RSA - Why does it work?

This document is incomplete and in preliminary condition.
Proof by Partha Dasgupta, based on a proof by Zvi M. Kedem

Outline of RSA algorithm:

p, q are distinct primes
N = p•q

Define: Φ = (p-1)(q-1)

Find a, b such that they are relatively prime to Φ and a•b = 1 mod Φ
- e.g. a = 65537 = 2^16+1

Encryption of message m: $E_e(m) = m^b \bmod n$ = C (cipher)
Decryption of C : $D_d(C) = y^a \bmod n = m$

Encryption Key: $e = (b, n)$
Decryption Key: $d = (a, n)$ [Note, e and d are interchangeable]

So in order for RSA to work we must have the property :

$$(m^b)^a = m \bmod n \quad \text{- [1]}$$

In this document (7 pages, large font text), we will
- Prove [1]
- Show how to find p, q, a and b.
- Show how to compute m^b fast

Preliminaries:

$\mathbf{Z}_n = \{0, 1, 2, \ldots n\text{-}1\}$
$\mathbf{Z}^*_n = \{x < n\text{-}1 \mid x \text{ and } n \text{ are relatively prime}\}$
$\phi(n)$ = number of elements of $\mathbf{Z}_n$ that are relatively prime to n.
Hence $\phi(n) = |\mathbf{Z}^*_n|$

How to find ϕ(n)?

All those number that are not multiples of p or q are in ϕ(n). If we count all the multiples of p and q.

0, 1
$p, 2p, 3p, \ldots (q-1)p$ $q-1$
$q, 2q, 3q, \ldots (p-1)q$ $p-1$

hence

$$\phi(n) = pq - 1 - (q-1) - (p-1) = pq - p - q + 1 = (p-1)(q-1)$$

Example: $p=3$, $q=5$ $n=15$.

Now $\phi(n) = 2*4 = 8 = \{1, 2, 4, 7, 8, 11, 13, 14\}$

Claim 1: Z^*_n is closed under multiplication mod *n*.

If $a,b \in Z^*_n$ then ab and n are relatively prime i.e. ab shares no primes with n. By definition of Z^*_n a, b do not share primes with n. Their product, ab, gets its primes from a and b and therefore does not share primes with n.

The product can be written as $ab = \alpha n + \gamma$. We just need to show that γ is in Z^*_n . But if it is not, then it shares primes with n and the right hand side is divisible by some prime that is a factor of n. But then, so is the left side, which is impossible as we showed that it is relatively prime with n.

EndClaim1

Definition:

$\mathbf{Z}^*_n = \{b_1, b_2, \ldots, b_{\phi(n)}\}$

For any $a \in \mathbf{Z}^*_n$,

Let $S_a = \{a \bullet b_1 \bmod n,\ a \bullet b_2 \bmod n, \ldots, a \bullet b_{\phi(n)} \bmod n\}$

Claim 2: $S_a \equiv Z^*$

First, by Claim 1, all elements of S_a are in $\mathbf{Z}^*_n$

Second, no two elements can be the same. Suppose they were, then for some b_i and b_j ($b_i < b_j$)

$a \bullet b_i = \alpha n + \gamma$

$a \bullet b_j = \beta n + \gamma$

Subtracting, $(b_i - b_j) \bullet a = (\beta - \alpha) \bullet n$ or $x \bullet a = y \bullet n$

$x \bullet a$ and $y \bullet n$ are the "same product of primes. Since a and n do not share any common primes. All primes that form n has to appear in x.
Hence $x >= n$. That is a contradiction, as $b_j < n$.

Since all elements of S_a are distinct, and in $\mathbf{Z}^*_n$, then S_a and $\mathbf{Z}^*_n$ are identical.
Note that since all elements of $\mathbf{Z}^*_n$ are produced when a is multiplied by each element of $\mathbf{Z}^*_n$, then the element 1 is also a result of such a multiplication. Hence we get the following: [Note: Not useful at this point]

Corollary: if $a \in Z^*_n$ then $\exists\ b_k \in Z^*_n$, s.t. $ab_k = 1 \bmod n$

Corollary is proven in the prior paragraph.

EndClaim2

Claim3: if $a \in Z^*_n$ then $a^{\phi(n)} = 1 \bmod n$

Define c and A such that:

$$b_1 \bullet b_2 \bullet \ldots \bullet b_{\phi(n)} = c \bmod n$$
$$(ab_1 \bullet ab_2 \bullet \ldots \bullet ab_{\phi(n)}) = A \bmod n \quad \text{[Note, } A \text{ and } c \text{ are less than n]}$$

Now since $ab_1 \bmod n \bullet ab_2 \bmod n \bullet \ldots \bullet ab_{\phi(n)} = A \bmod n$
By Claim 2, $ab_1 \bmod n, ab_2 \bmod n, \ldots \bullet ab_{\phi(n)} \bmod n$ is a permutation of $\mathbf{Z}^*_n$

Hence: $A = c$ (plain arithmetic, both are less than n)

Now we take the following equation:

$$(ab_1 \bullet ab_2 \bullet \ldots \bullet ab_{\phi(n)}) = a^{\phi(n)} \bullet (b_1 \bullet b_2 \bullet \ldots \bullet b_{\phi(n)}) \text{ (plain arithmetic)[1]}$$

Now take the modulus of both sides:

$$A = (a^{\phi(n)} \bullet c \bmod n),$$

Since c is less than n, $a^{\phi(n)} \bmod n$ must be 1.

Thus: $a^{\phi(n)} = 1 \bmod n$

EndClaim3

Claim4: if a and $\phi(n)$ are relatively prime then ∃ b, s.t. $a \bullet b = 1 \bmod \phi(n)$

If a and $\phi(n)$ are relatively prime, then $a \in \mathbf{Z}^*_{\phi(n)}$ and from Corollary of Claim 2 we know that b exists (and is a member of $\mathbf{Z}^*_{\phi(n)}$).

Thus there exists a and b, both relatively prime to $\phi(n)$, such that:

$a \bullet b = k\phi(n) + 1$ (regular arithmetic)

EndClaim4

Proof of RSA (for all messages in Z^*_n)

Take a message $m < n$ and choose a relatively prime to $\phi(n)$ and find b such that $a \bullet b = 1 \bmod \phi(n)$.

Now compute $(m^a)^b$ using modulo n arithmetic:

$$(m^a)^b = m^{k\phi(n)+1} = m^{k\phi(n)}\, m = m^{\phi(n)} m^{\phi(n)} \cdots \cdots m^{\phi(n)}\, m$$

Take the modulo of the last term and since $m^{\phi(n)} = 1 \bmod n$, then result is m.

Hence $(m^a)^b = m \bmod n$

Deficiency of this proof: The proof is for all messages in Z^*_n
If n=512 bit number, then the chance of a number being in Z_n but not in Z^*_n is about 10^{-25}. That is negligible ☺

To fix this problem: There is a proof that all numbers in have the above property, but that proof is rather complex.
EndProof

How to really find a, b?

We know that given a, b exists, but how to find them?

Find a, relatively prime to $\phi(n)$ (3, 5, 7 etc – start with a small odd number and work your way up). Note that $\phi(n)$ is even.

Then find b using extended Euclidean algorithm as follows

Extended Euclidean Algorithm:
Given p and q, $p>q$ the algorithm finds x and y, such that
$x \bullet p + y \bullet q = \text{GCD}(p, q)$
[note: regular arithmetic, x or y is negative]

So we use it as follows:

- We provide $\phi(n)$ and a as input (p and q) and get x and y [note: GCD(a,n) = 1] – that is we get the values of x and y, such that $a*y + \phi(n)*x = 1$.
- *Also note that* $a \bullet b = 1 \bmod \phi(n)$, that is
  $a \bullet b = \eta \bullet \phi(n) + 1$ *or* $a \bullet b - \eta \bullet \phi(n) = 1$
  So $b = y$

Hence in modulo arithmetic,

- $b = y$, if y is positive and
- $b = \phi(n) - y$, if y is negative

Now we need to find *p, q* and hence *N*.

p and q are large prime numbers. So the problem is to find large prime numbers. There is no good deterministic way of doing this. However we can do it with probabilistic algorithms.

Fact: There are lots of large prime numbers. The number of prime numbers below N is about $N/(\ln n)$ and hence for a random 2048 bit number, the probability of it being prime is about 0.0007(one in 1500).

Claim 5: If *p* is prime, for any *a* < *p*, $a^{p-1} = 1 \bmod p$

Since p is prime, $a \in Z^*_p$ and $\phi(p) = p - 1$
Thus $a^{p-1} = a^{\phi(p)} = 1 \bmod p$
EndClaim

Claim 6: If p is prime, the equation $x^2 = 1 \bmod p$. has only 2 solutions, 1 and $p-1$ (or $-1 \bmod p$)

Lets say the equation has 2 solutions, S_1 and S_2 .

Thus $S_1^2 = 1 + kp$

Hence $(S_1+1) \bullet (S_1-1) = kp$

This means either (S_1+1) or (S_1-1) or both is divisible by p.

Suppose both are divisible by p. But these two numbers are only 2 apart – and unless if p=2 this is not possible.

Thus only one of them is divisible by p. If (S_1+1) is divisible, then

$S_1 = 1 \bmod p$.

If (S_1-1) is divisible by p then

$S_1 = -1 \bmod p$ (or $S_{1+1} = p-1 \bmod p$)

Thus the two solutions have to be 1 and –1. (same happens for S_2)

EndClaim

PRIME NUMBER HUNT

Choose a number p, randomly. This number, if large has a chance of being prime in the order of 1 in several thousand (*good*!).

Then choose a number $a < p$, randomly. We will use a to test the primality of p. First, we know that if p is prime, a^{p-1} is 1 (mod p). Secondly, we know that if for some x (where x is not 1 or p-1), if x^2 is 1 or p-1 (mod p) then p is not prime.

Now we compute a^{p-1} **fast**. Since the computing involves squaring numbers, we can do the x^2 test also.

Computing a^x can be done as follows. Put 1 in a result variable. Take the binary representation of x. Then for every bit in the binary representation, working from left to right (MSB to LSB), for every 0 square the result variable. For every 1, square the result variable and multiply with a.

```
b[k] b[k-1] … b[1] b[0] is the binary representation of x

result = 1                    // start with the value of a^0
for i = k downto 0 {          //from MSB to LSB
   temp = result;             // store prev result for checking
   result = (result * result) mod n  //square prev result
   //if we are doing primality testing then add this step
   if (result = 1) and (temp!=1) and (temp!=n-1)
                 then p is not prime; //by Claim 6
                 break;
   if b[i] = 1 then result = (result*a) mod n //mult by a
}
// now we know n is possibly prime
```

If the above test says "*possibly prime*" then the number p is not prime with probability 0.5. Hence if we run the test R times, then p is not prime, with probability $(0.5)^R$. If R = 100 and for all the 100 tests the result was "*possibly prime*", then the chance of the number being not prime is a one in a million.

So we select 1 large number. Test for primality about 500 times. In about a 2000 choices, we will find a prime number. Do it again for another prime number. Now call them p and q. All this should take about 1-2 seconds. Definitely under 10 secs.

And the rest, as they say is trivial ☺

Note that encryption and decryption uses the same fast exponenting algorithm as shown above.
