Euler's Theorem and RSA Public Key Cryptography

Timothy Weaving

January 18, 2016

1 Euler's Theorem

1.1 Introduction to Euler's Totient Function $\varphi(n)$

Euler's totient function $\varphi(n), n \in \mathbb{Z}$ counts how many numbers less than n are coprime to n (such numbers are known as 'totatives'); two integers being coprime or 'relatively prime' if their greatest common divisor (gcd) is 1. Stated below is the totient or *phi* function in which the product $\prod_{p|n}\left(1-\frac{1}{p}\right)$ is taken over the unique prime factors p of n. For example $\varphi(10) = 10 \cdot \left(1-\frac{1}{2}\right) \cdot \left(1-\frac{1}{5}\right) = 4$, with 2 and 5 being the unique prime factors of 10, and indeed there are four numbers less than 10 which are coprime to it: $\{1, 3, 7, 9\}$.

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

It is important to note that $\varphi(n)$ is multiplicative. If m and n are coprime, ie. $gcd(m,n) = 1$, then $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$. More generally if $gcd(m,n) = d$ then $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n) \cdot \frac{d}{\varphi(d)}$ ($\frac{d}{\varphi(d)}$ of course equals 1 when m and n are coprime hence the previous result). See the link at the end of the document for code which verifies this for a large number of cases.

1.2 Interesting Patterns

Interestingly when we plot the values of $\varphi(n)$ distinct lines start to emerge from the plotted points (Figure 1). The uppermost of these lines is trivial as this corresponds to $\varphi(n)$ when n is prime: $\varphi(p) = p \cdot \left(1 - \frac{1}{p}\right) = p - 1$. An explanation for the apparent lines below this is perhaps slightly less obvious. Considering instead the plot of $\frac{\varphi(n)}{n}$ we again see lines appearing though now more clearly and constant at particular values (Figure 2). These values arise when n is prime: $\frac{\varphi(p)}{p} = \left(1 - \frac{1}{p}\right) = \frac{1}{2}, \frac{2}{3}, \frac{4}{5}, \frac{6}{7}, \frac{10}{11}, \frac{12}{13}, \frac{16}{17}\dots$ The lines drawn at these values do indeed coincide with the the lines seen in the plotted points (Figure 3).

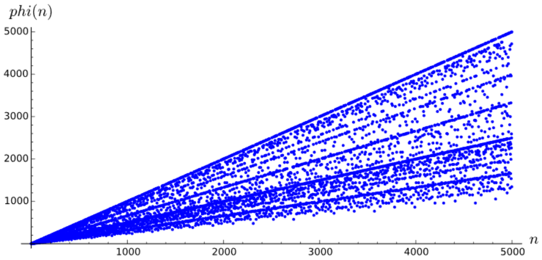


Figure 1: $\varphi(n)$ for $n < 5000$

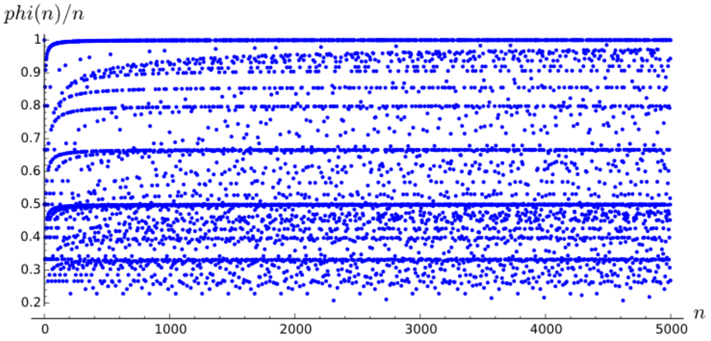


Figure 2: $\varphi(n)/n$ for $n < 5000$

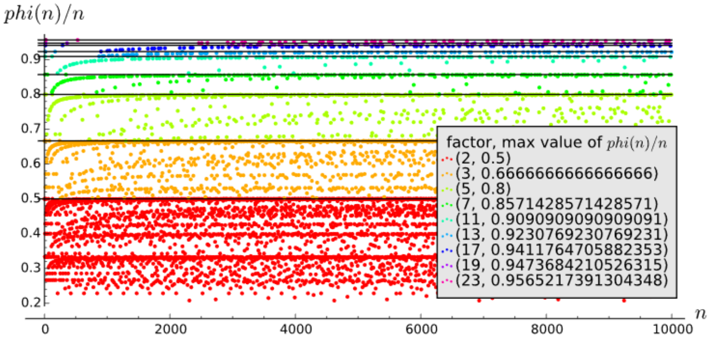


Figure 3: $\varphi(n)/n$ for $n < 10000$ with upper- bounds

The points have been coloured in figure 3 such that those which contain the same *lowest* prime factor are all coloured identically (so 6 is coloured red even though it contains a factor of 3, 15 is orange as its lowest prime factor is 3 etc.). The points have been presented in this way as the lowest prime factor of a number has the greatest weighting on its totient function.

Suppose n is a composite number resulting from the product of many unique primes. Then $\varphi(n) = n\big(1-\frac{1}{p_1}\big)\big(1-\frac{1}{p_2}\big)\big(1-\frac{1}{p_3}\big)...$ where the term $\big(1-\frac{1}{p_n}\big)$ gets arbitrarily close to 1 (as $\lim_{n\to\infty}\frac{1}{p_n} = 0$). We are multiplying by a term which gets closer and closer to 1 and so has less of an effect on the totient. Therefore if a number n contains a factor of 2 then $\varphi(n) \leq \frac{1}{2}n$, a factor of 3: $\varphi(n) \leq \frac{2}{3}n$, a factor of 5: $\varphi(n) \leq \frac{4}{5}n$ etc. which is visualised in figure 3. *code for plots included in link*

1.3 Euler's Theorem

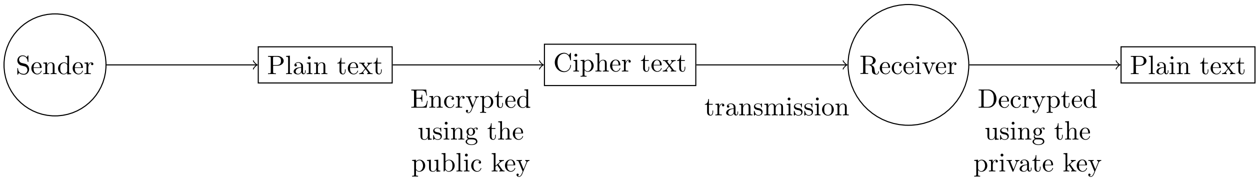
Modular or 'clock' arithmetic appears very often in number theory. With usual arithmetic it would seem odd to say $10+5 = 3$ but when considering time on a clock this is perfectly acceptable. This is because clocks run modulo 12, where the numbers 'wrap around' upon reaching a certain number which we call the modulus; $a \bmod b$ essentially finds the remainder of the division $\frac{a}{b}$. 17th century mathematician Pierre de Fermat stated in 1640 without proof (now dubbed 'Fermat's little theorem') that $a^p \equiv a \pmod p$, where the modular notation $x \equiv y \pmod n$ is the same as $x \bmod n = y \bmod n$. Euler discovered numerous proofs of this which eventually led to his generalisation of the statement. In 1763 he presented Euler's Theorem, which states (see code for verification):

$$a^{\varphi(n)} \equiv 1 \pmod n$$

2 RSA Public Key Cryptography

2.1 Introduction to RSA

RSA, named after its founders Rivest, Shamir and Adleman, is the most widely used public key cryptosystem. The idea behind public key cryptography is that a publicly available key is used to encrypt a message (called 'plain text'). The encrypted plain text is known as 'cipher text' and can be sent to an intended receiver without need for a secure connection. The cipher text cannot then be decrypted unless a private key is known; this is not published to the general public.


Sender
Plain text
Encrypted using the public key
Cipher text
transmission
Receiver
Decrypted using the private key
Plain text

Exactly *what* the public and private keys are however is where Euler's theorem comes in. The strength of RSA is in the difficulty of factoring very large numbers into their prime decomposition. Calculating the product of two very large primes is an easy task for a computer, but to find the initial primes when only the final product is given is very demanding. The following is a summary of the RSA algorithm altered from Wade Trappe's book [1, p.165]:

1. The receiver chooses secret primes p and q and computes $n = pq$.
2. They then choose e with $gcd(e, \varphi(n)) = 1$. (note that $\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1)$)
3. d is then computed with $de \equiv 1 \pmod{\varphi(n)}$.
4. n and e are made public, whereas p, q, d are kept secret.
5. A sender encrypts their plain text m as $c \equiv m^e \pmod n$ and sends c to the receiver.
6. The receiver decrypts the cipher text c by computing $m \equiv c^d \pmod n$.

If the cipher text was intercepted during its transmission along with the publicly available n and e it would not be readable as the secret exponent d is required to do so - even the sender cannot read c once encrypted. There is an issue here however as this method requires performing modular arithmetic efficiently on truly massive numbers. In the real world current RSA keys are usually 1024 - 2048 bits long; approximately 300 - 600 decimal digits [2]! Fortunately there is a method known as modular exponentiation which allows us to take the mod of a number raised to a large exponent very quickly.

2.2 Mini Sage RSA Cryptosystem

Below is a rudimentary RSA cryptosystem coded in sage which allows messages written with upper case characters to be encrypted and decrypted efficiently. The converter for text to ascii uses utility functions by Minh Van Nguyen [3] and some of the code for the main RSA algorithm is adapted from [4]. Note that $xgcd(x, y)$ is the extended euclidean algorithm and calculates the coefficients s and t of $gcd(x, y) = s \cdot x + t \cdot y$ where s in our case is the secret exponent d.

```
def textconverter(s): #for function see code in link

def keygenerator(p1, p2):
   n = p1 * p2 #calculates n (which would be made public)
   phi = (p1 - 1) * (p2 - 1) #totient function of n (secret)
   e = randint(phi / 2, phi) #create an initial pseudorandom value for e
   while gcd(e, phi) != 1: #while loop until e coprime to phi found
      e += 1 #increment e value
   d = xgcd(e, phi)[1] #extended euclidean algorithm outputs triplet (gcd, s, t) - we want our secret exponent d
   return [n, e], d

p = [random_prime(10 ^ 40), random_prime(10 ^ 40)] #generates two random primes (secret)
key = keygenerator(p[0], p[1]) #generates/calculates n, e, d
public = key[0] #these numbers are made publicly available
private = key[1] #this is kept secret

msg = "RSA IS A POWERFUL MEANS OF ENCRYPTION"; 'Message to encrypt: %s' % msg #create message to encrypt
plain = textconverter(msg); 'plaintext(Ascii) = %i' % plain #uppercase only due to length of ascii representation
cipher = power_mod(plain, public[1], public[0]); 'ciphertext = %i' % cipher #modular exponentiation to encrypt
-------------------------------------------------------------------------------------------------------------------
OUTPUT:
'Message to encrypt: RSA IS A POWERFUL MEANS OF ENCRYPTION'
'plaintext(Ascii) = 82836532738332653280798769827085763277696578833279703269786782898084737978'
'ciphertext = 6804216478003829339836876946833929657297198064970275291115636282874669653914797'
-------------------------------------------------------------------------------------------------------------------
def asciiconverter(a): #for function see code in link

dcrpt = power_mod(cipher, private, public[0]); 'decryption returns plaintext = %i' % dcrpt #the decryption step

print 'plaintext == decryption:', plain == dcrpt #bool: original text == decrypted text?
print 'Ascii converted back to text: %s' % asciiconverter(dcrpt) #convert ascii to text using asciiconverter()
-------------------------------------------------------------------------------------------------------------------
OUTPUT:
'decryption returns plaintext = 82836532738332653280798769827085763277696578833279703269786782898084737978'
'plaintext == decryption: True'
'Ascii converted back to text: RSA IS A POWERFUL MEANS OF ENCRYPTION'
-------------------------------------------------------------------------------------------------------------------
```

Some functions and comments have been omitted in order to fit in the document, see link at end of document for full code

2.3 Where is RSA used?

Whenever a small padlock appears next to the url in your internet browser RSA is being implemented. SSL (secure sockets layer) certificates are used to validate a website's legitimacy and operate using RSA. When a computer connects to a server with an SSL certificate what is known as an 'SSL handshake' is performed during which there is an exchange of messages which allow the server to authenticate itself. RSA is used also for 'digital signatures' which are attached to a message or document and can be used to show that it was sent from a trusted sender. Both SSL certificates and digital signatures are involved when a payment is made online; RSA public key is there to help ensure that the transaction is secure.

References

[1] Wade Trappe and Lawrence C Washington. *Introduction to cryptography with coding theory.* Pearson Education India, 2006.

[2] Wolfram mathworld - rsa number. `http://mathworld.wolfram.com/RSANumber.html`. Accessed: 2015-12-06.

[3] Utility functions for cryptography. `http://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/util.html`. Accessed: 2015-12-05.

[4] Number theory and the rsa public key cryptosystem. `http://doc.sagemath.org/html/en/thematic_tutorials/numtheory_rsa.html`. Accessed: 2015-12-05.

Code used can be seen here: **https://cloud.sagemath.com/projects/c58be8bd-55ba-4e9f-a792-edb915fb3276/files/**