공개키 암호 알고리즘

컴퓨터시스템보안

금오공과대학교 컴퓨터공학부
최태영

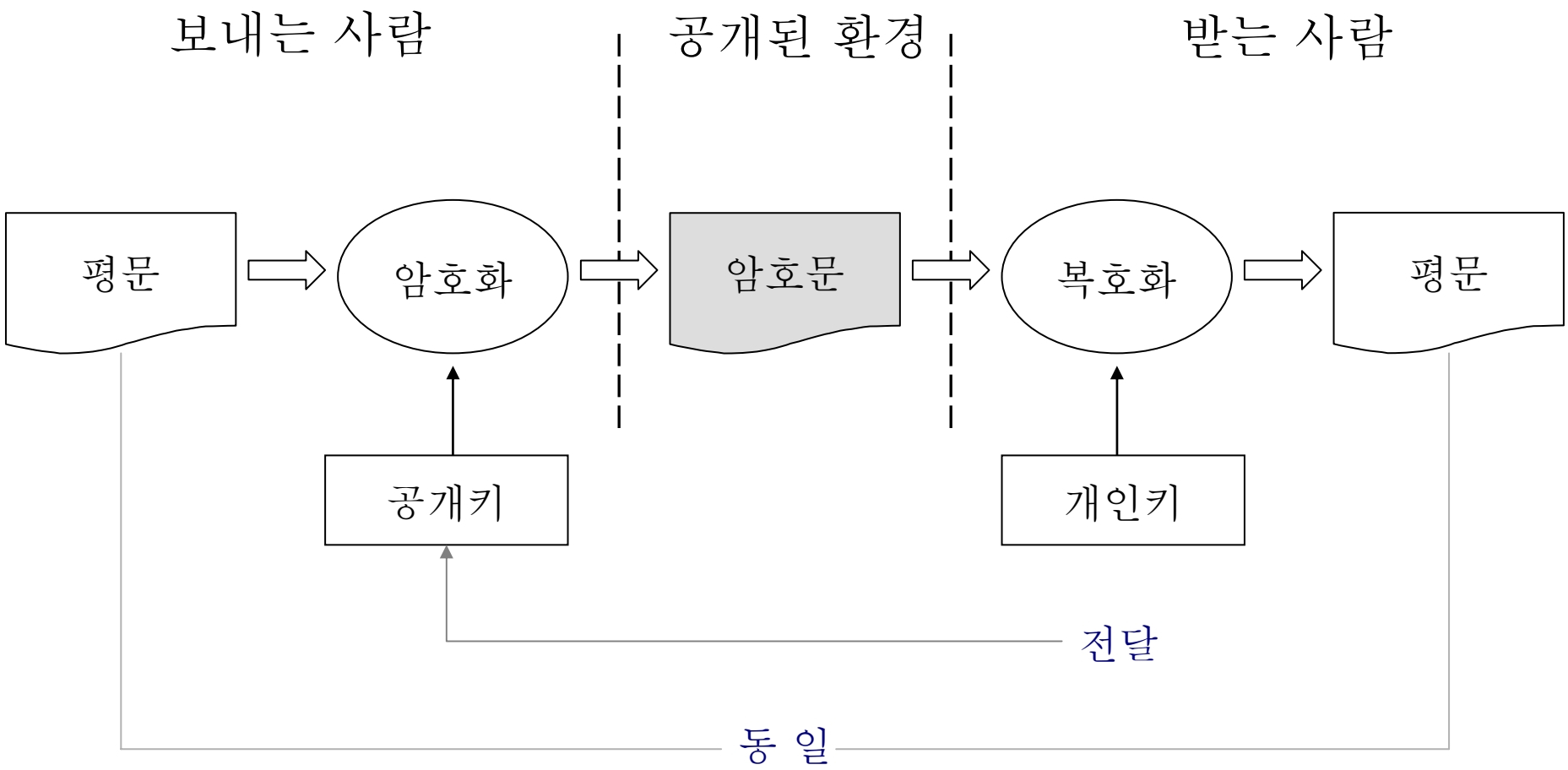
목차

- 키관리
- 모듈러 연산
- RSA 알고리즘
- 디피-헬먼 키 교환 알고리즘
- 전자봉투
- OpenSSL을 통한 공개키 암호화

공개키 암호의 필요성

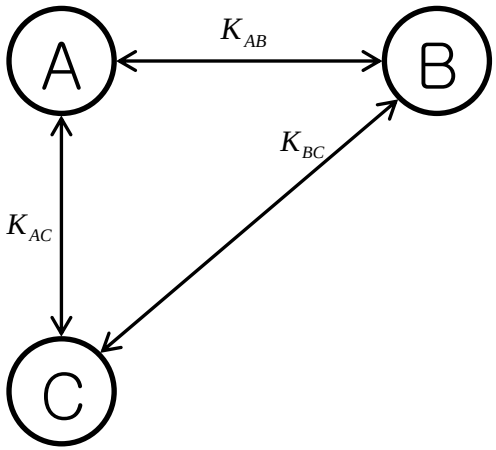
- 대칭키 / 비밀키 암호의 문제점
  - 키 전달 비용이 높음
    - 두 사용자들 중 한 명이 키를 만들어 상대방에게 전달해야 함
- 최소한 암호화를 위한 키는 쉽게 전달할 필요가 있음
- 비대칭키 / 공개키 암호
  - 공개키는 공개적으로 배포되며, 암호화에 사용됨
  - 암호문은 개인키로 복호화
  - 메시지를 받는 사람이 (공개키/개인키)쌍을 생성하여 공개키를 저장, 개인키는 보관
  - 키 전달 비용이 저렴함

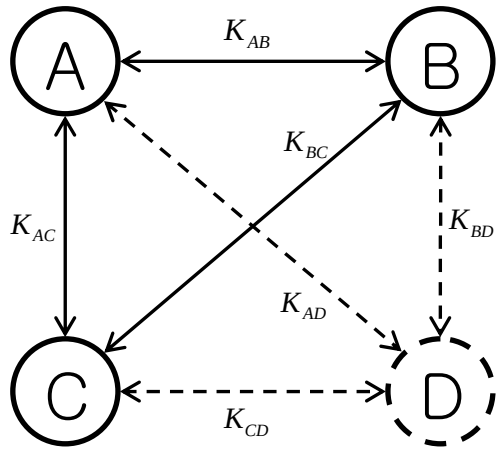
공개키 암호 시스템의 구조



비밀키 관리

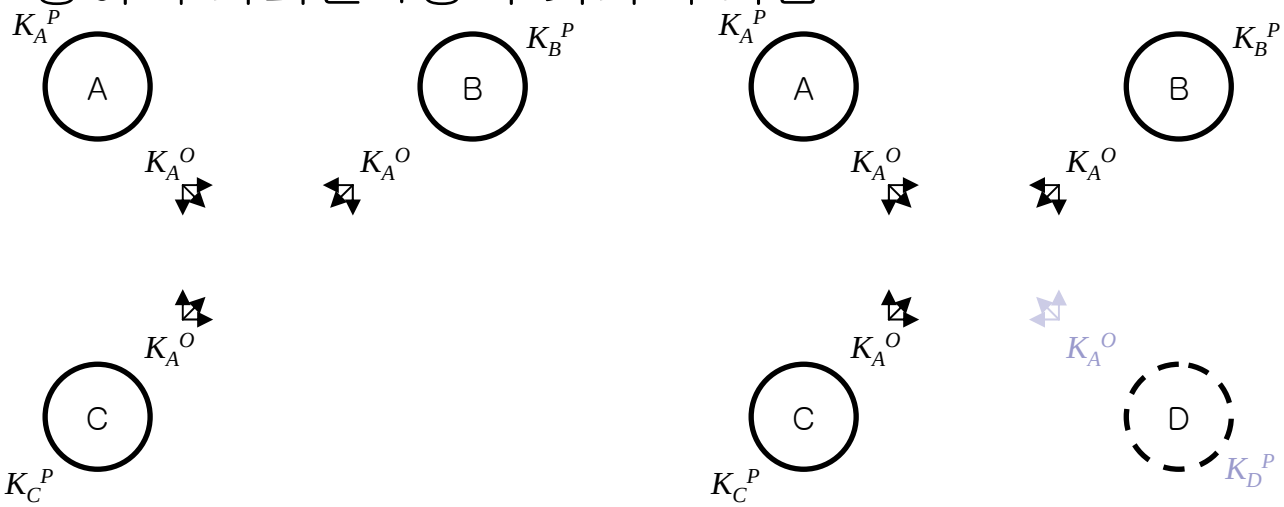
- 2명의 사용자 간에 1개의 비밀키 사용
- 3명의 사용자 간에 3개의 비밀키 사용
  - 사용자 A, B, C에 대해 비밀키 K_{AB}, K_{AC}, K_{BC} 필요
- 사용자 1명 추가 시 3개의 비밀키가 추가로 필요
  - 추가 사용자 D에 대해 K_{AD}, K_{BD}, K_{CD} 추가
- n명의 사용자에 대해 n(n+1)/2개의 키 필요
  - 1명이 추가되면 n개의 키가 추가로 필요





공개키 관리

- 1명의 사용자에게 2개의 키, 또는 한 쌍의 키 필요
- 3명의 사용자에 대해 3 쌍의 키 필요
  - (K_A^o, K_A^p), (K_B^o, K_B^p), (K_C^o, K_C^p)
- 1명의 추가 사용자에 대해 1 쌍의 키 추가
  - 사용자 D에 대해 (K_D^o, K_D^p) 추가
- n 명의 사용자에 대해 n 쌍의 키 필요
  - 1 명이 추가되면 1쌍의 키가 추가됨



소수와 인수분해

- 소수 : 1과 자신으로만 나누어지는 자연수
  - 2, 3, 5, 7, 11, 13, ...
- 합성수 : 소수가 아닌 수
  - 인수분해가 됨 (ex. 105 = 15×7)
- 인수분해 알고리즘 (자연수 n에 대한)
  - 기본형태 : 2부터 $\sqrt{n}$가 될 때까지 나누어 본다.
  - 시간복잡도 :
    - 입력의 크기 : $\log_2 n = l$
    - 시간복잡도 : $O(2^l)$
  - 시간복잡도가 상수 k에 대해 $O(i^k)$인 알고리즘은 알려져 있지 않다.
  - 1024 비트 크기 수의 인수분해 시간
    - 인수분해 연산 : 초당 2×10^5 회 가정
    - 수행 회수 : 약 $2^{512} \approx 4\times10^{153}$ 회
    - 약 6.6×10^{139}년 필요

모듈러 연산

- $a \equiv b \pmod n$
  - a와 b는 모듈로 n에 대해 동치이다.
  - $a - b = k \cdot n$인 정수 k가 존재, (ex. $38 \equiv 14 \pmod{12}$)
- $\bar{a}_n$: 모듈로 n에 대한 a의 동치 클래스
  - $\bar{a}_n = \{\cdots, a-2n, a-n, a, a+n, a+2n, \cdots\}$
- $Z_n = \{\bar{a}_n | a \in Z\}$
- 덧셈과 곱셈 연산
  - a, b $\in Z_n$의 덧셈 : $a + b \pmod n$
  - a, b $\in Z_n$의 곱셈 : $a \times b \pmod n$
- 항등원
  - $a \in Z_n$에 대해 $a + 0 \equiv 0 + a \equiv a \pmod n$
  - $a \in Z_n$에 대해 $a \times 1 \equiv 1 \times a \equiv a \pmod n$
- 역원
  - $a \in Z_n$에 대해 $a + (-a) \equiv a + (n - a) \equiv n \equiv 0 \pmod n$
  - n이 소수이면, $a \in Z_n$에 대해 $a \times a^{-1} \equiv 1 \pmod n$
  - gcd(a,n) = 1 인 경우에도 $a^{-1} \pmod n$이 존재함

최대공약수 구하기 (유클리드 호제법)

gcd(378, 243) = ?

r_{-2} = 378, r_{-1} = 243으로 설정

$r_i = r_{i-2} \% r_{i-1}$

i	r_i	q_i
−2	378	
−1	243	
0	135	1
1	108	1
2	27	1
3	0	4

Z_n에서 역원 구하기 (확장유클리드호제법)

- 입력 : 자연수 a과 n (a < n)
- $r_{-2} = n$; $r_{-1} = a$;
- $u_{-2} = 1$; $u_{-1} = 0$;
- $v_{-2} = 0$; $v_{-1} = 1$;
- i는 0부터 1씩 증가
  - $q_i = r_{i-2} / r_{i-1}$;
  - $r_i = r_{i-2} \% r_{i-1}$;
  - $u_i = u_{i-2} - q_i \times u_{i-1}$;
  - $v_i = v_{i-2} - q_i \times v_{i-1}$;
  - r_i가 0이면 r_{i-1}이 gcd(a, n)이 됨
  - r_i가 1이면 v_i가 a의 역원이 됨

확장 유클리드 호제법의 예

- $15^{-1} \bmod 442$
  - a = 15
  - n = 442
  - $442 \cdot u + 15 \cdot v = 1$
  - $442 \cdot (-2) + 15 \cdot 59 = 1$

i	u	v	r	q
−2	1	0	442	
−1	0	1	15	
0	1	−29	7	29
1	−2	59	1	2

확장 유클리드 호제법의 원리

- $a \cdot a^{-1} - k \cdot n = 1$
  - $a \cdot a^{-1} \equiv 1 \pmod{n}$이라면
  - $a \cdot a^{-1} - 1 = k \cdot n$을 만족하는 정수 k가 존재함
- 아래의 두 식에서 위 식을 유도
  - $n \cdot 1 + 0 \cdot a = n \quad (-2_{nd})$
  - $n \cdot 0 + 1 \cdot a = a \quad (-1_{st})$
- (i번째 식) = (i-2번째 식) – (i-1번째 식)·q_i
  - q_i는 i-2번째 오른 변을 i-1번째 오른 변으로 나누었을 때의 몫
- r_i가 1이 될 때까지 반복함

u	v	r	q
$n \cdot 1 +$	$0 \cdot a =$	n	
$n \cdot 0 +$	$1 \cdot a =$	a	
$n \cdot u_{i-2} +$	$v_{i-2} \cdot a =$	r_{i-2}	q_{i-2}
$n \cdot u_{i-1} +$	$v_{i-1} \cdot a =$	r_{i-1}	q_{i-1}
$(u_{i-2} - q_i \cdot u_{i-1}) \cdot n +$	$(v_{i-2} - q_i \cdot v_{i-1}) \cdot a =$	$r_{i-2} - q_i \cdot r_{i-1}$	

RSA 알고리즘

- 개요
  - 1973년 영국 GCHQ의 Clifford Cocks가 개발 (공개되지 않음)
  - 1978년 MIT의 Ron Rivest, Adi Shamir와 Len Adleman에 의해 개발
  - 1983년에 미국 특허 획득, 2000년에 Public domain으로 등록
- 구성
  - 키 생성 : 받는 사람이 키 쌍을 생성하여 공개키를 공개 채널을 통해 상대방에게 전달, 개인키는 보관
  - 암호화
  - 복호화

RSA 알고리즘 (계속)

- 키 생성
  1. 두 개의 큰 소수 P와 Q를 고른다.
  2. $N = P \times Q$를 계산한다.
  3. $\Phi(N) = (P - 1) \times (Q - 1)$이라 할 때, $\Phi(N)$과 서로소인 수 E를 선택하고 이를 공개키로 한다.
  4. 개인키 D는 $E^{-1} \bmod \Phi(N)$. 즉, $E \times D \equiv 1 \pmod{\Phi(N)}$
- 암호화 : $C \equiv M^E \pmod N$
- 복호화 : $M \equiv C^D \pmod N$

RSA 알고리즘 예제

- 키생성
  - 소수 P = 17, Q = 23을 선택
  - N = P × Q = 17 × 23 = 391,
    - $\Phi(N) = (P - 1) \times (Q - 1) = (17-1) \times (23-1) = 352$
  - E = 3 선택, E와 Φ(N)가 서로 소인지 확인
  - $D = 3^{-1} \bmod 352 = 235$
  - 공개키 (3, 391), 개인키 (235, 391)
- 암호화 (평문 75를 암호화)
  - $C = 75^3 \pmod{391} = 377$
- 복호화
  - $M = 377^{235} \pmod{391} = 75$

RSA 암호 알고리즘의 안전성

- 상황
  - 공격자는 공개키 (E, N), 암호문 C를 쉽게 얻을 수 있음
- 공격자의 시도
  - 개인키 D를 얻으려면 N을 인수분해하여 Φ(N)를 구해야 함
    - 큰 수에 대한 인수분해는 매우 많은 수행시간 요구
    - 인수분해 이외에 다른 방법이 있는지는 모름

RSA 알고리즘의 원리

- $M \equiv C^d \equiv M^{ed} \pmod{N}$ 입증 필요
- 오일러 정리
  - x와 N이 서로 소일 때 $x^{\Phi(N)} \equiv 1 \pmod{N}$
- $ED \equiv 1 \pmod{\Phi(N)}$
- $ED - 1 = k \cdot \Phi(N)$ for some integer k
- $M^{ED} = M^{k \cdot \Phi(N)+1} = M \cdot M^{k \cdot \Phi(N)} = M \cdot 1^k = M$

모듈러 지수 계산

- 곱셈연산의 횟수를 줄이는 방법
- 지수 표현: $6^{24} \approx 7.9 \times 10^{17}$
  - 하지만 $6^{24} \pmod{77} < 77$
  - 또한 $24 = 2^4+2^3$이므로
  - $6^{24} = 6^{16} \cdot 6^8$

$6^2 \equiv 6 \cdot 6 \equiv 36$	
$6^4 \equiv 6^2 \cdot 6^2 \equiv 36 \cdot 36 \equiv 64$	
$6^8 \equiv 6^4 \cdot 6^4 \equiv 64 \cdot 64 \equiv 15$	
$6^{16} \equiv 6^8 \cdot 6^8 \equiv 15 \cdot 15 \equiv 71$	$6^{24} \equiv 6^{16} \cdot 6^8 \equiv 15 \cdot 71 \equiv 64$

공개키 선택

- E = 3
  - 적은 회수의 곱셈, 암호화 시간 짧음
  - 세제곱근 공격
    - 평문 $M < N^{1/3}$이면 평문은 $C^{1/3}$로 구할 수 있음
    - 대처법 : 평문에 패딩을 추가하여 $M > N^{1/3}$가 되게 함
  - 중국인의 나머지 정리 공격
    - 동일한 공개키를 가진 3명 이상의 사용자에게 동일한 문장 M을 암호화하는 경우 공격 가능
    - 대처법 : 평문에 난수를 추가하여 암호문이 달라지게 함
- $E = 2^{16}+1$
  - 곱셈 회수가 많음
  - 공격에 비교적 강함

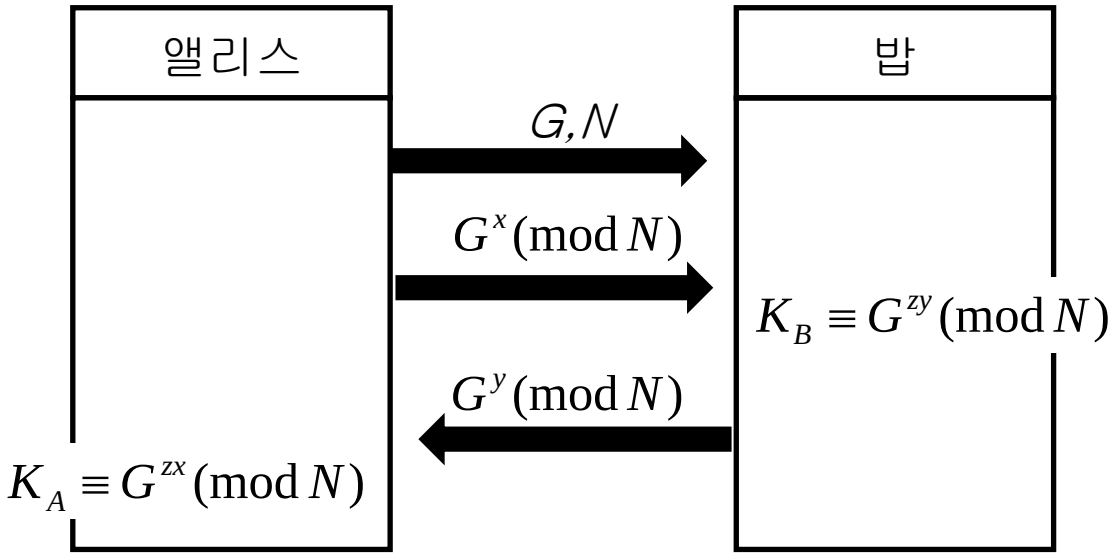
Diffie-Hellman Key Exchange

- 1970년 초에 영국 GCHQ의 Malcolm Williamson이 개발
- 1976년 Whitfield Diffie와 Martin Hellman이 공개
- 상호간에 비밀키를 교환하는 알고리즘
- 모듈러 지수 연산 이용
- 공격자는 이산 로그 (discrete logarithm) 문제를 풀어야 함
  - 이산로그 문제 : 양의 정수 N과 G가 알려져 있고, B를 얻었을 때, 다음 식을 만족하는 미지수 x를 구하는 문제

$$B \equiv G^x \pmod{N}$$

  - 다항시간복잡도를 가지는 알고리즘은 알려져 있지 않음

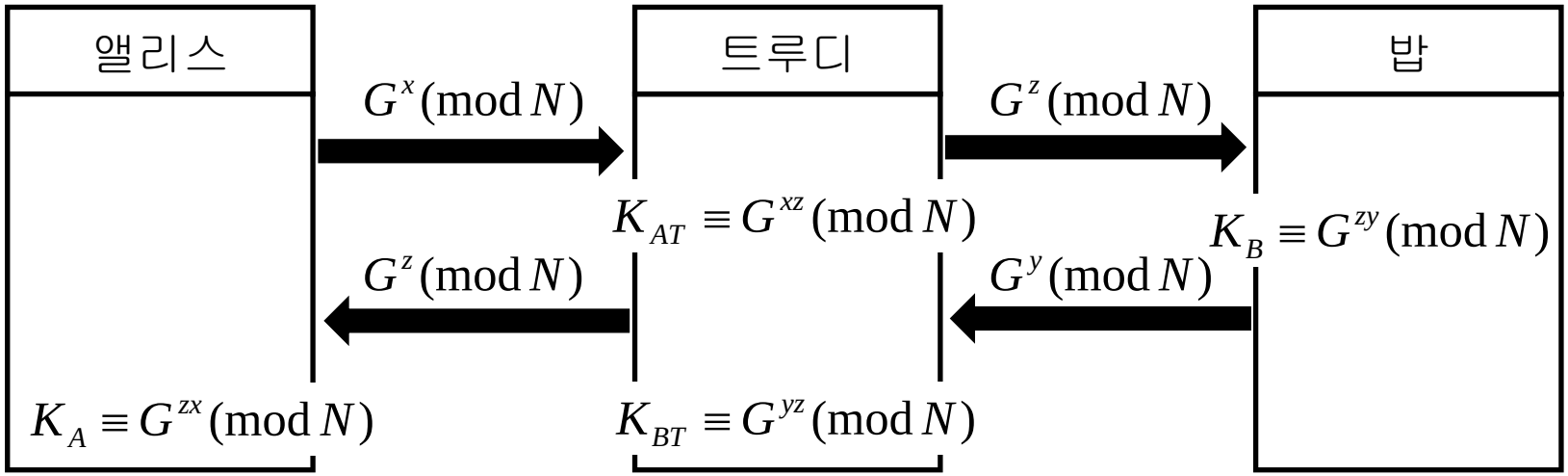
Diffie-Hellman Key Exchange



DH 알고리즘의 원리

- $K_A = B^x = (G^y)^x = G^{yx} = G^{xy} = (G^x)^y = A^y = K_B \mod N$
- 공격자는 G, N, A, B를 가지고 있음
  - x나 y를 알아야 하지만 이는 노출되지 않음
  - $A = B^x \mod N$으로부터 x를 알기가 어려움
- 안전성
  - 공격자가 수동적인 경우에는 쉽게 공격 당하지 않음
  - 공격자가 적극적인 경우에는 중간자 공격에 약함

DH에 대한 중간자 공격

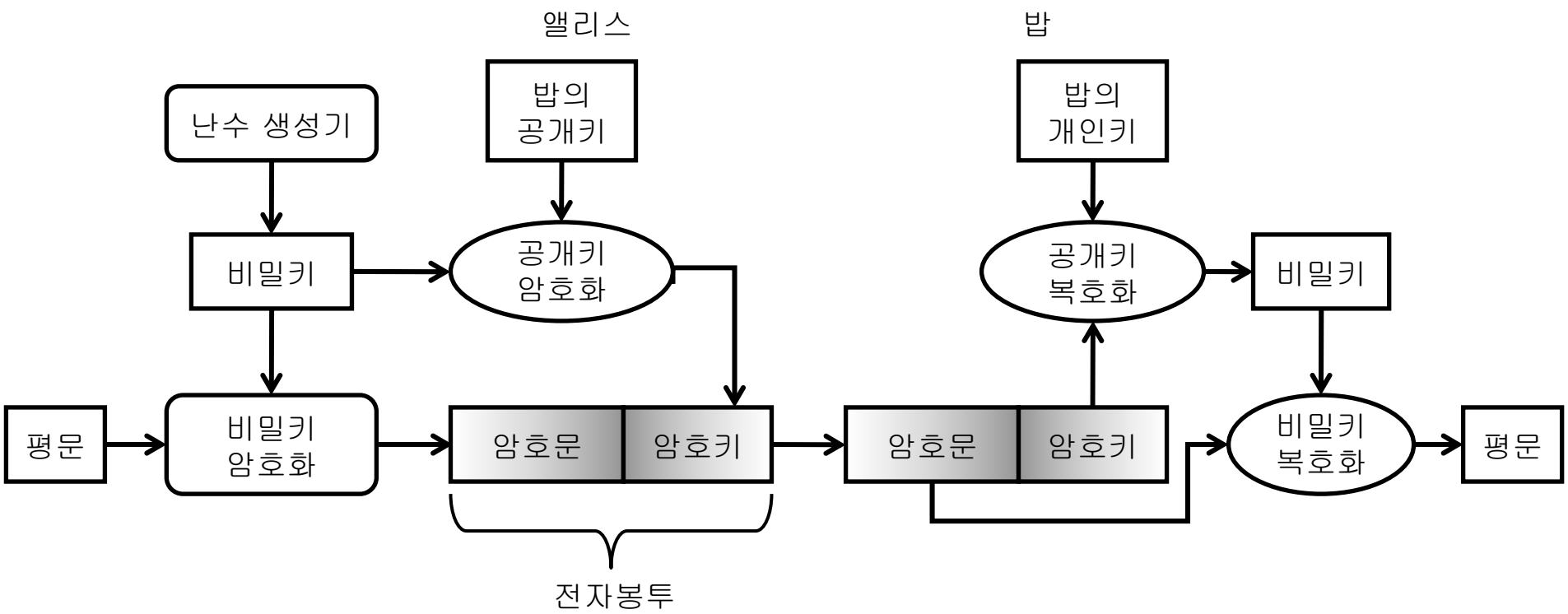

앨리스
트루디
밥
$G^x \pmod N$
$G^z \pmod N$
$K_{AT} \equiv G^{xz} \pmod N$
$K_B \equiv G^{zy} \pmod N$
$G^z \pmod N$
$G^y \pmod N$
$K_A \equiv G^{zx} \pmod N$
$K_{BT} \equiv G^{yz} \pmod N$

전자 봉투

- 동기
  - 공개키 암호 알고리즘은 공개키 전달이 용이하지만 복호화에 많은 시간이 걸림
  - 비밀키 암호 알고리즘은 암호화/복호화 시간이 공개키의 경우보다 짧지만 키 전달의 문제가 있음
- 전자봉투
  - 평문은 비밀키로 암호화하고 그 비밀키로 공개키로 암호화하여 첨부한 문서 형태
  - 전자봉투를 보내는 사람은 받는 사람의 공개키를 가지고 있어야 함.
  - 전자봉투의 형태 (평문 P, 비밀키 K)

    $E(P, K) \mid \{K\}_B$
  - 비밀키 K는 1번만 사용됨

전자봉투 진행 과정


앨리스
밥
난수 생성기
밥의
공개키
밥의
개인키
비밀키
공개키
암호화
공개키
복호화
비밀키
평문
비밀키
암호화
암호문
암호키
암호문
암호키
비밀키
복호화
평문
전자봉투

OpenSSL을 통한 공개키 암호화

- 공개키 암호화
- 개인키 복호화
- 전자봉투

RSA 공개키 암호화 함수

- 예제
  - csize = RSA_public_encrypt(psize, ptext, ctext, rsaPub, RSA_PKCS1_OAEP_PADDING);
- 문법
  - #include <openssl/rsa.h>
  - #include <openssl/engine.h>
  - int RSA_public_encrypt(int flen, unsigned char *from, unsigned char *to, RSA *rsa, int padding);
    - flen : 평문의 크기 (byte)
    - from : 평문이 있는 메모리 공간
    - to : 암호문이 저장될 공간
    - rsa : 공개키를 가리키는 포인터
    - padding : 평문 패딩 방법
    - Return value : 암호문 크기

패딩 옵션

패딩	최대 평문 크기 (byte)
RSA_PKCS1_PADDING	RSA_size(rsa) – 11
RSA_PKCS1_OAEP_PADDING	RSA_size(rsa) – 41
RSA_SSLV23_PADDING	RSA_size(rsa) – 11
RSA_NO_PADDING	RSA_size(rsa)

RSA 개인키 복호화 함수

- 예제
  - psize = RSA_private_decrypt(csize, ctext, dtext, rsaPriv, RSA_PKCS1_OAEP_PADDING);
- 문법
  - int RSA_private_decrypt(int flen, unsigned char *from, unsigned char *to, RSA *rsa, int padding);
    - Return value : 복호화된 평문 크기

RSA와 DES-CRC를 이용한 전자봉투

- C 프로그래밍 언어로 디자인한 전자봉투 형태 →
- 순서
  1. 비밀키 K 생성
  2. 비밀키 K를 A의 공개키로 암호화하여 저장
  3. 비밀키 K로 평문 m을 암호화하여 저장