F A C E - L I G H T

FACE–LIGHT: Fast AES CTR Mode Encryption for Low-end Microcontrollers

국가암호공모전

CONTENTS

01

Introduction

F A C E - L I G H T

AES (Advanced Encryption Standard)

- 전세계 블록 암호 표준
  - FIPS 197
  - ISO/IEC 18033-3
- 다양한 모드 존재
  - ECB, CBC, CFB, OFB, **CTR**

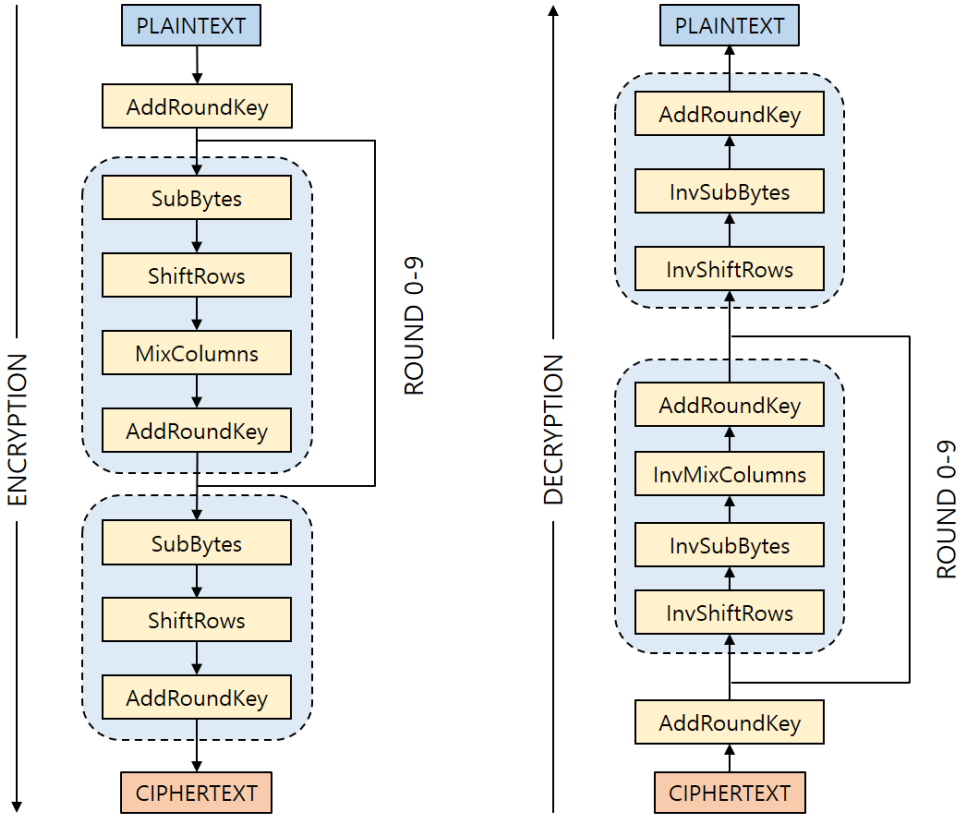


Fig 1. AES Structure

AES-CTR

- AES Counter Mode
- **병렬처리** 가능
- 128bits IV(Initial Vector)
  - 96bits Nonce
  - 32bits Counter
- Counter만 1씩 증가

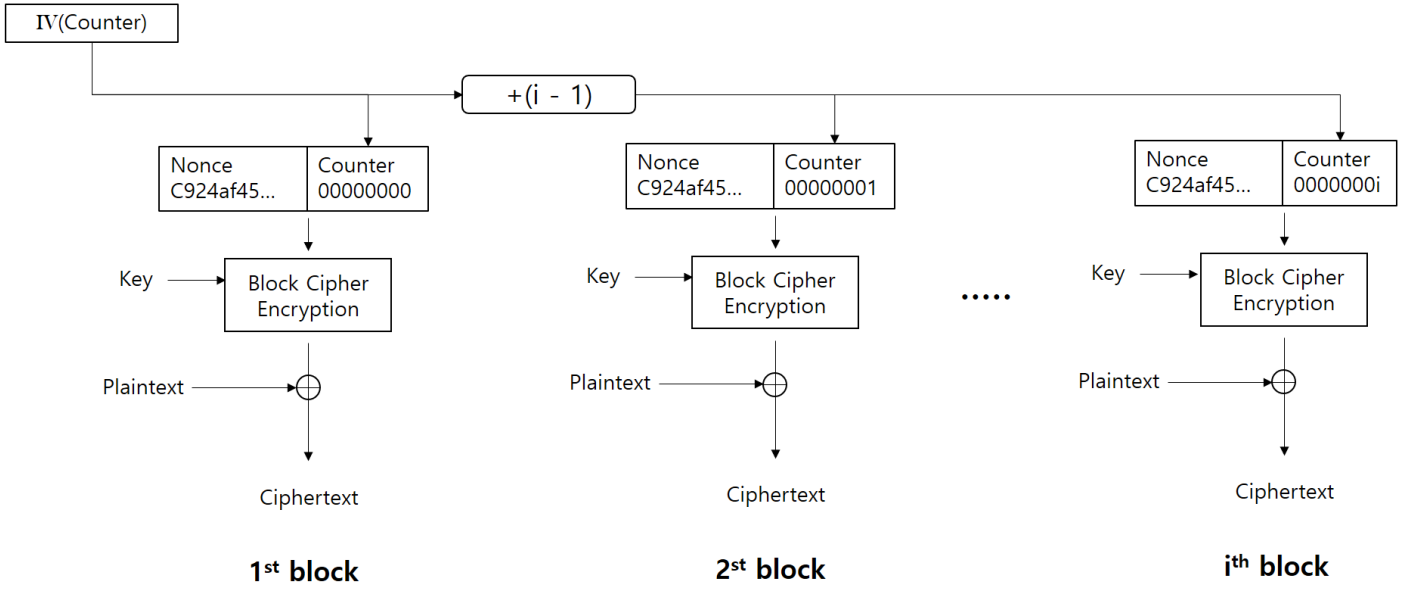


Fig 2. AES-CTR Structure

Side Channel Attack(SCA)

- 연산 중 발생하는 **부가적 요소**를 이용한 공격
- Power Analysis
  - SPA(Simple Power Analysis)
  - DPA(Differential Power Analysis)
  - CPA(Correlation Power Analysis)

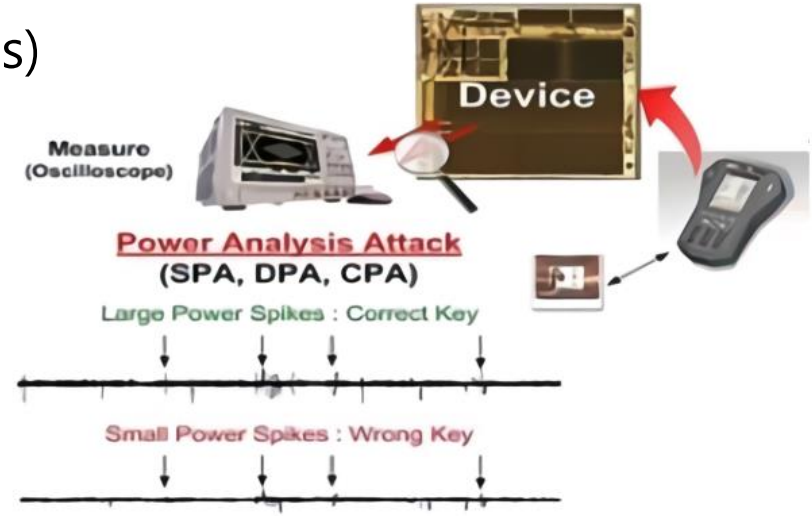


Fig 3. Power Analysis Attack

Masking

- SCA Countermeasure
  - 전력 분석 방해
- 공개된[1] 마스킹 기법 응용
  - 8bits Microcontroller **최적 구현**

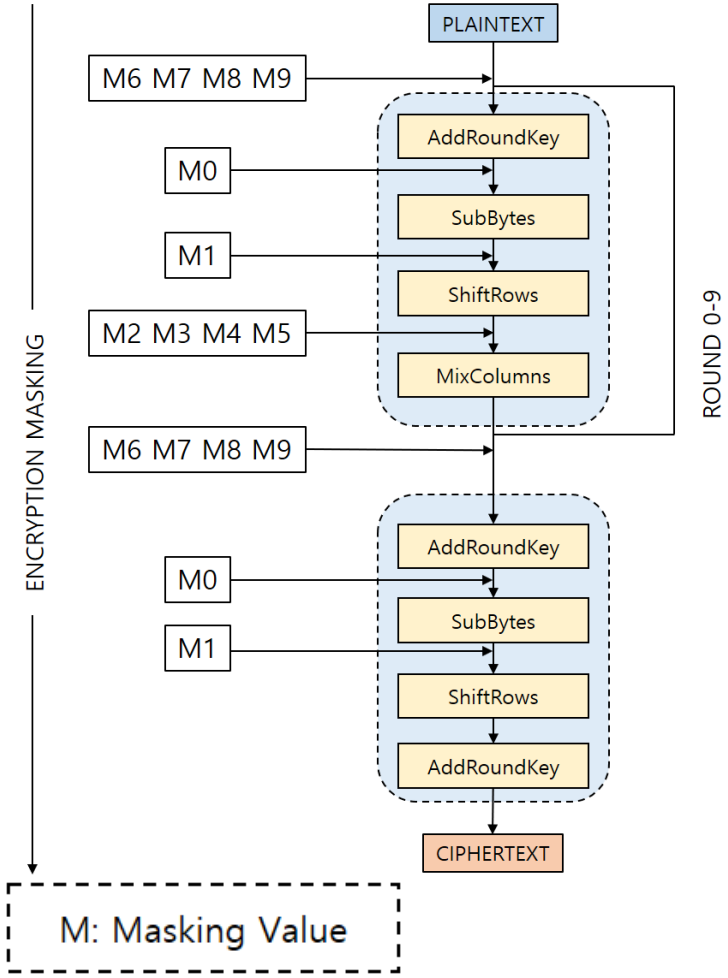


Fig 4. Masking Process

02

FACE

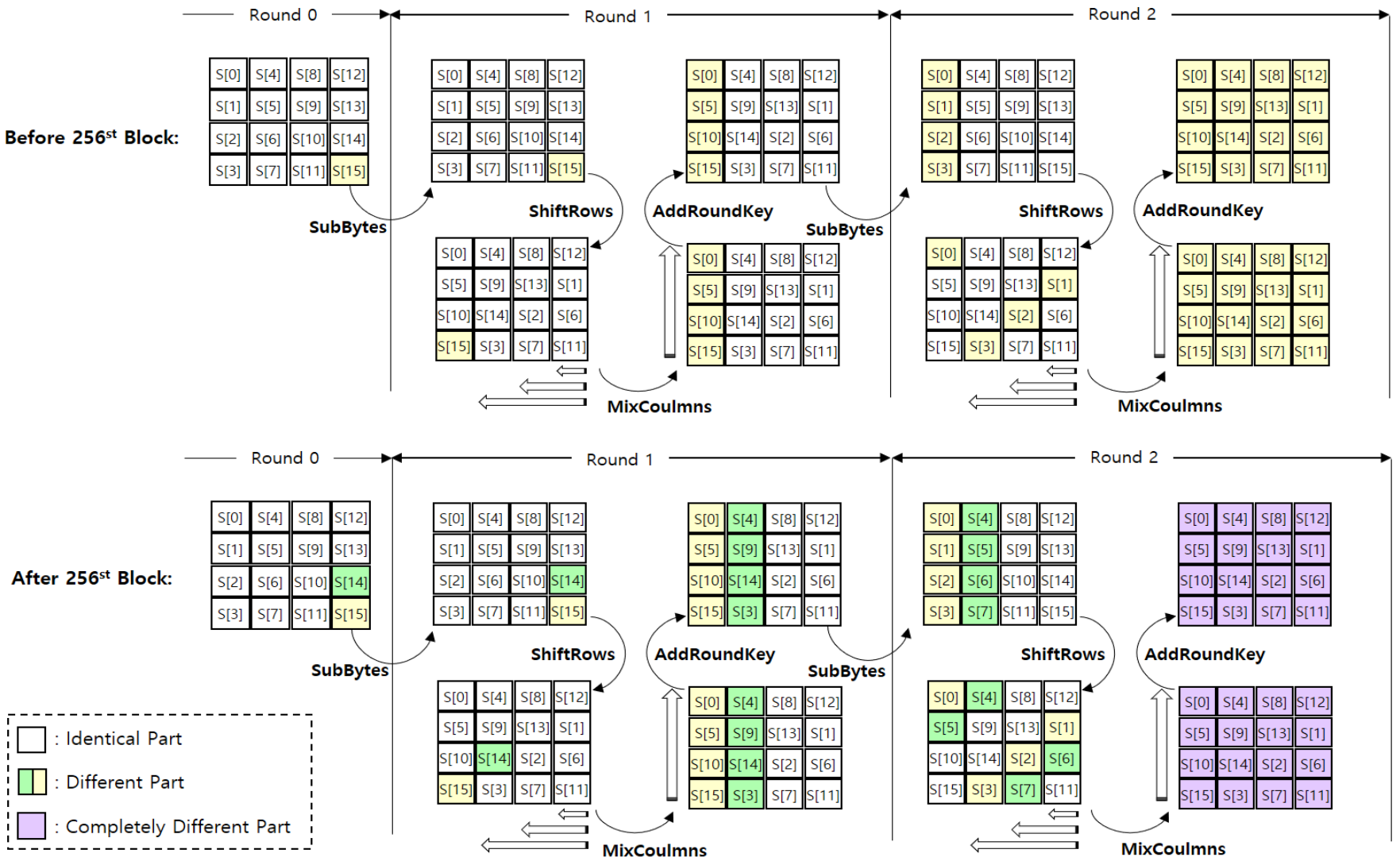
F A C E - L I G H T

Outline

- AES-CTR의 특징을 활용한 최적화 구현 기법
  - 마지막 byte는 Counter 값을 저장
  - 첫 블록과 다음 블록의 차이는 **마지막 byte**

- Counter를 제외하고 반복되는 부분을 저장
  - 사전 테이블(Look Up Table, **LUT**)로 저장
  - 특정 Round마다 정해진 LUT를 참조
  - 총 5개의 LUT가 필요 (**약 $5KB$**)

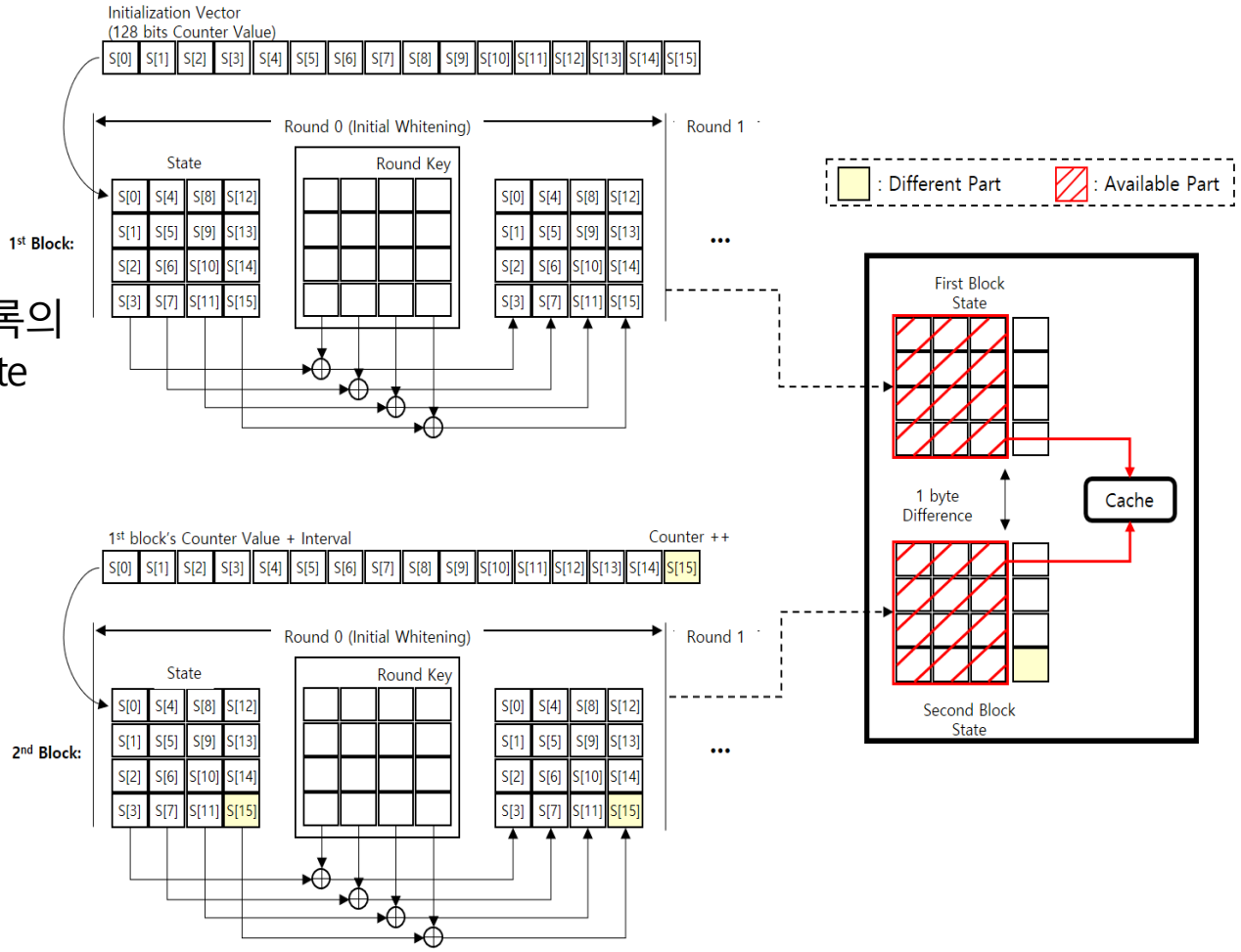
- Counter 값 변화에 따라 일정 주기마다 **LUT 업데이트 필요**


Round 0
Round 1
Round 2
Before 256st Block:
SubBytes
ShiftRows
AddRoundKey
SubBytes
ShiftRows
AddRoundKey
MixCoulmns
MixCoulmns
After 256st Block:
SubBytes
ShiftRows
AddRoundKey
SubBytes
ShiftRows
AddRoundKey
MixCoulmns
MixCoulmns
: Identical Part
: Different Part
: Completely Different Part

02

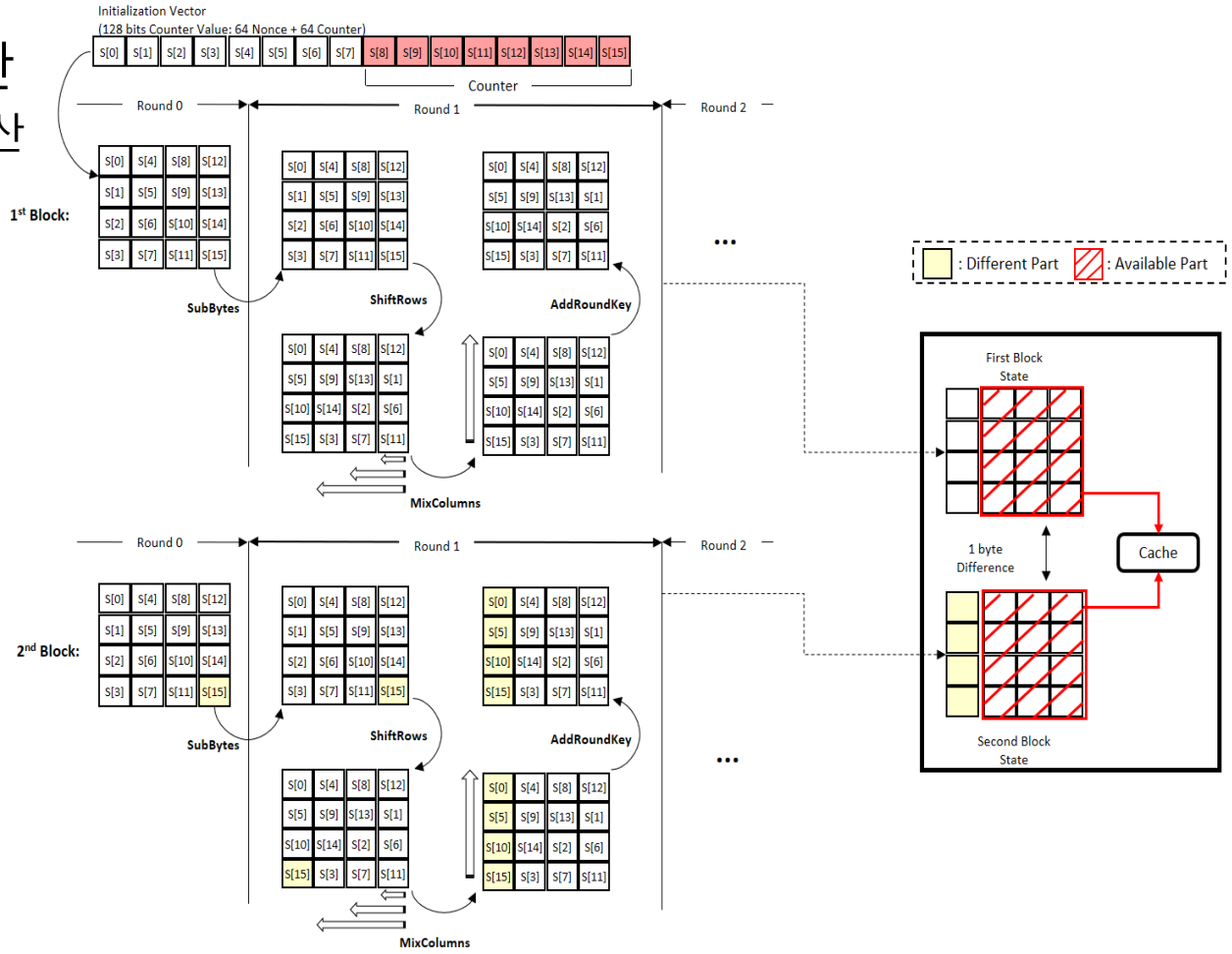
Structure (Round 0)

- 전체 데이터 중 마지막 byte의 변경을 활용
  - 첫 블록과 다음 블록의 차이는 마지막 1byte
- Round 0 이후 12bytes 사용 가능
  - S[12], S[13], S[14], S[15]
- $(2^{32}-1)$번 만큼 테이블 사용 가능
  - 갱신이 필요 없음



Structure (Round 1)

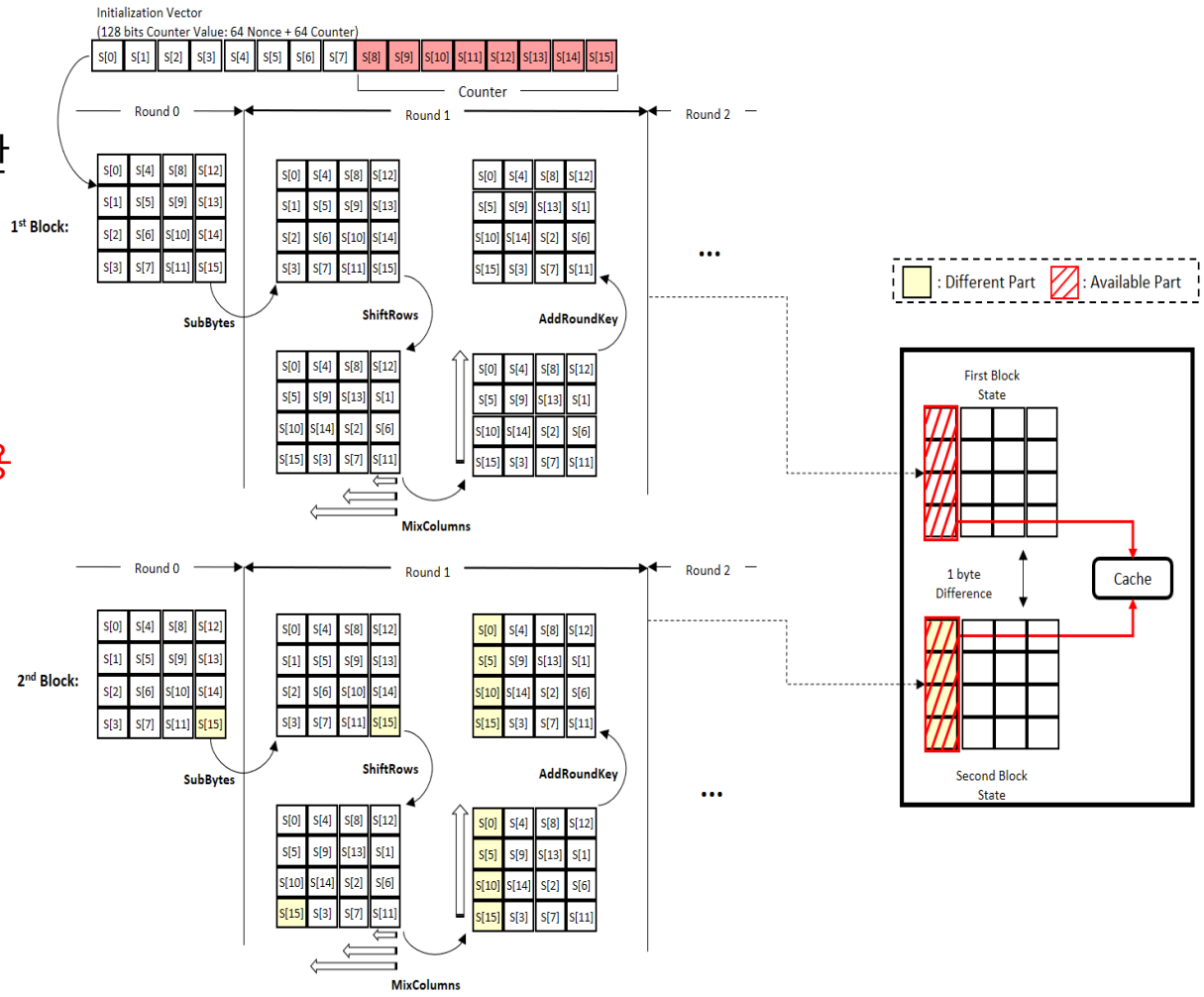
- 마지막 byte의 확산
  - 두 단계에 걸쳐 확산
  - ShiftRows
  - MixColumns
- LUT 생성 가능
  - 첫 열 제외
- $(2^8 - 1)$번 만큼 테이블 사용 가능



02

Structure (Round 1+)

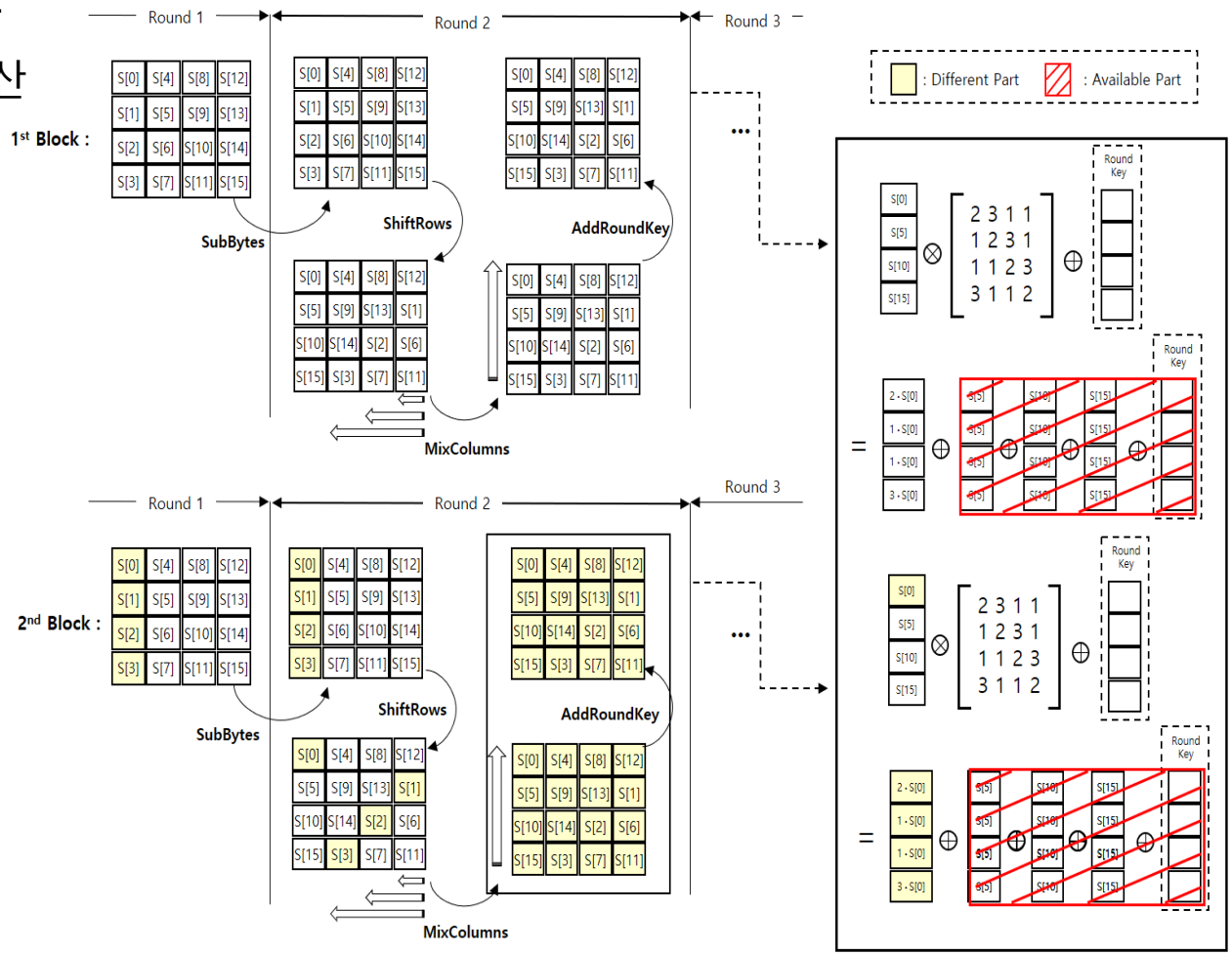
- 마지막 byte의 확산
  - 두 단계에 걸쳐 확산
  - ShiftRows
  - MixColumns
- LUT 생성 가능
  - S[15]를 index로 사용
  - Table Size: $1KB$
- $(2^8 \times 2^{32})$번 만큼 테이블 사용 가능



02

Structure (Round 2)

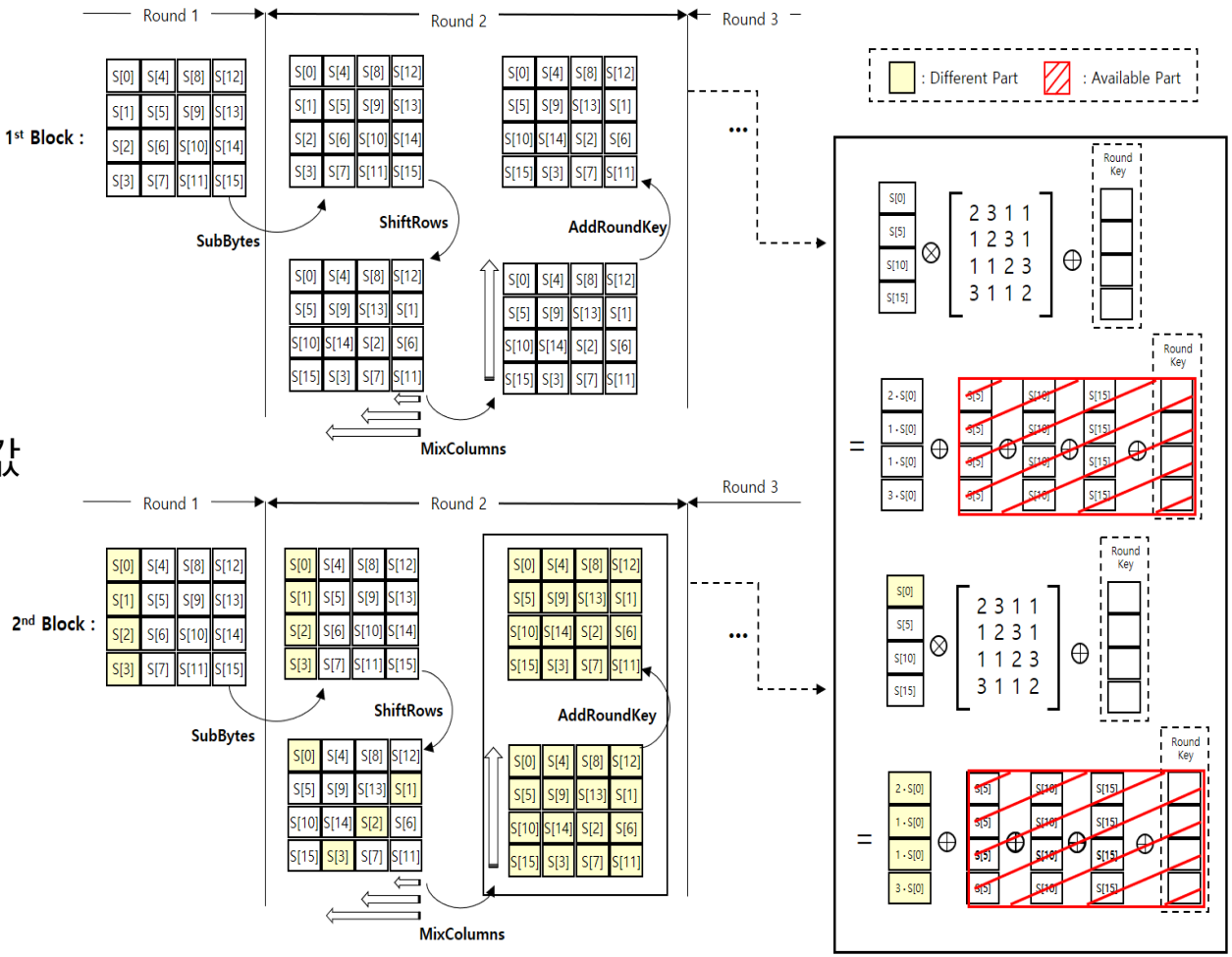
- 첫 열 bytes의 확산
  - 두 단계에 걸쳐 확산
  - ShiftRows
  - MixColumns
- Round 2 이후 모든 값 영향
- LUT 생성 가능
  - MixColumns의 중간 연산 값
- (2^8 - 1)번 만큼 테이블 사용 가능



02

Structure (Round 2+)

- Round 2의 LUT에 저장되지 않은 값
  - S[0], S[1], S[2], S[3]
- LUT 생성 가능
  - 미사용 값의 중간 값
  - Table Size: $4KB$
- $(2^8 \times 2^{32})$번 만큼 테이블 사용 가능



02

Limitation of FACE

- 8bits Microcontroller에서 사용하기 어려움
  - **LUT의 용량** 문제
  - 최소 $5KB$의 메모리 필요
- 일정 주기마다 **LUT의 갱신**이 필요

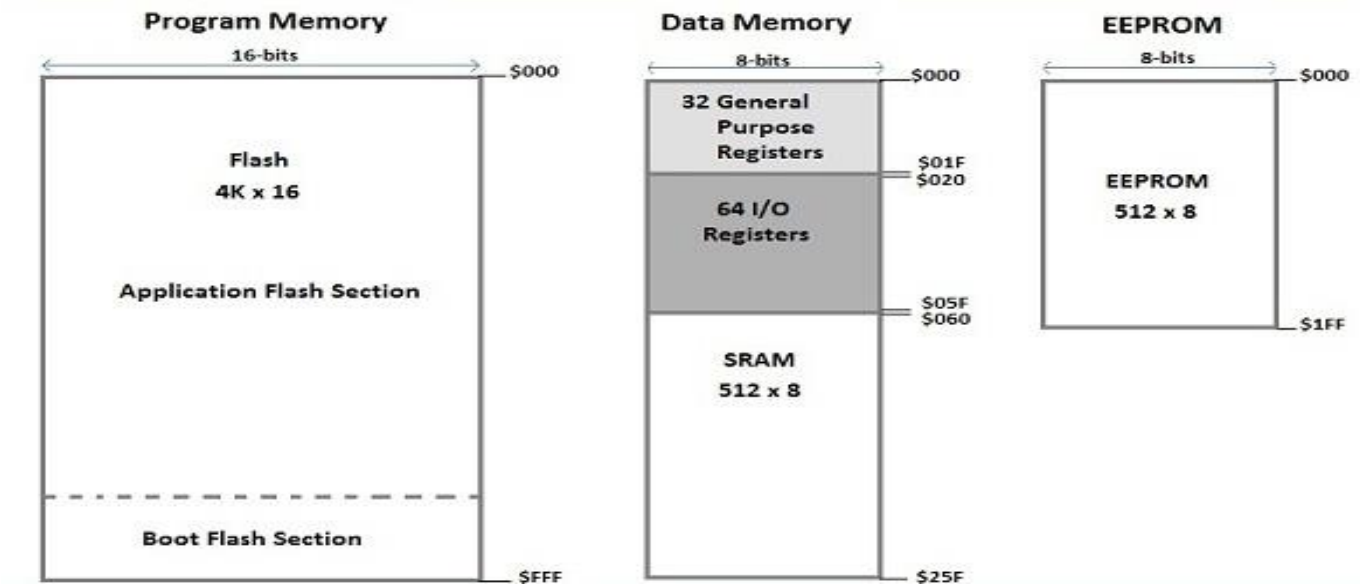


Fig 6. Arduino Uno Memory Structure

03

Our Work

FACE - LIGHT

F A C E - L I G H T

Target Board

- 8bits Microcontroller
  - **Arduino Uno ATmega328P**

- Hardware Spec
  - Flash Memory: $32KB$
  - SRAM: $2KB$
  - EEPROM: $1KB$
  - Clock Speed: 16MHz

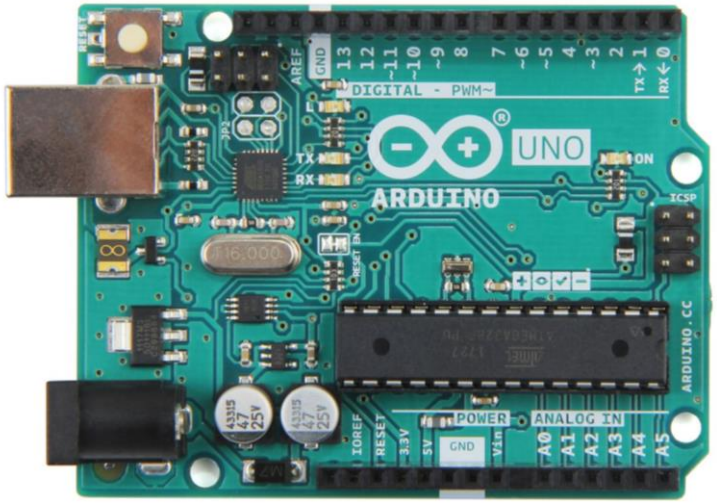


Fig 7. Arduino Uno

Overview

- FACE에 기반한 최적화 구현 기법
  - 저전력 프로세스에 최적화 된 구현 기법

- Counter값에 의존하는 반복 부분을 저장
  - 사전 테이블(Look Up Table, **LUT**)로 저장
  - 한번의 참조로 **다수의 Round를 생략** 가능
  - 총 4개의 LUT가 필요 (***4KB***)

- Counter 값 변화에 따라 **LUT 업데이트 필요 없음**

- 기존 FACE와 결합하여 성능 향상

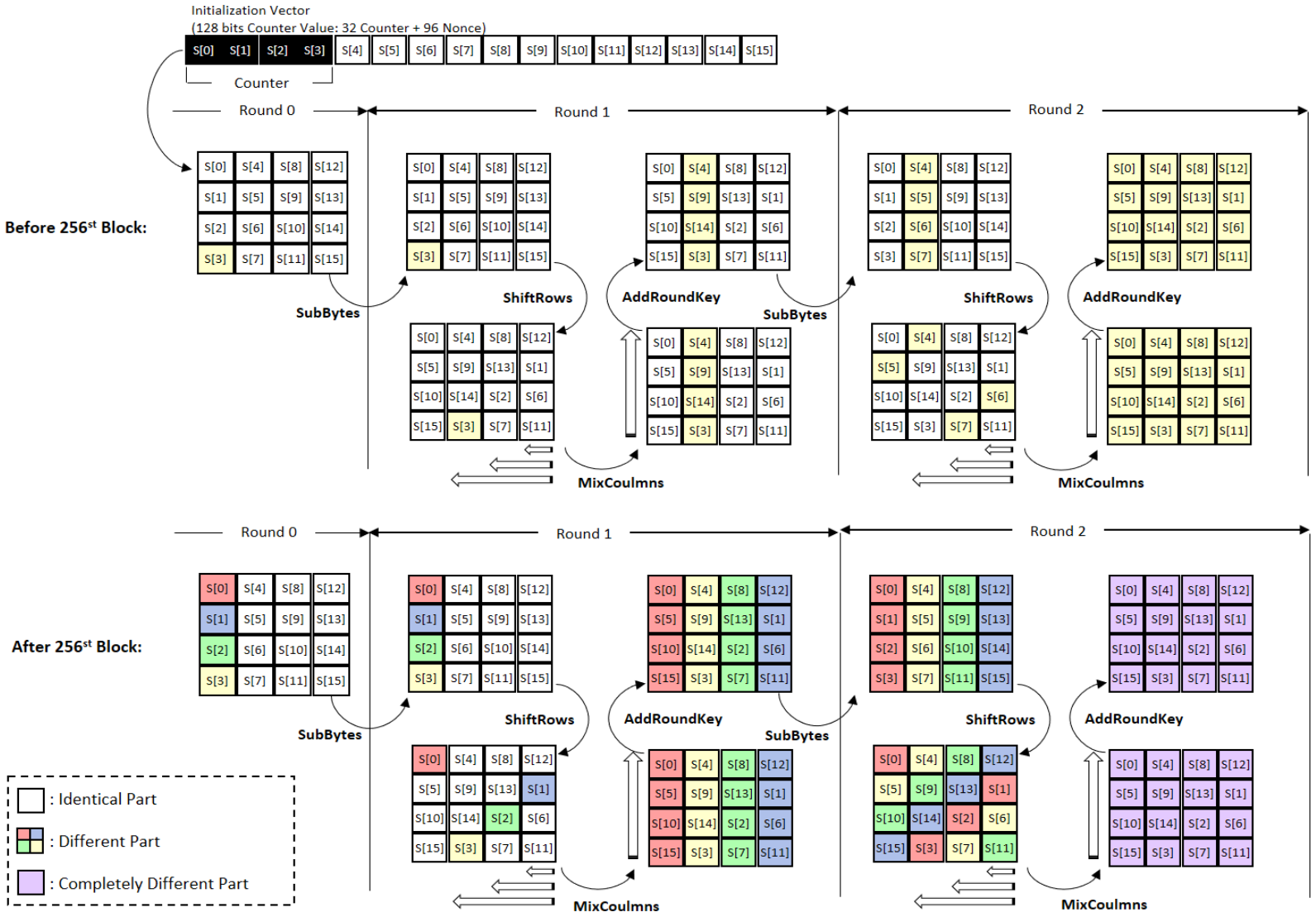


Fig 8. Overview of FACE-LIGHT

Initialization Vector

(128 bits Counter Value: 32 Counter + 96 Nonce)

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Counter

CIPHER KEY

00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f

	Start of round	After SubBytes	After ShiftRows	After MixColumns
Round 0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
Round 1	00 04 08 0c 01 05 09 0d 02 06 0a 0e 03 07 0b 0f	63 f2 30 fe 7c 6b 01 d7 77 6f 67 ab 7b c5 2b 76	63 f2 30 fe 6b 01 d7 7c 67 ab 77 6f 76 7b c5 2b	6a 2c b0 27 6a 6d d9 9c 5c 33 5d 21 45 51 61 5c
Round 2	bc fe 6a f1 c0 c2 7f 37 28 41 25 57 b8 ab 90 a2	65 bb 02 a1 ba 25 d2 9a 34 83 3f 5b 6c 62 60 3a	65 bb 02 a1 25 d2 9a ba 3f 5b 34 83 3a 6c 62 60	a0 37 e7 6f 54 85 13 30 70 6b 56 a6 c1 87 6c 01

Initialization Vector

(128 bits Counter Value: 32 Counter + 96 Nonce)

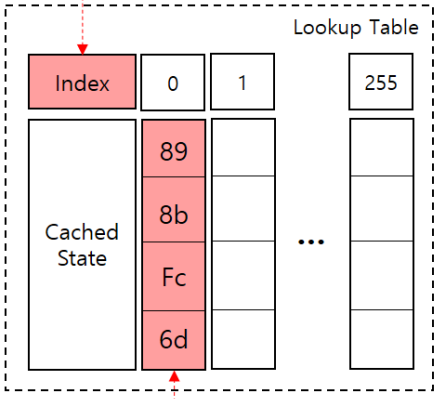
00	00	00	01	00	00	00	00	00	00	00	00	00	00	00	00

Counter

CIPHER KEY

00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f

	Start of round	After SubBytes	After ShiftRows	After MixColumns
Round 0	00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00			
Round 1	00 04 08 0c 01 05 09 0d 02 06 0a 0e 02 07 0b 0f	63 f2 30 fe 7c 6b 01 d7 77 6f 67 ab 77 c5 2b 76	63 f2 30 fe 6b 01 d7 7c 67 ab 77 6f 76 77 c5 2b	6a 20 b0 27 6a 61 d9 9c 5c 27 5d 21 45 49 61 5c
Round 2	bc f2 6a f1 c0 ce 7f 37 28 55 25 57 b8 b3 90 a2	65 89 02 a1 ba 8b d2 9a 34 fc 3f 5b 6c 6d 60 3a	65 89 02 a1 8b d2 9a ba 3f 5b 34 fc 3a 6c 6d 60	49 53 e8 10 13 b7 1c b1 de 59 47 58 6f d1 72 7e



Initialization Vector

(128 bits Counter Value: 32 Counter + 96 Nonce)

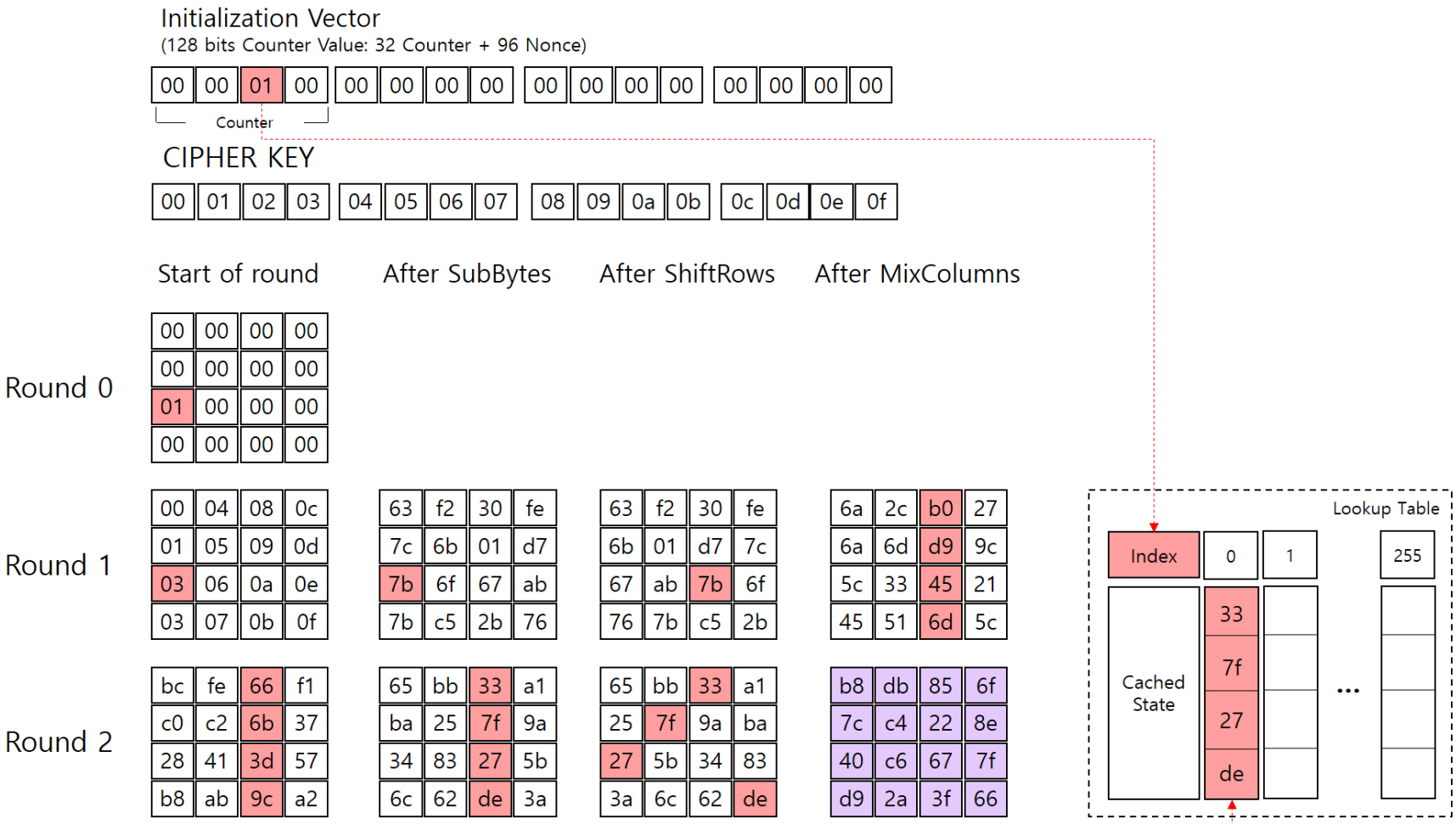
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Counter

CIPHER KEY

00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f

	Start of round	After SubBytes	After ShiftRows	After MixColumns
Round 0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
Round 1	00 04 08 0c 01 05 09 0d 02 06 0a 0e 03 07 0b 0f	63 f2 30 fe 7c 6b 01 d7 77 6f 67 ab 7b c5 2b 76	63 f2 30 fe 6b 01 d7 7c 67 ab 77 6f 76 7b c5 2b	6a 2c b0 27 6a 6d d9 9c 5c 33 5d 21 45 51 61 5c
Round 2	bc fe 6a f1 c0 c2 7f 37 28 41 25 57 b8 ab 90 a2	65 bb 02 a1 ba 25 d2 9a 34 83 3f 5b 6c 62 60 3a	65 bb 02 a1 25 d2 9a ba 3f 5b 34 83 3a 6c 62 60	a0 37 e7 6f 54 85 13 30 70 6b 56 a6 c1 87 6c 01


Initialization Vector
(128 bits Counter Value: 32 Counter + 96 Nonce)
00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00
Counter
CIPHER KEY
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
Start of round
After SubBytes
After ShiftRows
After MixColumns
Round 0
00 00 00 00
00 00 00 00
01 00 00 00
00 00 00 00
Round 1
00 04 08 0c
01 05 09 0d
03 06 0a 0e
03 07 0b 0f
63 f2 30 fe
7c 6b 01 d7
7b 6f 67 ab
7b c5 2b 76
63 f2 30 fe
6b 01 d7 7c
67 ab 7b 6f
76 7b c5 2b
6a 2c b0 27
6a 6d d9 9c
5c 33 45 21
45 51 6d 5c
Round 2
bc fe 66 f1
c0 c2 6b 37
28 41 3d 57
b8 ab 9c a2
65 bb 33 a1
ba 25 7f 9a
34 83 27 5b
6c 62 de 3a
65 bb 33 a1
25 7f 9a ba
27 5b 34 83
3a 6c 62 de
b8 db 85 6f
7c c4 22 8e
40 c6 67 7f
d9 2a 3f 66
Lookup Table
Index
0
1
255
Cached State
33
7f
27
de
...

Initialization Vector

(128 bits Counter Value: 32 Counter + 96 Nonce)

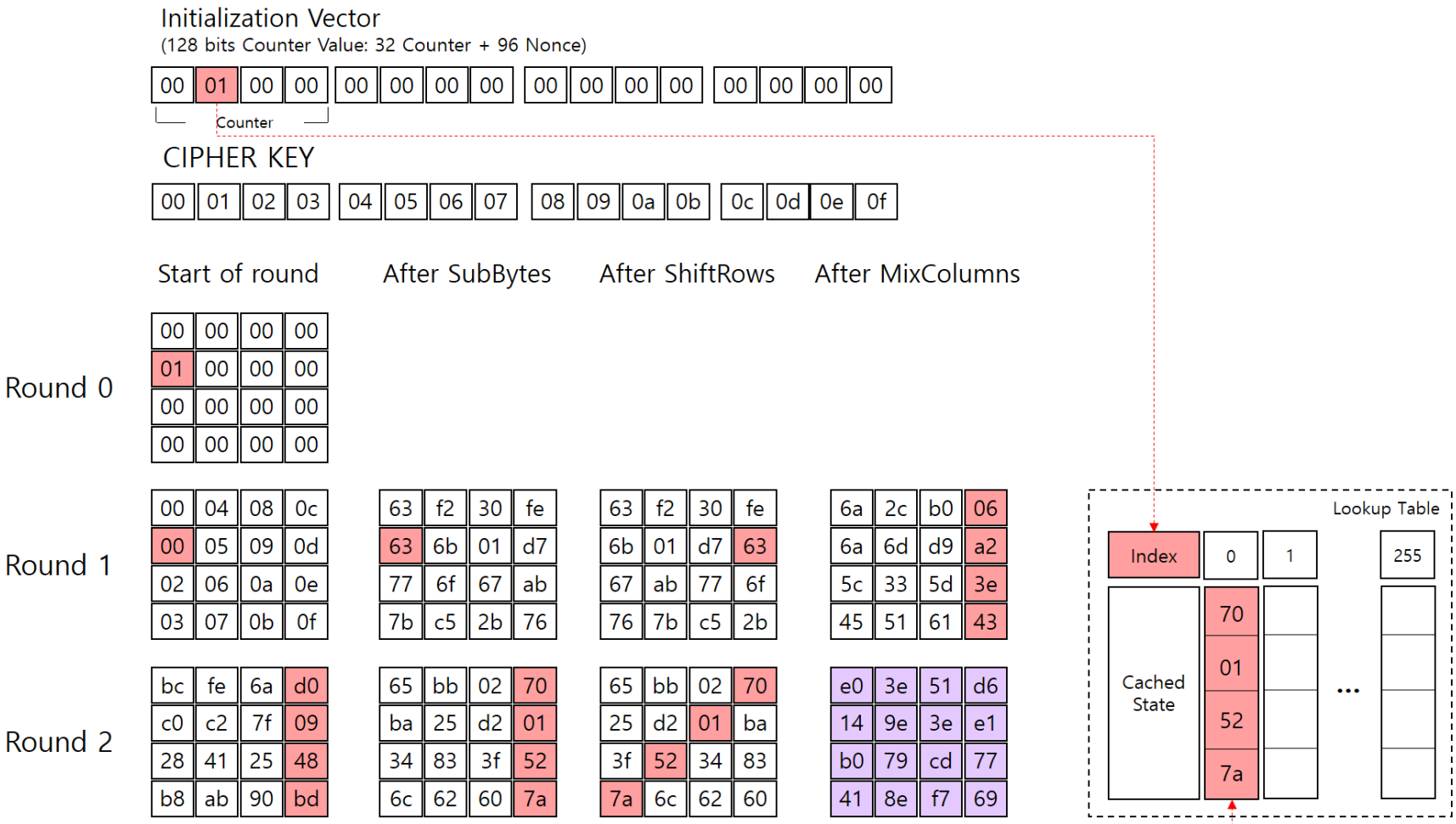
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Counter

CIPHER KEY

00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f

	Start of round	After SubBytes	After ShiftRows	After MixColumns
Round 0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
Round 1	00 04 08 0c 01 05 09 0d 02 06 0a 0e 03 07 0b 0f	63 f2 30 fe 7c 6b 01 d7 77 6f 67 ab 7b c5 2b 76	63 f2 30 fe 6b 01 d7 7c 67 ab 77 6f 76 7b c5 2b	6a 2c b0 27 6a 6d d9 9c 5c 33 5d 21 45 51 61 5c
Round 2	bc fe 6a f1 c0 c2 7f 37 28 41 25 57 b8 ab 90 a2	65 bb 02 a1 ba 25 d2 9a 34 83 3f 5b 6c 62 60 3a	65 bb 02 a1 25 d2 9a ba 3f 5b 34 83 3a 6c 62 60	a0 37 e7 6f 54 85 13 30 70 6b 56 a6 c1 87 6c 01


Initialization Vector
(128 bits Counter Value: 32 Counter + 96 Nonce)
00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Counter
CIPHER KEY
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
Start of round
After SubBytes
After ShiftRows
After MixColumns
Round 0
00 00 00 00
01 00 00 00
00 00 00 00
00 00 00 00
Round 1
00 04 08 0c
00 05 09 0d
02 06 0a 0e
03 07 0b 0f
63 f2 30 fe
63 6b 01 d7
77 6f 67 ab
7b c5 2b 76
63 f2 30 fe
6b 01 d7 63
67 ab 77 6f
76 7b c5 2b
6a 2c b0 06
6a 6d d9 a2
5c 33 5d 3e
45 51 61 43
Round 2
bc fe 6a d0
c0 c2 7f 09
28 41 25 48
b8 ab 90 bd
65 bb 02 70
ba 25 d2 01
34 83 3f 52
6c 62 60 7a
65 bb 02 70
25 d2 01 ba
3f 52 34 83
7a 6c 62 60
e0 3e 51 d6
14 9e 3e e1
b0 79 cd 77
41 8e f7 69
Lookup Table
Index
0
1
255
Cached State
70
01
52
7a
...

Initialization Vector

(128 bits Counter Value: 32 Counter + 96 Nonce)

00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Counter

CIPHER KEY

00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f

	Start of round	After SubBytes	After ShiftRows	After MixColumns
Round 0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
Round 1	00 04 08 0c 01 05 09 0d 02 06 0a 0e 03 07 0b 0f	63 f2 30 fe 7c 6b 01 d7 77 6f 67 ab 7b c5 2b 76	63 f2 30 fe 6b 01 d7 7c 67 ab 77 6f 76 7b c5 2b	6a 2c b0 27 6a 6d d9 9c 5c 33 5d 21 45 51 61 5c
Round 2	bc fe 6a f1 c0 c2 7f 37 28 41 25 57 b8 ab 90 a2	65 bb 02 a1 ba 25 d2 9a 34 83 3f 5b 6c 62 60 3a	65 bb 02 a1 25 d2 9a ba 3f 5b 34 83 3a 6c 62 60	a0 37 e7 6f 54 85 13 30 70 6b 56 a6 c1 87 6c 01

Initialization Vector

(128 bits Counter Value: 32 Counter + 96 Nonce)

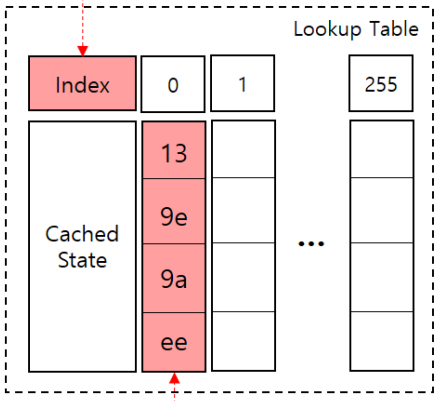
01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Counter

CIPHER KEY

00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f

	Start of round	After SubBytes	After ShiftRows	After MixColumns
Round 0	01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
Round 1	01 04 08 0c 01 05 09 0d 02 06 0a 0e 03 07 0b 0f	7c f2 30 fe 7c 6b 01 d7 77 6f 67 ab 7b c5 2b 76	7c f2 30 fe 6b 01 d7 7c 67 ab 77 6f 76 7b c5 2b	54 2c b0 27 75 6d d9 9c 43 33 5d 21 64 51 61 5c
Round 2	82 fe 6a f1 df c2 7f 37 37 41 25 57 99 ab 90 a2	13 bb 02 a1 9e 25 d2 9a 9a 83 3f 5b ee 62 60 3a	13 bb 02 a1 25 d2 9a 9e 3f 5b 9a 83 3a ee 62 60	4c b5 49 03 22 07 fa 78 06 f6 11 82 5b 98 c2 25



Look Up Table Structure

- Size of LUT: $4KB$
- 업데이트 **필요 없음**

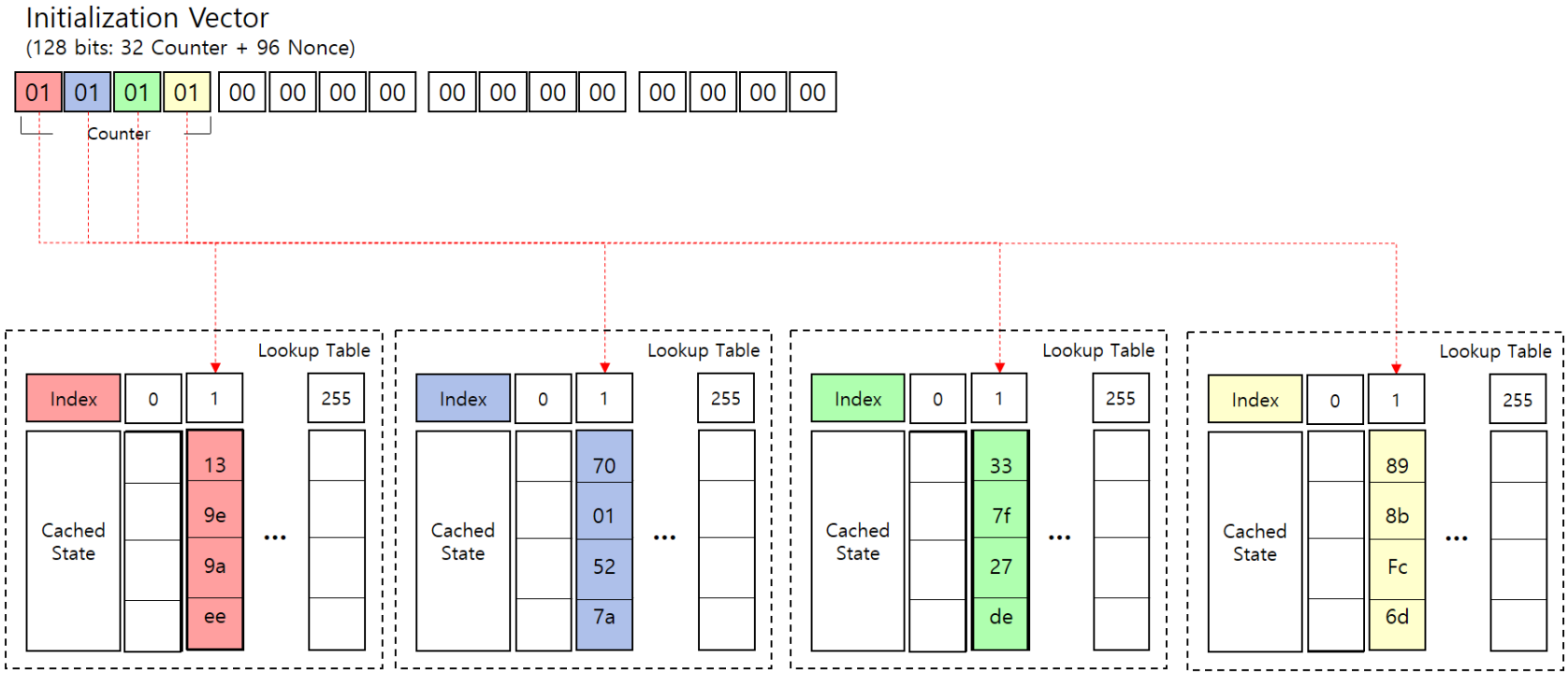
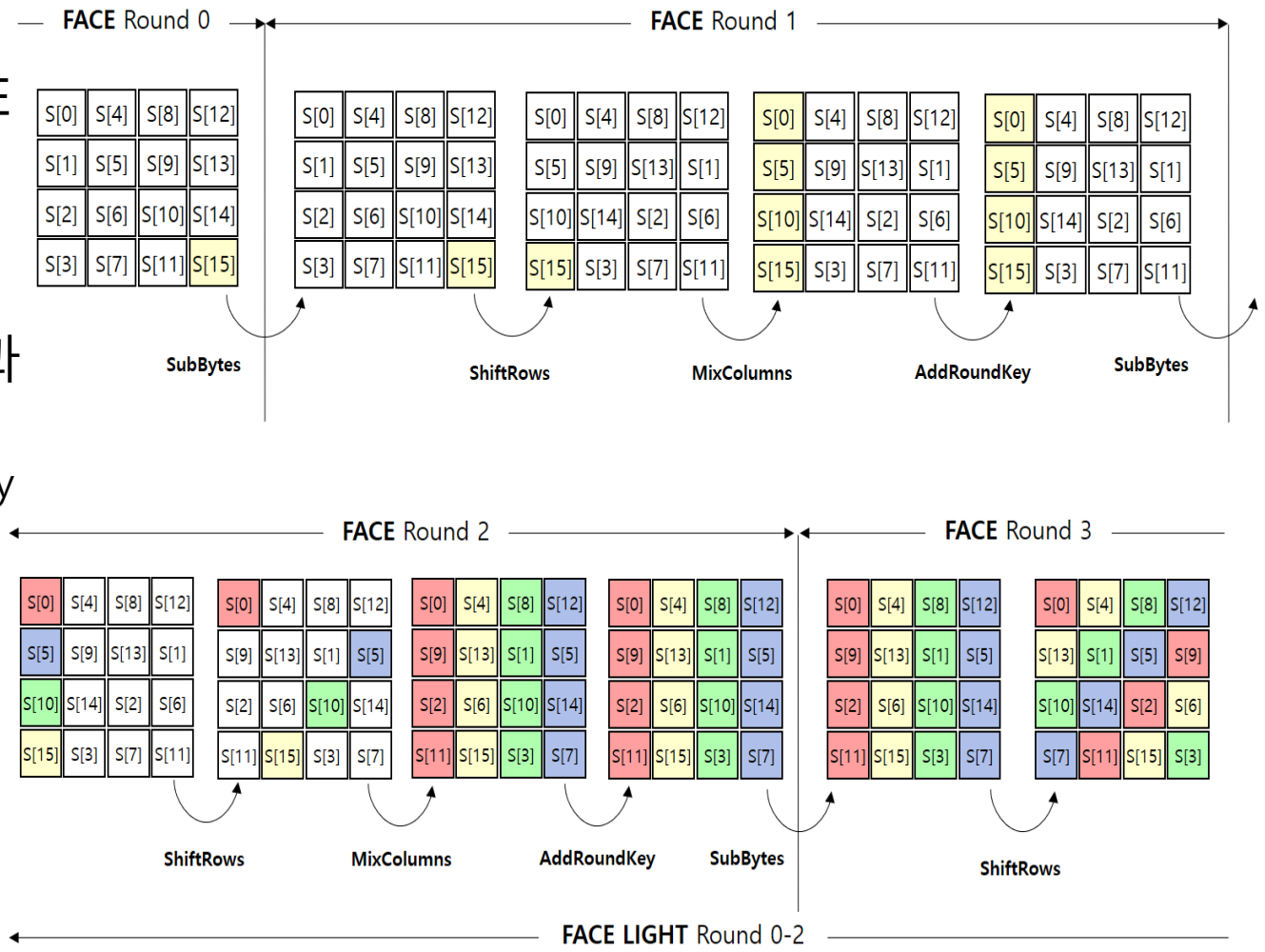


Fig 9. FACE-LIGHT Look Up Table

Extended FACE

- Extended FACE
  - Original FACE
  - FACE-Light
- 연산 절감 효과
  - Subbytes
  - AddRoundKey



Evaluation (Calculating Speed)

- 표준 AES보다 약 **22% 성능 향상**을 보임
- 추가적인 LUT 업데이트 시간 소요가 없음

Unit: Clock Cycles

Security Level	Dinu et al. [2]	Otte et al. [3]	FACE-Light (Our Work)	Ex-FACE (Our Work)
AES-128	2,835	2,507	**2,218**	**1,967**
AES-192	N/A	2,991	**2,702**	**2,449**
AES-256	N/A	3,473	**3,184**	**2,931**

Table 1. Comparison of calculating speed

Evaluation (vs FACE)

- FACE-LIGHT **8bits Microcontroller**에 최적화 됨
- LUT 업데이트가 없기 때문에 Constant Timing 유지
- 8bits 저전력 프로세서부터 제약없이 사용 가능

	FACE	FACE-LIGHT (Our Work)
Table Update	O	**X**
Constant Timing	Not Support	**Support**
Target Processor	32-bits or above	**8-bits or above**
Expandable Round	Round 2	**Round 3**

Table 2. Comparison with original FACE

Evaluation (Side Channel Attack Resistance)

- **파형 분석 공격**(CPA, DPA)에 대한 **저항성**을 지님

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	2B 0.7013	7E 0.7601	15 0.7126	16 0.7651	28 0.7064	AE 0.7453	D2 0.7093	A6 0.7518	AB 0.7095	F7 0.7581	15 0.7409	88 0.7640	09 0.7067	CF 0.7530	4F 0.7115	3C 0.7998
1	2A 0.3791	7F 0.3795	14 0.3778	17 0.3984	29 0.3711	AF 0.3961	D3 0.3647	A7 0.3987	AA 0.3669	F6 0.4060	14 0.3464	89 0.3737	08 0.3563	CE 0.3780	4E 0.3653	3D 0.3930
2	24 0.1953	9A 0.2181	83 0.2100	E3 0.2041	DC 0.1975	4A 0.2078	A7 0.2149	99 0.1848	5E 0.1874	13 0.2106	BF 0.1899	BD 0.1909	B1 0.1851	B4 0.1993	F5 0.2012	E1 0.1954
3	FB 0.1948	48 0.2059	1E 0.2012	38 0.1840	82 0.1939	C3 0.1845	60 0.2144	53 0.1823	3F 0.1819	8C 0.1918	7E 0.1888	FD 0.1892	91 0.1844	3A 0.1939	B1 0.2003	21 0.1892
4	20 0.1947	0B 0.2035	E5 0.1889	29 0.1821	47 0.1927	D5 0.1730	F9 0.2119	75 0.1794	97 0.1780	02 0.1888	B1 0.1811	33 0.1793	9D 0.1778	2B 0.1919	F7 0.1971	B1 0.1878
5	D8 0.1881	88 0.1896	A1 0.1879	E5 0.1780	9C 0.1890	98 0.1719	FD 0.2074	8C 0.1770	EE 0.1767	C2 0.1817	8C 0.1783	9E 0.1782	D1 0.1724	F0 0.1833	0C 0.1935	C9 0.1790
6	06 0.1844	68 0.1885	E0 0.1864	5C 0.1769	D8 0.1830	5B 0.1696	D9 0.2046	DA 0.1668	01 0.1747	E4 0.1671	7A 0.1740	7D 0.1775	99 0.1696	05 0.1765	BD 0.1857	96 0.1765
7	9F 0.1825	41 0.1845	5A 0.1864	36 0.1760	98 0.1783	E0 0.1674	64 0.1895	6C 0.1657	32 0.1719	C8 0.1634	56 0.1722	B7 0.1725	A7 0.1688	12 0.1630	B3 0.1816	02 0.1746
8	46 0.1791	75 0.1766	7E 0.1858	C6 0.1738	45 0.1778	5F 0.1671	20 0.1971	93 0.1644	28 0.1699	9A 0.1616	5A 0.1711	AE 0.1668	D4 0.1677	A2 0.1604	B9 0.1813	09 0.1713
9	F6 0.1756	6D 0.1747	5E 0.1858	18 0.1722	63 0.1766	BD 0.1663	42 0.1966	3A 0.1643	33 0.1692	CA 0.1608	C8 0.1704	58 0.1608	B9 0.1672	DC 0.1599	F9 0.1811	40 0.1712
10	64 0.1754	C6 0.1725	A3 0.1783	A5 0.1697	03 0.1756	90 0.1661	87 0.1952	51 0.1642	18 0.1667	3D 0.1591	3D 0.1704	83 0.1607	35 0.1662	0C 0.1580	40 0.1808	49 0.1706

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	201	87	49	195	145	140	231	80	238	146	220	216	29	249	86	151
0	42 0.0727	B4 0.0755	B8 0.0690	EF 0.0692	51 0.0683	69 0.0685	55 0.0728	7F 0.0686	78 0.0773	CB 0.0727	78 0.0672	1D 0.0729	D8 0.0696	F6 0.0689	E2 0.0691	73 0.0720
1	20 0.0704	60 0.0679	48 0.0682	CF 0.0687	B3 0.0680	C1 0.0682	0F 0.0697	CB 0.0673	DF 0.0705	B5 0.0699	1E 0.0643	B9 0.0723	C1 0.0676	A1 0.0673	9C 0.0682	43 0.0704
2	F0 0.0695	1E 0.0649	CE 0.0680	A0 0.0676	B7 0.0679	7E 0.0647	06 0.0689	EB 0.0665	96 0.0652	22 0.0676	44 0.0642	0E 0.0686	0C 0.0664	AA 0.0666	01 0.0676	B6 0.0701
3	19 0.0682	44 0.0648	E5 0.0679	25 0.0676	A1 0.0672	99 0.0644	4A 0.0669	34 0.0658	06 0.0651	17 0.0643	C5 0.0638	85 0.0671	E6 0.0663	89 0.0647	39 0.0672	24 0.0686
4	2F 0.0680	84 0.0646	7A 0.0663	92 0.0672	4F 0.0671	D9 0.0634	60 0.0669	FA 0.0645	EE 0.0645	7D 0.0641	1F 0.0636	21 0.0667	7B 0.0645	65 0.0639	F9 0.0666	D3 0.0686
5	44 0.0675	D4 0.0640	FF 0.0660	D4 0.0665	82 0.0670	D2 0.0633	C2 0.0655	A7 0.0640	8E 0.0639	EE 0.0638	DA 0.0635	3D 0.0667	BE 0.0644	5D 0.0636	86 0.0655	B7 0.0681
6	57 0.0674	2B 0.0636	7E 0.0654	84 0.0657	E4 0.0648	49 0.0623	13 0.0640	05 0.0638	70 0.0630	2A 0.0629	2B 0.0627	73 0.0657	41 0.0638	74 0.0634	92 0.0651	EE 0.0662
7	17 0.0665	7B 0.0635	57 0.0634	B0 0.0654	BD 0.0645	45 0.0621	93 0.0638	1E 0.0627	18 0.0627	37 0.0628	46 0.0627	3E 0.0650	CC 0.0635	51 0.0630	D3 0.0645	C3 0.0662
8	C3 0.0661	63 0.0629	FC 0.0627	71 0.0638	75 0.0644	6F 0.0618	63 0.0635	AF 0.0625	94 0.0624	00 0.0623	E5 0.0626	EF 0.0649	A1 0.0632	B9 0.0629	F6 0.0642	D6 0.0661
9	79 0.0659	96 0.0626	9A 0.0620	C5 0.0632	21 0.0643	6C 0.0618	BB 0.0631	AB 0.0624	17 0.0624	C2 0.0615	FA 0.0620	47 0.0640	9D 0.0629	CD 0.0629	FE 0.0635	BF 0.0647
10	0D 0.0627	5F 0.0623	D3 0.0619	F4 0.0631	48 0.0640	C9 0.0618	9E 0.0628	DA 0.0619	AD 0.0616	AA 0.0615	D0 0.0620	B1 0.0638	34 0.0627	01 0.0628	E3 0.0630	A7 0.0646

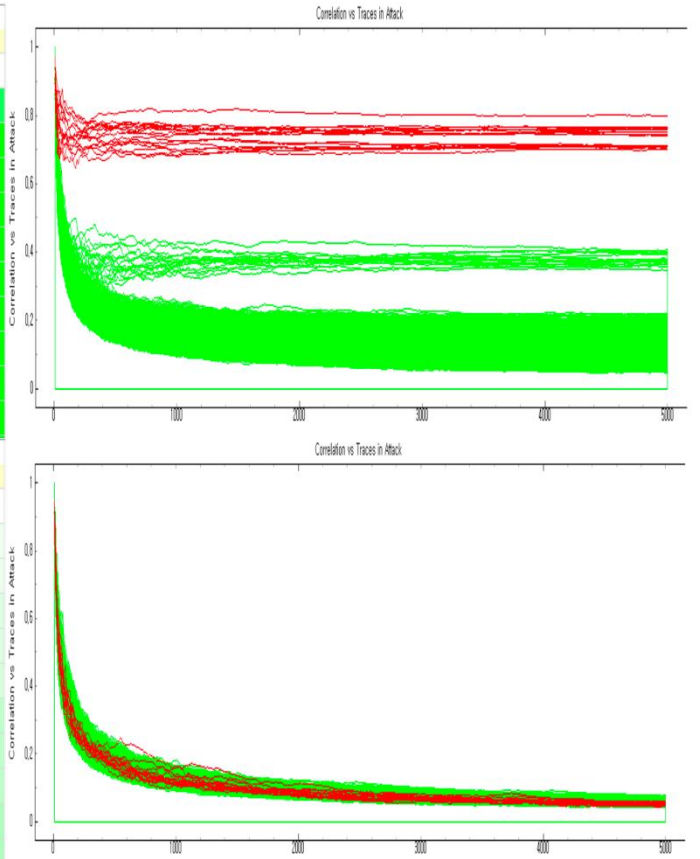


Fig 10. Graph of Power Analysis

Evaluation (vs LEA)

- ARX 연산을 사용하는 LEA에 비해 **매우 빠른 마스킹 속도**
- 기존의 Masked AES 연구 결과에 비해 성능 향상
  - **FACE-LIGHT, 소프트웨어 최적화**

Unit: Clock Cycles

LEA-128 [4]	Masked LEA-128 [5]	Masked AES-128 (Previous Work) [6]	Masked FACE-128 (Our Work)
2,688	36,589	25,970	**6,219**

Table 3. Comparison with LEA and Previous Work

04

Conclusion

F A C E - L I G H T

Contribution

- 저전력 프로세스 상에서 AES-CTR의 효과적인 최적화
  - Clock Cycles 최적화
- FACE에 비해 더 많은 Round 확장 가능
- Constant Timing을 통한 공격 지점 예측의 어려움
- 전력 분석 대응을 위한 Masking 적용
- AES의 경량화

Future Work

- 다양한 플랫폼 상에서의 최적화
  - 16bits MSP … ETC
- 다양한 국산 암호 제안 기법 적용
  - 사전 테이블 연산
  - 부채널 분석 내성
  - 소프트웨어 최적화
- 국제 저널 투고 시 SIMON/SPECK 등의 경량 암호와 비교

Fig 11. MSP430FR2433 LaunchPad kit

THANK YOU

Code (Header)

```
#ifndef AES_TYPES_H_
#define AES_TYPES_H_

#include <stdint.h>

typedef struct{
    uint8_t ks[16];
} aes_roundkey_t;

typedef struct{
    aes_roundkey_t key[10+1];
} aes128_ctx_t;

typedef struct{
    aes_roundkey_t key[12+1];
} aes192_ctx_t;

typedef struct{
    aes_roundkey_t key[14+1];
} aes256_ctx_t;

typedef struct{
    aes_roundkey_t key[1]; /* just to avoid the warning */
} aes_genctx_t;

#endif
```

Code (Macro)

```
/*****************************************************************************
*  MACRO SECTION                                                             *
*****************************************************************************/

.macro push_ p1:req, p2:vararg
    push \p1
.ifnb \p2
    push_ \p2
.endif
.endm

.macro pop_ p1:req, p2:vararg
    pop \p1
.ifnb \p2
    pop_ \p2
.endif
.endm

.macro push_range from:req, to:req
    push \from
.if     \to-\from
    push_range "(\from+1)",\to
.endif
.endm

.macro pop_range from:req, to:req
    pop \to
.if     \to-\from
    pop_range \from,"(\to-1)"
.endif
.endm
```

Code (Main)

```
u8 MaskingSBOX[256] = {0,};

const u8 SBOX[256] = {
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 };

#define MaskingPlainText    for(int i = 0; i < 16; i++)\
                                IV[i] ^= mask[(i%4)+6];

void aes128_init(const void *key, aes128_ctx_t *ctx);
void aes128_enc(void *buffer, aes128_ctx_t *ctx, void* mask);

void aes192_init(const void *key, aes192_ctx_t *ctx);
void aes192_enc(void *buffer, aes192_ctx_t *ctx, void *mask);

void aes256_init(const void *key, aes256_ctx_t *ctx);
void aes256_enc(void *buffer, aes256_ctx_t *ctx, void *mask);

void make_mask(void *mask, aes128_ctx_t *ctx, u8 round ,void *IV);
```

```
int main(void)
{
    u8 key[16] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
    aes256_ctx_t aes_test;
    aes256_init(key, &aes_test);
    int round = 14;

    while (1)
    {
        u8 IV[16] = {0,};
        u8 buffer[16] = {0,};
        u8 mask[10] = {0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x00,};

        make_mask(mask, &aes_test, round, IV);
        aes256_enc(IV,&aes_test, mask);

    }
}
```

Code (Table)

```
.balign 16
.global cachetable1
cachetable1:
.byte 0xdc, 0xa3, 0xda, 0x65, 0x3f, 0xd4, 0x62, 0xae, 0x6d, 0xd4, 0xfb, 0x58, 0xd2, 0x6f, 0xaf, 0x45

.balign 4096
.global cachetable2
cachetable2:
.byte 0xca, 0x65, 0x65, 0xaf, 0x6c, 0x6c, 0xb4, 0xd8, 0x34, 0x5c, 0x68, 0x34, 0xd5, 0x6f, 0xba, 0xba
.byte 0xc5, 0xef, 0xef, 0x2a, 0xd4, 0xd4, 0x67, 0xb3, 0x20, 0x60, 0x40, 0x20, 0xf7, 0x53, 0xa4, 0xa4
.byte 0x53, 0xa4, 0xa4, 0xf7, 0xf8, 0xf8, 0x13, 0xeb, 0x70, 0x90, 0xe0, 0x70, 0x2a, 0xc5, 0xef, 0xef
.byte 0x30, 0x18, 0x18, 0x28, 0x6d, 0x6d, 0xb7, 0xda, 0x62, 0xa6, 0xc4, 0x62, 0xf6, 0xa4, 0x52, 0x52
.byte 0xeb, 0xf8, 0xf8, 0x13, 0x77, 0x77, 0x99, 0xee, 0x8a, 0x85, 0xf, 0x8a, 0xe2, 0xbc, 0x5e, 0x5e
.byte 0x31, 0x95, 0x95, 0xa4, 0xb8, 0xb8, 0xd3, 0x6b, 0xaf, 0xea, 0x45, 0xaf, 0x42, 0x7c, 0x3e, 0x3e
.byte 0x3e, 0x1f, 0x1f, 0x21, 0xb1, 0xb1, 0xc8, 0x79, 0xc8, 0x43, 0x8b, 0xc8, 0xe0, 0x49, 0xa9, 0xa9
.byte 0x5a, 0x2d, 0x2d, 0x77, 0x18, 0x18, 0x28, 0x30, 0x98, 0xb3, 0x2b, 0x98, 0xcc, 0x88, 0x44, 0x44
.byte 0xec, 0x76, 0x76, 0x9a, 0xa6, 0xa6, 0xf1, 0x57, 0x8e, 0x89, 0x7, 0x8e, 0x8a, 0x5, 0x8f, 0x8f
.byte 0xc, 0x6, 0x6, 0xa, 0x7e, 0x7e, 0x82, 0xfc, 0x7b, 0x8d, 0xf6, 0x7b, 0x5f, 0x6a, 0x35, 0x35
.byte 0x64, 0x32, 0x32, 0x56, 0x13, 0x13, 0x35, 0x26, 0x76, 0x9a, 0xec, 0x76, 0x58, 0x99, 0xc1, 0xc1
.byte 0xe, 0x7, 0x7, 0x9, 0x62, 0x62, 0xa6, 0xc4, 0x8, 0x18, 0x10, 0x8, 0x2d, 0x36, 0x1b, 0x1b
.byte 0x8b, 0xc8, 0xc8, 0x43, 0x3a, 0x3a, 0x4e, 0x74, 0x75, 0x9f, 0xea, 0x75, 0xdc, 0x61, 0xbd, 0xbd
.byte 0xf4, 0x7a, 0x7a, 0x8e, 0xf4, 0xf4, 0x7, 0xf3, 0xf1, 0x8, 0xf9, 0xf1, 0x88, 0xf0, 0x78, 0x78
.byte 0x9c, 0x4e, 0x4e, 0xd2, 0x91, 0x91, 0xa8, 0x39, 0x5, 0xf, 0xa, 0x5, 0x9c, 0xe8, 0x74, 0x74
.byte 0xbd, 0xd3, 0xd3, 0x6e, 0x4f, 0x4f, 0xd1, 0x9e, 0xf0, 0xb, 0xfb, 0xf0, 0x5, 0x6, 0x3, 0x3
.byte 0xce, 0x67, 0x67, 0xa9, 0x8a, 0x8a, 0x85, 0xf, 0x1e, 0x22, 0x3c, 0x1e, 0x48, 0x70, 0x38, 0x38
.byte 0xd8, 0x6c, 0x6c, 0xb4, 0xe7, 0xe7, 0x32, 0xd5, 0x36, 0x5a, 0x6c, 0x36, 0x24, 0x38, 0x1c, 0x1c
```

Code (Masking)

```
#include "avr-asm-macros.S"

T0 = 18
T1 = 19
T2 = 20
T3 = 21
T4 = 22
T5 = 23

VAL0 = 24
VAL1 = 25
VAL2 = 26
xREDUCER = 27

/*
    mask      =   r24:25
    ks        =   r22:23
    round     =   r20:21
    plainText=   r18:19
*/

.global make_mask
make_mask:
    push r28
    push r29

    push_range 20, 23
    push r18
    push r19
    push r24
    push r25

    movw r30, r24
```

```
    ld  T0, Z
    ldd  T1, Z+1
    ldd  T2, Z+2
    ldd  T3, Z+3
    ldd  T4, Z+4
    ldd  T5, Z+5

    mov r28, T2
    mov r29, T3
    mov r30, T4
    mov r31, T5

    /*Mix*/
    mov r0, T4
    eor r0, T5
    mov VAL2, r0

    mov VAL0, T2
    eor T2, T3
    eor r0, T2
    lsl T2
    brcc 3f
    eor T2, xREDUCER
3:  eor T2, r0
    eor T2, VAL0

    mov VAL1, T3
    eor VAL1, T4
    lsl VAL1
    brcc 3f
    eor VAL1, xREDUCER
3:  eor VAL1, r0
    eor T3, VAL1
```

```
    lsl VAL2
    brcc 3f
    eor VAL2, xREDUCER
3:  eor VAL2, r0
    eor T4, VAL2

    eor VAL0, T5
    lsl VAL0
    brcc 3f
    eor VAL0, xREDUCER
3:  eor VAL0, r0
    eor T5, VAL0

    eor r28, T1
    eor r29, T1
    eor r30, T1
    eor r31, T1
    mov VAL0, r30
    mov VAL1, r31

    pop r31
    pop r30
    std Z+2, r28
    std Z+3, r29
    std Z+4, VAL0
    std Z+5, VAL1
    std Z+6, T2
    std Z+7, T3
    std Z+8, T4
    std Z+9, T5


    clr r0
```

Code (Masking)

```
    ldi r31, hi8(MaskingSBOX)
    ldi r29, hi8(SBOX)
    ldi xREDUCER, 0x1b
1:
    mov r28, r0
    ld  r26, Y
    eor r26, T1 // r26 = SBOX[i] ^ mask[1]

    mov r30, r0
    eor r30, T0

    st Z, r26

2:
    inc r0
    brne 1b

MASK1 = 19
MASK6 = 20
MASK7 = 21
MASK8 = 22
MASK9 = 23
HI = 26

    pop r31
    pop r30
    ldi HI, 0x4
1:
    ld r0, Z
    eor r0, MASK6
    st Z+, r0
```

```
    ld r0, Z
    eor r0, MASK7
    st Z+, r0

    ld r0, Z
    eor r0, MASK8
    st Z+, r0

    ld r0, Z
    eor r0, MASK9
    st Z+, r0

    dec HI
    brne 1b

    pop r31
    pop r30 // ks

    pop r26 // round
    pop r26

    lsl r26
    lsl r26

    eor MASK6, r18
    eor MASK7, r18
    eor MASK8, r18
    eor MASK9, r18
```

```
1:
    ld r0, Z
    eor r0, MASK6
    st Z+, r0

    ld r0, Z
    eor r0, MASK7
    st Z+, r0

    ld r0, Z
    eor r0, MASK8
    st Z+, r0

    ld r0, Z
    eor r0, MASK9
    st Z+, r0

    dec HI
    brne 1b

2:

    ldi HI, 0x10
3:
    ld r0, Z
    eor r0, MASK1
    st Z+, r0
    dec HI
    brne 3b
```

Code (Keyschedule)

```
 #include "avr-asm-macros.S"

.global aes256_init
aes256_init:
    movw r20, r22
    ldi r23, hi8(256)
    ldi r22, lo8(256)
    rjmp aes_init

.global aes192_init
aes192_init:
    movw r20, r22
    ldi r23, hi8(192)
    ldi r22, lo8(192)
    rjmp aes_init

.global aes128_init
aes128_init:
    movw r20, r22
    clr r23
    ldi r22, 128
```

```
SBOX_SAVE0 = 14
SBOX_SAVE1 = 15
XRC = 17
NK = 22
C1 = 18
NEXT_NK = 19
HI = 23
T0 = 20
T1 = 21
T2 = 24
T3 = 25
/*
 * param key:       r24:r25
 * param keysize_b: r22:r23
 * param ctx:       r20:r21
 */
.global aes_init
aes_init:
    push_range 14, 17
    push r28
    push r29
    movw r30, r20
    movw r28, r20
    movw r26, r24
    lsr r23
    ror r22
    lsr r22
    lsr r22 /* r22 contains keysize_b/8 */
    mov C1, r22
```

```
1:  /* copy key to ctx */
    ld r0, X+
    st Z+, r0
    dec C1
    brne 1b

    lsr NK
    lsr NK
    bst NK,3 /* set T if NK==8 */
    mov NEXT_NK, NK
    mov HI, NK
    subi HI, -7
    lsl HI
    lsl HI
    movw r26, r30
    sbiw r26, 4
    mov C1, NK
    ldi r31, hi8(SBOX)
    movw SBOX_SAVE0, r30
    ldi XRC, 1
1:
    ld T0, X+
    ld T1, X+
    ld T2, X+
    ld T3, X+
    cp NEXT_NK, C1
    breq 2f
    brtc 5f
    mov r16, C1
    andi r16, 0x07
    cpi r16, 0x04
    brne 5f
```

Code (Keyschedule)

```
    movw r30, SBOX_SAVE0
    mov r30, T0
    ld T0, Z

    mov r30, T1
    ld T1, Z

    mov r30, T2
    ld T2, Z

    mov r30, T3
    ld T3, Z

    rjmp 5f
2:
    add NEXT_NK, NK
    movw r30, SBOX_SAVE0
    mov r30, T0
    ld r16, Z

    mov r30, T1
    ld T0, Z


    mov r30, T2
    ld T1, Z

    mov r30, T3
    ld T2, Z

    mov T3, r16
    eor T0, XRC
    lsl XRC
    brcc 3f
```

```
    ldi XRC, 0x1b
3:
5:
    movw r30, r26

    ld r0, Y+
    eor r0, T0
    st Z+, r0
    ld r0, Y+
    eor r0 ,T1
    st Z+, r0
    ld r0, Y+
    eor r0, T2
    st Z+, r0
    ld r0, Y+
    eor r0, T3
    st Z+, r0

    inc C1
    cp C1, HI
    breq 6f
    rjmp 1b
6:

    clt
    pop r29
    pop r28
    pop_range 14, 17
    ret
```

Code (Encryption)

```
#include "avr-asm-macros.S"

/*
 * param a: r24
 * param b: r22
 * param reducer: r0
 */
A = 28
B = 29
P = 0
xREDUCER = 25

.global aes256_enc
aes256_enc:
    ldi r18, 12
    rjmp aes_encrypt_core

.global aes192_enc
aes192_enc:
    ldi r18, 10
    rjmp aes_encrypt_core

.global aes128_enc
aes128_enc:
    ldi r18, 8
```

```
MASK0 = 1
MASK1 = 2
MASK2 = 3
MASK3 = 4
T0= 5
T1= 6
T2= 7
ST00 =  8
ST01 =  9
ST02 = 10
ST03 = 11
ST10 = 12
ST11 = 13
ST12 = 14
ST13 = 15
ST20 = 16
ST21 = 17
ST22 = 18
ST23 = 19
ST30 = 20
ST31 = 21
ST32 = 22
ST33 = 23
CTR = 24
```

```
/*
 * param state:  r24:r25
 * param ks:     r22:r23
 * mask: r20:r21
 */
.global aes_encrypt_core
aes_encrypt_core:
    push_range 2, 17
    push r28
    push r29
    push r24
    push r25

    movw r26, r22
    mov  CTR, r18 // round
    clt

    adiw r26, 0x30
    ldi xREDUCER, 0x1b /* load reducer */

    movw r28, r20

    ld MASK0, Y

    ldi r31, hi8(cachetable3)
    ldi r30, lo8(cachetable3)
    .irp row, 0, 1, 2, 3
        .irp col, 0, 1, 2, 3
            lpm ST\row\col, Z+
            eor ST\row\col, MASK0
        .endr
    .endr
```

Code (Encryption)

```
    ldd MASK0, Y+2
    ldd MASK1, Y+3
    ldd MASK2, Y+4
    ldd MASK3, Y+5

    ldi r31, hi8(cachetable1)
    ldi r30, lo8(cachetable1)
    .irp row, 0, 1, 2, 3
        .irp col, 0, 1, 2, 3
            lpm T0, Z+
            eor ST\row\col, T0
        .endr
    .endr

    ldi r31, hi8(MaskingSBOX)
    brtc 2f

    /* key whitening */
1:
    .irp row, 0, 1, 2, 3
        .irp col, 0, 1, 2, 3
            ld r0, X+
            eor ST\row\col, r0
        .endr
    .endr
2:
    brtc 2f
exit:
    pop r31
    pop r30
```

```
    .irp row, 0, 1, 2, 3
        .irp col, 0, 1, 2, 3
            st Z+, ST\row\col
        .endr
    .endr

    pop r29
    pop r28
    pop_range 2, 17
    clr r1
    ret

2:  dec CTR
    brne 3f
    set
3:

    /* encryption loop */
    /* SBOX substitution and shifting */
    mov r30, ST00
    ld ST00, Z
    mov r30, ST10
    ld ST10, Z
    mov r30, ST20
    ld ST20, Z
    mov r30, ST30
    ld ST30, Z

    mov r30, ST01
    ld T0, Z
    mov r30, ST11
    ld ST01, Z
    mov r30, ST21
    ld ST11, Z
```

```
mov r30, ST31
ld ST21, Z
mov ST31, T0

mov r30, ST02
ld T0, Z
mov r30, ST12
ld T1, Z
mov r30, ST22
ld ST02, Z
mov r30, ST32
ld ST12, Z
mov ST22, T0
mov ST32, T1

mov r30, ST03
ld T0, Z
mov r30, ST33
ld ST03, Z
mov r30, ST23
ld ST33, Z
mov r30, ST13
ld ST23, Z
mov ST13, T0

brtc 2f
rjmp 1b
```

Code (Encryption)

```
2:
 /* mix */
    eor ST00, MASK0
    eor ST01, MASK1
    eor ST02, MASK2
    eor ST03, MASK3

    eor ST10, MASK0
    eor ST11, MASK1
    eor ST12, MASK2
    eor ST13, MASK3

    eor ST20, MASK0
    eor ST21, MASK1
    eor ST22, MASK2
    eor ST23, MASK3

    eor ST30, MASK0
    eor ST31, MASK1
    eor ST32, MASK2
    eor ST33, MASK3

 .irp row, 0, 1, 2, 3
    mov r0, ST\row\()2
    eor r0, ST\row\()3
    mov T2, r0

    mov T0, ST\row\()0
    eor ST\row\()0, ST\row\()1
    eor r0, ST\row\()0
    lsl ST\row\()0
    brcc 3f
    eor ST\row\()0, xREDUCER
3:  eor ST\row\()0, r0
    eor ST\row\()0, T0

    mov T1, ST\row\()1
    eor T1, ST\row\()2
    lsl T1
    brcc 3f
    eor T1, xREDUCER
3:  eor T1, r0
    eor ST\row\()1, T1

    lsl T2
    brcc 3f
    eor T2, xREDUCER
3:  eor T2, r0
    eor ST\row\()2, T2

    eor T0, ST\row\()3
    lsl T0
    brcc 3f
    eor T0, xREDUCER
3:  eor T0, r0
    eor ST\row\()3, T0

 .endr
    /* mixcolums done */

    /* add key*/
    rjmp 1b
```

References

[1] C. Herbst, E. Oswald, and S. Mangard, "An aes smart card implementation resistant to power analysis attacks," in International conference on applied cryptography and network security, pp. 239–252, Springer, 2006.

[2] D. Dinu, A. Biryukov, J. Großsch ¨adl, D. Khovratovich, Y. Le Corre, and L. Perrin,"FELICS–fair evaluation of lightweight cryptographic systems," inNIST Workshopon Lightweight Cryptography, vol. 128, 2015.

[3] D. Otteet al., "AVR-crypto-lib,"Online: http://www. das–labor. org/wiki/AVR–Crypto–Lib/en, 2009.

[4] H. Seo, I. Jeong, J. Lee, and W. Kim, "Compact implementations of ARX-based block ciphers on IoT processors," ACM TECS, 2018.

[5] E. Park, S. Oh, and J. Ha, "Masking-based block cipher LEA resistant to side channel attacks," KIISC, 2017.

[6] K. H. Kim, H. J. Seo, "Implementation of Optimized 1st-Order Masking AES Algorithm Against Side-Channel-analysis," KIPS, 2019.