**RSA**

RSA(Rivest-Shamir-Adleman)은 안전한 데이터 전송을 위해 널리 쓰이는 공개키 암호이다. 이것은 가장 오래된 공개키 암호 중 하나이다. RSA라는 이름은 1977년 이 알고리즘을 공개적으로 설명한 Ron Rivest, Adi Shamir, Leonard Adleman의 성에서 유래했다. 공개키 시스템은 1973년 GCHQ(영국의 신호 정보 기관)에서 영국의 수학자 Clifford Cocks에 의해 비밀리에 개발되었다. 그 시스템은 1997년에 기밀 해제되었다.

공개키 암호 시스템에서, 공개되는 암호화키와 비밀로(개인적으로) 유지되는 복호화키로 구별된다. RSA사용자는 보조 값을 덧붙여 두 개의 큰 소수에 기초한 공개키를 만들고 공개한다. 그 소수들은 비밀로 유지된다. 메시지는 공개키를 통해 누구나 암호화할 수 있지만, 소수를 아는 사람만이 해독할 수 있다.

RSA의 보안은 factoring problem이라고 하는, 두 개의 큰 소수를 곱한 값을 인수분해를 하는게 현실적으로 얼마나 어려운지에 달려있다. RSA 암호화가 깨지는 것을 RSA Problem이라고 한다. 그것이 factoring problem만큼 어려운지에 대해서는 해결되지 않았다. 충분히 큰 키를 사용한 시스템을 해독하는 방법은 공식적으로는 알려진 것이 없다.

RSA는 상대적으로 느린 알고리즘이다. 이 때문에, 사용자 데이터를 바로 암호화 하는데 일반적으로 사용되지 않는다. RSA는 대칭키 암호화를 위한 키를 전송하는데 사용되는 경우가 더 많다.

1. 역사

비대칭키 (공개키) 암호 시스템의 아이디어는 1976년에 이 개념을 발표한 Whitfield Diffie와 Martin Hellman이 고안한 것으로 여겨진다. 또한 그들은 디지털 서명을 소개하고 정수론을 적용하려고 시도했다. 그들의 공식은 소수로 모듈로한 숫자를 거듭제곱한 것으로 만들어진 공유되는 비밀키를 사용하였다. 그러나 당시 인수분해가 잘 연구되지 않았기 때문에 단방향 함수를 열린 문제로 남겨놓았다.

매사추세츠 공과대학교의 Ron Rivest, Adi Shamir, Leonard Adleman은 1년동안 역으로 돌리기 어려운 단방향 함수를 만들기 위해 여러 번 시도하였다. 컴퓨터 과학자인 Rivest와 Shamir는 많은 잠재적 함수를 제안했고, 수학자인 Adleman은 그것들의 취약점을 찾는 역할을 하였다. 그들은 “배낭 문제”와 순열 다항식을 포함한 많은 접근 방식을 시도했다. 한동안, 그들은 그 함수를 만드는 것은 모순된 것이므로 불가능하다고 생각했다. 1977년 4월, 그들은 유월절을 한 학생의 집에서 보냈고 자정 무렵 집으로 돌아가기 전에 마니스체비츠 와인을 많이 마셨다. 잠을 잘 수 없었던 Rivest는 수학 책을 들고 소파에 누워서 단방향 함수에 대해 생각하기 시작했다. 그는 그의 아이디어를 공식화하는데 밤을 샜고, 새벽까지 연구를 이어갔다. 이 알고리즘은 현재 논문을 쓴 세사람의 성을 딴 RSA로 알려져 있다.

영국 정보기관 정부통신본부(GCHQ)에서 일하는 영국 수학자 Clifford Cocks는 1973년 기관 내 문서에서 동일한 시스템을 기술했다. 그러나 당시 그것을 구연하는데 필요한 컴퓨터는 상대적으로 비쌌기 때문에 그것은 대부분 호기심에서 끝났고, 공개적으로 알려진 바로는 그것은 사용되지는 않았다. 하지만 그의 발견은 1997년까지 기밀처리하게 된다.

Kid-RSA(KRSA)는 1997년에 발표된 안전하지 않은 공개키 암호로, 교육 목적으로 설계되었다. 몇몇 사람들은 Kid-RSA를 학습하면 단순화된 DES와 유사한 RSA와 다른 공개키 암호에 대한 지식을 얻을 수 있다고 생각한다.

2. 특허

RSA 알고리즘을 기술하는 특허는 1983년 9월 20일 MIT에 부여되었다(U.S. Patent 4,405,829 "Cryptographic communications system and method". From DWPI's abstract of the patent)

3. 연산법

RSA 알고리즘은 key generation, key distribution, encryption, decryption(키 생성, 키 배포, 암호화, 복호화) 4가지 단계로 실행된다.

RSA의 기본 원리는 모든 정수 m(0 ≤ *m* < *n*)으로 모듈러 거듭제곱을 한 매우 큰 양의 정수 e, d, n을 찾는 것이 가능하고 e와 n, 또는 m을 아는 상태에서 d를 찾는 것이 매우 어렵다는 것에 기초한다.

여기서 ≡는 모듈러에서 합동이다는 것을 나타낸다. (이것은 너가 *(me)d* 를 n으로 나누든, m를 n으로 나누던간에 그것들의 나머지는 같다는 의미이다.)

추가로, 일부 연산의 경우 두 지수 순서의 변경이 가능하고, 이 관계는 을 포함한다.{\displaystyle (m^{d})^{e}\equiv m{\pmod {n}}.}

RSA는 공개키와 개인 키가 있다. 공개키는 누구나 알 수 있으며, 메시지를 암호화하는데 사용된다. 공개키로 암호화된 메시지는 개인키를 사용해야만 해독할 수 있다. 공개키는 정수 n과 e로, 개인키는 정수 d로 표시된다. (복호화 과정에서도 n이 사용되므로, n은 개인키의 일부로 간주될 수 있다.) m은 메시지(아래에서 설명하는 과정 이전에 준비됨)를 나타낸다.

3.1. Key generation(키 생성)

RSA 알고리즘의 키는 다음과 같은 방법으로 생성된다:

1. 두개의 큰 소수 p, q를 고른다.

-인수분해를 더 어렵게 하기 위해, p와 q는 무작위로 선택되어야 하고 크기는 비슷하지만 길이는 달라야 한다.

-p와 q는 비밀로 유지되어야 한다.

2. n=pq를 계산한다.

-n은 공개키와 개인키의 모듈러로 사용된다. 이것의 길이는 보통 비트로 표현되는 키의 길이다.

-n은 공개키의 일부로 공개된다.

3. *λ*(*n*)을 계산한다. (*λ*은 Carmichael's totient function) n=pq이므로, *λ*(*n*) = [lcm](https://en.wikipedia.org/wiki/Least_common_multiple)(*λ*(*p*), *λ*(*q*))이고, p와 q는 소수이므로, *λ*(*p*) = [*φ*](https://en.wikipedia.org/wiki/Euler_totient_function)(*p*) = *p* − 1이고 마찬가지로 *λ*(*q*) = *q* − 1이다. 따라서 *λ*(*n*) = lcm(*p* − 1, *q* − 1)이다.

-이므로, lcm은 유클리드 알고리즘에 의해 계산될 수 있다.

- *λ*(*n*)은 비밀로 유지된다.

4. 1<e<*λ*(*n*)이고 gcd(e, *λ*(*n*))=1인 즉 *λ*(*n*)와 서로소인 정수 e를 선택한다.

-짧은 비트 길이와 작은 해밍 무게(0이 아닌 비트의 수)를 가진 e는 더 효율적인 암호화가 가능하다. (가장 일반적으로 선택되는 e의 값은 ) e가 될 수 있는 작은(그리고 빠른)값은 3이지만, 이러한 작은 e값은 덜 안전한 것으로 나타난다.

-e는 공개키의 일부로 공개된다.

5. 인 즉, *λ*(*n*)를 모듈러한 e의 모듈러 역원인 d를 찾는다.

-이것은 을 만족하는 d를 구하라는 것을 의미한다. e와 *λ*(*n*)이 서로소인 덕분에 위의 항등식이 d가 계수 중에 하나인 베주 항등식(두 정수의 최대공약수를 원래 두 수의 배수의 합으로 나타낼 수 있다는 정리)의 형식이기 때문에 확장 유클리드 알고리즘을 사용하여 d는 효율적으로 계산될 수 있다.

-d는 개인키가 되어 비밀로 유지한다.

공개키는 계수 n과 공개된(혹은 암호화하는) 지수 e로 구성된다. 개인키는 비밀로 유지되는(혹은 복호화하는) 지수 d로 구성된다. p, q, *λ*(*n*)은 이것들을 사용하여 d를 구할 수 있으므로 비밀로 유지해야 한다. 사실, 그것들은 모두 d가 계산된 후에 파기할 수 있다.

원래 RSA 논문에서, 오일러 피 함수 *φ*(*n*) = (*p* − 1)(*q* − 1)는 *λ*(*n*)를 대신하여 개인키 지수 d를 계산하는데 사용된다. *φ*(*n*)은 언제나 *λ*(*n*)으로 항상 나누어지기 때문에, 알고리즘은 잘 작동한다. 또한 오일러 피 함수를 사용할 수 있다는 것은 pq를 모듈러한 정수의 곱셈군에 적용한 라그랑주 정리에서 유래한다. 따라서 *d*⋅*e* ≡ 1 (mod *φ*(*n*))을 만족하는 d는 *d*⋅*e* ≡ 1 (mod *λ*(*n*))또한 만족한다. 그러나 d modulo *φ*(*n*)을 계산하면 때때로 필요 이상의 결과를 얻을 수 있다. (즉, d> *λ*(*n*)) 대부분의 RSA구현은 두 가지 방법을 사용하여 생성된 지수를 받지만(아래 설명한 중국인의 나머지 정리에 기반한 복호화 방법을 사용하는 대신 개인키 지수 d를 사용하는 경우), FIPS 186-4와 같은 일부 표준에서는 d<*λ*(*n*)이라는 조건이 필요할 수도 있다. 이 기준을 충족하지 않는 “초과되는” 개인키 지수는 항상 작은 수의 같은 지수를 얻기 위해 *λ*(*n*)으로 모듈러하여 줄어들 수 있다.

n-1=pq-1=(p-1)(q-1) + (p-1) + (q-1)의 인수분해에서 (p-1)과 (q-1)의 공통인수는 존재하므로, (p-1)과 (q-1)은 만약 있더라도 2를 제외한 가장 작은 공통인수만 가지는 것이 좋다.

참고: RSA논문의 원작성자는 d를 선택하고 d modulo *φ*(*n*)의 모듈러 역원으로 e를 계산하여 키 생성을 수행하지만, PKCS#1을 따르는 것과 같은 현재 대부분의 구현은 그 반대를 수행한다.(e를 선택하고 d를 계산한다.) 선택한 키는 작을 수 있지만 계산된 키는 일반적으로 그렇지 않기 때문에, RSA 논문의 알고리즘은 암호화에 비해 복호화를 최적화 하였고, 현대의 알고리즘은 복호화를 최적화한다.

3.2. Key distribution

밥이 엘리스에게 정보를 보내고 싶어한다고 가정해보자. 만약 RSA를 사용하기로 결정했다면, 밥은 메시지를 암호화 하기 위해 엘리스의 공개키를 알아야하며, 엘리스는 그 메시지를 해독하기 위해 그녀의 개인키를 사용해야한다.

밥이 암호화된 메시지를 보낼 수 있도록 엘리스는 신뢰할 수 있지만 비밀은 아닌 경로를 통해 자산의 공개 키 (n, e)를 밥에게 전송한다. 엘리스의 개인키 (d)는 절대 배포되지 않는다.

3.3. Encryption

밥이 엘리스의 공개키를 얻은 후, 그는 메시지 M을 엘리스에게 전달할 수 있다.

이를 수행하기 위해, 그는 먼저 M(엄밀히 말하면, 패딩되지 않은 평문)을 정수 m(엄밀히 말하면, 패딩된 평문)으로 변환한다. (padding scheme으로 알려진 가역 프로토콜을 사용한 m(0 ≤ *m* < *n*)) 그런 다음 그는 앨리스의 공개 키 e를 사용하여 암호문 c를 계산한다.

암호문 c는 다음과 같은 과정을 따른다:

이것은 모듈러 지수화를 사용하여 아무리 큰 수라고 해도 빠르게 수행될 수 있다. 밥은 c를 앨리스에게 보낸다. m의 최소 9개의 값은 m과 같은 암호문 c를 생성하지만, 실제로 이런 일은 잘 일어나지 않는다.

3.4. Decryption

앨리스는 다음과 같은 계산으로 그녀의 개인키 지수 d를 사용하여 c를 m으로 만들 수 있다.

M이 주어진다면 그녀는 padding scheme을 역으로 하여 원래 메시지 M을 찾을 수 있다.

4. 정확성 증명

4.1. 페르마의 소정리를 이용한 증명

RSA의 정확성을 증명하는 것은 (a: 모든 정수, p: a를 나누지 않는 소수)을 시작으로 한 페르마의 소정리에 기초한다.

우리는 를 보이고 싶어한다.

(p, q: 서로 다른 두 소수, e, d: 를 만족하는 양의 정수, m: 모든 정수)

*λ*(*pq*) = lcm(*p* − 1, *q* − 1)은 (p-1)과 (q-1)로 나누어지기 때문에, 식 ed-1=h(p-1)=k(q-1)을 만들 수 있다. (h, k: 음이 아닌 정수)

두 숫자 와 m이 mod pq에서 같은지 확인하기 위해, 그것들이 mod p와 mod q에서 각각 같 같은 확인하면 충분하다.

를 보이기 위해, 우리는 두가지 경우를 고려한다:

1. 만약 라면, m은 p의 곱이다. 따라서 는 p의 곱이다. 그러므로 이다.

2. 만약 이라면, (를 1로 대신하는 페르마의 소 정리를 이용)

도 완전히 같은 방법으로 진행한다:

1. 만약 라면, m은 q의 곱이다. 따라서 는 q의 곱이다. 그러므로 이다.

2. 만약 이라면, (를 1로 대신하는 페르마의 소 정리를 이용)

이것은 을 완벽히 증명한다.

(p, q: 서로 다른 두 소수, e, d: 를 만족하는 양의 정수, m: 모든 정수)

4.2. 오일러 정리를 이용한 증명

Rivest, Shamir, and Adleman의 원래 논문에서는 RSA의 구현 원리를 설명하는데 페르마의 소정리를 사용하였지만, 일반적으로는 오일러의 정리를 이용하여 증명한다.

우리는 을 보이고 싶어한다. (n=pq, p, q: 서로 다른 두 소수, e, d: 를 만족하는 양의 정수, m: 모든 정수) e와 d가 양수이기 때문에, 우리는 식 ed=1+h*φ*(*n*)을 만들 수 있다. (h: 음이 아닌 정수) m이 n과 서로소라고 가정하면, 우리는 식 을 만들 수 있다. (마지막 2개의 합동 식은 오일러 정리에서 나옴)

보다 일반적으로, 를 만족하는 모든 e와 d에서, 같은 결론은 오일러 정리의 카마이클 함수(n과 서로소인 모든 m에 대하여 인 상태)로 나온다.

m이 n과 서로소가 아닐 때, 아까 주어진 식은 유효하지 않다. 이것은 매우 가능성이 낮지만, 이 경우에도 원하는 합동식은 여전히 참이다. *m* ≡ 0 (mod *p*) 또는 *m* ≡ 0 (mod *q*)이고, 이러한 경우는 이전 증명을 사용하여 풀 수 있다.

5. 패딩

5.1. 일반 RSA에 대한 공격

아래 설명된 바와 같이 RSA에 대한 여러가지 공격이 있다.

-낮은 암호화 지수(예: e=3)와 작은 값의 m(예: m<)으로 암호화하는 경우, 의 결과는 절대적으로 계수 n 보다 작다. 이 경우 암호문은 정수 위의 암호문의 e번째 루트를 가져옴으로써 쉽게 해독될 수 있다.

-만약 동일한 텍스트 메시지가 암호화된 방식으로 e 이상의 전송자에게 전해지고, 수신자가 같은 지수 (p, q, n과 다른)e를 공유한다면, 중국인의 나머지 정리를 통해 텍스트 메시지를 해독하는 것이 쉬워진다. Johan Hastad는 이 공격이 클리어 텍스트들이 동일하지 않더라도 공격자는 그것들 사이의 선형관계를 알수 있다는 것을 알아차렸다. 이 공격은 후에 Don Coppersmith에 의해 개선되었다.

-RSA암호화는 결정론적 암호화 알고리즘(즉, 랜덤한 요소가 없다)이기 때문에 공격자는 공개키와 그것들이 암호문과 동일한지 테스트를 통해 나온 비슷한 평문을 암호화하는 암호시스템에 대해 선택평문공격을 시작할 수 있다. 공격자가 해당하는 평문을 알고 있더라도 공격자가 그것과 다른 것의 암호화를 서로 구별할 수 없는 경우 암호 시스템을 의미론적으로 안전하다고 한다. 패딩을 사용하지 않는 RSA는 의미론적으로 안전하지 않다.

-RSA는 두 암호문의 곱이 각 평문의 곱의 암호화와 같다는 속성을 가지고 있다. 이는 을 의미한다. 이 곱셈 속성으로 인해, 선택평문공격이 가능하다.

-개인키 지수 d가 주어지면, 계수 n=pq를 효율적으로 인수분해할 수 있다. 그리고 계수 n=pq의 인수분해가 주어진다면, 공개키(e’, n)에 대해 생성된 임의의 개인키 (d’, n)을 얻을 수 있다.

5.2. Padding schemes

이러한 문제를 방지하기 위해, 실제 RSA 구현에서는 일반적으로 m을 암호화하기 전에 어떤 형태의 구조화된 랜덤 패딩을 포함시킨다. 이 패딩은 m이 안전하지 않은 평문 범위에 속하지 않도록하며, 주어진 메시지가 한번 패딩되면 다수의 서로 다른 가능한 암호문 중 하나로 암호화되게 한다.

PKCS#1과 같은 표준은 RSA 암호화 전에 메시지를 안전하게 패딩되도록 설계되었다. 이러한 방식은 평문 m을 일부 추가 비트들로 패딩하기 때문에, 패딩되지 않은 메시지 M은 어느 정도 작아야한다. RSA 패딩 방식은 예측가능한 메시지 구조에 의해 나올 수 있는 정교한 공격을 방지할 수 있도록 설계되어야한다. PKCS#1 표준의 초기버전(1.5버전까지)은 RSA를 의미론적으로 안전하게 만드는 것으로 보이는 구조를 사용했다. 그러나 1998년 Crypto에서 Bleichenbacher는 이 버전이 실제 선택평문공격에 취약하다는 것을 보여주었다. 게다가 2000년 Eurocrypt에서, Coron 등은 일부 유형의 메시지에 대해 이 패딩이 충분한 수준의 보안을 제공하지 않는다는 것을 보여주었다. 최신 버전의 표준에는 이러한 공격을 방지하는 Optimal Asymmetric Encryption Padding (OAEP)가 포함되어 있다. 따라서 OAEP는 모든 새로운 애플리케이션에서 사용해야 하며, PKCS#1 1.5 패딩은 가능한 한 교체해야 한다. 또한 PKCS#1 표준은 RSA 서명에 대한 추가 보안(예: RSA-PSS)을 제공하도록 설계된 체계를 포함한다.

RSA-PSS와 같은 보안 패딩 체계는 메시지 암호화와 마찬가지로 메시지 서명 보안에 필수적이다. PSS에 대한 2개의 미국 특허(U.S. Patent 6,266,771 and U.S. Patent 7,036,014)가 부여되었지만, 이 특허는 각각 2009년 7월 24일과 2010년 4월 25일에 만료되었다. PSS의 사용은 더 이상 특허로 인해 사용 불가능 하지는 않는 것처럼 보인다. 암호화 및 서명에 서로 다른 키 쌍을 사용하는 것이 잠재적으로 더 안전하다.