# Digital_Signature

```
# Digital Signature
```

## El Gamal Signature scheme

```
p = 859 # choose a prime
```

```
a = mod(2,p) # make all compuatations with a return answer with mod
p
```

```
A = 800
B = a^A;B # this is the pubic key
```

```
17
```

```
# create a message and use the
m= 499
k = mod(199,p-1)
gcd(k,p-1) # modulus of k and p-1
```

```
1
```

```
# Compuatation  of the 2 values to be sent to Bob.  Bob sends
(m,r,s) to Alice
# k is defined to be of type modulo p - 1
# r is already of type modulo p too, the “lift()” operation “lifts”
r out of that integer ring.
r = a^k;
print(r)
s = (m-A*r.lift())/k;s
```

```
67
755
```

```
lift?
```

**File:** /Applications/SageMath-7.6.app/Contents/Resources/sage/local/lib/python2.7/site-packages/sage/misc/functional.py

**Type:** <type ‘function’>

**Definition:** lift(x)

**Docstring:**

Lift an object of a quotient ring $R/I$ to $R$.

EXAMPLES: We lift an integer modulo $3$.

```
sage: Mod(2,3).lift()
2
```

We lift an element of a quotient polynomial ring.

```
sage: R.<x> = QQ['x']
sage: S.<xmod> = R.quo(x^2 + 1)
sage: lift(xmod-7)
x - 7
```

```
# example to demonstrate lift command
lift_example=Mod(2,3).lift()
print(parent(lift_example)) # a is integer ring
lift_example =Mod(2,3)
parent(lift_example)  # a is Ring is Integer  modulo 3
```

```
Integer Ring
Ring of integers modulo 3
```

Message verification

```
# m,r,s has be sent to Bob, to verify the message Bob computes
v1 = a^m;v1 # he computes a^m and this should be equal to B^r * r^s
as shown below
```

```
517
```

```
 v2 = B^r*r^s;v2
```

```
517
```

```
v1==v2 # v1 == v2 returns true
```

```
True
```

Signature Forgery.

In order for Elgamal signature to be safe it is important that r<p. Otherwise a forgery case can be exhibited as shown below:

```
# Note that we start with a message for which the gcd of m and p-1
equals 1
# And we attempt to forge a message without knowing the private key
A
gcd(m,p-1)
```

```
1
```

```
m1 = mod(600,p-1) # assume we want a forged message m1. m was 800
above
m1
```

```
600
```

```
u = (m1/m).lift(); u # compute this. This is implicitly mod p-1
```

```
744
```

```
s1 = s*u; s1 # compute new s1 to replace s.
```

```
588
```

```
r = r.lift() # get the original r outside the ring
```

```
# Compute the new r
r1 = crt([r*u,r],[p-1,p]); r1
```

```
14670
```

```
#(r1, s1) = (14670, 780) is the forged signature for m' = 600
# This then returns a valid check.
# If we checked r1<s1 we would have avoided this attack
a^m1==B^r1*r1^s1
```

```
True
```

## Digital Signature Scheme

```
# choose two primes

p = 1031
q = 103


print(is_prime(p))
is_prime(q)
```

```
True
48
```

```
# Choose a value less than p
a = mod(14,p)
# this value should satisfy the criteria that the
# equation below is not equals to one,
# Assign this to g
g = a^((p-1)/q); g
```

```
320
```

```
# Chooose private key A
A = 70
# Compute the public key B
# public key is p,g,b,p,q
B = g^A; B
```

```
48
```

```
# Generate a message to send
m = 500 # message
k = 25   # generate r as in Elgamal
```

```
r = mod(g^k,q); r
```

```
95
```

```
# Generate s too
# m,r and s are to be sent to Bob as in Elgamal
s = (m+A*r)/k; s
```

```
80
```

## Verification

```
# To verify compute s * inverse(s)
x = m/s; x
```

```
32
```

```
# Compute this r * inverse(s)
y = r/s; y
```

```
72
```

```
# Compute v as below and check it is equal to r
v = mod(g^x*B^y,q)
```

```
v==r
```

```
True
```

```

```