RSA (ECW?) – Ellis Cocks Williamson

See http://www.livinginternet.com/i/is_crypt_pkc_inv.htm#gchq

Reading:

Kaufman, C., Perlman, R. and Speciner, M. (2002)
Network Security. Private Communication in a Public World. 2nd ed. Upper Saddle River, NJ, Prentice Hall. Chapters 6 & 7.

Stallings, W. (2000) Network Security Essentials. New Jersey, Prentice-Hall. Chapter 3.

Menezes, A., van Oorschot, P., and Vanstone, S. (1996) Handbook of Applied Cryptography. CRC Press. Chapters 8 & 14. See http://www.cacr.math.uwaterloo.ca/hac/

Ellis, J.H. (1987) The history of Non-Secret Encryption. See http://www.jya.com/ellisdoc.htm

Crypto FAQ available from http://www.rsa.com/rsalabs/node.asp?id=2152

Prime Numbers and Modular Arithmetic

The public-key encryption algorithm by Rivest, Adelman and Shamir [1], uses prime numbers and modular arithmetic.

Natural Numbers

The term natural numbers commonly refers to the set $\mathbb{N} = \{1, 2, 3, 4, 5, \ldots\}$
The term integers refers to the set $\mathbb{Z} = \{0, 1, -1, 2, -2, 3, -3, \ldots\}$

Divisors

$x \neq 0$ divides y if $y = mx$, for some m and m, x and y are integers. i.e. x divides y if there is no remainder on division.
The notation $x|y$ is used to represent 'x divides y'. If $x|y$ then x is a divisor of y.
e.g. the positive divisors of 18 are 1, 2, 3, 6, 9 and 18.
Some relations involving divisors:

- If $x|1$, then $x = \pm 1$
- If $x|y$ and $y|x$, then $x = \pm y$
- Any $x \neq 0$ divides 0
- If $x|y$ and $x|z$, then $x|(my + nz)$ for arbitrary integers m and n

[1] Rivest, R.L., Shamir, A. and Adleman L. (1978) A method of obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2):120-126

Prime Numbers

A prime number is a natural number other than 1 divisible only by itself and 1. So the first few primes (prime numbers) are 2, 3, 5, 7, 11, 13, ...

Any natural number, x, (integer > 0) can be factored in a unique way as

$$x = p_1^{\alpha_1} p_2^{\alpha_2} p_3^{\alpha_3} \ldots p_t^{\alpha_t}$$

where $p_1 < p_2 < p_3 \ldots p_t$ are prime numbers and each $\alpha_i > 0$.

e.g. $85 = 5 \times 17$; $539 = 7^2 \times 11$.

If P is the set of all prime numbers then any positive integer can be uniquely expressed as:

$$x = \prod_{\mathrm{P}} p^{x_p}$$

For any particular x most x_p will be zero. The value of a positive integer can be expressed by listing the values of the non-zero exponents of its prime factorisation. e.g. 18 is represented by $\{x_2 = 1,\ x_3 = 2\}$

The product of two numbers is equivalent to adding the corresponding exponents:

$$c = ab \quad \rightarrow \quad c_p = a_p + b_p \qquad \text{for all } p$$

e.g. $a = 12$ and $b = 45$. a is represented by $\{a_2 = 2,\ a_3 = 1\}$ and b is represented by $\{b_3 = 2,\ b_5 = 1\}$.

So c is represented by $\{c_2 = 2,\ c_3 = 3,\ c_5 = 1\}$. So $c = 2^2 \times 3^3 \times 5^1 = 4 \times 27 \times 5 = 540$.

Any integer of the form p^k can only be divided by an integer that is a lesser or equal power of the same prime number, p^j with $j \leq k$. So if $x = \prod_{\mathrm{P}} p^{x_p}$, $y = \prod_{\mathrm{P}} p^{y_p}$ and $x|y$ then $x_p \leq y_p$ for all p

e.g. $x = 2^2.5 = 20$ and $y = 2^2.3.5^2 = 300$

then $x|y$ and $x_2 = y_2$, $x_3 < y_3$ and $x_5 < y_5$

Relative Prime Numbers

The positive integer c is said to be the **greatest common divisor** of a and b if c is a divisor of a and b and any divisor of a and b is a divisor of c.

This is equivalent to

$$\gcd(a,\, b) = \max[k, \text{ such that } k|a \text{ and } k|b\,]$$

where $\gcd(a,\, b)$ means the greatest common divisor of a and b.

$$k = \gcd(a,\, b) \qquad \rightarrow \qquad k_p = \min(a_p,\, b_p) \qquad \text{for all } p$$

Finding the prime factors of a large number is not easy so the above relationship does not give a direct way of finding the greatest common divisor.

The integers x and y are relatively prime (also referred to as being co-prime) if they have no prime factors in common i.e. their only common factor is 1. This is equivalent to saying that x and y are

relatively prime if gcd(x, y) = 1.
e.g. 14 and 33 are relatively prime as the divisors of 14 are 1, 2, 7 and 14, and the divisors of 33 are 1, 3, 11 and 33.

Modular Arithmetic

Modular arithmetic uses positive integers that are less than a certain value called the modulus.
x mod y is read as ‘x mod y’ or ‘x modulo y’ and means the remainder when x is divided by y.
So

x mod 4	x mod 7
0 mod 4 = 0	0 mod 7 = 0
1 mod 4 = 1	1 mod 7 = 1
2 mod 4 = 2	2 mod 7 = 2
3 mod 4 = 3	3 mod 7 = 3
4 mod 4 = 0	4 mod 7 = 4
5 mod 4 = 1	5 mod 7 = 5
6 mod 4 = 2	6 mod 7 = 6
7 mod 4 = 3	7 mod 7 = 0
.	.

Note the (mod n) operator maps all integers into the set of integers $\{0, 1, \ldots (n-1)\}$.
If an integer x is divided by another integer y we obtain a **quotient** q and a **remainder** (or **residue**) r such that:

$$x = qy + r \qquad 0 \leq r < y;\ q = \lfloor x/y \rfloor$$

where $\lfloor a \rfloor$ is the largest integer less than or equal to a.
For any integer x

$$x = \lfloor x/n \rfloor \times n + (x \bmod n)$$

where n is a positive integer.
Two integers x and y are **congruent** modulo n, if (x mod n) = (y mod n). This is written as $x \equiv y \bmod n$.
e.g. $71 \equiv 3 \bmod 17$; and $21 \equiv -9 \bmod 10$
If $x \equiv 0 \bmod n$, then $n|x$
Properties of the modulo operator

- $x \equiv y \bmod n$ if $n|(x-y)$
- $(x \bmod n) = (y \bmod n)$ implies $x \equiv y \bmod n$
- $x \equiv y \bmod n$ implies $y \equiv x \bmod n$
- $x \equiv y \bmod n$ and $y \equiv z \bmod n$ imply $x \equiv z \bmod n$

Chinese Remainder Theorem

If $x \equiv y \bmod p$ and $x \equiv y \bmod q$ with p and q being relative primes then $x \equiv y \bmod pq$

Fermat/Euler Theorem

If p is prime and $0 < x < p$ then $x^{p-1} \equiv 1 \bmod p$

The RSA Algorithm

RSA is a block cipher (processes one block of elements at a time - producing an output block for each input block) in which the plaintext, M, and ciphertext, C, are integers between 0 and n-1 for some n.

Key Generation

1. Generate two large prime numbers, p and q.
2. Let $n = pq$.
3. Let $\phi(n) = (p-1)(q-1)$. — ϕ - **totient** function - number of numbers $< n$ relatively prime to n.
4. Choose a number, the encryption exponent, e, relative prime to $\phi(n)$. — i.e. $\gcd(\phi(n),\ e) = 1;\ 1 < e < \phi(n)$
5. Find d, the decryption exponent, such that $de \bmod \phi(n) = 1$. — i.e. $d = e^{-1} \bmod \phi(n)$.
6. Publish e and n as the public key.
7. Keep d and n as the private key.

Encryption

8. $C = M^e \bmod n$.

Decryption

9. $M = C^d \bmod n$.

RSA – Example

Starting with the two primes 7 and 17. Find a public-private key pair, with the encryption exponent less than the decryption exponent. Use the public key to encrypt the message who's numerical encoding is 3.

Form the modulus i.e. the product of the primes $\mathrm{n} = 7 \times 17 = 119$

Obtain the totient $\phi = (\mathrm{p} - 1) \times (\mathrm{q} - 1) = 96$

Choose a low value encryption exponent, e, relative prime to ϕ e.g. 5

Find the decryption exponent, d, such that $\mathrm{de} \bmod \phi = 1$.

i.e. de is 1 plus an integer multiple of ϕ, hence, with $\mathrm{i} = 4$, $\mathrm{d} = \dfrac{1 + \mathrm{i} \times 96}{5} \rightarrow \mathrm{d} = 77$

So the public and private keys are $(5, 119)$ and $(77, 119)$

Encoding of 3: $3^5 \bmod 119 = 243 \bmod 119 = 5$

Proof that decryption works.

Since $de \equiv 1 \pmod{\phi}$ $\qquad de = 1 + k\phi$ $\qquad$ where k is an integer.

If $\gcd(M, p) = 1$ then by Fermat's theorem

$$M^{p-1} \equiv 1 \pmod{p}$$

Raising both sides of this congruence to the power $k(q-1)$ and then multiplying both sides by M giv

$$M^{1+k(p-1)(q-1)} \equiv M \pmod{p}$$

If $\gcd(M, p) = p$, the above congruence is still valid since each side is congruent to 0 mod p.

Therefore, in all cases

$$M^{ed} \equiv M \pmod{p}$$

By the same argument,

$$M^{ed} \equiv M \pmod{q}$$

Since p and q are distinct primes, by the Chinese remainder theorem, it follows that

$$M^{ed} \equiv M \pmod{n}$$

Therefore

$$C^d \equiv (M^e)^d \equiv M \pmod{n}$$

Some Practicalities

Exponentiation with large numbers

The RSA algorithm requires that large numbers (blocks of plaintext or ciphertext) be raised to the power of some large number mod another large number. With a multiple precision arithmetic package this is possible but slow. e.g. 117^{38} mod 568 could be obtained by multiplying 117 by itself 38 times (79 decimal digits, 262 binary digits) and then taking the remainder on division by 568. For RSA to be secure the numbers must be of the order of 150 digits.

> Raising a 150 digit number to a 150 digit power by this method would exhaust the capacity of all existing computers for more than the expected life of the universe.
> Kaufman, C. et al. (2002), p154

Modular reduction after each multiply keeps the numbers from getting out of hand.

e.g. $117^2 \bmod 568 = 13689 \bmod 568 = 57$
$117^3 \bmod 568 = 117 \cdot 57 \bmod 568 = 6669 \bmod 568 = 421$
$117^4 \bmod 568 = 117 \cdot 421 \bmod 568 = 49257 \bmod 568 = 409$
... and so on.

Even this is still inefficient - 38 multiplications and 38 'divisions'.

Raising a number to some power of two by multiple squaring. A somewhat similar technique can be used to raise a number to the power of any positive integer.
To obtain 117^{2x} we square 117^x, and to obtain 117^{2x+1} we square 117^x and then multiply by 117.

$$117^{38} = \left(117^{19}\right)^2 = \left(\left(117^9\right)^2 \cdot 117\right)^2 = \left(\left(\left(117^4\right)^2 \cdot 117\right)^2 \cdot 117\right)^2 = \left(\left(\left(\left(117^2\right)^2\right)^2 \cdot 117\right)^2 \cdot 117\right)^2$$

So evaluating 17^{38} mod 568 can be carried out using 7 multiplications and 7 'divisions'.

The most "inefficient" way

```
>>> 117**38 % 568
289L
```

7 multiplications and 7 'divisions'

```
>>> 117**2 % 568
57
>>> 57**2 % 568
409
>>> 409**2 % 568
289
>>> 289*117 % 568
301
>>> 301**2 % 568
289
>>> 289*117 % 568
301
>>> 301**2 % 568
289
```

Finding d and e

There are two strategies to ensure that e and $(p-1)(q-1)$ are relatively prime.

1. After p and q are selected choose e at random. Test to see if e relative prime to $(p-1)(q-1)$. If not select another e.

2. Choose e first then select p and q so that $(p-1)$ and $(q-1)$ are guaranteed to be relatively prime to e.

Common values of e

Two common values of e are 3 and 65537. A small e such as 3 means that encryption using the public key is relatively fast. Decryption is slower. $65537 = 2^{16} + 1$ is prime and only requires 17 multiplications to exponentiate and avoids the problems there are with 3.
If Alice sends the same message to Bob, Fred and Jim and Bob, Fred and Jim all have the same encryption exponent $e = 3$ then Alice sends ciphertexts $C_i = M^3 \bmod n_i$ for $i = 1, 2, 3$. An eavesdropper can find a solution $0 \leq x < n$ to the three congruences $x \equiv c_i \pmod{n_i}$ for $i = 1, 2, 3$. Since $M^3 < n_1 n_2 n_3$, by the Chinese remainder theorem $x = M^3$. So, by taking the integer cube root of x, M can be recovered. This can be avoided by **salting** the message.

Small or Predictable Messages

The plaintext from a ciphertext, C, can be discovered by simply encrypting all possible plaintext messages until the resultant ciphertext matches C. This is known as a **forward search attack**.

Simple Python RSA example to illustrate what happens when the message is greater then the product of primes and when the encryption exponent is not relative prime to the totient.

```
import primes
import sys
p = 3                   # one prime number
q = 11                  # another prime
n = p*q                 # the product of the primes
phi = (p-1)*(q-1)       # the totient
e = 3                   # 3 is relatively prime to 20

# Find the value of the decryption exponent
d = primes.invmod(phi,e)
print ' M\tE(M)\tD(E(M))'

# M < n
for i in range(33):
    print '%2i\t%2i\t%2i' % (i, pow(i,e,n),pow(pow(i,e,n),d,n))

# M >= n
print '\n\n M\tE(M)\tD(E(M))'
for i in range(33,66):
    print '%2i\t%2i\t%2i' % (i,pow(i,e,n),pow(pow(i,e,n),d,n))

# e not relative prime to totient
e = 5                   # 5 is not relatively prime to 20
# Find the value of the decryption exponent
d = primes.invmod(phi,e)
print '\n\n M\tE(M)\tD(E(M))'
for i in range(33):
    print '%2i\t%2i\t%2i' % (i, pow(i,e,n),pow(pow(i,e,n),d,n))
```

M^e mod 33

$M < 33$			$M \geq 33$			$M < 33$		
e relative prime to ϕ			e relative prime to ϕ			e not relative prime to ϕ		
M	E(M)	D(E(M))	M	E(M)	D(E(M))	M	E(M)	D(E(M))
0	0	0	33	0	0	0	0	0
1	1	1	34	1	1	1	1	1
2	8	2	35	8	2	2	32	32
3	27	3	36	27	3	3	12	12
4	31	4	37	31	4	4	1	1
5	26	5	38	26	5	5	23	23
6	18	6	39	18	6	6	21	21
7	13	7	40	13	7	7	10	10
8	17	8	41	17	8	8	32	32
9	3	9	42	3	9	9	12	12
10	10	10	43	10	10	10	10	10
11	11	11	44	11	11	11	11	11
12	12	12	45	12	12	12	12	12
13	19	13	46	19	13	13	10	10
14	5	14	47	5	14	14	23	23
15	9	15	48	9	15	15	12	12
16	4	16	49	4	16	16	1	1
17	29	17	50	29	17	17	32	32
18	24	18	51	24	18	18	21	21
19	28	19	52	28	19	19	10	10
20	14	20	53	14	20	20	23	23
21	21	21	54	21	21	21	21	21
22	22	22	55	22	22	22	22	22
23	23	23	56	23	23	23	23	23
24	30	24	57	30	24	24	21	21
25	16	25	58	16	25	25	1	1
26	20	26	59	20	26	26	23	23
27	15	27	60	15	27	27	12	12
28	7	28	61	7	28	28	10	10
29	2	29	62	2	29	29	32	32
30	6	30	63	6	30	30	21	21
31	25	31	64	25	31	31	1	1
32	32	32	65	32	32	32	32	32

RSA Examples using Python

```
# RSA demonstration of the encryption and decryption
# of text 16 bits at a time
# Note: the product of the primes, n, must be > 65535

p = 211
q = 373
n = p*q
phi = (p-1)*(q-1)
e = 11          # already checked
d = 42611       # already evaluated using invmod from primes module

text = "abcde"

# pad text so that there are a
# whole number of 2 byte blocks
if len(text) % 2:
    text += ' '

# split text into two byte integer blocks
# should be using a list/array here
block1 = (ord(text[0])<<8) + ord(text[1])
block2 = (ord(text[2])<<8) + ord(text[3])
block3 = (ord(text[4])<<8) + ord(text[5])

print block1, block2, block3

# encrypt each block
cblock1 = pow(block1,e,n)
cblock2 = pow(block2,e,n)
cblock3 = pow(block3,e,n)

print cblock1, cblock2, cblock3

# decrypt each block
dblock1 = pow(cblock1,d,n)
dblock2 = pow(cblock2,d,n)
dblock3 = pow(cblock3,d,n)

print dblock1, dblock2, dblock3

# recover original text plus any padding
dtext = chr(dblock1>>8)+chr(dblock1&0xFF) + chr(dblock2>>8)+chr(dblock2&0xFF)\
        + chr(dblock3>>8)+chr(dblock3&0xFF)

print dtext
```

```
# RSA demonstration of the encryption and decryption
# of a file read and encrypted in 16 bit blocks
# Note: the product of the primes, n, must be > 65535

p = 211
q = 373
n = p*q
phi = (p-1)*(q-1)
e = 11
d = 42611

infd = open('infile.txt','rb')
outfd = open('outfile.txt','wb')

while 1:
    text = infd.read(2)
    lentext = len(text)
    if lentext <= 2 and lentext > 0:
        while lentext < 2:
            text += ' '
            lentext += 1
    else:
        break
    ptext = (ord(text[0])<<8) + ord(text[1])
    ctext = pow(ptext,e,n)
    outfd.write('%s\n' % str(ctext))
infd.close()
outfd.close()

infd = open('outfile.txt','rb')
outfd = open('outfile2.txt','wb')

while 1:
    text = infd.readline()
    if len(text) == 0:
        break
    else:
        ctext = int(text)
        ptext = pow(ctext,d,n)
        outfd.write(chr(ptext>>8)+chr(ptext&0xFF))
infd.close()
outfd.close()
```