인터넷에서 사용되는 여러 기술 : HTTPS 5

조인준
KBS 미디어기술연구소 차장

지난 편에서는 보안 통신을 위한 SSL/TLS의 핸드셰이킹에 대해 알아보았습니다. SSL/TLS 핸드셰이킹 과정에서 암호화 통신 시 사용할 대칭키 생성을 위한 키 교환(Key Exchang) 단계에서 사용하는 방식 중 가장 널리 쓰이는 방식으로 RSA 방식과 DH(Diffie-Hellman) 방식이 있습니다. 지금부터 두 편에 나누어 각각의 방식에 대해 설명해보겠습니다.

RSA 알고리즘은 1977년에 이를 개발한 Ron Rivest, Adi Shamir, Leonard Adleman 3인의 이름에서 Rivest, Shamir, Adleman의 앞자 R, S, A를 따서 이름을 붙인 것입니다. RSA 알고리즘은 현재 가장 널리 사용되어 온 비대칭키(사설키와 공개키의 쌍으로 이루어진 키) 암호화 방식이며 가환성(commutative)의 연산 성질을 갖고 있습니다. 가환성의 연산 성질이란 사설키로 암호화하면 공개키로 해독이 가능하고 공개키로 암호화하면 사설키로 해독이 가능하여 어떤 키를 먼저 사용하던 결과는 같음을 의미합니다. RSA가 동작하는 방식을 이해하기 위해서는 다음과 같은 중학시절에 배운 수학 관련 내용을 환기할 필요가 있습니다.

인수(Factor)

인수는 어떤 수를 곱하여 원래의 수를 만들어내는 수를 가리킵니다. 예를 들어, 6의 인수는 1, 2, 3, 6입니다. 이는 1×6 = 6, 2×3 = 6과 같이 원래의 수를 만들어내기 위해 사용될 수 있습니다.

소수(Prime Number)

소수는 정확히 두 개의 양의 인수를 가지는 양의 정수입니다. 즉, 1과 자기 자신만을 인수로 가집니다. 예를 들어 2, 3, 5, 7, 11과 같이 1과 자신 이외에는 다른 양의 정수로 나누어지지 않는 특징을 가지고 있습니다.

준소수(Semi-Prime Number)

준소수는 두 개의 소수를 곱하여 생성된 수를 가리킵니다. 즉, 두 소수의 곱으로 이루어진 수입니다. 예를 들어, 15는 3과 5의 곱으로 만들어진 준소수입니다. 다른 예로는 21(3×7), 35(5×7) 등이 있습니다. 준소수는 암호학에서 중요한 개념 중 하나입니다. RSA 알고리즘과 같은 공개키 암호화 시스템에서는 두 큰 소수의 곱으로 생성된 숫자가 공개키와 관련되어 있습니다. 이때 소수의 곱을 분해하는 것이 매우 어렵기 때문에 높은 보안성을 가질 수 있습니다.

나머지(Modulo)

모듈로 연산자는 주어진 숫자를 다른 숫자로 나눈 후 나머지를 반환하는 연산자입니다. 모듈로 연산은 [수식 1]과 같은 형태로 나타낼 수 있습니다. [수식 1]은 7을 3으로 나눈 나머지를 구한다고 가정한 상태에서 MOD로 표시한 Modulo 연산자를 사용하여 표현한 것입니다. 7을 3으로 나눈 나머지는 1이니 [수식 1]의 결과도 당연히 1입니다.

$$7 \text{ MOD } 3 = 1 \cdots\cdots \textbf{[수식 1]}$$

구분	기호	값
두 소수	X, Y	17, 23
두 소수의 곱	M	391
Totient	T = (X-1)(Y-1)	352
공개키	PUB	59
사설키	PRI	179

표 1. RSA 암호화를 위해 사용되는 수

RSA를 이용한 비대칭키 생성은 [표 1]의 숫자들을 이용하여 다음과 같은 절차를 통해 이루어집니다.

① 임의의 두 소수 X, Y를 정합니다.

우선 임의의 두 소수를 정합니다. C군은 예로 [표 1]과 같이 17과 23으로 두 소수를 정해보겠습니다.

② 두 소수 X, Y의 곱 M을 구합니다.

X와 Y가 각각 17과 23이므로 M = 17×23 = 391이며, 두 소수의 곱을 통하여 생성되었으므로 M은 준소수가 됩니다.

③ Totient 함수값 T를 구합니다.

[수식 2]를 이용하여 T 값을 계산합니다. Totient 함수는 RSA 암호화에서 중요한 역할을 합니다. RSA 방식을 실제 보안에 사용할 때는 [표 1]의 예로 든 17과 23 등의 작은 소수와는 달리 매우 큰 소수를 이용하여 공개키를 만드는데 이 과정에서 Totient 함수를 사용하여 키 생성에 사용된 소수를 쉽게 찾을 수 없게 합니다. Totient 함수의 값이 큰 경우, RSA 암호화의 보안성이 높아집니다.

$$T = (X-1)\times(Y-1) = 16\times22 = 352 \cdots\cdots \textbf{[수식 2]}$$

④ 아래의 조건을 만족하는 수를 공개키 PUB로 정합니다.

- 소수이어야 함
- 계산된 Totient 함수값 보다 작은 값이어야 함
- Totient 함수값의 인수여서는 안 됨

C군은 임의로 59를 선택했습니다. 이로써 소수이어야 하는 첫째 조건은 만족했고, 59는 Totient 함수값 352보다 작은 값이니 둘째 조건도 만족합니다. 마지막 조건인 Totient 함수값의 인수여서는 안 되는 조건은 352 = 59×5 + 57로부터 확인이 되므로 59를 공개키로 이용할 수 있습니다.

⑤ 아래의 조건을 만족하는 수를 사설키 PRI로 정합니다.

- 구하려는 사설키와 공개키의 곱은 Totient 함수값으로 나누었을 때 [수식 3]과 같이 나머지가 1이 되어야 함

$$(Pub \times Pri) \ MOD \ T = 1 \cdots\cdots \ \textbf{[수식 3]}$$

위 식을 만족하는 Pri 중에 하나의 값을 Pri로 선택할 수 있고 179, 531... 등의 값 중에서 C군은 [표 1]과 같이 179를 골랐습니다. 이렇게 공개키와 사설키가 정해지면 아래의 식을 통해 메시지의 암호화 및 암호화된 메시지의 해독이 가능해집니다(아래 식에서 PUB, PRI, M은 [표 1]의 값들을 나타냄).

☑ 메시지 암호화(Encryption) 식

$$(메시지)^{PUB} \ MOD \ M = 암호화 \ 메시지$$
또는
$$(메시지)^{PRI} \ MOD \ M = 암호화 \ 메시지$$

☑ 암호화된 메시지 해독(Decryption) 식

$$(암호화 \ 메시지)^{PUB} \ MOD \ M = 메시지$$
또는
$$(암호화 \ 메시지)^{PRI} \ MOD \ M = 메시지$$

암호화 식을 이용하여 HELLO라는 메시지의 ASCII 코드를 암호화해보겠습니다. HELLO는 [표 2]의 ASCII 코드로 72(H) 69(E) 76(L) 76(L) 79(O)입니다. 이를 [표 1]의 공개키를 이용해 암호화하면 오른쪽과 같이 각 문자의 ASCII 코드가 숫자로 매핑됩니다.

$H(72) : 72^{59} \ MOD \ 391 = 81$
$E(69) : 69^{59} \ MOD \ 391 = 69$
$L(76) : 76^{59} \ MOD \ 391 = 359$
$O(79) : 79^{59} \ MOD \ 391 = 97$

위의 계산을 통한 매핑을 통해 72(H) 69(E) 76(L) 76(L) 79(O)는 81 69 359 359 97로 암호화됩니다. 재미난 우연이지만 C군이 임의로 만든 비대칭키로 암호화된 E는 암호화 전과 암호화 이후가 같습니다.
이제 사설키 179를 이용해 암호화된 메시지 81 69 359 359 97을 해독하면 오른쪽과 같습니다.

$81 : 81^{179} \ MOD \ 391 = 72 \ (H)$
$69 : 69^{179} \ MOD \ 391 = 69 \ (E)$
$359 : 359^{179} \ MOD \ 391 = 76 \ (L)$
$97 : 97^{179} \ MOD \ 391 = 79 \ (O)$

Decimal	ASCII	Decimal	ASCII	Decimal	ASCII	Decimal	ASCII
65	A	72	H	79	O	86	V
66	B	73	I	80	P	87	W
67	C	74	J	81	Q	88	X
68	D	75	K	82	R	89	Y
69	E	76	L	83	S	90	Z
70	F	77	M	84	T		
71	G	78	N	85	U		

표 2. ASCII 코드의 문자와 10진수 매칭

이로써 공개키로 암호화된 '72(H) 69(E) 76(L) 76(L) 79(O) → 81 69 359 359 97'를 사설키로 해독하면 '81 69 359 359 97 → 72(H) 69(E) 76(L) 76(L) 79(O)'가 되는 것을 확인할 수 있습니다. RSA 알고리즘에 의한 비대칭키가 가환성(commutative)의 연산 특성을 갖는 것을 확인하기 위해 사설키로 암호화한 후 공개키로 해독해보겠습니다. 사설키 179로 HELLO라는 메시지의 ASCII 코드를 암호화하면 아래와 같이 '72(H) 69(E) 76(L) 76(L) 79(O) → 234 69 274 274 379'가 됩니다.

$$H(72) : 72^{179} \text{ MOD } 391 = 234$$
$$E(69) : 69^{179} \text{ MOD } 391 = 69$$
$$L(76) : 76^{179} \text{ MOD } 391 = 274$$
$$O(79) : 79^{179} \text{ MOD } 391 = 379$$

이를 다시 공개키 59로 해독하면 아래와 같이 '234 69 274 274 379 → 72(H) 69(E) 76(L) 76(L) 79(O)'로 정확히 해독이 됩니다.

$$234 : 234^{59} \text{ MOD } 391 = 72 \text{ (H)}$$
$$69 : 69^{59} \text{ MOD } 391 = 69 \text{ (E)}$$
$$274 : 274^{59} \text{ MOD } 391 = 76 \text{ (L)}$$
$$379 : 379^{59} \text{ MOD } 391 = 79 \text{ (O)}$$

암호화된 코드는 0 ~ M-1 사이의 값을 가지므로 운 좋게 M값을 알아냈다고 가정해보겠습니다. 만약 [표 1]에서 사용한 391을 누군가 중간에서 알아내어 공개키와 사설키 쌍을 계산하려 한다면 우선 391의 인수인 17과 23을 알아내야 합니다. 아마 391을 알아낸다면 금방 맞추지는 못해도 17과 23을 계산할 수는 있을 겁니다. 하지만 M 값이 충분히 크다면 어떨까요? 1991년 3월 18일에 RSA 연구소에서 RSA Factoring Challenge를 열어 십진수로 100자리부터 1234자리 사이, 이진수로 330비트~4,096비트 사이의 54개 준소수를 제시하고 각 준소수의 소수 인수쌍을 구하는데 상금을 걸었습니다. 2007년에 종료된 이 챌린지에서 도전자들은 12개의 준소수만 소수 인수를 구했고 2020년 기준으로 11개가 더 풀려서 54개 중 23개의 문제만 풀려 있습니다. 23개 모두 제시된 54개 중에서는 작은 수들이고 더 큰 수들에 대해서는 소수 인수를 구하지 못했습니다. 챌린지에서 인수를 구한 제일 큰 수는 829비트였습니다. 1,024비트는 2002년부터 권고표준이며, 2015년부터는 2,048비트가 권고표준입니다. 1,024비트의 M값을 쓴다면 RSA Factoring Challenge에서 알 수 있듯이 쉽게 암호를 깰 수 없겠죠?

지금까지 RSA 알고리즘을 알아보았습니다. 다음 편에서는 DH(Diffie-Hellman) 방식에 대해 알아보겠습니다.

P.S.

C군이 여러분께 전하는 내용 중 전문적 성격이 짙은 것은 엄밀한 언어를 사용하여 설명하기에는 한계가 있습니다.
본 내용은 설명하는 대상에 대한 전체적 맥락의 이해에만 이용하시고, 그 이상은 권위 있는 전문자료를 참고하시기 바랍니다.