(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2010년04월08일
(11) 등록번호 10-0951397
(24) 등록일자 2010년03월30일

(51) Int. Cl.
H04L 9/06 (2006.01) *H04L 12/28* (2006.01)
G06F 12/00 (2006.01) *G06F 15/00* (2006.01)
(21) 출원번호 10-2007-0112152
(22) 출원일자 2007년11월05일
심사청구일자 2007년11월05일
(65) 공개번호 10-2009-0046165
(43) 공개일자 2009년05월11일
(56) 선행기술조사문헌
논문(2007.07.19)*
KR1020060094171 A
KR1020060094174 A
WO03005172 A2
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
인하대학교 산학협력단
인천 남구 용현동 253 인하대학교
(72) 발명자
양대헌
서울 서초구 서초동 신동아아파트 3동 210호
최영근
경기 광명시 하안3동 주공아파트 805호 1010호
강전일
충남 공주시 옥룡동 284-1
(74) 대리인
이원희

전체 청구항 수 : 총 8 항 심사관 : 유선중

(54) 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행코드-검증 방법

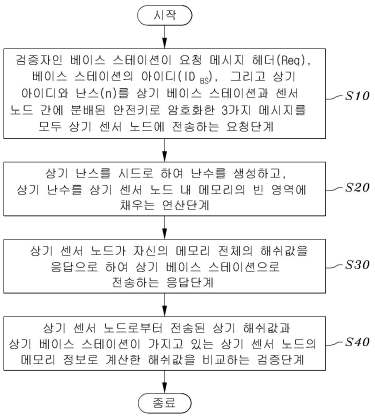
(57) 요 약

본 발명은 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법에 관한 것으로서,

검증자인 베이스 스테이션이 요청 메시지 헤더(Req), 베이스 스테이션의 아이디(ID_{BS}), 그리고 상기 아이디와 난스(n)를 상기 베이스 스테이션과 센서 노드 간에 분배된 안전키로 암호화한 3가지 메시지를 모두 상기 센서 노드에 전송하는 요청단계; 상기 난스를 시드로 하여 난수를 생성하고, 상기 난수를 상기 센서 노드 내 메모리의 빈 영역에 채우는 연산단계; 상기 센서 노드가 자신의 메모리 전체의 해쉬값을 응답으로 하여 상기 베이스 스테이션으로 전송하는 응답단계; 상기 센서 노드로부터 전송된 상기 해쉬값과 상기 베이스 스테이션이 가지고 있는 상기 센서 노드의 메모리 정보로 계산한 해쉬값을 비교하는 검증단계를 포함하여 구성되어,

매우 간단한 방식으로 동작하나 정확한 시간 정보에 의존하지 않고서도 센서 노드 내의 악의적 코드를 발견할 수 있는 효과가 있다.

대 표 도 - 도8



특허청구의 범위

청구항 1

검증자인 베이스 스테이션이 요청 메시지 헤더(Req), 베이스 스테이션의 아이디(ID_{BS}), 그리고 상기 아이디와 난스(n)를 상기 베이스 스테이션과 센서 노드 간에 분배된 안전키로 암호화한 3가지 메시지를 모두 상기 센서 노드에 전송하는 요청단계;

상기 난스를 시드로 하여 난수를 생성하고, 상기 난수를 상기 센서 노드 내 메모리의 빈 영역에 채우는 연산단계;

상기 센서 노드가 자신의 메모리 전체의 해쉬값을 응답으로 하여 상기 베이스 스테이션으로 전송하는 응답단계;

상기 센서 노드로부터 전송된 상기 해쉬값과 상기 베이스 스테이션이 가지고 있는 상기 센서 노드의 메모리 정보로 계산한 해쉬값을 비교하는 검증단계를 포함하며,

상기 요청단계는 상기 3가지 메시지를 수신한 후, 상기 암호화된 메시지를 복호화하여 상기 베이스 스테이션의 아이디와 비교하여 상기 메시지의 합법성 여부를 판단하는 과정을 더 포함하는 것을 특징으로 하는 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법.

청구항 2

삭제

청구항 3

제 1 항에 있어서,

상기 메모리의 빈 영역은 메모리 내에서 증명 코드, 펌웨어 코드, 데이터 영역을 제외한 메모리 영역인 것을 특징으로 하는 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법.

청구항 4

제 1 항에 있어서,

상기 연산단계는 상기 센서 노드 내 메모리의 빈 영역을 두 개의 난수 값의 XOR 연산으로 채우는 것을 특징으로 하는 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법.

청구항 5

제 4 항에 있어서,

상기 연산단계는 상기 센서 노드 내 메모리의 빈 영역을 두 개의 난수 값의 XOR 연산으로 채운 후, 반대 방향으로 동작하면서 기존의 난수가 있던 공간을 새롭게 생성된 난수와의 XOR 연산으로 덮어쓰는 것을 특징으로 하는 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법.

청구항 6

제 1 항에 있어서,

상기 해쉬값은 증명 코드, 펌웨어 코드와 빈 영역으로 이루어진 센서 노드의 마이크로 콘트롤러의 램(RAM) 메모리로부터 데이터 영역과 빈 영역으로 이루어진 프로그램 가능한 플래쉬 메모리까지의 모든 해쉬값인 것인 것을 특징으로 하는 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법.

청구항 7

제 1 항에 있어서,

상기 검증단계는 상기 센서 노드로부터 전송된 상기 해쉬값과 상기 베이스 스테이션이 가지고 있는 상기 센서 노드의 메모리 정보로 계산한 해쉬값을 비교한 후, 상기 두 해쉬값이 일치하지 않으면 상기 베이스 스테이션은 상기 센서 노드가 공격자에 의해 오염되거나 불법적으로 수정되었다고 판단하고, 이를 다른 센서 노드들에 공지

하는 과정을 더 포함하는 것을 특징으로 하는 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법.

청구항 8

제 1 항에 있어서,

상기 센서 노드의 응답 시간이 미리 결정된 기준 시간을 초과하는 경우, 상기 베이스 스테이션은 상기 센서 노드가 공격자에 의해 오염되거나 불법적으로 수정되었다고 판단하고, 이를 다른 센서 노드들에 공지하는 과정을 더 포함하는 것을 특징으로 하는 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법.

청구항 9

제 8 항에 있어서,

상기 미리 결정된 기준 시간은 15초인 것을 특징으로 하는 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법.

명 세 서

발명의 상세한 설명

기 술 분 야

[0001] 본 발명은 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법에 관한 것이며, 더욱 상세하게는 목표 노드의 메모리 안에 악의적 코드가 동작할 공간을 남기지 못하도록 만든 후 전체 메모리 내용의 값을 전달하는 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법에 관한 것이다.

배 경 기 술

[0002] 현재 무선 센서 네트워크에서 보안은 가장 중요한 이슈 중 하나이다. 예를 들어, 군사 감시 용도로 쓰이는 무선 센서 네트워크에서 노드가 적에 의해 포획 당하여 공격 용도로 쓰이게 된다면 이런 노드들은 적의 침입을 감지해낼 수 없고, 심지어는 아군의 패배를 초래하는 거짓 정보를 보고하게 할 수도 있다. 이러한 센서 노드들의 비정상적 행위를 막기 위해, 의심스러운 노드를 증명하는 방법이 필요하다.

[0003] 불행히도 노드에 대한 물리적 공격은 어떠한 방법으로도 막을 수 없다. 오직 노드 내부의 수정을 물리적으로 강제하는 하드웨어 모듈만이 포획 노드가 다른 노드에게 피해를 주는 악의적 노드로 변하는 것을 방지할 수 있을 뿐이다. 따라서, 지금까지의 연구는 대부분 어떻게 오염된 노드를 발견할 수 있는지에 대해 초점을 두어왔다. 대부분의 이러한 연구에선 오직 오염된 노드가 비정상적 행위를 하고 난 뒤에서야 그 오염된 노드를 발견할 수 있다. 따라서 노드의 비정상행위 전에 악의적 노드를 찾을 방법은 없었다. 이러한 이유 때문에 WSN(wireless sensor network) 은 악의적 노드에 의한 피해에 취약하였다.

[0004] 따라서, 악의적 노드가 비정상적 행위를 수행하기 전에 코드의 이상 여부를 알아낼 수 있는 사전 탐지 방법이 필요하다. SWATT(SoftWare-based Attestation for embedded device)은 이러한 소프트웨어 기반 증명을 위해 고안되었다. 상기 SWATT는 매우 간단하게 적용될 수 있지만 정확한 시간을 기반으로 동작하기 때문에 예상할 수 없는 네트워크 지연에 민감하고 하드웨어 플랫폼에 의존적이라는 한계가 있다.

[0005] 즉, 유비쿼터스 환경이 점차 그 영역을 넓혀가고 있는 현 시점에서, 그 응용 분야의 하나인 무선 센서 네트워크 환경에서 공격자가 악의적 목적으로 센서 노드 디바이스를 포획, 수정하였을 때 수반되는 피해에 대한 보안 솔루션이 절대적으로 필요하다. 그러나, 기존의 원격 검증방식은 악의적 코드를 검출하는데 있어서 연산 시간에 대한 면밀한 측정을 필요로 하며, 만약 노드가 더 강력한 마이크로프로세서를 사용한다면 연산 시간의 차이를 증폭시키기 위해서 검증의 더 많은 반복 실행이 필요하다는 문제점이 있다.

발명의 내용

해결 하고자하는 과제

[0006] 본 발명은 상기한 바와 같은 종래의 문제점을 개선하기 위한 것으로서, 목표 노드의 메모리 안에 악의적 코드

가 동작할 공간을 남기지 못하도록 만든 후 전체 메모리 내용의 값을 전달하여, 목표 노드의 전체 메모리 내용을 검증하고 SWATT와는 달리 정확한 시간 정보에 의지하지 않고서도 악의적 코드가 자신을 은폐시킬 확률을 현저히 떨어뜨릴 수 있는, 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법을 제공하는 데 있다.

과제 해결수단

[0007] 본 발명에 따른 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법은,

[0008] 검증자인 베이스 스테이션이 요청 메시지 헤더(Req), 베이스 스테이션의 아이디(ID_{BS}), 그리고 상기 아이디와 난스(n)를 상기 베이스 스테이션과 센서 노드 간에 분배된 안전키로 암호화한 3가지 메시지를 모두 상기 센서 노드에 전송하는 요청단계; 상기 난스를 시드로 하여 난수를 생성하고, 상기 난수를 상기 센서 노드 내 메모리의 빈 영역에 채우는 연산단계; 상기 센서 노드가 자신의 메모리 전체의 해쉬값을 응답으로 하여 상기 베이스 스테이션으로 전송하는 응답단계; 상기 센서 노드로부터 전송된 상기 해쉬값과 상기 베이스 스테이션이 가지고 있는 상기 센서 노드의 메모리 정보로 계산한 해쉬값을 비교하는 검증단계를 포함한다.

[0009] 또한, 상기 요청단계는 상기 3가지 메시지를 수신한 후, 상기 암호화된 메시지를 복호화하여 상기 베이스 스테이션의 아이디와 비교하여 상기 메시지의 합법성 여부를 판단하는 과정을 더 포함할 수 있다.

[0010] 또한, 상기 연산단계에서 상기 메모리의 빈 영역은 메모리 내에서 증명 코드, 펌웨어 코드, 데이터 영역을 제외한 메모리 영역이다.

[0011] 또한, 상기 연산단계는 상기 센서 노드 내 메모리의 빈 영역을 두 개의 난수 값의 XOR 연산으로 채울 수 있다. 바람직하게는, 상기 연산단계는 상기 센서 노드 내 메모리의 빈 영역을 두 개의 난수 값의 XOR 연산으로 채운 후, 반대 방향으로 동작하면서 기존의 난수가 있던 공간을 새롭게 생성된 난수와의 XOR 연산으로 덮어쓰도록 할 수 있다.

[0012] 또한, 상기 해쉬값은 증명 코드, 펌웨어 코드와 빈 영역으로 이루어진 센서 노드의 마이크로 콘트롤러의 램(RAM) 메모리로부터 데이터 영역과 빈 영역으로 이루어진 프로그램 가능한 플래쉬 메모리까지의 모든 해쉬값이다.

[0013] 또한, 상기 검증단계는 상기 센서 노드로부터 전송된 상기 해쉬값과 상기 베이스 스테이션이 가지고 있는 상기 센서 노드의 메모리 정보로 계산한 해쉬값을 비교한 후, 상기 두 해쉬값이 일치하지 않으면 상기 베이스 스테이션은 상기 센서 노드가 공격자에 의해 오염되거나 불법적으로 수정되었다고 판단하고, 이를 다른 센서 노드들에 공지하는 과정을 더 포함할 수 있다.

[0014] 또한, 상기 센서 노드의 응답 시간이 미리 결정된 기준 시간을 초과하는 경우, 상기 베이스 스테이션은 상기 센서 노드가 공격자에 의해 오염되거나 불법적으로 수정되었다고 판단하고, 이를 다른 센서 노드들에 공지하는 과정을 더 포함할 수 있으며, 이때 상기 미리 결정된 기준 시간은 15초인 것이 바람직하다.

효 과

[0015] 상기한 바와 같은 구성으로 이루어진 본 발명에 따른 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법에 의하면, 매우 간단한 방식으로 동작하나 정확한 시간 정보에 의존하지 않고서도 센서 노드 내의 악의적 코드를 발견할 수 있는 효과가 있다.

발명의 실시를 위한 구체적인 내용

[0016] 이하, 첨부된 도면을 참조하여 본 발명의 실시예를 상세히 설명한다. 우선, 도면들 중 동일한 구성요소 또는 부품들은 가능한 한 동일한 참조부호를 나타내고 있음에 유의해야 한다. 본 발명을 설명함에 있어서 관련된 공지기능 혹은 구성에 대한 구체적인 설명은 본 발명의 요지를 모호하게 하지 않기 위해 생략한다.

[0017] 도 1은 본 발명에서 사용되는 표기에 대한 설명을 도시한 도, 도 2는 센서 노드의 메모리를 개념적으로 도시한 도로서, 여기서 Mem^1은 마이크로 콘트롤러의 RAM, Mem^i는 프로그램 가능한 플래시 메모리를 도시한 도이다. 도 3a 내지 도 3c는 본 발명의 연산단계에서 목표 노드의 빈 메모리 공간을 채우는 과정을 도식적으로 도시한 도, 도 4는 상기 도 3a 내지 도 3c의 과정을 실행하기 위한 Fill_Memory(n) 알고리즘을 도시한 도, 도 5는 SHA-1을 사용하여 MIXED_MEM으로 표현한 혼합화의 일 예를 도시한 도, 도 6은 Fill_Memory(n) 알고리즘 전후의 메모리

내용 변화를 개념적으로 도시한 도, 도 7은 본 발명에 따른 선행 코드-검증 방법을 도식적으로 도시한 도, 도 8은 본 발명에 따른 선행 코드-검증 방법을 도시한 순서도, 도 9는 정상적 검증 방법의 총 연산시간과 공격코드의 총 연산시간의 차를 도시한 그래프이다.

[0018] 먼저, 본 명세서에서는 검증자가 목표인 센서 노드(이하, 목표 노드)의 메모리 내용의 사본을 가지고 있다고 가정한다. 각각의 엔티티는 서로 간의 안전한 통신 채널을 구축하기 위해 키 쌍을 분배한다. 이 키 쌍은 무작위 키 선-분배 방법 등에 의해 설정될 수 있다. 그리고, 검증자가 원격으로 실행 가능한 증명코드가 목표 노드의 메모리 안에 탑재되어 있다고 가정한다. 공격자는 포획한 노드의 메모리 접근에 대한 모든 권한을 가지고 있다. 하지만 그 외의 하드웨어 구조에 대한 수정 가능성은 공격자가 외부 자원을 이용할 수 있다는 의미가 되므로 본 명세서에선 논외로 한다. 프로그램 가능한 플래쉬 메모리는 특별한 목적이 없는 한, 어떠한 빈 공간도 가지고 있지 않는다고 가정한다. 또한 노드는 가상 메모리를 지원하지 않으며 데이터 스토리지에서는 직접적으로 코드를 실행시킬 수 없다고 가정한다.

[0019] 검증 방법의 가장 원시적 방법은 검증자가 목표 노드의 실제 메모리 내용과 자신이 알고 있는 메모리 내용을 비교해보는 것이다. 이 방법에서 검증자는 먼저 목표 노드의 증명을 요구하는 메시지를 보낸다. 그러면 목표 노드는 자신의 메모리에 있는 증명코드를 실행한다. 이 증명코드는 노드의 메모리 내용에 접근하여 그 내용을 검증자에게 전송한다. 마지막으로, 검증자는 수신한 내용을 자신이 가지고 있는 복사본과 비교한 후 목표 노드의 감염 여부를 결정한다.

[0020] 그러나, 공격자는 다음과 같은 공격방법을 사용함으로서, 포획한 노드 내부에서 악의적으로 수정된 펌웨어 코드가 이러한 원시적인 검증 방법에 의해 발견되는 것을 쉽게 피할 수 있다. 우선, 공격자는 원래 메모리 내용에 대한 유효한 정보를 가지고 있는 공격코드를 센서 노드의 빈 공간에 심어 놓는다. 그리고 노드 내의 증명코드가 실제 메모리 내용에 접근하는 대신에 심어 놓은 유효 정보에 접근하도록 수정한다. 그러면 증명코드는 악의적으로 수정된 펌웨어의 정보가 아닌 노드가 공격받기 이전의 유효한 메모리 정보 값을 전송할 것이다. 결과적으로, 검증자는 목표 노드의 감염 여부를 정확히 검증해내지 못한다.

[0021] 이를 해결하기 위해, 본 명세서에서는 소프트웨어 기반 검증 방법에 대한 새로운 방법을 제시한다. 본 발명에 따른 방법의 기본 원리는 검증자의 요청이 있을 때마다 목표 노드의 메모리 안에 악의적 코드가 동작할 공간을 남기지 못하도록 만든 후 전체 메모리 내용의 값을 전달하는 것이다.

[0022] WSN(Wireless Sensor Network)은 일반적으로 대규모의 센서들과 몇 개의 베이스 스테이션(Base Station)으로 구성되어 있다. 베이스 스테이션은 여러 업무를 동시에 처리할 수 있을 정도의 충분한 자원을 소유하고 있는 반면에 센서 노드는 작은 메모리 크기, 적은 배터리 용량, 좁은 라디오 범위 등 매우 제한된 자원을 가지고 있다.

[0023] 본 명세서에서는 도 1에 도시된 바와 같은 표기를 사용한다.

[0024] 센서 노드는 여러 종류의 메모리를 가지고 있다. 예를 들어 UC Berkeley에 의해 개발된 MICA mote는 4kB의 고정 RAM, 128kB 프로그램 가능한 플래쉬 메모리와 4Mbit의 외부 플래쉬 메모리를 가지고 있다. 대부분의 센서 노드 구조는 MICA mote와 유사하다. 본 명세서에서는 마이크로 컨트롤러의 SRAM을 Mem^1로 표시한다. 그 외 Mem^i는 플래쉬 프로그램 메모리나 외부 데이터 저장소를 나타낸다. 보통 오직 한 개의 프로그램 가능한 플래쉬 메모리가 센서 노드의 마이크로 콘트롤러 안에 내장되고 외부의 데이터 저장소는 EEPROM 이다. 본 명세서에서는 외부 데이터 저장소를 이용한 공격은 염두 해두지 않는다. 그 이유는 외부 데이터 저장소를 공격 목적으로 사용하기에는 접근 시간이 너무 길어서 쉽게 발견할 수 있기 때문이다. 센서 노드가 배치된 후에, 증명 코드(V)와 펌웨어 코드(C)는 노드가 동작하는 동안에 Mem^1에 상주하게 된다. Mem^1은 각각의 데이터 파트(D^i)를 가지고 있으므로, 하나의 노드가 가지고 있는 총 데이터는 $D = \{D^i\}^{\Lambda}_{i=1}$이다. 여기서 Λ는 하나의 센서 노드가 가지고 있는 모든 메모리 개수를 의미한다. 도 2는 이러한 메모리를 개념적으로 도시한 도이다.

[0025] 1. 요청단계(S10)

[0026] 본 발명에 따른 선행 코드-검증 방법은, 먼저 목표 노드의 메모리 내용을 검증하기 위한 요청 단계(S10)에서, 검증자인 베이스 스테이션은 요청 메시지헤더(Req)와 베이스 스테이션의 아이디(ID_{BS}), 그리고 상기 ID_{BS}와 난스(nonce; n)를 상기 베이스 스테이션과 상기 목표 노드 간의 분배된 안전키($K_{IDBS,mode}$)로 암호화하여 이 3가지 메시지를 모두 상기 목표 노드에 전송한다.

[0027] 그 다음, 상기 목표 노드가 상기 3가지 메시지를 수신하게 상기 암호화된 메시지를 복호화하여 상기 베이스 스

테이션의 아이디와 비교하여 상기 메시지의 합법성 여부를 판단한다. 이렇게 함으로써, 공격자가 상기 분배된 안전키($K_{IDBS,mode}$)를 알지 못하게 되고, 공격자는 베이스 스테이션으로 위장할 수 없게 되어, 특정 노드에 대한 서비스 거부 공격(Denial of Service)을 방지할 수 있게 된다.

[0028] 2. 연산단계(S20)

[0029] 검증자의 요청 단계에 이어 목표 노드의 연산단계에서, 목표 노드는 난스를 시드로 하여 난수를 생성한다. 그리고 목표 노드 내 메모리의 빈 영역은 상기 생성된 난수를 이용한 특정 알고리즘의 결과값인 비트 열로 채운다(S20). Mem^i의 빈 공간의 크기인 S^i_e는 증명코드, 펌웨어 코드, 데이터 영역을 제외한 빈 메모리 영역의 크기를 의미한다. 이는 다음처럼 계산할 수 있다.

[0030] 식(1) : $S^i_e = S^i - (S^i_v + S^i_c + S^i_d)$

[0031] 본 단계에서 사용하는 알고리즘은 메모리의 빈 공간을 두 개의 난수 값의 XOR연산으로 채우는 기능을 수행하게 된다. 먼저, 난수는 S^i_e를 채운다. 그리고 이러한 난수열이 빈 메모리 공간을 다 채우게 되면 반대 방향으로 동작하면서 기존의 난수가 있던 공간을 새롭게 생성된 난수와의 XOR 연산으로 덮어쓰게 된다.

[0032] 상기 알고리즘에서 사용하는 해쉬 함수의 입력 값은 이전에 채워진 512-bits의 메모리 값을 사용한다. 도 3a 내지 도 3c에 상기 알고리즘 동작과정의 일 예가 도시되어 있다. 도 3a에는 메모리의 빈 공간에 난수를 채우기 시작하는 과정이 개념적으로 도시되어 있고, 도 3b에는 난수열이 메모리 공간의 끝을 만났을 때, 반대방향으로 동작하는 과정이 도시되어 있다. 그리고, 도 3c에는 기존의 난수가 있던 공간을 새롭게 생성된 난수와의 XOR 연산으로 덮어 쓰는 과정이 도시되어 있다.

[0033] 그러나 이 알고리즘이 4번째 라운드까지 동작하기 이전에는 해시 입력을 위한 512-bits의 메모리 값이 존재하지 않으므로, 첫 번째 라운드에서의 난수 값은 단지 시드 n에서부터 생성되고, 두 번째부터 네 번째 라운드까지의 연산은 메모리 값의 빈 부분을 0으로 채운 값을 입력으로 한다. 이러한 증명단계의 알고리즘은 Fill_Memory(n)은 도 4에 도시된 바와 같다.

[0034] 3. 응답단계(S30)

[0035] 그 다음, 목표 노드의 응답단계에서 해당 노드는 자신의 메모리 전체의 해쉬값을 응답으로 하여 검증자인 베이스 스테이션에게 전송하게 된다(S30). 이 해쉬값은 Mem^1부터 Mem^i까지의 h(MIXED_MEM‖D)이다. 여기서 MIXED_MEM은 V‖C‖FMn을 특정 규칙에 의거하여 읽은 메모리 값의 열이다. 또한, 여기서 FMn은 Fill_Memory(n)을 뜻한다. 상기 특정 규칙은 연산단계에서 사용한 난수 생성기의 출력 크기만큼의 블록을 건너뛰며 읽는 방식을 순차적으로 수행한다. MIXED_MEM으로 표현한 이러한 혼합화는 공격자가 포획한 다른 노드를 이용하는 등의 방법으로 FMn 전체의 해쉬 값을 미리 사본으로 저장해 놓는 것을 방지한다. 도 5은 이 점을 설명하는 일 예가 도시되어 있다.

[0036] 베이스 스테이션은 노드가 임시적으로 보관하고 있는 데이터(D)를 모르기 때문에, 일반적으로 노드는 증명 단계 이전에 베이스 스테이션에게 데이터(D)를 보내야 한다. 그리고, 목표 노드의 메모리 안의 증명코드(V)와 펌웨어 코드(C)를 제외한 모든 공간을 난수로 채운 뒤, 메모리의 해쉬값으로 응답한다. 응답 단계가 끝나면 베이스 스테이션은 자신이 보관하고 있던 데이터를 본래의 자리인 노드에게로 되돌려줘야 한다.

[0037] 하지만 이러한 방식은 두 가지 이유로 부담이 되는데, 첫 번째 이유는 노드의 제한된 배터리 용량에 기인한다. 노드는 데이터(D)를 송신, 수신해야하기 때문에 에너지를 두 번 소비하게 된다. 결과적으로 노드는 검증을 위하여 자신이 원래 가지고 있던 데이터(D)를 지우고, 검증이 끝난 후엔 그것을 다시 얻기 위하여 불필요한 에너지 낭비를 하게 되는 것이다. 다른 이유는 불필요한 증명 시간의 증가이다.

[0038] 본 발명에 따른 방법에서는 노드는 데이터(D)를 다시 수신받아야할 필요가 없다. 응답 단계에서 노드는 단지 데이터를 해쉬값과 함께 베이스 스테이션에게 보내기만 하면 된다. 그러면 베이스 스테이션은 이미 자신이 알고 있는 목표 노드의 증명코드(V), 펌웨어 코드(C)와 함께 데이터(D)도 알게 된다.

[0039] 4. 검증단계(S40)

[0040] 마지막으로 검증자의 검증 단계에서 베이스 스테이션은 목표 노드로부터 수신한 해쉬값과 자신이 가지고 있는 목표 노드의 메모리정보로 계산한 해쉬값을 비교한다(S40). 만일 이 둘이 일치하지 않다면 베이스 스테이션은

목표 노드가 공격자에 의해 오염되었거나 불법적으로 수정되었다고 최종적으로 판단하게 되고, 다른 노드들에게 이를 공지할 것이다.

[0041] 또는, 이와는 다르게 목표 노드의 응답 시간이 미리 결정된 기준 시간을 초과하는 경우, 상기 베이스 스테이션은 상기 목표 노드가 공격자에 의해 오염되거나 불법적으로 수정되었다고 판단하고, 이를 다른 센서 노드들에 공지할 수도 있다. 여기서, 상기 미리 결정된 기준 시간은 ATmega128 마이크로 컨트롤러를 사용하는 센서 노드에서 SHA-1을 해쉬 연산기로 사용할 경우 15초인 것이 바람직하다.

[0042] 도 6은 본 발명에 따른 방법의 연산단계 전후 상황을 간략하게 보여준다. 만일 목표 노드가 오염되지 않았다면 노드는 전체 메모리 이미지를 해쉬한 유효한 값을 전송할 것이며, 목표 노드가 오염되었고 공격자가 유효한 값을 응답하기 위한 공격코드를 심어놓았더라도 난수가 채워짐으로 인하여 공격코드를 덮어쓰게 된다. 결과적으로 오염된 노드는 검증자인 베이스 스테이션에게 유효한 값을 전송할 방법이 사라지게 된다. 도 7은 지금까지 설명한 방법의 4단계에 대한 전체적인 개념도이고, 도 8은 본 발명에 따른 선행 코드-검증 방법의 순서도이다.

[0043] [본 발명의 기술적 장점]

[0044] 1. 시간에 민감하지 않은 검증

[0045] WSN에서 SWATT는 악의적 코드를 검출해 내는 데 있어 연산 시간에 대한 면밀한 측정을 필요로 한다. 만약 노드가 더 강력한 마이크로 프로세서를 사용한다면 연산 시간의 차이를 증폭시키기 위하여 검증의 더 많은 반복 실행을 필요로 한다.

[0046] 또한, 이것은 노드가 많은 에너지를 소비하게 될 것이라는 것을 의미한다. 또 SWATT는 예상할 수 없는 시간 지연을 고려하기 힘들다. 만약 네트워크 지연시간 등을 고려하게 된다면, 공격자는 검증자가 공격에 필요한 연산 시간의 증가를 네트워크 지연시간으로 인식하게 만들 수 있다. 따라서 WSN에서의 SWATT는 매우 신중히 사용되어야한다.

[0047] 반면에, 본 발명에 따른 코드-검증 방법은 이러한 연산 시간의 정확한 측정을 필요로 하지 않는다. 대신에 정확한 응답을 위한 메모리 공간을 필요로 하는데, 만약 센서 노드가 많은 메모리 공간을 가지고 있다면, 빈 메모리 공간을 채우기 위한 시간을 필요로 하게 된다. 본 발명에서는 메모리의 크기에 비례하는 검증연산 시간과 네트워크 지연 시간을 합한 적절한 응답한계 시간을 둠으로써 이후에 설명할 여러 가지 공격을 예방할 수 있다. 또한 본 발명에 따른 방법에서 네트워크의 지연 시간이 목표 노드의 감염 여부를 결정하는데 끼치는 영향은 미비하기 때문에 시간에 기반하지 않는 유연성을 제공한다.

[0048] 2. 재전송 공격 방어

[0049] 공격자에 의하여 오염된 노드는 재전송 공격을 할 수 있다. 이 노드는 이전에 다른 노드의 검증절차에 쓰였던 유효한 메시지를 도청할 수 있다. 그러나, 본 발명에 따른 방법은 기본적으로 challenge-and-response 방식이므로 설령 공격자가 다른 노드와 검증자 사이의 세션 키를 획득하게 되더라도 난스(nonce)의 특성상 재전송 공격은 불가능하다.

[0050] challenge-and-response 방식에서 만약 공격자가 두 엔티티 간의 비밀을 알고 있다면 그는 이러한 비밀을 계산한 응답 메시지를 전송함으로써 쉽게 공격을 성공시킬 수 있다. 본 방법에서 비밀은 코드 그 자체이다. 사실상 코드는 공격자에게 쉽게 노출될 수 있기 때문에 본 프로토콜에 대한 공격은 쉬워 보인다. 그러나 본 발명의 방법에서 공격자는 악의적 코드와 정상 코드 모두를 유지할 메모리 공간을 가질 수 없다. 그러므로, 오염된 노드에서 난스를 사용한 유효한 응답을 도출해내는 것은 불가능하다.

[0051] 3. 필요한 난수 블록을 직접 계산하는 공격과 그 방어

[0052] 다른 공격 방법은 증명, 응답단계에서 악의적 코드가 해쉬값을 계산하는데, 필요한 시점에서 랜덤 시퀀스의 블록을 만들어내는 것이다. n을 빈 메모리 블록의 개수라고 본다면, n번째 블록은 $R_n \oplus R_{\lceil |S_e^i|/S_{ho} \rceil \times 2-n}$ 이기 때문에 시드로부터 $\lceil |S_e^i|/S_{ho} \rceil \times 2-n$ 번의 계산 값으로 볼 수 있다. 여기서 S_{ho}는 해쉬 함수의 출력 값의 크기이고 ‖는 특정 메모리의 크기를 나타낸다. (해쉬 함수가 SHA-1이라면 S_{ho}는 160이 되고, MD5라면 128이 된다.) 빈 메모리 영역은 Fill_Memory(n)가 동작하는 동안 난수 값을 두 번 채우게 됨을 상기하라. 또한, 유사하게 (n-1)번째 블록도 생성할 수 있다.

[0053] 만약, 총 메모리의 길이가 공격코드에 비교하여 월등히 크다고 가정한다면, 공격 코드는 자신이 상주한 메모리

위치에 정상 코드가 검증받을 때와 비교하여 최소한 20배(SHA-1)에서 16배(MD5)의 해쉬 연산을 수행하게 된다. 그러나, 총 메모리 길이 대 공격코드의 길이의 비율은 상기 가정보다 크기 때문에, 보다 많은 시간이 소요됨을 예측할 수 있다.

[0054] 4. 증명메커니즘의 성능분석

[0055] 본 발명에 따른 코드-검증 방법에서 총 실행시간 T는 다음과 같이 계산된다.

[0056] 식 (2): $$T = T_h + \sum_{i=1}^{A} (T_r + T_a^i) \times S_E^i$$

[0057] 여기서, Tr은 난수 발생에 소요되는 시간이고 T^i_a는 Mem^i의 접근시간이다. 현재 대부분 종류의 센서 노드는 Mem^1, Mem^2, Mem^3 만을 사용하고 있는데 Mem1은 SRAM, Mem^2는 프로그램 가능한 플래쉬 메모리, Mem^3는 외부 EEPROM이다. 아래 [표 1]은 센서 노드의 하드웨어 플랫폼이다.

[0058] [표 1] 센서 노드의 하드웨어 플랫폼

[0059]

Sensor	MICA MICA2Dot MICA2	Telos
마이크로프로세서	ATmega128	MSP430F1611
워드 크기	8-bit	16-bit
클락 주기	16MHz	8MHz
SRAM 크기	4kB	10kB
플래쉬 크기	128kB	48kB
MIPS	16	8

[0060] 만약, 우리가 앞서 가정한 바와 같이 관리자가 이미 플래쉬 메모리의 빈 공간을 남겨두지 않았다면 EEPROM의 쓰기시간은 매우 느리기 때문에 본 발명의 방법을 이용할 때 오직 SRAM 만을 랜덤 값으로 채우면 된다. 몇몇 종류의 센서 노드에서는 외부 EEPROM의 크기가 몇 메가비트이다. 그리고, 이러한 EEPROM을 채우려면 수천 초가 소요된다. 예를 들어, EEPROM의 사이즈가 4Mbit이고 쓰기 시간이 1비트당 1ms가 걸릴 경우, 전체 쓰기 시간은 1시간 8분 16초가 소요된다. 그에 반해 SRAM의 쓰기 시간은 무시해도 좋을 만큼 매우 빠르다.

[0061] 위에서 언급한 바와 같이, 본 방법의 실행시간은 오로지 해쉬 연산의 횟수로 계산할 수 있으므로 식(2)를 다음과 같이 다시 쓸 수 있다.

[0062] 식 (3): $T = T_h + T_r \times S^1_E$

[0063] U_f를 빈 메모리 공간을 채울 때의 해쉬 라운드 수로 하고, δ를 한 라운드의 실행 시간으로 하면, $T_r \times S^1_E$은 $U_f \times \delta$로 다시 쓸 수 있다. 또한, U_f는 다음의 식으로 계산할 수 있다.

[0064] 식 (4): $U_f = [|SRAM|/S_{ho}]$

[0065] U_r을 응답 단계에서의 해쉬 라운드의 수로 보면,

[0066] 식 (5): $U_r = [(|SRAM|+|Flash|)/S_{hi}] + 1$

[0067] S_{hi}는 해쉬 함수의 한 라운드에서의 입력 값 크기이다. (보통 S_{hi}는 512이다. +1은 해쉬 함수에서 초기 패딩 연산이 있기 때문에 추가한다.) 따라서 T_h도 $U_f \times \delta$로 표현할 수 있다. 최종적으로 T는 다음과 같다. 아래 [표 2]는 라운드 시간과 필요한 해쉬 라운드 수(*는 추산 값이다.)

[0068] 식 (6): $T = U_r \times \delta + U_f \times \delta = (U_r + U_f) \times \delta$

[0069] [표 2] 라운드 시간과 필요한 해쉬 라운드 수(*는 추산 값이다.)

[0070]

마이크로프로세서	라운드 시간(δ)	라운드 수(U_f)	라운드 수(U_r)
ATmega128			
(SHA-1)	3636㎲	205	2112(+1)
(MD5)	1473㎲	256	2112(+1)
MSP430F1611			
(SHA-1)	*7272㎲	512	928(+1)
(MD5)	*2946㎲	640	928(+1)

[0071] ATmega128에서 라운드의 실행시간 δ는 SHA-1에서 3636㎲, MD5에서 1473㎲임을 Alexander Dean의 논문(P. Ganesan, R Venugopalan, P. Peddabachagari, A. Dean, F. Mueller, M. Sichitiu "Analyzing and Modeling Encryption Overhead for Sensor Network Nodes." In WSNA'03, September 19, 2003, San Diego, California, USA. ACM Press) 내 실험결과에서 언급하였다. 각각의 MIPS를 비교해봄으로써 ATmega128의 값으로 ATmega163과 MSP430F1611의 δ를 추산하였다.

[0072] 표 3은 각각의 마이크로프로세서에서의 실행시간을 보여주고 있다. 이 표에서 나타나다시피 실행시간은 마이크로프로세서에 따라 약 1초부터 10초에 이르기까지 큰 차이를 보이고 있다. 따라서 노드로부터의 응답 대기시간을 설정할 때, 네트워크 지연시간과 더불어 이러한 추정 실행시간을 충분히 반영하여야 할 것이다.

[0073] [표 3] 해쉬 연산에 필요한 시간

[0074]

마이크로프로세서	시간($U_f \times \delta$)	시간($U_r \times \delta$)	총 시간(T)
ATmega128			
(SHA-1)	0.745s	7.863s	8.428s
(MD5)	0.377s	3.112s	3.489s
MSP430F1611			
(SHA-1)	3.723s	6.756s	10.479s
(MD5)	1.855s	2.737s	4.622s

[0075] 5. 공격 모델의 성능분석

[0076] 상기 3.에서 언급된 공격 모델에 대한 분석을 한다. 공격자는 응답단계에서 해쉬 함수가 필요로 할 때, 난수 값으로 구성된 블록 값을 생성해야 함을 상기하라. 공격자가 자신의 코드를 숨기기 위해 수행해야 할 총 연산시간은 다음과 같다.

[0077] 식 (7): $Ta = T' + U_a \times \delta$

[0078] U_a는 공격자가 유효한 응답을 생성하기 위해 필요한 총 해쉬 라운드의 수이다. 공격 코드의 총 길이를 20-블록으로 가정하면, 아래의 [표 4]는 공격 모델의 총 실행시간의 근사치이다.

[0079] [표 4] 공격 코드가 해쉬 연산에 필요한 시간

[0080]

마이크로프로세서	라운드(U_a)	시간($U_a \times \delta$)	공격 코드의 총 시간(T')
ATmega128			
(SHA-1)	3600	13.089s	22.262s
(MD5)	2800	4.242s	8.128s
MSP430F1611			
(SHA-1)	3600	26.179s	40.381s
(MD5)	2800	8.484s	14.991s

[0081] 도 9는 정상적 검증기법의 총 연산시간과 공격 코드의 총 연산시간의 격차를 보여준다. 도 9에서 공격 코드의 시간은 정상의 그것에 최소한 두 배를 상회한다는 것을 알 수 있다. 이것은 검증자가 네트워크 트래픽등의 예상치 못한 시간 지연에도 불구하고 정상노드와 악의적 노드를 구분하기에 충분한 시간차를 얻을 수 있음을 의미한

다.

[0082] 본 발명의 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법에 의하면, 매우 간단한 방식으로 동작하나 정확한 시간 정보에 의존하지 않고서도 센서 노드 내의 악의적 코드를 발견해내는데 있어 효과적이다. 이 방법의 특성상 증명자와 목표 노드간의 시도-응답 방식을 노드와 노드가 서로 증명자와 목표 노드의 역할을 수행하는 방식으로 쉽게 변환할 수 있다. 본 발명은 이러한 노드-대-노드 증명방법을 제시하고 있으며, 특히 노드-대-노드 증명방법은 베이스 스테이션의 중계 없이 각각의 노드가 연계하여 포획된 노드를 네트워크 내에서 제외시키는 기능을 수행하도록 확장할 수 있다. 본 방법을 노드-대-노드 방법까지 확장할 경우 베이스 스테이션이 없는 환경에서도 공격자에 의해 악의적으로 수정되었다고 의심되는 노드를 선행 검증하고 네트워크에 피해를 주는 행동을 하기 전에 적절히 제거할 수 있을 것이다. 또한 본 방법의 적절한 수정을 통하여 특정 그룹 내 노드들의 코드를 증명하는 그룹 코드-증명 방법을 고안할 수 있을 것이다.

[0083] 이상과 같이 본 발명에 따른 무선 센서 네트워크에서의 메모리 공간 삭제를 이용한 선행 코드-검증 방법을 예시한 도면을 참조로 하여 설명하였으나, 본 명세서에 개시된 실시예와 도면에 의해 본 발명이 한정되는 것은 아니며, 본 발명의 기술사상 범위내에서 당업자에 의해 다양한 변형이 이루어질 수 있음은 물론이다.

도면의 간단한 설명

[0084] 도 1은 본 발명에서 사용되는 표기에 대한 설명을 도시한 도,

[0085] 도 2는 센서 노드의 메모리를 개념적으로 도시한 도로서, 여기서 Mem^1은 마이크로 콘트롤러의 RAM, Mem^i는 프로그램 가능한 플래시 메모리를 도시한 도이다.

[0086] 도 3a 내지 도 3c는 본 발명의 연산단계에서 목표 노드의 빈 메모리 공간을 채우는 과정을 도식적으로 도시한 도,

[0087] 도 4는 상기 도 3a 내지 도 3c의 과정을 실행하기 위한 Fill_Memory(n) 알고리즘을 도시한 도,

[0088] 도 5는 SHA-1을 사용하여 MIXED_MEM으로 표현한 혼합화의 일 예를 도시한 도,

[0089] 도 6은 Fill_Memory(n) 알고리즘 전후의 메모리 내용 변화를 개념적으로 도시한 도,

[0090] 도 7은 본 발명에 따른 선행 코드-검증 방법을 도식적으로 도시한 도,

[0091] 도 8은 본 발명에 따른 선행 코드-검증 방법을 도시한 순서도,

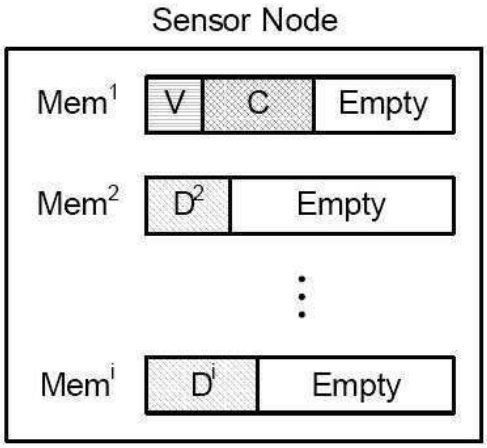
[0092] 도 9는 정상적 검증 방법의 총 연산시간과 공격코드의 총 연산시간의 차를 도시한 그래프이다.

도면

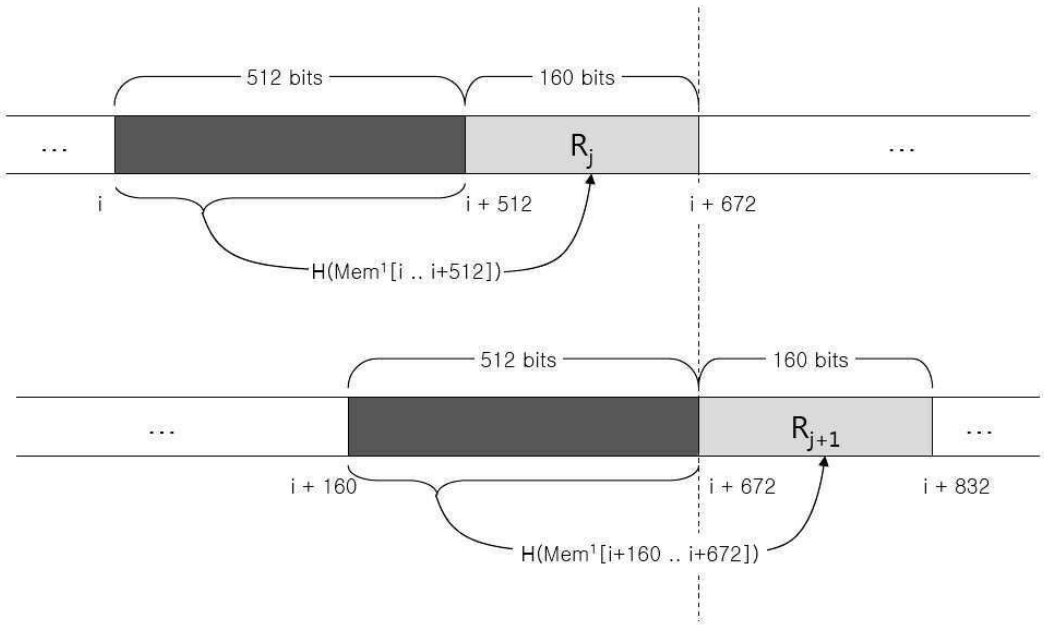
도면1

영어 약자	설 명
$V \parallel D$	코드 V, D 간의 연결
$D_i \parallel_{i=1}^{m}$	D_1부터 D_m까지의 연결
$K_{A,B}$	노드 A, B 간에 분배된 키.
$E_{K_{A,B}}(m)$	메시지 m을 키 $K_{A,B}$로 암호화
$D_{K_{A,B}}(c)$	암호문 c를 키 $K_{A,B}$로 복호화

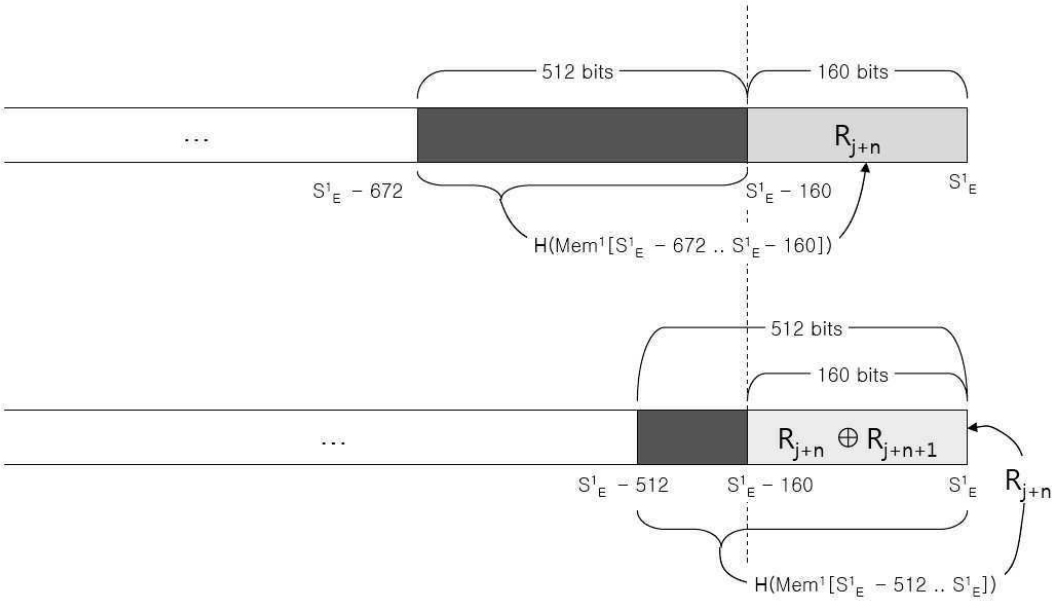
도면2


Sensor Node
Mem1
V
C
Empty
Mem2
D2
Empty
Memi
Di
Empty

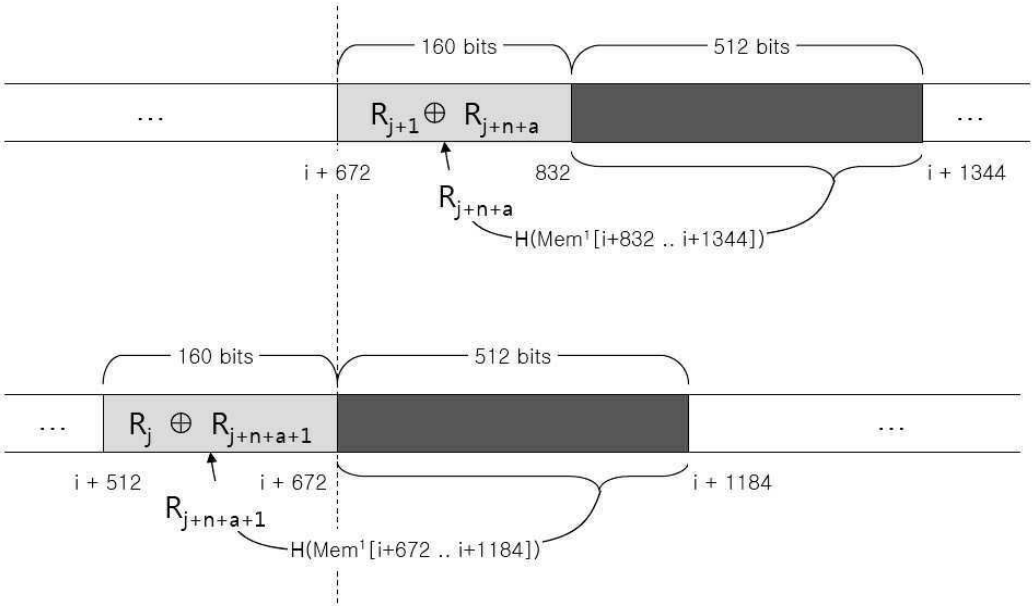
도면3a


512 bits
160 bits
Rj
i
i + 512
i + 672
H(Mem1[i .. i+512])
512 bits
160 bits
Rj+1
i + 160
i + 672
i + 832
H(Mem1[i+160 .. i+672])

도면3b


512 bits
160 bits
...
R_{j+n}
$S^1_E - 672$
$S^1_E - 160$
S^1_E
$H(Mem^1[S^1_E - 672 .. S^1_E - 160])$
512 bits
160 bits
...
$R_{j+n} \oplus R_{j+n+1}$
$S^1_E - 512$
$S^1_E - 160$
S^1_E
R_{j+n}
$H(Mem^1[S^1_E - 512 .. S^1_E])$

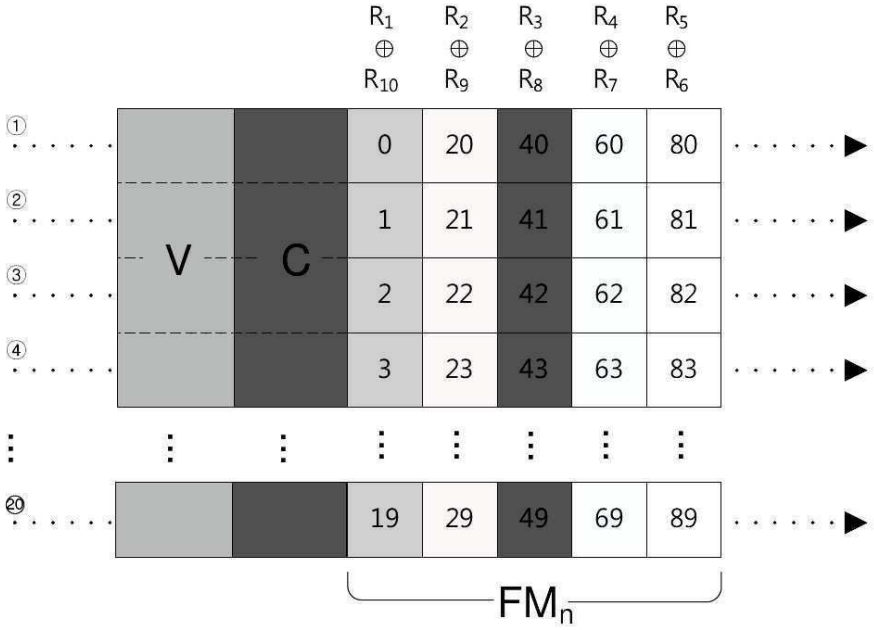
도면3c


160 bits
512 bits
...
$R_{j+1} \oplus R_{j+n+a}$
...
i + 672
832
i + 1344
R_{j+n+a}
$H(Mem^1[i+832 .. i+1344])$
160 bits
512 bits
...
$R_j \oplus R_{j+n+a+1}$
...
i + 512
i + 672
i + 1184
$R_{j+n+a+1}$
$H(Mem^1[i+672 .. i+1184])$

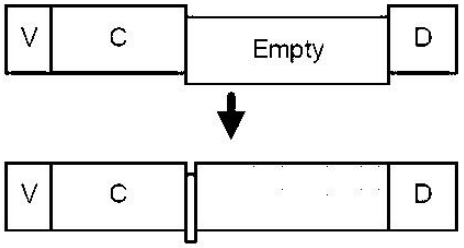
도면4

```
Algorithm Fill_Memory(n)
1   for i=1 to Λ
    /* Filling the empty memory with random number */
2       for j=1 to 4
3           n←h^512(n||padding 0)
4           Mem^i[j-1]←n
5       for j=5 to S_e^i
6           Mem^i[j-1]←h^512(previous 512-bit)
        /* XOR operation in the opposite direction */
7       for j=1 to S_e^i
8           Mem^i[S_e^i-j]←Mem^i[S_e^i-j]⊕h^512(previous 512-bit)
9   return
```

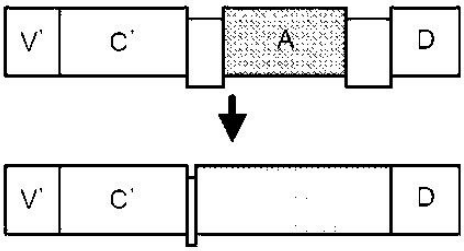
도면5



도면6

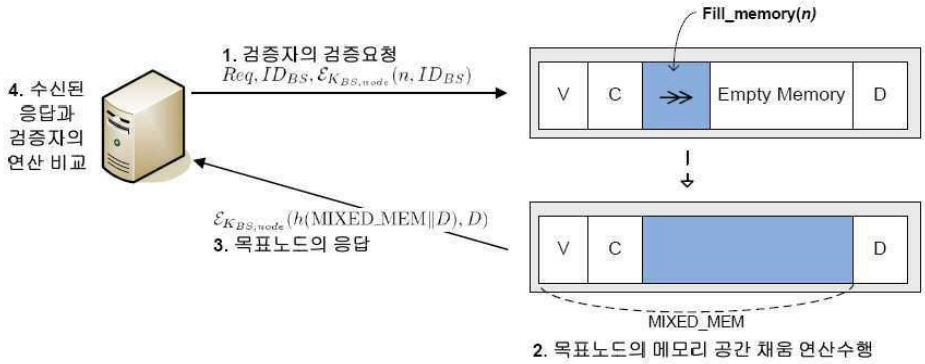


(a) 목표 노드가 정상일 때

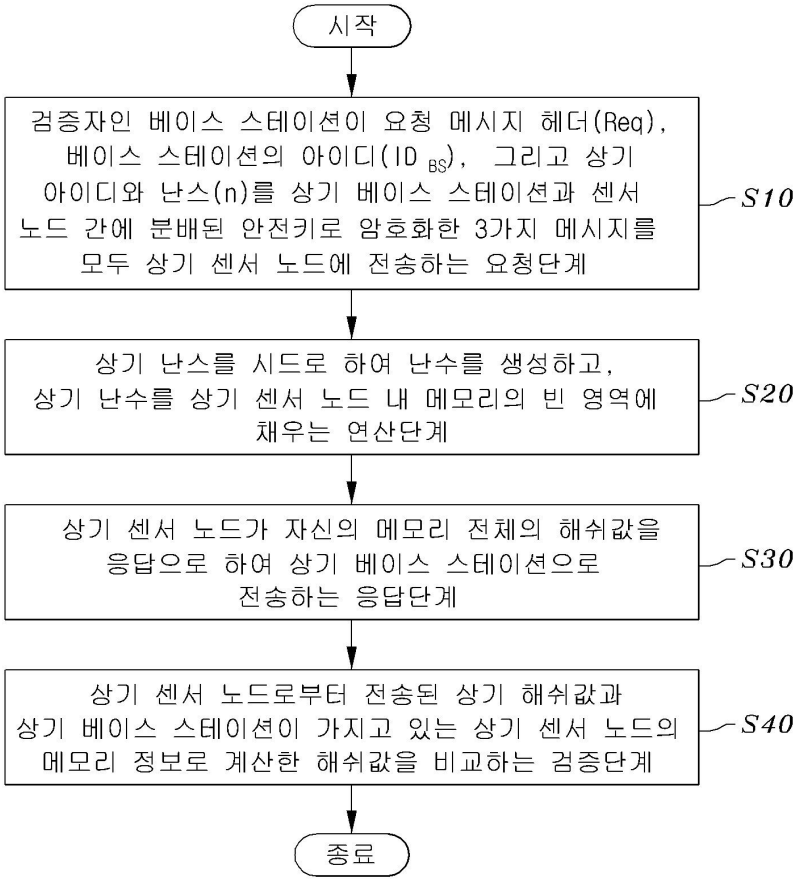


(b) 목표 노드가 공격당해 변환되었을 때

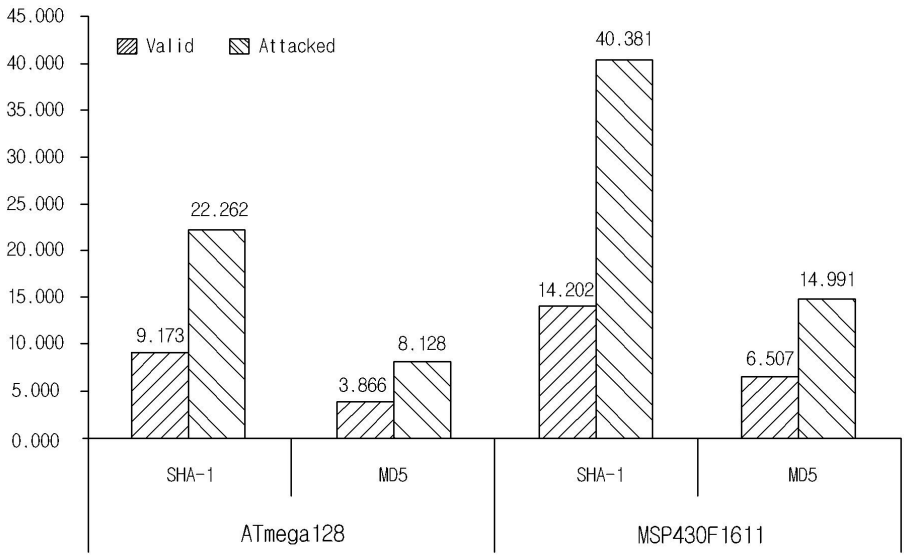
도면7



도면8


시작
검증자인 베이스 스테이션이 요청 메시지 헤더(Req), 베이스 스테이션의 아이디(ID BS), 그리고 상기 아이디와 난스(n)를 상기 베이스 스테이션과 센서 노드 간에 분배된 안전키로 암호화한 3가지 메시지를 모두 상기 센서 노드에 전송하는 요청단계
S10
상기 난스를 시드로 하여 난수를 생성하고, 상기 난수를 상기 센서 노드 내 메모리의 빈 영역에 채우는 연산단계
S20
상기 센서 노드가 자신의 메모리 전체의 해쉬값을 응답으로 하여 상기 베이스 스테이션으로 전송하는 응답단계
S30
상기 센서 노드로부터 전송된 상기 해쉬값과 상기 베이스 스테이션이 가지고 있는 상기 센서 노드의 메모리 정보로 계산한 해쉬값을 비교하는 검증단계
S40
종료

도면9


45.000
40.000
35.000
30.000
25.000
20.000
15.000
10.000
5.000
0.000
Valid
Attacked
9.173
22.262
3.866
8.128
14.202
40.381
6.507
14.991
SHA-1
MD5
SHA-1
MD5
ATmega128
MSP430F1611