(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2010-0079060
(43) 공개일자 2010년07월08일

(51) Int. Cl.
H04L 9/28 (2006.01) *G06F 9/06* (2006.01)
G06F 7/00 (2006.01)
(21) 출원번호 10-2008-0137471
(22) 출원일자 2008년12월30일
심사청구일자 없음

(71) 출원인
고려대학교 산학협력단
서울 성북구 안암동5가1 고려대학교 내
한국정보보호진흥원
서울특별시 송파구 가락동 78번지 IT벤처타워 서관
(72) 발명자
홍석희
서울 은평구 응암2동 451-12 202호
김희석
서울특별시 강서구 화곡6동 롯데 낙천대 아파트 103동 802호
(뒷면에 계속)
(74) 대리인
특허법인충현

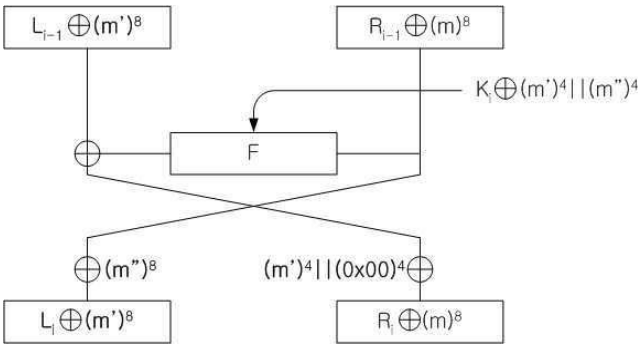
전체 청구항 수 : 총 11 항

(54) 마스킹이 적용된 ＳＥＥＤ를 이용한 암호화 방법

(57) 요 약

마스킹이 적용된 SEED를 이용한 암호화 방법이 개시된다. 본 발명의 일 실시 예에 따른 마스킹이 적용된 SEED를 이용한 암호화 방법은 매 라운드마다 해당 라운드의 라운드 키로 암호화된 오른쪽 입력을 해당 라운드의 왼쪽 입력과 배타적 논리합 (XOR) 연산하는, 페이스텔 구조의 SEED 알고리즘을 이용하는 암호화 방법에 있어서, 입력 평문을 제1 난수로 마스킹하는 단계; 및 상기 마스킹된 입력 평문을 최초 입력으로 하여, 매 라운드마다 해당 라운드의 라운드키를 제2 난수로 마스킹한 후, 각 라운드의 입력값 중 오른쪽 입력을 상기 마스킹된 라운드 키로 암호화하는 단계를 포함한다. 본 발명의 실시 예들에 의하면, 기존의 암호화 기법인 SEED 알고리즘에 마스킹을 도입하여 암호시스템의 안전성을 향상시킬 수 있고, 연산에 필요한 추가 비용의 증가를 최소한으로 억제할 수 있다.

대 표 도 - 도5



(72) 발명자

정해일

경상남도 통영시 인평동 430-50번지 3동 301호

조성민

서울특별시 성북구 장위3동 6-104 2층 201호

박해룡

인천광역시 연수구 연수동 우성2차아파트 203동 103호

전길수

경기도 성남시 분당구 정자동 정든마을 신화아파트 507동 2502호

특허청구의 범위

청구항 1

매 라운드마다 해당 라운드의 라운드 키로 암호화된 오른쪽 입력을 해당 라운드의 왼쪽 입력과 배타적 논리합(XOR) 연산하는, 페이스텔 구조의 SEED 알고리즘을 이용하는 암호화 방법에 있어서,

입력 평문을 제1 난수로 마스킹하는 단계; 및

상기 마스킹된 입력 평문을 최초 입력으로 하여, 매 라운드마다 해당 라운드의 라운드키를 제2 난수로 마스킹한 후, 각 라운드의 입력값 중 오른쪽 입력을 상기 마스킹된 라운드 키로 암호화하는 단계

를 포함하는, 마스킹이 적용된 SEED를 이용한 암호화 방법.

청구항 2

제 1 항에 있어서,

매 라운드마다 상기 암호화된 오른쪽 입력과 상기 왼쪽 입력을 XOR 연산한 후 제3 난수로 마스킹하고 상기 오른쪽 입력을 제4 난수로 마스킹하여, 각 라운드의 마스킹 값을 동일하게 유지하는 단계를 더 포함하는 것을 특징으로 하는, 마스킹이 적용된 SEED를 이용한 암호화 방법.

청구항 3

제 1 항에 있어서,

마지막 라운드에서 오른쪽 입력과 왼쪽 입력의 마스킹을 제거하는 단계를 더 포함하는 것을 특징으로 하는, 마스킹이 적용된 SEED를 이용한 암호화 방법.

청구항 4

제 1 항에 있어서,

상기 암호화하는 단계는,

매 라운드마다 각 라운드의 입력값 중 오른쪽 입력을 상기 마스킹된 라운드 키로 암호화하는 수정된 F함수를 적용하는 단계를 포함하는, 마스킹이 적용된 SEED를 이용한 암호화 방법.

청구항 5

제 4 항에 있어서,

상기 수정된 F함수를 적용하는 단계는,

상기 오른쪽 입력의 마스킹을 유지시키는 마스킹 S박스를 포함하는 수정된 G함수와 마스킹을 유지시키는 마스킹 덧셈 연산을 적용하는 단계를 포함하는 것을 특징으로 하는, 마스킹이 적용된 SEED를 이용한 암호화 방법.

청구항 6

제 4 항에 있어서,

상기 수정된 F함수를 적용하는 단계는,

상기 오른쪽 입력을 구성하는 제1입력과 제2입력을 마스킹하는 단계;

상기 마스킹된 제1입력과 제2입력을 XOR 연산한 후 마스킹을 유지시키는 마스킹 S박스를 포함하는 수정된 G함수를 적용하여 제1값을 출력하는 단계;

상기 제1값을 상기 마스킹된 제1입력과 마스킹 덧셈하고, 상기 마스킹 덧셈의 결과에 수정된 G함수를 적용하여 제2값을 출력하는 단계;

상기 제2값과 상기 제1값을 마스킹 덧셈하고, 상기 마스킹 덧셈의 결과에 수정된 G함수를 적용하여 제3값을 출력하는 단계;

상기 제2값과 상기 제3값을 마스킹 덧셈하는 단계

를 포함하는, 마스킹이 적용된 SEED를 이용한 암호화 방법.

청구항 7

제 5 항 또는 제 6 항 중 어느 한 항에 있어서,

상기 마스킹 덧셈 연산은,

불대수 마스킹을 산술 마스킹으로 변환하는 함수 및 산술 마스킹을 불대수 마스킹으로 변환하는 함수가 적용되는 것을 특징으로 하는, 마스킹이 적용된 SEED를 이용한 암호화 방법.

청구항 8

매 라운드마다 해당 라운드의 라운드 키로 암호화된 오른쪽 입력을 해당 라운드의 왼쪽 입력과 배타적 논리합(XOR) 연산하는, 페이스텔 구조의 SEED 알고리즘을 이용하는 암호화 방법에 있어서,

128비트의 입력 평문을 마스킹하는 단계;

상기 마스킹된 입력 평문을 최초 입력으로 하여, 매 라운드마다 해당 라운드의 라운드키를 마스킹한 후, 각 라운드의 입력값 중 최상위 64비트의 오른쪽 입력을 상기 마스킹된 라운드 키로 암호화하는 단계; 및

마지막 라운드에서 최상위 64 비트의 오른쪽 입력과 최하위 64비트의 왼쪽 입력의 마스킹을 제거하는 단계

를 포함하는, 마스킹이 적용된 SEED를 이용한 암호화 방법.

청구항 9

제 8 항에 있어서,

상기 암호화하는 단계

상기 오른쪽 입력의 마스킹을 유지시키는 마스킹 S박스를 포함하는 수정된 G함수와 마스킹을 유지시키는 마스킹 덧셈 연산을 적용하는 단계를 포함하는, 마스킹이 적용된 SEED를 이용한 암호화 방법.

청구항 10

제 9 항에 있어서,

상기 마스킹 덧셈 연산은,

4비트에 대한 산술 대 불대수 마스킹 변환 테이블 (FT)와 캐리 테이블 (CT)를 사용하는 마스킹 변환 함수를 적용하는 것을 특징으로 하는, 마스킹이 적용된 SEED를 이용한 암호화 방법.

청구항 11

제1항 내지 제10항 중 어느 한 항에 따른 암호화 방법을 컴퓨터 시스템에서 실행하기 위한 프로그램이 기록된 기록매체로서, 상기 컴퓨터 시스템이 판독할 수 있는 상기 기록매체.

명 세 서

발명의 상세한 설명

기 술 분 야

[0001] 본 발명은 암호하 기법에 관한 것으로, 특히, 암호시스템의 안전성을 보장하기 위해 기존의 암호화 기법인 SEED 알고리즘에 마스킹을 도입하는 마스킹이 적용된 SEED를 이용한 암호화 방법에 관한 것이다.

배 경 기 술

[0002] 스마트카드는 소형 컴퓨터의 능력을 가진 신용카드 크기의 보안장치이므로 휴대가 간편하다. 사이버세계에서 스마트카드는 가치이전의 수단뿐만 아니라 전화카드, 이동통신 보안수단, 신분증, 교통카드, 금융 IC 카드 등 그

활용분야가 아주 다양하기 때문에 정보통신망 환경에서 스마트카드가 중요한 보안장치로 수요나 활용 면에서 급격한 증가율을 보이고 있다. 생활의 많은 부분이 인터넷 환경으로 전이되는 시점에서 스마트카드는 중요한 역할을 수행할 수 있는 도구이다. 이러한 보안장치에 대한 안전성 평가는 증명 가능한 안전성이나 계산적 안전성과 같은 이론에 기반을 두고 있다. 그러나 1996년 폴 코처 (Paul Kocher)가 부채널 공격 (Side Channel Attack)에 대한 개념을 소개함으로써 이론적으로 안전한 암호시스템에 대해서도 물리적 공격인 부채널 공격에 의해 비밀정보가 누출될 수 있음을 보였다. 물리적 공격은 알고리즘의 수행 시간, 전력 소모량 등의 부가적인 정보를 이용한다. 알고리즘의 수행 시간을 이용한 타이밍 공격은 IC칩의 암호알고리즘이 비밀키와 관련된 정보를 처리하는데 걸리는 시간과 그렇지 않은 데이터를 처리하는데 걸리는 시간 사이의 차를 분석해 비밀키를 알아내는 방법이다. 전력분석 공격도 비밀키와 관련된 데이터를 처리할 때 IC 칩이 사용하는 전력량과 그렇지 않을 경우의 전력량을 비교분석하는 것으로 단순 전력 공격 (Simple Power Attack; SPA), 차분 전력 공격 (Diffrencial Power Attack; DPA) 등으로 나뉜다. 전력분석 공격은 폴 코처가 개발한 것으로 가장 위협적인 공격방법 중 하나로 알려져 있고 그중에서도 DPA는 강력한 분석방법으로 여러 가지 형태로 발전하고 있고 적용분야가 많다.

[0003] 이러한 부채널 공격이 소개되면서 많은 암호시스템 설계자들은 효율적인 대응법들을 연구하기 시작했고, 차분 전력 분석(Differential Power Analysis, DPA)에 대한 대응법으로는 마스킹 대응법(masking method)이 활발히 연구되고 있다. 마스킹 기법은 알고리즘의 변형을 통해 일차 차분 전력 분석을 방어하는 방법으로 잡음 삽입, 임의의 지연, 랜덤 클록과 같은 하드웨어적인 대응법에 비해 그 비용이 저렴하다. 따라서 단가가 저렴한 스마트카드와 같은 장비에서는 마스킹 기법이 일반적으로 가장 선호된다.

[0004] 마스킹 기법이란 암호화/복호화 도중 중간 값의 정보를 숨기는 방법으로 암호화 장비가 랜덤한 전력 소비를 하도록 구성해 일차 차분 전력 분석에 안전하도록 대응하는 방법이다. 구체적으로 평문 x에 대하여 암호문 y를 얻기 위해 마스킹 난수 m을 이용하여 $x \oplus m (\oplus : \text{xor})$의 암호문 $y'(=y \oplus m')$을 구한 후, 최종적으로 y를 얻기 위해 $y' \oplus m'$의 연산을 수행한다. 물론, 경우에 따라 마스킹 기법은 다르게 구성될 수도 있다. 따라서 암호화 과정에서 중간 값을 알 수 없기 때문에 일반적인 전력 분석 공격은 성공할 수 없다.

[0005] 이러한 마스킹 기법을 사용한 경우, $x \oplus m$의 암호문 $y'(=y \oplus m')$에서 m'을 알아야 실제 원하는 암호문 y를 얻을 수 있다. 하지만 블록암호 알고리즘은 비선형 연산을 수행하므로 수정되지 않은 블록 암호 시스템에서 m'값은 x에 따라 다르며 이 값을 중간 값의 누출 없이 아는 것도 상당한 연산을 필요로 한다. 따라서 마스킹 대응법은 이 비선형 연산에 대한 고려가 불가피하다.

[0006] 이하에서, $a \oplus b$는 'a'와 'b'의 배타적 논리합 (Exclusive-Or) 비트 연산이고, $a + b$는 $(a+b) \bmod 2^{32}$이다. $a \& b$는 'a'와 'b'의 논리곱 (AND) 비트 연산이다. 또한, $X^{\ll s}$는 X를 s비트 만큼 왼쪽으로 순환 이동하는 연산이고, $X^{\gg s}$는 X를 s비트 만큼 오른쪽으로 순환 이동하는 연산이다. L_i는 i라운드에서 출력된 왼쪽 메시지 블록(상위 64 비트)이고, R_i는 i라운드에서 출력된 오른쪽 메시지 블록(하위 64 비트)이다. 특히, L_0는 입력비트의 상위 64비트를 나타내고, R_0는 입력비트의 하위 64비트를 나타낸다.

[0007] 한편, $K_i=(K_{i,0}, K_{i,1})$는 i라운드의 라운드 키(64 비트)이다. 여기서, $K_{i,0}$는 i라운드 F함수의 오른쪽 입력키(32 비트)이고, $K_{i,1}$은 i라운드 F함수의 왼쪽 입력키(32 비트)이다. KC_i는 라운드 키 생성과정에서 사용되는 i+1라운드 상수이다.

[0008] 도 1은 종래의 SEED 알고리즘의 흐름을 나타낸 것이다.

[0009] SEED는 1999년 한국정보보호진흥원에 의해 개발된 블록암호로써 128 비트의 비밀키를 사용하는 페이스텔 (Feistel) 구조로 이루어져 있다. 페이스텔 구조란 각각 n/2비트인 L_0, R_0 으로 이루어진 n비트 평문 블록 (L_0, R_0)가 r라운드$(r \geq 1)$를 거쳐 암호문 (L_r, R_r)으로 변환되는 반복 구조이다.

[0010] SEED 알고리즘의 전체적인 구조는 페이스텔 구조로 이루어져 있으며, 128비트의 평문 블록과 128비트 키를 입력으로 사용하여 총 16라운드를 거쳐 128비트 암호문을 출력한다.

[0011] SEED 알고리즘은 크게 페이스텔 구조로 되어 있는 F함수와 라운드 키 생성함수로 나누어진다. F함수에는 SEED에서 유일하게 비선형성을 가지고 있는 S박스를 포함하는 G함수가 포함된다. 키 생성함수는 128비트의 키를 받아서 64비트의 16라운드의 키를 생성한다.

[0012] 도 2는 F함수의 구조를 도시한 것이다.

[0013] 페이스텔 구조를 갖는 블록 암호 알고리즘은 F함수의 특성에 따라 구분될 수 있다. SEED의 F함수는 수정된 64비트 페이스텔 형태로 구성된다. 도 2에서, 'F'로 표시된 부분이 F함수를 나타내고, '+'로 표시된 부분은 캐리가 없는 덧셈 연산을 의미한다. F함수는 각 32비트 블록 2개 (C,D)를 입력으로 받아 32비트 블록 2개 (C',D')를 출력한다. 여기서, F함수의 입력 중 상위 32비트를 C로 나타내고, 하위 32비트를 D로 나타낸다. 암호화 과정에서 64비트 블록 (C,D)와 64비트 라운드 키 $K_i=(K_{i,0};K_{i,1})$를 F함수의 입력으로 처리하여 64비트 블록 (C',D')을 출력한다.

[0014] 다음은 G함수에 대해 설명한다. 상술한 바와 같이, SEED의 F함수는 페이스텔 구조를 이루고 있으며 F함수 내에는 S박스를 포함하고 있는 G함수가 위치하고 있다.

[0015] G함수는 다음의 수학식 1과 같은 연산을 통해 이루어진다.

수학식 1

[0016] $$Y_3 = S_2[X_3],\ Y_2 = S_1[X_2],\ Y_1 = S_2[X_1],\ Y_0 = S_1[X_0],$$

[0017] $$\begin{aligned} Z_3 &= (Y_0 \& m_3) \oplus (Y_1 \& m_0) \oplus (Y_2 \& m_1) \oplus (Y_3 \& m_2) \\ Z_2 &= (Y_0 \& m_2) \oplus (Y_1 \& m_3) \oplus (Y_2 \& m_0) \oplus (Y_3 \& m_1) \\ Z_1 &= (Y_0 \& m_1) \oplus (Y_1 \& m_2) \oplus (Y_2 \& m_3) \oplus (Y_3 \& m_0) \\ Z_0 &= (Y_0 \& m_0) \oplus (Y_1 \& m_1) \oplus (Y_2 \& m_2) \oplus (Y_3 \& m_3) \end{aligned}$$

[0018] $(m_0 = 0xfc,\ m_1 = 0xf3,\ m_2 = 0xcf,\ m_3 = 0x3f)$

[0019] 여기서, X는 G함수의 입력 값(32 비트)으로서 $X=(X_3 \| X_2 \| X_1 \| X_0)$이고, Y는 G함수에서 S박스 (S-box) (S_1, S_2)의 출력 값(32 비트)으로서 $Y=(Y_3 \| Y_2 \| Y_1 \| Y_0)$이며, Z는 G함수의 출력 값(32 비트)으로서 $Z=(Z_3 \| Z_2 \| Z_1 \| Z_0)$이다. m_i는 상수이다. 또한, '||' 는 둘 이상의 비트들을 연속하여 배치하는 연쇄(concatenation)를 나타낸다.

[0020] 다음은 S박스에 대해 설명한다. G함수의 내부에 사용되는 비선형 S박스 S_1, S_2는 수학식 2를 이용하여 생성된다.

수학식 2

[0021] $$S_i : Z_{2^8} \rightarrow Z_{2^8},\ S(x) = A^{(i)} \cdot x^{n_i} \oplus b_i \quad (n_1 = 247,\ n_2 = 251,\ b_1 = 159, b_2 = 56)$$

[0022] 여기서, 행렬 $A^{(i)}$는 수학식 3과 같이 표현될 수 있다.

수학식 3

[0023] $$A^{(1)} = \begin{pmatrix} 10001010 \\ 11111110 \\ 10000101 \\ 01000010 \\ 01000101 \\ 00100001 \\ 10001000 \\ 00010100 \end{pmatrix}, \quad A^{(2)} = \begin{pmatrix} 01000101 \\ 10000101 \\ 11111110 \\ 00100001 \\ 10001010 \\ 10001000 \\ 01000010 \\ 00010100 \end{pmatrix}$$

[0024] 한편, SEED의 라운드 키 생성과정은 128비트 암호화 키를 64비트씩 좌우로 나누어 이들을 교대로 8비트씩 좌/우

로 회전이동한 후, 회전 이동된 결과의 4워드들에 대한 간단한 산술연산 및 G함수를 적용하는 과정이다. 라운드 키 생성과정은 암호화나 복호화 시 암호키로부터 필요한 라운드 키를 간단히 계산할 수 있도록 설계된다. 이는 기본적으로 모든 라운드 키를 저장할 수 없는 등으로 제한된 자원을 갖는 스마트카드 등의 응용에서 효율성을 향상시키기 위한 것이다.

[0025] 각 라운드 i에 사용되는 라운드 키 $K_i=(K_{i,0};K_{i,1})$는 표 1과 같이 생성한다.

표 1

Round Key Generation Algorithm :

입력 $K=(K_{0,0};K_{0,1})=A\|B\|C\|D$ (128 bits)

출력 round key $K_i=(K_{i,0};K_{i,1})$

1. For i from 1 to 16

1.1. $K_{i,0} \leftarrow G(A+C-KC_{i-1})$;

1.2. $K_{i,1} \leftarrow G(B-D+KC_{i-1})$;

[0026] 1.3. If (i%2==1) then $A\|D \leftarrow (A\|B)^{\gg 8}$;

[0027] 위 알고리즘에서 사용되는 라운드 상수는 표 2와 같다.

표 2

[0028]

라운드 상수	
$KC_0=0x9e3779b9$	$KC_8=0x3779b99e$
$KC_1=0x3c6ef373$	$KC_9=0x6ef3733c$
$KC_2=0x78dde6e6$	$KC_{10}=0xdde6e678$
$KC_3=0xf1bbcdcc$	$KC_{11}=0xbbcdccf1$
$KC_4=0xe3779b99$	$KC_{12}=0x779b99e3$
$KC_5=0xc6ef3733$	$KC_{13}=0xef3733c6$
$KC_6=0x8dde6e67$	$KC_{14}=0xde6e678d$
$KC_7=0x1bbcdccf$	$KC_{15}=0xbcdccf1b$

[0029] DPA는 암호 알고리즘에서 사용되는 키와 전력 소비파형에서의 연관관계에 기반한 공격이다. 즉, 암호 알고리즘 내부에서 연산되는 데이터가 전력 소비파형에 영향을 미친다는 특성을 이용한 것이다. DPA 공격 방법에 대응하기 위한 방법은 소프트웨어적인 방법과 하드웨어적인 방법으로 나뉜다. 소프트웨어적인 방법의 장점은 다양한 구현 환경에서 유연하게 적용 가능하다는 것이다. 하드웨어적인 방법은 특정 하드웨어에 구현되어 알고리즘이 변경되면 하드웨어 자체도 바꾸어야 한다는 점과 추가적인 실리콘이 필요하다는 단점이 있으나 소프트웨어적인 방법보다 효율성이 높다.

[0030] 소프트웨어적인 방법에는 대표적으로 마스킹 기법과 하이딩 (hiding) 기법이 있다. 마스킹 기법의 목적은 암호 연산 과정에서 계산되는 중간 결과들이 전력소비 파형에 어떤 확률적인 연관 관계를 갖지 못하도록 하기 위하여 입력과 중간 결과들을 랜덤한 값으로 마스킹을 씌운다. 하이딩 기법은 중간 결과값에 랜덤 마스킹을 씌워 중간 결과값을 예측하지 못하게 만드는 마스킹 기법과는 달리, 소비전력의 데이터 종속성을 없애기 위하여 알고리즘의 순서를 랜덤하게 조정하는 방법이다. 즉, 알고리즘의 연산 시퀀스를 랜덤하게 조정하여 중간 결과값들이 시간에 대하여 랜덤하게 분포하도록 한다.

[0031] 알고리즘 내에서 계산되는 중간 결과값들을 조정하여 소비전력에 독립적으로 만드는 소프트웨어적인 방법과는 달리, 하드웨어적인 방법에서는 중간 결과값들을 조정하지 않는다. 즉, 하드웨어적인 방법에서는 알고리즘 내에서 계산되는 중간 결과값들은 DPA 대응 방법에 영향을 받지 않는다. 예를 들어, 하드웨어 내부 버스(bus)를 랜덤하게 프리차징 (precharging)하거나 버스를 마스킹하는 등의 방법을 사용한다. 또는 더미 주기 (dummy cycle)를 랜덤하게 삽입하거나 클록 펄스 (clock pulse)를 건너뛰는 방법을 통하여 하드웨어적으로 더 많은 노이즈

가 발생하게 한다. 셀 (Cell) 레벨에서 곱셈기나 회로 자체에 마스킹을 적용한 대응 방법도 존재한다.

[0032] 종래에는 DPA에 안전하도록 하기 위한 많은 대응 방법들이 존재한다. 이들 각각이 장단점을 가지고 있기 때문에, 암호 알고리즘이 구현되는 환경을 고려하여 적절한 대응 방법을 선택하여 적용할 필요가 있다.

[0033] 상술한 SEED는 비선형 연산으로 S박스 연산과 덧셈 연산을 모두 사용하며, 한 라운드 당 두 연산을 각각 세 번씩 사용하는 구조로 다른 블록 암호에 비해 마스킹 적용이 용이하지 않다. 예를 들어 AES의 경우 하나의 S박스가 유일한 비선형 연산으로 마스킹 방법의 설계 시 이 S박스에 대한 마스킹 방법의 적용이 이루어진다면 전체 구조에서도 쉽게 설계될 수 있다. 반면 SEED의 경우, 두 개의 S박스 이외에도 32비트 덧셈 연산에 대한 마스킹 방법도 고려하여야 하며, 두 비선형 연산에 대한 마스킹 방법이 조화를 이룰 수 있는 전체 구조에 대한 설계도 동시에 이루어져야 한다.

[0034] 비록 SEED에 대한 마스킹 기법의 적용이 이와 같이 용이하지 않더라도 실제 상용 스마트카드에 구현되는 블록 암호에 대한 마스킹 방법의 적용이 이루어지지 않는다면, 전력 분석 공격자에게 스마트카드의 비밀 정보를 공개하는 것과 같다. 장기적인 관점에서 보다 안전한 암호 시스템의 하드웨어와 소프트웨어 설계 및 구현을 위해, 그리고 급격히 늘어나는 스마트카드 시장을 고려해 볼 때 이러한 암호시스템의 안전성 보장을 위한 기술의 개발은 시급하게 이루어져야 한다. 따라서 스마트카드에 대한 전력 분석 공격의 위협이 날로 증가하는 시점에서 SEED에 적합한 마스킹 기법의 개발이 필요하다.

발명의 내용

해결 하고자하는 과제

[0035] 본 발명이 이루고자 하는 기술적 과제는 암호시스템의 안전성을 보장하기 위해 기존의 암호화 기법인 SEED 알고리즘에 마스킹을 도입하는 마스킹이 적용된 SEED를 이용한 암호화 방법을 제공하는 데 있다.

과제 해결수단

[0036] 상기의 기술적 과제를 이루기 위하여, 본 발명의 일 실시 예에 따른 마스킹이 적용된 SEED를 이용한 암호화 방법은 매 라운드마다 해당 라운드의 라운드 키로 암호화된 오른쪽 입력을 해당 라운드의 왼쪽 입력과 배타적 논리합 (XOR) 연산하는, 페이스텔 구조의 SEED 알고리즘을 이용하는 암호화 방법에 있어서, 입력 평문을 제1 난수로 마스킹하는 단계; 및 상기 마스킹된 입력 평문을 최초 입력으로 하여, 매 라운드마다 해당 라운드의 라운드 키를 제2 난수로 마스킹한 후, 각 라운드의 입력값 중 오른쪽 입력을 상기 마스킹된 라운드 키로 암호화하는 단계를 포함한다.

[0037] 바람직하게는, 본 발명의 일 실시 예에 따른 마스킹이 적용된 SEED를 이용한 암호화 방법은 매 라운드마다 상기 암호화된 오른쪽 입력과 상기 왼쪽 입력을 XOR 연산한 후 제3 난수로 마스킹하고 상기 오른쪽 입력을 제4 난수로 마스킹하여, 각 라운드의 마스킹 값을 동일하게 유지하는 단계를 더 포함할 수 있다.

[0038] 바람직하게는, 본 발명의 일 실시 예에 따른 마스킹이 적용된 SEED를 이용한 암호화 방법은 마지막 라운드에서 오른쪽 입력과 왼쪽 입력의 마스킹을 제거하는 단계를 더 포함할 수 있다.

[0039] 바람직하게는, 상기 암호화하는 과정에서, 매 라운드마다 각 라운드의 입력값 중 오른쪽 입력을 상기 마스킹된 라운드 키로 암호화하는 수정된 F함수를 적용할 수 있다. 이 경우, 상기 수정된 F함수를 적용하는 과정에서, 상기 오른쪽 입력의 마스킹을 유지시키는 마스킹 S박스를 포함하는 수정된 G함수와 마스킹을 유지시키는 마스킹 덧셈 연산을 적용할 수 있다. 이 경우, 상기 수정된 F함수를 적용하는 과정은, 상기 오른쪽 입력을 구성하는 제1입력과 제2입력을 마스킹하는 단계; 상기 마스킹된 제1입력과 제2입력을 XOR 연산한 후 마스킹을 유지시키는 마스킹 S박스를 포함하는 수정된 G함수를 적용하여 제1값을 출력하는 단계; 상기 제1값을 상기 마스킹된 제1입력과 마스킹 덧셈하고, 상기 마스킹 덧셈의 결과에 수정된 G함수를 적용하여 제2값을 출력하는 단계; 상기 제2값과 상기 제1값을 마스킹 덧셈하고, 상기 마스킹 덧셈의 결과에 수정된 G함수를 적용하여 제3값을 출력하는 단계; 및 상기 제2값과 상기 제3값을 마스킹 덧셈하는 단계를 포함할 수 있다.

[0040] 바람직하게는, 상기 마스킹 덧셈 연산에는 불대수 마스킹을 산술 마스킹으로 변환하는 함수 및 산술 마스킹을 불대수 마스킹으로 변환하는 함수가 적용될 수 있다.

[0041] 상기의 기술적 과제를 이루기 위하여, 본 발명의 다른 실시 예에 따른 마스킹이 적용된 SEED를 이용한 암호화 방법은 매 라운드마다 해당 라운드의 라운드 키로 암호화된 오른쪽 입력을 해당 라운드의 왼쪽 입력과 배타적

논리합 (XOR) 연산하는, 페이스텔 구조의 SEED 알고리즘을 이용하는 암호화 방법에 있어서, 128비트의 입력 평문을 마스킹하는 단계; 상기 마스킹된 입력 평문을 최초 입력으로 하여, 매 라운드마다 해당 라운드의 라운드키를 마스킹한 후, 각 라운드의 입력값 중 최상위 64비트의 오른쪽 입력을 상기 마스킹된 라운드 키로 암호화하는 단계; 및 마지막 라운드에서 최상위 64 비트의 오른쪽 입력과 최하위 64비트의 왼쪽 입력의 마스킹을 제거하는 단계를 포함한다.

[0042] 바람직하게는, 상기 암호화하는 단계에서, 상기 오른쪽 입력의 마스킹을 유지시키는 마스킹 S박스를 포함하는 수정된 G함수와 마스킹을 유지시키는 마스킹 덧셈 연산을 적용할 수 있다. 이때, 상기 마스킹 덧셈 연산은 4비트에 대한 산술 대 불대수 마스킹 변환 테이블 (FT)와 캐리 테이블 (CT)를 사용하는 마스킹 변환 함수를 적용할 수 있다.

효 과

[0043] 본 발명의 실시 예들에 의하면, 기존의 암호화 기법인 SEED 알고리즘에 마스킹을 도입하여 암호시스템의 안전성을 향상시킬 수 있고, 연산에 필요한 추가 비용의 증가를 최소한으로 억제할 수 있다.

발명의 실시를 위한 구체적인 내용

[0044] 이하에서는 도면을 참조하여 본 발명의 바람직한 실시 예를 설명하기로 한다. 그러나, 다음에 예시하는 본 발명의 실시 예는 여러 가지 다른 형태로 변형될 수 있으며, 본 발명의 범위가 다음에 상술하는 실시 예에 한정되는 것은 아니다.

[0045] 마스킹 기법은 일차 차분 전력 분석에 안전한 소프트웨어적인 대응법으로 그 비용이 랜덤 지연, 랜덤 클럭, 잡음 삽입과 같은 하드웨어적인 대응법에 비해 비교적 적은 편이며 따라서 스마트카드와 같은 저가의 장비에 암호 설계를 위해 가장 선호되는 대응법이다. 마스킹 기법이란 암호화 장비에서 알고리즘이 수행되는 모든 중간 연산 값 a를 난수 m에 의해서 $a \oplus m$, $a+m$ 등의 형태로 숨기는 방법이며 이는 마스킹 기법을 사용 했을 경우 중간의 전력을 얻게 되어도 본 연산의 중간 값 a에 대한 전력이 아닌 난수 값 m에 의해 변형된 값에 대한 전력을 얻게 되어 일차 차분 전력 분석 공격에 대한 방어를 가능하게 한다.

[0046] 알고리즘의 특성에 따라 다양하게 적용 가능하지만 블록암호의 경우에는 일반적으로 산술 (Arithmetic) 또는 불대수 (Boolean)의 방법으로 마스킹이 적용된다. 이 두 마스킹 방법은 난수 r에 대해 중간 연산 값 x를 다음과 같이 마스킹 한다.

[0047] 산술 마스킹의 예는 수학식 4와 같다. 여기서, A는 마스킹 출력값이고, K는 임의의 정수이다.

수학식 4

[0048] $$A=x-r \bmod 2^K \quad \text{또는} \quad A=x+r \bmod 2^K$$

[0049] 불대수 마스킹의 예는 수학식 5와 같다. 여기서, x'은 마스킹 출력값이다.

수학식 5

[0050] $$x' = x \oplus r$$

[0051] 블록 암호에 대한 마스킹 기법은 중간 연산 결과 값을 랜덤하게 만들어 일차 차분 전력 분석에 안전하게 한다. 연산의 효율성을 위해 블록 암호의 전체 라운드가 아닌 첫 라운드와 마지막 라운드만을 마스킹 하는 방법이 제안되기도 하였다. 하지만 이 방법은 IPA (Inferential Power Analysis) 방법을 통해서 안전하지 않다는 것이 밝혀졌다.

[0052] 그에 반해 풀라운드 마스킹 방법 (Full Round Masking Method)은 알고리즘의 모든 라운드의 모든 수행과정에 대한 중간 연산 값이 마스킹에 의해 랜덤화 되도록 연산을 수행하는 방법이다. 이와 같이 풀라운드 마스킹에서 모든 중간 값에 대해 마스킹을 적용하기 위해서는 알고리즘의 수행 후에 제거할 마스킹의 값을 알 수 있어야 한다. 일반적으로 선형 연산에 대한 마스킹 값은 이전의 마스킹 값으로 쉽게 유추될 수 있지만 비선형 연산에 대한 마스킹 값의 변환은 유추하기가 힘들며 마스킹 기법의 설계 시 이 부분에 대한 고려가 불가피하다.

[0053] 이하에서는 다음의 각 연산들에 따라 연산 수행 후 결과 값에 대한 마스킹 값을 살펴본다.

[0054] 테이블 룩업 (Table Lookup), 비트의 불대수 (Bitwise Boolean), 시프트/로테이션 (Shift/Rotation), 곱셈 (Multiplication) $\bmod 2^n$, 덧셈 (Addition) $\bmod 2^n$, $GF(2^8)$에 대한 다항식 곱셈 (Polynomial Multiplication over $GF(2^8)$), 비트의 퍼뮤테이션 (Bitwise Permutation), 선형변환 (Linear Transformation) 등의 연산들에 대해 입력 값 및 중간 계산 값을 숨기기 위해서 상술한 불대수 마스킹과 산술 마스킹을 사용한다. 두 가지의 마스킹 기법을 사용하여 위의 연산들에 대한 부채널 공격의 대응방법을 구성한다.

[0055] 첫 번째로, 룩업 테이블 연산을 수행할 때는 입력 값 x에 대해서 연산을 수행하기 전에 r_{in}을 통해서 한 번의 랜덤화를 수행하고, 테이블 룩업 연산 수행 이후에 r_{out}을 이용해서 두 번째 랜덤화를 수행한다. 랜덤화를 수행하는 계산식은 수학식 6과 같다.

수학식 6

[0056] $$T'[x] = T[x \oplus r_{in}] \oplus r_{out}$$

[0057] $T[x]$는 x에 대한 테이블 룩업 연산을 의미한다. 마스킹을 수행하도록 설계된 T'은 마스킹된 값을 입력 값으로 받아서 마스킹된 결과 값을 출력한다. 즉 T'은 r_{in}으로 마스킹된 입력 값에 대해 r_{out}으로 마스킹된 결과 값을 출력한다.

[0058] 두 번째로, 불대수 연산이다. 불대수 연산 중 암호 수행 시에 자주 사용은 XOR와 AND이다. 입력 값에 대한 마스킹 연산을 수행하면서 정확한 마스킹 값을 계산해서 유지 하는 것이 가장 중요하다. 마스킹된 두 값에 대해 XOR 연산이 수행될 경우 다음과 같이 마스킹 값이 적용될 수 있다. 두 입력 값 x,y에 대한 마스킹 값을 각각 r_x, r_y라고 할 때 마스킹이 씌워진 값을 x'와 y'라고 가정하면, 마스킹이 씌워진 결과 값 $z' = x' \oplus y'$이고, 마스킹 값 $r_z = r_x \oplus r_y$ 이다. 또한 AND 연산에 대해서는 다음과 같이 마스킹이 적용될 수 있다. 이 경우에도 XOR에서 정의한 것과 마찬가지로 두 입력 값 x,y 에 대해서 마스킹이 적용될 값을 각각 r_x, r_y, 마스킹이 씌워진 값을 x'와 y' 라고 가정하면, 마스킹이 씌워진 결과 값 $z' = x' \wedge y'$ 이고, 마스킹 값 $r_z = (r_x \wedge y') \oplus (r_y \wedge x') \oplus (r_x \wedge r_y)$ 이다. AND 연산에서 마스킹 연산을 수행하는 도중, x나 y의 값이 마스킹 되지 않고 드러나지 않을 수도 있다. 이부분에 대해서 세심한 구현이 요구된다.

[0059] 세 번째로, 시프트나 로테이션이 수행되는 경우에는 다음과 같이 마스킹이 변형된다. 입력 값 x에 대해서 n비트의 시프트나 로테이션 연산이 수행되는 경우, x'는 x에 대해서 r_x의 마스킹 값을 사용하여 마스킹된 값이라 하면 x'에 대해서 n비트의 시프트나 로테이션이 수행되었을 때 그 값에 해당하는 마스킹 값은 다음과 같이 계산된다. 즉, 시프트나 로테이션이 수행된 입력 값 $x' \gg n$ 일때, $x' \gg n$ 에 해당되는 마스크 값은 $r_x \gg n$이다. 데이터에 의존해서 시프트나 로테이션이 수행되는 경우에는 다음과 같은 방법으로 시프트나 로테이션의 횟수도 랜덤하게 생성할 수 있다. 즉, 시프트나 로테이션이 수행된 입력 값이 $(x' \gg n) \gg r_n$ 일때, $(x' \gg n) \gg r_n$에 해당되는 마스크 값은 $(r_x' \gg n) \gg r_n$ 이다.

[0060] 네 번째로, 덧셈 또는 곱셈이 수행되는 경우에 대해 설명한다. $\bmod 2^n$ 에서의 산술연산의 경우에는 산술 마스킹 방법을 사용한다. 두 데이터 x,y에 대한 마스킹 값을 각각 r_x, r_y, x,y의 마스킹된 값을 x',y' 라고 가정하면, 두 값의 산술연산 + 및 ×에 대한 연산 결과의 변환된 마스킹 값들은 다음과 같이 정리 된다. 즉, + 에 대한 연

산 결과는 $z'=(x'+y') \bmod 2^n$ 이고, + 에 대한 연산 마스킹 값은 $r_z=(r_x+r_y) \bmod 2^n$ 이다. 또한, ×에 대한 연산 결과는 $z'=(x'\times y') \bmod 2^n$ 이고, ×에 대한 연산 마스킹 값 $r_z=(r_x\times y'+r_y\times x'+r_x\times r_y) \bmod 2^n$ 이다.

[0061] 다섯 번째로, 다항식 곱셈에 대해 설명한다. 불대수 마스킹 방법이 적용되어 있는 값에 대해서 다항식 곱셈을 수행할 때 마스킹 값을 변환하는 여러 가지 방식이 있다. 연산의 결과를 테이블에 저장해서 사용하는 경우에는 마스킹 테이블 룩업 방식을 적용하면 된다. 또한 $GF(2^8)$에서는 데이터에 r_g를 XOR 연산으로 마스킹 하는 것이 산술 마스킹 방법에서 $r_g(x)$를 더해주는 것과 같은 결과를 낳기 때문에 산술 마스킹 방법을 적용할 수 있다.

[0062] 여섯 번째로, 비트 퍼뮤테이션에 대해 설명한다. 데이터 x에 대한 마스킹 값을 r_x라고 할 때 x의 마스킹된 값을 x'이라 하자. 이 경우, 퍼뮤테이션 연산이 $z'=\pi_p[x']$로 수행되었을 때 마스킹 값 또한 $r_z=\pi_p[r_x]$로 계산된다.

[0063] 일곱 번째로, 선형변환의 경우에는 퍼뮤테이션에서 사용했던 것과 같은 방법으로 랜덤 마스킹 값을 변환할 수 있다. 데이터 x에 대한 마스킹 값은 r_x, x의 마스킹된 값을 x'라고 하자. 선형변환을 LT라고 할 때, $z'=LT[x']$에 대한 마스킹 값은 $r_z=LT[r_x]$로 계산된다.

[0064] 이러한 방법으로 알고리즘을 구현하면, 연산을 수행하는 과정과 랜덤한 마스킹 값을 유지하는 과정이 필요하므로, 일반 알고리즘에 비해 추가적인 연산비용이 소모된다. 또한 테이블 룩업을 수행하는 경우 테이블을 생성하는 부분, 불대수 마스킹과 산술 마스킹의 변환이 이루어지는 부분 등에서도 추가적인 비용이 발생한다. 이러한 추가적인 비용은 테이블의 크기와 마스킹의 변환이 이루어지는 빈도에 따라 좌우된다.

[0065] DPA 공격에 대한 마스킹 형태의 기존 대응 방법들은 중간 계산에서의 마스킹 값에 대한 상태를 계속적으로 살펴보아야 하는 문제가 있다. 이러한 문제를 해결하기 위해 고정된 단계에서만 마스킹 값을 알고, 알고리즘이 끝난 다음에 마스킹 값을 벗겨내서 원래 값을 복원하는 방법이 제안되었다.

[0066] 기존 마스킹 방법의 가장 큰 단점인 랜덤한 S박스를 만들기 위해 수행되는 연산비용을 극복할 수 있는 방법이 제안되기도 하였다. 이 방법은 랜덤한 S박스를 다수 계산해서 ROM에 저장하는 방법 즉, 고정된 (Fixed) 마스킹 방법인데, S박스를 사용하는 모든 암호의 구현 시에 사용할 수 있는 방법이다.

[0067] 유니크 (Unique) 마스킹 방법은 블록암호를 DPA에 안전하도록 설계한 방법이다. 유니크 마스킹 방법은 블록암호에 들어가는 F함수와 같은 기능을 수행 하면서 마스킹 값의 존재 여부에 상관없이 수행할 수 있도록 F의 변형된 형태를 사용한다. 이 방법을 사용한 DES (Data Encryption Standard) 의 예는 다음과 같다.

[0068] DES에서는 F함수를 수학식 7의 두 S박스를 사용해서 구성한다.

수학식 7

[0069] $$\overline{S_1}(x)=S(x\oplus E(\alpha))$$

[0070] $$\overline{S_2}(x)=S(x)\oplus P^{-1}(\alpha)$$

[0071] 여기서, E는 확장 퍼뮤테이션 (expansion permutation)이고, P^{-1}는 S박스에서의 역 퍼뮤테이션 (inverse permutation)이다.

[0072] f_{K_i}는 DES에서 K_i라는 키를 사용해서 수행되는 f함수라고 하고, $\tilde{f}_{1,K_i}$와 $\tilde{f}_{2,K_i}$는 K_i라는 키와 $\tilde{S}_1(x)$와 $\tilde{S}_2(x)$를 S박스로 사용해서 구현되는 f함수라고 하자. 3개의 f중에서 $\tilde{f}_{1,K_i}$는 마스킹 값이 있는 데이터를 마스킹 값이 없는 데이터로 변환하고, $\tilde{f}_{2,K_i}$는 마스킹 값이 없는 데이터를 마스킹 값이 있는 데이터로 변환한다.

[0073] 도 3은 이와 같이 구성된 3 개의 f함수에 대해서 5 가지 형태로 DES의 라운드를 수행하는 예를 도시한 것이다.

[0074] 도 3에서 실선이 나타내는 것은 마스킹 값이 없는 데이터에 대한 연산의 흐름을 나타내는 것이고, 점선이 나타내는 것은 마스킹 값이 씌워진 데이터에 대한 흐름을 나타낸다. 도 3과 같이 구성된 5가지의 라운드 수행의 조합을 통해서 기존의 DES 수행과 같은 결과 값을 출력하기 위해서는 도 4에서 표기되어 있는 연산 순서를 따르는 조합으로 라운드 연산을 수행해야한다. 의미 있는 연산 수행의 예로는 BCDCDCEBCDCDCDCE를 들 수 있다.

[0075] 유니크 마스킹 방법은 DPA에 안전한 것으로 분석되고 있다. 그리고 첫 번째 라운드와 마지막 라운드가 같은 마스크 α 로 마스킹 되어있는 것에 의해서 발생할 수 있는 문제를 막기 위해서 수학식 8의 조합으로 수행하는 것을 제안하고 있다.

수학식 8

[0076] $$B_{\alpha 1}C_{\alpha 1}D_{\alpha 1}C_{\alpha 1}D_{\alpha 1}C_{\alpha 1}E_{\alpha 1}B_{\alpha 2}C_{\alpha 2}D_{\alpha 2}C_{\alpha 2}D_{\alpha 2}C_{\alpha 2}D_{\alpha 2}C_{\alpha 2}E_{\alpha 2}$$

[0077] 유니크 마스킹 방법은 DES의 연산구조를 그대로 사용할 수 있다는 장점이 있다.

[0078] 마스킹 방법은 블록 알고리즘에 따라 설계 방법이 달라질 수 있으며 그 설계 방법에 따라 마스킹 보정작업, 추가적인 테이블 크기가 결정된다. 또한 다른 블록 암호와 달리 SEED는 S박스와 법 2^{32} (mod 2^{32})에 대한 덧셈 연산을 비선형 연산으로 모두 사용하기 때문에 그 설계 방법에 따라 효율성이 크게 저하될 수 있다.

[0079] 본 발명의 일 실시 예에서는 일차 차분 전력 분석에 안전하면서 효율적인 SEED의 마스킹 방법을 제공한다.

[0080] 이하에서, $(x)^y$는 $\underbrace{x||x||....||x}_{y}$ 로서, x를 y번 연쇄(concatenation)시킨 것을 나타낸다. 또한, a<<b는 a에 대한 b비트 왼쪽 시프트 (left-shift) 연산을 나타내고, a>>b는 a에 대한 b비트 오른쪽 시프트 (right-shift) 연산을 나타낸다. 그리고, $+_n$은 법 2^n 에 대한 덧셈 연산을 나타내고, $-_n$은 법 2^n에 대한 뺄셈 연산을 나타낸다.

[0081] 본 발명의 일 실시 예에 따른 마스킹 방법의 전체 흐름은 다음과 같다.

[0082] 먼저, 두 개의 8 비트 마스킹 난수 m, m'을 생성 후, $m'' = m \boxplus m'$을 계산한다.

[0083] 다음, S박스 S_1, S_2에 대하여 두 개의 마스킹 S박스 MS_1, MS_2를 표 5의 알고리즘에 따라 생성한다. 다음, 4비트 난수 t_1, λ_{ab}, m_{ab}를 생성하고 FT, CT를 표 6의 알고리즘에 따라 생성한다. 또한 32비트 난수 γ를 생성한다.

[0084] 한편, 키 스케쥴링 과정에서 라운드 키 $K_{i,0}$을 $(m'')^4$로, $K_{i,1}$을 $(m')^4$로 마스킹 되도록 조정한다.

[0085] 다음, 입력 평문 16 바이트를 $(m')^8 \| (m)^8$으로 마스킹한다. 그리고, 두 번째 라운드부터 각 라운드 연산 전에 L_i를 $(m'')^8$과 XOR 연산하고, R_i를 $(m')^4 \| (0x00)^4$과 XOR 연산한다. 각 라운드 연산은 도 8의 본 발명의 일 실시 예에 따라 수정된 G함수와 표 6의 수정된 덧셈연산을 이용한다.

[0086] 마지막으로, 출력 평문의 16 바이트 마스킹 값 $(m'')^4 \| (m)^{12}$을 벗겨낸다.

[0087] 위의 순서에 따른 본 발명의 일 실시 예에 따른 SEED 마스킹 구조의 라운드 함수의 예를 도시하면 도 5 및 도 6과 같다.

[0088] 도 5는 마지막 라운드를 제외한 본 발명의 일 실시 예에 따른 라운드 함수를 도시한 것이다. 도 6은 본 발명의 일 실시 예에 따른 마지막 라운드 함수를 도시한 것이다.

[0089] 상술한 과정에서 최초 128비트의 평문이 $(m')^8\|(m)^8$로 마스킹된 후, 도 5 및 6에서 보는 바와 같이, 각 라운드의 128 비트 입력 값은 $(m')^8\|(m)^8$로 마스킹이 유지된다. 입력 값 128비트 중 $(m)^8$으로 마스킹 된 하위 64 비트는 본 발명의 일 실시 예에 따라 수정된 F함수의 입력 값으로 들어가며, $(m)^4\|(m'')^4$으로 마스킹 되어 출력된다.

[0090] 64비트의 F함수 출력 값은 라운드 입력의 상위 64비트 입력 값과 XOR연산을 수행한 후, 다음 라운드의 하위 64비트 입력 값이 된다. 두 값을 XOR 하였을 경우, 중간 값에 대한 마스킹 값은 $(m'')^4\|(m)^4(=(m')^8\oplus(m)^4\|(m'')^4)$로 변형된다.

[0091] 그리고, 도 5에서와 같이 $(m')^4\|(0x00)^4$과 XOR되면 다음 라운드의 하위 64비트 입력 값은 $(m)^8$로 마스킹 값이 변형된다. 또한 다음 라운드의 상위 64비트 입력 값은 이전 라운드의 하위 64비트의 값에 $(m'')^8$을 XOR한 값으로 $(m')^8$으로 마스킹 값이 변형된다. 즉, 다음 라운드 함수의 입력 값 128비트 또한 $(m')^8\|(m)^8$으로 선택되며, 이 마스킹 값은 본 발명의 일 실시 예에서 동일하게 유지된다.

[0092] 도 6은 마지막 라운드에 대한 마스킹 기법으로 마스킹을 제거하는 과정을 포함한다.

[0093] SEED의 비선형 연산은 S박스와 법 2^{32}에 대한 덧셈 연산이다. 이러한 비선형 연산에 대한 마스킹 방법은 그 비용이 선형 연산에 비해 상당히 큰 편이며 전체 마스킹 기법의 효율성을 결정한다.

[0094] 이하에서는 S박스에 대한 마스킹에 대해 설명한다. 마스킹 방법의 종류에는 여러 가지가 있지만 대체적으로 마스킹을 사용하는 알고리즘에서의 주의해야 할 점은 다음과 같다. 즉, 암호의 모든 연산과정에서 마스킹을 사용하여 알고리즘에서 사용되는 전력 값이 누출 되어도 그 전력 값은 랜덤이어야 한다. 그리고, 마스킹을 사용 했을 때 마스킹의 값에 의해서 본래 연산 되어야 되는 값에 영향을 미치더라도 본래 알고리즘의 결과 값과 상이한 값이 나와서는 안 된다.

[0095] 일반적으로 마스킹 방법을 사용할 경우 입력 값에 마스킹을 하여 전체적으로 마스킹 씌어 진 값들을 연산하고 출력 값의 마스킹을 벗겨내는 방법을 사용한다. 이러한 마스킹 방법에서, 선형적인 연산이 일어나는 부분에서는 마스킹 값이 그 다음 연산에서 쉽게 유추될 수 있지만 비선형 연산이 일어나는 부분에서는 그 다음의 연산에 대해 영향을 주게 되어 원래의 연산되어야 할 값과는 다른 값이 연산되는 점을 고려해야 한다.

[0096] 많은 블록암호에서 S박스를 사용하여 비선형 연산을 하며 이에 마스킹 방법을 적용할 경우 비선형 연산이 일어나는 S박스를 반드시 고려해야 한다.

[0097] DES 알고리즘을 예로 들어보자.

[0098] 표 3은 DES의 연산 과정 중 라운드 키의 6비트 와 평문의 6비트 값이 XOR 된 이후의 6비트 값이 S박스의 입력 값으로 들어가서 4비트의 값이 출력되는 형태를 나타낸 것이다.

표 3

[0099]

입력	$a_i = a_1, a_2, a_3, a_4, a_5, a_6$
S박스 연산	$\Downarrow$ $S_box(a_i)$ $Low: a_1, a_6$ and $Column: a_2, a_3, a_4, a_5$ of S_box $table$ $\Downarrow$
출력	$c_i = c_1, c_2, c_3, c_4$

[0100] DES 알고리즘의 입력 값을 평문에 마스킹 방법을 사용한 $a_i\oplus m_i$라 가정 하면 알고리즘을 수행 했을 때 출력 값이 $c_i\oplus m_i$의 형태로 되어 마스킹을 $c_i\oplus m_i\oplus m_i = c_i$ 연산으로 벗겨내어 올바른 형태의 c_i가 되어야 한다.

[0101] 하지만 DES 알고리즘에서는 S박스를 사용하는 비선형적인 연산이 포함 되어 있기 때문에 실질적으로 $c_i \oplus m_i$의 형태로 출력되지 않는다.

[0102] DES 알고리즘에서 입력 값을 $a_i \oplus m_i$ 로 수행했을 경우 S박스에서 일어나는 연산을 살펴보면 $a_i \oplus m_i$의 형태로 표 3과 같은 연산이 수행이 되며, 표 4와 같은 연산과정으로 나타낼 수 있다.

표 4

[0103]

입력	$a_i \oplus m_i = a_1 \oplus m_1,\ a_2 \oplus m_2,\ a_3 \oplus m_3,\ a_4 \oplus m_4,\ a_5 \oplus m_5,\ a_6 \oplus m_6$
S박스 연산	$\Downarrow$ $S_box(a_i \oplus m_i)$ $Low: a_1 \oplus m_1, a_6 \oplus m_6$ and $Column: a_2 \oplus m_2,\ a_3 \oplus m_3,\ a_4 \oplus m_4,\ a_5 \oplus m_5$ $of\ S_box\ table$
출력	$\Downarrow$ $c'_i = c'_1, c'_2, c'_3, c'_4$

[0104] 따라서 마스킹 방법이 사용된 이후의 S박스의 출력 값은 표 3에서와 같은 $Low: a_1, a_6\ Column: a_2, a_3, a_4, a_5$가 선택 되어서 출력되는 것이 아니라 [표2]에서의 $Low: a_1 \oplus m_1, a_6 \oplus m_6$, $Column: a_2 \oplus m_2, a_3 \oplus m_3, a_4 \oplus m_1, a_5 \oplus m_5$와 같이 마스킹이 씌워진 값이 선택 되어 출력된다.

[0105] 즉, $c_i \oplus m_i = c_1 \oplus m_1, c_2 \oplus m_2, c_3 \oplus m_3, c_4 \oplus m_4$이 아니라 $c_i' = c_1', c_2', c_3', c_4'$ 의 형태로 나오게 되어 원래의 값과 전혀 다른 값이 출력 되게 된다.

[0106] 따라서 수학식 9를 만족할 수 있는 S'박스를 알고리즘 수행 전에 반드시 생성해야만 한다.

수학식 9

[0107] $$S'-box(a_i \oplus m_i) = S-box(a_i) \oplus m_i = c_i \oplus m_i$$

[0108] SEED의 S박스에 대한 마스킹 S박스 생성 알고리즘의 예는 표 5와 같다.

표 5

[0109]

마스킹 S-box 구성 알고리즘
입력 S_1, S_2, m, m'
출력 MS_1, MS_2
1. For x from 0x00 to 0xff
  1.1. $MS_1(x \oplus m) = S_1(x) \oplus m'$
  1.2. $MS_2(x \oplus m) = S_2(x) \oplus m'$
2. Return MS_1, MS_2

[0110] 표 5의 알고리즘은 두 S박스 S_1, S_2에 대하여 m으로 마스킹 된 x값이 입력되었을 때, m'으로 마스킹 된 $S_1(x), S_2(x)$의 값을 출력하는 테이블을 생성한다. 이 마스킹 S박스 MS_1, MS_2는 F함수의 G함수에서 S_1, S_2 대신에 사용되며, 이 두 마스킹 S박스의 사용을 위해서 MS_1, MS_2의 입력 값은 반드시 m으로 마스킹 되도록 전체 구조

를 설계해야만 한다.

[0111] SEED의 비선형 연산인 덧셈 연산은 법 2^{32}에 대한 덧셈 연산을 사용한다. 따라서 본 발명의 일 실시 예에 따른 마스킹 기법은 법 2^{32}의 덧셈 연산의 마스킹 방법을 고려한다. 법 2^{32}의 덧셈 연산의 마스킹 변환 테이블을 사용할 경우 테이블의 크기가 너무 커서 실제로 사용할 수 없다. 이러한 문제는 작은 비트에 대한 마스킹 변환 테이블과 캐리 테이블을 이용하여 해결해야만 한다. 본 발명의 일 실시 예에 따른 마스킹 기법에서는 마스킹 변환 테이블과 캐리 테이블의 크기를 4비트 16개의 원소를 지니는 것으로 선택하여 사용한다. 이 값은 실험적으로 가장 효율적인 사이즈를 선택한 것이지만, 본 발명의 내용이 여기에 한정되지 않는다. 다음 알고리즘은 4비트에 대한 산술에서 불대수 마스킹으로 변환하는 테이블 FT와 캐리 테이블 CT의 생성 알고리즘이다.

표 6

[0112]

FT, CT 구성 알고리즘
입력 $m_{ba}, m_{ab}, t_1, t_2, \lambda_{ba}, \lambda_{ab}$
출력 *FT, CT*
1. For x from 0 to 15
  1.1. $FT(x+t_1) = x \oplus m_{ab}$
  1.2. $CT(x) = (x < t_1)\,?\,(\lambda_{ab}+1) : \lambda_{ab}$
2. Return *FT, CT*

[0113] FT 테이블은 4비트 난수 t_1, m_{ab}를 이용, 입력 값 $x+t_1 (\mathrm{mod}\, 2^4)$에 대해 $x \oplus m_{ab}$를 출력한다. 또한 CT 테이블은 $\tilde{x} = x + t_1$이 16보다 크거나 같을 경우, 즉, $\tilde{x}$이 t_1보다 작은 경우, 캐리 값을 설정하기 위한 것이다. 이 테이블은 캐리 값이 설정된 경우 1, 설정되지 않는 경우 0값을 얻기 위함이지만, 작은 값에 대해 캐리 값이 1, 큰 값에 대해 캐리 값이 0이 될 확률이 높은 취약점을 방지하기 위해 새로운 4비트 난수 λ를 이용하며, 실제 본 연산에서는 4비트의 난수 λ를 다시 빼주는 방법을 사용할 수도 있다.

[0114] 본 발명의 일 실시 예에 따라 수정된 F함수에서 입력 값은 $(m)^8$으로 마스킹 된 값이며, 출력 값은 $(m)^4 \| (m'')^4$으로 마스킹 되어 출력된다.

[0115] 도 7은 본 발명의 일 실시 예에 따라 수정된 F함수를 도시한 것이다.

[0116] 본 발명의 일 실시 예에 따라 수정된 F함수이 올바르게 동작하기 위해서는 다음의 두 조건을 만족하면 된다. 즉, 본 발명의 일 실시 예에 따라 수정된 G함수(Modified G-Function)은 $(m)^4$으로 마스킹된 입력 값에 대해 $(m'')^4$으로 마스킹 된 값을 출력한다. 그리고, 본 발명의 일 실시 예에 따른 마스킹 덧셈(MA, Masking Addition)은 두 입력 값 $X \oplus (m'')^4$, $Y \oplus (m'')^4$에 대해 $(X+Y) \oplus (m)^4$을 출력한다.

[0117] 도 8은 본 발명의 일 실시 예에 따라 수정된 G함수를 도시한 것이다.

[0118] 도 8의 32비트 입력 값은 $(m)^4$으로 마스킹되어 있으며, 마스킹 S박스 (MS_1, MS_2)를 거친 후, $(m')^4$으로 마스킹 된다. 마스킹 S박스 각각의 출력 값은 수학식 10의 연산을 통해 32비트 출력 값을 결정한다.

수학식 10

[0119]
$$Z_3 \oplus m'' = m \oplus (MS_2(X_3 \oplus m) \& m_2) \oplus (MS_1(X_2 \oplus m) \& m_1) \oplus (MS_1(X_1 \oplus m) \& m_0) \oplus (MS_1(X_2 \oplus m) \& m_3)$$

[0120] $$Z_2 \oplus m'' = m \oplus (MS_2(X_3 \oplus m)\&m_1) \oplus (MS_1(X_2 \oplus m)\&m_0) \oplus (MS_1(X_1 \oplus m)\&m_3) \oplus (MS_1(X_2 \oplus m)\&m_2)$$

[0121] $$Z_1 \oplus m'' = m \oplus (MS_2(X_3 \oplus m)\&m_0) \oplus (MS_1(X_2 \oplus m)\&m_3) \oplus (MS_1(X_1 \oplus m)\&m_2) \oplus (MS_1(X_2 \oplus m)\&m_1)$$

[0122] $$Z_0 \oplus m'' = m \oplus (MS_2(X_3 \oplus m)\&m_3) \oplus (MS_1(X_2 \oplus m)\&m_2) \oplus (MS_1(X_1 \oplus m)\&m_1) \oplus (MS_1(X_2 \oplus m)\&m_0)$$

[0123] 4바이트의 출력 값에서 각 바이트의 마스킹 값은 $m \oplus (m'\&m_0) \oplus (m'\&m_1) \oplus (m'\&m_2) \oplus (m'\&m_3)$ 이 되며 이 값을 정리하면 $m \oplus (m'\&(m_0 \oplus m_1 \oplus m_2 \oplus m_3))$ 이 된다. $m_0 \oplus m_1 \oplus m_2 \oplus m_3$ 은 G함수의 고정된 상수 값으로 0xff 값이다. 따라서, 본 발명의 일 실시 예에 따라 수정된 G함수의 출력 바이트에 대한 마스킹 값은 m'' 이 된다.

[0124] 본 발명의 일 실시 예에 따른 마스킹 방법은 F함수에서 사용되는 모든 마스킹 덧셈 연산의 32 비트 입력 값이 $(m'')^4$ 으로 마스킹하며, 덧셈 연산 후에 마스킹 값이 $(m)^4$ 이 되게 한다. 즉, 마스킹 덧셈 연산의 32 비트의 두 입력값 $X(=x \oplus (m'')^4)$, $Y(=y \oplus (m'')^4)$ 에 대하여 $Z(=(x+y) \oplus (m)^4)$ 을 출력하는 마스킹 덧셈 연산을 수행해야만 한다. 이때, 입출력 마스킹 값은 마스킹 방법의 효율성을 결정하는 마스킹 보정작업을 최소화하기 위해 선택된다.

[0125] 표 7은 이렇게 선택된 마스킹 값을 사용하여, 32 비트의 두 입력값 $X(=x \oplus (m'')^4)$, $Y(=y \oplus (m'')^4)$ 에 대하여 $Z(=(x+y) \oplus (m)^4)$ 을 출력하는 마스킹 덧셈 연산 알고리즘이다.

표 7

[0126]

마스킹 덧셈 연산

입력 $X(=x \oplus (m'')^4) = x_7x_6x_5x_4x_3x_2x_1x_0$

$Y(=y \oplus (m'')^4) = y_7y_6y_5y_4y_3y_2y_1y_0$

출력 $Z(=(x +_{32} y) \oplus (m)^4) = z_7z_6z_5z_4z_3z_2z_1z_0$

1. $temp_1 = convertBtoA(X)$
2. $Z = (t_1)^8 +_{32} temp_1$
3. $temp_1 = convertBtoA(Y)$
4. $Z = Z +_{32} temp_1 +_{32} ((m'')^4 \ll 1)$
5. $Z = convertAtoB(Z)$
6. Return Z

[0127] 표 7에서, $temp_1$은 임시 변수를 나타낸다. 단계 1과 단계 3의 $convertBtoA$ 함수는 불대수 마스킹을 산술 마스킹으로 변환시키는 함수로 표 8과 같은 과정에 따라 동작한다. $convertBtoA$ 함수를 통해 $x \oplus (m'')^4$ 는 $x - (m'')^4$ 로 변형되며, 4비트 난수 t_1 을 이용하여 단계 4의 Z값은 $Z = (t_1)^8 +_{32} (x - (m'')^4) +_{32} (y - (m'')^4) +_{32} 2(m'')^4 = (x+y) + (t_1)^8$ 이 된다. 단계 5의 $convertAtoB$ 함수는 산술 마스킹이 적용된 $(x+y)+(t_1)^8$ 값을 $(x+y) \oplus (m)^4$ 의 불대수 마스킹으로 변형시켜주는 함수로서 표 9와 같은 과정에 따라 동작한다.

[0128] 이와 같이, 다양한 연산이 수행되는 과정에서 불대수 마스킹과 산술 마스킹은 혼용 되어야 하는 경우에는 표 8 및 9와 같이 마스킹의 형태를 변환하는 방법이 수행되어야 한다. 또한, SEED에 마스킹 값을 그대로 유지 시켜

주기 위해서는 S박스 뒤에서, 그리고 덧셈 연산 뒤에서 각각 마스킹의 변환이 필요하다.

표 8

$convertBtoA$ 함수 입력 $X(=x\oplus(m'')^4),\ \gamma$ 출력 $temp_1(=x-_{32}(m'')^4)$ 1. $temp_1=(X\oplus\gamma-_{32}\gamma)\oplus X\oplus(((X\oplus\gamma)\oplus(m'')^4)-_{32}(\gamma\oplus(m'')^4))$ 2. Return $temp_1$

[0129]

[0130] 표 8에서, $temp_1$은 임시 변수를 나타낸다.

[0131] 산술에서 불대수로 마스킹을 변환하는 방법에서는 추가 연산비용이 많이 요구된다. 이러한 문제를 해결하기 위해 미리 계산된 테이블 (Pre-computation Table)을 사용할 수도 있다.

표 9

$convertAtoB$함수 입력 $\tilde{A}(=A+_{32}(t_1)^8)=a_7a_6a_5a_4a_3a_2a_1a_0$ 출력 $\tilde{B}(=A\oplus(m)^4)=b_7b_6b_5b_4b_3b_2b_1b_0$ 1. $b_0=FT(a_0)$ 2. $a=0$ 3. For k from 1 to 7 3.1. $a_7....a_k=a_7....a_k-_{(32-4k)}CT(a_{k-1})+_{(32-4k)}\lambda_{ab}$ 3.2. $b_k=FT(a_k)$ 4. $\tilde{B}=\tilde{B}\oplus(m)^4\oplus(m_{ab})^8$ 5. Return $\tilde{B}$

[0132]

[0133] 표 9에서는, 표 6에서 생성된 변환 테이블 FT와 캐리 테이블 CT가 사용된다. 표 9에서, $\tilde{A},\tilde{B}$ 는 마스킹에 의해 가려진 값을 나타낸다.

[0134] 이하에서는, 본 발명의 일 실시 예에 따른 마스킹 기법의 안전성을 증명하기 위해 메사지 (Messarges)가 제안한 전력 소비 모델 ($C(t)=offset+\varepsilon Hw(data)+N$ ($offset$,ε:상수, N:노이즈,$Hw(a)$:a의해밍웨이트)에 따라 일차 차분 전력 분석에 대한 결과를 시뮬레이션한 결과를 설명한다.

[0135] 오프셋이 10mA, ε이 3.72mA, 노이즈가 가우시안 (Gaussian) 정규분포를 따르며 그 분산 값이 6.7236 mA를 따른다고 가정했을 때, 마스킹 방법이 적용되지 않은 SEED는 상관계수가 0.613에 수렴하는 반면, 본 발명의 일 실시 예에 따른 마스킹 방법이 적용된 SEED는 상관계수가 0에 수렴함을 확인할 수 있다. 즉, 옳은 키에 대해 예측한 데이터의 중간 값과 전력사이의 상관성이 존재하지 않으며, 이는 본 발명의 일 실시 예에 따른 마스킹 방법이 일차 차분 전력 분석에 안전함을 의미한다.

[0136] 좀더 실질적인 안전성 확인을 위해, 1라운드 라운드 키인 $K_{1,0}$의 상위 한 바이트와 $K_{1,1}$의 상위 한 바이트를 추측하고 1라운드에서 처음 사용되는 G함수의 첫 S박스에 대한 일차 차분 전력 분석을 시행한 경우, 마스킹 기법이 적용되지 않은 SEED의 옳은 키에 대한 분석결과는 도 9와 같았고, 본 발명의 일 실시 예에 따른 마스킹 기법이 적용된 SEED의 옳은 키에 대한 분석결과는 도 10과 같았다.

[0137] 도 9에서 보는 바와 같이 마스킹되지 않은 SEED에 대한 일차 차분 전력 분석은 옳은 키에 대해 상관도 (correlation)가 피크를 보였으며 이는 성공적으로 키를 찾을 수 있음을 의미한다.

[0138] 반면, 도 10과 같이 마스킹된 SEED의 경우, 중간 연산 값이 모두 랜덤한 값을 지닌 상태로 연산되므로 옳은 키

에 대해서도 어떠한 연관성이 발견되지 않았다. 이는 본 발명의 일 실시 예에 따른 마스킹 기법이 일차 차분 전력 분석에 안전함을 의미한다.

[0139] 본 발명은 소프트웨어를 통해 실행될 수 있다. 바람직하게는, 본 발명의 일 실시 예에 따른 마스킹이 적용된 SEED를 이용한 암호화 방법을 컴퓨터에서 실행시키기 위한 프로그램을 컴퓨터로 읽을 수 있는 기록매체에 기록하여 제공할 수 있다. 소프트웨어로 실행될 때, 본 발명의 구성 수단들은 필요한 작업을 실행하는 코드 세그먼트들이다. 프로그램 또는 코드 세그먼트들은 프로세서 판독 가능 매체에 저장되거나 전송 매체 또는 통신망에서 반송파와 결합된 컴퓨터 데이터 신호에 의하여 전송될 수 있다.

[0140] 컴퓨터가 읽을 수 있는 기록매체는 컴퓨터 시스템에 의하여 읽혀질 수 있는 데이터가 저장되는 모든 종류의 기록 장치를 포함한다. 컴퓨터가 읽을 수 있는 기록 장치의 예로는 ROM, RAM, CD-ROM, DVD±ROM, DVD-RAM, 자기 테이프, 플로피 디스크, 하드 디스크(hard disk), 광데이터 저장장치 등이 있다. 또한, 컴퓨터가 읽을 수 있는 기록매체는 네트워크로 연결된 컴퓨터 장치에 분산되어 분산방식으로 컴퓨터가 읽을 수 있는 코드가 저장되고 실행될 수 있다.

[0141] 본 발명은 도면에 도시된 일 실시 예를 참고로 하여 설명하였으나 이는 예시적인 것에 불과하며 당해 분야에서 통상의 지식을 가진 자라면 이로부터 다양한 변형 및 실시 예의 변형이 가능하다는 점을 이해할 것이다. 그리고, 이와 같은 변형은 본 발명의 기술적 보호범위 내에 있다고 보아야 한다. 따라서, 본 발명의 진정한 기술적 보호범위는 첨부된 특허청구범위의 기술적 사상에 의해서 정해져야 할 것이다.

산업이용 가능성

[0142] 본 발명은 암호시스템의 안전성을 보장하기 위해 기존의 암호화 기법인 SEED 알고리즘에 마스킹을 도입하는 마스킹이 적용된 SEED를 이용한 암호화 방법에 관한 것으로, 전화카드, 이동통신 보안수단, 신분증, 교통카드, 금융 IC 카드, PDA 등 스마트 카드가 적용되는 시스템, 그 밖의 보안이 필요한 정보 시스템에 적용될 수 있다.

도면의 간단한 설명

[0143] 도 1은 종래의 SEED 알고리즘의 흐름을 나타낸 것이다.

[0144] 도 2는 F함수의 구조를 도시한 것이다.

[0145] 도 3은 변형된 형태의 f함수에 대해서 5 가지 형태로 DES의 라운드를 수행하는 예를 도시한 것이다.

[0146] 도 4는 도 3과 같이 구성된 5가지의 라운드 수행의 조합을 통해서 기존의 DES 수행과 같은 결과 값을 출력하기 위한 연산 순서의 조합을 나타낸다.

[0147] 도 5는 마지막 라운드를 제외한 본 발명의 일 실시 예에 따른 라운드 함수를 도시한 것이다.

[0148] 도 6은 본 발명의 일 실시 예에 따른 마지막 라운드 함수를 도시한 것이다.

[0149] 도 7은 본 발명의 일 실시 예에 따라 수정된 F함수의 예를 도시한 것이다.

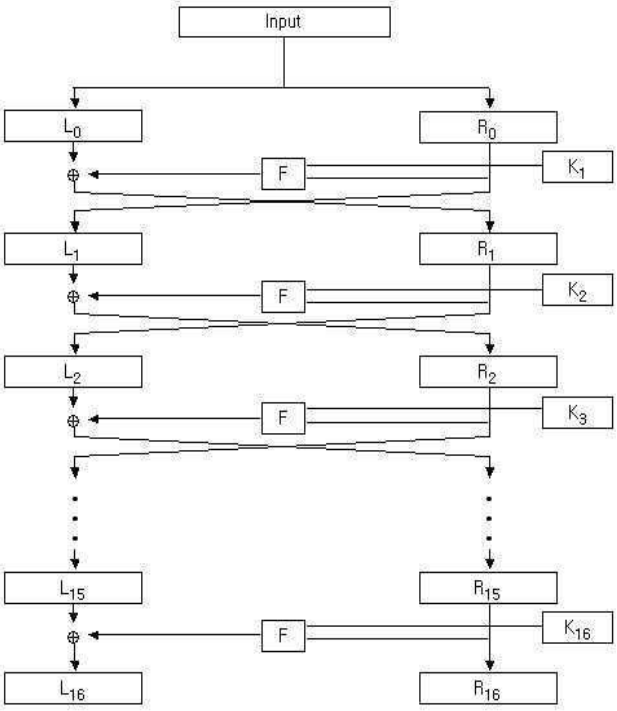
[0150] 도 8은 본 발명의 일 실시 예에 따라 수정된 G함수의 예를 도시한 것이다.

[0151] 도 9는 마스킹 기법이 적용되지 않은 SEED의 옳은 키에 대한 상관도 분석결과를 도시한 것이다.

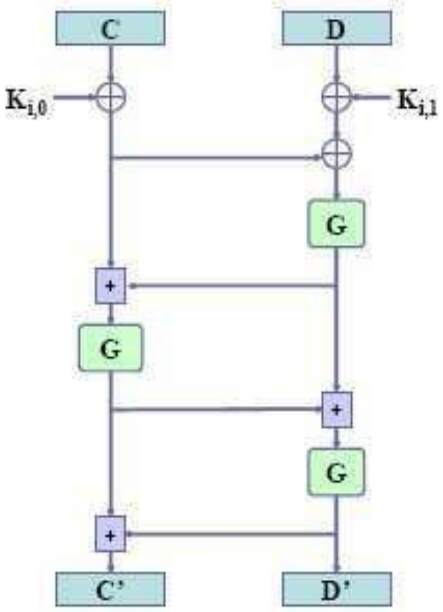
[0152] 도 10은 본 발명의 일 실시 예에 따른 마스킹 기법이 적용된 SEED의 옳은 키에 대한 상관도 분석결과를 도시한 것이다.

도면

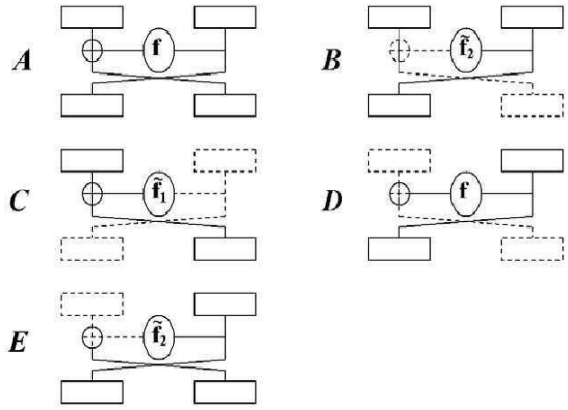
도면1


Input
L_0
R_0
F
K_1
L_1
R_1
F
K_2
L_2
R_2
F
K_3
L_{15}
R_{15}
F
K_{16}
L_{16}
R_{16}

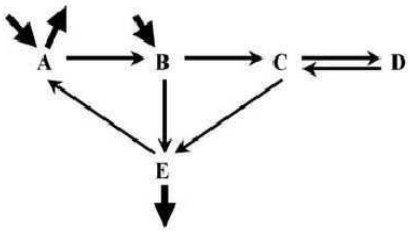
도면2


C
D
$K_{i,0}$
$K_{i,1}$
G
+
G
+
G
+
C'
D'

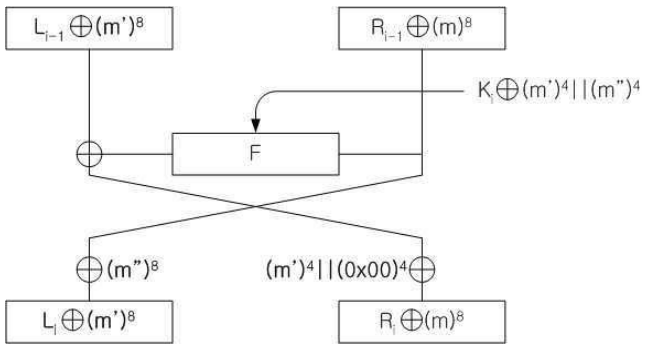
도면3



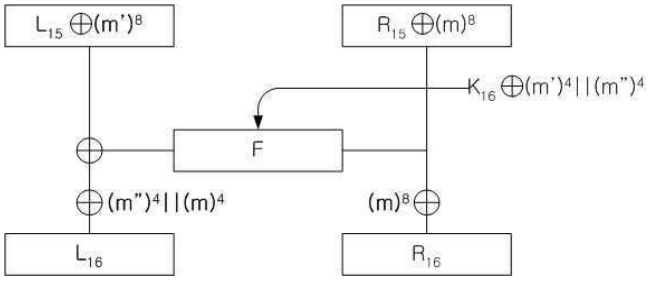
도면4



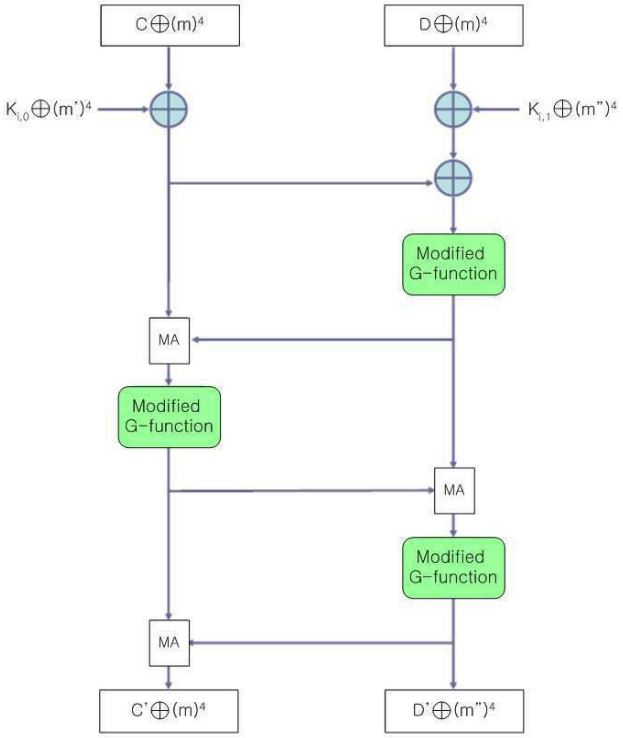
도면5



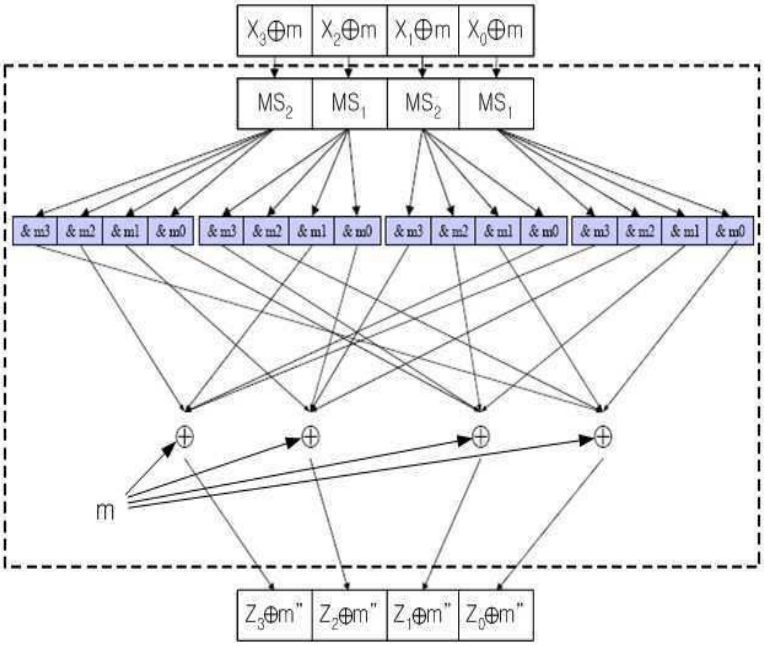
도면6



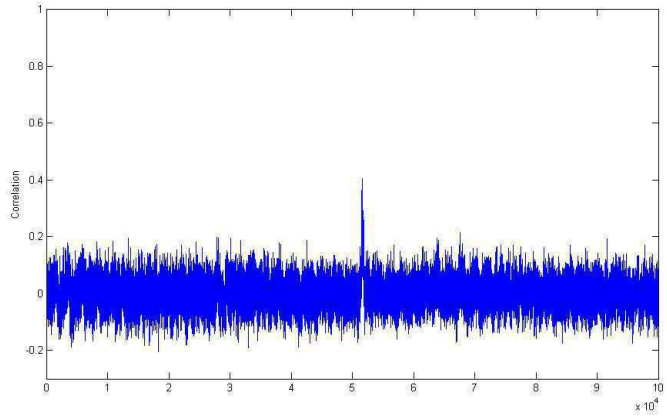
도면7


C⊕(m)⁴
D⊕(m)⁴
K_{i,0}⊕(m')⁴
K_{i,1}⊕(m'')⁴
Modified G-function
MA
Modified G-function
MA
Modified G-function
MA
C'⊕(m)⁴
D'⊕(m'')⁴

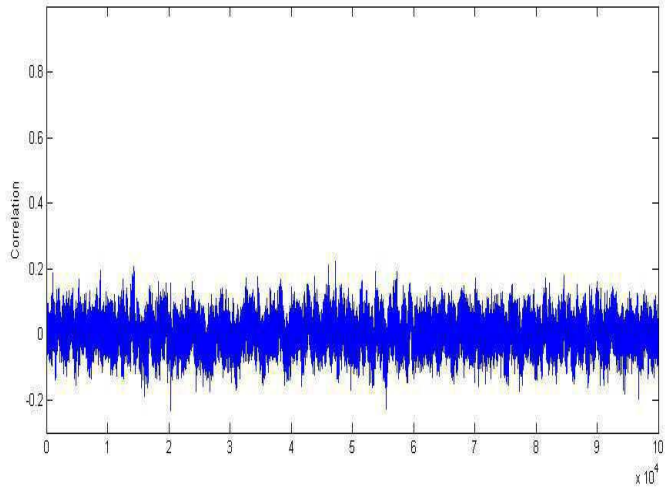
도면8


X3⊕m
X2⊕m
X1⊕m
X0⊕m
MS2
MS1
MS2
MS1
& m3
& m2
& m1
& m0
m
Z3⊕m''
Z2⊕m''
Z1⊕m''
Z0⊕m''

도면9


Correlation
1
0.8
0.6
0.4
0.2
0
-0.2
0
1
2
3
4
5
6
7
8
9
10
x 10^4

도면10


Correlation
0.8
0.6
0.4
0.2
0
-0.2
0
1
2
3
4
5
6
7
8
9
10
x 10^4