Mathematics of Computer Science. - M.I.T. opencourseware

Abstract:
this lecture contains the content of how Number theory works in cryptography system like RSA for instance. Euler totient function and Fermat little theorem is the base of the system

For the English:
map, straight forward, given, congruent, modulo, multiplicative, relatively prime(mutually prime, coprime), consequence of, in turn, n times k

Lec 5. NUMBER THEORY II:
encryption : application of number theory => transform message to m' <-> decryption

-Turing code V1
ex) victory => m = 22, 9, 3, 20, 15, 18, 25 + 13(prime - k)
Beforehand = exchange secret prime key = k
Enc : m' = mk / Dec : m'/k = m // it is hard to factor a product of 2 larrrrge primes

gcd(m1', m2') = k // which is not secured

-Turing code V2
Beforehand = exchange a public prime p, a secret prime k
Enc : message as a number m - {0 ~ p-1}, compute m' = rem(mk, p)
Dec : ? [a, b relatively prime iff gcd(a, b) =1 iff sa+tb =1]
DEF: x is congruent to y modulo n => x =_y (mod n) iff n|(x-y) (ex) 31 =_ 16 (mod 5)

DEF: the multiplicative inverse of x mod n is a number x^-1, in {o ~ n-1} => xx^-1 = _1(mod n) ex)2*3 =_ 1(mod 5) => 2 =_ 3^-1(mod 5) // 5*5 =_ (mod 6) => 5 =_5^-1(mod 6)

m' = rem(mk, p) =_ mk (mod p)
if kk^-1 =_ 1 (mod p), then m'k^-1 =_ mkk^-1 (m) =_ m(mod p) // m - {0 ~ p-1}
m = rem(m'k^-1, p) -> Dec

-Know plaintext attack:
know message m and encryption m' = rem(mk, p) // m' =_ mk(mod p)
compute m^-1 => mm^-1 =_ 1(mod p) <= gcd(m, p) = 1
m'm^-1 =_ kmm^-1 =_ k(mod p)
compute k^-1 (mod p)

-Euler totient(phi - total quotient) function
∅(n) denote the number of int in {1 ~ n-1} that are relatively prime to N

Euler's theorem : if gcd(n, k) = 1 => k^∅(n) =_ 1 (mod n)
lemma1 : if gcd(n, k) = 1, then ak =_ bk(mod n) => a =_ b (mod n)
gcd(n, k) = 1 iff k has a multiplicative inverse
lemma2 : suppose that gcd(n, k) = 1
let k1 ~ kr in {1 ~ n-1} denote that rem(k1*k, n) ... rem(kr*k, n) = {k1 ~ kn}
integers relatively prime to n (r = ∅(n))

-Fermat's little theorem :
suppose p is prime and k in {1 ~ p-1} then k^p-1 =_ 1 (mod n)
pf : {1 ~ p-1} are relatively prime to p // because p is prime => ∅(p) = p-1
k^∅(p) =_ 1 (mod p) // k^p-1 =_ 1 (mod k)
k*k^p-2=k^p-1 =_ 1 (mod p) // k^-1 =_ k^p-2 (mod k)

-RSA = public key method
Beforehand : receiver creates public key and secret key
1. generate two distinct primes p and q
2. let N = pq
3. select int e s.t. gcd(e, (p-1)(q-1)) = 1 // public key is the pair consist itself and n (e, n)
4. compute d s.t. de =_1 (mod (p-1)(q-1)), the secret key is the pair(d, n)

Enc : m ' =rem(m^e, n)
Dec : m = rem(m'^d, n)
PF : m' = rem(m^e, n) =_ m^e (mod n) => m'^d =_m^ed (mod n)
for some r, ed = 1 + r(p-1)(q-1) <= gcd(e, (p-1)(q-1)) = 1
so, m'^d =_ m^ed =_ mm^r(p-1)(q-1) (mod n) // n = pq => m'^d =_ mm^r(p-1)(q-1) (mod p or q)
if m !=_ 0(mod p or q) then m^(p-1 or q-1) =_ (mod p or q)

m'^d =_ m (mod p) => p|(m'^d - m) => pq|(m'^d - m)
p or q both can be alternative
m'^d =_ m (mod n) // m = rem(m'^d, n) => dec rule equation truly holds

[from:to:step]