

암호학과 네트워크 보안

-9장 암호수학-

김민식 (ms6127@naver.com)

세종대학교 프로토콜공학연구실

목 차

- 보충
- 기초 개념
- 소수
- 소수 기반 정리
- 소수 판정
- 소인수분해

보충

- AES (Advanced Encryption Standard)

- 정의

- 128비트 블록 단위의 데이터를 암호화하는 대칭 키 블록 암호 알고리즘임

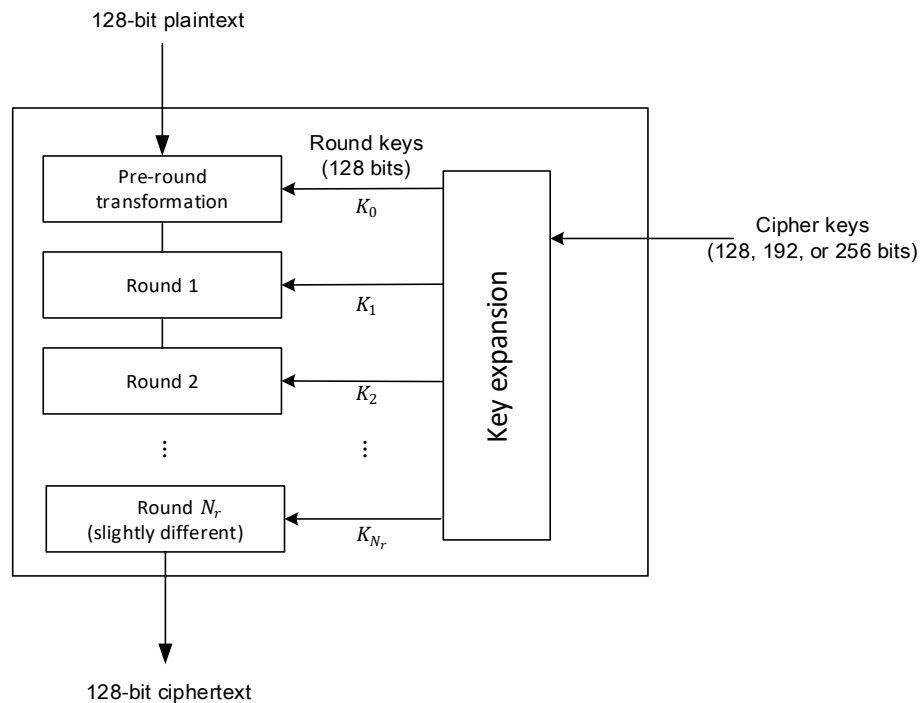
- 특징

- 128비트 평문 블록을 128비트 암호문으로 변환함
- 키 길이에 따라 AES-128, AES-192, AES-256으로 구분됨
- Feistel 구조가 아닌 SPN 구조를 기반으로 함
 - SubBytes, ShiftRows, MixColumns, AddRoundKey를 반복 수행함

보충

• AES 구조

- AES는 128비트 블록을 처리하는 SPN 구조의 대칭키 블록 암호 알고리즘임
- 생성한 라운드 키의 수는 총 라운드 수보다 하나 더 많음
 - 라운드 키의 수 = $N_r + 1$



N_r	Key size
10	128
12	192
14	256

AES는 128, 192, 256비트 키를 사용하고 키 길이에 따라 각각 10, 12, 14 라운드를 갖는 3가지 버전이 있음

보충

- AES 데이터 단위

- 비트(Bit)

- 0 또는 1을 나타내는 가장 작은 데이터 단위임
- 표현 시, 소문자로 표현
 - e.g., b_1, b_2

- 바이트(Byte)

- 8개의 비트로 구성된 단위임
- 비트를 원소로 하는 행 행렬(1×8) 또는 열 행렬(8×1)로 표현
- 표현 시, 볼드 체 문자로 표현
 - e.g., **b**, **B**

$$\begin{array}{l} \boxed{\text{Byte}} = [b_0 \quad b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5 \quad b_6 \quad b_7] = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \\ \text{B, b} \qquad \qquad \qquad \text{B, b} \\ \text{Byte} \qquad \qquad \qquad \text{B, b} \end{array}$$

보충

- AES 데이터 단위

- 워드(Word)

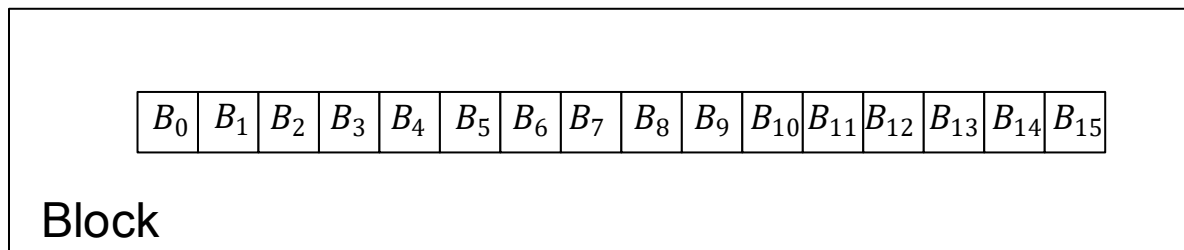
- 4개의 바이트로 구성된 32비트 단위임
- 바이트를 원소로 하는 행 행렬 (1×4)
또는 열 행렬(4×1)로 표현
- 표현 시, 볼드 체의 소문자 **w** 사용
 - e.g., **w**

$$\begin{array}{c} \boxed{\text{Word}} \\ \mathbf{w} \end{array} = [B_0 \quad B_1 \quad B_2 \quad B_3] = \begin{array}{c} [B_0 \\ B_1 \\ B_2 \\ B_3] \\ \mathbf{w} \end{array}$$

Word

- 블록(Block)

- 128비트(16바이트)로 이루어진 단위
- 16바이트를 원소로 하는 행 행렬(1×16)로 표현



보충

- AES 데이터 단위

- 상태(State)

- 16바이트를 원소로 가지는 4×4 행렬로 표현
 - 상태의 각 원소는 $s_{r,c}$ 로 표현함(r 은 행으로, c 는 열로서 0에서 3까지 사용함)
- 각 열 4바이트를 하나의 워드로 볼 수 있음
 - State는 4개의 워드 w_0, w_1, w_2, w_3 로 표현 가능함
- 표현 시, S 라 나타내며 임시 상태일 시에는 T 로 표현

$$\mathbf{S} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = [w_0 \quad w_1 \quad w_2 \quad w_3]$$

State

보충

- AES 데이터 단위

- 예제 7.1

16개의 문자로 이루어진 블록이 어떻게 4×4 행렬로 표현되는지 살펴본다.
주어진 문자가 “AES uses a matrix” 라고 가정하자

- 마지막에 가짜 문자를 추가하여 16개의 문자를 얻음

A E S U S E S A M A T R I X Z Z

- 각 문자를 00과 25 사이의 정수로 나타내고 이를 16진수로 표현함

00 04 12 14 12 04 12 00 0C 00 13 11 08 23 19 19

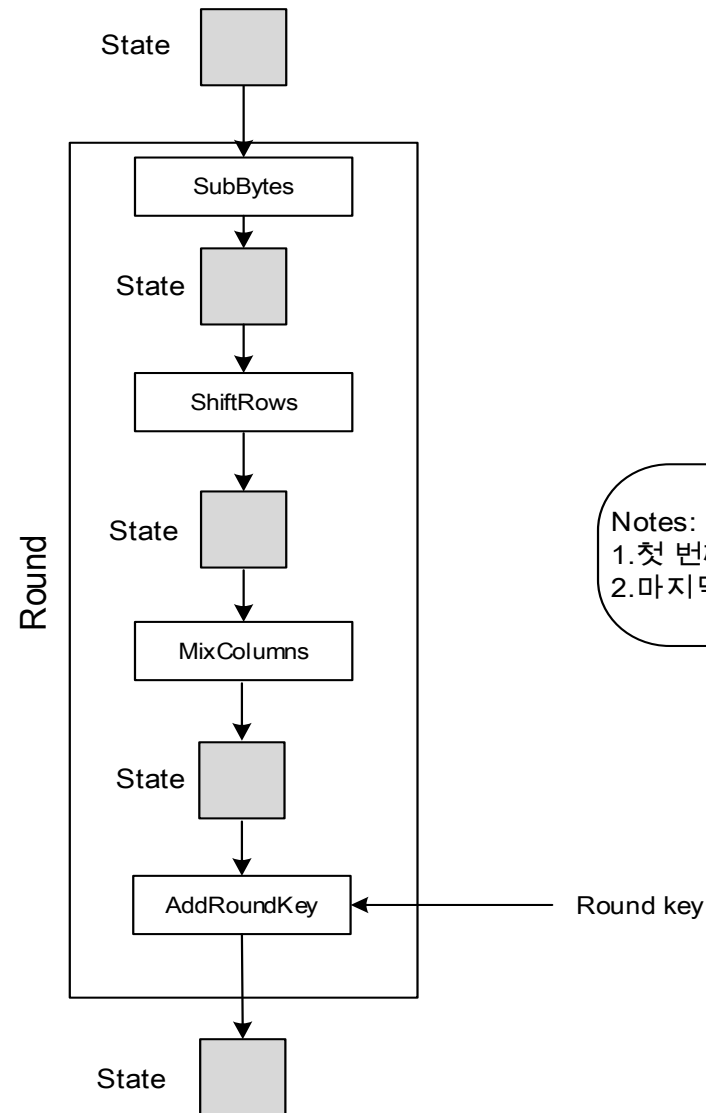
- 왼쪽 문자열부터 한 열씩 채움

$$\begin{bmatrix} 00 & 12 & 0C & 08 \\ 04 & 04 & 00 & 23 \\ 12 & 12 & 13 & 19 \\ 14 & 00 & 11 & 19 \end{bmatrix}$$

보충

- AES 라운드(Round)

- 구조



Notes:

1. 첫 번째 라운드 전에 AddRoundKey가 한 번 적용됨
2. 마지막 라운드에서는 세 번째 변환이 빠짐

- AES 라운드

- AES의 라운드 함수는 4가지 형태의 변환을 사용함

- 변환

- 대치(Substitution)

- 입력 값을 정해진 규칙에 따라 다른 값으로 바꾸는 변환
 - SubBytes, InvSubBytes

- 치환(Permutation)

- 데이터 값 자체는 바꾸지 않고, 위치만 재배열하는 변환
 - ShiftRows

- 뒤섞음(Mixing)

- 여러 바이트 값을 서로 섞어서, 한 바이트의 변화가 여러 바이트에 영향을 주도록 만드는 변환
 - MixColumns, InvMixColumns

- 키 덧셈(Key-Adding)

- 상태(State) 배열과 라운드 키(Round Key)를 XOR 연산하여 결합하는 변환
 - AddRoundKey

- AES 라운드

- 대치(Substitution)

- SubBytes (1/5)

- 정의

- State의 각 바이트를 S-box를 이용해 다른 바이트 값으로 대치하는 변환임

- 특징

- 16개의 바이트에 독립적으로 적용됨
 - 바이트의 위치는 변하지 않고 값만 변경됨
 - 모든 바이트 변환에 하나의 S-box 테이블을 사용함
 - 비선형 변환으로 혼돈 효과를 제공함
 - 복호화에서는 역변환인 InvSubBytes를 사용함

보충

- AES 라운드

- 대치(Substitution)

- SubBytes (2/5)

- SubBytes 변환 표 참조 과정

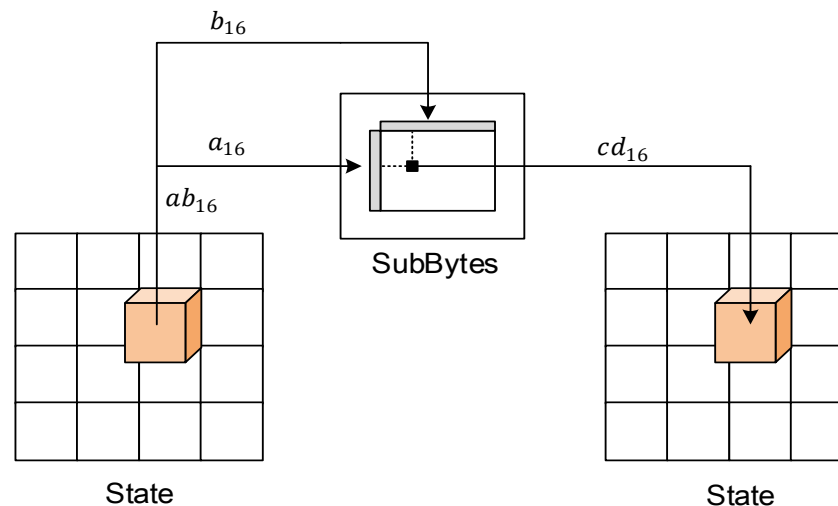
- 1. 입력 바이트를 16진수 두 자리로 표현함

- ab_{16}

- 2. 앞자리 a 는 S-Box의 행을 선택하고 뒷 자리 b 는 S-Box의 열을 선택함

- 3. 선택된 행과 열이 만나는 값 cd_{16} 이 새로운 출력 바이트가 됨

- 4. 변환된 바이트는 State의 같은 위치에 저장됨



보충

- AES 라운드

- 대치(Substitution)

- SubBytes (3/5)

- SubBytes 변환 표 참조 과정

<SubBytes 변환표>

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	CB	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	E8	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

보충

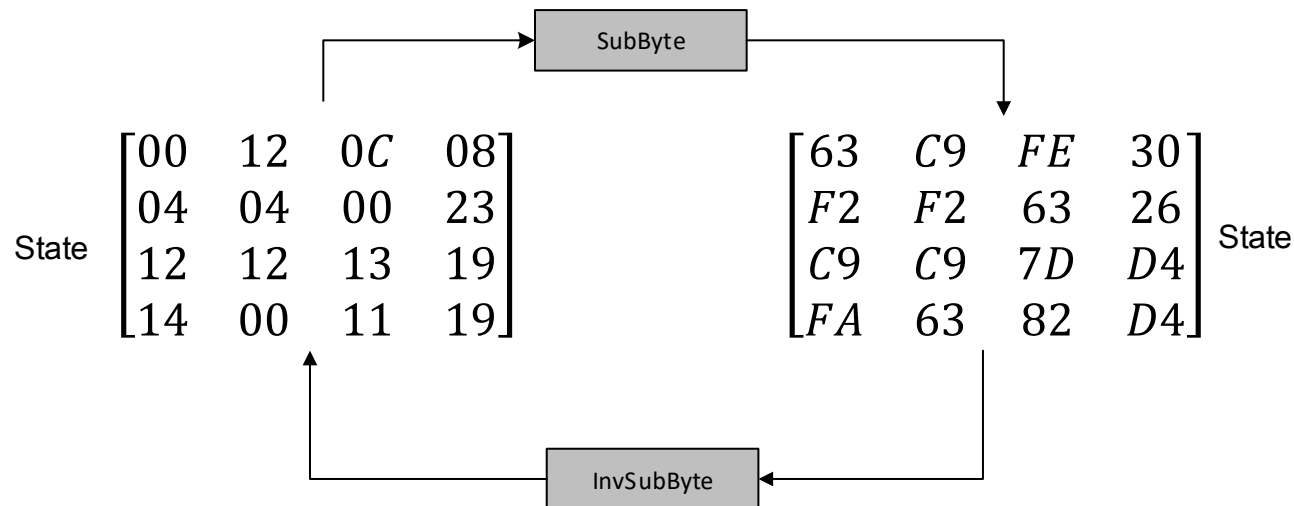
- AES 라운드

- 대치(Substitution)

- SubBytes (4/5)

- SubBytes 변환 표 참조 과정

- InvSubBytes를 통해서 원래의 값으로 복원할 수 있음



보충

• AES 라운드

• 대치(Substitution)

• SubBytes (5/5)

• $GF(2^8)$ 을 이용한 변환 방법

• 변환 원리

- 입력 바이트를 $GF(2^8)$ 의 원소로 보고 역원을 구함
- 구한 역원에 아핀 변환을 적용함
- 최종 8비트 값이 SubBytes 출력 값이 됨

• 구성 요소

- $S_{r,c}$: State의 r 행 c 열에 위치한 입력 바이트임
- $(S_{r,c})^{-1}$: 입력 바이트의 곱셈 역원임
- X : AES에서 정해진 고정 행렬임
- y : AES에서 정해진 고정 벡터임
- d : SubBytes 변환 후 생성되는 출력 바이트임

• 관계식

- SubByte: $d = X(S_{r,c})^{-1} \oplus y$
- InvSubByte: $[X^{-1}(d \oplus y)]^{-1} = [X^{-1}(X(S_{r,c})^{-1} \oplus y \oplus y)]^{-1} = S_{r,c}$

*아핀 변환: 역원으로 바뀐 값을 AES가 정한 방식으로 한 번 더 섞어 최종 S-box 출력 값을 만드는 과정임

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

고정 행렬 X

$$\begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

고정 벡터 y

- AES 라운드
 - 대치(Substitution)
 - 부분바이트 (SubBytes)
 - 예제 7.3

16진수 $0C$ 가 SubByte 통해 FE 로 계산되는 과정을 계산해라

- SubBytes
 1. $GF(2^8)$ 체에서 $0C_{16}(00001100_2)$ 의 곱셈에 대한 역원을 구함
 $(0C_{16})^{-1} = B0_{16} = 10110000_2$
 2. 행렬 X 와 곱셈 연산 후 값은 10011101_2 이 됨
 3. XOR 연산 후의 값은 $d = 11111110_2$ 이 됨, 이는 FE_{16} 의 비트 단위 표현임

- AES 라운드

- 대치(Substitution)
 - 부분바이트 (SubBytes)
 - 예제 7.3

InSubByte를 통해 다시 0C로 계산하는 과정을 설명하라(역부분바이트 관계식)

- InvSubBytes
 1. XOR 연산 후의 값으로 10011101_2 임
 2. 행렬 X^{-1} 를 곱한 이후 값은 10110000_2 또는 B0가 됨
 3. B0의 곱셈에 대한 역원은 0C가 됨

보충

• AES 라운드

• 치환(Permutation) 또는 이동 연산(Shifting)

• ShiftRows

• 정의

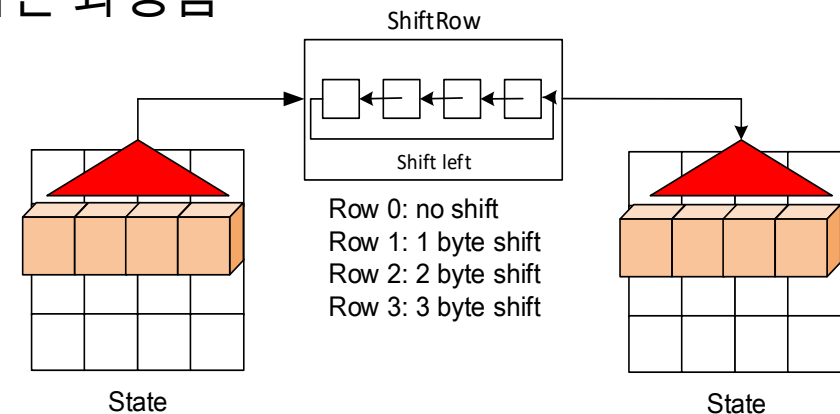
- State의 각 행을 왼쪽으로 순환 이동하는 과정임

• 이동 방식

- 0번째 행: 이동하지 않음
- 1번째 행: 왼쪽으로 1바이트 이동함
- 2번째 행: 왼쪽으로 2바이트 이동함
- 3번째 행: 왼쪽으로 3바이트 이동함

• 특징

- 바이트 값은 변하지 않고 행렬 상의 위치만 이동함
- 이동된 바이트는 사라지지 않고 반대쪽 끝으로 이동함
- 바이트의 위치를 재배열하여 한 열의 값들이 여러 열로 퍼지게 하기 때문에 확산성을 확보함
- InvShiftRows는 복호화 과정에서 사용하고 오른쪽으로 이동을 수행함
 - ShiftRows와 InvShiftRows는 서로 역관계임

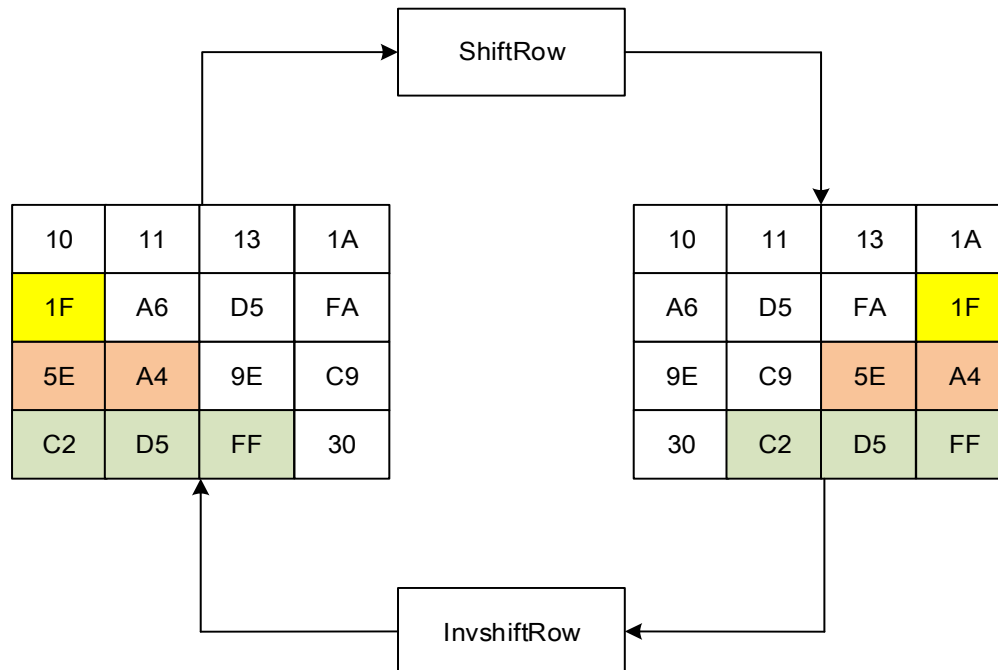


보충

- AES 라운드

- 치환(Permutation) 또는 이동 연산(Shifting)
 - 예제 7.4

다음 상태(State)에 대해 ShiftRows 변환과 InvShiftRows 변환을 수행하시오.



보충

- AES 라운드(Round)

- 뒤섞음(Mixing)

- MixColumns

- 정의

- State의 각 열을 고정 행렬과 곱하여 새로운 열로 변환하는 과정임

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

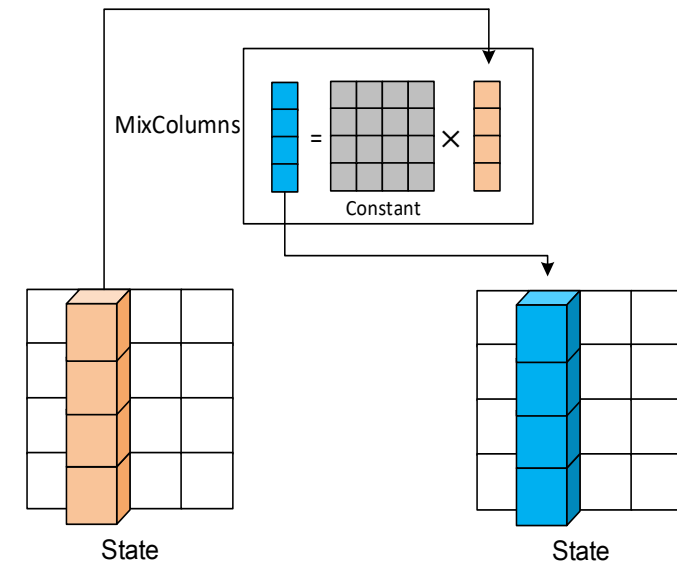
AES 고정 행렬

- 변환 방식

- 각 열을 AES에서 정한 고정 행렬과 곱함
 - 이때 곱셈은 $GF(2^8)$ 에서 수행되며, 덧셈은 XOR로 처리함

- 특징

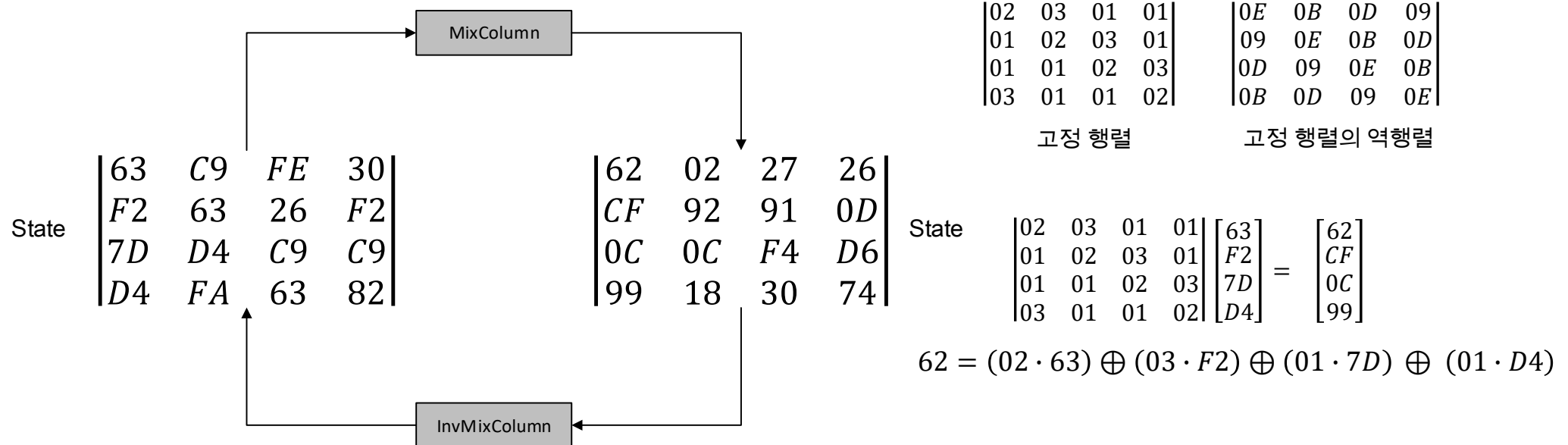
- AES의 확산 효과를 높이는 역할임
 - 키나 평문에 따라 고정 행렬은 바뀌지 않음
 - InvMixColumns는 MixColumns의 역변환임
 - 암호화 과정에서 섞인 State의 각 열을 복호화 과정에서 다시 되돌리는 변환임



보충

- AES 라운드
 - 뒤섞음(Mixing)
 - MixColumns

다음 상태(State)를 MixColumns 변환과 InvMixColumns 변환을 수행하시오.



보충

• AES 라운드

• 키 덧셈(Key-Adding)

• AddRoundKey

• 정의

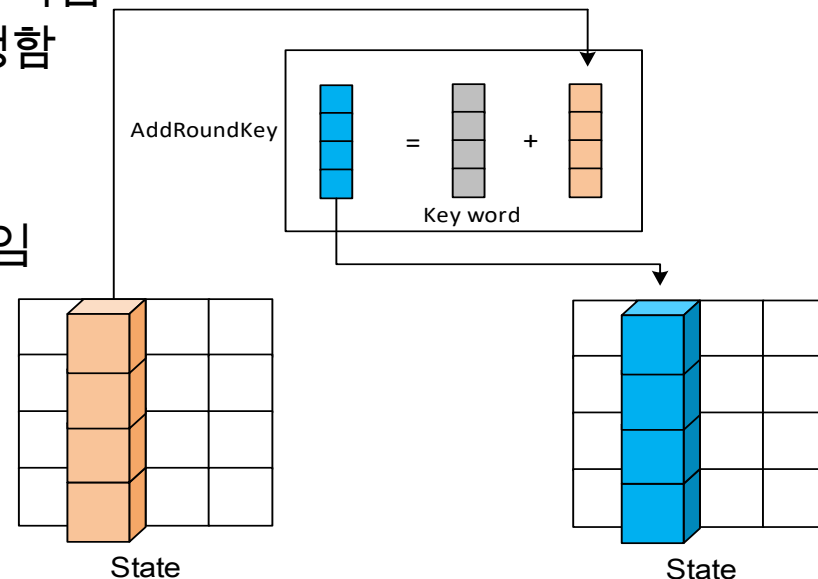
- State와 라운드 키를 XOR하여 키를 결합하는 변환임

• 변환 과정

- 키 확장으로 생성된 라운드 키를 사용함
- State의 각 열과 라운드 키 word를 대응시킴
- 대응되는 바이트끼리 XOR 연산을 수행함
- XOR 결과가 새로운 State 값이 됨

• 특징

- 실제 암호 키가 State에 반영되는 단계임
- XOR은 자기 자신의 역연산 이므로 복호화에서도 같은 연산을 사용함



보충

- AES 암호화 과정

- (1단계)

- 평문 블록을 State 배열로 초기화

- (2단계)

- 초기 암호 키를 여러 라운드 키로 확장

- (3단계)

- 라운드 수행

- SubBytes
 - ShiftRows
 - MixColumns
 - AddRoundKey

- AES 암호화 과정

- (1단계) State 초기화

- 128비트 평문 블록을 4×4 State 배열로 변환하는 과정임
- AES는 128비트(16바이트) 단위로 데이터를 처리함
- 16바이트를 4×4 행렬 형태로 배치함
- 왼쪽부터 한 열씩 채워 구성함

$$\text{State} = \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix}$$

보충

- AES 암호화 과정

- (2단계)키 확장

- $n(128, 192, 256)$ 비트 암호 키로부터 $N_r + 1$ 개의 n 비트 라운드 키를 생성(N_r 은 라운드 수)
- 워드가 4개의 바이트들로 이루어진다고 할 때, 각 라운드는 워드 단위로 라운드 키를 생성함
 - $4 \times (N_r + 1)$ 개의 워드를 생성함
 - e.g., AES-128는 라운드 수가 10임($N_r = 10$)
필요한 워드 수는 $4 \times (10 + 1) = 44$

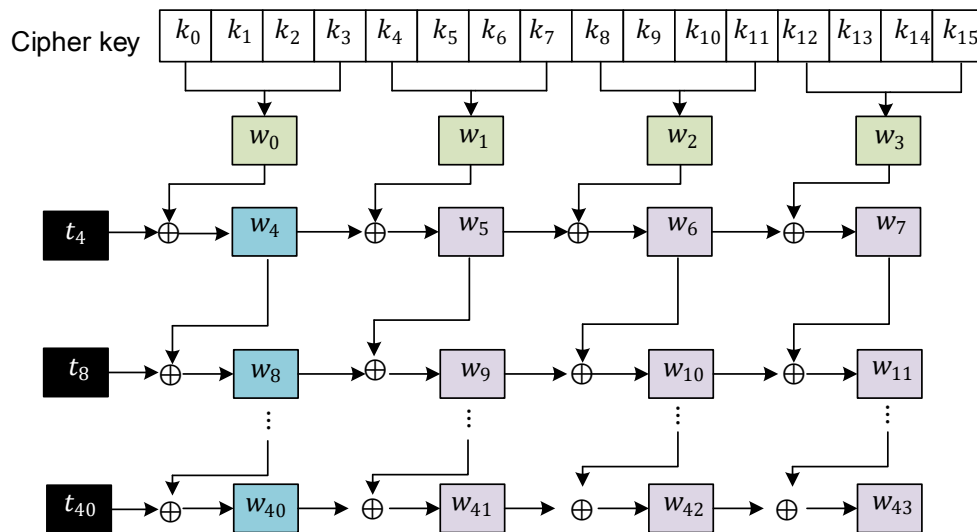
Round	Words
Pre-Round	$w_0 \ w_1 \ w_2 \ w_3$
1	$w_4 \ w_5 \ w_6 \ w_7$
2	$w_8 \ w_9 \ w_{10} \ w_{11}$
...	...
N_r	$w_{4N_r} \ w_{4N_r+1} \ w_{4N_r+2} \ w_{4N_r+3}$

보충

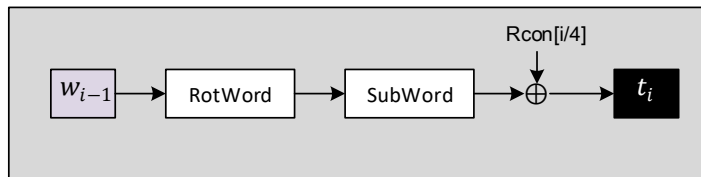
• AES 암호화 과정

• (2단계)키 확장

- AES-128은 총 44개의 워드를 생성함



- 처음 네 개의 워드 w_0, w_1, w_2, w_3 는 암호 키로 부터 만들어짐
- 암호 키를 16개의 바이트 k_0 부터 k_{15} 까지 구성된 배열로 간주함
- w_0, w_1, w_2, w_3 를 하나로 연결하면 이것이 암호 키임
- $i \bmod 4 \neq 0$ 일 경우
 - $w_i = w_{i-1} \oplus w_{i-4}$
- $i \bmod 4 = 0$ 일 경우
 - $w_i = t \oplus w_{i-4}$
 - $t = \text{SubWord}(\text{RotWord}(w_{i-1})) \oplus \text{RCon}_{i/4}$



- AES 암호화 과정

- (2단계) 키 확장

- RotWord

- 워드 안의 바이트 위치를 왼쪽으로 한 칸 순환 이동하는 연산임

- SubWord

- RotWord 결과의 각 바이트를 S-box를 이용해 다른 값으로 바꾸는 연산임

- Rcon

- 라운드마다 넣어주는 고정 상수값임
 - RC_i 라고 불리는 라운드 상수는 4바이트 값이며, 표 또는 체 $GF(2^8)$ 를 이용하여 상수들의 최상위 바이트들을 계산하여 얻을 수 있음
 - RC_i 라 불리는 라운드 상수의 최상위 바이트들은 x^{i-1} 로 표현되고 i 는 라운드 수를 뜻하고 기약다항식 $x^8 + x^4 + x^3 + 1$ 를 사용함

보충

- AES 암호화 과정

- (2단계) 키 확장

- Rcon

- e.g ., AES-128 키

$$RC_1 \rightarrow x^{1-1} = x^0 \bmod \text{prime} = 1 \rightarrow 00000001 \rightarrow 01_{16}$$

$$RC_5 \rightarrow x^{5-1} = x^4 \bmod \text{prime} = x^4 \rightarrow 00010000 \rightarrow 10_{16}$$

$$RC_9 \rightarrow x^{9-1} = x^8 \bmod \text{prime} = x^4 + x^3 + x + 1 \rightarrow 0001101 \rightarrow 1B_{16}$$

Round	RCon	Round	RCon
1	(01 00 00 00) ₁₆	6	(20 00 00 00) ₁₆
2	(02 00 00 00) ₁₆	7	(40 00 00 00) ₁₆
3	(03 00 00 00) ₁₆	8	(80 00 00 00) ₁₆
4	(08 000000) ₁₆	9	(1B 00 00 00) ₁₆
5	(10 000000) ₁₆	10	(36 00 00 00) ₁₆

- AES 암호화 과정

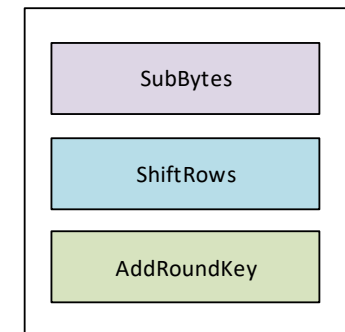
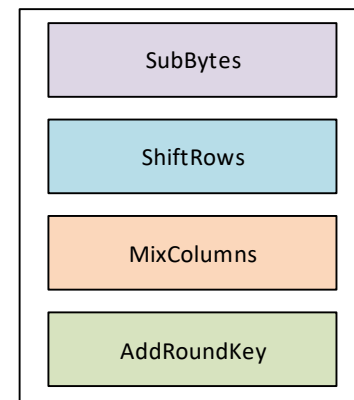
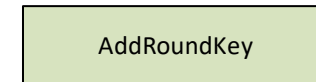
- (2단계) 키 확장

- 키 확장 과정 특징

1. 공격자가 암호 키 일부나 라운드 키의 일부 값을 얻는다고 해도 모든 라운드 키를 알기 위해서는 남은 암호 키를 복구해야함
 - SubWord의 비선형성 때문임
2. 두 개의 암호 키가 오직 한 비트만 다른 값을 갖더라도 라운드 키 확장 과정을 통해 적은 수의 라운드 만에 서로 다른 라운드 키를 생성함
3. 암호 키의 각 비트는 여러 라운드에 영향을 줌
 - 한 비트를 바꿀 경우 여러 라운드에서 여러 비트가 바뀌게 됨
4. DES와 달리 AES에서는 심각한 취약 키가 존재하지 않음
 - S-박스의 비선형성이나 ShiftRows와 MixColumns 단계에서 데이터의 비트와 바이트 단위로 섞기 때문임

보충

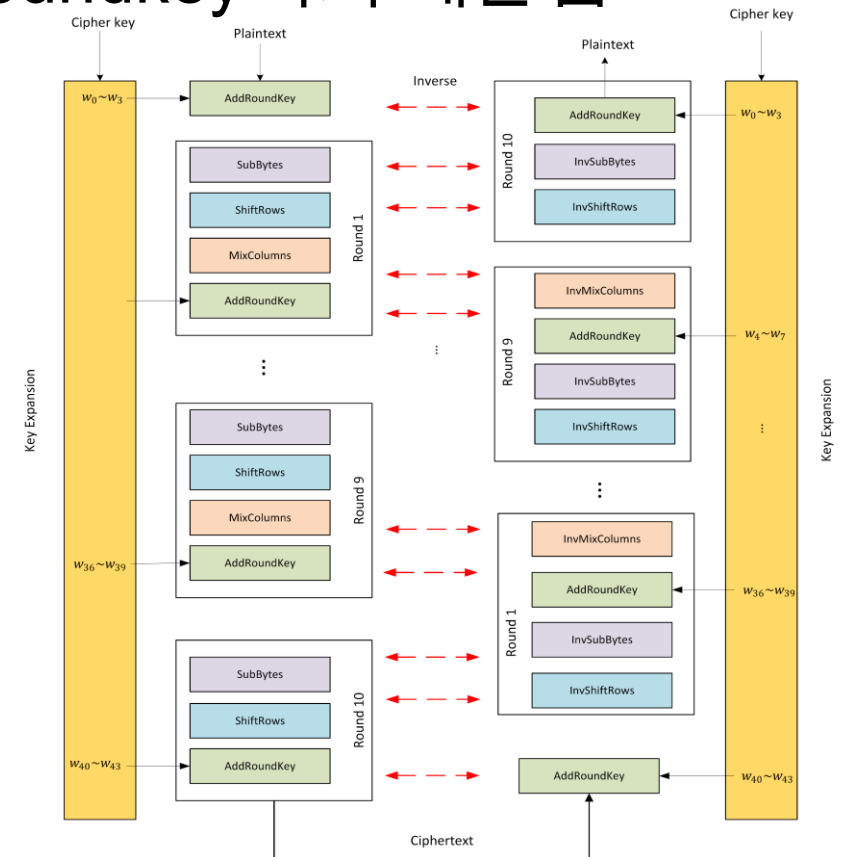
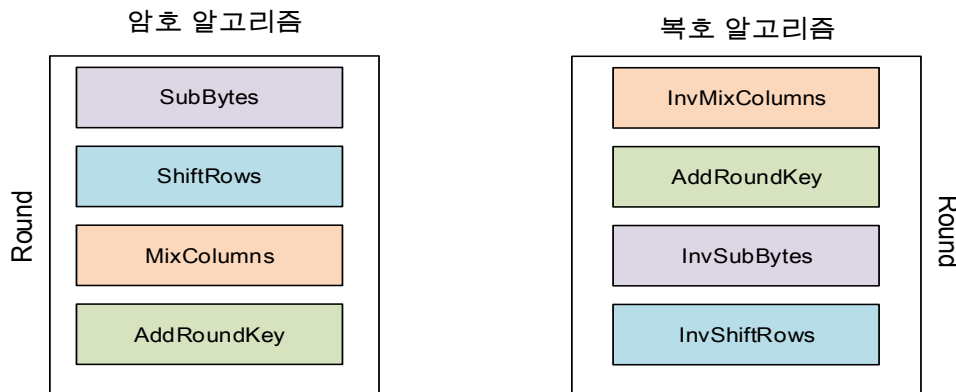
- AES 암호화 과정
 - (3단계)라운드 수행
 - 초기 라운드
 - AddRoundKey
 - 일반 라운드
 - SubBytes
 - ShiftRows
 - MixColumns
 - AddRoundKey
 - 최종 라운드
 - SubBytes
 - ShiftRows
 - AddRoundKey



보충

• AES 복호화 과정

- 라운드 키는 암호화와 반대로 역순으로 사용함
- 암호화의 마지막 라운드 키로 AddRoundkey를 수행함
 - 암호화의 마지막 단계가 AddRoundkey이기 때문임
- 이후 중간 라운드에서는 아래 그림과 같은 순서로 수행함



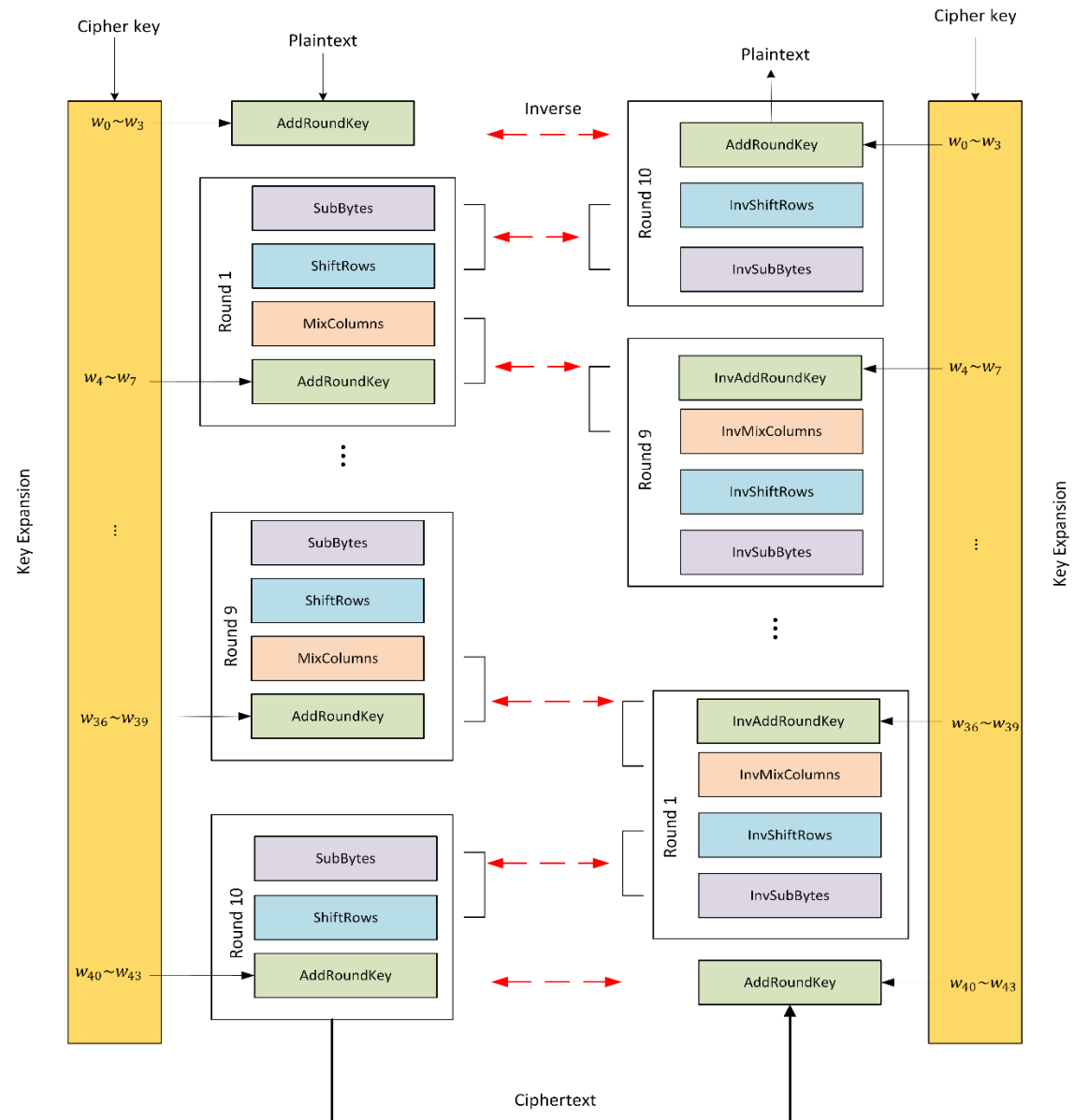
- AES 구조

- 대체 설계 버전

- 복호화 과정을 암호화 과정과 유사한 구조로 표현하기 위한 구현 방식임
- 보안성 향상 목적이 아니라, 암호화/복호화 구조를 통일하고 구현을 효율적으로 하기 위해 사용함
 - 원래 버전
 - 암호화: $SubBytes \rightarrow ShiftRows \rightarrow MixColumns \rightarrow AddRoundKey$
 - 복호화: $AddRoundKey \rightarrow InvMixColumns \rightarrow InvShiftRows \rightarrow InvSubBytes$
 - 대체 설계 버전
 - 암호화: 원래 버전과 동일함
 - 복호화:
 $InvSubBytes \rightarrow InvShiftRows \rightarrow InvMixColumns \rightarrow InvAddRoundKey$

보충

- AES 구조
- 대체 설계 버전

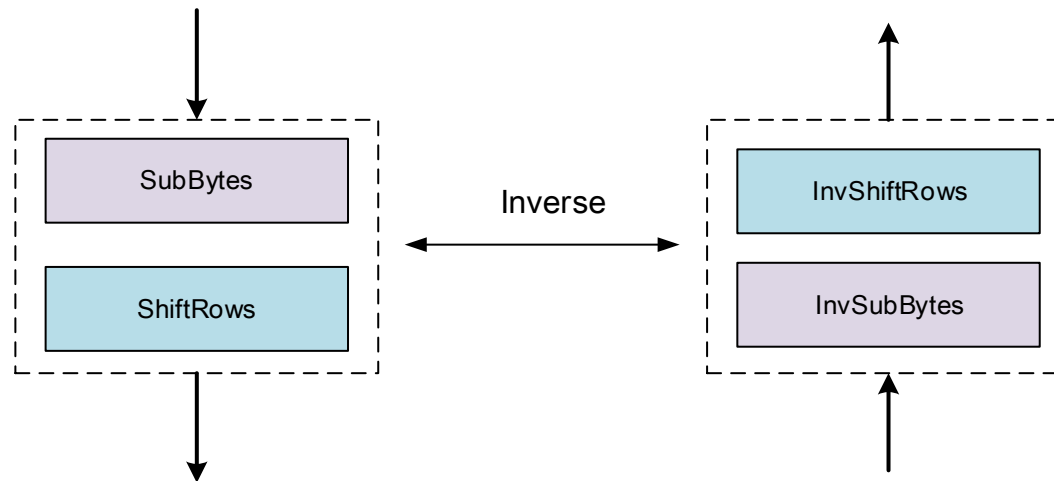


- AES 구조

- 대체 설계 버전

- SubBytes와 ShiftRow

- SubBytes는 바이트의 순서를 변경하지 않고 각 바이트의 값을 변경함
- ShiftRows에서 바이트 값을 변경하지 않고 순서만 변경함
- 서로 독립적이기 때문에 변경 가능함

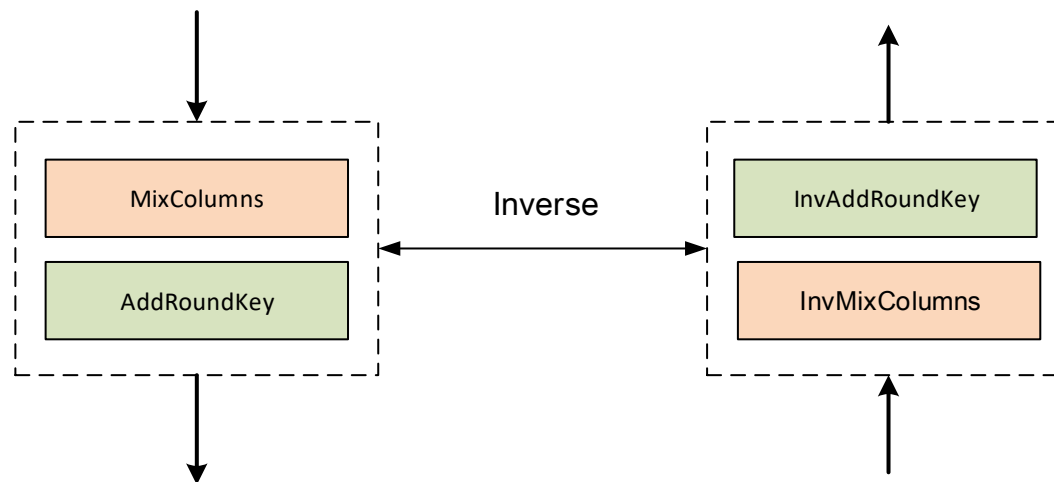


- AES 구조

- 대체 설계 버전

- MixColumns과 AddRoundKey

- MixColumns와 AddRoundKey에서 사용된 상수 행렬의 역행렬을 키 행렬에 곱한다면, 서로 역함수 관계가 되는데 이렇게 생성된 새로운 변환을 InvAddRoundKey라고 함
- MixColumns가 선형 변환이므로, 라운드 키에도 InvMixColumns를 적용하면 두 연산의 순서를 바꿔 표현할 수 있음



- AES 안정성

- 전수조사 공격

- AES는 128, 192, 256비트 키를 사용함
 - DES보다 훨씬 큰 키 공간을 가지므로 전수조사 공격에 강함

- 통계적 공격

- SubBytes, ShiftRows, MixColumns의 조합으로 혼돈과 확산을 제공함
 - 평문과 암호문 사이의 패턴 관계를 복잡하게 만들어 통계적 패턴 분석이 어려움

- 차분 공격과 선형 공격

- DES 이후 알려진 공격을 고려하여 설계되었고 비선형 S-box와 강한 확산 구조를 사용함
 - 현재까지 실용적인 공격은 알려져 있지 않음

목 차

- 보충
- 기초 개념
- 소수
- 소수 판정
- 소인수분해

기초 개념

- 양의 정수

- 정의

- 0보다 큰 정수이며, 자연수와 같은 의미로 사용됨

- 분류

- 1

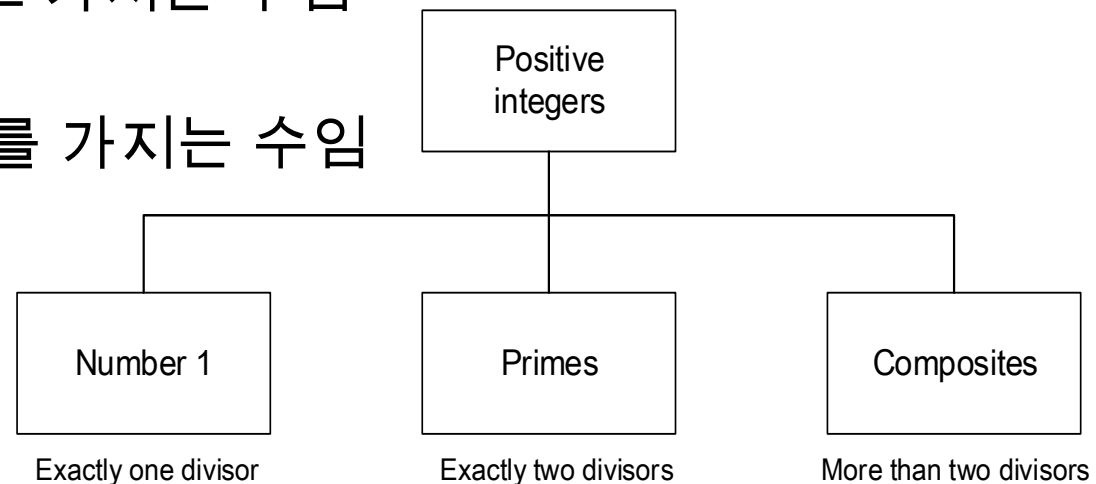
- 소수도 합성수도 아님

- 소수

- 1과 자기 자신만을 약수로 가지는 수임

- 합성수

- 1과 자기 자신 외의 약수를 가지는 수임



기초 개념

- 서로 소(Relatively Prime, Coprime)

- 정의

- 두 정수의 최대공약수가 1인 관계임

- 특징

- 1 이외의 공통 약수를 가지지 않음
- p 가 소수라면 1부터 $p - 1$ 까지의 모든 정수는 p 와 서로 소가 됨
 - e.g ., $p = 7$ 이면 1, 2, 3, 4, 5, 6은 모두 7과 서로 소

목 차

- 보충
- 기초 개념
- 소수
- 소수 판정
- 소인수분해

소수

- 집합의 크기

- 무한히 많은 소수

- 증명

- 소수의 개수가 유한개라고 가정
 - 가장 큰 소수를 p 라고 둡
 - 유한개의 소수를 모두 곱한 수를 $P = 2 \times 3 \times \dots \times p$ 임
 - P 는 모든 소수로 나누어떨어지지만, $P + 1$ 은 기존 소수들로 나누면 항상 나머지가 1임 그러므로 $P + 1$ 은 새로운 소수를 가지거나, 자기 자신이 새로운 소수임
 - 가장 큰 소수 p 가 있다는 가정과 모순되므로, 소수의 개수는 무한함

* p :가장 큰 소수
* P :유한개의 소수를 모두 곱한 수
* q : p 보다 작거나 같은 소수

소수

- 집합의 크기

- 예제 9.3

소수 전체 집합이 $\{2, 3, 5, 7, 11, 13, 17\}$ 이라고 가정하자.

1. 위 소수들을 모두 곱한 수 P 를 구하시오.
2. $P + 1$ 을 계산하시오.
3. $P + 1$ 의 소인수분해 결과를 통해, 17보다 큰 소수가 존재함을 보이시오.

- 풀이

1. $P = 2 \times 3 \times 5 \times 7 \times 11 \times 13 \times 17 = 510510$
2. $P + 1 = 510511$
3. $510511 = 19 \times 97 \times 277$ 로 표현됨
 - 17보다 큰 소수가 3개 존재함

소수

- 집합의 크기

- 소수의 개수

- 함수 $\pi(n)$ 은 n 보다 작거나 같은 소수의 개수를 나타냄

- $\pi(1) = 0, \pi(2) = 1, \pi(3) = 2, \pi(10) = 4, \pi(20) = 8$

- n 의 값이 커지게 된다면 근사 값으로 구할 수 있음

- $\left\lfloor \frac{n}{(\ln n)} \right\rfloor < \pi(n) < \left\lfloor \frac{n}{(\ln n - 1.08366)} \right\rfloor$

소수

- 집합의 크기
- 소수의 개수
 - 예제 9.4

1,000,000보다 작은 소수의 개수를 구하시오.

- 폴이
 - $\left[\frac{n}{(\ln n)} \right] < \pi(n) < \left[\frac{n}{(\ln n - 1.08366)} \right]$ 이 범위에 들어오는 소수의 개수는 72,383에서 78,543개 사이의 수임
 - 실제 1,000,000보다 작은 소수 개수는 78,498개임

소수

- 판정

- 방법

- 나눴음 검증

- 주어진 수 n 이 $\sqrt{n}$ 보다 작은 소수로 나누어지는지 확인하는 방법
 - 하나라도 나누어지면 합성수, 아무것도 나누어지지 않으면 소수

- 에라토스테네스의 체

- 일정 범위 안의 소수를 한 번에 찾는 방법

- 필요성

- 공개키 암호를 만들 때 큰 소수를 사용하기 위함

- 큰 소수를 사용하여 소인수분해를 어렵게 만들고, 암호의 안전성을 확보하기 위함

소수

- 판정
- 나눴음 검증
 - 예제 9.5

수 97은 소수인가?

- 풀이
 - $9 < \sqrt{97} < 10$ 이므로 $\sqrt{97}$ 보다 작은 소수는 2, 3, 5, 7임
 - 2, 3, 5, 7은 97을 나누지 못하므로 97은 소수임

- 예제 9.6

수 301은 소수인가?

- 풀이
 - $17 < \sqrt{301} < 18$ 이므로 $\sqrt{301}$ 보다 작은 소수는 2, 3, 5, 7, 11, 13, 17임
 - 2, 3, 5, 11, 13, 17은 301을 나누지 못하지만 7은 301을 나누므로 301은 소수가 아님

소수

- 판정

- 에라토스테네스의 체(Sieve of Eratosthenes)

- e.g ., 100보다 작은 모든 소수를 구하시오.

1. 2부터 100까지 모든 수를 씀
2. $\sqrt{100} = 10$ 이므로 10보다 작은 소수 2, 3, 5, 7로 나누어지는 수는 아래 표에서 모두 지우고 2, 3, 5, 7은 남겨둠
3. 지워지지 않고 남아있는 수가 소수임

<에라토스테네스의 체를 적용한 표>

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

소수

- 생성

- 메르센 소수(Mersenne Prime)

- 정의

- 메르센 수(Mersenne number) $M_n = 2^n - 1$ 중에서 소수인 수

$$M_2 = 2^2 - 1 = 3$$

$$M_3 = 2^3 - 1 = 7$$

$$M_5 = 2^5 - 1 = 31$$

$$M_7 = 2^7 - 1 = 127$$

$$M_{11} = 2^{11} - 1 = 2047(\text{이 수는 소수가 아님. } 2047 = 23 \times 89)$$

$$M_{13} = 2^{13} - 1 = 8191$$

$$M_{17} = 2^{17} - 1 = 131071$$

소수

- 생성

- 페르마 소수(Fermat Prime)

- 정의

- 페르마 수(Fermat Number) $F_n = 2^{2^n} + 1$ 중에서 소수인 수

$$F_0 = 2^{2^0} + 1 = 3$$

$$F_1 = 2^{2^1} + 1 = 5$$

$$F_2 = 2^{2^2} + 1 = 17$$

$$F_3 = 2^{2^3} + 1 = 257$$

$$F_4 = 2^{2^4} + 1 = 65537$$

$$F_5 = 2^{2^5} + 1 = 4294967297 = 641 \times 6700417 \text{ (이 수는 소수가 아님)}$$

소수

- 오일러 함수 φ (Euler's Phi-function)

- 정의

- 1부터 n 까지의 정수 중 n 과 서로소인 양의 정수의 개수를 구하는 함수

- 성질

- $\varphi(1) = 1$
 - $\varphi(p) = p - 1$, p 는 소수일 때
 - e.g., $\varphi(13) = 13 - 1 = 12$
 - $\varphi(m \times n) = \varphi(m) \times \varphi(n)$, m 과 n 이 서로 소일 때
 - e.g., $\varphi(10) = \varphi(2) \times \varphi(5) = (2 - 1) \times (5 - 1) = 1 \times 4 = 4$
 - $\varphi(p^e) = p^e - p^{e-1}$, p 는 소수일 때
 - e.g., $\varphi(240) = \varphi(2^4) \times \varphi(3) \times \varphi(5) = (2^4 - 2^3) \times (3 - 1) \times (5 - 1) = 64$

소수

- 오일러 함수 φ (Euler's Phi-function)

- 활용

- 만약 n 의 소인수분해꼴이 $p_1^{e_1} \times p_2^{e_2} \times \dots \times p_k^{e_k}$ 의 경우

$\varphi(n) = (p_1^{e_1} - p_1^{e_1-1}) \times (p_2^{e_2} - p_2^{e_2-1}) \times \dots \times (p_k^{e_k} - p_k^{e_k-1})$ 로
 $\varphi(n)$ 값을 구할 수 있음

- e.g ., $n = 60$ $60 = 2^2 \times 3^1 \times 5^1$
 $\varphi(60) = (2^2 - 2^{2-1})(3^1 - 3^{1-1})(5^1 - 5^{1-1})$
 $= (4 - 2)(3 - 1)(5 - 1)$
 $= 2 \times 2 \times 4 = 16$
 $\varphi(60) = 16$

소수

- 오일러 함수 φ (Euler's Phi-function)
- 예제 9.11

집합 Z_{14}^* 에 속하는 원소의 개수는 몇 개인가?

- 풀이
 - $\varphi(14) = \varphi(7) \times \varphi(2) = 6 \times 1 = 6$ 임
 - 실제 이 집합의 원소는 1, 3, 5, 7, 9, 11, 13으로 총 6개임

소수

- 페르마의 리틀 정리(Fermat's Little Theorem)

- 필요성

- RSA는 암호화와 복호화 과정에서 모듈러 거듭제곱 연산을 사용하므로 소수 모듈러에서 큰 거듭제곱의 나머지를 효율적으로 계산하는 데 필요함

- 정의

- 첫 번째 버전

- 가정

- 만약 p 가 소수이고 a 가 하나의 어떤 정수로서 p 는 a 를 나누지 못한다면 $a^{p-1} \equiv 1 \pmod{p}$ 가 성립함

- 두 번째 버전

- 가정

- 만약 p 가 소수이고 a 가 정수라면, $a^p \equiv a \pmod{p}$ 가 성립됨

소수

- 페르마의 리틀 정리(Fermat's Little Theorem)
- 첫 번째 버전
 - 예제 9.12

$6^{10} \bmod 11$ 을 계산하시오.

- 풀이
 - 첫 번째 버전 $a^{p-1} \equiv 1 \bmod p$
 - $a = 6, p = 11$
 $6^{p-1} = 6^{10} \equiv 1 \bmod 11$
 $6^{10} \bmod 11 = 1$ 임

소수

- 페르마의 리틀 정리(Fermat's Little Theorem)

- 두 번째 버전

- 예제 9.13

$3^{12} \bmod 11$ 을 계산하시오.

- 풀이

- 두 번째 버전 $a^p \equiv a \bmod p$
 - $p = 11, a = 3$
 - $3^{11} \equiv 3 \bmod 11$
 - $3^{12} \bmod 11 = (3^{11} \times 3) \bmod 11 = (3^{11} \bmod 11)(3 \bmod 11)$
 $= (3 \times 3) \bmod 11 = 9$

소수

- 페르마의 리틀 정리(Fermat's Little Theorem)

- 곱에 대한 역원

- 페르마 정리를 응용하면 모듈로 값이 소수일 경우 곱에 관한 역원을 구할 수 있음

- 가정

- 만약 p 가 소수이고 a 가 정수로서 p 로 나누어지지 않는 수임

$$a^{-1} \bmod p = a^{p-2} \bmod p$$

- e.g ., $p = 7, a = 3$ $\gcd(3, 7) = 1$

$$3^{-1} \equiv 3^{7-2} \bmod 7 \quad 3^5 = 243 \quad 243 \equiv 5 \bmod 7 \quad 3^{-1} \equiv 5 \bmod 7$$

- 증명

$$a \times a^{-1} \bmod p = a \times a^{p-2} \bmod p = a^{p-1} \bmod p = 1 \bmod p$$

- e.g ., $3 \times 3^{-1} \bmod 7 = 3 \times 3^{7-2} \bmod 7 = 3^6 \bmod 7 = 729 \bmod 7 = 1 \bmod 7$

소수

- 페르마의 리틀 정리(Fermat's Little Theorem)

- 예제 9.14

모듈로 값이 소수일 때 곱에 관한 역원을 계산하시오.

- 풀이

a. $8^{-1} \bmod 17 = 8^{17-2} \bmod 17 = 8^{15} \bmod 17 = 15 \bmod 17$

- $8^{15} = 8^8 \times 8^4 \times 8^2 \times 8^1 \equiv 1 \times 16 \times 13 \times 8 \equiv 15 \bmod 17$

$$8^2 = 64 \equiv 13 \bmod 17$$

$$8^4 = 13^2 = 169 \equiv 16 \bmod 17$$

$$8^8 \equiv 16^2 = 256 \equiv 1 \bmod 17$$

b. $60^{-1} \bmod 101 = 60^{101-2} \bmod 101 = 60^{99} \bmod 101 = 32 \bmod 101$

- $60^{99} = 60^{64} \times 60^{32} \times 60^2 \times 60^1 \equiv 84 \times 36 \times 65 \times 60 \equiv 32 \bmod 101$

- $60^{64} \equiv 36^2 = 1296 \equiv 84 \bmod 101$

- 실제 역원 계산은 확장 유클리드 알고리즘이 더 효율적임

소수

- 오일러 정리(Euler's Theorem)

- 필요성

- RSA에서는 $n = p \times q$ 처럼 두 소수의 곱인 합성수를 사용하기 때문에 합성수 모듈러에서 성립하는 오일러 정리가 필요함

- 정의

- 첫 번째 버전

- a 와 p 가 서로소라면, $a^{\varphi(n)} \equiv 1 \pmod n$ ($\gcd(a, n) = 1$)
 - e.g ., $6^{24} \pmod{35} = 6^{\varphi(35)} \pmod{35} = 1$

- 두 번째 버전

- $a^{k \times \varphi(n) + 1} \equiv a \pmod n$ ($n = p \times q, a < n, k \in \mathbb{Z}$)
 - e.g ., $20^{62} \pmod{77}$ 을 구할 때 $k = 1$ 이라고 한다면

$$n = 77, a = 20$$

$$\varphi(77) = \varphi(7) \times \varphi(11) = 6 \times 10 = 60$$

$$20^{62} \pmod{77} = (20 \pmod{77}) (20^{60+1} \pmod{77}) = (20)(20) \pmod{77} = 15$$

소수

- 오일러 정리(Euler's Theorem)

- 첫 번째 버전 증명 (1/2)

- n 과 서로소인 수들의 집합을 S 라 둬

- $S = \{s_1, s_2, \dots, s_{\varphi(n)}\}$

- e.g ., 만

- 각 원소에 a 를 곱한 집합을 aS 라 둬

- $aS = \{as_1, as_2, \dots, as_{\varphi(n)}\}$

- e.g ., $3S = \{3 \times 1, 3 \times 3, 3 \times 7, 3 \times 9\} = \{3, 9, 21, 27\}$

- a 와 n 이 서로소 이므로, aS 의 원소들도 n 과 서로소임

- aS 의 원소들을 n 으로 나눈 나머지는 서로 중복되지 않음

- e.g ., 각각 10으로 나눈 나머지

- $3 \bmod 10 = 3, 9 \bmod 10 = 9, 21 \bmod 10 = 1, 27 \bmod 10 = 7$

- 따라서 aS 는 S 와 같은 원소들을 순서만 바꾼 집합임

- e.g ., 나머지 집합은 $\{3, 9, 1, 7\}$ S 와 원소는 같고 순서만 다름

소수

- 오일러 정리(Euler's Theorem)

- 첫 번째 버전 증명 (2/2)

- 두 집합의 모든 원소를 곱하면

- $(as_1)(as_2) \cdots (as_{\varphi(n)}) \equiv s_1 s_2 \cdots s_{\varphi(n)} \pmod n$

- e.g ., $(3 \times 1)(3 \times 3)(3 \times 7)(3 \times 9) \equiv 1 \times 3 \times 7 \times 9 \pmod{10}$

- 왼쪽을 정리하면

- $a^{\varphi(n)} s_1 s_2, \cdots, s_{\varphi(n)} \equiv s_1 s_2, \cdots, s_{\varphi(n)} \pmod n$

- e.g .,

- 는 n 과 서로소이므로 약분 가능함

- $a^{\varphi(n)} \equiv 1 \pmod n$

- e.g ., $1 \times 3 \times 7 \times 9 = 189, \gcd(189, 10) = 1$

- 따라서 오일러 정리가 성립함

- e.g ., $3^4 = 81 \equiv 1 \pmod{10}$

소수

- 오일러 정리(Euler's Theorem)

- 두 번째 버전 증명 (1/3)

- 경우 1

- a 가 p 와 q 의 배수가 아닌 경우, $\gcd(a, n) = 1$

$$a^{k \times \varphi(n) + 1} \bmod n = (a^{\varphi(n)})^k \times a \bmod n = 1^k \times a \bmod n = a \bmod n$$

- e.g ., $n = 15 = 3 \times 5$, $\varphi(15) = 8$, $k = 1$, $a = 2$

$$2^{1 \times 8 + 1} \equiv (2^8)^1 \times 2 \bmod 15$$

$$2^8 = 256 \equiv 1 \bmod 15 \text{ 따라서 } 2^9 \equiv 1^1 \times 2 \equiv 2 \bmod 15$$

따라서 $a^{k \times \varphi(n) + 1} \equiv a \bmod n$ 이 성립함

소수

- 오일러 정리(Euler's Theorem)

- 두 번째 버전 증명 (2/3)

- 경우 2

- a 가 p 의 배수 가 되어 $a = i \times p$ 이지만 q 의 배수는 아니라고 함

$$a^{\varphi(n)} \bmod q = (a^{\varphi(q)} \bmod q)^{\varphi(p)} \bmod q = 1 \rightarrow a^{\varphi(n)} \bmod q = 1$$

$$a^{k \times \varphi(n)} \bmod q = (a^{\varphi(n)} \bmod q)^k \bmod q = 1 \rightarrow a^{k \times \varphi(n)} \bmod q = 1$$

$$a^{k \times \varphi(n)} \bmod q = 1 \rightarrow a^{k \times \varphi(n)} = 1 + j \times q$$

$$a^{k \times \varphi(n)+1} = a \times (1 + j \times q) = a + j \times q \times a = a + (i \times j) \times q \times p = a + (i \times j) \times n$$

$$a^{k \times \varphi(n)+1} = a + (i \times j) \times n \rightarrow a^{k \times \varphi(n)+1} = a \bmod n$$

- e.g ., $k = 1$ $n = 77 = 7 \times 11$ 이고, $a = 14$ 라고 할 때 a 와 11은 서로소이므로,
오일러 정리 적용가능함 $14^{\varphi(11)} \equiv 1 \bmod 11$
 $\varphi(11)$ 은 10 이므로, $14^{10} \bmod 11$ 에서 1과 같음
 $n = 77 = 7 \times 11$ 일 때 $\varphi(77) = 60$ 따라서 $14^{60} \equiv 1 \bmod 11$
여기서 $a = 14$ 를 한번 더 곱하면 $14^{61} \equiv 14 \bmod 11$
14는 7의 배수이므로, 14^{61} 과 14는 둘 다 $\bmod 7$ 에서 0임
 14^{61} 과 14는 $\bmod 7$ 에서도 같고, $\bmod 11$ 에서도 같음

소수

- 오일러 정리(Euler's Theorem)
- 두 번째 버전 증명 (3/3)
 - 경우 3
 - 만약 a 가 q 의 배수로서 $a = i \times q$ 이지만 p 의 배수는 아니라고 함
 - 증명은 2의 경우와 동일함(위 증명에서 p 와 q 의 역할만 바뀜)

목 차

- 보충
- 기초 개념
- 소수
- 소수 판정
- 소인수분해

소수 판정

- 개요

- 큰 정수가 주어졌을 때 이 수가 소수인지 합성수인지 정확하고 효율적인 판단 방법

- 종류

- 결정적 알고리즘(Deterministic algorithm)
 - 항상 정확한 답을 제시함
- 확률적 알고리즘(Probabilistic algorithm)
 - 거의 정확한 판단을 할 수 있지만 항상 맞는 것은 아님

소수 판정

- 결정적 알고리즘 (Deterministic algorithm)
 - 정의
 - 결정적 소수 판정 알고리즘은 주어진 정수 n 에 대해 정해진 절차로 소수 여부를 판별하는 알고리즘임
 - 종류
 - 나눗셈 알고리즘
 - AKS 알고리즘

소수 판정

- 결정적 알고리즘 (Deterministic algorithm)
 - 나눴 알고리즘
 - 정의
 - 2부터 $\sqrt{n}$ 보다 작은 수까지 차례로 나누어, n 의 약수가 존재하는지 확인하는 소수 판정 알고리즘임
 - 의사 코드

```
Divisibility_Test (n)    //n 에 대해서 소수 검증을 한다.  
{  
    r ← 2  
    while(r <  $\sqrt{n}$ )  
    {  
        if(r | n) return "a composite"  
        r ← r + 1  
    }  
    return "a prime"  
}
```

소수 판정

- 결정적 알고리즘 (Deterministic algorithm)
 - 나눴 알고리즘
 - 비트-연산 복잡도
 - $f(n_b) = \sqrt{2^{n_b}} = 2^{n_b/2}$
 - n_b 는 n 을 2진수로 변경하였을 때 비트 수임
 - Big-O 기호를 사용하여 표현하면 시간 복잡도 $O(2^{n_b})$ 임
 - 나눴 알고리즘은 n_b 가 커지면 실용적이지 못함

소수 판정

- 결정적 알고리즘 (Deterministic algorithm)
 - 나눴 알고리즘
 - 예제 9.18

n 의 비트 수가 200이라고 가정하자. 나눴 알고리즘을 수행한다고 한다면 비트 연산 회수는 얼마나 되는가?

- 풀이
 - 비트-연산 복잡도는 $2^{n_b/2}$ 으로 2^{100} 번의 비트 연산을 수행해야 함
 - 초당 2^{30} 번의 비트 연산을 수행할 수 있는 컴퓨터를 사용하는 경우, 이 알고리즘을 이용해서 소수 검증을 수행하는데 2^{70} 초가 걸림
 - 2^{70} 초는 거의 영원한 시간이므로 비효율적임

소수 판정

- 결정적 알고리즘 (Deterministic algorithm)
 - AKS 알고리즘 (Agrawal-Kayal-Saxena algorithm)
 - 정의
 - 다항식 합동식을 이용해 소수 여부를 판정하는 결정적 소수 판정 알고리즘
 - 동작 과정
 1. $(x - a)^n \equiv (x^n - a) \pmod n$ 성립 여부를 확인함
 2. 성립하면 소수 조건을 만족하고, 성립하지 않으면 합성수로 판정함
 - e.g., $n = 5, a = 1$
 $(x - 1)^5$ 을 전개하면 $x^5 - 5x^4 + 10x^3 - 10x^2 + 5x - 1$ 임
중간 계수들 $(-5, 10, -10, 5)$ 모두 5로 나누어 떨어져 중간항 사라짐
 $(x - 1)^5 \equiv x^5 - 1 \pmod 5$ 이므로 성립함

소수 판정

- 결정적 알고리즘 (Deterministic algorithm)
 - AKS 알고리즘 (Agrawal-Kayal-Saxena algorithm)
 - 알고리즘 복잡도
 - $O((\log_2 n_b)^{12})$ 인 다항식 비트-연산 시간을 가짐
 - 예제 9.19

n 의 비트 수가 200이라고 가정하자. AKS 알고리즘을 수행한다고 한다면 비트 연산 회수는 얼마나 되는가?

- 풀이
 - 이 알고리즘의 비트-연산 복잡도는 $(\log_2 n_b)^{12}$ 으로 $(\log_2 200)^{12} = 39,547,615,483$ 번의 비트 연산을 수행하는 경우, 이 수가 소수인지 아닌지를 알아낼 수 있음

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)
 - 정의
 - 확률적 알고리즘은 임의로 선택한 밑수나 난수를 이용해 주어진 수가 소수인지 합성수인지 판정하는 알고리즘임
- 종류
 - 페르마 검증(Fermat Primality Test)
 - 제곱근 검증(Square Root Primality Test)
 - 밀러-라빈 검증 (Miller-Rabin Primality Test)

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)

- 규칙

- 만약 검증하려는 정수가 실제로 소수라면, 이 알고리즘은 반드시 소수라고 판정함
- 만약 검증하려는 정수가 실제로 합성수인 경우, 이 알고리즘은 이 수를 합성수라고 판정할 확률이 $1 - \varepsilon$ 이고, 소수라고 판정할 확률이 ε 이 됨
 - 여기서 ε 은 한 번 검증했을 때 합성수를 소수로 잘못 판단할 확률임
 - 이 알고리즘을 m 번 수행한다면, 실수할 확률은 ε^m 으로 줄어듦

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)

- 페르마 검증

- 정의

- 페르마의 리틀 정리 (n 이 소수면 $a^{n-1} \equiv 1 \pmod{n}$)를 이용해 소수 여부를 확률적으로 판정하는 알고리즘

- 성질

만약 n 이 소수라면, $a^{n-1} \equiv 1 \pmod{n}$ 임

만약 n 이 합성수라면, $a^{n-1} \equiv 1 \pmod{n}$ 일 수도 있음

- 소수는 페르마 검증을 해보면 소수로 판정되지만 합성수가 소수라고 판정 날 확률이 ε 임
 - 여러 가지의 밑수를 사용해서 검증하면 오류 확률을 더 작게 줄일 수 있음

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)

- 페르마 검증
 - 예제 9.20

수 561을 페르마 검증하면 어떻게 되는가?

- 풀이

- 2를 밑수로 사용할 경우 $2^{561-1} \equiv 1 \pmod{561}$ 을 만족하므로 페르마 검증을 통과함
- 3을 밑수로 사용할 경우 $3^{561-1} \equiv 375 \pmod{561}$ 이므로 $1 \pmod{561}$ 을 만족하지 않아서 페르마 검증을 통과하지 못함
- 5를 밑수로 사용할 경우 $5^{561-1} \equiv 1 \pmod{561}$ 을 만족하므로 페르마 검증을 통과함
- 7을 밑수로 사용할 경우 $7^{561-1} \equiv 1 \pmod{561}$ 을 만족하므로 페르마 검증을 통과함

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)

- 제곱근 검증

- 정의

- 모듈로 계산에서 1의 제곱근이 ± 1 인지 확인하는 소수 판정 알고리즘

- 성질

만약 n 이 소수라면, $\sqrt{1} \bmod n = \pm 1$

만약 n 이 합성수라면, $\sqrt{1} \bmod n = \pm 1$ 이며 다른 값들을 가질 수도 있다

- 특징

- 합성수 판정에 활용 가능함
 - 모듈로 연산에서 -1 이 의미하는 것은 $n - 1$ 이라는 것에 유의
 - e.g., $-1 \equiv 4 \bmod 5$

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)

- 제곱근 검증

- 예제 9.21

n 이 7(소수)이라면 모듈로 n 으로 1의 제곱근은 무엇인가?

- 풀이

- $1^2 = 1 \bmod 7$ $(-1)^2 = 1 \bmod 7$
 $2^2 = 4 \bmod 7$ $(-2)^2 = 4 \bmod 7$
 $3^2 = 2 \bmod 7$ $(-3)^2 = 2 \bmod 7$
 $4^2 = 2 \bmod 7$ $(-4)^2 = 2 \bmod 7$
 $5^2 = 4 \bmod 7$ $(-5)^2 = 4 \bmod 7$
 $6^2 = 1 \bmod 7$ $(-6)^2 = 1 \bmod 7$

- 1과 6(= $n - 1$)을 제외한 수들을 제곱하여 모듈로 연산을 하면 1이 나오는 값이 존재하지 않음 그러므로 7은 소수일 가능성이 높음

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)
- 제곱근 검증
 - 예제 9.22

n 이 8이라면 모듈로 n 으로 1의 제곱근은 무엇인가?

- 풀이
 - $1^2 = 1 \bmod 8$ $(-1)^2 = 1 \bmod 8$
 $3^2 = 1 \bmod 8$ $5^2 = 1 \bmod 8$
 - 근을 4개 가짐 그 근은 1, 3, 5, 7(이것이 -1임)

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)
- 제곱근 검증
 - 예제 9.23

n 이 17이라면 모듈로 n 으로 1의 제곱근은 무엇인가?

- 풀이

- $1^2 = 1 \bmod 17$ $(-1)^2 = 1 \bmod 17$
 $2^2 = 4 \bmod 17$ $(-2)^2 = 4 \bmod 17$
 $3^2 = 9 \bmod 17$ $(-3)^2 = 9 \bmod 17$
 $4^2 = 16 \bmod 17$ $(-4)^2 = 16 \bmod 17$
 $5^2 = 8 \bmod 17$ $(-5)^2 = 8 \bmod 17$
 $6^2 = 2 \bmod 17$ $(-6)^2 = 2 \bmod 17$
 $7^2 = 15 \bmod 17$ $(-7)^2 = 15 \bmod 17$
 $8^2 = 13 \bmod 17$ $(-8)^2 = 13 \bmod 17$
 $9^2 = 13 \bmod 17, \dots$ 로 앞의 값들과 중복됨
- $\bmod 17$ 에서 1의 제곱근은 1과 $-1(16)$ 2개의 근을 가짐

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)

- 밀러-라빈 검증(Miller-Rabin Primality Test)

- 정의

- 페르마 검증과 제곱근 검증을 결합한 확률적 소수 판정 방법

$n - 1 = m \times 2^k$ 로 분해하면 단계적으로 계산할 수 있음

- e.g ., $n = 21$, 밑수 $a = 2$
 $20 = 5 \times 2^2$ 이므로 $m = 5, k = 2$ 임

- 밑수 a 를 사용한 페르마 검증은 다음과 같이 나타냄

$$a^{n-1} = a^{m \times 2^k} = (a^m)^{2 \cdots 2}$$

K times

- $a^{n-1} \bmod n$ 을 계산할 때 1단계로 하는 대신 $k + 1$ 단계로 할 수 있음
 - 각 단계에서 제곱근 검증을 할 수 있어서 유리함
 - e.g ., $2^5 \bmod 21 = 32 \bmod 21 = 11$
 $11^2 = 121 \bmod 21 = 16$ $16^2 = 256 \bmod 21 = 4 \cdots, 2^{20} \bmod 21 = 4$
최종 값이 1이 아니므로 21은 합성수임

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)
- 밀러-라빈 검증(Miller-Rabin Primality Test)
 - 초기화 단계
 - 밑수 a 를 선택하고 $T = a^m \bmod n$ 을 구함(여기서 $m = (n - 1)/2^k$)
 - a. 만약 T 가 $+1$ 이거나 -1 이라면 n 은 해당 밑수 a 에 대한 검증을 통과함
 - 페르마 검증과 제곱근 검증 2가지를 통과했다고 간주함
 - 소수로 확정된 것은 아니며 실제 합성수라면 강한 의사소수라고 함
 - b. 만약 T 가 그 외의 값을 가진다면 n 이 소수인지 합성수인지 판단할 수 없기에 다음 단계로 넘어가게 됨

* T :현재 검사 중인 중간 값
*강한 의사소수:실제로는 합성수이지만
밀러-라빈 검증을 통과한 수

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)

- 밀러-라빈 검증(Miller-Rabin Primality Test)

- T 를 제공하는 단계(1 단계)

- a. 만약 제공한 결과가 +1이라면, n 을 합성수로 판단함

- 소수라면 $x^2 \equiv 1 \pmod n$ 의 해는 $x \equiv 1$ 또는 $x \equiv -1$ 뿐이어야 함 하지만 합성수에서는 1, -1이 아닌 값도 제공하면 1이 될 수 있음

- e.g., $\pmod{15}$ 을 보면

- $1^2 = 1 \equiv 1 \pmod{15}$ $(-1)^2 \equiv 14^2 = 196 \equiv 1 \pmod{15}$ 은 정상 제공근임

- $4^2 = 16 \equiv 1 \pmod{15}$ $11^2 = 121 \equiv 1 \pmod{15}$ 도 제공 결과가 1이 됨

- 4, 11은 1도 아니고 -1도 아니므로 제공근 검증을 통과하지 못함

- b. 만약 제공한 결과가 -1이 된다면, 해당 밑수에 대한 밀러-라빈 검증을 통과함

- 페르마 검증과 제공근 검증 모두 통과하였기 때문임

- e.g., $\pmod{5}$ 에서

- $2^2 = 4 \equiv -1 \pmod{5}$ 따라서 제공 결과가 -1이므로 밑수 검증을 통과함

- c. 만약 제공한 결과가 ± 1 이 아닌 다른 값을 갖는 경우, n 이 소수인지 아닌지 알 수 없으므로 다음 단계로 넘어감

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)
- 밀러-라빈 검증(Miller-Rabin Primality Test)
 - 2단계에서 $k - 1$ 단계
 - 해당 단계에서 부터 $k - 1$ 까지는 모두 1단계와 같음
 - k 단계
 - 이 단계는 결론에 이르지 못하면 더 이상 의미가 없음
 - 이 단계의 결과가 1이라면, 페르마 검증은 통과하지만 앞 단계의 결과가 ± 1 이 아니었기 때문에 제공된 검증은 통과하지 못함
 - $k - 1$ 번째 단계 후까지 절차가 끝나지 않았다면 n 을 합성수라고 함

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)
- 밀러-라빈 검증(Miller-Rabin Primality Test)
 - 의사코드

```
Miller_Rabin_Test( $n, a$ )           //  $n$ 은 수이고,  $a$ 는 밑수이다.
{
  Find  $m$  and  $k$  such that  $n - 1 \equiv m \times 2^k$ 
   $T \leftarrow a^m \bmod n$ 
  if ( $T = \pm 1$ ) return "a prime"
  for ( $i \leftarrow 1$  to  $k - 1$ )       //  $k - 1$ 은 최대 단계 수임
  {
     $T \leftarrow T^2 \bmod n$ 
    if( $T = +1$ ) return "a composite"
    if( $T = -1$ ) return "a prime"
  }
  return "a composite"
}
```

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)
- 밀러-라빈 검증(Miller-Rabin Primality Test)
 - 예제 9.25

수 561은 밀러-라빈 검증을 통과하는가?

- 풀이
 - 밑수 a 를 2로 사용하면, $561 - 1 = 35 \times 2^4$ 로 표현되어 $m = 35, k = 4, a = 2$ 임
 - 초기화 $T = 2^{35} \bmod 561 = 263 \bmod 561$
 - $k = 1: T = 263^2 \bmod 561 = 166 \bmod 561$
 - $k = 2: T = 166^2 \bmod 561 = 67 \bmod 561$
 - $k = 3: T = 67^2 \bmod 561 = +1 \bmod 561 \rightarrow$ 합성수

소수 판정

- 확률적 알고리즘(Probabilistic Algorithm)
- 밀러-라빈 검증(Miller-Rabin Primality Test)
 - 예제 9.26

27은 소수가 아닌 것을 알고 있다. 밀러-라빈 검증을 하시오.

- 풀이
 - 밑수 20이면 $27 - 1 = 13 \times 2^1$ 이므로 $m = 13$, $k = 1$, $a = 2$ 임
 - $k - 1 = 0$ 이므로 초기화 단계만 수행하면
 $T = 2^{13} \bmod 27 = 11 \bmod 27$

소수 판정

- 나눅 검증과 밀러-라빈 검증을 조합한 검증

- 정의

- 나눅 검증과 밀러-라빈 검증을 조합한 혼합형 소수 판정 방식
 - 현재 가장 많이 사용되는 소수 검증 방법

1. 홀수 정수를 하나 선택함

- 2를 제외한 짝수는 모두 합성수임

2. 알려진 작은 소수들로 먼저 나누어 합성수 후보를 제거함

- 나누어떨어지면 합성수이므로 탈락, 통과하면 밀러-라빈 검증으로 넘어감

3. 검증을 하기 위해서 밑수로 사용할 수들의 집합을 결정함

- 밑수를 여러 개 사용할수록 오류 확률이 줄어듦

4. 각 밑수로 밀러-라빈 검증을 실시하고 통과했다면 그 수를 소수일 가능성이 높은 수로 판단함

소수 판정

- 추천하는 소수 검증

- 예제 9.28

4033은 37×109 이므로 합성수이다. 이 수는 위에 언급한 추천하는 소수 검증을 통과하는가?

- 풀이

1. 나눴 검증을 실시함

- 2,3,5,7,11,17,23은 4033의 약수가 아님

2. 밑수를 2로 사용하여 밀러-라빈 검증을 실시함

- $4033 - 1 = 63 \times 2^6$ 이므로 $m = 63, k = 6$ 임
- 초기화: $T = 2^{63} \bmod 4033 = 3521 \bmod 4033$
- $k = 1$: $T = 3521^2 \bmod 4033 = -1 \bmod 4033 \rightarrow$ 통과

소수 판정

- 추천하는 소수 검증

- 예제 9.28

4033은 37×109 이므로 합성수이다. 이 수는 위에 언급한 추천하는 소수 검증을 통과하는가?

- 풀이

- 3. 만족스럽지 못하므로 밑수로 3을 사용함

- 초기화: $T = 3^{63} \bmod 4033 = 3551 \bmod 4033$

- $k = 1: T = 3551^2 \bmod 4033 = 2443 \bmod 4033$

- $k = 2: T = 2443^2 \bmod 4033 = 3442 \bmod 4033$

- $k = 3: T = 3442^2 \bmod 4033 \equiv 2443 \bmod 4033$

- $k = 4: T = 2443^2 \bmod 4033 \equiv 3442 \bmod 4033$

- $k = 5: T = 3442^2 \bmod 4033 \equiv 2443 \bmod 4033 \rightarrow \text{실패(합성수)}$

목 차

- 보충
- 기초 개념
- 소수
- 소수 판정
- 소인수분해

소인수분해

- 산술 기본 정리

- 1보다 큰 모든 자연수는 소수들의 곱으로 표현됨

$$n = p_1^{e_1} \times p_2^{e_2} \times \cdots p_k^{e_k}$$

- 최대 공약수나 최소 공배수를 구할 때 편리함

- 최대 공약수(gcd)

$$a = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_k^{a_k} \quad b = p_1^{b_1} \times p_2^{b_2} \times \cdots p_k^{b_k}$$
$$\gcd(a, b) = p_1^{\min(a_1, b_1)} \times p_2^{\min(a_2, b_2)} \times \cdots p_k^{\min(a_k, b_k)}$$

- 공통 소인수의 지수 중 작은 값을 선택함

- 최소 공배수(lcm)

$$a = p_1^{a_1} \times p_2^{a_2} \cdots p_k^{a_k} \quad b = p_1^{b_1} \times p_2^{b_2} \times \cdots p_k^{b_k}$$
$$\text{lcm}(a, b) = p_1^{\max(a_1, b_1)} \times p_2^{\max(a_2, b_2)} \times \cdots p_k^{\max(a_k, b_k)}$$

- 각 소인수의 지수 중 큰 값을 선택함
- $\text{lcm}(a, b) \times \gcd(a, b) = a \times b$ 관계가 있음

소인수분해

- 방법
 - 전수 나눴음 소인수분해 방법
 - 페르마 방법
 - Pollard $p - 1$ 방법
 - Pollard rho 방법

소인수분해

- 방법

- 전수 나눴을 소인수분해 (1/2)

- 정의

- 2부터 $\sqrt{n}$ 까지 n 을 나누는 수를 찾아내는 방법

- 의사코드

```
Trial_Division_Factorization(n)    //n은 인수분해를 하려고 하는 수이다
{
  a ← 2
  while(a ≤ √n)
  {
    output a                        // 인수가 하나 하나 추출된다
    n = n/a
  }
  a ← a + 1
}
if(n > 1) output n                // n은 더 이상 인수가 없다
}
```

1. $a = 2$ 부터 시작
2. $a \leq \sqrt{n}$ 동안 반복
3. n 이 a 로 나누어지면 a 를 인수로 출력
4. n 을 a 로 나누어 더 작은 문제로 바꿈
5. a 를 1 증가시켜 다음 수로 검사
6. 마지막에 n 이 1보다 크면 남은 n 도 인수로 출력

- 복잡도

- $n < 2^{10}$ 인 경우에 효과가 있음
 - 2^{10} 이상의 큰 정수를 소인수분해 하는 것은 효율성이 떨어짐

소인수분해

- 방법

- 전수 나눗셈 소인수분해 (2/2)

- 예제 9.29

1233을 소인수분해하기 위해서 전수 나눗셈 알고리즘을 사용하시오.

- 풀이

- 알고리즘에 따라 프로그램을 실행하면 $1233 = 3^2 \times 137$ 이라는 결과를 얻음

- 예제 9.30

1523357784를 소인수분해하기 위해서 전수 나눗셈 알고리즘을 사용하시오.

- 풀이

- 알고리즘에 따라 프로그램을 실행하면 $1523357784 = 2^3 \times 3^2 \times 13 \times 37 \times 43987$ 이라는 결과를 얻음

소인수분해

- 방법

- 페르마 방법 (1/2)

- 정의

- 정수 n 을 $x^2 - y^2$ 형태로 표현하여 $n = (x - y)(x + y)$ 로 인수분해하는 방법

- 의사코드

```
Feramat_Factorization(n)    //n은 인수분해를 하려고 하는 수이다
{
    x ← √n                    //√n보다 큰 수중 가장 작은 정수
    while(x < n)
    {
        w ← x2 - n

        if(w is perfect square) y ← √w; a ← x + y; b ← x - y; return a and b
        x ← x + 1
    }
}
```

1. x 를 $\sqrt{n}$ 이상인 가장 작은 정수로 설정함
2. $w = x^2 - n$ 계산함
3. w 가 완전제곱수인지 확인함
4. 완전제곱수이면 $y = \sqrt{w}$ 를 구함
5. $n = x^2 - y^2 = (x - y)(x + y)$ 이므로 두 인수 $x - y, x + y$ 를 반환함
6. w 가 완전제곱수가 아니면 x 를 1 증가시키고 반복함

- 복잡도

- 두 인수가 가까우면 빠르지만, 차이가 크면 비효율적임

소인수분해

- 방법

- 페르마 방법 (2/2)

- 기본 원리

- 만약 $n = x^2 - y^2$ 을 만족하는 x 와 y 를 찾을 수 있다면 다음이 성립됨

$$n = x^2 - y^2 = a \times b \text{ 여기서 } a = x + y \text{ 이고 } b = x - y$$

- 탐색 과정

- 서로 크기가 비슷한 두 정수 a, b 를 찾음
 - 먼저 $x = \sqrt{n}$ 보다 크거나 같은 가장 작은 정수에서 시작함
 - $y^2 = x^2 - n$ 을 만족하는 다른 정수 y 를 구함
 - 각 반복마다 $x^2 - n$ 값을 계산함 y 를 찾을 수 있으면 a 와 b 를 계산해내고 루프로부터 빠져나옴
 - 완전제곱수가 아니면 x 를 1 증가시켜 다시 반복함

소인수분해

- 방법

- Pollard $p - 1$ 방법 (1/3)

- 정의

- $p - 1$ 의 인수 구조를 이용해 합성수 n 의 소인수 p 를 찾는 방법

$$p = \gcd(2^{B!} - 1, n)$$

- 의사코드

```
Pollard_(p - 1)_Factorization(n, B)    //n은 인수분해를 하려고 하는 수이다
{
  a ← 2
  e ← 2
  while(e ≤ B)
  {
    a ← ae mod n
    e ← e + 1
  }
  p ← gcd(a - 1, n)
  if 1 < p < n return p
  return failure
}
```

1. 기준값 B 를 정하고 $a \leftarrow 2$, $e \leftarrow 2$ 로 설정함
2. e 가 B 이하인 동안 반복함
($a \leftarrow a^e \bmod n$, $e \leftarrow e + 1$)
3. 반복이 끝나면 a 는 $2^{B!} \bmod n$ 형태가 됨
4. $p \leftarrow \gcd(a - 1, n)$ 을 계산함
5. $1 < p < n$ 이면 p 는 n 의 소인수이므로 반환함
6. 조건을 만족하지 않으면 실패로 판단함

소인수분해

- 방법

- Pollard $p - 1$ 방법 (2/3)

- 복잡도

- $B - 1$ 번의 지수 연산($a = a^e \bmod n$)이 필요함
 - $\gcd(a - 1, n)$ 계산을 통해 소인수 후보를 찾고 B 가 적절하지 않으면 소인수를 찾지 못하고 실패할 수 있음
 - B 가 $\sqrt{n}$ 에 충분히 가깝지 않으면 성공 확률이 낮음

소인수분해

- 방법

- Pollard $p - 1$ 방법 (3/3)
 - 예제 9.31

57247159의 소인수를 구하기 위해서 Pollard $p - 1$ 소인수분해 방법을 사용하시오. 여기서 $B = 8$ 을 사용하시오

- 풀이
 - 알고리즘에 따라 프로그램을 실행하면 $p = 421$ 을 찾아낼 수 있음
 - $57247159 = 421 \times 135979$ 이고 421은 소수이고 $p - 1 = 421 - 1$ 은 8보다 큰 인수를 갖지 않음 ($420 = 2^2 \times 3 \times 5 \times 7$)

소인수분해

- 방법

- Pollard rho 방법 (1/4)

- 정의

- 반복 수열의 충돌을 이용해 $\gcd(x - y, n)$ 으로 소인수를 찾는 방법

- 전제 사실

- a. 두 개의 정수 x_1 과 x_2 가 존재해서 p 가 $x_1 - x_2$ 를 나누지만, n 은 $x_1 - x_2$ 을 나누지 못한다고 가정함
 - b. $p = \gcd(x_1 - x_2, n)$ 을 증명 할 수 있음
 - p 가 $x_1 - x_2$ 를 나누기 때문에 $x_1 - x_2 = q \times p$
 - n 은 $x_1 - x_2$ 을 나누지 못하기 때문에 q 는 n 을 나누지 못하고 이것은 $\gcd(x_1 - x_2, n)$ 이 1이거나 n 의 인수라는 말임

소인수분해

- 방법

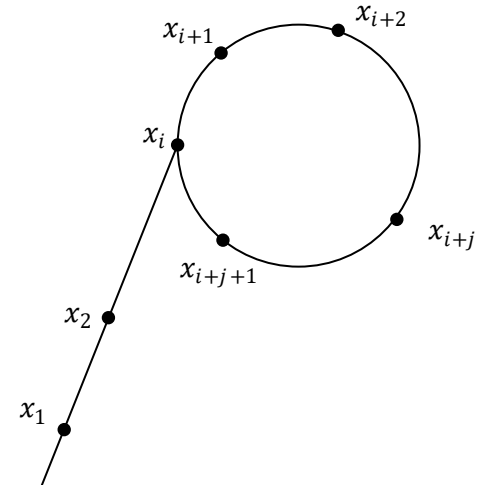
- Pollard rho 방법 (2/4)

- 과정

1. 종자(seed)라고 부르는 랜덤한 정수 x_1 을 선택함
2. n 은 $x_1 - x_2$ 을 나누지 못하게 하는 x_2 를 계산하기 위해 $x_2 = f(x_1) = x_1^2 + a$ 함수를 사용함(여기서 a 는 1임)
3. $\gcd(x_1 - x_2, n)$ 을 계산함
 - 이 값은 1이 아니면 그 값은 n 의 인수임
4. 만약 이 값이 1이면 1 단계로 돌아가서 x_2 를 가지고 반복함

e.g ., $n = 91$, $f(x) = x^2 + 1$, $x_1 = 2$

1. seed로 $x_1 = 2$ 선택
2. $x_2 = f(2) = 2^2 + 1 = 5$ 계산
3. $\gcd(|2 - 5|, 91) = \gcd(3, 91) = 1$ 이므로 인수 발견 실패
4. $x_1 = 5$ 로 다시 반복
 $x_2 = f(5) = 26, \gcd(|5 - 26|, 91) = \gcd(21, 91) = 7$



소인수분해

- 방법

- Pollard rho 방법 (3/4)

- 의사코드

```
Pollard_rho_Factorization( $n, B$ )    //  $n$ 은 인수분해를 하려고 하는 수이다
{
   $x \leftarrow 2$ 
   $y \leftarrow 2$ 
   $p \leftarrow 1$ 
  while( $p = 1$ )
  {
     $x \leftarrow f(x) \bmod n$ 
     $y \leftarrow f(f(y) \bmod n) \bmod n$ 
     $p \leftarrow \gcd(x - y, n)$ 
  }
  return  $p$                                 // 만약  $p = n$  이라면 프로그램은 실패이다.
}
```

1. x, y 를 같은 초기값 2로 설정함
2. p 를 1로 설정하고, p 가 1인 동안 반복함
3. x 는 한 번 이동: $x \leftarrow f(x) \bmod n$
4. y 는 두 번 이동: $y \leftarrow f(f(y) \bmod n)$
5. $p \leftarrow \gcd(x - y, n)$ 을 계산함
6. p 가 1보다 크고 n 보다 작으면 p 는 n 의 인수임

- 복잡도

- $\sqrt{p}$ 정도의 연산 양이 필요함 하지만 p 는 $\sqrt{n}$ 과 같거나 이보다 작은 값이기 때문에 대략 $n^{1/4}$ 정도의 연산 양이 필요함
- Pollard rho는 전수 나눗셈보다 빠르지만, 비트-연산 복잡도는 여전히 지수적(2의 거듭제곱 형태로 증가)이므로 큰 수에서는 한계가 있음

소인수분해

- 방법

- Pollard rho 방법 (4/4)

- 예제 9.32

어떤 컴퓨터가 초당 2^{30} (거의 10억 번)번 정도의 비트 연산을 할 수 있다고 하자. 다음과 같은 크기의 정수를 소인수분해 할 때 걸리는 시간이 대략 어느정도 되는가?

- a. 60자리 10진수
- b. 100자리 10진수

- 풀이

- a. 60자리를 갖는 10진수를 2진수로 바꿔보면 비트 수가 약 200자리 정도가 됨
복잡도($2^{n_b/4}$)는 대략 2^{50} 이고 초당 2^{30} 연산 능력을 가진 컴퓨터를 이용하면 2^{20} 초인 약 12일 정도 걸림
 - b. 100자리를 갖는 10진수를 2진수로 바꿔보면 비트 수가 약 300자리 정도가 됨
복잡도($2^{n_b/4}$)는 대략 2^{75} 이고 초당 2^{30} 연산 능력을 가진 컴퓨터를 이용하면 2^{45} 초가 걸림

소인수분해

- 방법

- 더 효율적인 방법

- 2차 체

- $x^2 \bmod n$ 의 값을 구함
 - 100자리 이상인 정수를 인수분해 가능
 - 복잡도 $O(e^C)$ 이고 $C \approx (\ln n \ln \ln n)^{1/2}$ 임
 - 부지수적인 복잡도를 가짐

- 정수체 체

- $x^2 = y^2 \bmod n$ 의 값을 구함
 - 120자리 이상인 정수 소인수 분해가 빠름
 - 복잡도 $O(e^C)$ 이고 $C \approx (\ln n)^{1/3} (\ln \ln n)^{2/3}$
 - 부지수적인 복잡도를 가짐

소인수분해

- 방법

- 더 효율적인 방법

- 예제 9.34

어떤 컴퓨터가 초당 2^{30} (거의 10억 번)번 정도의 비트 연산을 할 수 있다고 하자. 다음과 같은 방법을 사용했을 때 100자리 10진 정수를 소인수분해 하는데 걸리는 시간이 대략 얼마나 걸리는가?

- a. 2차 체 방법
- b. 정수체 체 방법

- 풀이

- 자릿수가 100인 10진 정수는 대략 2진수로 바꿔보면 비트 수가 약 300자리 정도임
 - $n = 2^{300}$ 이고, $\ln(2^{300}) = 207$ 이고 $\ln \ln(2^{300}) = 5$ 임
 - a. 2차 체 방법을 사용하면 $(207)^{1/2} \times (5)^{1/2} \approx 14 \times 2.23 \approx 32$ 임
 - e^{32} 번의 비트연산이 필요하고 주어진 컴퓨터를 이용하면 약 20시간 정도 걸림
 - b. 정수체 체 방법을 사용하면 $2 \times (207)^{1/3} \times (5)^{2/3} = 2 \times 5.9 \times 2.9 \approx 34$ 임
 - e^{34} 번의 비트연산이 필요하고 주어진 컴퓨터를 이용하면 약 140시간 정도 걸림

Thanks!

김민식(ms6127@naver.com)