암호알고리즘
Part II

NOTE 09

한국기술교육대학교 컴퓨터공학부 김상진

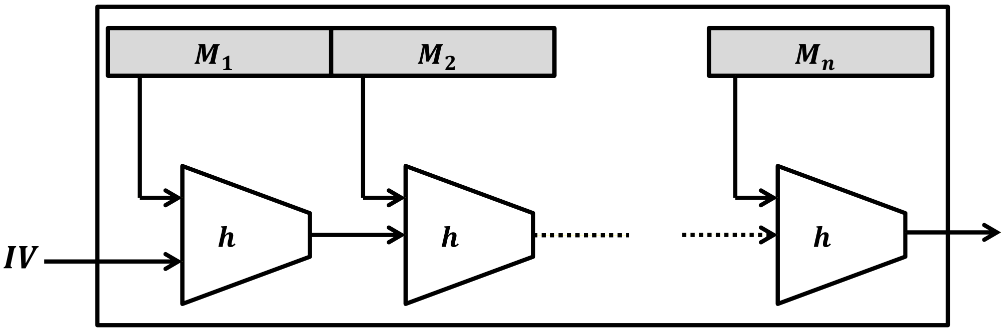
sangjin@koreatech.ac.kr
www.facebook.com/sangjin.kim.koreatech

교육목표

- 해시함수: SHA-1, SHA-2, SHA-3: KECCAK
- MAC
- 공개키 암호알고리즘: RSA, ElGamal
- 전자서명: RSA, ElGamal

충돌회피 – Merkle-Damgard 구조

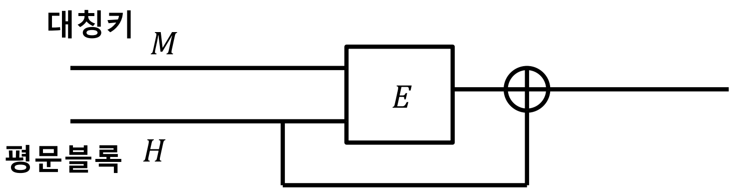
- Merkle-Damgard 구조에서 h가 충돌회피 함수이면 결과 H는 충돌회피 함수가 됨



  - 마지막 블록은 채우기를 해야 함
    - 1000…0||msg len
      - msg_len: 64비트
  - SHA-1에서 h는 512비트 블록과 160비트 입력을 받아 160비트를 출력함

블록 암호화를 이용한 충돌회피 함수

- Davis-Meyer 충돌회피 함수: $h(H, M) = E(M, H) \oplus H$



  - E가 안전한 블록 암호 함수일 때 이와 같은 형태로 함수를 만들면 h는 충돌회피 함수가 됨
    - $H_i = E.M_i(H_{i-1}) \oplus H_{i-1}$
  - SHA-2는 SHACAL-2라는 블록 암호를 이용하고 있음
    - 기존 살펴본 블록 암호 알고리즘과 달리 키 길이(512비트)가 블록 길이(256비트)보다 큼

Secure Hash Algorithm (1/2)

- SHA(Secure Hash Algorithm)
  - NIST에서 개발
  - 1993년에 표준으로 발표 (FIPS 180) ⟹ SHA-0
    - 1990년에 R. Rivest는 MD4를 개발하였고, 1992년에 이를 개선한 MD5를 개발함
  - SHA-0은 MD5와 마찬가지로 Merkle-Damgard 구조 기반 해시함수
  - 1995년에 개선된 버전 발표 (FIPS 180-1) ⟹ SHA-1
  - 2005년 Wang 등은 SHA-1에 대한 공격 발표
    - $O(2^{69})$의 비용으로 충돌을 찾을 수 있음
      - SHA-1의 해시값 길이는 160비트이므로 생일 파라독스에 의해 그것의 안전성은 $O(2^{80})$임
  - 2013년 Stevens은 $O(2^{61})$의 비용으로 충돌을 찾을 수 있음을 보임
  - 이 문제 때문에 SHA-2로 표준을 갱신함
    - SHA-2 해시값의 길이: 224, 256, 386, 512비트 (4종류의 버전 존재)

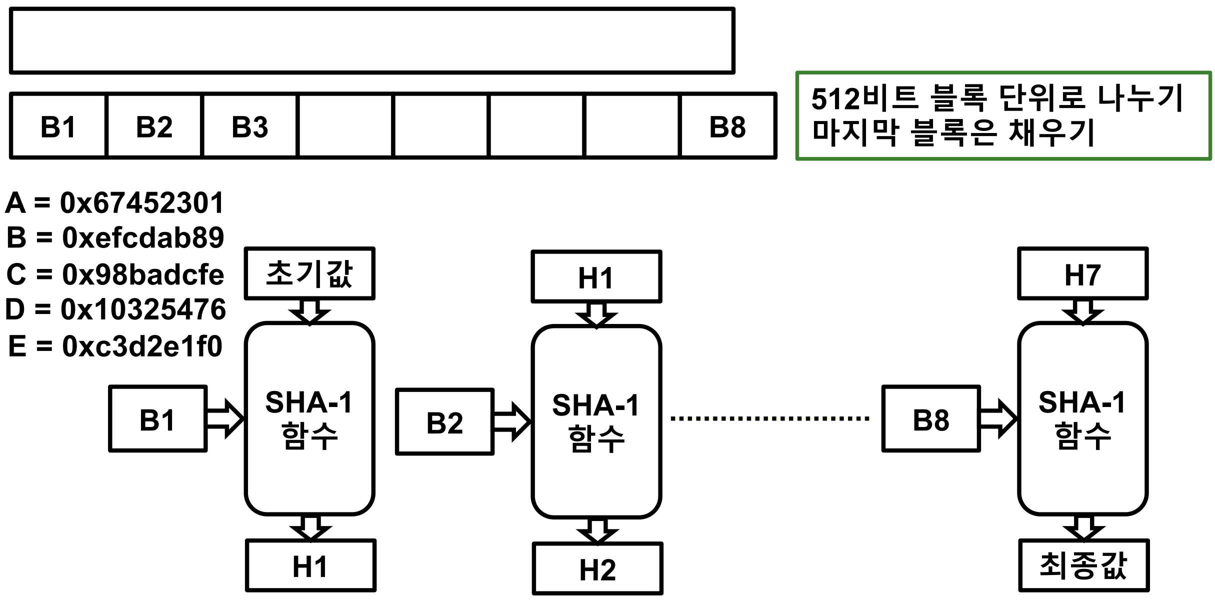
SHA (2/2)

- SHA의 문제점들이 계속 나타남에 따라 새 표준의 필요성이 제기되어 2008년부터 새 표준 선발 작업을 진행함
- 2015년에 KECCAK이라는 알고리즘이 SHA-3로 채택됨
  - SHA-2와 마찬가지로 224, 256, 386, 512비트 출력을 지원함
  - 알고리즘 구성이 기존 MD5, SHA 방식과 전혀 다르게 설계되어 있음

SHA-1 (1/2)

- 해시값의 길이: 160비트(20바이트, 5개의 32비트 워드)
- Merkle-Damgard 구조
- 입력: 2^{64}비트보다 작은 임의의 크기의 입력
  - 512비트(64바이트) 단위로 적용
  - 항상 채우기를 함
    - 마지막 64 비트에는 실제 크기를 기록함
    - 나머지는 비트 채우기를 함
      - 1000…000||msgLen
    - 블록 암호에서 채우기와 달리 나중에 채우기를 확인하고 제거할 필요는 없음
    - 하지만 동일 메시지를 해시하면 항상 해시값은 동일해야 함
      - 매번 같은 채우기를 사용해야 함

SHA-1 (2/2)



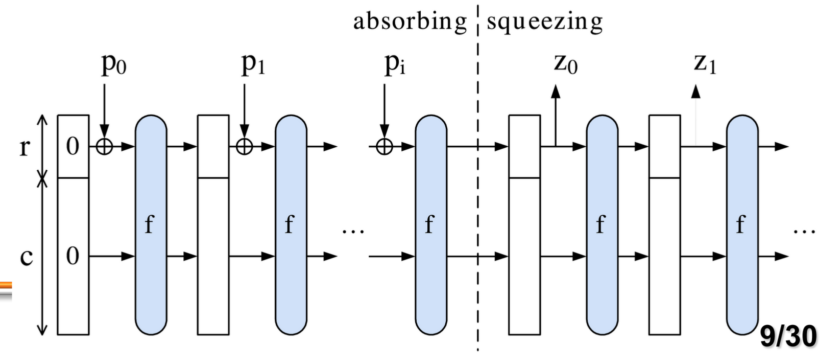
x가 512비트의 배수(채우기가 필요없으면)이면
H(x||y)는 초기값 대신에 H(x)를 초기값으로 사용하여
H(y)를 계산하는 것과 같음

SHA-3

- 입력 M을 채우기하여 길이가 r의 배수가 되도록 함
  - $(M||pad) = p_0|| \dots ||p_{n-1}$ (채우기는 10*1 사용)
- 모두 0인 b(1600)비트 초기값 S 준비 ($|f()| = b$)
- 각 p_i에 대해 다음을 수행
  - p_i가 b비트가 되도록 0으로 채우기하고, 이것을 S와 XOR한 값을 f함수의 입력으로 사용하여 새 S를 계산함
    - $S = f(S \oplus (p_i||0 \dots 0))$
- $Z = \emptyset$
  - S에서 r비트를 Z에 추가
  - 필요한 출력값을 얻을 때까지 다음을 반복
    - $S = f(S)$에서 r비트를 Z에 추가

스펀지 구조

SHA-3 256에서
r은 1088비트, c는 512비트



SHA-3

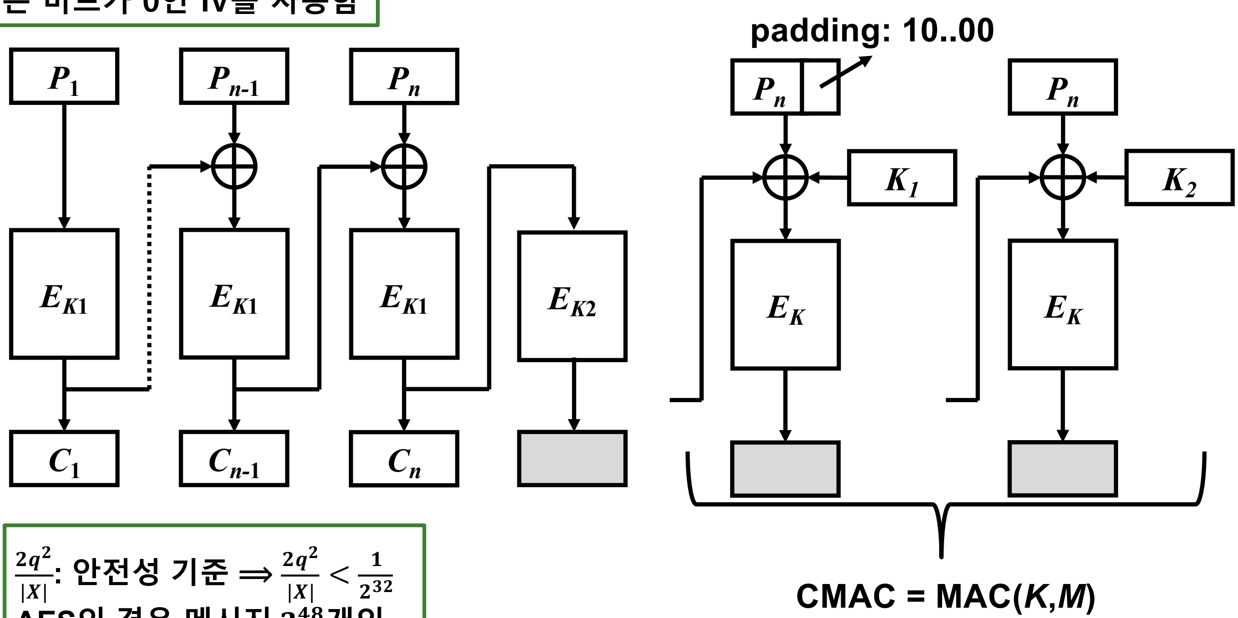
함수	출력 크기 d	블록 크기 r	용량 c	정의	안전성		
					충돌	역	약한충돌
SHA3-224(M)	224	1,152	448	Keccak[448](M||01, 224)	112	224	224
SHA3-256(M)	256	1,088	512	Keccak[512](M||01, 256)	128	256	256
SHA3-384(M)	384	832	768	Keccak[768](M||01, 384)	192	384	384
SHA3-512(M)	512	576	1,024	Keccak[1024](M||01, 512)	256	512	512
SHAKE128(M, d)	d	1,344	256	Keccak[256](M||1111, d)	min($d/2$, 128)	≥min(d, 128)	min(d, 128)
SHAKE128(M, d)	d	1,088	512	Keccak[512](M||1111, d)	min($d/2$, 256)	≥min(d, 256)	min(d, 256)

MAC을 구성하는 방법

- 전용 알고리즘
- 일반 대칭 암호알고리즘을 이용
  - CBC-MAC, NMAC(Nested MAC), CMAC, PMAC(Parallel MAC)
- 일반 해시함수를 이용
  - HMAC
- 무결성 검증 코드를 위한 대안 또는 동일 효과
  - 메시지 해시값을 대칭 암호: *E.K*(*H*(*M*))
    - 프로토콜 상호작용 문제
    - 블록 크기 문제
  - 메시지 해시값에 전자서명: Sig.-*K*(*H*(*M*))
    - 차이점
      - 부인방지는 더 확실함
      - 누구나 확인 가능
      - 고비용

CBC-MAC, CMAC

모든 비트가 0인 IV를 사용함



$\frac{2q^2}{|X|}$: 안전성 기준 $\Rightarrow \frac{2q^2}{|X|} < \frac{1}{2^{32}}$
AES의 경우 메시지 2^{48}개의 메시지에 대해 MAC을 계산한 이후에는 키를 변경해야 함

CMAC은 마지막 블록을 암호화하는 방법만 CBC-MAC과 차이가 있음
또한 K_1, K_2를 독립적으로 생성하지 않고 K를 이용하여 생성함

CBC-MAC

$C_i = E.K(P_i \oplus C_{i-1})$

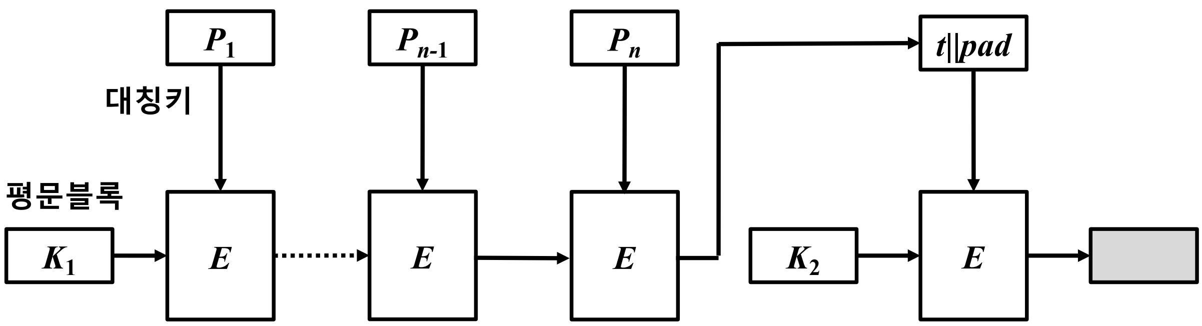
- 기본 CBC-MAC: 모든 비트가 0인 블록을 IV를 사용하여 메시지를 CBC 모드로 암호화하고 얻은 마지막 암호 블록을 MAC 값으로 사용
- 기본 CBC-MAC 위조 방법
  - 공격자가 $(x_1, \text{MAC}.K(x_1))$, $(x_2, \text{MAC}.K(x_2))$, $(x_1||z, \text{MAC}.K(x_1||z))$를 가지고 있으면 각 MAC 값은 다음과 같음

$$\text{MAC}.K(x_i) = E.K(x_i \oplus \text{IV}) = E.K(x_i)$$
$$\text{MAC}.K(x_1||z) = E.K(z \oplus E.K(x_1))$$

  - 공격자는 키를 알지 못하여도 $x_2||\text{MAC}.K(x_1) \oplus z \oplus \text{MAC}.K(x_2)$에 대한 MAC 값을 알 수 있으며, 이 MAC 값은 $\text{MAC}.K(x_1||z)$와 동일함
    - $E.K(E.K(x_1) \oplus z \oplus E.K(x_2) \oplus E.K(x_2)) = E.K(z \oplus E.K(x_1))$
  - 참고. 한 블록 크기의 메시지 X의 기본 CBC-MAC값이 T이면 $X||X \oplus T$의 기본 CBC-MAC값이 T임
  - 이 슬라이드 있는 내용은 채우기는 고려하고 있지 않음

NMAC(Nested MAC)

128비트 키, 128비트 블록 AES 알고리즘을 사용하면 마지막 채우기가 필요 없음

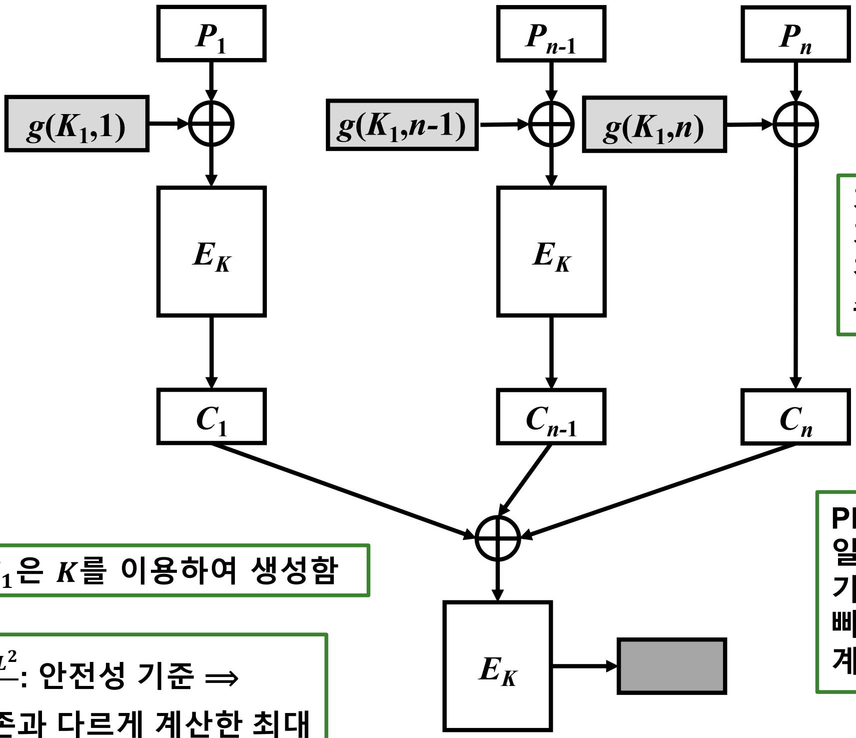


$\frac{q^2}{2|K|}$: 안전성 기준 ⇒
AES의 경우 메시지 2^{48}개의 메시지에 대해 MAC을 계산한 이후에는 키를 변경해야 함

$E: K \times X \rightarrow K$
$K \leq X$이면 C_n에 채우기를 하여 최종적 MAC값을 계산해야 함

문제점. AES와 같은 대칭암호알고리즘을 사용하면 매번 키 스케줄을 다시 해야 하기 때문에 성능이 상대적으로 좋지 않음

PMAC(Parallelizable MAC)



PMAC에서 g를 생략하면 문제점은?

기존 대칭 암호알고리즘 기반 MAC들과 달리 각 블록을 병행 처리할 수 있음

K_1은 K를 이용하여 생성함

PMAC은 incremental 일부 블록만 변경되면 기존 값을 이용하여 빠르게 새 MAC 값을 계산할 수 있음

$\frac{2q^2L^2}{|X|}$: 안전성 기준 ⇒ 기존과 다르게 계산한 최대 메시지 길이 L에도 의존함

$E(K, D(K,t) \oplus E(K, g(K_1,j) \oplus P_j) \oplus E(K, g(K_1,j) \oplus P'_j)$

One-time MAC ⟹ Many-time MAC

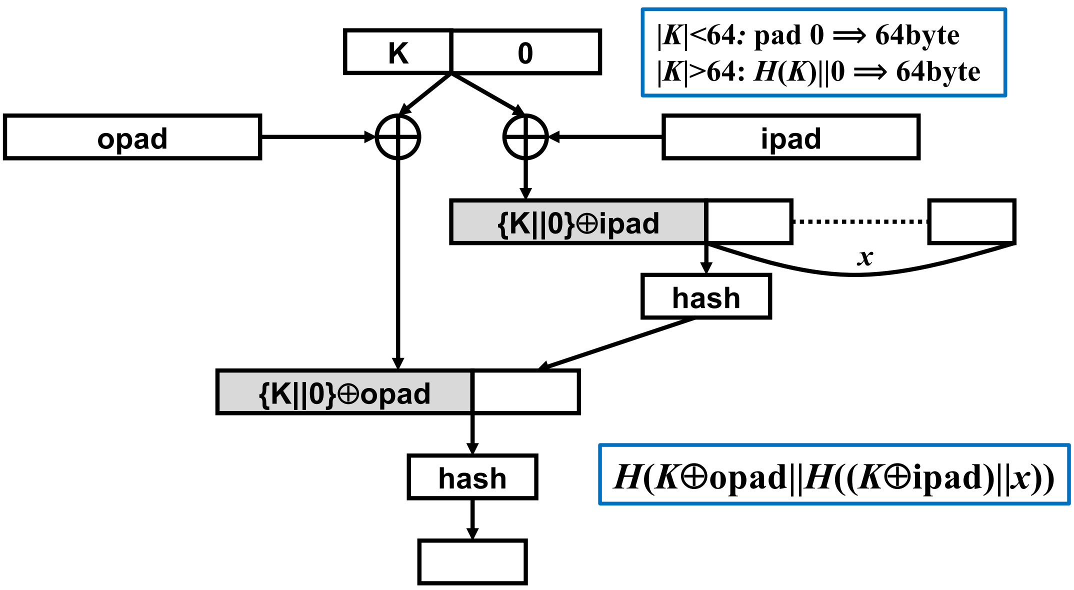
- 일회용 MAC: 키를 한 번만 사용하는 MAC
  - 앞서 살펴본 MAC 구조 형태보다 효율적으로 구성할 수 있음
- Carter-Wegman MAC
  - 일회용 MAC을 이용하여 여러 번 사용할 수 있는 MAC을 구성하는 방법
  - $CW(K_1, K_2, M) = \big(r, E(K_1, r) \oplus S(K_2, M)\big)$
    - E: 안전한 블록 암호 (PRF)
    - S: 안전한 일회용 MAC

HMAC

- 일반 해시함수를 이용한 MAC 구성
  - Merkle-Damgard 구조 해시 함수의 특성 때문에 단순하게 암호키를 추가하여 MAC을 구성할 경우 안전하지 못함
  - 예) $\mathrm{MAC}.K(M) = H(K||M)$ 형태
    - $\mathrm{MAC}.K(x)$를 이용하여 $\mathrm{MAC}.K(x||x')$을 계산할 수 있음
      - $\mathrm{MAC}.K(x)$를 초깃값으로 하여 $H(x')$을 계산하면 이 값이 $\mathrm{MAC}.K(x||x')$이 됨
  - 예) $\mathrm{MAC}.K(M) = H(M||K)$ 형태
    - H에 대한 충돌을 알고 있으면 MAC에 대한 충돌을 찾을 수 있음
      - $H(x) = H(x')$이면 $H(x||K) = H(x'||K)$임
  - $H(K||p||x||K)$ 형태로 하는 것이 안전함
    - p는 K를 해시 함수에 한 블록으로 만들기 위한 채우기 값임
      - 이것은 최소 두 번의 내부 연산이 수행되도록 하기 위한 것임
    - 다른 대안: $H\big(K_1||H(K_2||x)\big)$

HMAC 표준 (RFC 2104)

- ipad: 0x36…0x36 (입력길이 만큼, SHA-1의 경우 64바이트)
- opad: 0x5c…0x5c (입력길이 만큼, SHA-1의 경우 64바이트)
- 권장사항. $|K| > L$ (출력크기, SHA-1의 경우 20)



SHA-3와 HMAC

- SHA-3는 length-extension attack이 가능하지 않기 때문에 $MAC.K(M) = H(K||M)$ 형태로 MAC 값을 계산할 수 있음
  - length-extension attack: $H(x)$을 이용하여 $H(x||y)$를 계산할 수 있는 문제
- $MAC.K(M) = H(M||K)$ 형태는 여전히 안전하지 않음
  - H에 대한 충돌을 알고 있으면 MAC에 대한 충돌을 찾을 수 있음

RSA 공개키 암호알고리즘

- MIT의 Rivest, Shamir, Aldeman에 의해 1977년도에 개발
  - ACM Turing Award 수상 (2002년도)
- 인수분해 문제의 어려움에 기반한 알고리즘
- 현재 가장 널리 사용하고 있는 공개키 암호알고리즘

R. Rivest, A. Shamir, L. Adleman

암호알고리즘 안전성: 키 길이 비교

https://www.keylength.com/
NIST 기준 (2020)

시기	안전성	대칭 암호	인수분해법	이산대수		타원곡선	hash	MAC
				키	군			
Legacy	80		1,024	160	1,024	160	SHA-1	
2019 - 2030	112	AES-128	2,048	224	2,048	224	SHA-224 SHA-512/224 SHA3-224	
2019 - 2030 & 그 이후	128	AES-128	3,072	256	3,072	256	SHA-256 SHA-512/256 SHA3-256	SHA-1 KMAC128
2019 - 2030 & 그 이후	192	AES-192	7,680	384	7,680	384	SHA-384 SHA3-384	SHA-224 SHA-512/224 SHA3-224
2019 - 2030 & 그 이후	256	AES-256	15,360	512	15,360	512	SHA-512 SHA3-512	SHA-256 SHA-512/256 SHA-384 SHA-512 SHA3-256 SHA3-384 SHA3-512 KMAC256

RSA 공개키 쌍 생성 및 암/복호화

- 단계 1. 두 개의 소수 p와 q를 선택
  - 같은 크기의 소수 선택, 크기는 원하는 n크기의 반
  - 소수는 랜덤하게 생성한 후 소수검사를 통해 생성
- 단계 2. $n = pq$를 계산
- 단계 3. $\phi(n) = (p-1)(q-1)$를 계산

e = 3, 65537

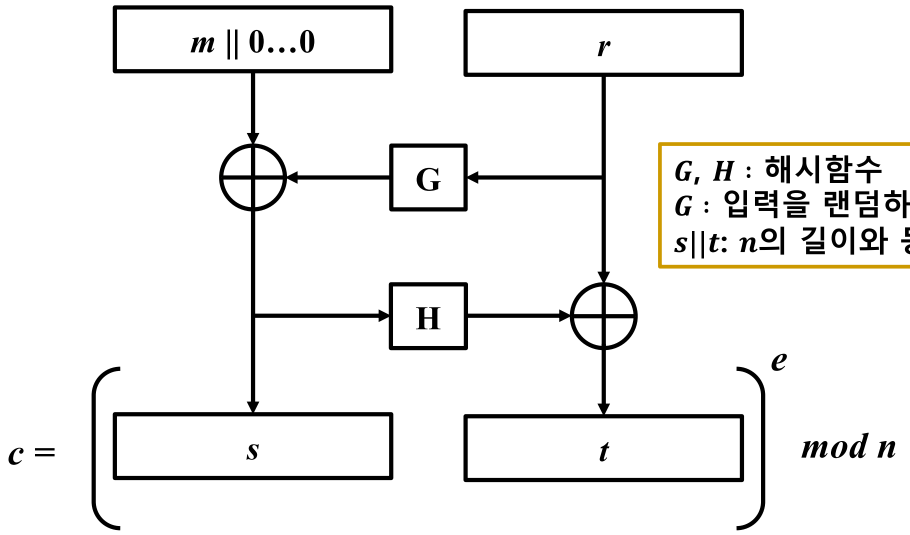
- 단계 4. $\gcd(e, \phi(n)) = 1$인 $e \le \phi(n) - 1$ 를 임의로 선택
- 단계 5. $ed \equiv 1 \pmod{\phi(n)}$인 $d \le \phi(n) - 1$를 계산
  - 공개키: (n, e), 개인키: (n, d)
  - 키 쌍을 구한 다음에는 p와 q는 더 이상 필요가 없음
    - 더 빠르게 암호화하기 위해 p와 q를 활용하기도 함
- 암호화: $M < n$

$$C = M^e \mod n$$

- 복호화
  - $C^d = (M^e)^d = (M^{ed}) = M^{k\phi(n)+1} = (M^{\phi(n)})^k M \equiv M \pmod{n}$

OAEP RSA

- OAEP(Optimal Asymmetric Encryption Padding) RSA



G, H : 해시함수
G : 입력을 랜덤하게 확장 필요
$s||t$: n의 길이와 동일

- 암호화 속도를 향상하기 위해 공개키 값으로 e = 3을 사용하는 경우가 많음
- 이때 메시지를 직접 이 공개키로 거듭제곱하면 결과 암호문 C로부터 평문을 계산할 수 있음 (예, $M = 3$이면 $C = 27$임)
- 이 문제를 극복하기 위해 OAEP를 사용함

RSA-KEM

- RSA-OAEP 대신에 혼합 암호화를 하는 경우도 많음
- RSA-KEM 동작 원리
  - 상대방 공개키: (n, e)
  - $x \in_R \mathbb{Z}_n^* \Rightarrow K_1||K_2 = KDF(x)$
  - $c = x^e\ mod\ n, C = \{M\}.K_1, MAC.K_2(C)$

ElGamal 공개키 암호알고리즘 (1/2)

- 이산대수 기반 공개키 암호알고리즘
- 확률 알고리즘(probabilistic algorithm): 같은 메시지를 암호화하여도 결과가 다름
  - 비고. 결정 알고리즘(deterministic algorithm)
    - 대칭 암호알고리즘에서는 이런 문제 때문에 암호모드를 사용
- 환경 설정 1. (군 $\mathbb{Z}_p^*$을 사용)
  - 단계 1. 2048비트 정도의 소수 p를 선택함
  - 단계 2. $\mathbb{Z}_p^*$의 생성자 g를 선택함

$\mathbb{Z}_7^* = \langle 3 \rangle = \{1,2,3,4,5,6\}$
$\mathbb{G}_3 = \langle 2 \rangle = \{1,2,4\}$

- 환경 설정 2. ($\mathbb{Z}_p^*$의 부분군 $\mathbb{G}_q$군을 사용)
  - 단계 1. 224비트 정도의 소수 q를 선택함
  - 단계 2. $p = kq + 1$이 2048비트 정도의 소수가 되도록 랜덤 k를 반복적으로 선택함
  - 단계 3. $\mathbb{G}_q$군의 생성자 g를 선택함
    - 참고. $\mathbb{Z}_p^*$에서 임의의 수 h를 선택한 후 $g = h^k$를 계산함. 이 값이 1이 아니면 g는 $\mathbb{G}_q$군의 생성자임

ElGamal 공개키 암호알고리즘 (2/2)

- 실제 응용: 랜덤하게 군 환경을 설정하지 않음
  - 약한 군을 사용할 확률이 있음
  - 표준에 제시된 군 사용 (RFC 7919), 타원곡선 이용
- 환경설정 2에서 ElGamal 암호화/복호화 알고리즘
  - 개인키: $x \in \mathbb{Z}_q^*$
  - 공개키: $y = g^x \mod p$
  - 암호화: $M \in \mathbb{G}_q$의 암호문 (A, B)
    - $r \in_R \mathbb{Z}_q^*$를 선택한 다음 $(A, B) = (g^r, y^r M)$을 계산
  - 복호화
    - $B(A^x)^{-1} \equiv y^r M g^{-rx} \equiv g^{rx} M g^{-rx} \equiv M \pmod{p}$
- 실제 응용에서는 주로 혼합 암호를 사용
  - $g^r, C = E.K_1(M), \mathrm{MAC}.K_2(C)$
    - 여기서 $K_1 || K_2 = KDF(y^r)$임 (RSA-KEM과 유사)

RSA 전자서명

- $M < n$에 대해 직접 개인키로 암호화하는 것은 위조가 가능하기 때문에 전자서명으로 적합하지 않음
  - 예) $M = 1$이면 개인키 값과 상관없이 누구나 위조 가능
  - 예) $\sigma_1 = M_1^d \mod n$, $\sigma_2 = M_2^d \mod n$이면 $\sigma_1\sigma_2$는 M_1M_2에 대한 유효한 서명이 됨
  - 예) $\sigma < n$를 임의로 선택하면 $M = \sigma^e$에 대한 유효한 서명임
- 이와 같은 문제점과 $M > n$보다 클 수 있으므로 해시 함수를 이용함
  - 이때 안전성을 높이기 위해 해시값의 길이가 RSA 법 길이와 같아야 하며, 해시함수가 충돌회피 해시함수이어야 함
  - RSA-PSS(Probabilistic-Signature Scheme)

$$r \in_R \{0,1\}^{k_0}$$
$$w = H(M||r)$$
$$M' = 0||w||(G_1(w) \oplus r)||\ G_2(r)$$
$$\sigma = M'^d \mod n$$

$$M' = \sigma^e \mod n$$
$$\text{split } M' = b||w||\alpha||\gamma$$
$$r = \alpha \oplus G_1(w)$$
$$b \mathrel{?=} 0$$
$$G_2(r) \mathrel{?=} \gamma$$
$$H(M||r) \mathrel{?=} \mathrm{w}$$

DSA

- 미국 표준 DSA는 RSA, Schnorr의 특허 문제로 Schnorr를 변형한 알고리즘을 사용하게 되었음

- 224비트 정도의 소수 q 선택,
- $p = kq + 1$가 2048비트 소수가 되도록 q를 반복적으로 선택
- 충돌회피 해시함수: $H_q: \{0,1\}^* \rightarrow \mathbb{Z}_q^*$
- A의 서명키: $x \in_R \mathbb{Z}_q^*$
- A의 확인키: $y = g^x \bmod p$

$$W ?\equiv g^{u_1}y^{u_2} \pmod p$$
$$g^w ?\equiv g^{H_q(m)s^{-1}}g^{xWs^{-1}} \pmod p$$
$$g^w ?\equiv g^{s^{-1}}g^{(H_q(m)+xW)} \pmod p$$
$$g^w ?\equiv g^{w(H_q(m)+xW)^{-1}}g^{(H_q(m)+xW)} \pmod p$$
$$g^w ?\equiv g^w \pmod p$$

DSA	
서명 생성	서명 검증
Step 1. $w \in_R \mathbb{Z}_q^*$ Step 2. $W = (g^w \bmod p) \bmod q$ Step 3. $s = (H_q(m) + xW)w^{-1} \bmod q$	Step 1. $u_1 = H_q(m)s^{-1} \bmod q$ Step 2. $u_2 = Ws^{-1} \bmod q$ Step 3. $W ?\equiv (g^{u_1}y^{u_2} \bmod p) \pmod q$

m, W, s

ECDSA

- 안전한 타원곡선의 선택: Secp256k1, Curve25519
- 타원곡선 군의 부분군 위수: 소수 q
- 타원곡선 군의 부분군 생성자: P
- 충돌회피 해시함수: $H_q: \{0,1\}^* \rightarrow \mathbb{Z}_q^*$
- A의 서명키: $x \in_R \mathbb{Z}_q^*$
- A의 확인키: $Q = xP$

ECDSA	
서명 생성	서명 검증
Step 1. $w \in_R \mathbb{Z}_q^*$ Step 2. $(a, b) = wP$ Step 3. $s = (H_q(m) + xa)w^{-1} \mod q$	Step 1. $(\tilde{a}, \tilde{b}) = H_q(m)s^{-1}P + as^{-1}Q$ Step 2. $\tilde{a} ? \equiv a$

W, a, s $\Rightarrow$

EdDSA

- 타원곡선: Edwards25519
- 타원곡선 군의 부분군 위수: 소수 q
- 타원곡선 군의 부분군 생성자: P
- 충돌회피 해시함수: H: SHA−512
- A의 서명키: $K \in_R \mathbb{Z}_n^*$, $x||r = H(K)$
- A의 확인키: $Q = xP$

- Schnorr 서명의 특허 만료로 Schnorr 서명에 가깝게 바뀜

- w를 랜덤하게 생성하지 않고 메시지를 이용하여 계산
- 동일 w의 사용을 방지하는 것이 목적

ECDSA	
서명 생성	서명 검증
Step 1. $w = H(r\|\|M) \mod q$ Step 2. $W = wP$ Step 3. $c = H(W\|\|Q\|\|M) \mod q$ Step 4. $s = w + cx \mod q$	Step 1. $\tilde{c} = H_p(W\|\|Q\|\|M) \mod q$ Step 2. $sP ? \equiv W + \tilde{c}Q$

W, M, s $\Rightarrow$