(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2014-0145967
(43) 공개일자 2014년12월24일

(51) 국제특허분류(Int. Cl.)
G06F 21/71 (2013.01) *G06F 13/38* (2006.01)
(21) 출원번호 10-2014-0015181
(22) 출원일자 2014년02월11일
심사청구일자 없음
(30) 우선권주장
1020130068071 2013년06월14일 대한민국(KR)
1020130070677 2013년06월20일 대한민국(KR)

(71) 출원인
장동훈
서울특별시 강서구 곰달래로35길 35 ,42-36(화곡동1층)()
(72) 발명자
장동훈
서울특별시 강서구 곰달래로35길 35 ,42-36(화곡동1층)()

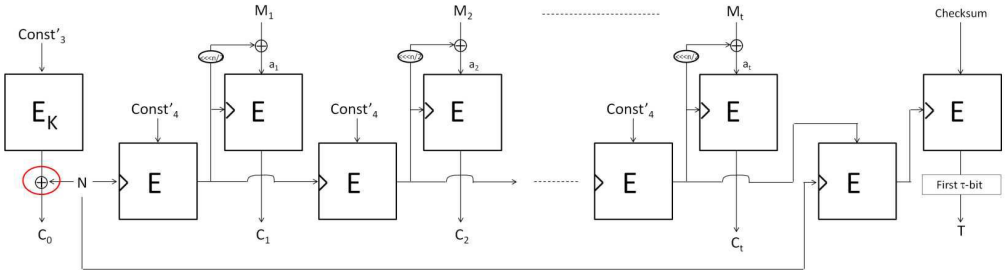
전체 청구항 수 : 총 1 항

(54) 발명의 명칭 **암호화 시스템 및 그것의 암호 통신 방법**

(57) 요 약

본 발명의 실시 예에 따른 암호화 시스템의 안전한 암호 연산 방법은, 제 1 기반 암호 함수에서 암호 연산을 수행하는 단계; 상기 제 1 기반 암호 함수에서 연산한 값과 제 1 부가적인 정보를 이용하여 제 1 부채널 방지 외부 연산을 수행하는 단계; 상기 1 부채널 방지 외부 연산 과정으로부터 산출된 값과 제 2 부가적인 정보를 이용하여 제 2 부채널 방지 연산을 수행하는 단계; 및 제 2 암호함수에서 상기 제 2 부채널 방지 연산된 값을 이용하여 상기 기반 암호 함수에서 암호 연산을 수행하는 단계를 포함한다. 본 발명에 의하면, 안전한 암호 통신을 위하여 부채널 대응 기법의 비용을 획기적으로 낮출 수 있다.

대 표 도 - 도19



특허청구의 범위

청구항 1

암호화 시스템의 암호 연산 방법에 있어서:

제 1 기반 암호 함수에서 암호 연산을 수행하는 단계;

상기 제 1 기반 암호 함수에서 연산된 값과 제 1 부가적인 정보를 이용하여 제 1 부채널 방지 외부 연산을 수행하는 단계;

상기 1 부채널 방지 외부 연산 과정으로부터 산출된 값과 제 2 부가적인 정보를 이용하여 제 2 부채널 방지 연산을 수행하는 단계; 및

제 2 암호함수에서 상기 제 2 부채널 방지 연산된 값을 이용하여 상기 기반 암호 함수에서 암호 연산을 수행하는 단계를 포함하는 암호 연산 방법.

명 세 서

기 술 분 야

[0001] 본 발명은 암호화 시스템에 관한 것으로, 좀 더 자세하게는 안전한 통신 환경을 위한 암호화 시스템 및 그것의 암호 통신 방법에 대한 것이다.

배 경 기 술

[0002] 최근 스마트폰 등 스마트 디바이스에 대한 부채널 취약점 공격에 대한 확대가 예상되며, 부채널 공격 기술은 점차 준전문가 저비용 공격 기술로의 변화로 말미암아, 고등 부채널 대응 기법에 대한 요구가 증가하고 있다. 하지만, 고등 부채널 대응을 위해 기존의 대응 기법을 적용할 경우 심하게는 300배에서 500배의 성능 저하가 있어서 실제로 사용이 불가능한 상황이며, 따라서 성능 저하를 고도화할 수 있는 신규 부채널 방지를 위한 원천 기술의 확보가 필요하다.

[0003] 배터리가 필요 없는 교통 카드처럼 인증 및 지불 용도로만 사용할 경우, 주고 받는 데이터 양이 적기 때문에, 성능 저하의 체감도가 낮을 수 있으나, 통신의 효율성이 중시되는 통신 환경에서의 안전한 암호 통신의 구현은 현존하는 방식으로 구현이 불가능하다. 무엇보다, 배터리에 기반하여 에너지 소비에 매우 민감한 구현 환경의 경우, 현존하는 부채널 대응 기법 적용에 따른 상당한 에너지 손실에 의해 실용성 및 가용성 측면에서 큰 타격을 입게 된다.

[0004] 위처럼 300배에서 500 배의 성능 저하를 가져오는 근본 이유는 암호, 복호화 및 인증 등을 위한 설계 기법의 암호학적 복잡성에 기인한다. 예를 들면, 블록암호 기반 암호화 (또는 블록 암호 기반 암호화 운영 모드)의 경우, 부채널 대응 기법 비용 중 블록 암호 자체에 대한 대응 기법 비용이 대부분을 차지한다. 그 이유는 블록 암호에 사용된 비밀 키를 보호하기 위하여 블록 암호 연산 전체 과정에 대한 부채널 대응 기법 적용이 요구되기 때문이다. 동시에 공격 기법이 준 전문가용, 저비용, 고도화됨에 따라 단위 연산 당 적용해야 할 부채널 대응 기법 비용이 지속적으로 증가하기 때문이다. 이로 인하여, M2M, 의료기기, 스마트기기, RFID 등과 같이 구현 환경에 제약이 있는 경우에 대해 저비용, 효율성, 안전성을 제공할 수 있는 설계 논리 및 구현 기법 마련에 대한 연구가 시급한 상황이다.

[0005] 현존하는 암호화 및 인증 코드 값 생성 방식들은 기반이 되는 알고리즘이 부채널 공격에 안전하게 구현되어야 한다는 근본적인 문제점을 지니고 있다. 이러한 근본적인 문제점을 안고 가는 이상, 부채널 공격에 대해 저비용, 고효율, 고 안전성을 갖는 구현 방법은 사실상 없는 상황이다.

[0006] 또한, 기존의 대부분의 암호인증 기법은 암호문의 변조 유무를 파악하기 위한 메모리 사용 요구량에 대한 기준 없이 개발되어 사용되어 왔다. 해킹 등 다양한 공격 기법의 발달로 인하여 암호 모듈에 대한 필요성이 대두됨에 따라 암호 모듈에 기반한 암호인증 기법이 중요하게 되었다. 하지만, 현존하는 대부분의 암호인증 기법은 암호문 변조 검증 시 메모리 사용에 대한 언급이 없으며, 무엇보다 이들 대부분의 암호인증 기법은 대상이 되는 암

호문의 크기가 길어짐에 따라 요구되는 메모리 크기 양도 증가한다. 따라서 효율적으로 동시에 적은 메모리 사용량으로 암호문 변조를 검증하고 안전하게 평문을 출력하는 기법에 대한 개발이 요구된다.

발명의 내용

해결하려는 과제

[0007] 본 발명에서는 운영 모드 차원에서 접근하고자 한다. 어떻게 하면, 기반 알고리즘에 대한 부채널 대응 기법 적용 없이 운영 모드 차원에서의 대응만으로 안전성을 제공할 수 있는가에 대한 것이다. 본 발명에서는, 운영 모드 차원에서, 부채널 공격에 대해 저비용, 고 효율성, 고 안전성을 제공하는 암호화 및 MAC 값 생성하는 설계 논리 및 구현 방식을 제안한다.

[0008] 본 발명에서는 신규 암호, 복호화, 인증 등의 설계 기법을 제안한다. 이는 현존하는 알려진 모든 기술들과 달리, 기반 함수가 되는 블록암호, 치환 함수, 압축 함수 등에 대한 부채널 공격에 대한 대응 기법 적용이 없이도, 단지 기반 함수 외부에 적용된 매우 단순한 연산들을 보호만 해도 전체 암호, 복호화, 인증 등의 설계 기법이 부채널 공격에 강한 안전성을 제공한다.

과제의 해결 수단

[0009] 본 발명의 실시 예에 따른 암호화 시스템의 안전한 암호 연산 방법은, 제 1 기반 암호 함수에서 암호 연산을 수행하는 단계; 상기 제 1 기반 암호 함수에서 연산된 값과 제 1 부가적인 정보를 이용하여 제 1 부채널 방지 외부 연산을 수행하는 단계; 상기 1 부채널 방지 외부 연산 과정으로부터 산출된 값과 제 2 부가적인 정보를 이용하여 제 2 부채널 방지 연산을 수행하는 단계; 및 제 2 암호함수에서 상기 제 2 부채널 방지 연산된 값을 이용하여 상기 기반 암호 함수에서 암호 연산을 수행하는 단계를 포함한다.

발명의 효과

[0010] 상술한 바와 같이 본 발명을 통해에 따른 간단한 논리 연산을 이용하여 기반 암호 함수 대신 외부 연산 과정을 보호하는 데에 초점을 맞추므로 전체 암호 시스템을 부채널 공격으로부터 저비용, 고 안전성, 고 효율성을 갖는 방식으로 보호할 수 있다. 또한, 본 발명에 따른 논리 연산을 이용하여 저 메모리로 암호인증 기법을 암호 모듈에서 안전하게 구현하여 비밀 정보를 안전하게 기반 함수를 보호할 수 있다. 부채널 공격으로부터 기반 함수를 보호할 수 있다.

도면의 간단한 설명

[0011] 도 1은 본 발명에서 고려하는 구현 환경 모델을 보여주는 도면이다.

도 2는 일반적인 부채널 공격 방지 기법이 적용된 암호 시스템의 구조를 보여주는 도면이다.

도 3은 본 발명의 실시 예에 따른 신규 암호 시스템의 안전성을 위해 기반이 되는 암호 함수에 대한 부채널 방지 기법 적용 없이도 단지 외부 연산 과정에 대한 부채널 방지 기법 적용만으로 신규 암호 시스템이 안전함을 기반 함수에 대한 부채널 공격 방지를 개념적으로 설명하는 도면이다.

도 4는 블록 암호 기반 암호인증 방식에서 공유한 키 K를 이용하여 난스 열을 생성하는 방식을 보여준다.

도 5는 블록 암호 기반 암호인증 방식에서 난스 열 중의 하나의 난스를 N으로 표현했을 때, 난스로부터 마스크 열을 생성하는 방식을 보여준다.

도 6는 블록 암호 기반 암호인증 방식에서, 도 4과 도 5에서 제시된 난스 열과 마스크 열을 이용하여 공유한 키 K를 이용한 첫 번째 암호화 및 인증 방식을 보여준다.

도 7은 블록 암호 기반 암호인증 방식에서, 도 6에서 제시한 첫 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 8은 블록 암호 기반 암호인증 방식에서, 도 4과 도 5에서 제시된 난스 열과 마스크 열을 이용하여 공유한 키 K를 이용한 두 번째 암호화 및 인증 방식을 보여준다.

도 9는 블록 암호 기반 암호인증 방식에서, 도 8에서 제시한 두 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 10은 블록 암호 기반 암호인증 방식에서, 도 4에서 제시된 난스 열을 이용하여 공유한 키 K를 이용한 세 번째 암호화 및 인증 방식을 보여준다.

도 11은 블록 암호 기반 암호인증 방식에서, 도 10에서 제시한 세 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 12는 블록 암호 기반 암호인증 방식에서, 도 4에서 제시된 난스 열을 이용하여 공유한 키 K를 이용한 네 번째 암호화 및 인증 방식을 보여준다.

도 13은 블록 암호 기반 암호인증 방식에서, 도 12에서 제시한 네 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 14는 블록 암호 기반 암호인증 방식에서, 도 4에서 제시된 난스 열을 이용하여 공유한 키 K를 이용한 다섯 번째 암호화 및 인증 방식을 보여준다.

도 15는 블록 암호 기반 암호인증 방식에서, 도 14에서 제시한 다섯 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 16은 블록 암호 기반 암호인증 방식에서, 도 4에서 제시된 난스 열을 이용하여 공유한 키 K를 이용한 여섯 번째 암호화 및 인증 방식을 보여준다.

도 17은 블록 암호 기반 암호인증 방식에서, 도 16에서 제시한 여섯 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 18은 블록 암호 기반 암호인증 방식에서, 도 4에서 제시된 난스 열을 이용하여 공유한 키 K를 이용한 일곱 번째 암호화 및 인증 방식을 보여준다.

도 19는 블록 암호 기반 암호인증 방식에서, 도 18에서 제시한 여섯 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 20은 블록 암호 기반 암호인증 방식에서, 공유 키 K로부터 암호화 키 열을 생성하는 방식을 보여주는 도면으로, 이는 본 발명에서 제시된 두 가지 암호화 및 인증 방식의 안전성을 강화한다.

도 21은 블록 암호 기반 암호인증 방식에서 공유 키 K로부터 난스 열을 생성하는 방식을 보여주는 도면으로, 이는 본 발명에서 제시된 두 가지 암호화 및 인증 방식의 안전성을 강화한다.

도 22는 블록 암호 기반 암호인증 방식에서, 도 6에서 보여준 첫 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 23은 블록 암호 기반 암호인증 방식에서, 도 8에서 보여준 두 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 24는 블록 암호 기반 암호인증 방식에서, 도 10에서 보여준 세 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 25는 블록 암호 기반 암호인증 방식에서, 도 12에서 보여준 네 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 26은 블록 암호 기반 암호인증 방식에서, 도 14에서 보여준 다섯 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 27은 블록 암호 기반 암호인증 방식에서, 도 16에서 보여준 여섯 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 28은 블록 암호 기반 암호인증 방식에서, 도 18에서 보여준 일곱 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 29는 블록 암호 기반 암호인증 방식에서, 본 발명에서 앞서 보인 난스 암호화 부분을 나타낸다.

도 30은 블록 암호 기반 암호인증 방식에서, 난스 암호화를 하기 위하여, 일방향 함수 f를 이용한 방식을 보여준다.

도 31은 블록 암호 기반 암호인증 방식에서, 도 30에서 일방향 함수 f 설계 시 블록암호 E를 이용한 방식을 보

여준다.

도 32는 블록 암호 기반 암호인증 방식에서, 도 26에서 보인 다섯 번째 방식을 메모리가 적은 환경에서 구현하는 방식을 보여주는 도면이다.

도 33은 블록 암호 기반 암호인증 방식에서, 도 27에서 보인 여섯 번째 방식을 메모리가 적은 환경에서 구현하는 방식을 보여주는 도면이다.

도 34는 블록 암호 기반 암호인증 방식에서, 도 28에서 보인 일곱 번째 방식을 메모리가 적은 환경에서 구현하는 방식을 보여주는 도면이다.

도 35는 치환 함수 기반 암호인증 방식에서 공유한 키 K를 이용하여 난스 열을 생성하는 방식을 보여준다.

도 36은 치환 함수 기반 암호인증 방식에서, 도 35로부터 제시된 난스 열과 공유한 키 K를 이용한 첫 번째 암호화 및 인증 방식을 보여준다.

도 37은 치환 함수 기반 암호인증 방식에서, 도 36에서 제시한 첫 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 38은 치환 함수 기반 암호인증 방식에서, 도 35로부터 제시된 난스 열과 공유한 키 K를 이용한 두 번째 암호화 및 인증 방식을 보여준다.

도 39는 치환 함수 기반 암호인증 방식에서, 도 38에서 제시한 두 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 40은 치환 함수 기반 암호인증 방식에서, 도 35로부터 제시된 난스 열과 공유한 키 K를 이용한 세 번째 암호화 및 인증 방식을 보여준다.

도 41은 치환 함수 기반 암호인증 방식에서, 도 40에서 제시한 첫 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 42는 치환 함수 기반 암호인증 방식에서, 도 35로부터 제시된 난스 열과 공유한 키 K를 이용한 네 번째 암호화 및 인증 방식을 보여준다.

도 43은 치환 함수 기반 암호인증 방식에서, 도 42에서 제시한 네 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 44는 치환 함수 기반 암호인증 방식에서 공유 키 K로부터 암호화 키 열을 생성하는 방식을 보여주는 도면이다.

도 45는 치환 함수 기반 암호인증 방식에서 공유 키 K로부터 난스 값 사이에 연관성을 줄이기 위하여 난스 열을 생성하는 방식을 보여주는 도면이다.

도 46은 치환 함수 기반 암호인증 방식에서, 도 36에서 보여준 첫 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 47은 치환 함수 기반 암호인증 방식에서, 도 38에서 보여준 두 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 48은 치환 함수 기반 암호인증 방식에서, 도 40에서 보여준 세 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 49는 치환 함수 기반 암호인증 방식에서, 도 42에서 보여준 네 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 50은 치환 함수 기반 암호인증 방식에서 난스 열을 생성하는 방식을 보여주는 도면이다.

도 51은 치환 함수 기반 암호인증 방식에서 키들 사이의 연관성이 없도록 키 열을 생성하는 방식을 보여주는 도면이다.

도 52는 치환 함수 기반 암호인증 방식에서 방식 1에서 부가정보 및 인증 코드의 크기를 늘리는 방식을 보여주는 도면이다.

도 53은 치환 함수 기반 암호인증 방식에서 방식 2에서 부가정보 및 인증 코드의 크기를 늘리는 방식을 보여주

는 도면이다.

도 54는 치환 함수 기반 암호인증 방식에서 방식 3에서 부가정보 및 인증 코드의 크기를 늘리는 방식을 보여주는 도면이다.

도 55는 치환 함수 기반 암호인증 방식에서 방식 4에서 부가정보 및 인증 코드의 크기를 늘리는 방식을 보여주는 도면이다.

도 56은 치환 함수 기반 암호인증 방식에서 방식 1에 대해 키 없이 난스 만으로 암호화 및 인증 코드를 생성하는 방식을 보여주는 도면이다.

도 57은 치환 함수 기반 암호인증 방식에서 방식 2에 대해 키 없이 난스 만으로 암호화 및 인증 코드를 생성하는 방식을 보여주는 도면이다.

도 58은 치환 함수 기반 암호인증 방식에서 방식 3에 대해 키 없이 난스 만으로 암호화 및 인증 코드를 생성하는 방식을 보여주는 도면이다.

도 59는 치환 함수 기반 암호인증 방식에서 방식 4에 대해 키 없이 난스 만으로 암호화 및 인증 코드를 생성하는 방식을 보여주는 도면이다.

도 60은 치환 함수 기반 암호인증 방식에서 난스 N으로부터 N'을 생성하는 방법을 보여주는 도면이다.

도 61은 치환 함수 기반 암호인증 방식에서 방식 4를 이용하여 메모리가 제한된 경우에 대해 암호화 및 인증을 하는 방식을 보여주는 도면이다.

도 62는 압축 함수 기반 암호인증 방식에서 공유한 키 K를 이용하여 난스 열을 생성하는 방식을 보여준다.

도 63은 압축 함수 기반 암호인증 방식에서, 도 62로부터 제시된 난스 열과 공유한 키 K를 이용한 첫 번째 암호화 및 인증 방식을 보여준다.

도 64는 압축 함수 기반 암호인증 방식에서, 도 63에서 제시한 첫 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 65는 압축 함수 기반 암호인증 방식에서, 도 62로부터 제시된 난스 열과 공유한 키 K를 이용한 두 번째 암호화 및 인증 방식을 보여준다.

도 66은 압축 함수 기반 암호인증 방식에서, 도 65에서 제시한 두 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 67은 압축 함수 기반 암호인증 방식에서, 도 62로부터 제시된 난스 열과 공유한 키 K를 이용한 세 번째 암호화 및 인증 방식을 보여준다.

도 68은 압축 함수 기반 암호인증 방식에서, 도 67에서 제시한 첫 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 69는 압축 함수 기반 암호인증 방식에서, 도 62로부터 제시된 난스 열과 공유한 키 K를 이용한 네 번째 암호화 및 인증 방식을 보여준다.

도 70은 압축 함수 기반 암호인증 방식에서, 도 69에서 제시한 네 번째 방식에서 부채널 공격에 대한 안전성을 위해 보호해야 할 연산 부분을 표시한 도면이다.

도 71은 압축 함수 기반 암호인증 방식에서 공유 키 K로부터 암호화 키 열을 생성하는 방식을 보여주는 도면이다.

도 72는 압축 함수 기반 암호인증 방식에서, 도 63에서 보여준 첫 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 73은 압축 함수 기반 암호인증 방식에서, 도 65에서 보여준 두 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 74는 압축 함수 기반 암호인증 방식에서, 도 67에서 보여준 세 번째 암호화 및 인증 방식에 헤더와 같은 부가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 75는 압축 함수 기반 암호인증 방식에서, 도 69에서 보여준 네 번째 암호화 및 인증 방식에 헤더와 같은 부

가정보를 추가로 인증하고자 하는 경우를 표현한 방식이다.

도 76은 압축 함수 기반 암호인증 방식에서 키들 사이의 연관성이 없도록 키 열을 생성하는 방식을 보여주는 도면이다.

도 77은 압축 함수 기반 암호인증 방식에서 방식 1에서 부가정보 및 인증 코드의 크기를 늘리는 방식을 보여주는 도면이다.

도 78은 압축 함수 기반 암호인증 방식에서 방식 2에서 부가정보 및 인증 코드의 크기를 늘리는 방식을 보여주는 도면이다.

도 79는 압축 함수 기반 암호인증 방식에서 방식 3에서 부가정보 및 인증 코드의 크기를 늘리는 방식을 보여주는 도면이다.

도 80은 압축 함수 기반 암호인증 방식에서 방식 4에서 부가정보 및 인증 코드의 크기를 늘리는 방식을 보여주는 도면이다.

도 81은 압축 함수 기반 암호인증 방식에서 난스 기반 암호화 및 인증 방식 위한 난스 열 생성 방식을 보여주는 도면이다.

도 82는 압축 함수 기반 암호인증 방식에서 방식 1에 대해 키 없이 난스 만으로 암호화 및 인증 코드를 생성하는 방식을 보여주는 도면이다.

도 83은 압축 함수 기반 암호인증 방식에서 방식 2에 대해 키 없이 난스 만으로 암호화 및 인증 코드를 생성하는 방식을 보여주는 도면이다.

도 84는 압축 함수 기반 암호인증 방식에서 방식 3에 대해 키 없이 난스 만으로 암호화 및 인증 코드를 생성하는 방식을 보여주는 도면이다.

도 85는 압축 함수 기반 암호인증 방식에서 방식 4에 대해 키 없이 난스 만으로 암호화 및 인증 코드를 생성하는 방식을 보여주는 도면이다.

도 86은 압축 함수 기반 암호인증 방식에서 난스에 대한 동기화 과정 없이 암호화 및 인증 방식을 제공하기 위하여 난스 N으로부터 N'과 N''을 생성하는 방법을 보여주는 도면이다.

도 87은 압축 함수 기반 암호인증 방식에서 방식 4를 이용하여 메모리가 제한된 경우에 대해 암호화 및 인증을 하는 방식을 보여주는 도면이다.

도 88은 암호 모듈에 기반한 장치를 나타낸다.기반 함수에 대한 부채널 공격 방지를 암호 모듈 안에서의 동작 방식은 도 89에 설명한다.

도 89는 본 발명의 실시 예에 암호인증 기법 논리를 기반 함수에 대한 부채널 공격 방지를 개념적으로 설명하는 도면이다.

도 90은 함수 1에 대하여 Output 1, Nonce, Associate data를 알 경우 Key 값을 구하는 것이 어려워야 함을 보여주는 도면이다.

도 91은 함수 2에 대해 동일한 Output 2를 갖는 서로 다른 (Output 1, Plaintext)과 (Output 1', Plaintext')을 얻는 것이 어려워야 함을 보여주는 도면이다.

도 92는 함수 3에 대해 Output 2, Tag, Nonce, Associate data가 주어졌을 경우 Key를 찾는 것이 어려워야 함을 보여주는 도면이다.

도 93은 Tag 값이 동일하도록 하는 서로 다른 (Key, Nonce, Associate data, Plaintext)와 (Key', Nonce', Associate data', Plaintext')를 얻는 것이 어려워야 함을 보여주는 도면이다.

도 94는 함수 2에 대해, Output 2를 알지 않은 상태에서도 Output 1과 Ciphertext만 알아도 Plaintext를 계산할 수 있어야 함을 보여주는 도면이다.

도 95는 치환 함수 f에 기반한 암호 인증 기법의 예를 보여주는 도면이다.

도 96은 도 94에 보여준 암호인증 기법을 도 89에 제시된 본 발명에 따라 함수 1, 2, 3 부분으로 나눈 것을 보

여주는 도면이다. 그리고 도 89에서 output 1에 해당하는 값은 도 969에서 $(a_0||b_0)$로 표현된다.

도 97은 Nonce를 직접 생성하지 않고 Key, Associate data, Plaintext로부터 생성하는 함수 4를 보여주는 도면이다.

도 98은 함수 4에서 키 값을 제외한 나머지 입출력 값 정보를 알 경우 키 값을 계산하는 것이 어려워야 함을 보여주는 도면이다.

도 99는 함수 4에서 동일한 출력 값을 생성하는 서로 다른 (Key, Associate data, Plaintext)과(Key', Associate data', Plaintext')를 찾는 것이 어려워야 함을 보여주는 도면이다.

발명을 실시하기 위한 구체적인 내용

[0012] 아래에서는 도면들을 이용하여 본 발명의 기술 분야에서 통상의 지식을 가진 자가 용이하게 실시할 수 있을 정도로 본 발명의 내용을 명확하고 상세하게 기재할 것이다.

[0013] 본 발명에서는 현 상황을 획기적으로 타개할 수 있는 신규 암호화 및 메시지 인증코드 (Message Authentication Code, MAC) 값 생성 논리 및 구현 방법을 제시한다. 해킹 및 다양한 공격 기술의 발달로 인하여, 암호 모듈에 기반한 암호 연산의 필요성이 대두됨에 따라 암호 모듈에 기반한 암호 연산 기법에 대한 연구가 중요시되고 있다. 암호 모듈은 암호 모듈 안에 비밀 키 값과 암호 알고리즘이 있어서 외부의 공격자로부터 안전하게 모듈 내에서 연산을 수행함으로 비밀 정보를 보호하고 비밀 정보를 안전하게 사용하여 원하는 연산을 안전하게 수행하는 데에 있다.

[0014] 암호 모듈은 부채널 공격, 오류 주입 공격 등에 다양한 공격에 안전하도록 설계되어야 하기 때문에, 암호 모듈의 면적이 크면 클수록 설계 및 구현 비용이 증가하므로 암호 모듈 안에는 사용 가능한 메모리 크기에 제약이 따른다.

[0015] 메시지 해싱이나 인증의 경우에는 일반적으로 작은 메모리만으로 구현할 수 있지만, 메시지를 암호화와 인증을 동시에 요구하는 암호인증의 경우에는 지금까지 제안된 대부분의 알고리즘들이 많은 메모리 사용을 필요로 한다. 구체적으로, 암호문의 크기가 증가할수록 요구되는 메모리 크기 또한 증가한다. 따라서, 기존의 대부분의 암호인증 기법은 메모리 사용이 제약된 암호 모듈에서는 빅 데이터에 대한 암호인증 구현이 불가능하다.

[0016] 구체적으로, 암호인증 기법은 암호문에 대한 변경 또는 위조를 파악하기 위하여 태그를 먼저 검증한 후, 태그가 올바른 값이면 그 이후에 암호문을 복호화하여 평문을 출력하도록 되어 있다. 하지만, 현존하는 대부분의 암호 인증 기법은 태그 검증 및 평문 출력에 대해 메모리 사용량에 대한 언급을 하지 않고 있으며, 실제로 무엇보다 빅 데이터에 대해서 큰 메모리 사용을 요구한다.

[0017] 본 발명에서는 저 메모리에 사용가능하도록 하는 암호 인증 방법을 제시한다.

[0018] **일반적인 암호 시스템 설계 및 구현 논리**

[0019] 블록암호 DES가 NIST와 NSA에 의해 1977년 개발되고, ECB, CBC, OFB, CFB와 같이 블록암호를 이용하여 암호화를 수행하는 방식 (블록암호 운영모드라고 부름)이 1981년 미국의 NIST에 의해 FIPS 81로 발표되었다. 암호화 방식 이외에 메시지 인증코드를 생성 하는 방식에도 여러 가지가 있는데, 대표적인 MAC 생성 운영모드로 CBC-MAC, CMAC이 대표적이다. 그리고 암호화와 인증을 모두 제공하는 방식에는 CCM, GCM, OCB등이 있다. 하지만, 이들은 모두 부채널 공격을 고려하지 않고 단지 수학적인 안전성과 구현 효율성에만 근거하여 설계되어, 부채널 공격에 매우 취약하며, 대응 기법에 대한 비용 또한 매우 높아 현실적으로 효율성과 전력 소모에 민감한 구현 환경에서 사용되기에 어려움이 많은 상황이다.

[0020] 이로 인해, 최근 암호학자들은 부채널 공격에 대한 대응 기법 비용을 줄이고 효율성 및 안전성을 늘리는 방식의 암호 설계 논리 연구를 진행하고 있는 중이다. 하지만, 대부분의 연구가 기존의 기법에 대해 아주 미미한 수준의 개선일 뿐, 근본적인 비용을 획기적으로 줄이는 방식이 아니라는 데에 한계를 갖고 있다. 좀 더 구체적으로 설명하면, 대부분의 현존하는 연구가 암호화 과정 동안 수행되는 연산 수를 줄이거나, 또는 덧셈이나 곱셈처럼 복잡한 연산 대신에 효율적인 부채널 대응 기법 적용을 가능케 하는 비트단위의 AND, OR, XOR, <<< 등과 같은 연산에 기반한 암호 시스템 개발에 초점을 맞추고 있다. 하지만, 연산 수를 줄이는 경우, 또 다른 안전성의 문제점을 낳게 되고, 비트 단위의 연산 과정에 기반한 설계 논리 역시, 연산 수가 늘어남에 따라 부채널 대응 기법 비용 또한 크게 증가한다는 점에 있어서 비용 절감 효과는 여전히 미비한 상태이다.

[0021] **현존하는 부채널 공격 및 대응 기법 아이디어**

[0022] 부채널 공격은 Deterministic 암호 설계 기법에 강력한 공격 기법이다. 즉, 동일한 입력 값에 대해 동일한 출력 값을 갖는 경우, 내부 연산 과정에서 발생하는 부채널 정보, 즉 전력 소모량, 전자파 등을 이용하여 내부 비밀 상태 정보 또는 비밀 키 값 추출이 가능하게 된다. 이때, 부채널 정보는 내부 연산 과정 및 상태와 깊은 상관 관계를 갖는다. 따라서, 현존하는 대부분의 부채널 공격 대응 기법들은 내부 상태 정보를 랜덤하게 변형시킨다든가 또는 내부 상태를 노출되지 않도록 추가 연산을 도입하여 부채널 정보로부터 내부 상태 정보를 숨기는 데에 초점을 맞춘다. 이처럼, 부채널 공격에 대한 대응 기법 아이디어는 단순하지만, 현존의 대응 방식을 적용하기에는 비용 측면에서 비현실적인 상황이다.

[0023] 스마트폰 등 스마트 디바이스에 대한 부채널 취약점 공격에 대한 확대가 예상되며, 부채널 공격 기술은 점차 준전문가 저비용 공격 기술로의 변화로 말미암아, 고등 부채널 대응 기법에 대한 요구가 증가하고 있다고 말하고 있다. 하지만, 고등 부채널 대응을 위해 기존의 대응 기법을 적용할 경우 심하게는 300배에서 500배의 성능 저하가 있어서 실제로 사용이 불가능한 상황이며, 따라서 성능 저하를 고도화할 수 있는 신규 부채널 방지를 위한 원천 기술의 확보가 필요함을 알 수 있다.

[0024] 교통 카드처럼 인증 및 지불 용도로 사용할 경우, 주고 받는 데이터 양이 적기 때문에, 성능 저하의 체감도가 낮을 수 있으나, 스마트 디바이스로 디바이스 내의 데이터를 암호화하거나, 또는 암호화 통신 등을 요구할 경우 너무나 큰 비용이 요구되어 실제 구현이 불가능하다. 무엇보다, 배터리에 기반하여 에너지 소비에 매우 민감한 스마트 디바이스의 경우, 현존하는 부채널 대응 기법 적용에 따른 상당한 에너지 손실에 의해 실용성 및 가용성 측면에서 큰 타격을 입게 된다.

[0025] 위처럼 300배에서 500 배의 성능 저하를 가져오는 근본 이유에는 암호, 복호화 및 인증 등을 위한 설계 기법의 암호학적 복잡성에 기인한다. 좀 더 구체적으로 설명하면, 첫째, 현존하는 모든 설계 기법들은 부채널 대응을 위하여 암호, 복호화 및 인증의 기반이 되는 블록암호, 치환 함수, 압축 함수 등에 부채널 대응 기법을 반드시 적용해야 한다. 둘째, 기반이 되는 기반 함수들이 암호학적 안전성을 위하여 복잡하게 설계되어, 기반함수의 복잡성으로 말미암아 효율적인 대응 기법의 적용이 어렵게 된 것이다.

[0026] 이러한 두 가지 이유로 인해, 현존하는 암호 및 인증 등의 설계 방식으로는 효율적인 부채널 대응 기술의 확보가 매우 어려운 상황이다. 이처럼 부채널 대응 기법에 효율적인 암호 및 인증 기법을 개발하기 위하여, 위의 두 가지 문제점 중 하나만이라도 극복해야 하는데, 문제는 현존하는 모든 암호, 복호화, 인증 등의 모든 설계 기법들은 기반 함수가 반드시 부채널 공격에 안전해야 한다는 가정을 두고 있다는 것이고, 둘째로 기반이 되는 함수를 단순화시켜 부채널 대응 기법 비용을 낮추고자 할 경우 기반 함수의 단순성으로 인한 암호학적 안전성에 문제가 발생할 수 있으므로 기반 함수를 단순화시키는 것도 어려운 상황이다.

[0027] 부채널 공격의 핵심은 deterministic 또는 랜덤하지 않은 연산 과정을 수행하는 과정에서 비밀 정보를 부채널 정보로부터 축출하는 데에 있다. 따라서 매번 비밀 키가 랜덤하게 바뀌든지 또는 비밀 난스(Nonce) 값이 매번 랜덤 하게 바뀌든지 해야 적은 연산으로 부채널 공격에 안전한 대응기법 마련이 가능하다. 보통 난스는 공개가 되는 값이나, 본 발명에서는 난스가 비밀 값의 일부인 경우로 고려한다. 만약, 키와 난스가 둘 다 고정되거나 추측이 가능하게 되면, 전체 연산과정의 보호 없이는 부채널 공격에 의해 키 추출이 가능하게 된다.

[0028] 대표적인 블록 암호 국제 표준으로 AES가 있다. 우리나라에서 개발한 국제 표준 블록 암호 알고리즘으로는 SEED, ARIA, HIGHT 가 있다. 본 발명에서는, 통신 환경에 적합한, 임의의 국제 표준 블록 암호 알고리즘으로 구현해도 상관이 없는 암호화 및 인증 코드 값을 생성하도록 돕는 암호시스템 설계 방식을 제안한다. 또한, 치환 함수와 압축 함수에 기반한 방식 또한 제안한다.

[0029] 암호 통신에서 난스 값이 반복되어 사용될 경우, 전체 연산 과정이 deterministic이기 때문에, 부채널 공격에 의해 비밀 키가 노출될 위험이 있다. 따라서, 난스 값은 재 사용이 안되도록 하는 조치가 요구된다. 특히, 복호화 시 난스 재사용을 막아야 한다. 또한, 난스가 노출될 경우, 키가 노출될 위험 또한 높게 되어 난스 자체도 암호화를 통해 보호를 해야 한다. 또한, 메모리가 작은 환경 내에서도 동작할 수 있는 암호 인증 기법이 요구된다.

[0030] 본 발명에서는 이러한 한계 상황을 극복하기 위해 신규 암호인증 기법들을 제안한다. 부채널 공격 대응이 효과적으로 가능할 뿐만 아니라 이는 현존하는 대부분의 알려진 모든 기술들과 달리, 메모리 크기가 작은 암호 모듈에서 암호인증 기능을 효율적으로 수행할 수 있다. 구체적으로, 암호문에 대한 위조 또는 변조를 파악하기 위해 태그 값을 검사하는 데, 이때 작은 메모리 사용으로 효율적으로 구현할 수 있는 설계 논리를 제시한다.

[0031] 본 발명에서 고려하는 구현 환경 모델은 도 1에서 보여주는 것처럼 암호 모듈을 장착한 두 시스템 사이의 통신 환경에 초점을 맞춘다. 암호화 및 인증 코드 값 생성은 암호 모듈이 담당하며, 각 암호 모듈은 자신을 구동하는 시스템조차도 신뢰하지 않는다. 하나의 암호 모듈을 A라고 하고 다른 또 하나의 암호 모듈을 B라고 하자. 본 문서는 어떻게 두 암호 모듈이 안전하게 통신하기 위한 논리 및 구현 방법을 제시한다. 도 2는 부채널 공격 방지 기법이 적용된 일반적인 암호 시스템을 보여주는 도면이다. 도 2에서 보는 바와 같이 일반적인 암호 시스템들은 상기 암호 시스템의 안전성을 위하여 기반 암호 함수를 반드시 부채널 방지 기법에 의해 보호해야만 한다. 또한 기반 암호 함수의 사용 횟수에 따라 부채널 방지 기법에 의한 시간 및 에너지 비용 또한 따라서 함께 증가하게 된다.

[0032] 도 3은 본 발명의 실시 예에 따른 신규 암호 시스템에 대한 부채널 공격 방지를 개념적으로 설명하는 도면이다. 도 2에서 설명한 일반적인 암호 시스템과는 달리 도 3에서 제시된 본 발명 원리는 기반 암호 함수 대신 외부 연산을 보호하는 데에 초점을 맞춘다.

[0033] 외부 연산을 보호하는 목적은 크게 두 가지로 나뉜다. 첫째, 외부 연산을 수행하는 과정 중에 사용되는 비밀 정보를 보호하는 것과, 둘째, 외부 연산을 보호함으로써 기반 암호 함수의 입출력 정보를 보호하는 데에 있다. 상기 외부 연산이 기반 암호 함수에 비해 단순하고, 효율적인 경우라면 부채널 방지 기법 비용 측면에서 보았을 때, 도 2에 비해 도 3이 비용 및 효율성 측면에서 매우 큰 장점을 갖게 된다. 그 이유는 부채널 방지 연산을 수행 시 연산량이 크고 연산 과정이 복잡할수록 부채널 방지 기법 비용이 매우 빠르게 증가하기 때문이다. 기반 암호 함수는 큰 연산량으로 복잡하게 설계되어야 하는 반면, 기반 암호 함수 밖의 외부 연산 과정의 경우 매우 단순하면서도 효율적으로 수행할 수 있기 때문이다.

[0034] 도 3의 방식에서는 기반 암호 함수 대신 외부의 단순한 연산에 대한 보호만 요구되기 때문에 부채널 방지 기법이 적용된다고 할지라도 저비용으로 고 효율성, 고 안전성을 갖도록 암호, 복호화, 인증 기법을 구현할 수 있음을 보여준다.

[0035] **본 발명에 따른 블록 암호 기반 신규 설계 논리 아이디어**

[0036] 현존하는 암호화 및 인증 코드 값 생성 방식들은 기반이 되는 알고리즘이 부채널 공격에 안전하게 구현되어야 한다는 근본적인 문제점을 지니고 있다. 이러한 근본적인 문제점을 안고 가는 이상, 부채널 공격에 대해 저비용, 고효율, 고 안전성을 인갖는 구현 방법은 사실상 없는 상황이다. 따라서, 본 발명에서는 관점을 바꾸어, 운영 모드 차원에서 접근하고자 한다. 어떻게 하면, 기반 알고리즘에 대한 부채널 대응 기법 적용 없이 운영 모드 차원에서의 대응만으로 안전성을 제공할 수 있는가에 대한 것이다. 본 발명에서는, 운영모드 차원에서, 부채널 공격에 대해 저비용, 고 효율성, 고 안전성을 제공하는 암호화 및 인증 코드 값 생성하는 설계 논리 및 구현 방식을 제안한다.

[0037] 부채널 공격의 핵심은 deterministic 또는 랜덤하지 않은 연산 과정을 수행하는 과정에서 비밀 정보를 부채널 정보로부터 축출하는 데에 있다. 따라서 매번 비밀 키가 랜덤하게 바뀌든지 또는 비밀 난스 (Nonce) 값이 매번 랜덤 하게 바뀌든지 해야 적은 연산으로 부채널 공격에 안전한 대응기법 마련이 가능하다. 보통 난스는 공개가 되는 값이나, 본 발명에서는 난스가 비밀 값의 일부인 경우로 고려한다. 만약, 키와 난스가 둘 다 고정되거나 추측이 가능하게 되면, 전체 연산과정의 보호 없이는 부채널 공격에 의해 키 추출이 가능하게 된다.

[0038] 대표적인 블록 암호 국제 표준으로 AES가 있다. 우리나라에서 개발한 국제 표준 블록 암호 알고리즘으로는 SEED, ARIA, HIGHT 가 있다. 본 발명에서는, 통신 환경에 적합한, 임의의 국제 표준 블록 암호 알고리즘으로 구현해도 상관이 없는 암호화 및 MAC 값을 생성하도록 돕는 암호시스템 설계 방식을 제안한다.

[0039] 암호 통신에서 난스 값이 반복되어 사용될 경우, 전체 연산 과정이 deterministic이기 때문에, 부채널 공격에 의해 비밀 키가 노출될 위험이 있다. 따라서, 난스 값은 재 사용이 안되도록 하는 조치가 요구된다. 특히, 복호화시 난스 재사용을 막아야 한다. 또한, 난스가 노출될 경우, 키가 노출될 위험 또한 높게 되어 난스 자체도 암호화를 통해 보호를 해야 한다. 난스 재 사용에 대한 위험은 복호화 과정에서 나타난다. 공격자가 암호문을 위조할 경우 인증 코드 값이 다르게 되어, 인증 코드가 틀리게 나온 난스를 블랙리스트에 올려서 다시는 사용하지 못하도록 하는 것이다. 만약, 블랙 리스트의 크기가 매우 크게 되면, 다시 새롭게 키를 공유하고, 블랙 리스트들을 삭제한다. 이러한 블랙리스트 기반 방식 외에 여기서는 다른 방식들을 또한 소개한다. 난스 열 생성 방식, 마스크 열 생성 방식, 암호화 및 인증 방식과 같이 세 부분으로 나누어 설명하고자 한다.

[0040] **블록 암호 기반 난스 열 생성 방식**

[0041] 다행스럽게도 통신 환경에서의 암호화는 저장 데이터의 암호화가 아닌, 통신 자체의 암호화에 초점을 두고 있기 때문에, 난스 재사용을 막는 방법이 쉽게 구현될 수 있다. 두 암호 모듈 A와 B는 공유된 비밀 키 K로 부터난스 열 N1, N2, N3, …를 생성하여 공유하게 된다. 난스 열을 다루는 방식에는 두 가지로 나누어 살펴볼 수 있는데, 첫째로, 난스 열을 암호 모듈 내에 저장해 놓는 방법이고, 둘째는 난스 열을 실시간으로 생성하는 방법이다.

[0042] 먼저 난스 열을 암호 모듈 내에 저장해 놓는 방법을 설명하면, 암호문 복호화 시 얻어진 난스 값이 테이블에 존재할 경우 올바른 난스 값으로 받아들이고 복호화를 진행한다. 그리고 한번 사용된 난스는 삭제하여 난스 재사용을 막는다. 만약 올바른 난스 값이 아니면 복호화 중지하여 부채널 공격을 무력화시킨다. 두 번 째, 난스 열을 실시간으로 생성하는 방법을 설명하면, 암호문 복호화시 얻어진 난스 값이 사용되기로 예정된 난스 값이면 받아들이고 복호화를 진행하며, 아니면 복호화를 중지한다.

[0043] 예를 들어, 다음의 도 4와 같이 공유된 비밀 키 K로부터 난스 열 N1, N2, N3, …를 생성한다. 공유 키로부터 난스 열을 생성하였기에 두 암호 모듈 A와 B는 난스 열 또한 공유하게 된다. 여기서 보인 예는 한가지 예에 불과하며, 여기서의 핵심은 키를 통해 난스 열을 생성한다는 것이며, 키 K는 초기에만 사용되고 그 이후에는 사용되지 않으며, feedforward 연산 과정으로 인해 역 연산이 어렵도록 하였다.

[0044] 공유 키를 이용하여 새로운 메시지를 암호화할 경우, 난스 사용은 N1부터 순차적으로 사용하며, 만약 복호화 시 사용되어야 할 난스가 사용되지 않을 경우, 암호 모듈은 복호화를 수행하지 않는다. 실제 난스 열을 생성하는 것은 처음 비밀 키 K가 공유된 시점에서 난스 열을 생성하여 암호 모듈 내에 저장하는 방식을 취하거나, 아니면, 실시간으로 매번 이전 난스로부터 다음에 사용될 난스를 생성하여 사용할 수도 있다.

[0045] 위의 난스 열 생성 알고리즘을 보면, 공유 비밀 키 K는 처음에 한번만 사용되고, const는 유일한 공개된 상수 값이며, 매번 비밀 난스 값이 업데이트되어 사용되기에, 어떤 상태에서도 동일한 비밀 정보가 반복적으로 사용되지 않아 부채널 공격 적용이 어렵게 설계되어 있음을 알 수 있다.

[0046] 특히, 난수 열을 생성 시, 단순한 연산 대신 복잡한 블록 암호 연산을 사용하는 이유는, 잘 설계가 된 블록 암호를 사용할 경우, 각 난스 값 생성 시 좋은 난수 성질 또한 갖게 되기 때문이다. 좋은 난수성을 갖는 난스의 사용은 부채널 공격을 대응하는 데에 매우 필수적이다.

[0047] **블록 암호 기반 마스크 열 생성 방식**

[0048] 여기서 우리가 제안하는 암호화 방식을 설명하기 위해 마스크 열 생성에 대해 언급하고자 한다. 마스크 값들은 후에 암호화를 진행할 때 사용되는 공유 비밀 키의 정보를 부채널 공격으로부터 보호하는 데에 있다. 구체적으로, 도 5에서와 같이, 공유 키가 사용된 블록 암호의 입 출력 정보를 랜덤하게 하기 위한 마스크 열 Z_1, Z_2,... 생성한다. 여기서 마스크 열을 생성하기 위하여 난스 열로부터 현재 사용되는 난스를 이용하는데, 이때의 난스를 N으로 표현한다. $const_1$은 난스 열 생성에 사용되는 const와 다른 상수이다.

[0049] 마스크 열을 블록 암호를 이용한 이유도 부채널 공격에 강한 안전성을 제공하기 위하여 블록 암호 입출력 정보에 공격자로 하여금 예측할 수 없도록 좋은 난수성이 요구되기 때문이다. 매번 랜덤한 마스크 값을 사용함으로, 공격자로 하여금 암호화 과정 동안 블록 암호의 입출력 정보를 철저히 숨김으로, 부채널 공격자로 하여금 블록 암호에 사용된 비밀 키 값을 보호할 수 있게 된다.

[0050] **일곱 가지 블록 암호 기반 암호화 및 인증 방식**

[0051] 위에서 설명한 방식으로 난스와 마스크 열이 주어졌을 경우, 먼저 처음 세 가지 암호화 및 인증 방식을 소개한다. 구성 방법은 OCB 모드와 유사한 점이 있으나, OCB 모드는 부채널 공격에 대한 안전성을 고려하지 않고 설계된 암호 인증 방법인 반면, 본 발명에서 제안하는 방식은 부채널 공격까지 고려했을 때에 저비용, 고효율, 고안전성으로 통신 환경에서 구현될 수 있는 암호화 및 인증코드를 생성하는 방식이라는 특징을 지닌다. 네 번째부터 일곱 번째 방식은 ECB 모드와 유사하며 블록 암호에 사용되는 키 값을 난스를 이용하여 매 블록 마다 다르게 생성한다는 특징을 지닌다. 여기서 ECB 모드는 하나의 예로써 다른 모드들에 대한 비슷한 방식의 응용이 가능하

다.

[0052] **블록 암호 기반 암호화 및 인증방식 1**

[0053] 임의의 길이의 메시지 M이 주어졌을 때, M을 블록 암호의 블록 크기 단위로 쪼개어 이를 $M=M_1||M_2||...||M_t$ 와 같이 표기한다. 이때, 마지막M_t의 크기는 1에서 블록 크기 사이의 임의의 값을 취할 수 있다. 그러면, 도 6과 같이 암호화 및 인증을 수행하여, 암호문$C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드T를 얻는다. K는 두 암호 모듈 사이에 공유된 키, $const_2$는 이전에 사용된const, $const_1$과는 다른 상수이며, 그리고 $Checksum=M_1 \oplus M_2 \oplus ... \oplus (M_t||0^*)$ 으로 정의된다.

[0054] 블록 암호 기반 암호화 및 인증 방식 1을 설명하면, 암호문의 일부인 C_0는 난스 N의 암호문이라고 보면 된다. 난스 N을 보호하기 위하여 암호화된 C_0가 상대방 암호 모듈에 안전하지 않은 통신 채널을 통해 전달된다. 암호문을 전달받는 암호 모듈은 암호문을 복호화하기 위하여, 먼저 주어진 암호문의 첫 번째 블록인 C_0로부터 N을 생성하고, N이 앞서 설명한 난스 생성 알고리즘의 요구 조건에 부합할 경우 N으로부터 마스크 열을 생성하면서 동시에 복호화 과정을 수행하여 M을 얻어내고, 최종적으로 인증 코드 값 T이 맞는 경우, M을 올바른 메시지로 받아들이고, 틀리면 메시지에 변조가 있음을 알고, 오류 값을 출력한다.

[0055] 도 7은 부채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 1에서 보호되어야 할 연산을 보여준다.

[0056] **블록 암호 기반 암호화 및 인증 방식 2**

[0057] 첫 번째 방식과 다른 점은 Checksum 생성하는 방법에 있다. 임의의 길이의 메시지 M이 주어졌을 때, M을 블록 암호의 블록 크기 단위로 쪼개어 이를 $M=M_1||M_2||...||M_t$ 와 같이 표기한다. 이때, 마지막M_t의 크기는 1에서 블록 크기 사이의 임의의 값을 취할 수 있다. 그러면, 도 8과 같이 암호화 및 인증을 수행하여, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드T를 얻는다. K는 두 암호 모듈 사이에 공유된 키, $const_3$는 이전에 사용된const, $const_1$, $const_2$와는 다른 상수이며, $Checksum=a_1 \oplus a_2 \oplus ... \oplus a_t \oplus (M_t||0^*)$ 으로 정의된다.

[0058] 도 9는 부 채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 2에서 보호되어야 할 연산을 보여준다.

[0059] **블록 암호 기반 암호화 및 인증 방식 3**

[0060] 두 번째 방식과 다른 점은 Checksum 생성하는 방법과 마지막 메시지 블록 처리에 있다. 임의의 길이의 메시지 M이 주어졌을 때, 10^* 패딩을 이용하여 $M||10^*=M_1||M_2||...||M_t$ 와 같이 표기한다. 그러면, 도 10과 같이 암호화 및 인증을 수행하여, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드T를 얻는다. K는 두 암호 모듈 사이에 공유된 키, $const_3$는 이전에 사용된const, $const_1$, $const_2$와는 다른 상수이며, $Checksum=a_1 \oplus a_2 \oplus ... \oplus a_t$ 으로 정의된다.

[0061] 도 11은 부 채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 3에서 보호되어야 할 연산을 보여준다.

[0062] **블록 암호 기반 암호화 및 인증 방식 4**

[0063] 도 12는 방식 4를 보여 준다. 앞의 두 방식과 다른 점은 마스크 값을 이용하지 않는다는 특징이 있다. 임의의 길이의 메시지 M이 주어졌을 때, M을 블록 암호의 블록 크기 단위로 쪼개어 이를 $M=M_1||M_2||...||M_t$ 와 같이 표기한다. 이때, 마지막 M_t의 크기는 1에서 블록 크기 사이의 임의의 값을 취할 수 있다. 그러면, 다음 그림과 같이 암호화 및 인증을 수행하여, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다. $Checksum=M_1 \oplus M_2 \oplus ... \oplus (M_t||0^*)$ 으로 정의된다.

[0064] 도 13은 부 채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 4에서 보호되어야 할 연산을 보여준다.

[0065] **블록 암호 기반 암호화 및 인증 방식 5**

[0066] 도 14는 방식 5를 보여 준다. 방식 4과 차이점은 마지막 Checksum를 생성할 때 처음 사용된 난스 N을 다시 사용한다는 특징을 지닌다. 임의의 길이의 메시지 M이 주어졌을 때, M을 블록 암호의 블록 크기 단위로 쪼개어 이를 $M=M_1||M_2||...||M_t$ 와 같이 표기한다. 이때, 마지막 M_t의 크기는 1에서 블록 크기 사이의 임의의 값을 취할 수 있다. 그러면, 다음 그림과 같이 암호화 및 인증을 수행하여, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다.

[0067] 도 15는 부 채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 5에서 보호되어야 할 연산을 보여준다.

[0068] **블록 암호 기반 암호화 및 인증 방식 6**

[0069] 도 16은 방식 6을 보여 준다. 도 16을 보면, 방식 5와의 차이점은 매번 블록 암호 연산 앞에 마스크 값을 블록 암호의 키 부분에 적용된 값을 이용하여 생성한다는 특징이 있다. 이러한 방식은 앞에서의 마스크 열이 적용 안된 경우들에 유사하게 적용이 가능하다. 그리고 Checksum을 평문 자체로부터가 아닌, 평문과 마스크 값을 XOR한 이후의 값들을 이용하여 Checksum를 생성한다는 특징이 있다. $Checksum=a_1 \oplus a_2 \oplus ... \oplus a_t \oplus (M_t||0^*)$ 으로 정의된다. 임의의 길이의 메시지 M이 주어졌을 때, M을 블록 암호의 블록 크기 단위로 쪼개어 이를 $M=M_1||M_2||...||M_t$ 와 같이 표기한다. 이때, 마지막 M_t의 크기는 1에서 블록 크기 사이의 임의의 값을 취할 수 있다. 그러면, 다음 그림과 같이 암호화 및 인증을 수행하여, 암호문$C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드T를 얻는다.

[0070] 도 17은 부 채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 6에서 보호되어야 할 연산을 보여준다.

[0071] **블록 암호 기반 암호화 및 인증 방식 7.** 도 18는 방식 7을 보여 준다. $Checksum=a_1 \oplus a_2 \oplus ... \oplus a_t$ 으로 정의된다. 이를 위해, 임의의 길이의 메시지 M이 주어졌을 때, 10^* 패딩을 이용하여 $M=M_1||M_2||...||M_t$ 와 같이 표기한다. 그러면, 다음 그림과 같이 암호화 및 인증을 수행하여, 암호문$C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드T를 얻는다.

[0072] 도 19는 부 채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 7에서 보호되어야 할 연산을 보여준다.

[0073] **블록 암호 기반 안전성 강화 옵션**

[0074] 앞에서 보인 일곱 가지 방식에서 보호되어야 할 연산이 부채널 공격에 의해 안전하게 보호될 경우 안전하다고 말할 수 있으나, 만에 하나, 먼 훗날 신규 부 채널 공격 기술의 발전으로 대응 기법의 취약성으로 임의의 상태의 암호화 키 값 또는 난스 값 또는 마스크 값이 노출되더라도 과거와 미래의 통신 정보를 보호하기 위하여, 이제부터 비밀 공유 키 K로부터 암호화 키 열 생성 알고리즘, 난스 열 생성 알고리즘을 제시한다. 마스크 열 생성 알고리즘 역시 비슷하게 정의될 수 있으나, 실시간 통신의 효율성과 직결되어, 통신의 효율성을 떨어뜨리기에 여기서는 생략하나, 이 역시 가능함을 명시해 둔다.

[0075] **블록 암호 기반 안전성 강화를 위한 암호화 키 열 생성 방식**

[0076] 현재에 사용된 암호화 키가 노출된다고 할지라도, 과거와 이전에 사용된 키를 보호하기 위하여 다음 그림과 같이 매번 암호화에 대해 다르게 사용될 일회용 암호화 키 열$K_1, K_2,...$ 생성한다. 각 일회용 암호화 키는 암호화 시 난스와 동일하게 한 번의 암호화 당 한번 씩 만 사용되고, 다음 번의 암호화 시 암호화 키 열에서의 다음 번의 일회용 암호화 키를 이용한다.

[0077] 앞에서 설명한 방식과의 차이는, 공격자가 임의 상태의 암호화 키 K_i 또는 난스 값 N_i를 알았다고 할지라도, 다른 암호화 키 값들 또는 난스 값들을 보호할 수 있다는 장점을 가진다.

[0078] 도 20에서 보인 예를 보면, 각 암호화 키를 생성하기 위하여, 블록 크기를 네 번 얻어서 각 암호화 키를 정의함을 알 수 있다. 따라서, K_1을 공격자가 얻었다고 할지라도, K_2를 얻기 위해서는 키 열 생성 알고리즘의 내부 전체 블록 상태를 알아야 하는데, 블록 정보씩만 공격자에게 주어지기에, K_2를 얻기 위하여 공격자는 블록 크기에 대해 전수 조사를 해야 하고, 또한 추측한 키가 맞는지에 대해 통신을 통해 검증해야 하는 부가적인 오버헤드가 요구되기에 과거와 미래에 사용될 암호화 키를 보호할 수 있다는 장점을 가진다. 무엇보다, 공유 키 K가 주기적으로 업데이트가 될 경우, 어떤 시점에 사용된 공유 비밀 키에 대응되는 암호화 키 열을 설사 모두 알았다고 할지라도, 과거 혹은 새로 갱신된 공유된 비밀 키 정보를 얻는 데에는 전혀 도움이 되지 않는다.

[0079] **블록 암호 기반 안전성 강화 위한 난스 열 생성 방식**

[0080] 도 21에서 보여주는 것처럼, 난스 열 생성 방식 또한 비슷하게 정의될 수 있다.

[0081] **블록 암호 기반 헤드 정보 등 부가 정보의 인증까지 처리하고자 할 경우**

[0082] 앞에서는 난스와 메시지에 대한 암호문, 그리고 이에 대한 인증 코드 값만이 존재했는데, 때론 헤드 정보 등 공개된 정보와 함께 인증을 처리할 경우가 있다. 헤드 등과 같이 부가적인 정보를 처리하는 암호화 및 인증 방식을 AEAD (Authenticated Encryption with Associate Data) 라고 부른다. 이제부터 부가 정보를 인증할 수 있도록 앞서 제시된 방식들을 어떻게 수정할 것인가에 대해 설명하고자 한다.

[0083] 부가정보를 A라고 할 때, 먼저 A에 패딩을 하여 패딩 후 A의 크기가 블록 크기의 배수가 되도록 한다. 구체적으로, 패딩 방법을 pad라고 할 때, $pad(A)=A_1||...||A_j$ 처럼 패딩 후의 크기가 j 블록으로 표현된다고 하자. 예를 들어, 패딩 방법으로 10* 패딩 방법을 쓸 수 있다.

[0084] 그러면 먼저 방식 1을 도 22와 같이 변형시킬 수 있다. 그리고 $Checksum= \alpha_1 \oplus ... \oplus \alpha_j \oplus M_1 \oplus M_2 \oplus ... \oplus (M_t||0^*)$ 으로 정의하여 도 22와 같이 동작하여, 암호문과 인증 코드를 생성한다. 즉, 부가 정보 A는 암호문의 크기에는 변화를 주지 않고, 단지 인증 코드 생성시에만 영향을 준다. 즉, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드T를 얻는다.

[0085] 방식 2를 도 23과 같이 변형시킬 수 있다. 그리고 $Checksum= \alpha_1 \oplus ... \oplus \alpha_j \oplus a_1 \oplus ... \oplus a_t \oplus (M_t||0^*)$ 으로 정의하여 도 23와 같이 동작하여, 암호문과 인증 코드를 생성한다. 즉, 부가 정보 A는 암호문의 크기에는 변화를 주지 않고, 단지 인증 코드 생성시에만 영향을 준다. 즉, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다.

[0086] 방식 3을 도 24과 같이 변형시킬 수 있다. 그리고 $Checksum= \alpha_1 \oplus ... \oplus \alpha_j \oplus a_1 \oplus ... \oplus a_t$으로 정의하여 도 24와 같이 동작하여, 암호문과 인증 코드를 생성한다. 즉, 부가 정보 A는 암호문의 크기에는 변화를 주지 않고, 단지 인증 코드 생성시에만 영향을 준다. 즉, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다.

[0087] 방식 4를 도 25와 같이 변형시킬 수 있다. 그리고 $Checksum= \alpha_1 \oplus ... \oplus \alpha_j \oplus M_1 \oplus M_2 \oplus ... \oplus (M_t||0^*)$ 으로 정의하여 도 25와 같이 동작하여, 암호문과 인증 코드를 생성한다. 즉, 부가 정보 A는 암호문의 크기에는 변화를 주지 않고, 단지 인증 코드 생성시에만 영향을 준다. 즉, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다.

[0088] 방식 5를 도 26과 같이 변형시킬 수 있다. 그리고 $Checksum= \alpha_1 \oplus ... \oplus \alpha_j \oplus M_1 \oplus M_2 \oplus ... \oplus (M_t||0^*)$ 으로 정의하여 도 26과 같이 동작하여, 암호문과 인증 코드를 생성한다. 즉, 부가 정보 A는 암호문의 크기에는 변

화를 주지 않고, 단지 인증 코드 생성시에만 영향을 준다. 즉, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다. 이때 한가지 예로 부가정보를 처리할 때와 평문을 암호화할 때에 사용된 상수 값을 다르게 했는데, 부가 정보와 암호문 처리 방식을 구별되게 하게 하는 방법에는 여러 가지가 있다. 가령, 부가 정보 및 메시지 패딩 방법에 변화를 주는 방법도 생각할 수 있다.

[0089] 방식 6을 도 27과 같이 변형시킬 수 있다. 그리고 Checksum= $\alpha_1 \oplus ... \oplus \alpha_j \oplus a_1 \oplus ... \oplus a_t \oplus (M_t||0^*)$ 으로 정의하여 도 27과 같이 동작하여, 암호문과 인증 코드를 생성한다. 즉, 부가 정보 A는 암호문의 크기에는 변화를 주지 않고, 단지 인증 코드 생성시에만 영향을 준다. 즉, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다.

[0090] 방식 7을 도 28과 같이 변형시킬 수 있다. 그리고 Checksum= $\alpha_1 \oplus ... \oplus \alpha_j \oplus a_1 \oplus ... \oplus a_t$으로 정의하여 도 28과 같이 동작하여, 암호문과 인증 코드를 생성한다. 즉, 부가 정보 A는 암호문의 크기에는 변화를 주지 않고, 단지 인증 코드 생성시에만 영향을 준다. 즉, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다.

[0091] **블록 암호 기반에서 암호문 없이 인증 코드만 생성하고자 하는 경우**

[0092] 상황에 따라서 암호화 없이 인증 코드만 요구되는 경우 부채널 공격에 안전하도록 인증 코드 값인 MAC 값을 생성하고자 경우, 앞서 제시된 모든 방식들을 인증 코드만 생성하는 MAC 알고리즘으로 변경이 가능하다. 구체적으로, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T 중 $C_1||...||C_t$을 생략한 채, $C_0||T$ 값 전체를 인증 코드로 정의를 내린다.

[0093] **블록 암호 기반에서 인증 코드 없이 암호문만 생성하고자 하는 경우**

[0094] 상황에 따라서 인증 코드에 대한 필요 없이 암호화만 요구되는 경우 부채널 공격에 안전하도록 암호화를 수행하고자 하는 경우, 앞서 제시된 방식들의 출력 값 중 τ-비트 값인 인증 코드 T을 제외한 나머지 암호문 $C(=C_0||C_1||...||C_t)$만을 출력함으로 암호문을 생성할 수 있다.

[0095] **블록 암호 기반에서 보호해야 할 연산의 수를 제로(0)로 만드는 방법**

[0096] 본 발명에서 제시한 방법에서 보호해야 할 연산이 XOR 연산 한 개임을 보였다. 본 절에서는 부채널 공격에 대해 전혀 대응 방법을 적용하지 않고서도 안전하도록 하는 방법을 제안한다. 즉, 보호해야 할 연산의 수를 0으로 줄이는 방법이다. 구체적으로, 난스로부터 C_0 값을 생성하는 방식에 변화를 준다. 우선 도 29을 보면, 지금까지 보인 난스를 다루는 방식을 보여준다. 도 29를 보면, 난스 값을 보호하기 위하여 C_0 값을 생성하는 것을 볼 수 있다.

[0097] 도 29에서 사용된 암호화 키 K 값을 보호하기 위하여 XOR 연산을 부채널 공격에 안전하도록 대응 방법을 적용해야 한다. 도 29에서는 암호화 키 K를 알고 있는 C_0로부터 N를 복호화 해서 사용 가능한 난스 N인지를 판별할 수 있게 된다. 만약 위의 방식을 일방향 성질을 갖는 일방향 함수 f와 난스 N으로부터 도 30과 같이 C_0를 생성한다고 하자. 이때, 더 이상 C_0로부터 직접 N를 복호화할 수는 없으나, 난스 열을 알고 있는 암호 모듈은 사용될 난스를 이미 알고 있는 관계로, 해당 난스 값을 직접 입력하여 C_0 값이 나오는가를 보고, 올바른 난스를 사용했는지 확인이 가능하며, 올바른 난스가 사용된 경우 난스를 이용하여 복호화를 진행한다. 만약, 잘못된 난스라면 복호화를 더 이상 진행하지 않고 중지한다. 매번 일방향 함수를 계산하는 대신 난스 열과 C_0 열을 암호 모듈 안에서 동시에 생성하여, C_0 값으로부터 해당되는 난스 값을 바로 알 수 있게 할 수 있다.

[0098] 구체적으로 f 를 블록 암호 E를 이용하여 도 31와 같이 구현할 수 있다. 도 31를 보면, 록 암호 E에서 Nonce는 블록 암호의 키로 사용되고, 입력 값은 constant가 됨을 알 수 있다.

[0099] **블록 암호 기반에서 난스에 대한 동기화 과정 없는 암호화 및 인증 방식**

[0100] 지금까지 소개된 방식들은 두 암호 모듈 사이에 복호화가 제대로 이루어지기 위해서는 난스가 동기화 되어야 한다는 요구 조건을 필요로 하였다. 만약, 난스에 대한 동기화가 어려운 경우, 난스 열을 동기화하지 않고, 두 암호 모듈은 서로 다른 난스 열을 생성하고, 다만, 복호화 시 인증 코드가 맞지 않은 경우에 각 암호 모듈은 사용된 난스 값을 자신의 모듈 내의 테이블에 저장한다.

[0101] 두 암호 모듈 사이에 테이블을 공유할 필요는 없다. 왜냐하면, 공격자가 이전에 사용된 C_0에 대해 암호문 또는 인증 코드에 변조를 가할 경우, 복호화 시 인증 코드가 불일치하게 되어 공격 목적으로 반복 사용된 C_0 값에 대한 검증이 가능하게 된다. 또한 공격자가 공격에 이용할 수 있는 동일 C_0가 사용된 암호화 및 복호화 과정은 최대 4 번이다. 이는 각 모듈에 동일한 C_0 값이 최대 2번씩 적용될 수 있기 때문이다. 하지만, 동일 C_0가 사용된 4번의 부채널 정보를 이용하여 공격자가 공격하는 것이 현실적으로 어렵기 때문에, 이는 안전성 위협의 고려 대상이 아니다.

[0102] **블록 암호 기반에서 복호화해야 할 암호문의 크기가 큰 경우(즉, 암호 모듈 내의 메모리 크기가 제한적인 경우)**

[0103] 암호문에 대한 변조가 없음이 파악되기 전까지는 암호모듈 밖으로 복호화된 정보, 즉 평문 값의 일부라도 암호 모듈 밖으로 새나가서는 안 된다. 그런데, 만약 복호화해야 할 암호문의 크기가 매우 큰 경우라면 제한된 메모리에 복호화된 정보를 무한정 저장해 놓고 있을 수가 없다. 여기서는 부가 정보를 고려한 암호화 및 인증 방식 5를 수정하여 적은 메모리로 암호문 변조 유무 파악 및 복호화 방식을 가능케 하는 방법을 소개한다. 이는 한 가지 예일 뿐, 비슷한 방식으로 다양한 암호화 및 인증 방식을 설계할 수 있다.

[0104] 도 32와 같이 동작하여, 암호화 시 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다. n은 블록 크기를 뜻하며, n/2 만큼 순환 이동 양으로 예를 들었다. 이제는 복호화하는 방식을 설명한다. 복호화시 암호문과 인증 코드 값으로부터 암호 모듈은 복호화를 진행한다. 진행하는 동안 그림 32에서 부가 정보 처리 이후, 파란색으로 친 부분의 X 값을 암호 모듈은 임시로 저장해 놓는다. 그리고, 최종 인증 코드 T 값이 올바른 경우, X 값을 암호 모듈 밖으로 리턴하게 되면, X 값을 가지고 시스템은 복호화를 진행할 수 있게 된다.

[0105] 이번에는 방식 6를 어떻게 바꿀 수 있는지 살펴보자. 도 33과 같이 동작하여, 암호화시 암호문 $C(=C0||C1||...||Ct)$와 τ-비트 값인 인증 코드 T를 얻는다. 이제는 복호화하는 방식을 설명한다. 복호화시 암호문과 인증 코드 값으로부터 암호 모듈은 복호화를 진행한다. 진행하는 동안 도 33에서 부가 정보 처리 이후, 파란색으로 친 부분의 X 값을 암호 모듈은 임시로 저장해 놓는다. 그리고, 최종 인증 코드 T 값이 올바른 경우, X 값을 암호 모듈 밖으로 리턴하게 되면, X 값을 가지고 시스템은 복호화를 진행할 수 있게 된다.

[0106] 마지막으로 방식 7를 어떻게 바꿀 수 있는지 살펴보자. 도 34과 같이 동작하여, 암호화시 암호문 $C(=C0||C1||...||Ct)$와 τ-비트 값인 인증 코드 T를 얻는다. 이제는 복호화하는 방식을 설명한다. 복호화시 암호문과 인증 코드 값으로부터 암호 모듈은 복호화를 진행한다. 진행하는 동안 도 34에서 부가 정보 처리 이후, 파란색으로 친 부분의 X 값을 암호 모듈은 임시로 저장해 놓는다. 그리고, 최종 인증 코드 T 값이 올바른 경우, X 값을 암호 모듈 밖으로 리턴하게 되면, X 값을 가지고 시스템은 복호화를 진행할 수 있게 된다.

[0107] **블록 암호 기반 키 공유 관련 비용에 관하여**

[0108] 먼저 공유 키를 암호 통신을 필요로 하는 두 암호 모듈에 직접 저장하는 방법을 사용할 수도 있다. 이는 높은 안전성을 요구하는 국방 혹은 의료 분야 등에서 사용될 수 있는 방법이다. 아니면, 공개키 기반 구조에 기반하여 통신하는 시점에 키 공유가 이루어지는 경우도 있다. 안전하지 않은 채널을 통한 키 공유는 일반적으로 공개키 기반 구조에 기반하여 설계될 수 있는데, 공개키 암호는 대칭키 암호에 비해서 속도가 현저히 느리나, 키와 같이 매우 짧은 데이터를 한번 만 처리하면 되기에 키 공유에 따른 부채널 대응 기법 비용이 전체 암호 통신 효율성 및 전력 소모량 측면에서 별 영향을 주지 않는다.

[0109] 따라서 공개키 기반 구조를 사용하든 하지 않든, 실제 부채널 대응 비용은 비밀 키가 공유된 시점으로부터 대부분이 발생하기 때문에, 본 논문에서 제시된 방식이 크게 실제 암호 통신의 가용성 및 효용성 그리고 안전성에 큰 기여를 할 것으로 예상된다.

[0110] 본 발명에서 제시된 안전한 암호 시스템을 위한 설계 논리는 앞으로 스마트 디바이스에서 부채널 공격에 대한 저메모리, 고안전성, 고효율성, 저비용을 요구할 경우에 사용될 수 있는 논리이며, 현존하는 설계 논리에 대한 근본적인 문제점을 해결한다는 점에 있어서 가치가 있다고 볼 수 있다.

[0111] 또한, 향후 본 발명에서 제시한 논리의 방향에 의해 설계된 암호, 복호화, 인증 등 다양한 암호시스템 설계 기법의 신규 표준화가 요구되며, 이를 기초로 저비용으로 부채널 공격에 안전하면서도 고효율성을 갖는 스마트 디바이스 개발 및 제품화가 가능할 것으로 예상된다. 또한 본 발명을 통해 스마트 디바이스를 통한 다양한 신규 서비스 창출이 예상된다. 가령, 데이터 암호화, 음성 암호화, 영상 암호화, 온라인 결제, 인증 등을 스마트 디바이스 상에서 현실적으로 가능하게 된다.

[0112] 한편, 상술 된 본 발명의 내용은 발명을 실시하기 위한 구체적인 실시 예들에 불과하다. 본 발명은 구체적이고 실제로 이용할 수 있는 수단 자체뿐 아니라, 장차 기술로 활용할 수 있는 추상적이고 개념적인 아이디어인 기술적 사상을 포함할 것이다.

[0113] **본 발명에서 제시하는 치환 함수 기반 신규 설계 논리 아이디어**

[0114] 현존하는 암호화 및 인증 코드 값 생성 방식들은 기반이 되는 알고리즘이 부채널 공격에 안전하게 구현되어야 한다는 근본적인 문제점을 지니고 있다. 이러한 근본적인 문제점을 안고 가는 이상, 부채널 공격에 대해 저비용, 고효율, 고 안전성을 인갖는 구현 방법은 사실상 없는 상황이다. 따라서, 본 발명에서는 관점을 바꾸어, 운영 모드 차원에서 접근하고자 한다. 어떻게 하면, 기반 알고리즘에 대한 부채널 대응 기법 적용 없이 운영 모드 차원에서의 대응만으로 안전성을 제공할 수 있는가에 대한 것이다. 본 발명에서는, 운영모드 차원에서, 부채널 공격에 대해 저비용, 고 효율성, 고 안전성을 제공하는 암호화 및 인증 코드 값 생성하는 설계 논리 및 구현 방식을 제안한다.

[0115] 부채널 공격의 핵심은 deterministic 또는 랜덤하지 않은 연산 과정을 수행하는 과정에서 비밀 정보를 부채널 정보로부터 축출하는 데에 있다. 따라서 매번 비밀 키가 랜덤하게 바뀌든지 또는 비밀 난스 (Nonce) 값이 매번 랜덤 하게 바뀌든지 해야 적은 연산으로 부채널 공격에 안전한 대응기법 마련이 가능하다. 보통 난스는 공개가 되는 값이나, 본 발명에서는 난스가 비밀 값의 일부인 경우로 고려한다. 만약, 키와 난스가 둘 다 고정되거나 추측이 가능하게 되면, 전체 연산과정의 보호 없이는 부채널 공격에 의해 키 추출이 가능하게 된다.

[0116] 치환 함수가 기반한 여러 대표적인 알고리즘으로는 SHA-3, Grstl, JH 등이 있다. 본 발명에서는, 통신 환경에 적합한, 임의의 치환 함수 알고리즘으로 구현해도 상관이 없는 암호화 및 인증 코드 값을 생성하도록 돕는 암호 시스템 설계 방식을 제안한다.

[0117] 암호 통신에서 난스 값이 반복되어 사용될 경우, 전체 연산 과정이 deterministic이기 때문에, 부채널 공격에 의해 비밀 키가 노출될 위험이 있다. 따라서, 난스 값은 재 사용이 안되도록 하는 조치가 요구된다. 특히, 복호화시 난스 재사용을 막아야 한다. 또한, 난스가 노출될 경우, 키가 노출될 위험 또한 높게 되어 난스 자체도 암호화를 통해 보호를 해야 한다. 난스 재 사용에 대한 위험은 복호화 과정에서 나타난다. 공격자가 암호문을 위조할 경우 인증 코드 값이 다르게 되어, 인증 코드가 틀리게 나온 난스를 블랙리스트에 올려서 다시는 사용하지 못하도록 하는 것이다. 만약, 블랙 리스트의 크기가 매우 크게 되면, 다시 새롭게 키를 공유하고, 블랙 리스트들을 삭제한다. 이러한 블랙리스트 기반 방식 외에 여기서는 다른 방식들을 또한 소개한다. 난스 열 생성 방식과 암호화 및 인증 방식과 같이 두 부분으로 나누어 설명하고자 한다.

[0118] **치환 함수 기반에서 난스 열 생성 방식**

[0119] 다행스럽게도 통신 환경에서의 암호화는 저장 데이터의 암호화가 아닌, 통신 자체의 암호화에 초점을 두고 있기 때문에, 난스 재사용을 막는 방법이 쉽게 구현될 수 있다. 두 암호 모듈 A와 B는 공유된 비밀 키 K로 부터 두 개의 난스 열 N_1, N_2, N_3,...와 N'_1, N'_2, N'_3,...를 생성한다.

[0120] 난스 열을 다루는 방식에는 두 가지로 나누어 살펴볼 수 있는데, 첫째로, 난스 열을 암호 모듈 내에 저장해 놓는 방법이고, 둘째는 난스 열을 실시간으로 생성하는 방법이다. 먼저 난스 열을 암호 모듈 내에 저장해 놓는 방법을 설명하면, 암호문 복호화 시 얻어진 난스 값이 테이블에 존재할 경우 올바른 난스 값으로 받아들이고 복호화를 진행한다. 그리고 한번 사용된 난스는 삭제하여 난스 재사용을 막는다. 만약 올바른 난스 값이 아니면 복

호화 중지하여 부채널 공격을 무력화시킨다. 두 번 째, 난스 열을 실시간으로 생성하는 방법을 설명하면, 암호문 복호화시 얻어진 난스 값이 사용되기로 예정된 난스 값이면 받아들이고 복호화를 진행하며, 아니면 복호화를 중지한다.

[0121] 예를 들어, 다음의 도 35와 같이 공유된 비밀 키 K로 부터난스 열 N_1, N_2, N_3,...와 N'_1, N'_2, N'_3,...를 생성한다. 공유 키로부터 난스 열을 생성하였기에 두 암호 모듈 A와 B는 난스 열 또한 공유하게 된다. 여기서 보인 예는 한가지 예에 불과하며, 본 발명은 키를 통해 난스 열을 생성한다. 키 K는 초기에만 사용되고 그 이후에는 사용되지 않는다. Feedforward 연산 과정으로 인해 역 연산을 어렵게 한다.

[0122] 공유 키를 이용하여 새로운 메시지를 암호화할 경우, 난스 사용은 (N_1, N'_1) 부터 순차적으로 사용하며, 만약 복호화 시 사용되어야 할 난스가 사용되지 않을 경우, 암호 모듈은 복호화를 수행하지 않는다. 실제 난스 열을 생성하는 것은 처음 비밀 키 K가 공유된 시점에서 난스 열을 생성하여 암호 모듈 내에 저장하는 방식을 취하거나, 아니면, 실시간으로 매번 이전 난스로부터 다음에 사용될 난스를 생성하여 사용할 수도 있다.

[0123] 위의 난스 열 생성 알고리즘을 보면, 공유 비밀 키 K는 처음에 한번만 사용되고, const는 공개된 상수 값이며, 매번 비밀 난스 값이 업데이트되어 사용되기에, 어떤 상태에서도 동일한 비밀 정보가 반복적으로 사용되지 않아 부채널 공격 적용이 어렵도록 설계된다.

[0124] 특히, 난수 열을 생성 시, 단순한 연산 대신 복잡한 압축 함수 연산을 사용하는 이유는, 잘 설계가 된 치환 함수를 사용할 경우, 각 난스 값 생성 시 좋은 난수 성질 또한 갖게 되기 때문이다. 좋은 난수성을 갖는 난스의 사용은 부채널 공격을 대응하는 데에 매우 필수적이다.

[0125]

[0126] **네 가지 치환 함수 기반 암호화 및 인증 방식**

[0127] 위에서 설명한 방식으로 난스 열이 주어졌을 경우, 네 가지 치환 함수 기반 암호화 및 인증 방식을 소개한다. 구성 방법은 SpongeWrap 모드와 유사한 점이 있으나, SpongeWrap모드는 부채널 공격에 대한 안전성을 고려하지 않고 설계된 암호 인증 방법인 반면, 본 문서에서 제안하는 방식은 부채널 공격까지 고려했을 때에 저비용, 고효율, 고안전성으로 통신 환경에서 구현될 수 있는 암호화 및 인증코드를 생성하는 방식이라는 특징을 지닌다.

[0128] **치환 함수 기반 암호화 및 인증방식 1**

[0129] 임의의 길이의 메시지 M를 처리하기 위해서 패딩 방법 pad를 적용한 후, $pad(M)=M_1||M_2||...||M_t$ 와 같이 표기한다. 특히, 안전성을 위하여 패딩 방법 pad는 prefix-free이어야 한다. 이는 임의의 서로 다른 두 메시지 M, M'에 대해, pad(M)은 결코 pad(M')의 prefix가 되어서는 안된다는 뜻이다. 도 36에서 보면, IV1과 IV2는 상수 값을 뜻하며, 이전에 사용된 const와 다른 상수 값이어야 한다. 그러면, 다음 그림과 같이 암호화 및 인증을 수행하여, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드T를 얻는다.

[0130] 치환 함수 기반 암호화 및 인증 방식 1을 설명하면, 암호문의 일부인 C_0는 난스 N의 암호문이라고 보면 된다. 난스 N을 보호하기 위하여 암호화된 C_0가 상대방 암호 모듈에 안전하지 않은 통신 채널을 통해 전달된다. 암호문을 전달받는 암호 모듈은 암호문을 복호화하기 위하여, 먼저 주어진 암호문의 첫번째 블록인C_0로부터 N을 생성하고, N이 앞서 설명한 난스 생성 알고리즘의 요구 조건에 부합할 경우 N과 N'를 얻고, 복호화 과정을 수행하여 $M_1||...||M_t$를 얻어내고, 최종적으로 인증 코드 값 T이 맞고 동시에 $pad(M)= M_1||...||M_t$ 인 M이 존재하면 M을 올바른 메시지로 받아들이고 M을 최종 출력하며, 틀리면 메시지에 변조가 있음을 알고, 오류 값을 출력한다.

[0131] 부채널 공격 대응 방법 관점에서 방식 1의 설계 논리를 좀 더 설명하면, 매번 암호화 마다 다른 난스를 사용하므로, 치환 함수의 입력 값이 매번 랜덤하게 변하게 되어, 공격자로 하여금 내부 상태 값을 얻기가 어렵도록 만들었다. 도 37은 부채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 1에서 보호되어야 할 연산을 보여준다. 놀랍게도 메시지 크기와 상관 없이 초기의 XOR 연산을 최대 두 번만 보호만 해도 안전하게 된다.

[0132] **치환 함수 기반 암호화 및 인증 방식 2**

[0133] 첫 번째 방식과 차이점은 임의의 패딩 함수 pad를 사용해도 된다는 장점이 있다. 단, 마지막 블록 처리를 다르게 하기 위해 const1를 연산하는 과정을 추가하였다. 다음 도 38과 같이 암호화 및 인증을 수행하여, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다.

[0134] 치환 함수 기반 암호화 및 인증 방식 2를 설명하면, 암호문의 일부인 C_0는 난스 N의 암호문이라고 보면 된다. 난스 N을 보호하기 위하여 암호화된 C_0가 상대방 암호 모듈에 안전하지 않은 통신 채널을 통해 전달된다.

[0135] 암호문을 전달받는 암호 모듈은 암호문을 복호화하기 위하여, 먼저 주어진 암호문의 첫 번째 블록인 C_0로부터 N을 생성한다. N이 앞서 설명한 난스 생성 알고리즘의 요구 조건에 부합할 경우 N과 N'를 얻고, 복호화 과정을 수행하여 $M_1||...||M_t$를 얻는다. 최종적으로 인증 코드값 T이 맞고 동시에 $pad(M)= M_1||...||M_t$ 인 M이 존재하면 M을 올바른 메시지로 받아들이고 M을 최종 출력하며, 틀리면 메시지에 변조가 있음을 알고, 오류 값을 출력한다.

[0136] 도 39는 부 채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 2에서 보호되어야 할 연산을 보여준다.

[0137] **치환 함수 기반 암호화 및 인증 방식 3**

[0138] 두 번째 방식처럼 임의의 패딩 함수 pad를 사용해도 되며, 단 마지막 블록 처리를 다르게 하기 위해 상수 값을 더하는 대신, 마지막 인증 코드 생성 값 처리를 다르게 하였다. 도 40과 같이 암호화 및 인증을 수행하여, 암호문$C(=C_0||C_1||...||C_t)$와 -비트 값인 인증 코드 T를 얻는다.

[0139] 도 41은 부 채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 3에서 보호되어야 할 연산을 보여준다.

[0140] **치환 함수 기반 암호화 및 인증 방식 4**

[0141] 방식 4는 방식 1을 좀 더 변형한 방식으로, 암호화하는 과정 동안 공격자가 내부 상태 값을 안다고 할지라도 공격자는 이전의 암호문 및 올바른 인증 코드를 생성할 수 없도록 설계된다. 임의의 길이의 메시지 M를 처리하기 위해서 임의의 패딩 방법 pad를 적용한 후, $pad(M)=M_1||M_2||...||M_t$ 와 같이 표기한다.

[0142] 여기서는 방식 1과는 달리 패딩 방법 pad가 prefix-free일 필요 없다. 그 이유는 마지막에 N과 N'을 XOR 하는 연산을 수행하기 때문이다. 도 42와 같이 암호화 및 인증을 수행하여, 암호문$C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드T를 얻는다. 방식 4의 특징은 매번 FeedFoward 연산 과정을 적용하고, 마지막에 난스 값 N, N'을 다시 적용함으로 N, N' 값을 모르고서는 내부 상태 정보를 알더라도 올바른 인증 코드 값을 생성할 수 없도록 한다.

[0143] 도 43은 부 채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 4에서 보호되어야 할 연산을 보여준다.

[0144] **치환 함수 기반에서 안전성 강화 옵션**

[0145] 앞에서 보인 치환 함수 기반 네 가지 방식에서 보호되어야 할 연산이 부채널 공격에 의해 안전하게 보호될 경우 안전하다고 말할 수 있으나, 만에 하나, 먼 훗날 신규 부 채널 공격 기술의 발전으로 대응 기법의 취약성으로 임의의 상태의 난스 값이 노출되면, 위의 방식들이 역연산을 할 수 있기 때문에, 공유 키 K가 쉽게 노출될 염려가 있다. 결국 공유 키의 노출은 과거의 통신 정보나 미래의 통신 정보 마저 노출될 위험을 안고 있다.

[0146] 이처럼 극단적인 상황에서조차도, 과거와 미래의 통신 정보를 보호하기 위하여, 본 장에서는 비밀 공유 키 K로부터 암호화 키 열 생성 알고리즘, 난스 열 생성 알고리즘을 제시한다. 실시간 통신의 효율성과 직결되어, 통신의 효율성을 떨어뜨리기에 이 장에서는 생략하나, 이 역시 가능함을 명시해 둔다.

[0147] **치환 함수 기반에서 안전성 강화를 위한 암호화 키 열 생성 방식**

[0148] 현재에 사용된 암호화 키가 노출된다고 할지라도, 과거와 이전에 사용된 키를 보호하기 위하여 도 44와 같이 매번 암호화에 대해 다르게 사용될 일회용 암호화 키 열K_1, K_2,... 생성한다. 각 일회용 암호화 키는 암호화 시 난스와 동일하게 한번의 암호화 당 한번씩만 사용되고, 다음 번의 암호화 시 암호화 키 열에서의 다음 번의 일회용 암호화 키를 이용한다.

[0149] 앞에서 설명한 방식과의 차이는, 공격자가 임의 상태의 암호화 키 K_i 또는 난스 값 N_i를 알았다고 할지라도, 다른 암호화 키 값들 또는 난스 값들을 보호할 수 있다는 장점을 가진다. K_1을 공격자가 얻었다고 할지라도, K_2를 얻기 위해서는 키 열 생성 알고리즘의 내부 전체 블록 상태를 알아야 한다. 이를 위하여 공격자는 나머지 c 비트 정보에 대한 전수 조사를 해야 한다.

[0150] 또한 추측한 키가 맞는지에 대해 통신을 통해 검증해야 하는 부가적인 오버헤드가 요구되기에 과거와 미래에 사용될 암호화 키를 보호할 수 있다는 장점을 가진다. 무엇보다, 공유 키 K가 주기적으로 업데이트가 될 경우, 어떤 시점에 사용된 공유 비밀 키에 대응되는 암호화 키 열을 설사 모두 알았다고 할지라도, 과거 혹은 새로 갱신된 공유된 비밀 키 정보를 얻는 데에는 전혀 도움이 되지 않는다.

[0151] **치환 함수 기반에서 안전성 강화 위한 난스 열 생성 방식**

[0152] 난스 열 생성 방식은 약간 차이가 있는데, 예를 들어, 도 45와 같이, 매번 N_i, N'_i 값을 얻기 위하여, 1/4 씩 정보만을 수집하여 계산한다. 따라서, 공격자가 만의 하나 난스 열 중 특정 난스 값을 얻었다고 할지라도, 이전의 난스와 이후의 난스를 얻는 데에는 도움이 되지 않음을 알 수 있다.

[0153]

[0154] **치환 함수 기반에서 헤드 정보 등 부가 정보의 인증까지 처리하고자 할 경우 옵션**

[0155] 앞에서는 난스와 메시지에 대한 암호문, 그리고 이에 대한 인증 코드 값만이 존재한다. 그런데 헤드 정보 등 공개된 정보와 함께 인증을 처리할 경우가 있다. 헤드 등과 같이 부가적인 정보를 처리하는 암호화 및 인증 방식을 AEAD (Authenticated Encryption with Associate Data) 라고 부른다. 이하에서는 부가 정보를 인증할 수 있도록 앞서 제시된 방식들을 어떻게 수정할 것인가에 대해 설명하고자 한다.

[0156] 부가정보를 A라고 할 때, 먼저 A에 패딩을 하여 패딩 후 A의 크기가 블록 크기의 배수가 되도록 한다. 구체적으로, 패딩 방법을 pad라고 할 때, $pad(A)=A_1||...||A_j$ 처럼 패딩 후의 크기가 j 블록으로 표현된다고 하자. 예를 들어, 패딩 방법으로 10* 패딩 방법을 쓸 수 있다.

[0157] 그러면 먼저 네 가지 방식에 대해 도 46부터 49에서 보여준 바와 같이 변형시켜 암호문과 인증 코드를 생성한다. 즉, 부가 정보 A는 암호문의 크기에는 변화를 주지 않고, 단지 인증 코드 생성시에만 영향을 준다. 즉, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다.

[0158] **치환 함수 기반에서 r값이 작은 경우의 구현 방법에 대하여**

[0159] 블록 사이즈 r 값이 작게 되면, 비밀 키 값, 난스 값, 인증 코드 값의 비트 길이가 여러 블록이 될 수도 있다. 하지만, 보통은 키 값, 난스 값, 태그 값들은 일반적으로 고정된 길이의 비트 열이 되기에, 본 명세서에서는 이들의 비트 길이가 고정된 경우라 생각하고 앞서 설명한 방식들을 어떻게 변경할 것인가에 대한 아이디어를 설명하고자 한다.

[0160] **치환 함수 기반 r 값이 작은 경우의 난스 열 생성 방식**

[0161] 치환 함수의 블록 크기 (r+c) 비트 크기 중 안전성의 문제로 인해 c 값은 반드시 커야 한다. 반면, r 값은 임의의 크기가 될 수 있어서 r 값이 작은 경우, 어느 정도의 안전도를 위해 적정 키 크기를 가져야 하는 비밀 키 K를 r 비트로 표현하는 데에 무리가 있다. 따라서, 다음 도 50과 같은 방식으로 난스 열 생성 방식을 수정할 수 있다.

[0162] 키 값은 아무리 커도 r+c 비트 값보다는 작다고 가정할 때, $keypad(K)=K*_1||K*_2$ 라고 하자. 여기서 keypad는 패

딩 방법으로 10* 패딩이 될 수도 있고, 단순히 0* 패딩이 될 수도 있다. 이때K_1은 r 비트, K_2는 c 비트 값이 된다. 이때 난스 열을 도 50과 같이 생성한다. 여기서 feedforward 연산은 생략될 수 있다.

[0163] 위의 방식 대신 K 값을 여러 r 비트 값으로 나누어 각 r 비트 마다 치환 함수를 적용하는 방법도 있으나, 위의 방식에 비해 구현 효율성이 떨어지는 관계로 생략한다.

[0164] **치환 함수 기반에서 r 값이 작은 경우에 대한 암호화 키 열 생성 방식**

[0165] 공유 키 K로부터 암호화 키 열을 생성하고자 할 경우 다음과 같이 암호화 키를 생성할 수 있다. 키 값은 아무리 커도 r+c 비트 값보다는 작다고 가정할 때, keypad(K)=$K*_1||K*_2$ 라고 하자. 여기서 keypad는 패딩 방법으로 10* 패딩이 될 수도 있고, 단순히 0* 패딩이 될 수도 있다. 이때 K_1은 r 비트, K_2는 c 비트 값이 된다. 이때 암호화 키 열은 도 51과 같은 방식으로 생성할 수 있다.

[0166] **치환 함수 기반에서 r 값이 작은 경우에 대한 암호화 및 인증 방식**

[0167] 앞서 r 값이 작은 경우에 대해 난스 열, 암호화 키 열 생성 방법을 설명하였다. 이를 기초로, 암호화 및 인증 방식을 설명하고자 한다. 앞 서 네 가지 방식을 설명했는데, 여기서 부가 정보가 추가된 경우의 네 가지 방식을 어떻게 변경할 것인가를 살펴본다. 여기서 소개하는 방식으로, 부가 정보 인증이 필요 없는 경우에도 동일하게 적용할 수 있음을 말해 둔다.

[0168] 난스 열 중 현재 상태의 난스 값을 N, N'으로 표현하고, 암호화 키 열 중 현재 상태의 암호화 키 값을 K, K' 로 표현한다. 각 방식들에 대해, 복호화 시 C_0로부터 N의 값이 올바른 값인지 난스 열을 통해서 검증 한 후, N이 올바른 값이면 복호화를 진행한다.

[0169] 여기서, r의 값이 1 비트 정보와 같이 작게 되면, 임의의 변조된 C_0 값에 대해서 성립할 확률은 1/2이 된다. 하지만, 이 경우 설사 변조에 성공하더라도, 나머지 N' 값의 비트 크기가 크기 때문에 내부 상태 값들을 랜덤하게 만들게 된다. 다만, r의 값이 1이 되면, 바로 난스의 변조 유무 또는 반복 유무를 파악하는 시간이 늦어질 뿐, 결국 인증 코드로 변조 유무 또는 반복 사용 여부를 파악할 수 있게 된다. 따라서, 난스의 변조 유무 또는 반복 유무를 일찍 파악하기 위하여, r이 최소 10 비트 이상이 될 것을 권장한다.

[0170] 그러면 네 가지 방식을 도 52에서 도 55에서 보인 바와 같이 변형시켜 암호문과 인증 코드를 생성한다. 즉, 부가 정보 A는 암호문의 크기에는 변화를 주지 않고, 단지 인증 코드 생성시에만 영향을 준다. 즉, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다.

[0171] **치환 함수 기반에서 암호문 없이 인증 코드만 생성하고자 하는 경우**

[0172] 상황에 따라서 암호화 없이 인증 코드만 요구되는 경우 부채널 공격에 안전하도록 인증 코드 값인 인증 코드 값을 생성하고자 경우, 앞서 제시된 모든 방식들을 인증 코드만 생성하는 MAC 알고리즘으로 변경이 가능하다. 구체적으로, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T 중 $C_1||...||C_t$을 생략한 채, $C_0||T$ 값 전체를 인증 코드로 정의를 내린다.

[0173] **치환 함수 기반에서 인증 코드 없이 암호문만 생성하고자 하는 경우**

[0174] 상황에 따라서 인증 코드에 대한 필요 없이 암호화만 요구되는 경우 부채널 공격에 안전하도록 암호화를 수행하고자 하는 경우, 앞서 제시된 방식들의 출력 값 중 τ-비트 값인 인증 코드 T을 제외한 나머지 암호문 $C(=C_0||C_1||...||C_t)$만을 출력함으로 암호문을 생성할 수 있다.

[0175] **치환 함수 기반에서 내부 상태 값이 공격자에게 노출되었을 때 암호화 키 또는 난스 값을 보호하는 기법에 관하여**

[0176] 지금까지 제시한 방식들은 내부 상태 값이 공격자에게 노출될 경우, 역 연산이 가능하게 되어 암호화 키 또는 난스 값이 노출될 위험이 노출한다. 이러한 상황은 구현 상의 실수에 의해서 발생할 수도 있고, 아니면 암호화 모듈이 아닌 일반 CPU에서 암호화를 수행하는 경우 발생할 수도 있다. 따라서, 이러한 극단적인 상황 속에서도 사용된 난스 또는 암호화 키 값을 보호하기 위하여 암호화하는 동안 치환 함수 과정 시 feedforward 연산 과정을 삽입함으로써 역연산이 어렵도록 할 수 있게 된다. 이때 역연산의 어려움은 바로 난스 값 혹은 암호화 키를 공격자가 얻는 것이 어렵게 하는 데 효과적인 대응 방법으로 사용될 수 있다.

[0177] **치환 함수 기반에서 난스 값만을 이용한 암호화 및 인증 방법**

[0178] 지금까지는 암호화 키 (열)와 난스 열을 기반한 암호화 및 인증 방법을 소개하였다. 여기서는 암호화 키 없이 난스 열만을 이용한 암호화 방식을 소개한다. 여기서 소개한 방식은 부가 정보가 적용된 경우에 대해서도 동일하게 적용된다. 또한, 여기서 소개하는 방식들은 부채널 공격 대응 기법 적용이 전혀 필요가 없는 방식들이다.

[0179] **치환 함수 기반 난스 기반 암호화 및 인증 방식 1**

[0180] 앞서 소개한 방식 1을 난스 기반으로 변환하는 방법을 제시한다. 도 56을 보면 난스 열 중 해당 (N,N') 을 이용하여 복호화 하고자 하는 경우, C_0로부터 동일한 N의 값이 복호화가 되면, 올바른 난스로 인식하고, 계속해서 복호화를 진행하고, 최종 인증 코드 T 값이 올바르고, 메시지 패딩이 올바르게 되어 있으면, 해당 메시지를 출력하고, 만약 오류가 있으면 메시지 출력 없이 오류가 있다고만 출력한다.

[0181] **치환 함수 기반 난스 기반 암호화 및 인증 방식 2**

[0182] 앞서 소개한 방식 2를 난스 기반으로 변환하는 방법을 제시한다. 도 57를 보면 난스 열 중 해당 (N,N') 을 이용하여 복호화하고자 하는 경우, C_0로부터 동일한 N의 값이 복호화가 되면, 올바른 난스로 인식하고, 계속해서 복호화를 진행하고, 최종 인증 코드 T 값이 올바르고, 메시지 패딩이 올바르게 되어 있으면, 해당 메시지를 출력하고, 만약 오류가 있으면 메시지 출력 없이 오류가 있다고만 출력한다.

[0183] **치환 함수 기반 난스 기반 암호화 및 인증 방식 3**

[0184] 앞서 소개한 방식 3을 난스 기반으로 변환하는 방법을 제시한다. 도 58를 보면 난스 열 중 해당 (N,N') 을 이용하여 복호화하고자 하는 경우, C_0로부터 동일한 N의 값이 복호화가 되면, 올바른 난스로 인식하고, 계속해서 복호화를 진행하고, 최종 인증 코드 T 값이 올바르고, 메시지 패딩이 올바르게 되어 있으면, 해당 메시지를 출력하고, 만약 오류가 있으면 메시지 출력 없이 오류가 있다고만 출력한다.

[0185] **치환 함수 기반 난스 기반 암호화 및 인증 방식 4**

[0186] 앞서 소개한 방식 4를 난스 기반으로 변환하는 방법을 제시한다. 도 59를 보면 난스 열 중 해당 (N,N') 을 이용하여 복호화 하고자 하는 경우, C_0로부터 동일한 N의 값이 복호화가 되면, 올바른 난스로 인식하고, 계속해서 복호화를 진행하고, 최종 인증 코드 T 값이 올바르고, 메시지 패딩이 올바르게 되어 있으면, 해당 메시지를 출력하고, 만약 오류가 있으면 메시지 출력 없이 오류가 있다고만 출력한다.

[0187] **치환 함수 기반 난스에 대한 동기화 과정 없는 암호화 및 인증 방식**

[0188] 지금까지 소개된 방식들은 두 암호 모듈 사이에 복호화가 제대로 이루어지기 위해서는 난스가 동기화 되어야 한다는 요구 조건을 필요로 한다. 만약, 난스에 대한 동기화가 어려운 경우, 초기 XOR 연산에 대한 부채널 공격 대응 기법 적용이 요구되며, 그리고 난스 N에 대한 암호화 값인 C_0로부터 난스 N을 계산 한 후, N으로부터 N'을 생성할 수 있는 방식을 따로 개발해야 한다. 예를 들어, 도 60과 같은 방식으로 N으로부터 N'을 생성한다. 단,

N'의 난수성은 N의 난수성에 의존하기에, N의 비트 크기가 커야 한다는 요구 사항이 추가된다.

[0189] 복호화 시 인증 코드가 맞지 않은 경우에 각 암호 모듈은 사용된 난스 값을 자신의 모듈 내의 테이블에 저장한다. 이는 사용을 금지해야 할 난스 값들을 저장한 테이블이다. 두 암호 모듈 사이에 테이블을 공유할 필요는 없다. 왜냐하면, 공격자가 이전에 사용된 C_0에 대해 암호문 또는 인증 코드에 변조를 가할 경우, 복호화 시 인증 코드가 불일치하게 되어 공격 목적으로 반복 사용된 C_0 값에 대한 검증이 가능하게 된다.

[0190] 또한, 공격자가 공격에 이용할 수 있는 동일 C_0가 사용된 암호화 및 복호화 과정은 최대 4 번이다. 이는 각 모듈에 동일한 C_0 값이 최대 2번씩 적용될 수 있기 때문이다. 하지만, 동일 C_0가 사용된 4번의 부채널 정보를 이용하여 공격자가 공격하는 것이 현실적으로 어렵기 때문에, 이는 안전성 위협의 고려 대상이 아니다.

[0191] **치환 함수 기반에서 복호화해야 할 암호문의 크기가 큰 경우 (즉, 암호 모듈 내의 메모리 크기가 제한적인 경우)**

[0192] 암호문에 대한 변조가 없음이 파악되기 전까지는 암호모듈 밖으로 복호화된 정보, 즉 평문 값의 일부라도 암호모듈 밖으로 새나가서는 안된다. 그런데, 만약 복호화해야 할 암호문의 크기가 매우 큰 경우라면 제한된 메모리에 복호화된 정보를 무한정 저장해 놓고 있을 수가 없다. 여기서는 부가 정보를 고려한 암호화 및 인증 방식 4를 수정하여 적은 메모리로 암호문 변조 유무 파악 및 복호화 방식을 가능케 하는 방법을 소개한다. 이는 한 가지 예일 뿐, 비슷한 방식으로 다양한 암호화 및 인증 방식을 설계할 수 있다.

[0193] 도 61과 같이 동작하여, 암호화시 암호문$C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드T를 얻는다. 이제는 복호화하는 방식을 설명한다. 복호화시 암호문과 인증 코드 값으로부터 암호 모듈은 복호화를 진행한다. 진행하는 동안 도 61에서 부가 정보 처리 이후, 파란색으로 친 부분의 X, Y 값을 암호 모듈은 임시로 저장해 놓는다. 그리고, 최종 인증 코드 T 값이 올바른 경우, X, Y 값을 암호 모듈 밖으로 리턴하게 되면, X, Y 값을 가지고 시스템은 복호화를 진행할 수 있게 된다.

[0194] **치환 함수 기반에서 키 공유 관련 비용에 관하여**

[0195] 먼저 공유 키를 암호 통신을 필요로 하는 두 암호 모듈에 직접 저장하는 방법을 사용할 수도 있다. 이는 높은 안전성을 요구하는 국방 혹은 의료 분야 등에서 사용될 수 있는 방법이다. 아니면, 공개키 기반 구조에 기반하여 통신 하는 시점에 키 공유가 이루어지는 경우도 있다. 안전하지 않는 채널을 통한 키 공유는 일반적으로 공개키 기반 구조에 기반하여 설계될 수 있다. 공개키 암호는 대칭키 암호에 비해서 속도가 현저히 느리나, 키와 같이 매우 짧은 데이터를 한 번만 처리하면 되기에 키 공유에 따른 부채널 대응 기법 비용이 전체 암호 통신 효율성 및 전력 소모량 측면에서 별 영향을 주지 않는다.

[0196] 따라서 공개키 기반 구조를 사용하든 하지 않든, 실제 부채널 대응 비용은 비밀 키가 공유된 시점으로부터 대부분이 발생하기 때문에, 본 논문에서 제시된 방식이 크게 실제 암호 통신의 가용성 및 효용성 그리고 안전성에 크게 기여할 것으로 예상된다.

[0197] 본 문서에서는 치환 함수에 기반한 암호화 및 인증 코드 생성 방법을 제시하고, 현존하는 방식들과 달리 저비용, 고 효율, 고 안전성으로 부채널 공격 대응이 가능함을 보여주었다. 특히, 보호해야 할 XOR 연산의 수는 메시지 크기와 상관 없이 2 개로 고정된 것을 보게 된다. 이는 부채널 대응 기법을 오직 2 개의 XOR 연산만 해도 되기에 구현 시 매우 큰 장점을 지닌다. 블록 암호처럼 아직까지 치환 함수에 대한 표준은 존재하지 않으나, 본 연구 결과는 치환 함수의 표준화의 중요성을 보여주며, SHA-3와 같이 표준화될 알고리즘의 기반이 되는 치환 함수를 사용하여 본 문서에서 제시한 암호화 및 인증 코드 생성 방법을 구현할 수도 있다.

[0198] **본 발명에 따른 압축 함수 기반 신규 설계 논리 아이디어**

[0199] 현존하는 암호화 및 인증 코드 값 생성 방식들은 기반이 되는 알고리즘이 부채널 공격에 안전하게 구현되어야 한다는 근본적인 문제점을 지니고 있다. 이러한 근본적인 문제점을 안고 가는 이상, 부채널 공격에 대해 저비용, 고효율, 고 안전성을 갖는 구현 방법은 사실상 없는 상황이다. 따라서, 본 발명에서는 관점을 바꾸어, 운영 모드 차원에서 접근하고자 한다. 어떻게 하면, 기반 알고리즘에 대한 부채널 대응 기법 적용 없이 운영 모

드 차원에서의 대응만으로 안전성을 제공할 수 있는가에 대한 것이다. 본 발명에서는, 운영모드 차원에서, 부채널 공격에 대해 저비용, 고 효율성, 고 안전성을 제공하는 암호화 및 인증 코드 값 생성하는 설계 논리 및 구현 방식을 제안한다.

[0200] 부채널 공격의 핵심은 deterministic 또는 랜덤하지 않은 연산 과정을 수행하는 과정에서 비밀 정보를 부채널 정보로부터 축출하는 데에 있다. 따라서 매번 비밀 키가 랜덤하게 바뀌든지 또는 비밀 난스 (Nonce) 값이 매번 랜덤 하게 바뀌든지 해야 적은 연산으로 부채널 공격에 안전한 대응기법 마련이 가능하다. 보통 난스는 공개가 되는 값이나, 본 발명에서는 난스가 비밀 값의 일부인 경우로 고려한다. 만약, 키와 난스가 둘 다 고정되거나 추측이 가능하게 되면, 전체 연산과정의 보호 없이는 부채널 공격에 의해 키 추출이 가능하게 된다.

[0201] 압축 함수가 기반한 여러 대표적인 알고리즘으로는 SHA-1, SHA-2 등이 있다. 본 발명에서는, 통신 환경에 적합한, 임의의 치환 함수 알고리즘으로 구현해도 상관이 없는 암호화 및 인증 코드 값을 생성하도록 돕는 암호시스템 설계 방식을 제안한다.

[0202] 암호 통신에서 난스 값이 반복되어 사용될 경우, 전체 연산 과정이 deterministic이기 때문에, 부채널 공격에 의해 비밀 키가 노출될 위험이 있다. 따라서, 난스 값은 재 사용이 안되도록 하는 조치가 요구된다. 특히, 복호화시 난스 재사용을 막아야 한다. 또한, 난스가 노출될 경우, 키가 노출될 위험 또한 높게 되어 난스 자체도 암호화를 통해 보호를 해야 한다.

[0203] 난스 재 사용에 대한 위험은 복호화 과정에서 나타난다. 공격자가 암호문을 위조할 경우 인증 코드 값이 다르게 되어, 인증 코드가 틀리게 나온 난스를 블랙리스트에 올려서 다시는 사용하지 못하도록 하는 것이다. 만약, 블랙 리스트의 크기가 매우 크게 되면, 다시 새롭게 키를 공유하고, 블랙 리스트들을 삭제한다. 이러한 블랙리스트 기반 방식 외에 여기서는 다른 방식들을 또한 소개한다. 난스 열 생성 방식과 암호화 및 인증 방식과 같이 두 부분으로 나누어 설명하고자 한다.

[0204] **압축 함수 기반 난스 열 생성 방식**

[0205] 다행스럽게도 통신 환경에서의 암호화는 저장 데이터의 암호화가 아닌, 통신 자체의 암호화에 초점을 두고 있기 때문에, 난스 재사용을 막는 방법이 쉽게 구현될 수 있다. 두 암호 모듈 A와 B는 공유된 비밀 키 K로 부터두 개의 난스 열 (N_1, N'_1, N''_1), (N_2, N'_2, N''_2),...를 생성한다.

[0206] 난스 열을 다루는 방식에는 두 가지로 나누어 살펴볼 수 있다. 첫째는 난스 열을 암호 모듈 내에 저장해 놓는 방법이다. 난스 열을 암호 모듈 내에 저장해 놓는 방법을 설명하면, 암호문 복호화 시 얻어진 난스 값이 테이블에 존재할 경우 올바른 난스 값으로 받아들이고 복호화를 진행한다. 그리고 한번 사용된 난스는 삭제하여 난스 재사용을 막는다. 만약 올바른 난스 값이 아니면 복호화 중지하여 부채널 공격을 무력화시킨다.

[0207] 둘째는 난스 열을 실시간으로 생성하는 방법이다. 난스 열을 실시간으로 생성하는 방법을 설명하면, 암호문 복호화시 얻어진 난스 값이 사용되기로 예정된 난스 값이면 받아들이고 복호화를 진행하며, 아니면 복호화를 중지한다. 예를 들어, 다음의 도 62와 같이 공유된 비밀 키 K로 부터 두 개의 난스 열 (N_1, N'_1, N''_1), (N_2, N'_2, N''_2),...를 생성한다. 공유 키로부터 난스 열을 생성하였기에 두 암호 모듈 A와 B는 난스 열 또한 공유하게 된다.

[0208] 여기서 보인 예는 한가지 예에 불과하며, 본 발명은 키를 통해 난스 열을 생성하며, 키 K는 초기에만 사용되고 그 이후에는 사용되지 않으며, feedforward 연산 과정으로 인해 역 연산을 어렵게 할 수 있다.

[0209] 공유 키를 이용하여 새로운 메시지를 암호화할 경우, 난스 사용은 (N_1, N'_1, N''_1) 부터 순차적으로 사용하며, 만약 복호화 시 사용되어야 할 난스가 사용되지 않을 경우, 암호 모듈은 복호화를 수행하지 않는다. 실제 난스 열을 생성하는 것은 처음 비밀 키 K가 공유된 시점에서 난스 열을 생성하여 암호 모듈 내에 저장하는 방식을 취하거나, 아니면, 실시간으로 매번 이전 난스로부터 다음에 사용될 난스를 생성하여 사용할 수도 있다.

[0210] 위의 난스 열 생성 알고리즘을 보면, 공유 비밀 키 K는 처음에 한번만 사용되고, const, const', const''은 공개된 상수 값이며, 매번 비밀 난스 값이 업데이트되어 사용되기에, 어떤 상태에서도 동일한 비밀 정보가 반복적으로 사용되지 않아 부채널 공격 적용이 어렵게 설계되어 있음을 알 수 있다.

[0211] 특히, 난수 열을 생성 시, 단순한 연산 대신 복잡한 압축 함수 연산을 사용하는 이유는, 잘 설계가 된 압축 함

수를 사용할 경우, 각 난스 값 생성 시 좋은 난수 성질 또한 갖게 되기 때문이다. 좋은 난수성을 갖는 난스의 사용은 부채널 공격을 대응하는 데에 매우 필수적이다.

[0212] **네 가지 압축 함수 기반 암호화 및 인증 방식**

[0213] 위에서 설명한 방식으로 난스 열이 주어졌을 경우, 네 가지 암호화 및 인증 방식을 소개한다.

[0214] **압축 함수 기반 암호화 및 인증방식 1**

[0215] 임의의 길이의 메시지 M를 처리하기 위해서 패딩 방법 pad를 적용한 후, pad(M)=$M_1||M_2||...||M_t$ 와 같이 표기한다. 특히, 안전성을 위하여 패딩 방법 pad는 prefix-free이어야 한다. 이는 임의의 서로 다른 두 메시지 M, M'에 대해, pad(M)은 결코 pad(M')의 prefix가 되어서는 안 된다는 뜻이다. 도 63을 보면, IV1과 IV2는 상수 값을 뜻하며, 이전에 사용된 const, const', const'' 와 다른 상수 값이어야 한다. 그러면, 다음 그림과 같이 암호화 및 인증을 수행하여, 암호문C(=$C_0||C_1||||C_t$)와 -비트 값인 인증 코드T를 얻는다.

[0216] 압축 함수 기반 암호화 및 인증 방식 1을 설명하면, 암호문의 일부인 C_0는 난스 N의 암호문이라고 보면 된다. 난스 N을 보호하기 위하여 암호화된 C_0가 상대방 암호 모듈에 안전하지 않은 통신 채널을 통해 전달된다.

[0217] 암호문을 전달받는 암호 모듈은 암호문을 복호화하기 위하여, 먼저 주어진 암호문의 첫번째 블록인 C_0로부터 N을 생성한다. N이 앞서 설명한 난스 생성 알고리즘의 요구 조건에 부합할 경우, N'과 N''를 얻고, 복호화 과정을 수행하여 $M_1||...||M_t$를 얻는다. 최종적으로 인증 코드 값 T이 맞고 동시에 pad(M)= $M_1||...||M_t$ 인 M이 존재하면 M을 올바른 메시지로 받아들이고 M을 최종 출력하며, 틀리면 메시지에 변조가 있음을 알고, 오류 값을 출력한다.

[0218] 부채널 공격 대응 방법 관점에서 방식 1의 설계 논리를 좀 더 설명하면, 매번 암호화 마다 다른 난스를 사용하므로, 압축 함수의 입력 값이 매번 랜덤하게 변하게 되어, 공격자로 하여금 내부 상태 값을 얻기 어렵게 한다.

[0219] 도 64는 부채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 1에서 보호되어야 할 연산을 보여준다. 메시지 크기와 상관없이 초기의 XOR 연산을 최대 세 번만 보호만 해도 안전하게 된다.

[0220] **압축 함수 기반 암호화 및 인증 방식 2**

[0221] 첫 번째 방식과 차이점은 임의의 패딩 함수 pad를 사용해도 된다는 장점이 있다. 단, 마지막 블록 처리를 다르게 하기 위해 const1를 연산하는 과정을 추가하였다. 도 65와 같이 암호화 및 인증을 수행하여, 암호문 C(=$C_0||C_1||...||C_t$)와 τ-비트 값인 인증 코드 T를 얻는다.

[0222] 도 66은 부 채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 2에서 보호되어야 할 연산을 보여준다.

[0223] **압축 함수 기반 암호화 및 인증 방식 3**

[0224] 두 번째 방식처럼 임의의 패딩 함수 pad를 사용해도 되며, 단 마지막 블록 처리를 다르게 하기 위해 상수 값을 더하는 대신, 마지막 인증 코드 생성 값 처리를 다르게 하였다. 도 67과 같이 암호화 및 인증을 수행하여, 암호문C(=$C_0||C_1||...||C_t$)와 τ-비트 값인 인증 코드 T를 얻는다.

[0225] 도 68은 부 채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 3에서 보호되어야 할 연산을 보여준다.

[0226] **압축 함수 기반 암호화 및 인증 방식 4**

[0227] 임의의 패딩 함수 pad를 사용해도 되며, 마지막에 블록 처리시 난스 N', N''을 이용하였다. 도 69와 같이 암호

화 및 인증을 수행하여, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다.

[0228] 도 70은 부 채널 공격에 안전하도록 하기 위해서 암호화 및 인증 방식 4에서 보호되어야 할 연산을 보여준다.

[0229] **압축 함수 기반에서 안전성 강화 옵션**

[0230] 앞에서 보인 네 가지 방식에서 보호되어야 할 연산이 부채널 공격에 의해 안전하게 보호될 경우 안전하다고 말할 수 있다. 신규 부 채널 공격 기술의 발전으로 대응 기법의 취약성으로 임의의 상태의 난스 값이 노출 되면, 위의 방식들이 역연산이 가능하기에, 공유 키 K가 쉽게 노출될 염려가 있다. 결국 공유 키의 노출은 과거의 통신 정보나 미래의 통신 정보마저 노출될 위험을 안고 있다.

[0231] 이처럼 극단적인 상황에서조차도, 과거와 미래의 통신 정보를 보호하기 위하여, 본 장에서는 비밀 공유 키 K로부터 암호화 키 열 생성 알고리즘, 난스 열 생성 알고리즘을 제시한다. 실시간 통신의 효율성과 직결되어, 통신의 효율성을 떨어뜨리기에 이 장에서는 생략하나, 이 역시 가능함을 명시해 둔다.

[0232] **압축 함수 기반에서 안전성 강화를 위한 암호화 키 열 생성 방식**

[0233] 현재에 사용된 암호화 키가 노출된다고 할지라도, 과거와 이전에 사용된 키를 보호하기 위하여 도 71과 같이 매번 암호화에 대해 다르게 사용될 일회용 암호화 키 열$K_1, K_2,$ 생성한다. 각 일회용 암호화 키는 암호화 시 난스와 동일하게 한번의 암호화 당 한 번씩만 사용된다. 다음 번의 암호화 시 암호화 키 열에서의 다음 번의 일회용 암호화 키를 이용한다.

[0234] 앞에서 설명한 방식과의 차이는, 공격자가 임의 상태의 암호화 키 K_i 또는 난스 값 N_i를 알았다고 할지라도, 다른 암호화 키 값들 또는 난스 값들을 보호할 수 있다는 장점을 가진다. K1을 공격자가 얻었다고 할지라도, K2를 얻기 위해서는 키 열 생성 알고리즘의 내부 전체 블록 상태를 알아야 한다. 이를 위하여 공격자는 나머지 c 비트 정보에 대한 전수 조사를 해야 한다. 또한, 추측한 키가 맞는지에 대해 통신을 통해 검증해야 하는 부가적인 오버헤드가 요구되기에 과거와 미래에 사용될 암호화 키를 보호할 수 있다는 장점을 가진다.

[0235] 무엇보다, 공유 키 K가 주기적으로 업데이트가 될 경우, 어떤 시점에 사용된 공유 비밀 키에 대응되는 암호화 키 열을 설사 모두 알았다고 할지라도, 과거 혹은 새로 갱신된 공유된 비밀 키 정보를 얻는 데에는 전혀 도움이 되지 않는다.

[0236] **압축 함수 기반에서 헤드 정보 등 부가 정보의 인증까지 처리하고자 할 경우 옵션**

[0237] 앞에서는 난스와 메시지에 대한 암호문, 그리고 이에 대한 인증 코드 값만이 존재했는데, 때론 헤드 정보 등 공개된 정보와 함께 인증을 처리할 경우가 있다. 헤드 등과 같이 부가적인 정보를 처리하는 암호화 및 인증 방식을 AEAD (Authenticated Encryption with Associate Data) 라고 부른다. 본 절에서는 부가 정보를 인증할 수 있도록 앞서 제시된 방식들을 어떻게 수정할 것인가에 대해 설명하고자 한다.

[0238] 부가정보를 A라고 할 때, 먼저 A에 패딩을 하여 패딩 후 A의 크기가 블록 크기의 배수가 되도록 한다. 구체적으로, 패딩 방법을 pad라고 할 때, $pad(A)=A_1||...||A_j$ 처럼 패딩 후의 크기가 j 블록으로 표현된다고 하자. 예를 들어, 패딩 방법으로 10* 패딩 방법을 쓸 수 있다.

[0239] 그러면 먼저 네 가지 방식에 대해 도 72부터 75에서 보여준 바와 같이 변형시켜 암호문과 인증 코드를 생성한다. 즉, 부가 정보 A는 암호문의 크기에는 변화를 주지 않고, 단지 인증 코드 생성시에만 영향을 준다. 즉, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다.

[0240] **압축 함수 기반에서 r값이 작은 경우의 구현 방법에 대하여**

[0241] 블록 사이즈 r 값이 작게 되면, 비밀 키 값, 난스 값, 인증 코드 값의 비트 길이가 여러 블록이 될 수도 있다. 하지만, 보통은 키 값, 난스 값, 인증 코드 값들은 일반적으로 고정된 길이의 비트 열이 되기 때문에, 본 명세서에서는 이들의 비트 길이가 고정된 경우라 생각하고 앞서 설명한 방식들을 어떻게 변경할 것인가에 대한 아이

디어를 설명하고자 한다.

[0242] **압축 함수 기반에서 r 값이 작은 경우에 대한 암호화 키 열 생성 방식**

[0243] 공유 키 K로부터 암호화 키 열을 생성하고자 할 경우 다음과 같이 암호화 키를 생성할 수 있다. 키 값은 아무리 커도 r+c 비트 값보다는 작다고 가정할 때, keypad(K)=$K^*_1||K^*_2$ 라고 하자. 여기서 keypad는 패딩 방법으로 10* 패딩이 될 수도 있고, 단순히 0* 패딩이 될 수도 있다. 이때 K_1은 r 비트, K_2는 c 비트 값이 된다. 이때 암호화 키 열은 도 76과 같은 방식으로 생성할 수 한다.

[0244] **압축 함수 기반에서 r 값이 작은 경우에 대한 암호화 및 인증 방식**

[0245] 앞서 r 값이 작은 경우에 대해 난스 열, 암호화 키 열 생성 방법을 설명하였다. 이를 기초로, 암호화 및 인증 방식을 설명하고자 한다. 앞 서 네 가지 방식을 설명했는데, 여기서 부가 정보가 추가된 경우의 세 가지 방식을 어떻게 변경할 것인가를 살펴본다. 여기서 소개하는 방식으로, 부가 정보 인증이 필요 없는 경우에도 동일하게 적용할 수 있음을 말해 둔다.

[0246] 난스 열 중 현재 상태의 난스 값을 N, N', N''으로 표현하였고, 암호화 키 열 중 현재 상태의 암호화 키 값을 K, K' 으로 표현하였다. 각 방식들에 대해, 복호화 시 C_0로부터 N의 값이 올바른 값인지 난스 열을 통해서 검증한 후, N이 올바른 값이면 복호화를 진행한다.

[0247] 여기서 r의 값이 1 비트 정보와 같이 작게 되면, 임의의 변조된 C_0 값에 대해서 성립할 확률은 1/2이 된다. 하지만, 이 경우 설사 변조에 성공하더라도, 나머지 N' 값의 비트 크기가 크기 때문에 내부 상태 값들을 랜덤하게 만들게 된다. 다만, r의 값이 1이 되면, 바로 난스의 변조 유무 또는 반복 유무를 파악하는 시간이 늦어질 뿐, 결국 인증 코드로 변조 유무 또는 반복 사용 여부를 파악할 수 있게 된다. 따라서, 난스의 변조 유무 또는 반복 유무를 일찍 파악하기 위하여, r이 최소 10 비트 이상이 될 것을 권장한다.

[0248] 그러면 네 가지 방식을 도 77에서 도 80에서 보인 바와 같이 변형시켜 암호문과 인증 코드를 생성한다. 즉, 부가 정보 A는 암호문의 크기에는 변화를 주지 않고, 단지 인증 코드 생성시에만 영향을 준다. 즉, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다.

[0249] **압축 함수 기반에서 암호문 없이 인증 코드만 생성하고자 하는 경우**

[0250] 상황에 따라서 암호화 없이 인증 코드만 요구되는 경우 부채널 공격에 안전하도록 인증 코드 값인 인증 코드 값을 생성하고자 경우, 앞서 제시된 모든 방식들을 인증 코드만 생성하는 MAC 알고리즘으로 변경이 가능하다. 구체적으로, 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T 중 $C_1||...||C_t$을 생략한 채, $C_0||T$ 값 전체를 인증 코드로 정의를 내린다.

[0251] **압축 함수 기반에서 인증 코드 없이 암호문만 생성하고자 하는 경우**

[0252] 상황에 따라서 인증 코드에 대한 필요 없이 암호화만 요구되는 경우 부채널 공격에 안전하도록 암호화를 수행하고자 하는 경우, 앞서 제시된 방식들의 출력 값 중 τ-비트 값인 인증 코드 T을 제외한 나머지 암호문 $C(=C_0||C_1||...||C_t)$만을 출력함으로 암호문을 생성할 수 있다.

[0253] **압축 함수 기반에서 내부 상태 값이 공격자에게 노출되었을 때 암호화 키 또는 난스 값을 보호하는 기법에 관하여**

[0254] 지금까지 제시한 방식들은 압축 함수가 역연산이 쉽게 가능할 경우에 대해 내부 상태 값이 공격자에게 노출될 경우, 역 연산으로 말미암아 암호화 키 또는 난스 값이 노출될 위험이 노출한다. 이러한 상황은 구현 상의 실수에 의해서 발생할 수도 있고, 아니면 암호화 모듈이 아닌 일반 CPU에서 암호화를 수행하는 경우 발생할 수도 있

다. 따라서, 이러한 극단적인 상황 속에서도 사용된 난스 또는 암호화 키 값을 보호하기 위하여 암호화하는 동안 압축 함수 과정 시 내부적으로 역연산이 어렵도록 하도록 압축 함수를 설계하거나 또는 압축 함수 외부에 feedforward 연산 과정을 삽입함으로써 역연산이 어렵도록 할 수 있게 된다. 이때 역연산의 어려움은 바로 난스 값 혹은 암호화 키를 공격자가 얻는 것이 어렵게 하는 데 효과적인 대응 방법으로 사용될 수 있다.

[0255] **압축 함수 기반에서 난스 값만을 이용한 암호화 및 인증 방법**

[0256] 지금까지는 암호화 키 (열)와 난스 열을 기반한 암호화 및 인증 방법을 소개하였다. 여기서는 암호화 키 없이 난스 열만을 이용한 암호화 방식을 소개한다. 여기서 소개한 방식은 부가 정보가 적용된 경우에 대해서도 동일하게 적용된다. 또한, 여기서 소개하는 방식들은 부채널 공격 대응 기법 적용이 전혀 필요가 없는 방식들이다. 특히, 난스 열 생성 방식에 좀 더 개선할 여지가 있는데 도 81를 먼저 설명하면, 한번의 압축 함수 연산으로 (N, N') 각각을 을 생성한다.

[0257] **압축 함수 기반에서 난스 기반 암호화 및 인증 방식 1**

[0258] 앞서 소개한 방식 1을 난스 기반으로 변환하는 방법을 제시한다. 도 81에서 생성된 난스를 이용하여 도 82에서와 같이 난스 열 중 해당 (N,N') 을 이용하여 복호화 하고자 하는 경우, C_0로부터 동일한 N의 값이 복호화가 되면, 올바른 난스로 인식하고, 계속해서 복호화를 진행하고, 최종 인증 코드 T 값이 올바르고, 메시지 패딩이 올바르게 되어 있으면, 해당 메시지를 출력하고, 만약 오류가 있으면 메시지 출력 없이 오류가 있다고만 출력한다.

[0259] **압축 함수 기반에서 난스 기반 암호화 및 인증 방식 2**

[0260] 앞서 소개한 방식 2를 난스 기반으로 변환하는 방법을 제시한다. 도 81에서 생성된 난스를 이용하여 도 83에서와 같이 난스 열 중 해당 (N,N') 을 이용하여 복호화 하고자 하는 경우, C_0로부터 동일한 N의 값이 복호화가 되면, 올바른 난스로 인식하고, 계속해서 복호화를 진행하고, 최종 인증 코드 T 값이 올바르고, 메시지 패딩이 올바르게 되어 있으면, 해당 메시지를 출력하고, 만약 오류가 있으면 메시지 출력 없이 오류가 있다고만 출력한다.

[0261]

[0262] **압축 함수 기반에서 난스 기반 암호화 및 인증 방식 3**

[0263] 앞서 소개한 방식 3를 난스 기반으로 변환하는 방법을 제시한다. 도 81에서 생성된 난스를 이용하여 도 84에서와 같이 난스 열 중 해당 (N,N') 을 이용하여 복호화 하고자 하는 경우, C_0로부터 동일한 N의 값이 복호화가 되면, 올바른 난스로 인식하고, 계속해서 복호화를 진행하고, 최종 인증 코드 T 값이 올바르고, 메시지 패딩이 올바르게 되어 있으면, 해당 메시지를 출력하고, 만약 오류가 있으면 메시지 출력 없이 오류가 있다고만 출력한다.

[0264] **압축 함수 기반에서 난스 기반 암호화 및 인증 방식 4**

[0265] 앞서 소개한 방식 4를 난스 기반으로 변환하는 방법을 제시한다. 도 81에서 생성된 난스를 이용하여 도 85에서와 같이 난스 열 중 해당 (N,N') 을 이용하여 복호화 하고자 하는 경우, C_0로부터 동일한 N의 값이 복호화가 되면, 올바른 난스로 인식하고, 계속해서 복호화를 진행하고, 최종 인증 코드 T 값이 올바르고, 메시지 패딩이 올바르게 되어 있으면, 해당 메시지를 출력하고, 만약 오류가 있으면 메시지 출력 없이 오류가 있다고만 출력한다.

[0266] **압축 함수 기반에서 난스에 대한 동기화 과정 없는 암호화 및 인증 방식**

[0267] 지금까지 소개된 방식들은 두 암호 모듈 사이에 복호화가 제대로 이루어지기 위해서는 난스가 동기화 되어야 한

다는 요구 조건을 필요로 하였다. 만약, 난스에 대한 동기화가 어려운 경우, 초기 세 개의 XOR 연산에 대한 부채널 대응 기법 적용이 요구되며, 그리고, 난스 N에 대한 암호화 값인 C_0로부터 난스 N을 계산 한 후, N으로부터 N', N''을 생성할 수 있는 방식을 따로 개발해야 한다. 예를 들어, 도 86과 같은 방식으로 N으로부터 N', N''을 생성한다. 단, N'과 N''의 난수성은 N의 난수성에 의존하기에, N의 비트 크기가 커야 한다는 요구 사항이 추가된다.

[0268] 복호화 시 인증 코드가 맞지 않은 경우에 각 암호 모듈은 사용된 난스 값을 자신의 모듈 내의 테이블에 저장한다. 이는 사용을 금지해야 할 난스 값들을 저장한 테이블이다. 두 암호 모듈 사이에 테이블을 공유할 필요는 없다. 왜냐하면, 공격자가 이전에 사용된 C_0에 대해 암호문 또는 인증 코드에 변조를 가할 경우, 복호화 시 인증 코드가 불일치하게 되어 공격 목적으로 반복 사용된 C_0 값에 대한 검증이 가능하게 된다. 또한 공격자가 공격에 이용할 수 있는 동일 C_0가 사용된 암호화 및 복호화 과정은 최대 4 번이다. 이는 각 모듈에 동일한 C_0 값이 최대 2번씩 적용될 수 있기 때문이다. 하지만, 동일 C_0가 사용된 4번의 부채널 정보를 이용하여 공격자가 공격하는 것이 현실적으로 어렵기 때문에, 이는 안전성 위협의 고려 대상이 아니다.

[0269] **압축 함수 기반에서 복호화해야 할 암호문의 크기가 큰 경우 (즉, 암호 모듈 내의 메모리 크기가 제한적인 경우)**

[0270] 암호문에 대한 변조가 없음이 파악되기 전까지는 암호모듈 밖으로 복호화된 정보, 즉 평문 값의 일부라도 암호 모듈 밖으로 새나가서는 안된다. 그런데, 만약 복호화해야 할 암호문의 크기가 매우 큰 경우라면 제한된 메모리에 복호화된 정보를 무한정 저장해 놓고 있을 수가 없다. 여기서는 부가 정보를 고려한 암호화 및 인증 방식 4를 수정하여 적은 메모리로 암호문 변조 유무 파악 및 복호화 방식을 가능케 하는 방법을 소개한다. 이는 한 가지 예일 뿐, 비슷한 방식으로 다양한 암호화 및 인증 방식을 설계할 수 있다.

[0271] 도 87과 같이 동작하여, 암호화시 암호문 $C(=C_0||C_1||...||C_t)$와 τ-비트 값인 인증 코드 T를 얻는다. 이제는 복호화하는 방식을 설명한다. 복호화시 암호문과 인증 코드 값으로부터 암호 모듈은 복호화를 진행한다. 진행하는 동안 도 87에서 부가 정보 처리 이후, 파란색으로 친 부분의 X, Y 값을 암호 모듈은 임시로 저장해 놓는다. 그리고, 최종 인증 코드 T 값이 올바른 경우, X, Y 값을 암호 모듈 밖으로 리턴하게 되면, X, Y 값을 가지고 시스템은 복호화를 진행할 수 있게 된다.

[0272]

[0273] **압축 함수 기반에서 키 공유 관련 비용에 관하여**

[0274] 먼저 공유 키를 암호 통신을 필요로 하는 두 암호 모듈에 직접 저장하는 방법을 사용할 수도 있다. 이는 높은 안전성을 요구하는 국방 혹은 의료 분야 등에서 사용될 수 있는 방법이다. 아니면, 공개키 기반 구조에 기반하여 통신하는 시점에 키 공유가 이루어지는 경우도 있다. 안전하지 않은 채널을 통한 키 공유는 일반적으로 공개키 기반 구조에 기반하여 설계될 수 있다. 공개키 암호는 대칭키 암호에 비해서 속도가 현저히 느리나, 키와 같이 매우 짧은 데이터를 한번만 처리하면 된다. 키 공유에 따른 부채널 대응 기법 비용이 전체 암호 통신 효율성 및 전력 소모량 측면에서 별 영향을 주지 않는다.

[0275] 따라서 공개키 기반 구조를 사용하든 하지 않든, 실제 부채널 대응 비용은 비밀 키가 공유된 시점으로부터 대부분이 발생하기 때문에, 본 논문에서 제시된 방식이 크게 실제 암호 통신의 가용성 및 효용성 그리고 안전성에 크게 기여할 것으로 예상된다.

[0276] 본 발명에서는 압축 함수에 기반한 암호화 및 인증 코드 생성 방법을 제시하고, 현존하는 방식들과 달리 저비용, 고 효율, 고 안전성으로 부채널 공격 대응이 가능함을 보여주었다. 특히, 보호해야 할 XOR 연산의 수는 메시지 크기와 상관 없이 3 개로 고정된 것을 보게 된다. 이는 부채널 대응 기법을 오직 세 개의 XOR 연산만 해도 되기에 구현 시 매우 큰 장점을 지닌다. 블록 암호처럼 아직까지 압축 함수에 대한 표준은 존재하지 않으나, 본 연구 결과는 압축 함수의 표준화의 중요성을 보여주며, 설사 표준화가 아직 진행 중에 없더라도, SHA-3와 같이 표준화될 알고리즘의 기반이 되는 압축 함수를 사용하여 본 문서에서 제시한 암호화 및 인증 코드 생성 방법을 구현할 수도 있다.

[0277] **저 메모리 환경에서 암호 인증 방법에 관하여**

[0278] 도 88은 본 발명의 실시 예에 따른 암호 모듈을 포함한 장치에 암호인증 방법을 설명하기 위한 블록도이다. 도 88를 참조하면, 장치(10)는 암호 모듈(11)를 포함한다. 장치(10)에는 암호를 필요로하는 모든 장치, 예를 들면, 컴퓨터, 태블릿 PC, 스마트 폰, 모바일 폰, 스마트 카드, 메모리 카드 등이 포함될 수 있다.

[0279] 암호 모듈(11)은 그 내부에 비밀 키 값과 암호 알고리즘이 있어서 외부의 공격자로부터 안전하게 모듈 내에서 암호화 연산을 수행한다. 암호 모듈(11)은 부채널 공격, 오류 주입 공격 등에 다양한 공격에 안전하도록 설계되어야 한다. 암호 모듈(11)은 면적이 크면 클수록 설계 및 구현 비용이 증가하므로 암호 모듈 안에는 사용 가능한 메모리 크기에 제약이 따른다. 암호 모듈(11)의 구성 및 동작 원리는 도 89에서 설명될 것이다.

[0280] 도 89는 도 88에 도시된 암호 모듈을 예시적으로 보여주는 블록도이다. 도 89를 이용하여 암호 모듈(11) 내에서 어떻게 암호인증을 수행할 것인가에 대한 기법을 설명한다.

[0281] 본 발명을 통한 암호인증 기법은 함수 1, 2, 3을 이용하여 입력값으로 Key, Nonce, Associate Data, Plaintext를 입력으로 받아 Ciphertext와 Tag 값을 출력한다. 함수 1은 Key, Nonce, Associate Data를 입력으로 하여, Output 1를 출력한다. 함수 2는 Output 1과 Plaintext를 입력으로 하여 Ciphertext와 Output 2를 출력한다. 함수 3은 Key, Nonce, Associate data를 입력으로 하여 Tag 값을 최종 출력한다.

[0282] 도 89를 참조하면 Key는 비밀 키값, Nonce는 랜덤하게 생성된 공개된 값, Associate data는 공개된 값이며, 함수 1, 2, 3을 이용하여 Ciphertext (암호문)와 Tag (태그)를 생성한다. 이제 암호문과 태그에 대한 변조 유무를 검증하고 최종 평문(plaintext)를 출력하기 위하여 다음과 같은 과정을 거친다.

[0283] **저 메모리 환경에서 변조 검증 단계 및 최종 평문 출력 과정**

[0284] **단계 1.** 암호 모듈(11)은 장치(10)로부터 Nonce, Associate data, Ciphertext, Tag 값을 입력받는다. 여기에서, 이 값들은 암호 모암호 모듈(11)에 한번에 전달되지 않고, 암호 모듈 안의 작은 메모리에 한 블록씩 전달되어 Tag 값을 계산한다.

[0285] **단계 2.** 암호 모듈에서는 주어진 키 Key를 이용하여 Output 1을 계산하여 저장한다. 그리고 최종 Tag 값이 맞는지 검증한다. Tag 값이 맞지 않으면 주어진 Ciphertext는 변조된 것을 처리하고 그 어떤 평문의 일부도 출력하지 않는다.

[0286] **단계 3.** Tag 값이 맞는 경우 암호 모듈은 Output 1를 암호 모듈 밖으로 출력한다.

[0287] **단계 4.** 암호 모듈 밖에서 Output 1을 이용하여 암호문으로부터 최종 평문을 복호화한다.

[0288] 도면 88에서, 주어진 방식으로 설계된 암호인증 기법의 안전성을 위하여 요구되는 안전성을 설명하면 다음과 같다.

[0289] 도면 89에서 보는 바와 같이, 암호문과 태그가 올바를 경우 output 1이 최종 암호 모듈 밖으로 나가기 때문에 output 1과 Nonce, Associate data로부터 비밀키 key가 보호되어야 한다. 이는 key 값을 계산하는 것이 어렵도록 함수 1에 일방향 성질이 요구된다.

[0290] 도면 90에서 보는 것처럼, 함수 2에는 비밀키 key 값이 없이 수행이 된다. 그리고 output 1이 암호 모듈 밖으로 나가기 때문에, 공격자가 output 2 값이 같도록 하는 서로 다른(Output 1, Plaintext)과 (Output 1', Plaintext')를 찾을 수 있을 경우 (Output 1, Plaintext)에 해당하는 태그 값을 이용하여(Output 1', Plaintext')에 대한 위조가 가능하게 된다.

[0291] 따라서, 함수 2는 충돌저항성 성질을 지녀야 한다. 한편, 암호화 과정과 복호화 과정을 통해 공격자는 Output 2와 Tag, Nonce, Associate data, Plaintext, Ciphertext를 얻게 된다.

[0292] 도 91에서 보는 것처럼, 공격자로부터 비밀키 Key 정보를 보호하기 위하여, 함수 3에 대해 Ouput 2, Tag, Nonce, Associate data가 주어졌을 경우 Key를 찾는 것이 어려워야 한다. 이는 함수 3에 일방향 성질이 요구됨을 뜻한다.

[0293] 도 92에서와 같이, Tag 값이 동일하도록 하는 서로 다른 (Key, Nonce, Associate data, Plaintext)와 (Key', Nonce', Associate data', Plaintext')를 얻는 것이 어려워야 한다. 여기서 key는 비밀 값이 아니며, 공격자가

임의의 key, key' 값을 찾기 어렵도록 하는 데에 있다. 이를 통해 함수 3을 이용한 위조 공격를 차단한다.

[0294] 도 93은 함수 2에 대해, Output 2를 알지 않은 상태에서도 Output 1과 Ciphertext만 알아도 Plaintext를 계산할 수 있어야 함을 보여주는 도면이다. 이는 최종 암호문과 태그가 올바를 경우, output 1의 정보만 가지고 비밀 키에 대한 정보 없이도 암호 모듈 밖에서 암호문에 대한 평문을 복구할 수 있어야 하기 때문이다.

[0295] 이제부터 본 발명에 따른 실제적인 암호인증 설계 논리 예를 제시한다.

[0296] **본 발명에 따른 암호 인증 기법 설계 논리 예**

[0297] 본 발명을 기초로 어떻게 메모리가 작은 암호 모듈과 같은 경우에 대해 암호문 변조를 어떻게 검증할 것인가를 보여준다. 도 94에서 $P_0...P_v$=pad(Key, Nonce, Associate data) (여기서 pad는 임의의 패딩기법), $M_0...M_w$는 평문, $C_0...C_w$는 암호문, $z_0z_1...z_t$은 태그를 나타낸다.

[0298] 도 94에서 주어진 기법은 본 발명을 기초로 도 95에서 보여준 것과 같이 세가지 부분으로 나누어 볼 수 있다. 그리고 암호문과 태그가 올바를 경우 본 발명에 의해 암호 모듈은 $(a_0||b_0)$를 암호 모듈 밖으로 내보내면, $(a_0||b_0)$과 주어진 암호문으로부터 원래의 평문을 복구할 수 있게 되므로, 메모리가 작은 환경에서 구현이 가능하게 된다.

[0299] 동시에 도 94에서 제시된 예는 도 89에서 93까지에서 요구되는 조건을 만족시키는 데 그 이유는 Feedforward 연산 과정으로 말미암아 비밀키 복구가 어렵도록 하는 일방향 성질을 제공하며 도 94에서 보는 것처럼 r+c 비트 크기의 내부 상태 값의 중 마지막 c 비트 값을 외부의 값에 의해 조정이 안되기에 충돌 쌍을 찾기 어렵게 되어 충돌 저항성을 제공한다.

[0300] **저 메모리 환경에서 Nonce가 반복되는 것을 막기 위한 조치**

[0301] 실제 암호인증 기법을 구현할 경우, Nonce가 매번 바뀌는 것이 아니라, 반복되는 경우가 발생한다. 이때 Nonce가 재사용되는 것을 막기 위하여 도 96에서 보인 바와 같이 Nonce를 직접 생성하지 않고 함수 4를 이용하여 Key, Associate data, Plaintext로부터 생성한다. 이때 함수 4에 요구되는 성질은 도 97에 보인 바와 같이 함수 4의 출력값 (Nonce로 사용)과 Associate Data, Plaintext로부터 비밀 키 Key를 생성하는 것이 어려워야 하며, 동시에 도 98에서 보인 바와 같이 동일한 출력 값을 갖는 서로 다른 (Key, Associate data, Plaintext)와 (Key', Associate data', Plaintext')를 찾는 것이 어려워야 한다.

[0302] 본 발명에서는 메모리가 작은 환경에서 암호인증 기법을 어떻게 설계할 것인가에 대해 아이디어를 제시하고, 이를 구체화하기 위해서 도 94에서와 같이 실제 예를 보여주었다. 새로운 공격 기법의 출현과 동시에 스마트 디바이스와 같이 새로운 기술의 발달로 말미암아 암호 모듈에 기반한 안전한 암호화 및 인증의 필요성이 점점 중요시 되고 있다. 본 발명에서는 어떻게 암호 모듈과 같이 작은 메모리를 갖는 환경에서 안전하게 암호인증 기법을 구현할 것인가에 대해 설계 논리 및 실제 예를 제시함으로 본 발명의 가용성 및 효용성을 보여 주었다.

[0303] 한편, 상술 된 본 발명의 내용 및 예들은 발명을 실시하기 위한 구체적인 실시 예들에 불과하다. 본 발명은 구체적이고 실제로 이용할 수 있는 수단 자체뿐 아니라, 장차 기술로 활용할 수 있는 추상적이고 개념적인 아이디어인 기술적 사상을 포함할 것이다.

부호의 설명

[0304] 1: 기반 암호함수

2: 단순 외부 연산

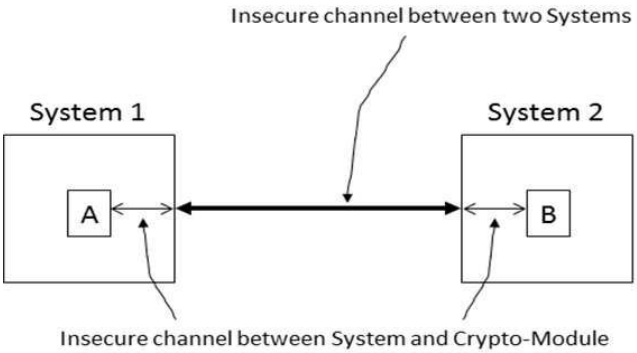
10: 장치(device)

11: 암호 모듈

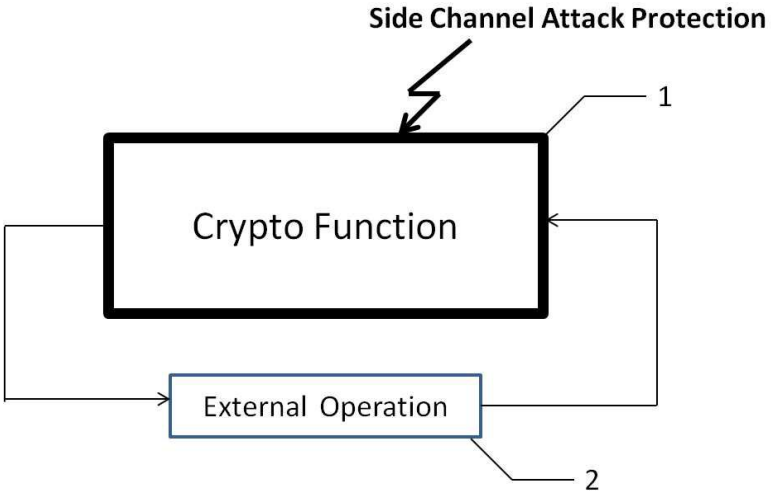
100, 200, 300, 400: 암호 함수

도면

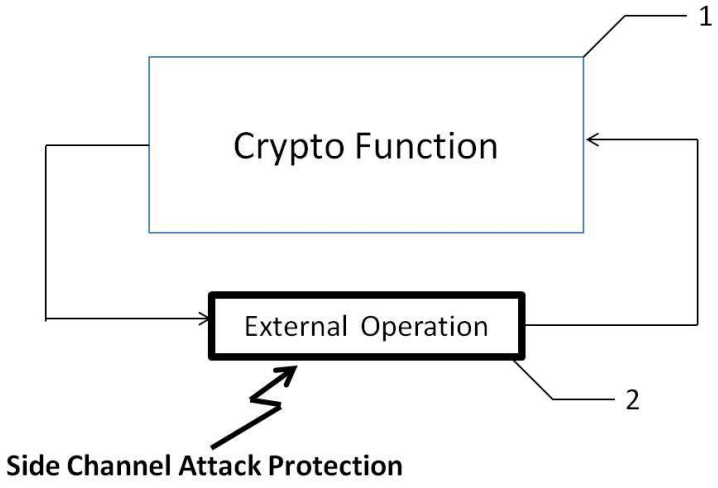
도면1



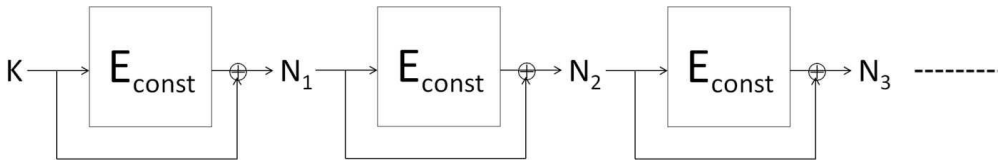
도면2



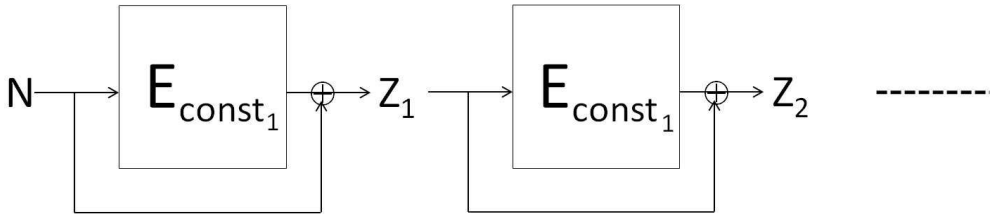
도면3



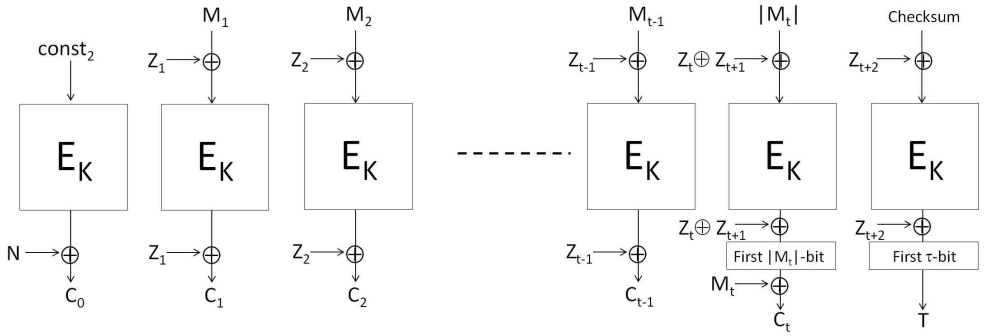
도면4



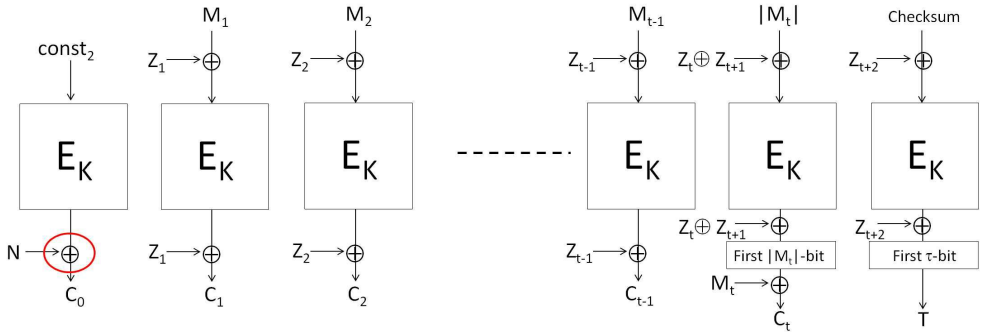
도면5



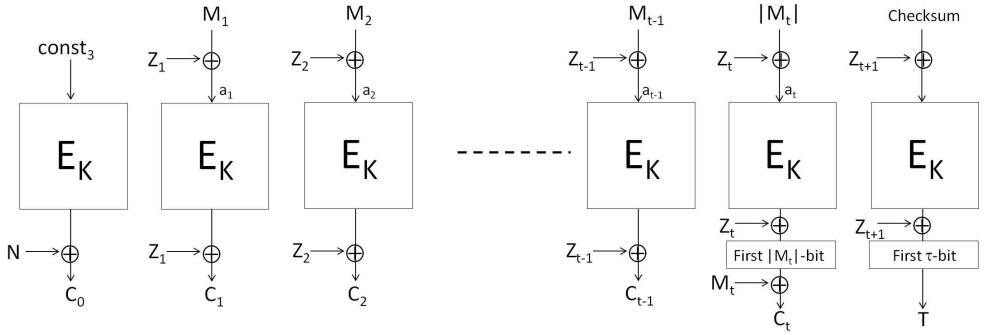
도면6



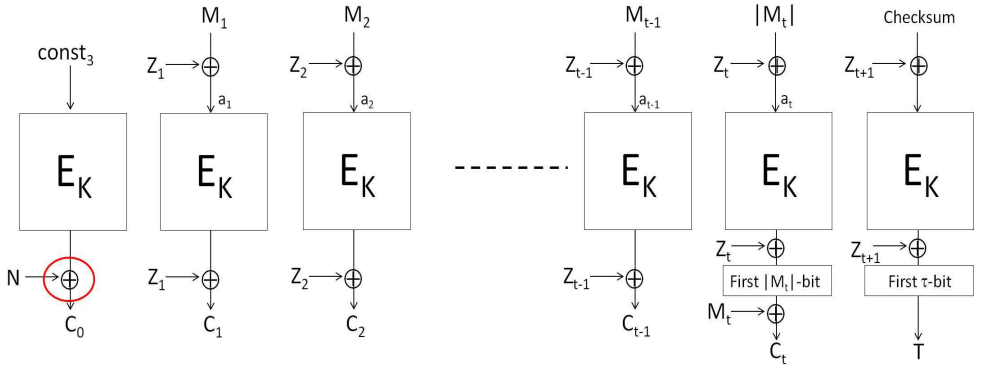
도면7



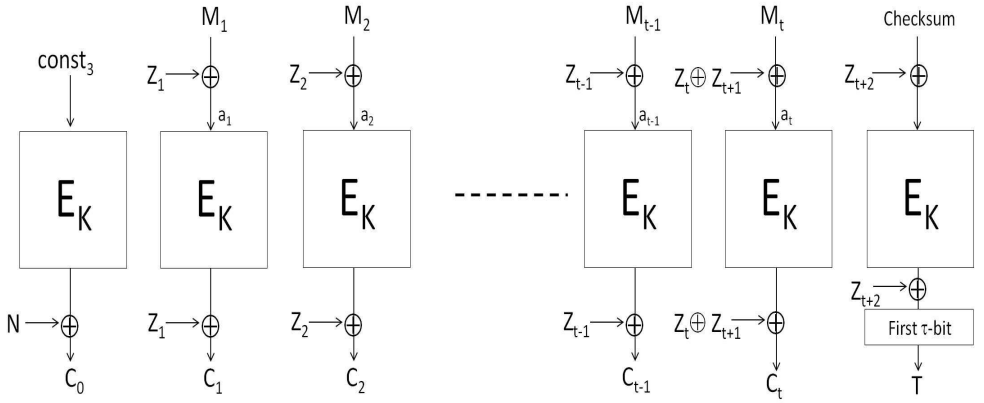
도면8


const3
M1
M2
Mt-1
|Mt|
Checksum
EK
N
C0
C1
C2
Ct-1
Ct
T
First |Mt|-bit
First τ-bit

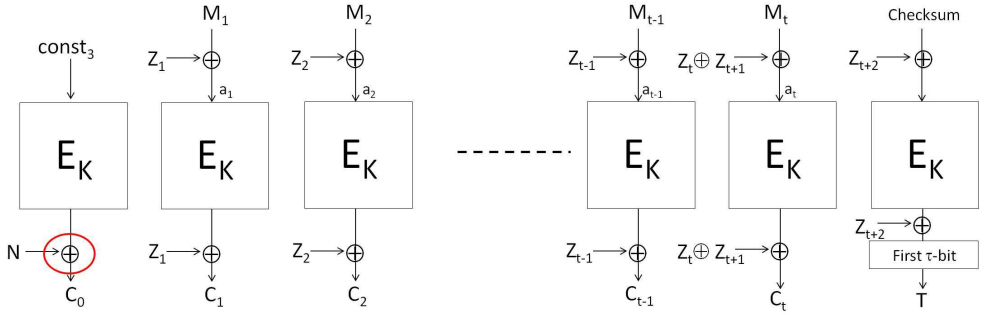
도면9


const3
M1
M2
Mt-1
|Mt|
Checksum
EK
N
C0
C1
C2
Ct-1
Ct
T
First |Mt|-bit
First τ-bit

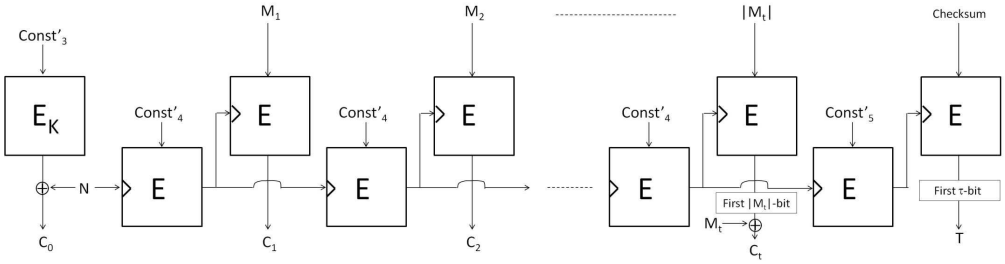
도면10


const3
M1
M2
Mt-1
Mt
Checksum
EK
N
C0
C1
C2
Ct-1
Ct
T
First τ-bit

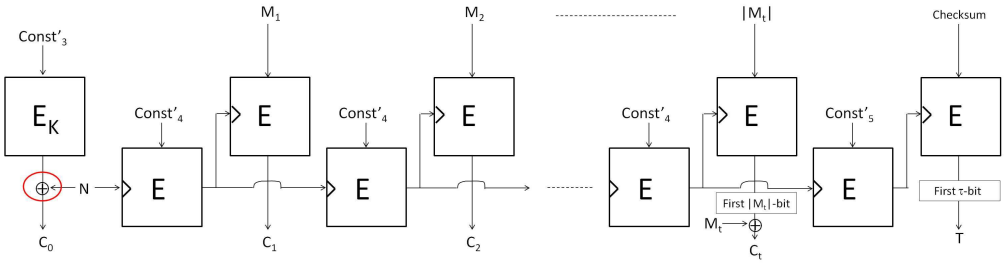
도면11



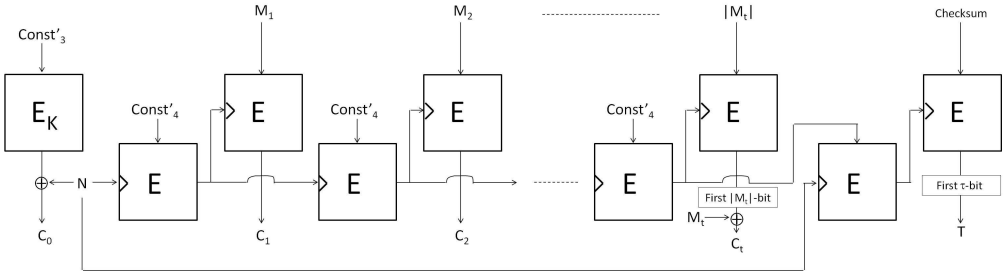
도면12



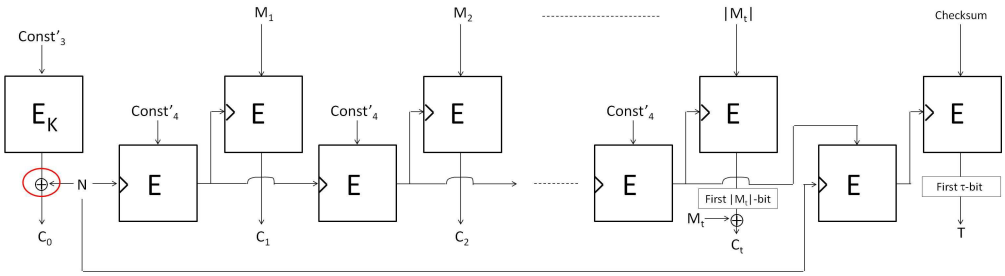
도면13



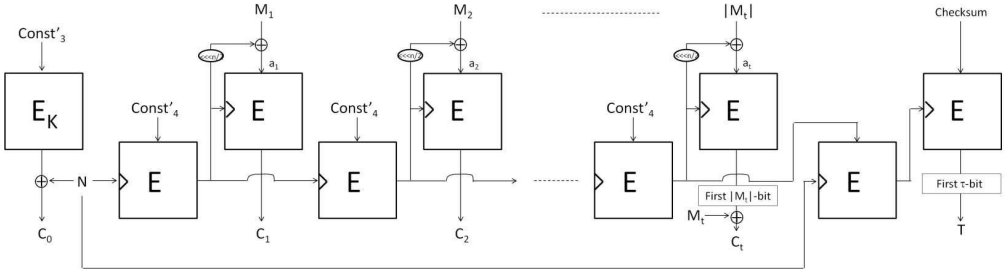
도면14



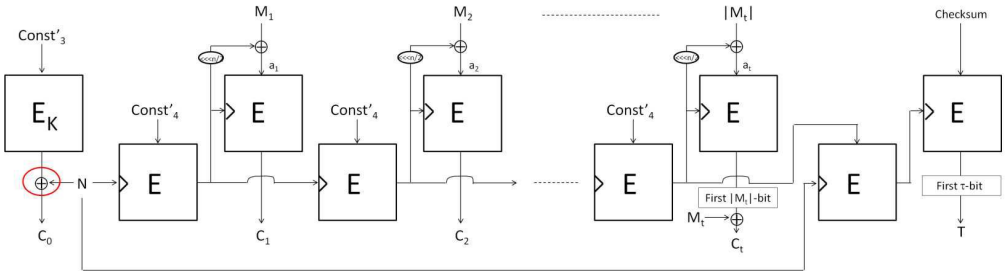
도면15



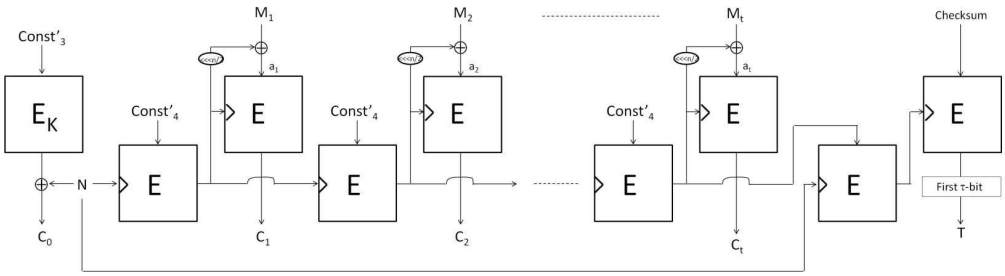
도면16



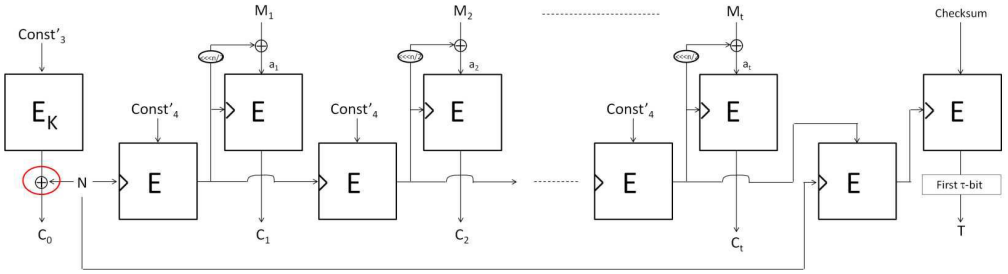
도면17



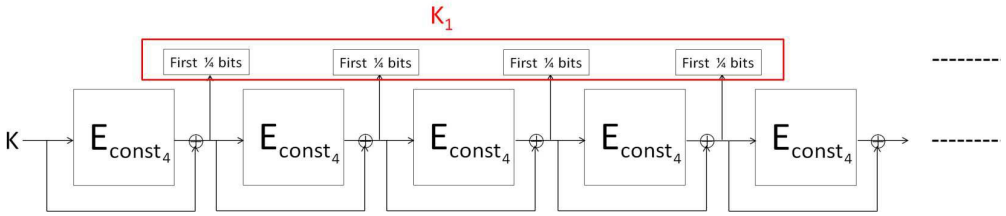
도면18



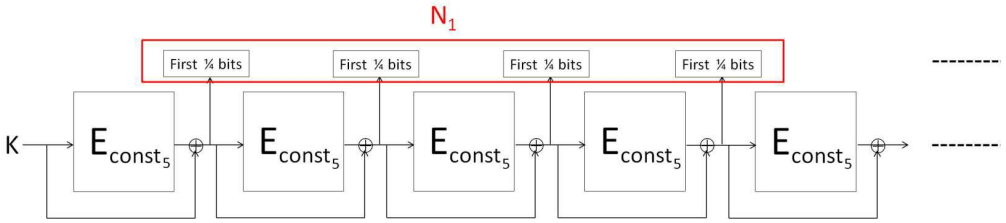
도면19



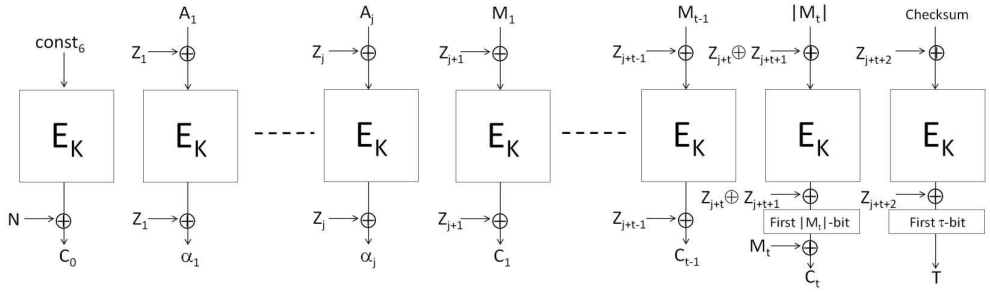
도면20



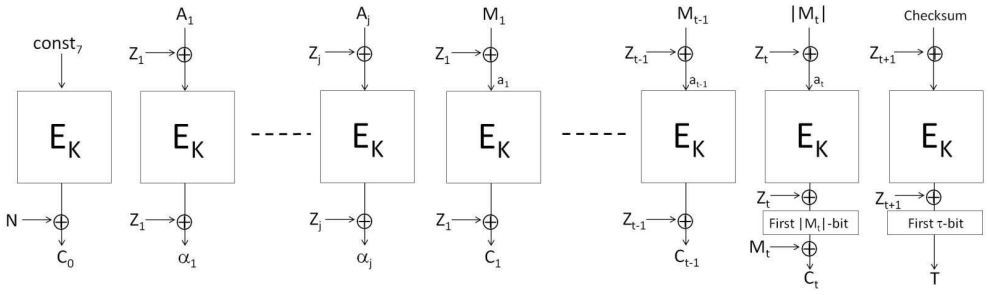
도면21



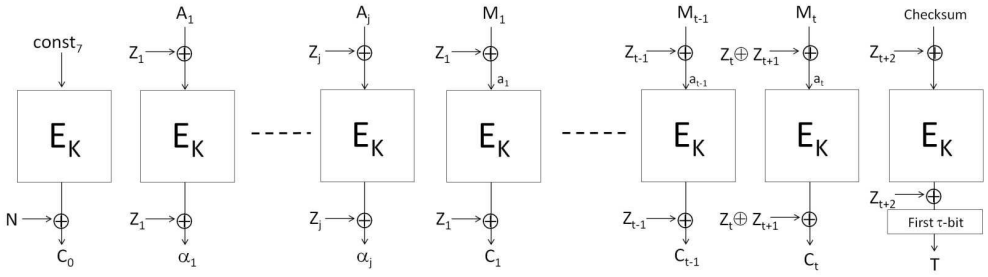
도면22



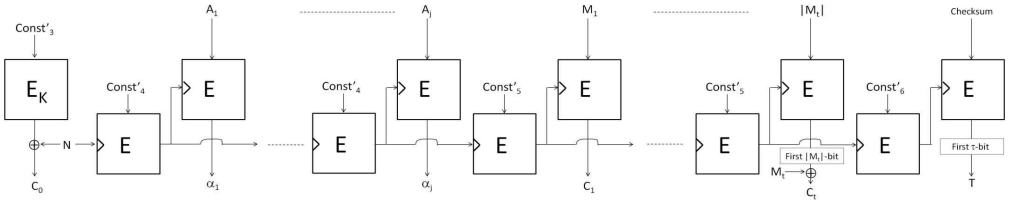
도면23



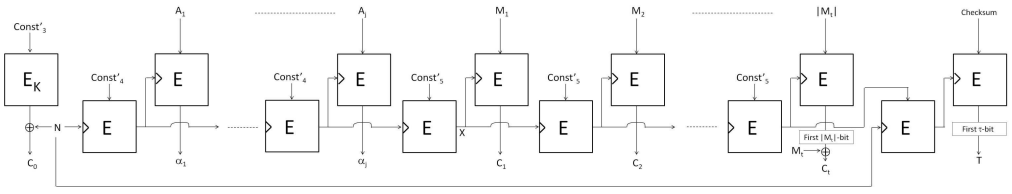
도면24



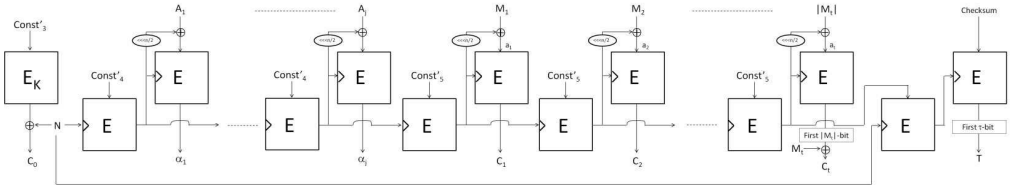
도면25



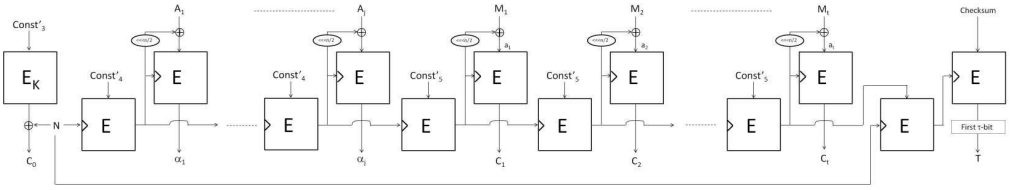
도면26



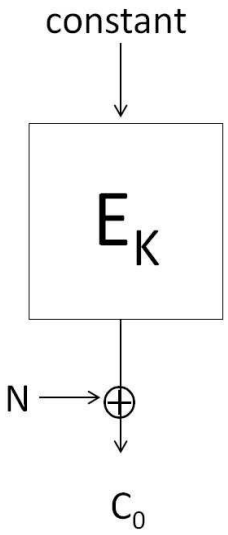
도면27



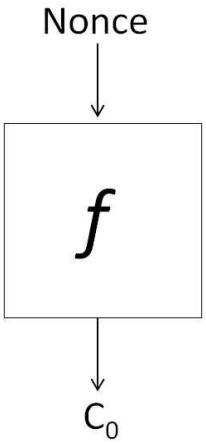
도면28



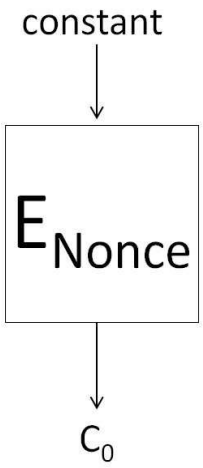
도면29



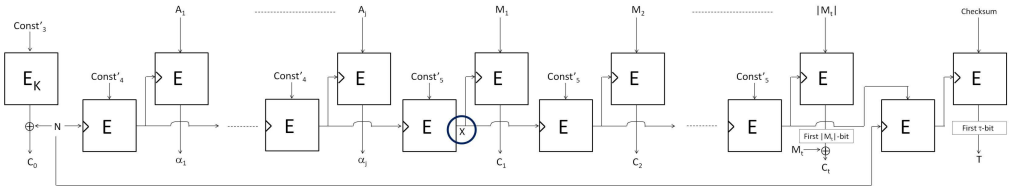
도면30



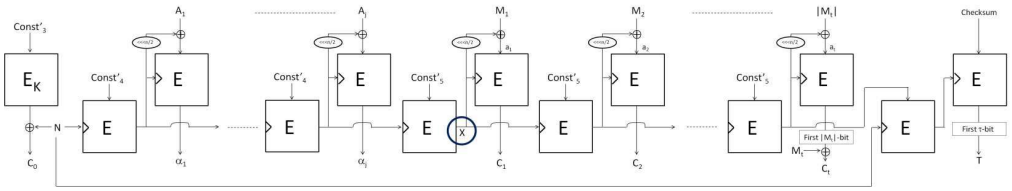
도면31



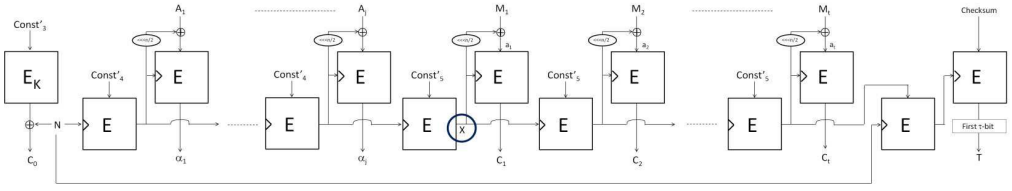
도면32



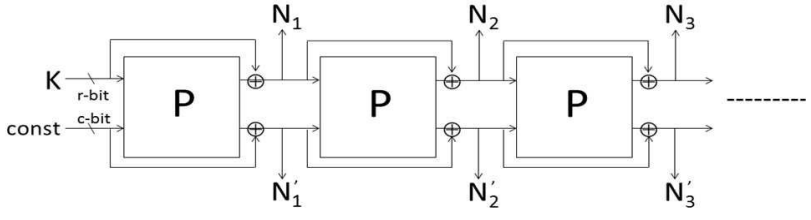
도면33



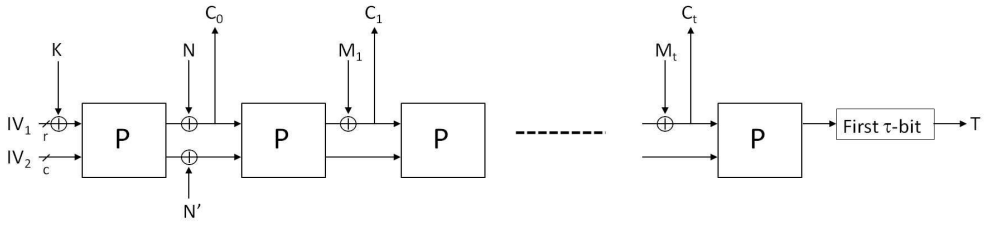
도면34



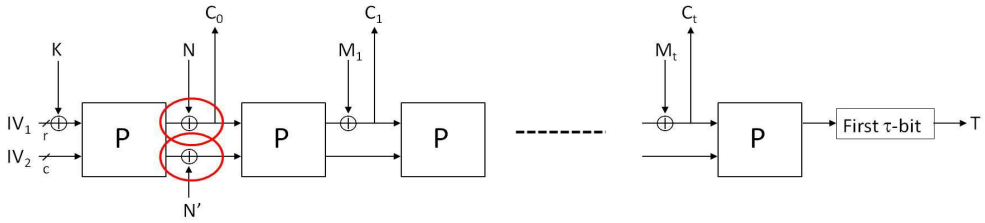
도면35



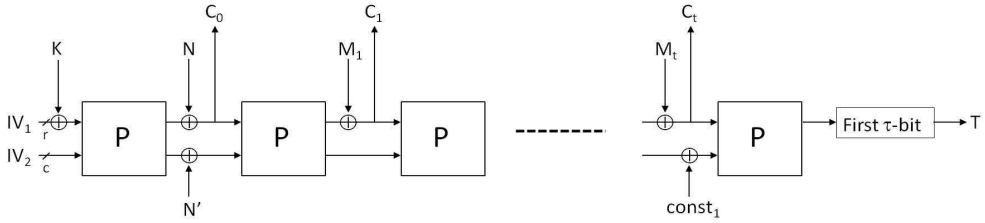
도면36



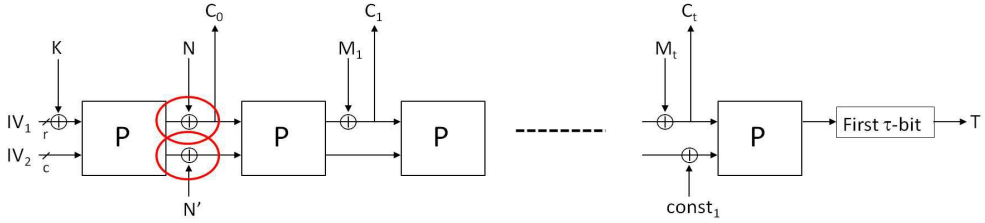
도면37



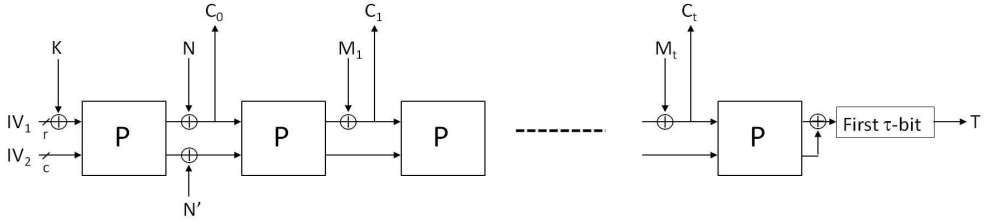
도면38



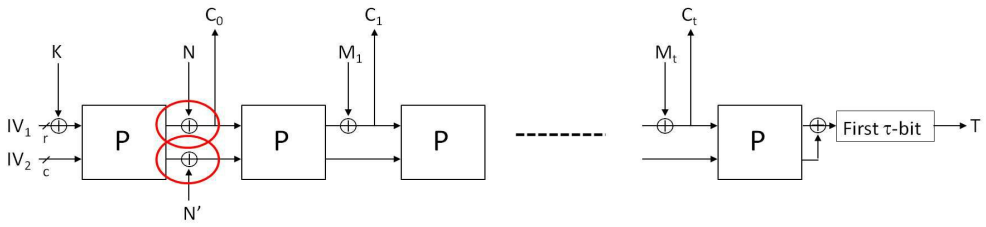
도면39



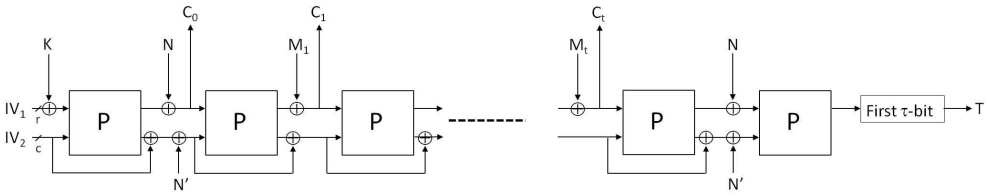
도면40



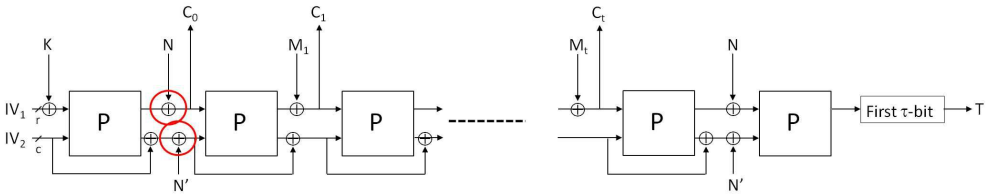
도면41



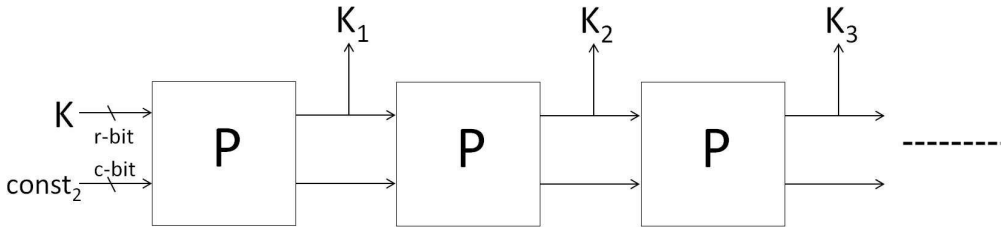
도면42



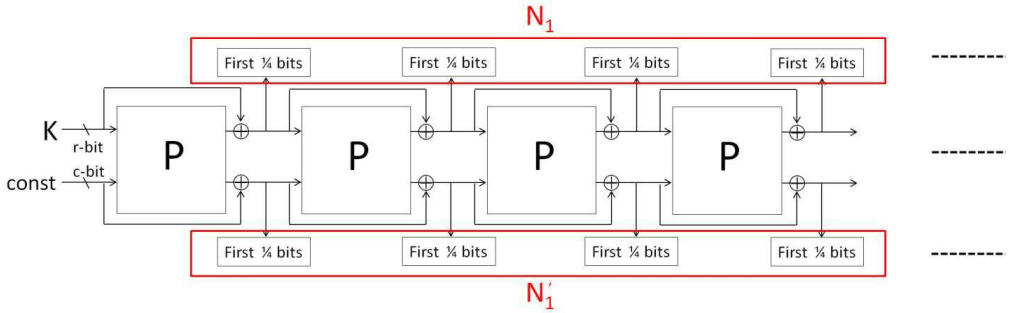
도면43



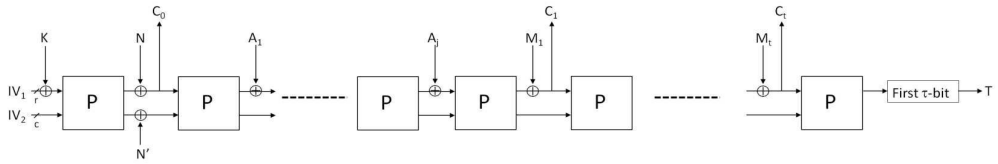
도면44



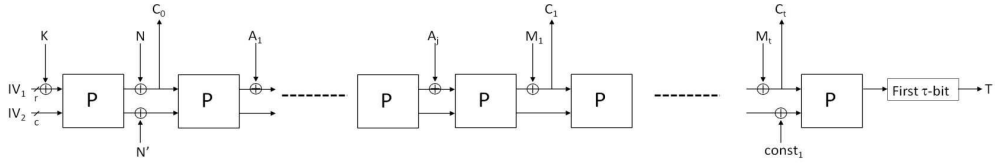
도면45



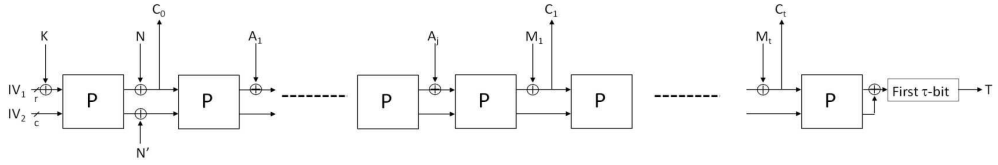
도면46



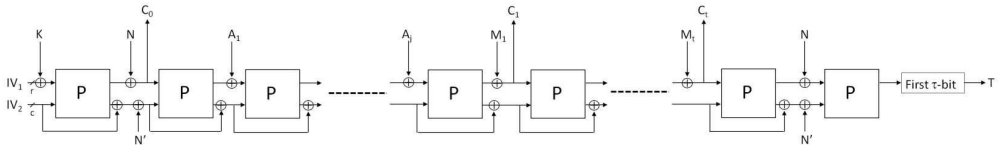
도면47



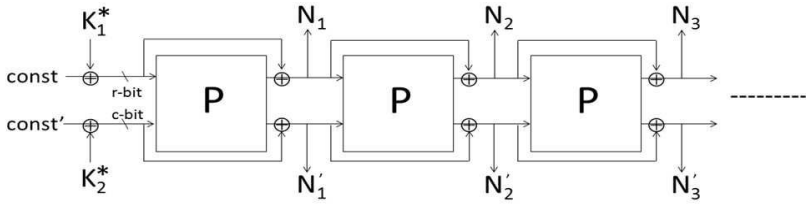
도면48



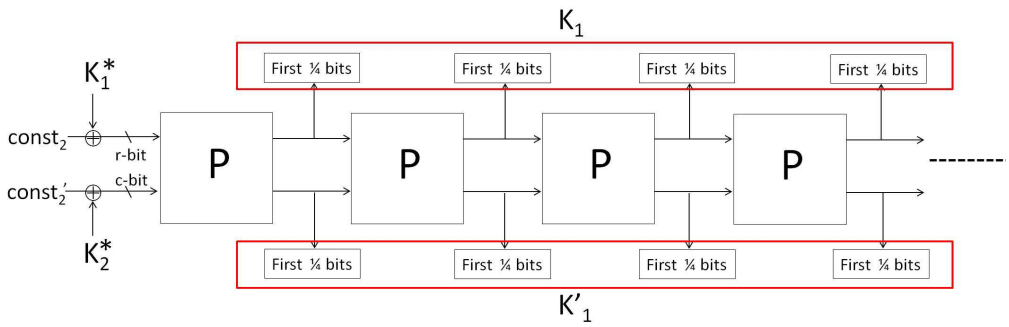
도면49



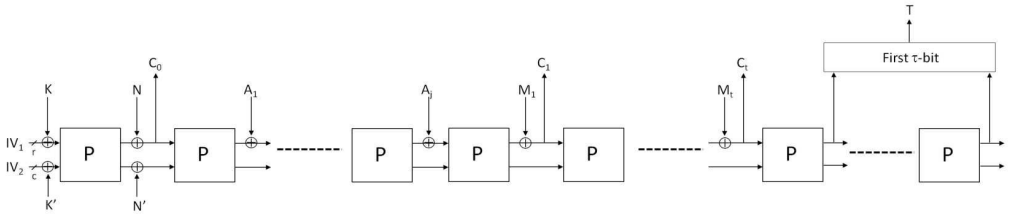
도면50



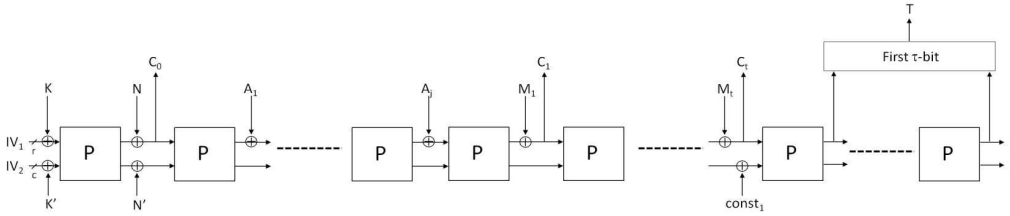
도면51



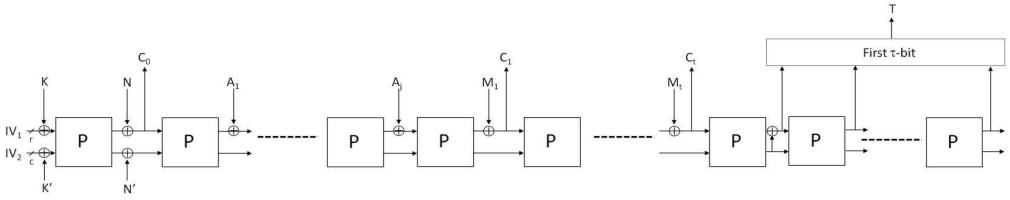
도면52



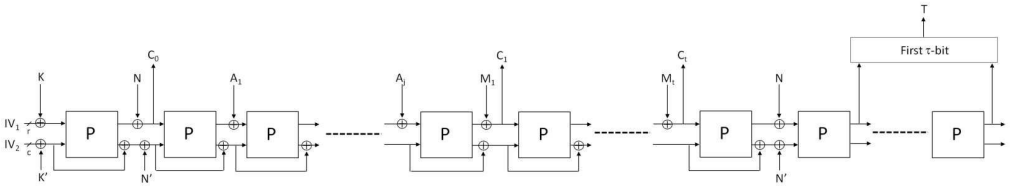
도면53



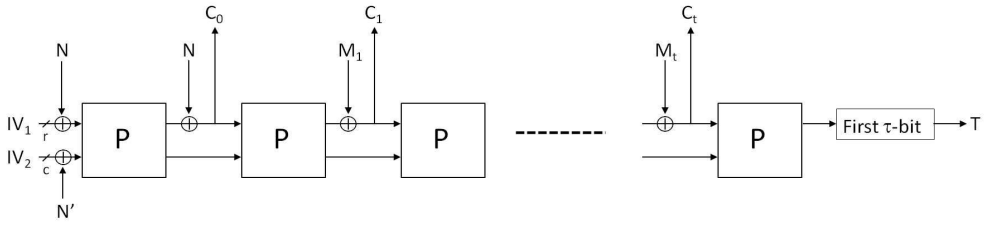
도면54



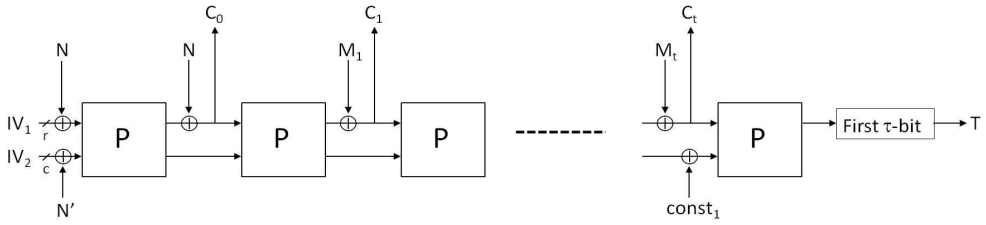
도면55



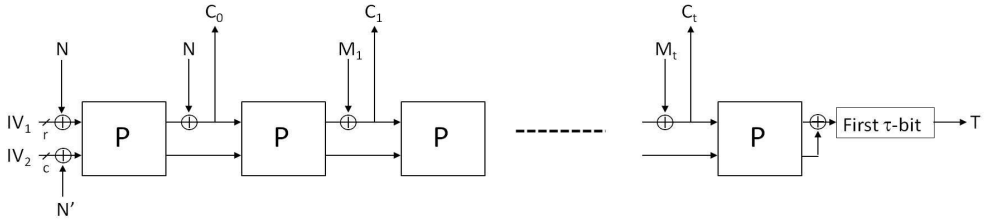
도면56



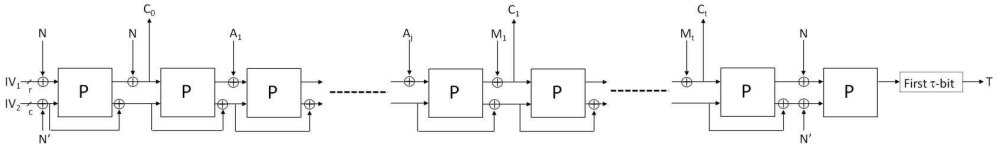
도면57



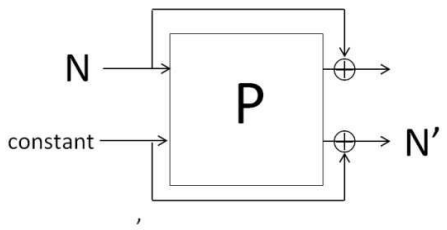
도면58



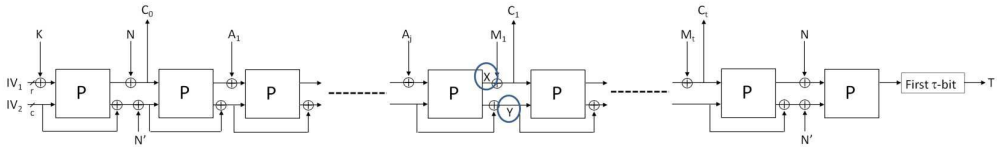
도면59



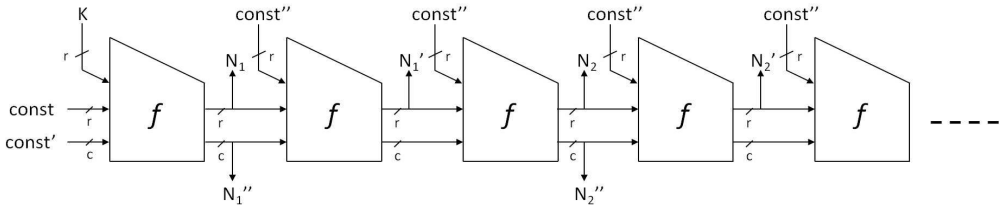
도면60



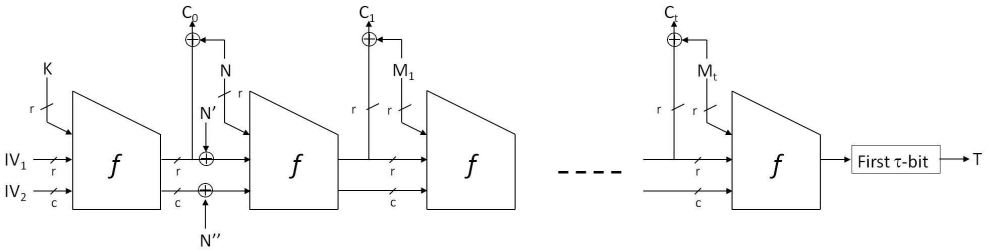
도면61



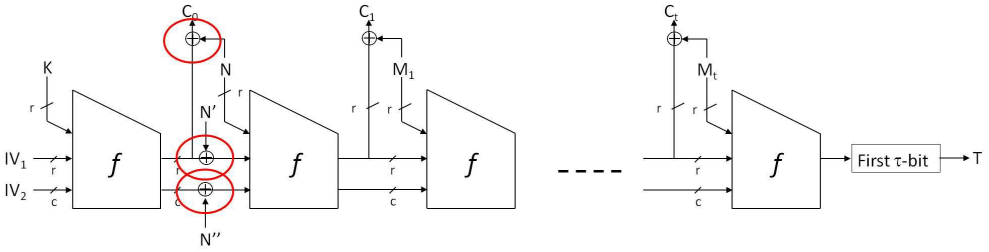
도면62



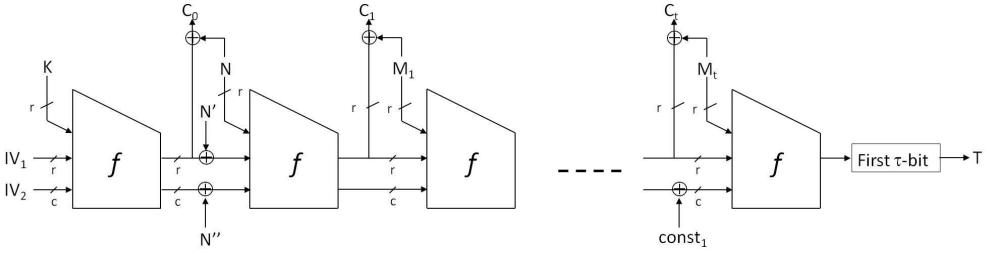
도면63



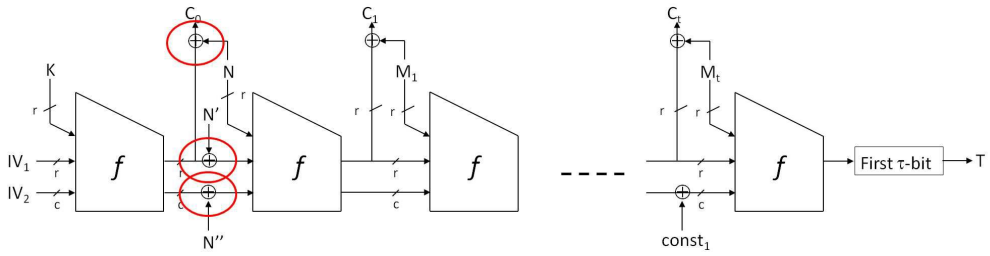
도면64



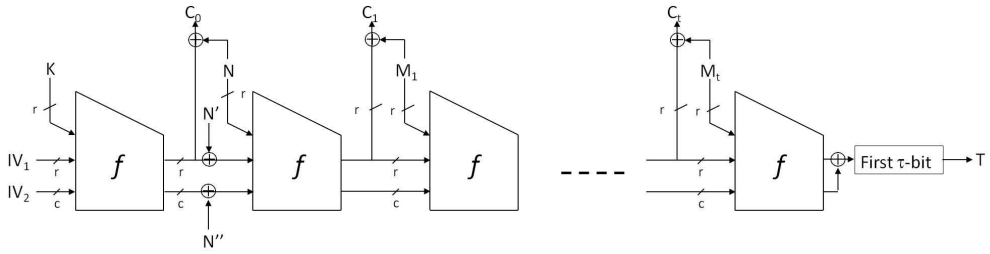
도면65



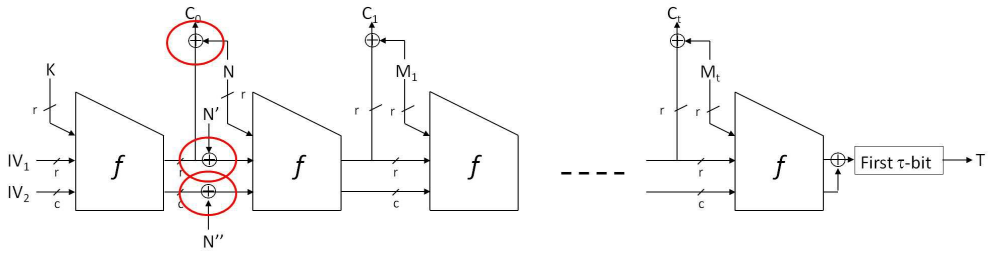
도면66



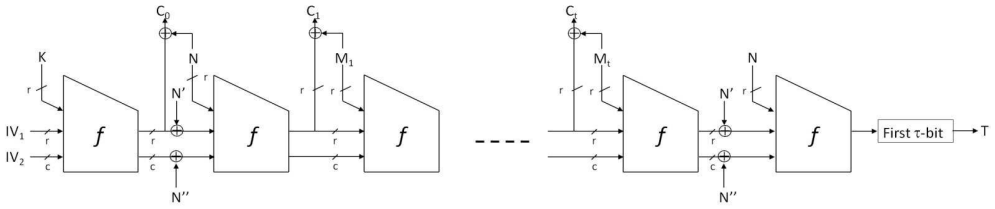
도면67



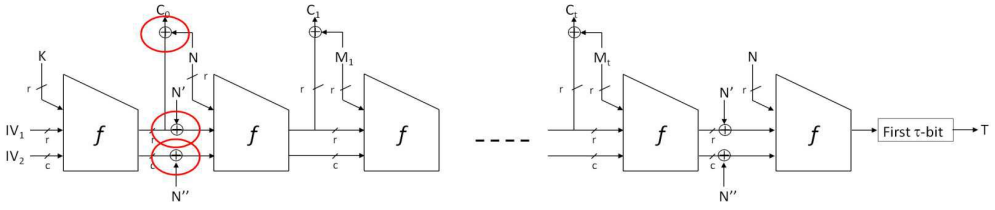
도면68



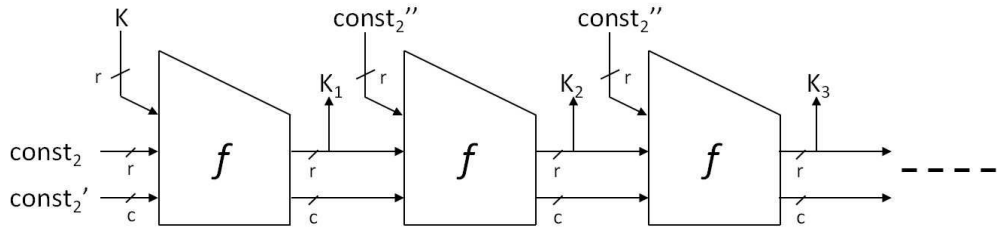
도면69



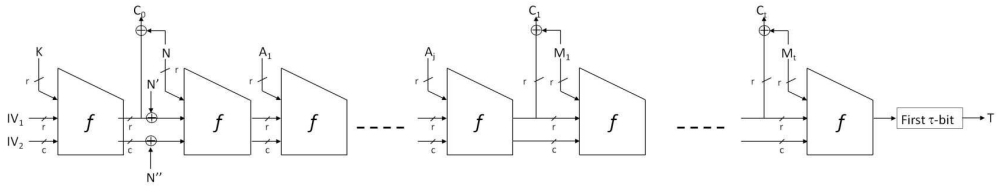
도면70



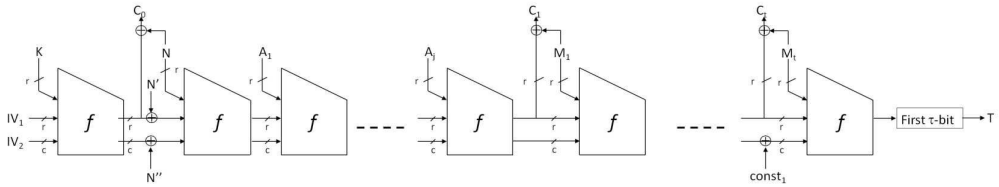
도면71



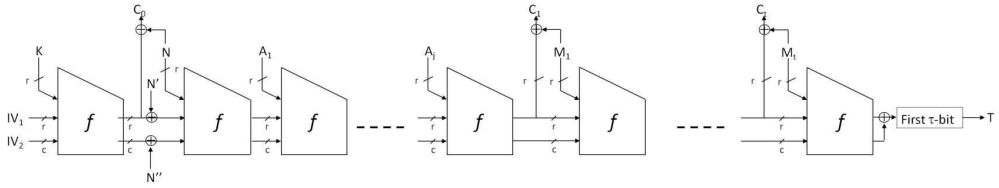
도면72



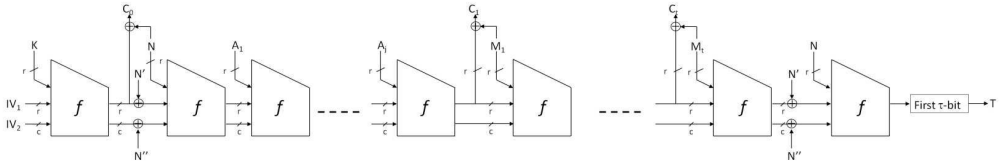
도면73



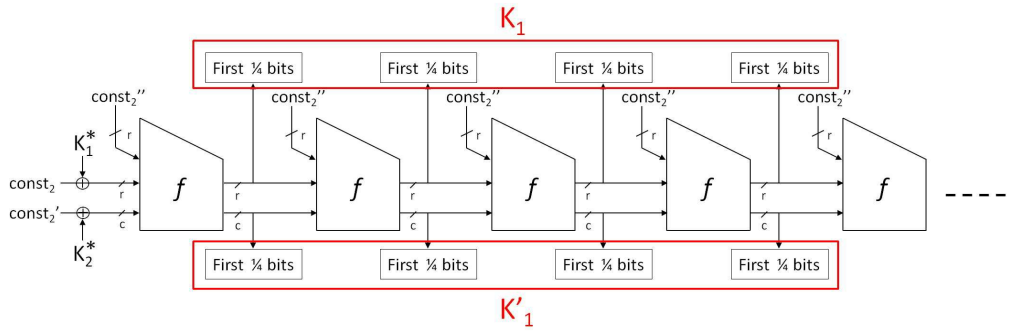
도면74



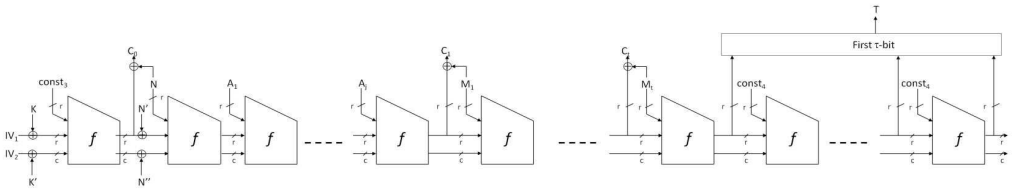
도면75



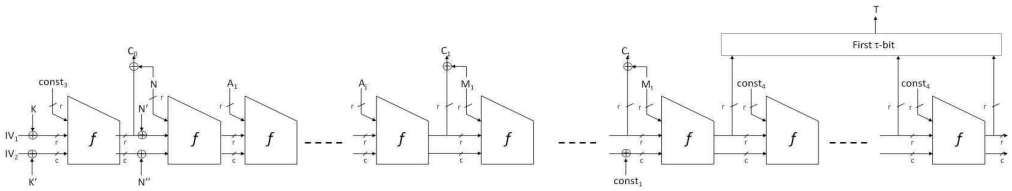
도면76



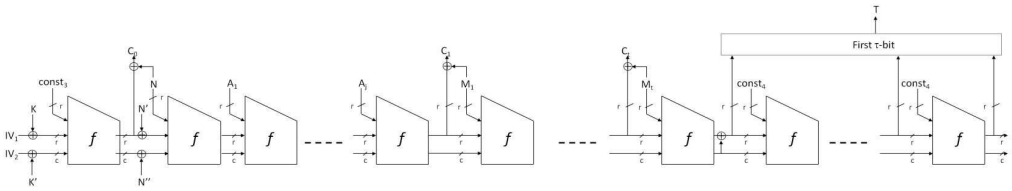
도면77



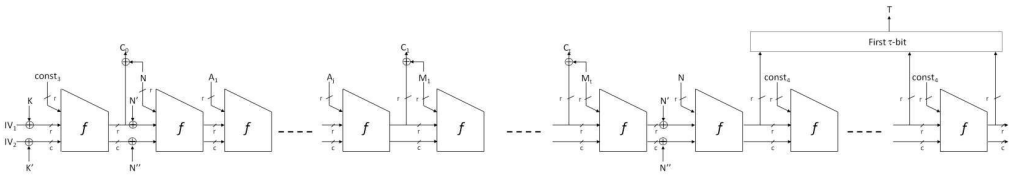
도면78



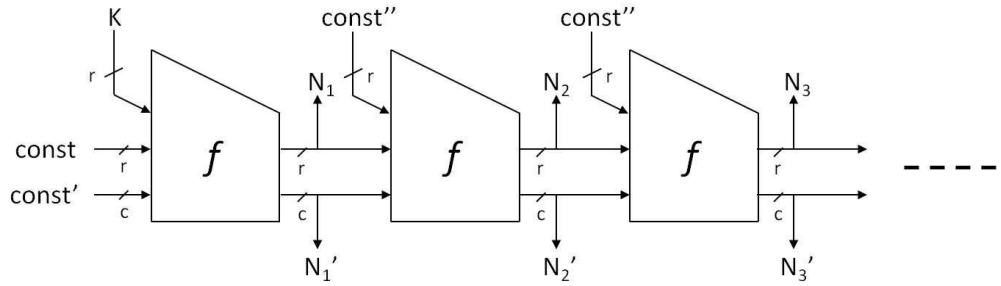
도면79



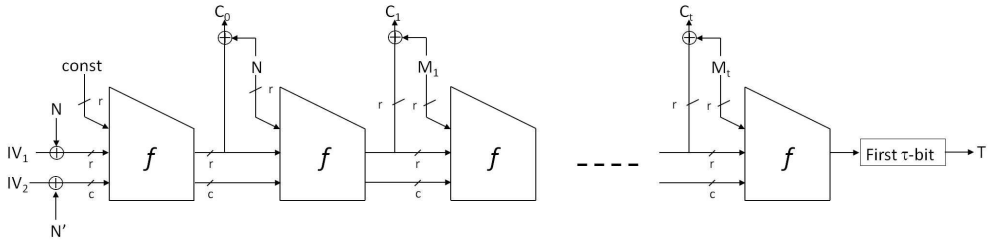
도면80



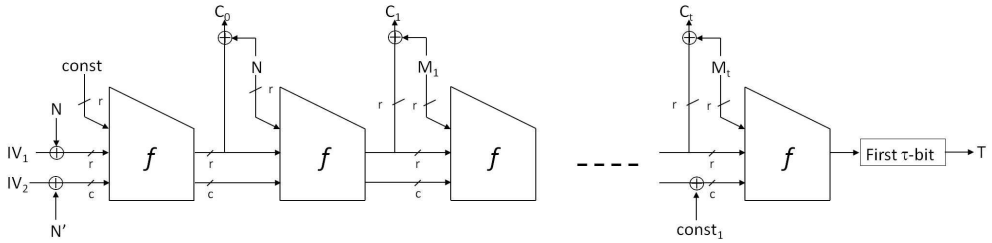
도면81



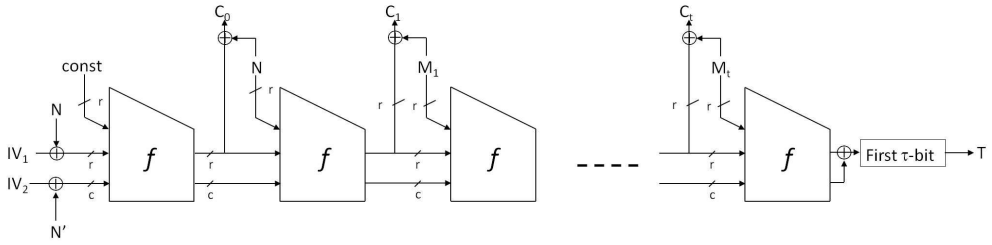
도면82



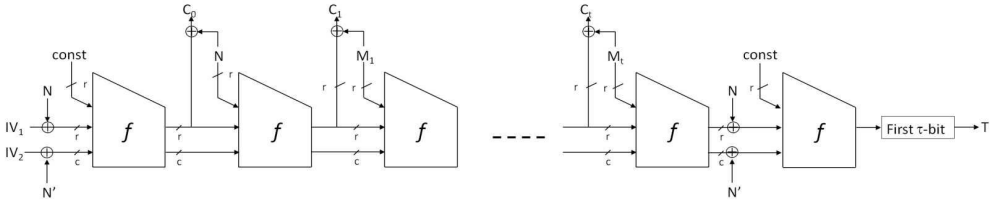
도면83



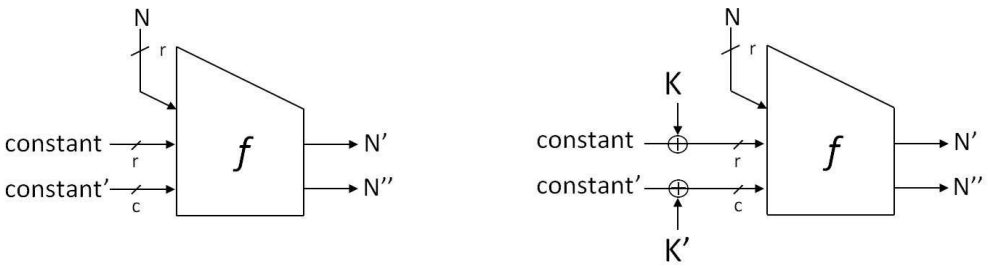
도면84



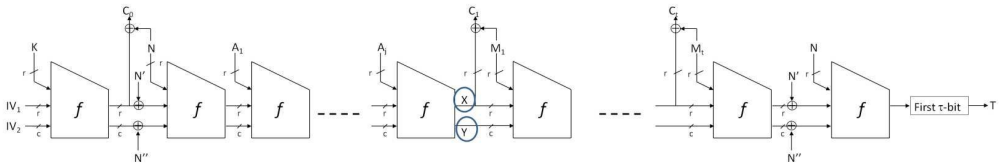
도면85



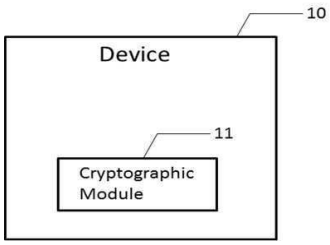
도면86



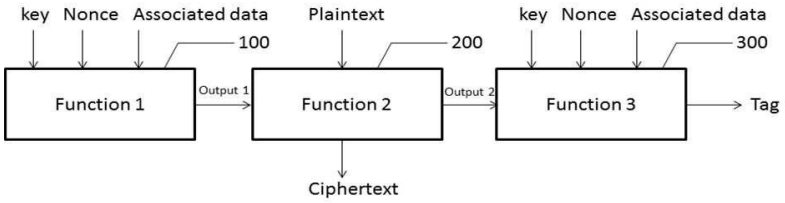
도면87



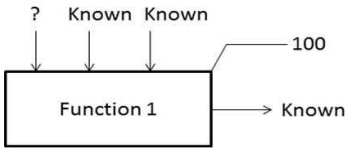
도면88



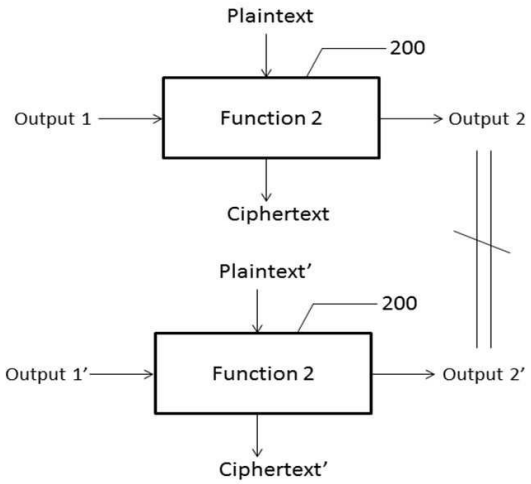
도면89



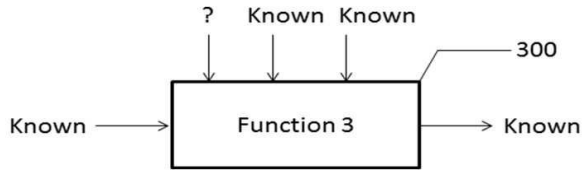
도면90



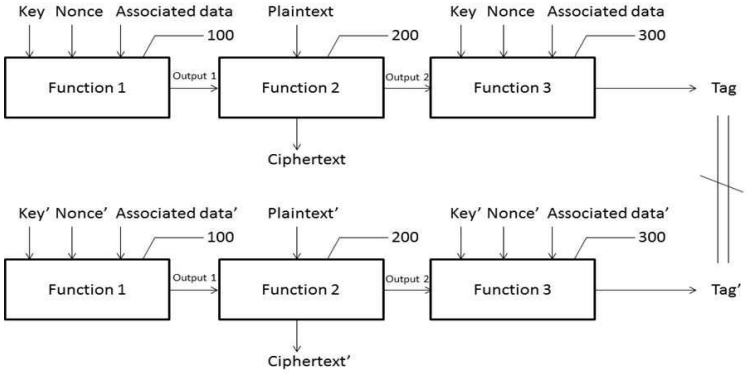
도면91



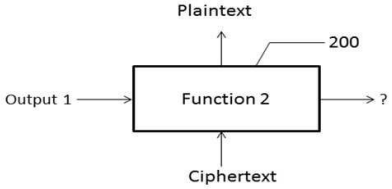
도면92



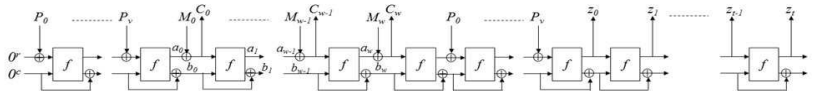
도면93



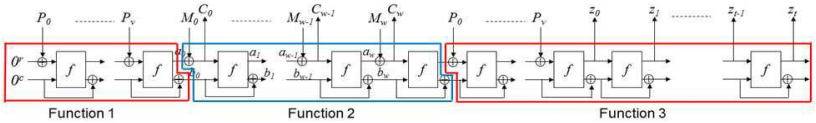
도면94



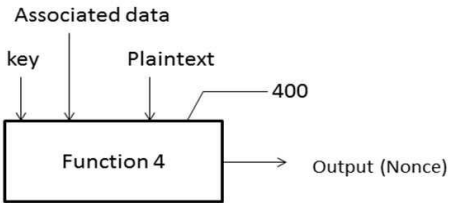
도면95



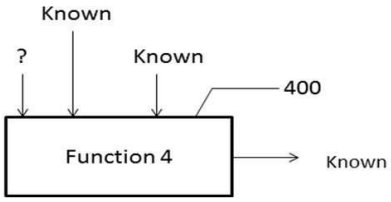
도면96



도면97



도면98



도면99