RSA

named after the three researchers who created it

Rivest, Shamir, Adleman

can be used to encrypt and decrypt data

one of the first implemented public-key-based cryptosystems

is widely used even today

its security relies on the large prime number factoring problem

protocol

step 1: two large prime numbers p and q are chosen

step 2: calculate n, such that $n=p*q$

this is known as the modulus (in modular arithmetic) – more on this later

step 3: calculate z, such that $z=lcm(p-1,q-1)$

this is an interpretation of the Charmichael function

lcm (least common multiple) is the smallest number that is a multiple of both numbers

e.g., 3 and 5: *lcm* is 15

z can be calculated as follows: $z=\frac{(p-1)*(q-1)}{gcd(p-1,q-1)}$

the numerator is the Euler totient function

in fact, the Euler totient function was originally used in RSA instead

gcd (greatest common divisor) is the largest number that evenly divides both

e.g., 5 and 25: *gcd* is 5

*see below for an algorithm to compute the *gcd**

step 4: select e, such that $1<e<z$, where e is a prime number and is **coprime** to z

coprime? $gcd(e,z)=1$

step 5: now solve for d, such that $(d*e) \bmod z=1$

this is the same as $d=e^{-1} \bmod z$

this is a numerical method concept that expresses the modular inverse

in regular arithmetic, a number multiplied by its inverse is 1

for A, the inverse is $\frac{1}{A}$

because $A*\frac{1}{A}=1$

in modular arithmetic, there is no division operator

but we do have modular inverses

the modular inverse of $A(\bmod C)=A^{-1}$

or $(A*A^{-1}) \bmod C=1$

only the numbers coprime to C have a modular inverse $(\bmod C)$

how do we calculate this?

naive: calculate $e*d \bmod z$ for values of d from 0 through $z-1$

the modular inverse of $e \bmod z$ is the value of d that makes $(e*d) \bmod z=1$

e.g., $e=3, z=7$

$3*0=0(\bmod 7)=0$

$3*1=3(\bmod 7)=3$

$3*2=6(\bmod 7)=6$

$3*3=9(\bmod 7)=2$

$3*4=12(\bmod 7)=5$

$3*5=15(\bmod 7)=1$ ← here we go!

$3*6=18 \pmod 7=4$ ← we didn't need to go this far since we already found d

there is a better/faster way using the extended Euclidean algorithm

look it up!

step 6: we can now generate the public and private keys:

private key: $K_{priv}=(d,n)$

public key: $K_{pub}=(e,n)$

to encrypt message M, ciphertext C is calculated as follows:

$$C=M^e \pmod n$$

to decrypt ciphertext C, plaintext message M is calculated as follows:

$$M=C^d \pmod n$$

note: the keys can be used to encrypt and decrypt any message smaller than the value of n

of course, characters are just integers in a computer...right?

gcd algorithm

we can do this recursively: $gcd(a,b)$

the idea is to repeatedly calculate $gcd(b, a\%b)$ until $b=0$ (in which case, we return a)

$$gcd(a,b)=\begin{cases} a & \text{, if } b=0 \\ gcd(b, a\%b) & \text{, otherwise} \end{cases}$$

e.g., $a=7, b=180$:

$gcd(180,7)$

$gcd(7,5)$

$gcd(5,2)$

$gcd(2,1)$

$gcd(1,0)$

1

e.g., $a=5, b=25$:

$gcd(25,5)$

$gcd(5,0)$

5

e.g., $a=10, b=18$:

$gcd(18,10)$

$gcd(10,8)$

$gcd(8,2)$

$gcd(2,0)$

2

in practice

step 1: two large prime numbers p and q are chosen

$p=11, q=19$

step 2: calculate n, such that $n=p*q$

$n=11*19=209$

step 3: calculate z, such that $z=lcm(p-1,q-1)=\frac{(p-1)*(q-1)}{gcd(p-1,q-1)}$

$$z=\frac{10*18}{gcd(10,18)}=\frac{180}{2}=90$$

step 4: select *e*, such that $1<e<z$, where *e* is a prime number and is **coprime** to *z*

so, $1<e<90$ and $gcd(e, 90)=1$

potential *e*'s: 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89

let's choose $e=7$:

$gcd(7,90)=1$

step 5: now solve for *d*, such that $(d*e) mod\, z=1$

this is the same as $d=e^{-1} mod\, z$

using the extended Euclidean algorithm: $d=7^{-1}(mod\, 90)=13$

or, we can try the naive method:

$e=7, z=90$

$7*0=0(mod\, 90)=0$

$7*1=7(mod\, 90)=7$

$7*2=14(mod\, 90)=14$

$\ldots$

$7*12=84(mod\, 90)=84$

$7*13=91(mod\, 90)=1$ ← here we go!

show "modulus" code

step 6: we can now generate the public and private keys:

private key: $K_{priv}=(d, n)=(13, 209)$

public key: $K_{pub}=(e, n)=(7, 209)$

to encrypt message $M=200$, ciphertext *C* is calculated as follows:

$C=M^e(mod\, n)=200^7(mod\, 209)=205$

to decrypt ciphertext $C=205$, plaintext message M is calculated as follows:

$M=C^d(mod\, n)=205^{13}(mod\, 209)=200$

show in python

final notes

finding *p* and *q* solely based on *n* is infeasible for large prime numbers

so, $n=p*q$ is a trapdoor one-way function

show "factoring" code

this problem is considered hard

the security of the entire RSA cryptosystem is based on this "probably" hard problem

the size of the chosen *p* and *q* values dictates the size of the data that can be encrypted

the larger these values, the more difficult it is to break the system

even though *p* and *q* are both large prime numbers of (usually) the same length

ideally, they are not close to one another in value

with really long numbers, this makes sense

coding this?

set a min and max range for picking *p* and *q*

build a list of all primes within this range

to check if a number is prime, we must confirm that it is only evenly divisible by 1 and itself

we can check for potential divisors from 3 through its square root

why the square root?

randomly pick two different primes from the list, *p* and *q*

calculate $n=p*q$

calculate $z=lcm(p-1, q-1)=\frac{(p-1)*(q-1)}{gcd(p-1, q-1)}$

- build a list of all possible values for *e*
  - start with 3 and continue for every odd number through $z-1$
  - or just pick values that are 2^n+1
  - first, check if a potential *e* is prime
  - next, check if it is coprime with *z* (i.e., does $gcd(z,e)=1$?)
- randomly pick a value for *e*
- calculate *d* as the modular inverse of *e* and *z*
  - use the naive method (or perhaps the extended Euclidean algorithm)
- at this point, we basically have K_{priv} and K_{pub}
- encrypting and decrypting a value is a straightforward arithmetic problem

cryptanalysis?

- suppose that we know K_{pub}
- therefore, we know *e* and *n*
- but to “break” this, we need to determine *d*
- and to get *d*, we need *e* and *z*
- luckily, we have *e* (but we don't have *z*)
- to get z, we need *p* and *q* (and those are the prime factors of *n* – unfortunately)
- in the end, it ends up being a difficult factoring problem after all
- specifically, the problem of factoring a large number (*n*) into two prime factors (*p* and *q*)