해시(HASH)

해시 함수 요건

1. 임의 크기의 데이터 블록에 적용
2. 일정한 길이의 출력
3. 계산 용이성과 구현 가능성
4. 일방향 성질(one-way property)
5. 약한 충돌 저항성(weak collision resistance)
6. 강한 충돌 저항성(strong collision resistance)

일 방향 성질

- 주어진 값 h에 대하여 H(x)=h 가 성립되는 x를 찾는 것이 계산적으로 불가능 하다.

해시 충돌

- **해시 충돌**이란 해시 함수가 서로 다른 두 개의 입력값에 대해 동일한 출력값을 내는 상황을 의미한다.
- 해시 함수가 무한한 가짓수의 입력값을 받아 유한한 가짓수의 출력값을 생성하는 경우, 비둘기집 원리에 의해 해시 충돌은 항상 존재한다.
- 해시 충돌은 해시 함수를 이용한 자료구조나 알고리즘의 효율성을 떨어뜨리며, 따라서 해시 함수는 해시 충돌이 자주 발생하지 않도록 구성되어야 한다.
- 암호학적 해시 함수의 경우 해시 함수의 안전성을 깨뜨리는 충돌 공격이 가능할 수 있기 때문에 의도적인 해시 충돌을 만드는 것이 어렵도록 만들어야 한다.

해시함수 충돌 저항성

주어진 조건:해시 함수 H, 패스워드 x, y.

약한 충돌 저항성: 주어진 x에 대해, $H(x) = H(y)$인 $y \neq x$를 찾는 것이 어려울 때 해시 함수가 약한 충돌 저항성을 가지고 있다고 한다. 주어진 패스워드를 입력 값으로 해시 함수에 넣고, 그것을 초기 값 (x)이라고 하자. 만약 x에 대한 출력 값과 같은 출력 값을 갖는 또다른 패스워드를 찾을 확률이 해시 함수의 출력 값 범위 내에서 무시 가능(negligible)할 때 이 함수는 약한 충돌 저항성을 가지고 있다고 한다.

강한 충돌 저항성: $H(x) = H(y)$와 같이 같은 해시 출력 값을 갖는 x와 y를 찾는 것이 함수의 출력 값 범위 내에서 무시 가능(negligible)할 때 해시 함수 H는 강한 충돌 저항성을 가진다고 한다.

단순 해시 함수

	비트 1	비트 2	• • •	비트 n
블록 1	b_{11}	b_{21}		b_{n1}
블록 2	b_{12}	b_{22}		b_{n2}
	• • •	• • •	• • •	• • •
블록 m	b_{1m}	b_{2m}		b_{nm}
해시 코드	C_1	C_2		C_n

비트별 XOR 단순 해시 함수

- $$C_i = b_{i1} \oplus b_{i2} \oplus \cdots \oplus b_{im}$$

여기서

- C_i:해시코드의 i 번째 비트, $1 \leq i \leq n$
- M: 입력의 n 비트 블록의 수
- b_{ij} :j 번째 블록의 i 번째 비트
- $\oplus$: XOR 연산

- 충돌 저항성이 없다. Why?

MD5

MD5는 임의의 길이의 메시지(variable-length message)를 입력받아, 128비트짜리 고정 길이의 출력값을 낸다. 입력 메시지는 512 비트 블록들로 쪼개진다; 메시지를 우선 패딩하여 512로 나누어떨어질 수 있는 길이가 되게 한다. 패딩은 다음과 같이 한다: 우선 첫 단일 비트, 1을 메시지 끝부분에 추가한다. 512의 배수의 길이보다 64 비트가 적은 곳까지 0으로 채운다. 나머지 64 비트는 최초의(오리지널) 메시지의 길이를 나타내는 64 비트 정수(integer)값으로 채워진다.

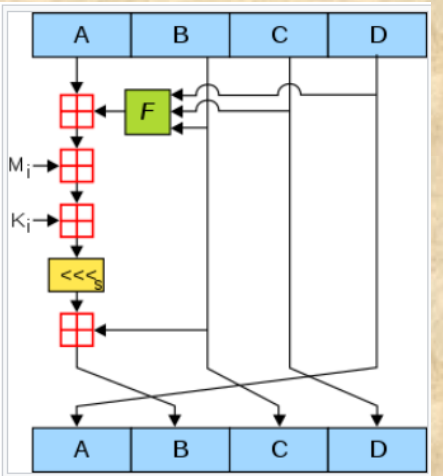
메인 MD5 알고리즘은 A,B,C,D라고 이름이 붙은 32 비트 워드 네 개로 이루어진 하나의 128 비트 스테이트(state)에 대해 동작한다. A,B,C,D는 소정의 상수값으로 초기화된다. 메인 MD5 알고리즘은 각각의 512 비트짜리 입력 메시지 블록에 대해 차례로 동작한다. 각 512 비트 입력 메시지 블록을 처리하고 나면 128 비트 스테이트(state)의 값이 변하게 된다.

하나의 메시지 블록을 처리하는 것은 4 단계로 나뉜다. 한 단계를 "라운드"(round)라고 부른다; 각 라운드는 비선형 함수 F, 모듈라 덧셈, 레프트 로테이션(left rotation)에 기반한 16개의 동일 연산(similar operations)으로 이루어져 있다. 오른쪽 그림은 한 라운드에서 이루어지는 한 연산(operation)을 묘사하고 있다.

함수 F에는 4가지가 있다; 각 라운드마다 각각 다른 F가 쓰인다:

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

$\oplus, \wedge, \vee, \neg$는 각각 XOR, 논리곱, 논리합 그리고 NOT 연산을 의미한다.



단일 MD5 연산. MD5에서는 이 단일 연산을 64번 실행한다. 16개의 연산을 그룹화한 4 라운드로 묶인다. F는 각 라운드에서 사용하는 비선형 함수를 가리키며, 각 라운드에서는 각각 다른 함수를 사용한다. M_i는 입력 메시지의 32-비트 블록을 의미한다.

$\lll_s$는 s칸 만큼의 레프트 로테이션을 가리키며, s는 각 연산 후 값이 변한다. ⊞ 은 모듈로 2^{32} 덧셈을 말한다.

현재는 MD5 알고리즘을 보안 관련 용도로 쓰는 것은 권장하지 않으며, 심각한 보안 문제를 야기할 수도 있다. 2008년 12월에는 MD5의 결함을 이용해 SSL 인증서를 변조하는 것이 가능하다는 것이 발표되었다.

SHA 안전 해시 함수

- 1993년에 FIPS PUB 180
- SHA-1
  - 1995년에 FIPS PUB 180-1: MD4 해시 함수에 기초, 설계: MD4를 모델
- SHA-2
  - SHA-256, SHA-384, SHA-512
- SHA-3:
  - 2008년에 수정된 문서가 FIP PUB 180-3
  - SHA-224

SHA 매개변수 비교

	SHA-1	SHA-256	SHA-384	SHA-512
메시지 다이제스트 길이	160	256	384	512
메지시 길이	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
블록 길이	512	512	1024	1024
단어 길이	32	32	64	64
단계 수	80	64	80	80
보안	80	128	192	256

SHA-512 (SHA2)

- 입력메시지 크기
  - 최대 길이가 2^{128} 비트 이하인 메시지
- 출력
  - 512비트 해시
- 처리 단위
  - 1024비트 블록

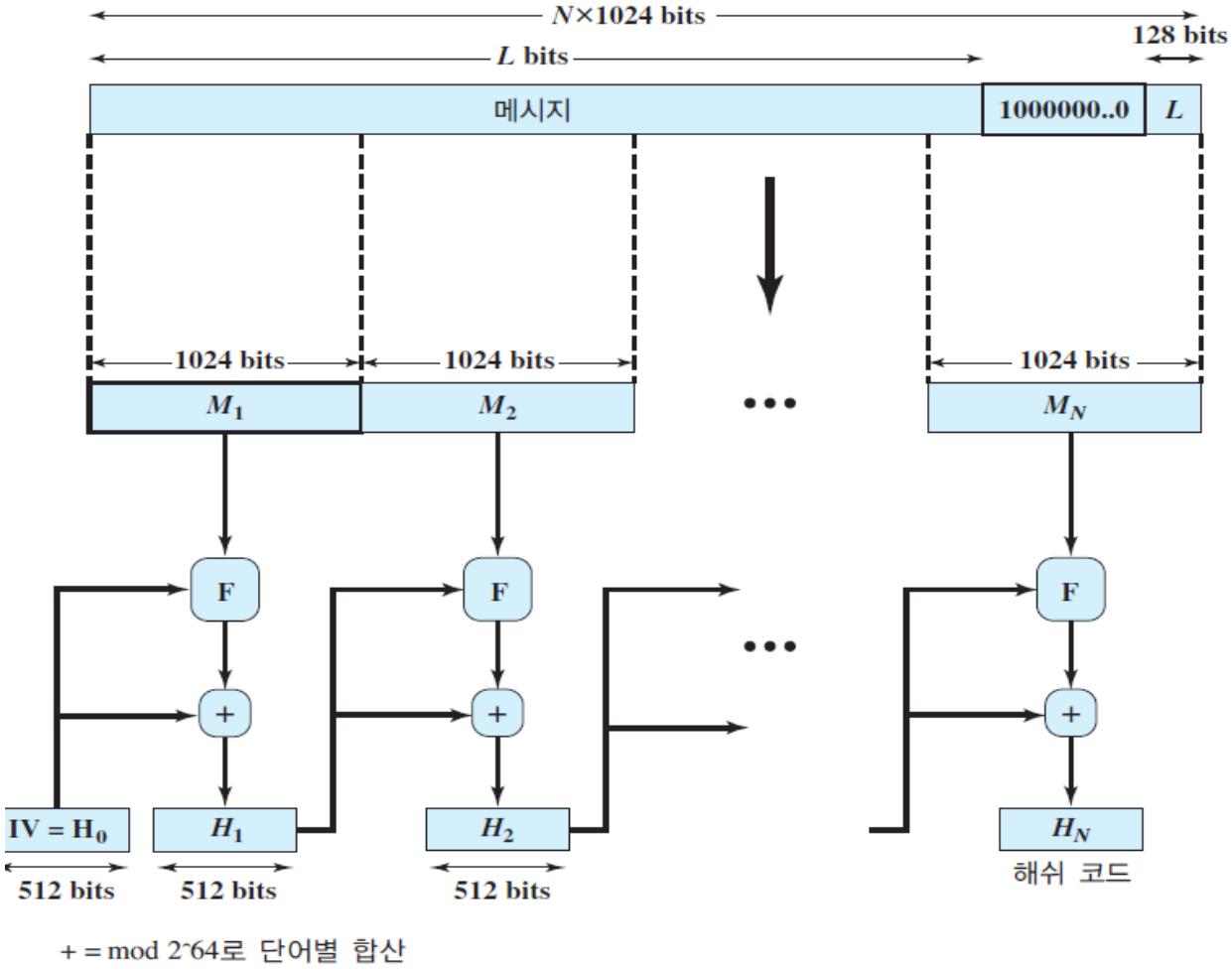
SHA-512 처리 단계

- **단계 1**: 패딩 비트 붙이기(Appending padding bits)
- **단계 2**: 길이 붙이기(Append length)
- **단계 3**: MD 버퍼 초기화(Initialize MD buffer)
- **단계 4**: 1024-비트 블록 메시지 처리(Process message in 1024-bit blocks)
- **단계 5**: 출력(Output)

패딩 비트 붙이기

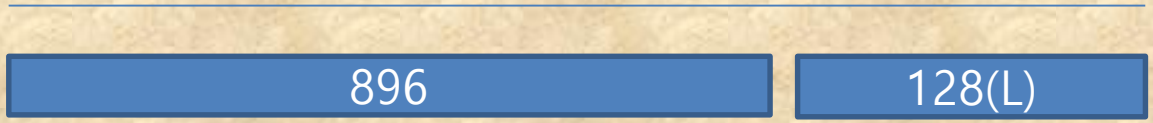
- 총 길이를 896 (mod 1024)가 되게 만든다
- 메지시 길이가 1024의 배수이어도 패딩을 추가
- 패딩을 구성하는 비트는 첫 번째 비트가 1이고 나머지 비트는 모두 0

SHA-512를 사용하는 메시지 다이제스트 생성



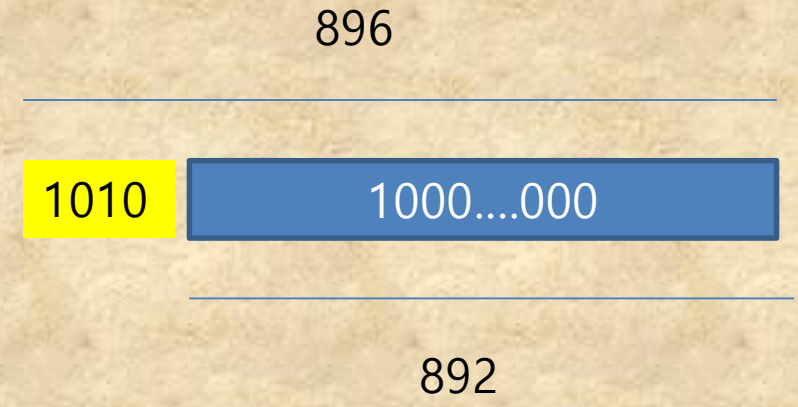
길이 붙이기

- 128 비트 블록을 메시지에 추가



예

- 메시지 길이가 1010일 때 패딩을 구성해 보아라

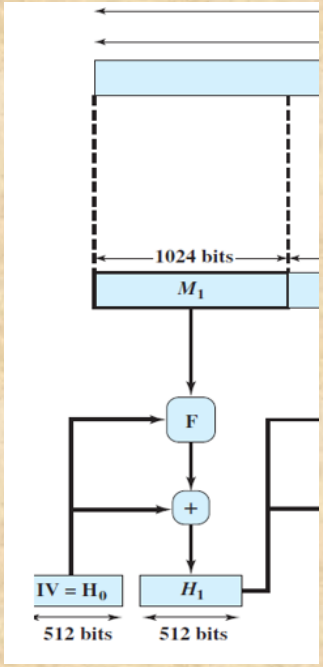


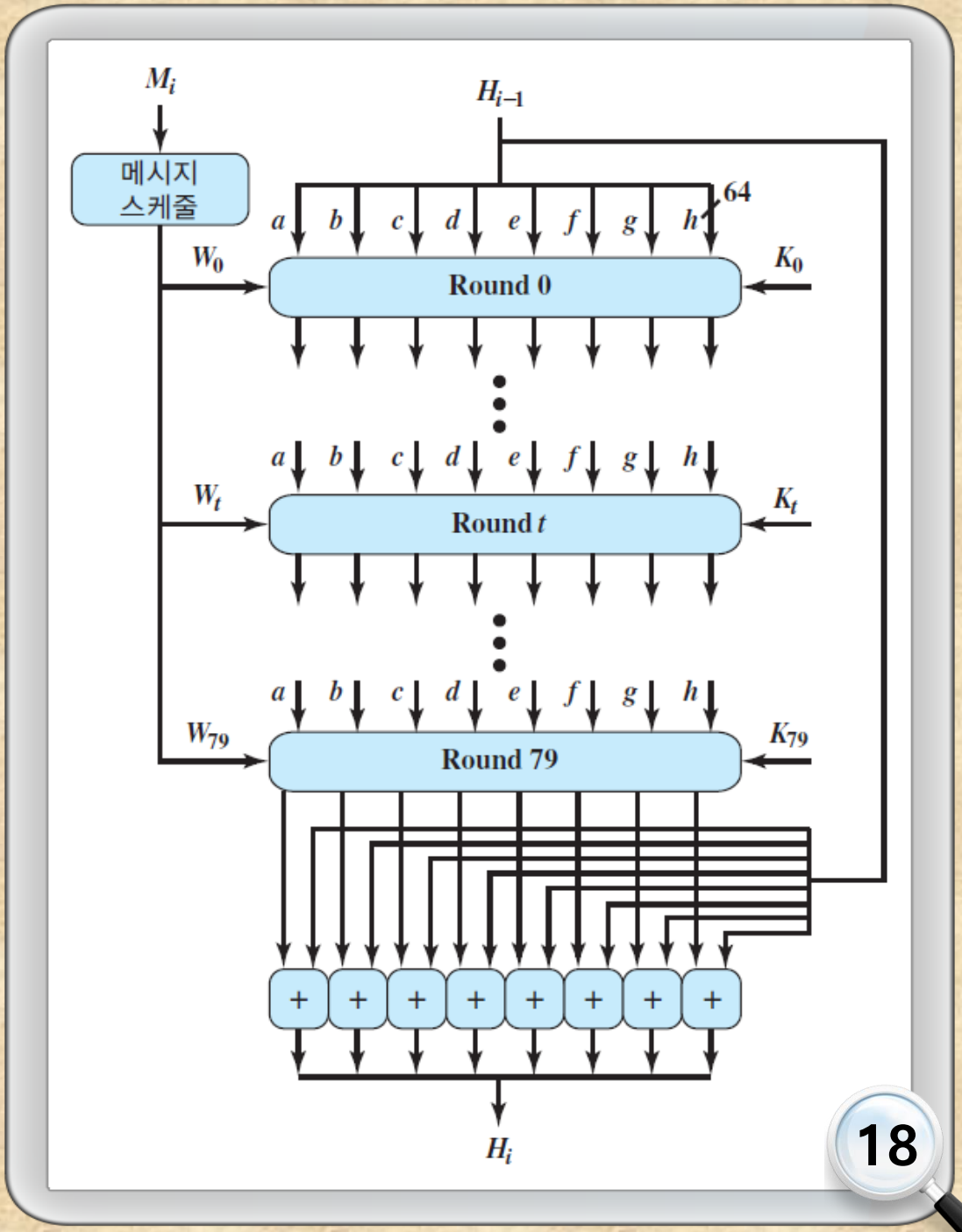
MD 버퍼 초기화

- 512-비트 버퍼를 해시함수의 중간 값과 최종 값을 저장하기 위해 사용
- 버퍼를 8 개의 64-비트 레지스터(a, b, c, d, e, f, g, h)로 나타낸다

a = 6A09E667F3BCC908	e = 510E527FADE682D1
b = BB67AE8584CAA73B	f = 9B05688CEB3E6C1F
c = 3C6EF372FE94F82B	g = 1F83D9ABFB41BD6B
d = A54FF53A5F1D36F1	h = 5BE0CDI9137E2179

SHA-512를 이용한 메시지 다이제스트 생성





1024-비트(128-워드)블록 메시지 처리

- 각 라운드가 80 라운드
  - 512-비트 버퍼 값인 abcdefgh를 입력으로 사용하고 이 버퍼의 내용을 갱신

출력

- N 개의 1024-비트 블록 모두가 처리된 뒤에 N 번째 단계에서 512-비트 메시지 다이제스트를 출력

해시 알고리즘 비교

<표 1> MD-SHA 패밀리 알고리즘 비교

알고리즘		해시크기(비트)	보안 레벨[8]	발표연도
MD5		128	충돌발견	1992
SHA-0		160	충돌발견	1993
SHA-1		160	충돌발견	1995
SHA-2	*SHA-224*	224	112	2001
	SHA-256	256	128	
	SHA-384	384	192	2001
	SHA-512	512	256	
	SHA-512/224	224	112	
	SHA-512/256	256	128	
SHA-3	*SHA3-224*	224	112	2015
	SHA3-256	256	128	
	SHA3-384	384	192	
	SHA3-512	512	256	
	SHAKE128	*d* (임의)	min(*d*/2, 128)	2015

Binary to Text

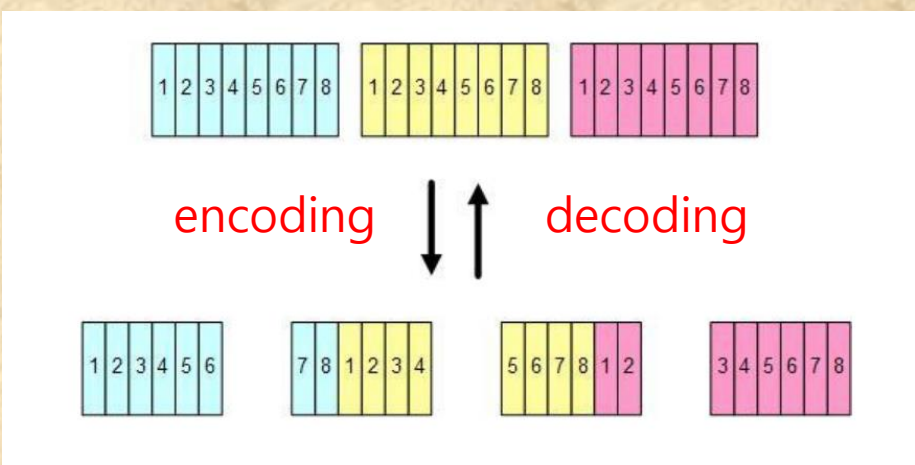
Binary to text

Base58	Integer	~73%	C++, Python	printed (0 - zero, I - capital i, O - capital o and l - lower case L). Satoshi Nakamoto invented the base58 encoding scheme when creating bitcoin.[1] Some messaging and social media systems line break on non-alphanumeric strings. This is avoided by not using URI reserved characters such as +. For segwit it was replaced by Bech32, see below.
				`// Copyright (c) 2009 Satoshi Nakamoto`
				`// Distributed under the MIT/X11 software license, see the accompanying`
				`// file license.txt or http://www.opensource.org/licenses/mit-license.php.`
				`//`
				`// Why base-58 instead of standard base-64 encoding?`
				`// - Don't want 0OIl characters that look the same in some fonts and`
				`//  could be used to create visually identical looking account numbers.`
				`// - A string with non-alphanumeric characters is not as easily accepted as an account number.`
				`// - E-mail usually won't line-break if there's no punctuation to break at.`
				`// - Doubleclicking selects the whole number as one word if it's all alphanumeric.`
				`//`
				`static const char* pszBase58 = "123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz";`
				Base58 in the original bitcoin source code

Bech32	1 bit (mainnet or testnet) plus 3 to 40 bytes	not a simple percentage since it has a 6-byte error correcting code	C, C++, JavaScript, Go, Python, Haskell, Ruby, Rust	Specification. Used in Bitcoin and the Lightning Network.[2]
Base62				Similar to Base64, but contains only alphanumeric characters.
Base64	Arbitrary	75%	awk, C, C (2), Python, many others	
Base85 (RFC 1924)	Arbitrary	80%	C, Python Python (2)	Revised version of Ascii85.
BinHex	Arbitrary	75%	Perl, C, C (2)	MacOS Classic

Base64 encoding/decoding

6-bit Value	Encoding	6-bit Value	Encoding	6-bit Value	Encoding	6-bit Value	Encoding
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/



24bit -> 6bit => 2^^6 = 64

그런데 입력되는 정보가 모두 3바이트씩 떨어진다는 보장이 없으므로 3바이트로 나누어떨어지지 않는 경우 = **문자로 채우기를 한다.** 즉 Base64로 인코딩 된 데이타에서 = 가 보이면 그 것은 다시 원래의 정보로 되돌아 갈때 (디코딩 될때) 아무 것도 없는 것이라는 소리가 된다.
(Base64로 인코딩 정보의 끝에 최대 나올 수 있는 = 의 수는 2개가 되겠다. 즉 끝부분에 =가 없거나 1개가 있거나 2개가 있는 것이 모두 나올 수 있는 경우가 되겠다.)

Q/A