The RSA Algorithm

The RSA (Rivest-Shamir-Adleman algorithm) is the most important public-key cryptosystem.

The RSA works because:

> If $n = pq$, where p and q are large primes (several hundred digits), then
>
> i) Given p and q, we can easily multiply them to obtain n, but
>
> ii) Given n, there is no known way to factor n as pq in any reasonable amount of time.

We also need these lemmas.

Lemma 1. If $n = p_1 p_2 \ldots p_h$ is a product of *distinct* primes, then

i) $\varphi(n) = (p_1-1)(p_2-1)\ldots(p_h-1)$, and

ii) $p_i - 1$ divides $\varphi(n)$ for all i.

Proof: We know in general that

$$\begin{aligned}\varphi(n) &= n\,(1-1/p_1)\,(1-1/p_2)\ldots(1-1/p_h) \\ &= p_1 p_2 \ldots p_h\,(1-1/p_1)\,(1-1/p_2)\ldots(1-1/p_h) \\ &= p_1(1-1/p_1)\; p_2(1-1/p_2)\;\ldots\; p_h(1-1/p_h) \\ &= (p_1-1)(p_2-1)\ldots(p_h-1).\end{aligned}$$

This proves (i), and (ii) follows immediately.

Lemma 2: If $n = p_1 p_2 \ldots p_h$ is a product of *distinct* primes, then

$$k \equiv 1 \pmod{\varphi(n)} \;\;\Rightarrow\;\; a^k \equiv a \pmod{n} \text{ for } \underline{\text{any}}\ a.$$

Proof: It suffices to show that, for any a,

$$a^k \equiv a \pmod{p_i} \quad \text{for } i = 1, 2, \ldots, h.$$

(If this holds, p_i divides $a^k - a$ for all i, so n must divide $a^k - a$, showing that $a^k \equiv a \pmod{n}$.

Consider each prime p_i separately.

i) If $a \equiv 0 \pmod{p_i}$, then $a^k \equiv 0 \equiv a \pmod{p_i}$.

ii) Otherwise Fermat's Little Theorem tells us that $a^{p_i-1} \equiv 1 \pmod{p_i}$. Since $p_i - 1$ divides $\varphi(n)$, $a^{\varphi(n)} \equiv 1 \pmod{p_i}$. So if $k \equiv 1 \pmod{\varphi(n)}$, $k = \varphi(n)\,t + 1$ for some integer t, and

$$a^k \equiv a^{\varphi(n)\,t+1} \equiv (a^{\varphi(n)})^t a \equiv a \pmod{p_i}.$$

Note: None of these results hold if the square of some prime divides n.

For example, if $n = 12 = 2^2 3$, then

$\varphi(12) = 4 \neq (2^2-1)(3-1)$.

$5 \equiv 1 \pmod{\varphi(12)}$, but $2^5 = 32 \not\equiv 2^1 = 2 \pmod{12}$.

The RSA works like this:

i) Alice chooses two large primes p_A and q_A.

ii) Alice computes $n_A = p_A q_A$ and $\varphi(n_A) = (p_A - 1)(q_A - 1)$

iii) Alice chooses an integer e_A with $\gcd(e_A, \varphi(n_A)) = 1$, possibly at random.

iv) Alice computes $d_A \equiv e_A^{-1} \pmod{\varphi(n_A)}$.

v) Alice's <u>public key</u> is $(\boldsymbol{n_A}, \boldsymbol{e_A})$. She distributes this.
Her <u>private key</u> is $\boldsymbol{d_A}$. She keeps this secret.
Alice can discard $\boldsymbol{p_A}$, $\boldsymbol{q_A}$, and $\boldsymbol{\varphi(n_A)}$.

vi) If $2^k \le n_A < 2^{k+1}$, Alice's encryption function for short messages (k bits or less, so $M < n_A$) is:

$$\boldsymbol{E_A(M) = M^{e_A} \pmod{n_A}}.$$

Anyone can compute $E_A(M)$. A longer message is encrypted by splitting it into k-bit blocks, and encrypting each block separately. Note that each encrypted block has k+1 bits.

vii) Alice's decryption function for short messages is:

$$\boldsymbol{D_A(M) = M^{d_A} \pmod{n_A}}, \text{ provided } 0 \le M < n_A.$$

No one except Alice (or someone else who has discovered Alice's private key) can compute this.

Note: $D_A(E_A(M)) \equiv (M^{e_A})^{d_A} \equiv M^{e_A d_A} \equiv M \pmod{n_A}$ since $e_A d_A \equiv 1 \pmod{\varphi(n_A)}$

Once Alice has done this, she can

1) <u>receive</u> encrypted messages from Bob (or anyone else), and
2) <u>send</u> digitally-signed messages to Bob (or anyone else).

In order for Alice to send encrypted messages to Bob, or to receive digitally-signed messages from Bob, Bob will need to choose his own public and private keys, $(\boldsymbol{n_B}, \boldsymbol{e_B})$ and $\boldsymbol{d_B}$.

Bob sends a short message M (at most k bits) to Alice like this:

i) Bob encrypts M as $M^{e_A} \pmod{n_A}$, and sends M^{e_A} to Alice. (Note Bob knows e_A and n_A.)

ii) Alice decrypts M^{e_A} as $(M^{e_A})^{d_A} \equiv M \pmod{n_A}$. Thus Alice recovers M.
(Note Alice actually recovers the value of $M \pmod{n_A}$, but this equals M as long as $M < n_A$.)

For longer messages, Bob could break the message up into k-bit blocks, and encrypt each block separately. Alice would break the encrypted message in k+1 bit blocks, and decrypt each block separately.

Example:

i) Alice chooses: $p_A = 59$, $q_A = 71$.

ii) Alice computes: $n_A = 59 \cdot 71 = 4189$,
$\varphi(n_A) = (59-1)\cdot(71-1) = 4060$.

iii) Alice chooses: $e_A = 671$.

iv) Alice computes: $d_A \equiv e_A^{-1} \pmod{4060} \equiv 1791$.

She may do this using Euclid's extended algorithm, which uses only $O(\log(n_A))$ steps, so is feasible even if n_A has hundreds of digits.

i	*q*[*i*]	*r*[*i*]	*x*[*i*]	*y*[*i*]
-1		4060	1	0
0		671	0	1
1	6	34	1	-6
2	19	25	-19	115
3	1	9	20	-121
4	2	7	-59	357
5	1	2	79	-478
6	3	**1**	**-296**	**1791**
7	2	0		

v) Alice distributes her public key **(4189, 671)** and keeps her private key **1791** secret.

vi) Alice's encryption function is: $\boldsymbol{E_A(M) \equiv M^{671} \pmod{4189}}$, provided $0 \le M < 2^{12}-1 = 4095$.

Alice's decryption function is: $\boldsymbol{D_A(M) \equiv M^{1791} \pmod{4189}}$, provided $0 \le M < 4095$.

Both functions can be computed using at most $2\log_2(n_A)$ modular multiplications, using fast exponentiation.

Bob sends Alice the message "RSA" as follows:

RSA = $\boxed{01010010\ 01010011\ 01000001}$ in ASCII.

Bob breaks this up into two 12-bit integers:

$\boxed{01010010\ 0101}\ \boxed{0011\ 01000001}$, or 1317, 833

He computes $1317^{671} \equiv 3530$, $833^{671} \equiv 3050 \pmod{4189}$.

The ciphertext is 3530, 3050, or

$\boxed{0110111001010}\ \boxed{0101111101010}$.

(Note that 13-bit blocks were used, as $M^{671} \pmod{4189}$ could be greater than 4095.)

Alice decrypts the message by computing

$3530^{1791} \equiv 1317$, $3050^{1791} \equiv 833 \pmod{4189}$

giving plaintext 1317, 833, or $\boxed{01010010\ 0101}\ \boxed{0011}$ $\boxed{01000001}$, or "RSA".