RSA 암호화

10421 이민기 / 10405 김연준

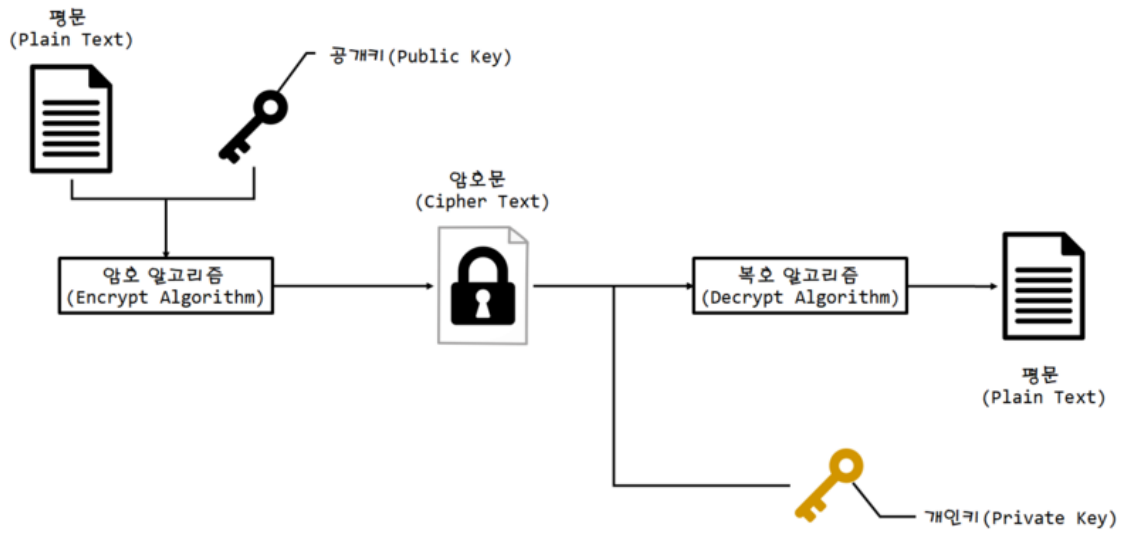
탐구 개요:

RSA 암호란 무엇인가:

현재 RSA 암호화 방식은 SSL/TLS 에서 가장 많이 사용되는 공개키 암호화 알고리즘 이다.

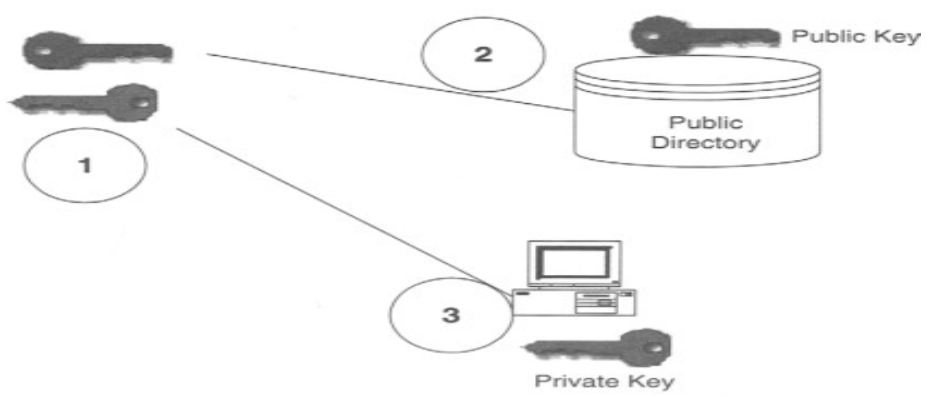
1978년 로널드 라이베스트(Ron Rivest), 아디 샤미르(Adi Shamir), 레너드 애들먼(Leonard Adleman)의 연구에 의해 체계화되었으며, RSA라는 이름은 이들 3명의 이름 앞글자를 딴 것이다.

RSA 암호화 과정:



A가 B에게 정보를 안전하게 보내기 위해 RSA 암호화 알고리즘을 사용한다고 가정해 보자. 우선 B가 공개키와 개인키를 만들어 A에게 공개키를 보낸다. 여기서 개인키는 B가지고 있는다. (개인키는 나중에 복호화 과정에서 사용) A가 B로 부터 받은 공개키를 사용해 보낸 정보를 암호화 한다. A가 B에게 암호화된 정보를 보낸다. 그런 다음 B는 암호화된 정보를 받고 개인키를 이용하여 암호를 해독한다.

수학적 접근:
RSA 암호화의 첫 단계는 공개키와 개인키를 만드는 것이다.



공개키는 n,e (정수) 개인키는 n,d (정수) 로 이루어져 있다.

n은 임의의 두 소수 p,q의 곱이다. n = p x q
두 소수를 곱하는 계산은 누구나 쉽게 할수 있을 것 이다. 하지만 곱해진 수를 소인수분해하여 p 와 q 를 알아낸 것은 아주 어려운 일이다. 또한 RSA 알고리즘에는 100자리 이상의 소수가 사용된다. 이러한 큰 소수 2개의 곱을 소인수 분해하는 데는 천문학적인 시간이 걸린다.
ex) 1341783164471 = p x q --> p,q는 ????
--> 이러한 특성 때문에 RSA 암호화 과정의 보안성이 높아진다.

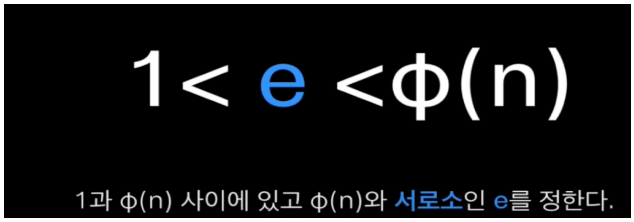
p,q 소수를 지정하는 방법: 큰 홀수를 지정한 후 2씩 더해가면서 그 수가 소수인지 판별한다. (소수의 희소성 활용) 여기서 소수인지 판별할때 페르마 소정리를 사용한다.

페르마 소정리: p가 소수이면 p를 약수로 가지지 않는 정수 a (즉 p 와 서로수) 에 대하여 a의 p-1승을 p로 나눈 나머지는 항상 1이다.

$$a^{p-1} \equiv 1 \pmod{p} \qquad (a \neq 0)$$

그런 다음

N의 오일러 피 함수 $\phi(N) = (p-1)(q-1)$



1 보다 크고 오일러 피 함수 값보다 작고 오일러 피 함수 값과 서로소인 e를 지정한다.

e 와 d 를 곱한 수를 오일러 피 함수 값으로 나눴을때 나머지가 1인 d (개인키) 값을 구한다.

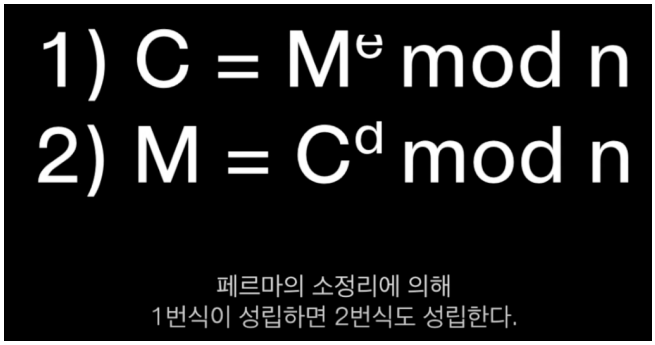
이제 구한 공개키를 사용해 정보를 암호화 해야 한다.

원래 정보를 M이라 하고 암호화 된 정보를 C 라고 한다.

이때 송신측은 수신측으로 부터 받은 공개키(n,e)가 있어야 정보를 암호화 할 수 있다.

마지막으로 수신측은 송신측 으로부터 받은 암호 데이터 C를 복호화 해야 할 차례이다.

복호화란 수신된 부호화 메시지로부터 원시 데이터를 복구하는 과정 이다.



페르마의 소정리에 희해 1번식 과 2 번식은 동시에 성립하게 된다. (증명 생략)
여기서 1번은 암호화 할때 사용했음으로 2 번식을 사용해 복호화 한다.

여기서 수신측 은 d 와 n 의 값을 알아야 한다.

따라서 개인키 d와 n을 가지고 있는 사람 즉 수신측만 암호를 풀 수 있다. --> 최종적 암호화 원리 및 결과

c언어로 구현한 RSA 암호화 알고리즘

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

// 소수 판별 함수
int is_prime(int n) {
    if (n <= 1) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0 || n % 3 == 0) return 0;
    for (int i = 5; i * i <= n; i += 6) {
        if (n % i == 0 || n % (i + 2) == 0)
            return 0;
    }
    return 1;
}
// 최대 공약수(GCD) 계산 함수
int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}
// 모듈러 역원 계산 함수
int mod_inverse(int a, int m) {
    a = a % m;
    for (int x = 1; x < m; x++) {
        if ((a * x) % m == 1)
            return x;
    }
    return 1;
}
int main() {
    // RSA 파라미터 설정
    int p, q, n, phi, e, d;
    p = 61; // 소수 1
```

```
    q = 53; // 소수 2
    if (!is_prime(p) || !is_prime(q)) {
        printf("p와 q는 모두 소수여야 합니다.\n");
        return 1;
    }
    n = p * q; // 모듈러 n 계산
    phi = (p - 1) * (q - 1); // 오일러 피 함수(phi) 계산
    // 공개 키 (e, n) 생성
    e = 17; // 일반적으로 1보다 크고 phi와 서로소인 값을 선택합니다.
    // 개인 키 (d, n) 생성
    d = mod_inverse(e, phi);
    printf("공개 키 (e, n): (%d, %d)\n", e, n);
    printf("개인 키 (d, n): (%d, %d)\n", d, n);
    // 평문과 암호문 설정
    int plaintext = 42;
    int ciphertext = (int)pow(plaintext, e) % n;
    printf("평문: %d\n", plaintext);
    printf("암호문: %d\n", ciphertext);
    // 복호화
    int decrypted = (int)pow(ciphertext, d) % n;
    printf("복호화된 메시지: %d\n", decrypted);
    return 0;
}
```