SEED 병렬 구현

송민호

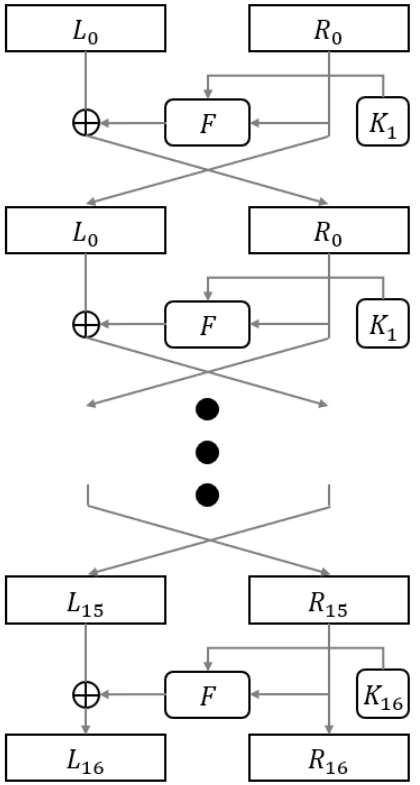
유튜브: https://youtu.be/K3JUSEMl9mg

한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB

SEED

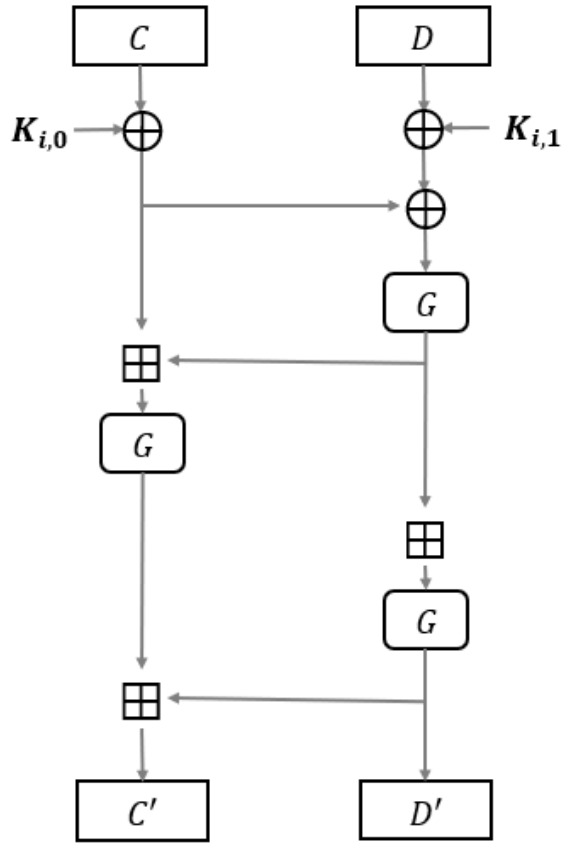
- SEED
  - 1999년 KISA에서 개발한 Feistel 구조의 블록 암호
    - ISO/IEC 표준 블록 암호 선정
  - 민감한 정보의 보호와 개인 프라이버시 등을 보호하기 위해 개발됨
  - 128 비트의 평문 블록과 128비트 키를 입력으로 사용
    - 16라운드를 거쳐 128비트 암호문 블록 출력



SEED

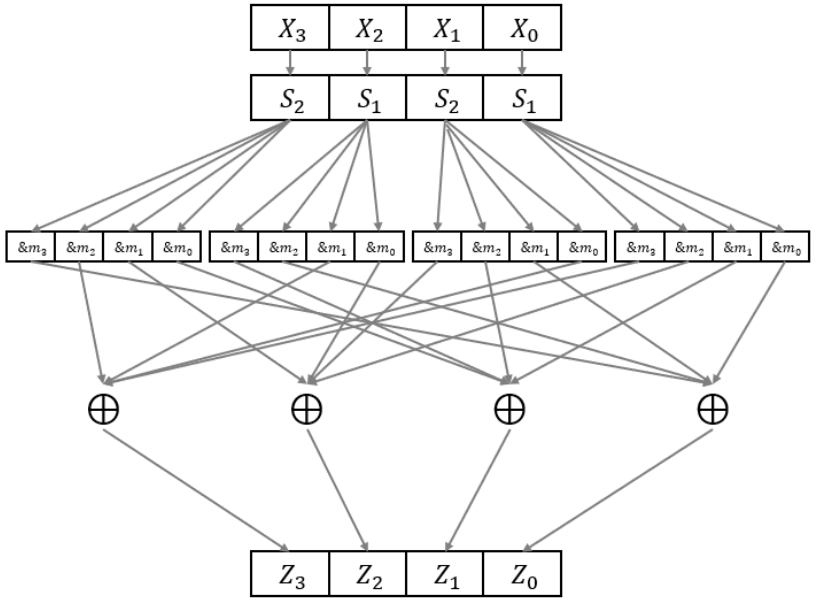
- F 함수
  - Feistel 구조를 갖는 블록 암호알고리즘은 F 함수의 특성에 따라 구분됨
  - SEED의 F 함수는 수정된 64비트 Feistel 형태로 구성
  - 각 32비트 블록 2개(C, D)를 입력으로 받아 32비트 블록 2개(C', D')를 출력
    - 암호화 과정에서 64비트 블록과 64비트 라운드 키 $K_i = (K_{i,0}; K_{i,1})$를 F 함수의 입력으로 처리하여 64비트 블록 출력

$$C' = G[G[G\{(C \oplus K_{i,0}) \oplus (D \oplus K_{i,1})\} \boxplus (C \oplus K_{i,0})] \boxplus G\{(C \oplus K_{i,0}) \oplus (D \oplus K_{i,1})\}]$$
$$\boxplus G[G\{(C \oplus K_{i,0}) \oplus (D \oplus K_{i,1})\} \boxplus (C \oplus K_{i,0})]$$
$$D' = G[G[G\{(C \oplus K_{i,0}) \oplus (D \oplus K_{i,1})\} \boxplus (C \oplus K_{i,0})] \boxplus G\{(C \oplus K_{i,0}) \oplus (D \oplus K_{i,1})\}]$$



SEED

- G 함수
  - F 함수는 G 함수에 따라 특정됨
  - 비선형 Sbox S1, S2 사용
  - 32비트 블록을 8비트 4개 블록으로 나누어 Sbox 변환 등을 거쳐 8비트 4개 블록 출력



$$Y_3 = S_2(X_3),\quad Y_2 = S_1(X_2),\quad Y_1 = S_2(X_1),\quad Y_0 = S_1(X_0),$$

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

$$(m_0 = 0\text{xfc},\quad m_1 = 0\text{xf3},\quad m_2 = 0\text{xcf},\quad m_3 = 0\text{x3f})$$

ARMv8

- ARMv8은 ARM의 64비트 임베디드 아키텍처
  - 32비트 모드(AArch32)와 64비트 모드(AArch64) 지원
  - 31개의 범용 레지스터
    - x0-x30는 64비트 유닛에 사용, w0-w30는 32비트 유닛에 사용
  - 32개의 128비트 벡터 레지스터 포함(v0-v31)
    - 병렬 구현에 사용되는 명령어

Asm	Operands	Description	Operation
EOR	V_d, V_n, V_m	Bitwise exclusive OR	$V_d \leftarrow V_n \oplus V_m$
SUB	V_d, V_n, V_m	Subtract	$V_d \leftarrow V_n - V_m$
TBL	V_d, V_n, V_m	Table vector lookup	$V_d \leftarrow V_n [V_m]$
TBX	V_d, V_n, V_m	Table vector lookup extension	$V_d \leftarrow V_n [V_m]$
TRN	V_d, V_n, V_m	Transpose vectors	$V_d \leftarrow V_n [V_m]$

구현 방법

- 벡터 레지스터 정렬
  - TBL 명령어 사용
  - (a)와 같이 8개의 레지스터에 평문을 로드
    - 각 레지스터에 서로 다른 Sbox 사용
  - (b)의 레지스터에서는 모든 인덱스가 같은 Sbox를 사용
    - Sbox 연산구현을 위해 TBL 명령어 사용 가능

S1: S2:

V0	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15
V1	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15
V2	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
V3	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
V4	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15
V5	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
V6	G0	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12	G13	G14	G15
V7	H0	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12	H13	H14	H15

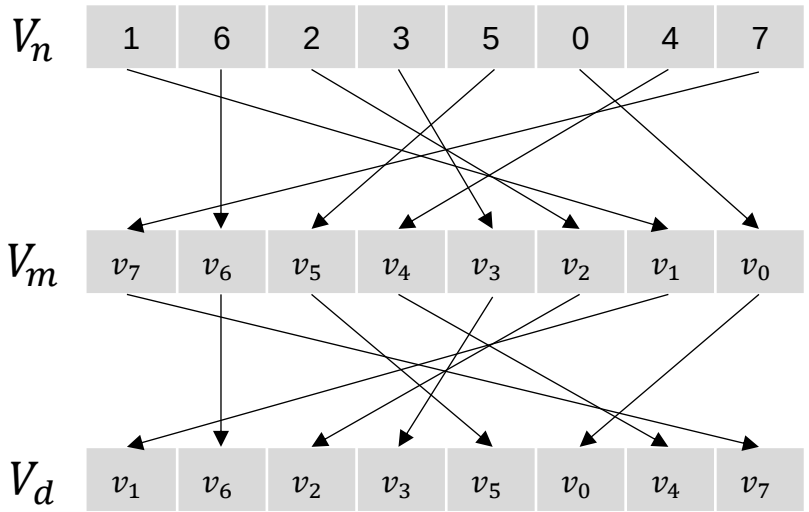
(a)

V0	A0	A2	B0	B2	C0	C2	D0	D2	E0	E2	F0	F2	G0	G2	H0	H2
V1	A1	A3	B1	B3	C1	C3	D1	D3	E1	E3	F1	F3	G1	G3	H1	H3
V2	A4	A6	B4	B6	C4	C6	D4	D6	E4	E6	F4	F6	G4	G6	H4	H6
V3	A5	A7	B5	B7	C5	C7	D5	D7	E5	E7	F5	F7	G5	G7	H5	H7
V4	A8	A9	B8	B9	C8	C9	D8	D9	E8	E9	F8	F9	G8	G9	H8	H9
V5	A10	A11	B10	B11	C10	C11	D10	D11	E10	E11	F10	F11	G10	G11	H10	H11
V6	A12	A13	B12	B13	C12	C13	D12	D13	E12	E13	F12	F13	G12	G13	H12	H13
V7	A14	A15	B14	B15	C14	C15	D14	D15	E14	E15	F14	F15	G14	G15	H14	H15

(b)

구현 방법

- TBL 명령어를 사용하면 Sbox 연산을 효율적으로 구현할 수 있음
  - TBL 명령어는 테이블 벡터 룩업을 불러옴
  - V_n 레지스터(입력 벡터)에 들어 있는 벡터 값을 읽어서 그 값들은 V_m 레지스터(룩업 테이블 저장되있음)의 인덱스로 사용됨
  - V_m 레지스터의 해당 인덱스에 저장된 값은 V_d 레지스터에 저장됨
    - V_d 레지스터에 저장된 위치는 V_n 레지스터에서 값을 읽을 때의 인덱스
  - 매 라운드마다 Sbox 로드를 안해도됨
    - 2개의 Sbox



구현 방법

- 레지스터 초기화 없이 G Function 구현 가능
  - 레지스터에 계속 로드하고 저장하는 과정은 성능 저하를 일으킴
  - 정렬된 레지스터에 맞춰 m0-m3 정렬

$$Y_3 = S_2(X_3),\quad Y_2 = S_1(X_2),\quad Y_1 = S_2(X_1),\quad Y_0 = S_1(X_0),$$

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

$$(m_0 = 0\text{xfc},\quad m_1 = 0\text{xf3},\quad m_2 = 0\text{xcf},\quad m_3 = 0\text{x3f})$$

V0	A0	A2	B0	B2	C0	C2	D0	D2	E0	E2	F0	F2	G0	G2	H0	H2
V1	A1	A3	B1	B3	C1	C3	D1	D3	E1	E3	F1	F3	G1	G3	H1	H3
V2	A4	A6	B4	B6	C4	C6	D4	D6	E4	E6	F4	F6	G4	G6	H4	H6
V3	A5	A7	B5	B7	C5	C7	D5	D7	E5	E7	F5	F7	G5	G7	H5	H7
V4	A8	A9	B8	B9	C8	C9	D8	D9	E8	E9	F8	F9	G8	G9	H8	H9
V5	A10	A11	B10	B11	C10	C11	D10	D11	E10	E11	F10	F11	G10	G11	H10	H11
V6	A12	A13	B12	B13	C12	C13	D12	D13	E12	E13	F12	F13	G12	G13	H12	H13
V7	A14	A15	B14	B15	C14	C15	D14	D15	E14	E15	F14	F15	G14	G15	H14	H15

V8	m0	m2	m0	m2	m0	m2	m0	m2	m0	m2	m0	m2	m0	m2	m0	m2
V9	m1	m3	m1	m3	m1	m3	m1	m3	m1	m3	m1	m3	m1	m3	m1	m3
V10	m2	m0	m2	m0	m2	m0	m2	m0	m2	m0	m2	m0	m2	m0	m2	m0
V11	m3	m1	m3	m1	m3	m1	m3	m1	m3	m1	m3	m1	m3	m1	m3	m1

Q & A