Public Key Cryptography

Examples

Steven Gordon

1 RSA Conditions

If the RSA encryption algorithm is $C = M^e \bmod n$, and the decryption algorithm is $M = C^d \bmod n$, then that implies that:

$$\begin{aligned} M &= C^d \bmod n \\ &= (M^e \bmod n)^d \bmod n \\ &= (M^e)^d \bmod n \\ &= M^{ed} \bmod n \end{aligned}$$

So for the encryption algorithm and decryption algorithm to work, then e, d and n must be chosen such that: $M = M^{ed} \bmod n$

Lets look at Euler's theorem: $a^{\phi(n)} \equiv 1 (\bmod n)$ where a and n are relatively prime. Or an alternative form is: $a^{\phi(n)+1} \equiv a (\bmod n)$ and in this form a and n are NOT required to be relatively prime. Compare this to our condition above:

$M^{ed} \equiv M (\bmod n)$

This is true if $\phi(n)+1 = ed$. This implies: $ed (\bmod \phi(n)) \equiv 1$ or in other words, e and d are multiplicative inverses in modular arithmetic $\phi(n)$.

A number e has a multiplicative inverse in $\bmod \phi(n)$ if e and $\phi(n)$ are relatively prime. Therefore we choose a number e such that it is relatively prime to $\phi(n)$ (that is, $\gcd(e, \phi(n)) = 1$) and less than $\phi(n)$. Then d can be calculated (Euclids algorithm is an efficient way to determine d).

What about choosing n? As we will show in Section 3, it should be hard to calculate $\phi(n)$ if you are an attacker. But for the person generating the keys, it should be easy to calculate $\phi(n)$. So, if $n = pq$, where p and q are (very large) prime numbers, then:

$$\begin{aligned} \phi(n) &= \phi(pq) \\ &= \phi(p)\phi(q) \\ &= (p-1)(q-1) \end{aligned}$$

If you know p and q, then it is very easy to calculate $\phi(n)$. But if you don't know p and q, and only know n, then it is very hard to calculate $\phi(n)$. See Section 3.

2 RSA Encryption and Decryption

2.1 Key Generation

1. Select the prime numbers $p = 17$ and $q = 11$.
   a. $n = pq = 187$
   b. $\varnothing(n) = (p-1)(q-1) = 16 \times 10 = 160$
2. Select e such that it is relatively prime to 160 and less than 160. Lets choose $e = 7$.
3. Choose d such that $de \equiv 1 \pmod{160}$ and $d < 160$
   a. $d = 23$. Since $23 \times 7 = 161 = 10 \times 160 + 1$ (extended Euclid's algorithm can be used to calculate d)

Therefore:

Public Key = $(e,n) = (7,187)$
Private Key = $(d,n) = (23,187)$

(Note that the item that must be secret is d. Both e and n are publicly available).

2.2 Encryption

Remember, the plaintext and ciphertext in RSA are integers.

Now encrypt M = 88

$C = 88^7 \bmod 187$
$= ((88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)) \bmod 187$
$= (132 \times 77 \times 88) \bmod 187$
$= 894432 \bmod 187$
$= 11$

2.3 Decryption

$M = 11^{23} \bmod 187$
$= (11 \times 121 \times 55 \times 33 \times 33) \bmod 187$
$= 79720245 \bmod 187$
$= 88$

3 RSA Attack

The attacker knows: C, e, and n (as well as the algorithms). What do they need to do to find M or d?

To find M: the attacker knows: $C = M^e \bmod n$. The attacker knows three of the four variables in this equation, and therefore one could expect it is easy to calculate M. But (for reasonably large values of M and e), it is not because the inverse of the exponentiation (finding the eth root of M modulo n) is vary hard to calculate. In fact, currently there are no known methods of doing so that are faster than factoring n (to derive d – see below).

To find *d*: the attacker knows: *e*, *n* and $ed(\mathrm{mod}\,\phi(n)) \equiv 1$. Hence, again we have an equation with three variables, of which two are known. But again, its not easy to determine the third variable *d*! Two approaches:

1. Calculate $\phi(n)$ using an algorithm that counts the relatively prime numbers less than *n*. For a large *n* (and *n* is large - remember it is the result of multiplying two large numbers *p* and *q*), there are no known algorithms that can calculate $\phi(n)$ faster than the factoring problem in approach 2 below.
2. Factor *n* into its prime factors, *p* and *q*, and then simply calculate ∅(*n*) = (*p*-1)(*q*-1). This is considered the fastest technique for breaking RSA, and hence all measures of security of RSA are based on how long it takes to factor a large number into its prime factors. However, factoring a large number *n* into its primes is hard! For a value of *n* with 640 bits (slightly less than 200 decimal digits), in 2005 it took about 30 2.2GHz-Opteron-CPU years (about 5 months real time) to find the factors. Currently, most users of RSA use 1024 or 2048 bit keys (size of *n*). It is expected it could take 100 times longer for a 1024 bit key, and millions of times longer for a 2048 bit key.

Hence, breaking RSA through calculating M or d is not feasible. However, as with every security algorithm there are special cases that make attacks possible (for example, bad values of parameters can be chosen to make an attack practical). But if it is used correctly, RSA is secure.

4 Diffie Hellman Example

Choose a prime number q = 353.

Choose α a primitive root of q, 3.

α and q are known to both A and B (and anyone else, including attacker).

A selects $X_A = 97$	B selects $X_B = 233$
$Y_A = \alpha^{XA} \bmod q$	$Y_B = \alpha^{XB} \bmod q$
$= 3^{97} \bmod 353$	$= 3^{233} \bmod 353$
$= 40$	$= 248$

A and B exchange their keys, Y_A and Y_B. This can be done publicly (that is, the attacker can see the keys).

A calculates:	B calculates:
$Y_B{}^{XA} \bmod q$	$X_B{}^{YA} \bmod q$
$= 248^{97} \bmod 353$	$= 40^{233} \bmod 353$
$= 160$	$= 160$

The secret key is 160.

The attacker has the following information: α=3, q=353, Y_A = 40, Y_B = 248. Can they determine the secret key?

To find the secret, they need to solve:

$3^{XA} \bmod 353 = 40$ or $3^{XB} \bmod 353 = 248$

A brute force attack would apply every possible value of X_A (or X_B) to find the answer. To counter this attack, large values of X_A and X_B are used. For large values of X_A and X_B, calculating the inverse of the exponentiation (that is the discrete logarithm) is very difficult! Hence, hard to break.