블록 암호 알고리즘에서 선형변환 행렬식의 MDS 코드 생성 확인 알고리즘

윤성훈*, 박창수*, 조경연*
*부경대학교 전자컴퓨터정보통신공학부
e-mail : tankk1@hanmail.net

MDS code Confirmation Algorithms of Linear Transformation Matrix in Block Cipher Algorithms

Seong-Hoon Youn*, Chang-Soo Park*, Gyeong-yeon Cho*
*Dept. of Electric, Computer, Telecommunication, Pukyong National University

요 약

정보통신의 발달과 인터넷의 확산으로 인해 정보보안의 필요성이 중요한 문제로 대두되면서 여러 종류의 암호 알고리즘이 개발되어 활용되고 있다. Substitution Permutation Networks(SPN) 등의 블록 암호 알고리즘에서는 확산선형변환 행렬을 사용하여 안전성을 높이고 있다. 확산선형변환 행렬이 Maximum Distance Separable(MDS) 코드를 생성하면 선형 공격과 차분 공격에 강한 특성을 보인다.

본 논문에서는 선형변환 행렬이 MDS 코드를 생성하는 가를 판단하는 새로운 알고리즘을 제안한다. 입력 코드는 $GF(2^n)$상의 원소들로 구성되며, 원소를 변수로 해석하여, 변수를 소거시키면서 선형변환행렬이 MDS 코드를 생성하는 가를 판단한다. 본 논문에서 제안한 알고리즘은 종래의 모든 정방 부분행렬이 정칙인가를 판단하는 알고리즘과 비교하여 연산 수행 시간을 크게 줄였다.

1. 서론

정보통신의 발달과 인터넷의 확산으로 인해 정보보안의 필요성이 중요한 문제가 되고 있다. 미국, 유럽의 국가들은 독자적으로 암호 알고리즘에 대한 연구를 하고 있으며, 암호기술 및 제품의 수출을 규제하고 있다. 이에 우리나라도 한국정보보호센터를 주축으로 연구하여 128 비트 블록암호 알고리즘인 SEED[1]를 개발하여 공개하였다.

암호화 기술이 발달하자 그에 따라서 암호문을 해독하기위한 기술도 같이 발달하게 되었다. 그래서 현재는 안전성이 더욱 강화된 암호 알고리즘이 요구되고 있다. 요즈음 많이 사용되고 있는 암호 알고리즘은 블록암호 알고리즘이다. 블록암호 알고리즘의 대표적인 공격 방법으로는 선형 공격(linear attack)[2]과 차분 공격(differential attack)[3]이다. Heys 와 Tavares는 확산선형변환(diffusive linear transformation)을 이용해서 Substitution Permutation Networks(SPN)의 치환부분(permutation layer)을 교체하여, 암호문의 퍼져나가

는 특성이 개선되고, 선형과 차분 공격에 대한 저항성이 증가되는 것을 보였다[4][5].

선형변환을 이용하면 선형과 차분 공격에 강하게 할 수 있는데, 선형변환의 여러 방법 중에서 관심이 있는 것은 Maximum Distance Separable(MDS) 코드[6]에 기반 한 선형변환이다. 이러한 선형변환은 Vaudenay[7]이 제안하였고, SHARK[8] 및 SQUARE[9]에 적용되었다.

본 논문에서는 블록 암호 알고리즘이 선형과 차분 공격에 강한 특성을 가지기 위하여 선형변환행렬을 사용하였을 때 MDS 코드를 생성하는지 판단하는 알고리즘을 제안한다. 선형변환의 입력코드는 □□(2□)상의 원소들로 구성하고, 그 원소들을 변수로 해석하여, 변수를 소거 시키면서 선형변환행렬이 MDS 코드를 생성하는지 판단하는 알고리즘이다. 기존의 알고리즘인 선형변환행렬의 모든 정방부분행렬이 정칙인지를 확인하는 것과 비교하였을 때 연산 수를 크게 감소시켜, 연산 수행 시간을 많이 줄인 것으로 앞으로 많이 활용할 수 있을 것으로 기대된다.

본 논문의 구성은 2장에서 본 논문에서 제안한 알고리즘을 설명하고, 3장에서 두 알고리즘의 성능을 비교하고, 4장에서 결론을 맺는다.

2. MDS 코드 생성 검사 알고리즘

블록 암호 알고리즘에서 확산층은 □□(2□)상의 ⋮개의 □-bit 코드 □를 ⋮개의 □-bit 코드 □로 선형변환한다. 선형변환행렬을 □라 하면 식(1)로 표현된다.

$$\begin{bmatrix} Y_0 \\ Y_1 \\ \cdots \\ Y_{M-1} \end{bmatrix} = \begin{bmatrix} A_{0,0} & \cdots & A_{0,M-1} \\ A_{1,0} & \cdots & A_{1,M-1} \\ \cdots & \cdots & \cdots \\ A_{M-1,0} & \cdots & A_{M-1,M-1} \end{bmatrix} \times \begin{bmatrix} X_0 \\ X_1 \\ \cdots \\ X_{M-1} \end{bmatrix} \quad (1)$$

□ₕ(□)를 코드 □의 해밍가중치라고 하면, 선형변환행렬 □의 branch number ß 는 식(2)로 주어진다.

$$ß = \mathrm{Min}\,(□_h(□) + □_h(□)) \quad \text{while } □ \neq 0 \quad (2)$$

ß 가 ⋮+ 1 이 되는 선형변환행렬 □는 MDS 코드를 생성한다고 한다. 행렬 □가 MDS 코드를 생성하는 가를 판단하기 위해서는 모든 □에 대하여 식(2)를 산출해야 한다. 이를 위해서 □의 원소 □□(0 □ ⋮ 1)을 변수로 취급하여 이들을 소거하면서 연산하는 방법을 제안한다.

Lemma 1 : 선형변환행렬 □의 모든 원소가 '0'이 아니면 '□ₕ(□) = 1'인 경우에 MDS 코드를 생성한다.

증명 : □□(0 □ ⋮ 1) 하나만이 0 이 아니고 이를 제외한 나머지 □의 원소가 0 인 경우에 식(1)은 식(3)과 같이 된다.

$$□_0 = □_{0,□} \times □_□$$
$$\cdots\cdots\cdots\cdots \quad (3)$$
$$□_{⋮-1} = □_{⋮-1,□} \times □_□$$

식(3)에서 '□ₕ(□) = 1'이므로 '□ₕ(□) = ⋮'이어야 ß 는 ⋮+ 1 이 된다. □□ 0 라고 가정하였으므로 선형변환행렬 □의 모든 원소는 '0'이 아니어야 한다.

Lemma 2 : '□ₕ(□) = 2'이면 '□ₕ(□) = ⋮ 1'인 경우에 MDS 코드를 생성한다. □₀와 □□ 두 개의 원소가 '0'이 아니라면, □ₓ(0 ≺ ⋮ 2) 원소를 '0'으로 가정하여 □□를 □₀의 함수로 표현하고, 이것을 □□(≺ < □ ⋮ 1)에 대입하여 '□□ 0'가 되는 선형변환은 MDS 코드를 생성한다.

증명 : □□와 □□ 두 개의 원소만이 0 이 아니고, 다른 모든 원소가 0 인 경우에 식(1)은 식(4)가 된다.

$$□_0 = (□_{0,□} \times □_□) + (□_{0,□} \times □_□)$$
$$\cdots\cdots\cdots\cdots\cdots\cdots \quad (4)$$
$$□_{⋮-1} = (□_{⋮-1,□} \times □_□) + (□_{⋮-1,□} \times □_□)$$

식(4)에서 '□ₕ(□) = 2'이므로 '□ₕ(□) = ⋮ 1'이 되어야 한다. □₀= 0 이라고 가정하면

$$□_□ = (□_{0,□} \times □_□) / □_{0,□}$$

이 된다. □□를 소거하기 위하여 □□를 □₁ ⋯ □⋮-1에 대입하면 식(5)가 된다.

$$□_1 = (□_{1,□} - (□_{1,□} \times □_{0,□}) / □_{0,□}) \times □_□$$
$$□_2 = (□_{2,□} - (□_{2,□} \times □_{0,□}) / □_{0,□}) \times □_□$$
$$\cdots\cdots\cdots\cdots\cdots\cdots \quad (5)$$
$$□_{⋮-1} = (□_{⋮-1,□} - (□_{⋮-1,□} \times □_{0,□}) / □_{0,□}) \times □_□$$

식(5)에서 □□ 0 이므로, (...)의 값이 '0'이 아니면 □₁ ⋯ □⋮-1은 0 이 아니다. 또한, 식(4)에서 □₁= 0 이라고 가정하면, 앞에서 □₀= 0 이면서 □₁= 0 이 되는 경우가 없음을 확인하였으므로, □₀ 0 이다. 이 때 □₀ = (□₁,□ × □□) / □₁,□가 되며, 이를 식(4)에 대입하면 식(6)이된다.

□₂ = (□₂,□ − (□₂,□ × □₁,□) / □₁,□) × □₀

····· (6)

□ᵢ:₋₁ = (□ᵢ:₋₁,□ − (□ᵢ:₋₁,□ × □₁,□) / □₁,□) × □₀

식(6)에서 □₂ ··· □ᵢ:₋₁가 모두 0이 아니면 ß = ⋮ + 1 이 된다.

Lemma 3 : '□ₕ(□) = 3'이면 '□ₕ(□) = ⋮ 2'인 경우에 MDS 코드를 생성한다. □₀, □₀와 □₀ 세 개의 원소만이 '0'이 아니라면, □ₓ(0 ≺ ⋮ 3) 원소를 '0'으로 가정하여 □₀를 □₀와 □₀의 함수로 표현하고, 이것을 □₀(≺ < □ ⋮ 2)에 대입하여 □₀를 소거하고, □₀ = 0 으로 가정하여 □₀를 □₀의 함수로 표현하고, 이것을 □₋(□ < ⌣ ⋮ 1)에 대입하여 '□₋ 0'가 되는 선형변환행렬은 MDS 코드를 생성한다.

증명 : □₀, □₀, □₀ 세 개의 원소만이 0이 아니고, 다른 모든 원소가 0인 경우에 식(1)은 식(7)이 된다.

□₀ = (□₀,□ × □₀) + (□₀,□ × □₀) + (□₀,□ × □₀)

□₁ = (□₁,□ × □₀) + (□₁,j × □₀) + (□₁,□ × □₀)

····· (7)

□ᵢ:₋₁ = (□ᵢ:₋₁,□ × □₀) + (□ᵢ:₋₁,□ × □₀) + (□ᵢ:₋₁,□ × □₀)

식(7)에서 '□ₕ(□) = 3'이므로 '□ₕ(□) = ⋮ 2'이 되어야 한다. □₀ = 0 이라고 가정하면

□₀ = ((□₀,□ × □₀) / □₀,□) − ((□₀,□ × □₀) / □₀,□)

이 된다. 이것을 식(7)에 대입하여 □₀를 소거하면 식(8)이 된다.

□₁ = (□₁,□ − ((□₀,□ × □₁,□) / □₀,□) × □₀) +
(□₁,□ − ((□₀,□ × □₁,□) / □₀,□) × □₀

□₂ = (□₂,□ − ((□₀,□ × □₂,□) / □₀,□) × □₀) +
(□₂,□ − ((□₀,□ × □₂,□) / □₀,□) × □₀ (8)

····· (8)

□ᵢ:₋₁ = (□ᵢ:₋₁,□ − ((□₀,□ × □ᵢ:₋₁,□) / □₀,□) × □₀) +
(□ᵢ:₋₁,□ − ((□₀,□ × □ᵢ:₋₁,□) / □₀,□) × □₀

식(8)을 간소화 하여 식(9)로 표현할 수 있다.

□₁ = (□ ₁,□ × □₀) + (□ ₁,□ × □₀)

□₂ = (□ ₁,□ × □₀) + (□ ₁,□ × □₀)

····· (9)

□ᵢ:₋₁ = (□ ᵢ:₋₁,□ × □₀) + (□ ᵢ:₋₁,□ × □₀)

식(9)에서 □₁ = 0 이라고 가정하면

□₀ = (□ ₁,□ × □₀) / □ ₁,□

이 된다. 이것을 식(9)에 대입하여 □₀를 소거하면 식(10)이 된다.

□₂ = (□ ₂,□ (□ ₂,□ × □ ₁,□) / □ ₁,□) × □₀

····· (10)

□ᵢ:₋₁ = (□ ᵢ:₋₁,□ (□ ᵢ:₋₁,□ × □ ₁,□) / □ ₁,□) × □₀

식(10)에서 □₂ ··· □ᵢ:₋₁ 이 모두 '0'이 아니면 '□ₕ(□) = ⋮ 2'를 만족한다. 또한, 식(9)에서 □₂ = 0 이라고 가정하면, 앞에서 □₁ = 0 이면서 Y_2 = 0 이 되는 경우가 없음을 확인하였으므로, □₁ 0 이다. 이 때

□₀ = (□ ₂,□ × □₀) / □ ₂,□가 되며, 이를 식(9)에 대입하여 변수 □₀를 소거하여 □₃ ··· □ᵢ:₋₁이 모두 '0'이 아니면 '□ₕ(□) = ⋮ 2'를 만족한다.

Lemma 4 : '□ₕ(□) = ∓ (4 ∓ ⋮)'이면 '□ₕ(□) = ⋮ + ∓ + 1'인 경우에 MDS 코드를 생성한다. □₀, ···, □∓까지 ∓개의 원소가 '0'이 아니라면, □ₓ(0 ≺ ⋮ ∓) 원소를 '0'으로 가정하여 □∓을 □₀, ···, □∓₋₁ 의 함수로 표현하고 이것을 □₀(≺ < □ ⋮ ∓+1)에 대입하여 □∓을 소거하고, 이러한 과정을 순차적으로 반복하여 □₋(⌣ ⋮ 1)을 □₀만의 함수로 표현해서 '□₋ 0'가 되는 선형변환행렬은 MDS 코드를 생성한다.

증명 : ***lemma 3*** 의 증명을 확장하는 것으로 증명할 수 있다.

3. 구현 및 비교 검토

기존의 알고리즘은 선형변환행렬의 모든 정방부분행렬이 정칙인지를 판단하는 것이다. 이것은 정칙인지를 확인하기 위하여 모든 정방부분행렬의 역행렬을 구해서 MDS 코드를 생성하는지 판단한다.

본 논문에서 제안하는 알고리즘은 반복적 연산을 필요로 하기 때문에 재귀호출 프로그램으로 작성하였다.

두 알고리즘을 Visual C 를 사용하여 구현하고, 수행시켜서 연산 수를 구하여 표 1-1 과 표 1-2 에 나타내었다.

표 1-1 정칙인지 확인하는 알고리즘의 연산 수

	8 × 8	12 × 12	16 × 16
Multiply	312,312	21.5×10^7	11.1×10^{10}
Reciprocal	38,611	13.5×10^6	42.1×10^8

표 1-2 변수를 소거하는 알고리즘의 연산 수

	8 × 8	12 × 12	16 × 16
Multiply	111,698	30.4×10^6	73.6×10^8
Reciprocal	11,185	2.5×10^6	56.6×10^7

두 알고리즘의 연산 수를 그림 1-1 에 그래프로 나타내었다.

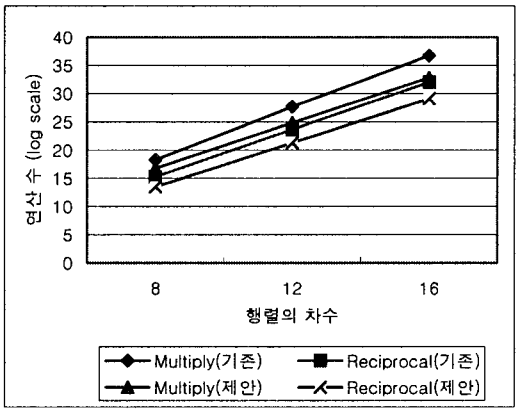

그림 1-1 연산 수 표시 그래프

두 알고리즘의 연산 수를 비교하여 보면 행렬의 차수가 커질수록 본 논문에서 제안한 알고리즘의 연산 수가 큰 폭으로 감소하는 것을 알 수 있다. 따라서 연산 수행 속도가 빨라지고, 연산 수행 시간을 많이 단축할 수 있다.

4. 결 론

현재 많이 사용하는 알고리즘은 블록 암호 알고리즘인데, 이 알고리즘에 대한 가장 강한 공격이 선형과 차분 공격이다. 이 공격들에 강한 특성을 지니기 위해서 선형변환을 이용한다. 이 때 선형변환행렬이 MDS 코드를 생성하면 선형과 차분 공격에 강한 특성을 가진다.

본 논문에서는 선형변환행렬이 MDS 코드를 생성하는지 판단하는 알고리즘을 제안한다. 선형변환의 입력 코드는 □□(2^n)상의 원소들로 구성되는데, 원소들을 변수로 해석하여 소거 시키면서 연산을 수행하여 선형변환행렬이 MDS 코드를 생성하는지 확인한다. 제안하는 알고리즘은 종전의 모든 정방부분행렬이 정칙인 것을 확인하는 알고리즘과 비교하여 연산 수를 크게 감소시켰다.

향후 안전성이 강화된 블록 암호 알고리즘을 개발하는데 본 논문에서 제안한 알고리즘이 많이 활용될 수 있을 것이다.

[참고문헌]

[1] 한국정보보호센터, 128 비트 블록 암호 알고리즘(SEED) 개발 및 분석 보고서, Dec. 1998

[2] E. Biham and A. Shamir, "Differential cryptanalysis of DES-like crytosystems", Journal of Cryptology, vol. 4, no. 1, pp. 3-72, 1991

[3] M.Matsui, "The first experimental cryptanalysis of the Data Encryption Stan dard", Advances in Cryptology: Proc. Of EUROCRYPT '91, Springer-Verlag, Berlin, pp. 1-11, 1994

[4] H.M. Heys and S.E. Tavares, "The design of substitution-permutation networks resistant to differential and linear cryptanalysis", Proceedings of 2nd ACM Conference on Computer and Communications Security, Fairfax, Virginia, pp. 148-155, 1994

[5] H.M. Heys and S.E. Tavares, "The design of product ciphers resistant to differential and linear cryptanalysis", Journal of Cryptology, Vol. 9, no. 1, pp. 1-19, 1996

[6] F.J. MacWilliams and N.J.A. Sloane, "The theory of error correcting codes", North-Holland Publishing Company, 1977

[7] S. Vaudenay, "On the need for multipermutations: Crytanalysis of MD4 and SAFER," Proc. of Fast Software Encryption (2), LNCS 1008, Springer-Verlag, pp. 286-297, 1995

[8] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. De Win, "The cipher SHARK", Fast Software Encryption, LNCS 1039, D. Gollmann, Ed., Springer-Verlag, pp. 99-112, 1996

[9] J. Daemen, L. Knudsen, and V. Rijmen, "The block cipher SQUARE", Proc. of Fast Software Encryption (4), LNCS, Springer-Verlag, 1997