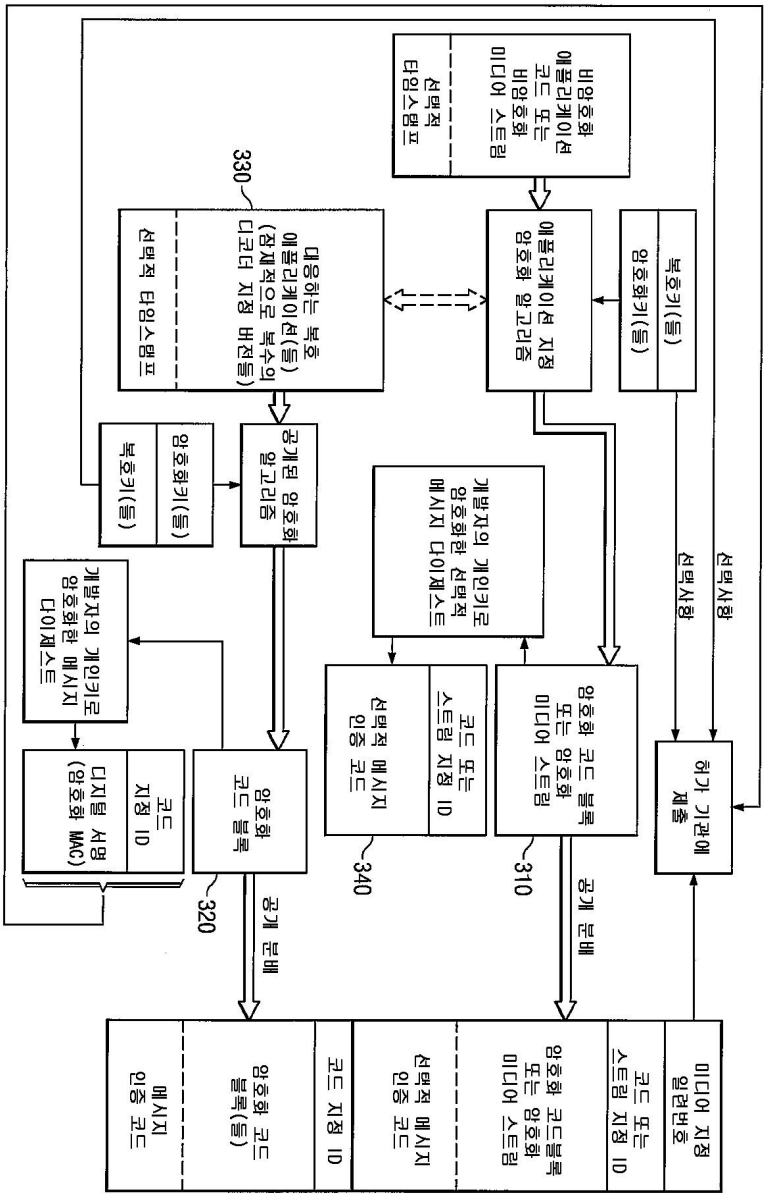
도면161


100
일련번호
106
140
120
CPU 실행 유닛
범용 레지스터
비밀키
104
108
공개키
개인키
108
"정상"
I-캐시
150
"보안"
I-캐시
130
암호화/
복호 블록
170
난수 발생기
180
일방향
해시 함수
블록
키 리스트
타임스탬프
110
160
날짜/시간
레지스터
102

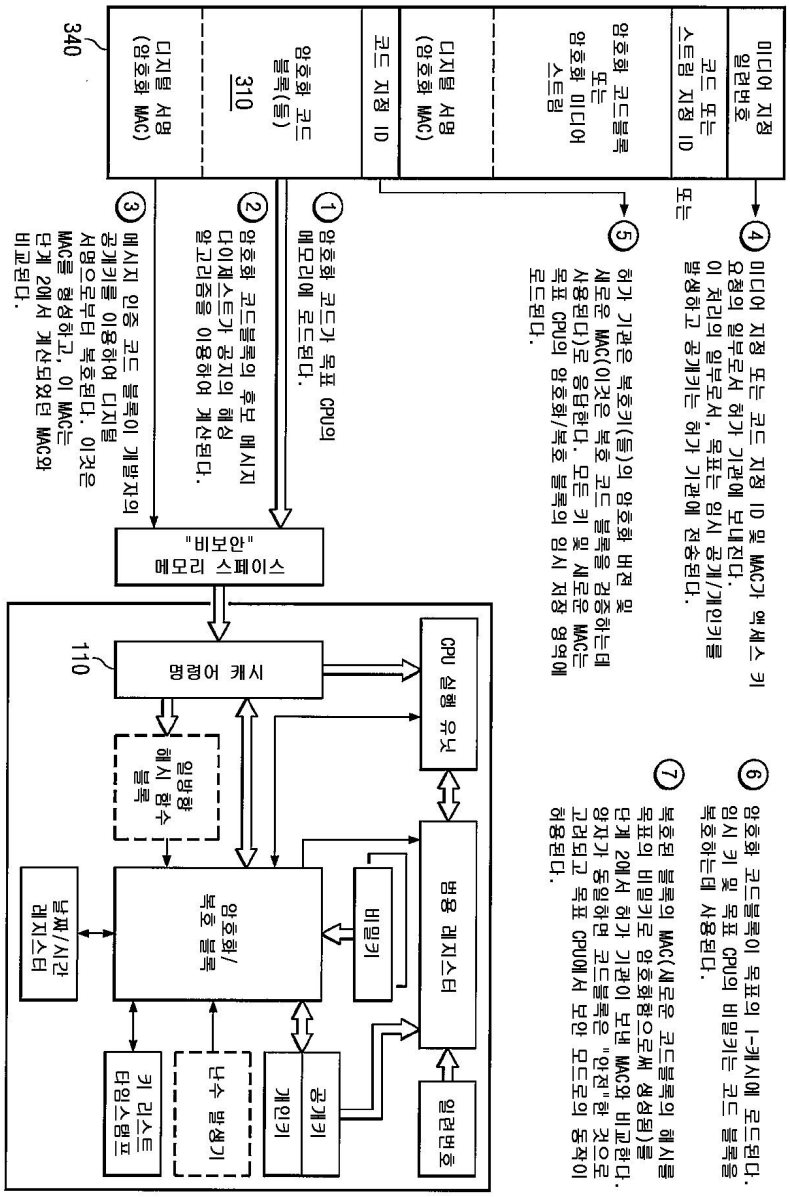
도면162


210
애플리케이션 지정 복호키 데이터 구조
코드 또는 미디어 스트림 지정 ID(명문으로 저장됨)
260
실제 복호키(목표의 2차 비밀키로 암호화됨)
220
타임스탬프 수정자 플래그
타임스탬프 값(명문으로 저장됨)
230
타임스탬프 마스크 값(명문으로 저장됨)
240
카운트 기반 수정자 플래그
반복 카운트(명문으로 저장됨)
250

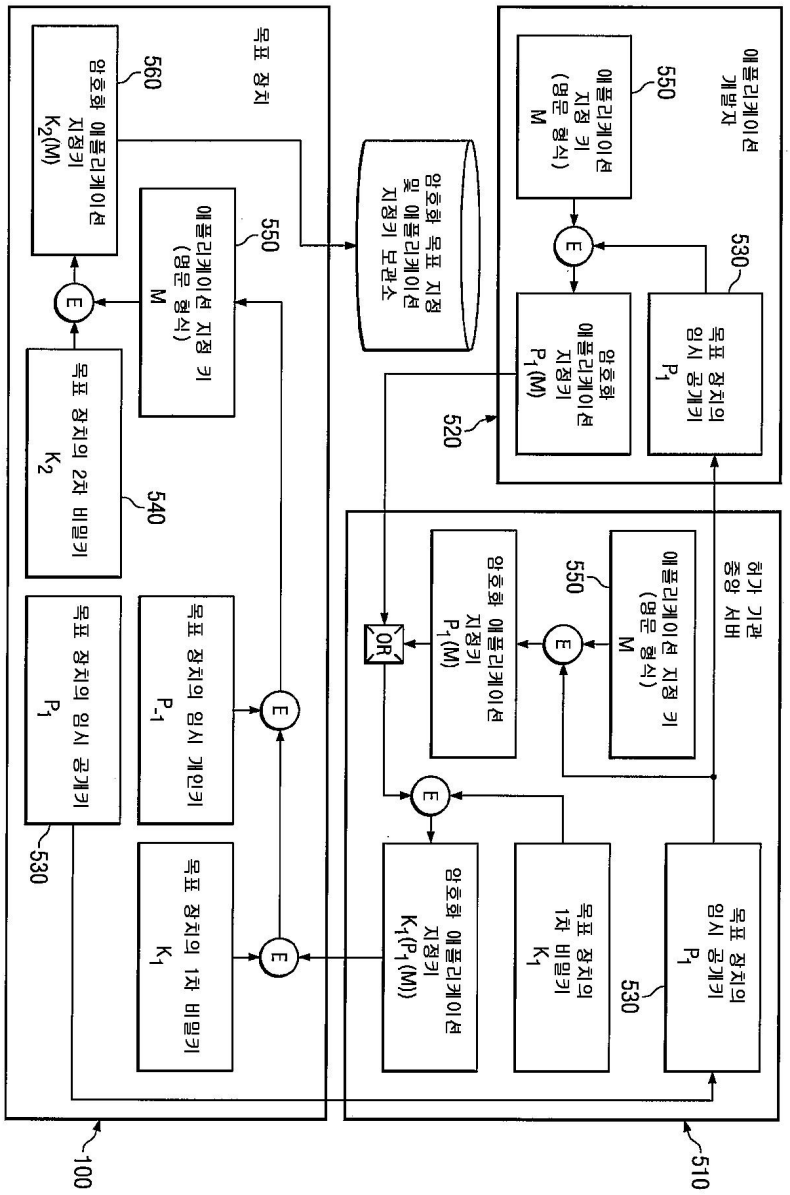
도면163


비암호화 애플리케이션 코드 또는 비암호화 미디어 스트림
선택적 타임스탬프
330
대응하는 복호 애플리케이션(들) (잠재적으로 복수의 디코더 지정 버전들)
선택적 타임스탬프
애플리케이션 지정 암호화 알고리즘
복호키(들)
암호화키(들)
선택사항
복호키(들)
암호화키(들)
공개된 암호화 알고리즘
개발자의 개인키로 암호화한 선택적 메시지 다이제스트
개발자의 개인키로 암호화한 메시지 다이제스트
디지털 서명 (암호화 MAC)
코드 지정 ID
암호화 코드 블록
320
선택적 메시지 인증 코드
코드 또는 스트림 지정 ID
340
암호화 코드 블록 또는 암호화 미디어 스트림
310
허가 기관에 제출
공개 분배
공개 분배
메시지 인증 코드
암호화 코드 블록(들)
코드 지정 ID
선택적 메시지 인증 코드
암호화 코드블록 또는 암호화 미디어 스트림
코드 또는 스트림 지정 ID
미디어 지정 일련번호

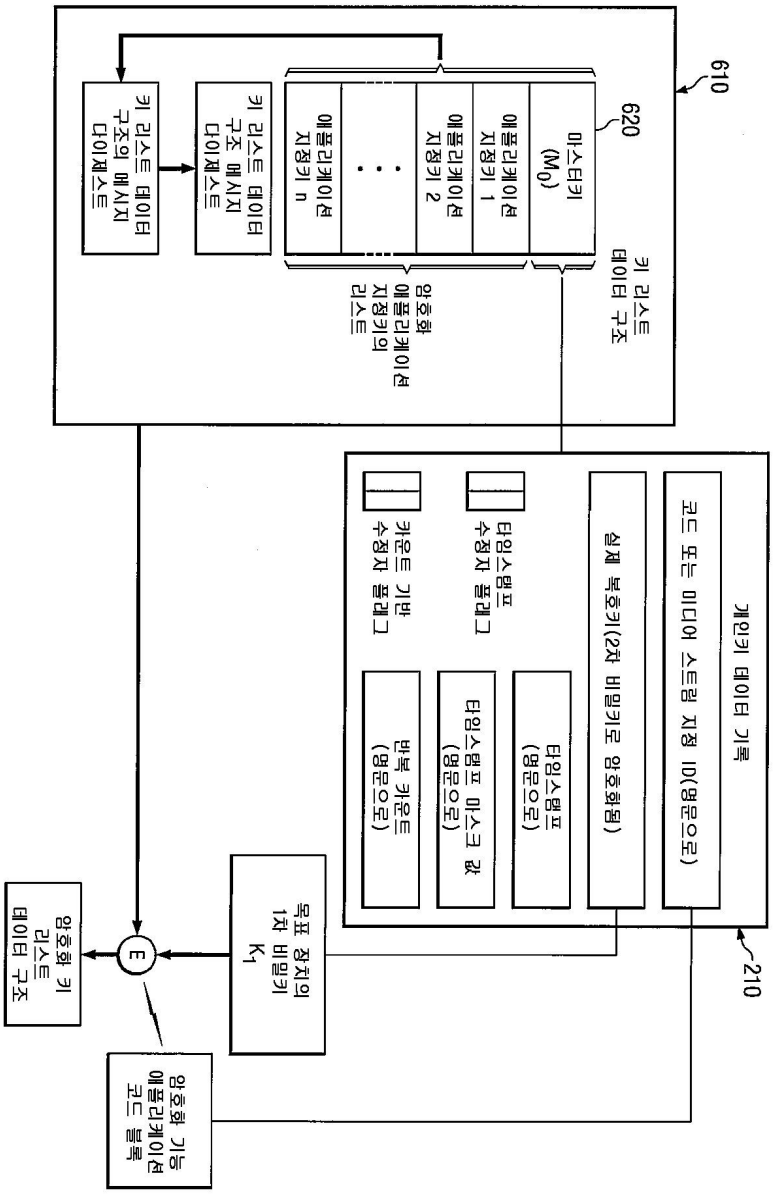
도면164


340
미디어 지정 일련번호
코드 또는 스트림 지정 ID
또는
암호화 코드블록 또는 암호화 미디어 스트림
디지털 서명 (암호화 MAC)
코드 지정 ID
디지털 서명 (암호화 MAC)
암호화 코드 블록(들) 310
① 암호화 코드가 목표 CPU의 메모리에 로드된다.
② 암호화 코드블록의 후보 메시지 다이제스트가 공지의 해싱 알고리즘을 이용하여 계산된다.
③ 메시지 인증 코드 블록이 개발자의 공개키를 이용하여 디지털 서명으로부터 복호된다. 이것은 MAC를 형성하고, 이 MAC는 단계 20에서 계산되었던 MAC와 비교된다.
④ 미디어 지정 또는 코드 지정 ID 및 MAC가 액세스 키 요청의 일부로서 허가 기관에 보내진다. 이 처리의 일부로서, 목표는 임시 공개/개인키를 발생하고 공개키는 허가 기관에 전송된다.
⑤ 허가 기관은 복호키(들)의 암호화 버전 및 새로운 MAC(이것은 복호 코드 블록을 검증하는데 사용된다)로 응답한다. 모든 키 및 새로운 MAC는 목표 CPU의 암호화/복호 블록의 임시 저장 영역에 로드된다.
⑥ 암호화 코드블록이 목표의 I-캐시에 로드된다. 임시 키 및 목표 CPU의 비밀키는 코드 블록을 복호하는데 사용된다.
⑦ 복호된 블록의 MAC(새로운 코드블록의 해시를 목표의 비밀키로 암호화함으로써 생성됨)를 단계 20에서 허가 기관이 보낸 MAC와 비교한다. 양자가 동일하면 코드블록은 "안전"한 것으로 고려되고 목표 CPU에서 보안 모드로의 동작이 허용된다.
"비보안" 메모리 스페이스
110
명령어 캐시
CPU 실행 유닛
일방향 해시 함수 블록
암호화/복호 블록
비밀키
범용 레지스터
날짜/시간 레지스터
키 리스트 타임스탬프
난수 발생기
공개키
개인키
일련번호

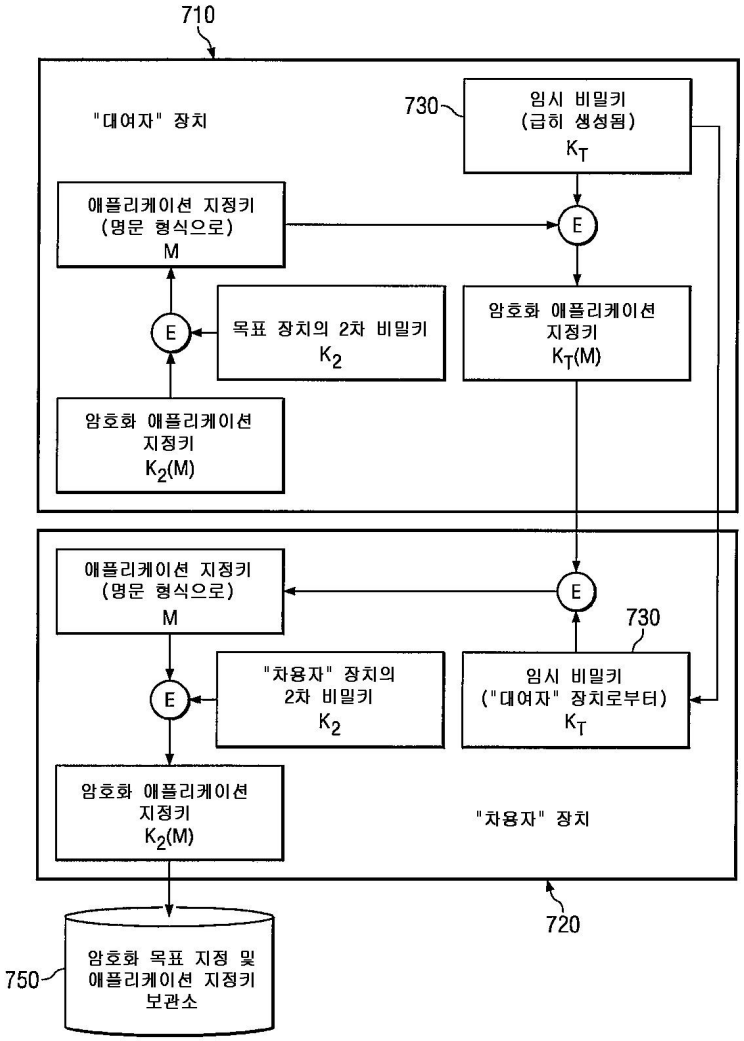
도면165


목표 장치
암호화 애플리케이션 지정키 $K_2(M)$
560
애플리케이션 지정 키 (영문 형식) M
550
목표 장치의 2차 비밀키 K_2
540
목표 장치의 임시 개인키 P_{-1}
목표 장치의 임시 공개키 P_1
530
목표 장치의 1차 비밀키 K_1
100
E
애플리케이션 개발자
애플리케이션 지정 키 (영문 형식) M
550
530
목표 장치의 임시 공개키 P_1
암호화 애플리케이션 지정키 $P_1(M)$
520
암호화 목표 지정 및 애플리케이션 지정키 보관소
허가 기관 중앙 서버
애플리케이션 지정 키 (영문 형식) M
550
암호화 애플리케이션 지정키 $P_1(M)$
OR
목표 장치의 1차 비밀키 K_1
530
목표 장치의 임시 공개키 P_1
암호화 애플리케이션 지정키 $K_1(P_1(M))$
510

도면166


610
620
마스터키 (M_0)
애플리케이션 지정키 1
애플리케이션 지정키 2
애플리케이션 지정키 n
키 리스트 데이터 구조
암호화 애플리케이션 지정키의 리스트
키 리스트 데이터 구조의 메시지 다이제스트
키 리스트 데이터 구조 메시지 다이제스트
개인키 데이터 기록
코드 또는 미디어 스트림 지정 ID(영문으로)
실제 복호키(2차 비밀키로 암호화됨)
타임스탬프 (영문으로)
타임스탬프 마스크 값 (영문으로)
반복 카운트 (영문으로)
타임스탬프 수정자 플래그
카운트 기반 수정자 플래그
210
목표 장치의 1차 비밀키 K_1
E
암호화 키 리스트 데이터 구조
암호화 기능 애플리케이션 코드 블록

도면167a


710
"대여자" 장치
730
임시 비밀키
(급히 생성됨)
K_T
애플리케이션 지정키
(명문 형식으로)
M
E
목표 장치의 2차 비밀키
K_2
암호화 애플리케이션
지정키
$K_T(M)$
암호화 애플리케이션
지정키
$K_2(M)$
애플리케이션 지정키
(명문 형식으로)
M
730
"차용자" 장치의
2차 비밀키
K_2
임시 비밀키
("대여자" 장치로부터)
K_T
암호화 애플리케이션
지정키
$K_2(M)$
"차용자" 장치
720
750
암호화 목표 지정 및
애플리케이션 지정키
보관소

도면167b

① "차용자" 장치와 "대여자" 장치는 임시 PKI 키 쌍을 이용하여 보안 통신 채널을 구성하여 안전한 임시 비밀키 교환 처리를 시행한다. "차용자" 장치는 "대여자" 장치로부터의 요청된 키들 각각에 대해 요청된 "대여 기간"과 함께 하나 이상의 애플리케이션 또는 미디어 스트림 지정키들을 요청한다.

② "대여자" 장치는 영구적 및 임시 키 리스트 데이터 구조를 통해 조사함으로써 대응하는 키(들)을 현재 소유하고 있는지를 결정한다. 소유하고 있으면, 요청된 키(들)을 짧은 기간("체크아웃 협상 타임아웃 기간") 동안 "체크아웃"된 때 그 임시 키 리스트에 저장하고, 그 다음에 이용가능한 키(들)을 포함한 암호화 리스트로 그 대응하는 "대여 만료 시간"과 함께 "차용자" 장치에 응답한다. 이 키 리스트를 암호화하기 위해 사용한 값은 "대여자" 장치에서 즉시 생성된 임시 비밀키이다. 새로운 키의 만료기간은 요청된 "대여기간"의 기간과 "차용자" 장치에 의해 허용된 최대 대여기간 중에서 더 작은 쪽에 기초를 둔다.

③ "차용자"는 그 내부 키 리스트 타임아웃 레지스터를 새로운 대여 만료 시간을 반영하도록(만일 필요하면) 갱신한다. 그 다음에, "차용자"는 암호화 키 리스트 및 협상된 "대여기간"의 수령을, 단계 2에서 수신한 암호화 키 리스트로부터 계산한 메시지 다이제스트를 돌려보냄으로써 통지한다.

④ "대여자" 장치는 요청된 키 각각에 대해 "체크아웃 기간"을 연장하고, 그 다음에 키 리스트를 인코드하기 위해 단계 2에서 사용한 임시 비밀키를 "차용자" 장치에 보낸다.

⑤ "차용자"는 단계 2에서 수신한 키 리스트 메시지를 비밀키를 이용하여 복호하고, 임시 키 리스트 데이터 구조를 갱신한다.

⑥ 각 임시 키가 만료되면, 양측의 장치는 각자의 임시 키 리스트 및 키 리스트 타임아웃 레지스터를 갱신한다.