Eric Gonzalez
Z23411215

Cryptographic Engineering: Assignment 1 Prelab

1. Compute the residue of $a = 2^{30} - 18 = 1073741806 = \{0x3FFFFFEE\}$ over the following numbers **using the method you learned in class.** Show your work. Then verify your results using SageMath. Show all results in Hexadecimal.

   (a) $p_1 = 2^{17} - 1 = \{0x1FFFF\}$ (Mersenne prime)

   (b) $p_2 = 2^{26} - 5 = \{0x3FFFFFB\}$ (Pseudo-mersenne prime)

   (c) $b = 2^{16} = \{0x10000\}$ (Not a prime number)

```
 Question 1.
 Section A. Mersenne prime
a=       1073741806
a in hex=        0x3fffffee
a mod p1= 8174
a_low= 0x1ffee
a_high= 0x1fff
r:       0x21fed
Is r>=p1 true?
True
result of r= 0x1fee
r= 8174

Section B. Psuedo Marsenne prime
a=       1073741806
a in hex=        0x3fffffee
a mod p3= 62
a_low= 0x3ffffee
a_high= 0xf
r:       0x4000039
Is r>=p3 true?
True
result of r= 0x3e
r= 62
r= 62

Section C. Not a prime
b= 65536
b in hex=        0x10000
(In hex)amb=a & (b-1)= 0xffee
(In hex) a mod b= 0xffee
---------------------------------------
```

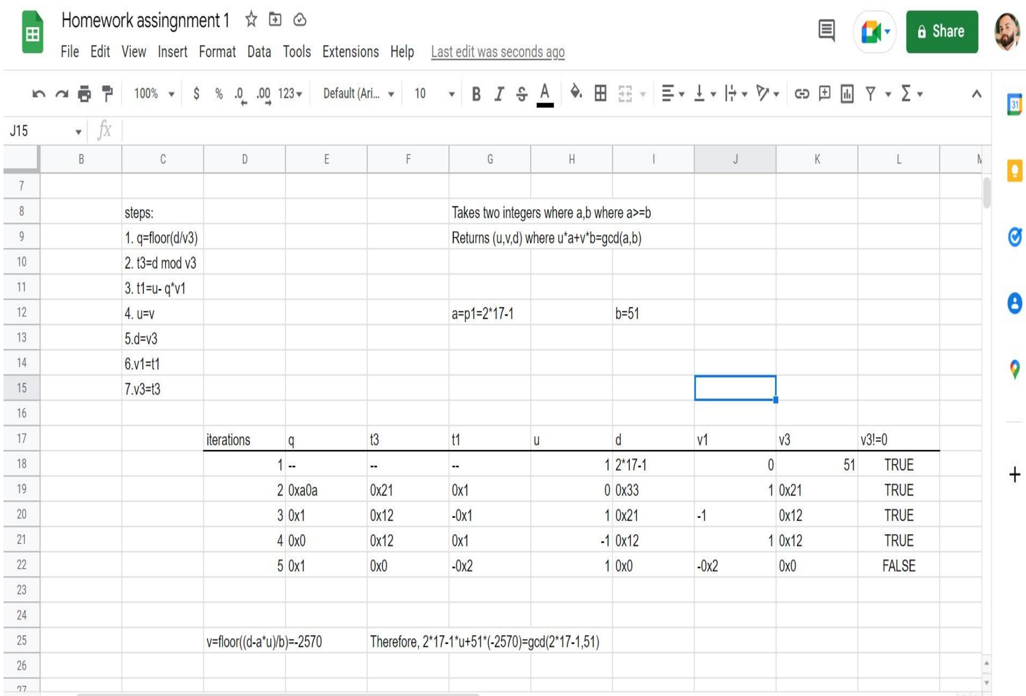
```
from math import floor




print(" Question 1. ")
print(" Section A. Mersenne prime ")
#A. Mersenne prime


a=2**30-18
p1=2**17-1
print("a= \t",a)
print("a in hex= \t",hex(a))
print("a mod p1=",a%p1)

#1.)split the number stored in a variable by the exponent of the mersenne prime
a_low= a &  (2**17-1)
a_high= a >> 17

print('a_low=',hex(a_low))
print("a_high=",hex(a_high))

#2.) calculate r where r =a_low+a_high
r=a_low+a_high

print("r: \t",hex(r))

#3. check if r>=p1
print("Is r>=p1 true?")
print(r>=p1)
if r>=p1:
  r-=p1
print("result of r=",hex(r))
assert(r==(a%p1)) #no error means true

print("r=",r)

print("\nSection B. Psuedo Marsenne prime")
a=2**30-18
p3=2**26-5
print("a= \t",a)
print("a in hex= \t",hex(a))
print("a mod p3=",a%p3)
#1.)split the number stored in a variable by the exponent of the mersenne prime
a_l= a &  (2**26-1)
a_h= a >> 26

print('a_low=',hex(a_l))
print("a_high=",hex(a_h))

#2.) calculate r where r =a_low+a_high
r=a_l+(5*a_h)

print("r: \t",hex(r))

#3. check if r>=p3
print("Is r>=p3 true?")
print(r>=p3)
if r>=p3:
  r-=p3
print("result of r=",hex(r))
```

```
print("result of r=",hex(r))
print("r=",r)
assert(r==(a%p3)) #no error means true

print("r=",r)

print("\nSection C. Not a prime")
#C.Not a prime
# Because we have a power of 2 we can do a very cheap, inexpensive modular reduction
b=2**16
print("b=",2**16)
print("b in hex= \t",hex(b))
amb=a & (b-1)
print("(In hex)amb=a & (b-1)=",hex(amb))
print("(In hex) a mod b=",hex(a%b))
print("-------------------------------------")
```

2. In class, you learned two methods to compute the multiplicative inverse of an operand over a finite field; Fermat's Little Theorem (FLT) and Extended Euclidean Algorithm (EEA). The finite field is constructed over $p = 2^{17} - 1$ (Mersenne prime from the previous exercise). Compute the multiplicative inverse of $a = 51$ over F_p using the below methods. Show your work. Then verify your results using SageMath. Show all results in Hexadecimal.

(a) Fermat's Little Theorem (FLT)

(b) Extended Euclidean Algorithm (EEA)


Homework assingnment 1

File Edit View Insert Format Data Tools Extensions Help Last edit was seconds ago

J15

	B	C	D	E	F	G	H	I	J	K	L
7											
8		steps:				Takes two integers where a,b where a>=b					
9		1. q=floor(d/v3)				Returns (u,v,d) where u*a+v*b=gcd(a,b)					
10		2. t3=d mod v3									
11		3. t1=u- q*v1									
12		4. u=v				a=p1=2*17-1		b=51			
13		5.d=v3									
14		6.v1=t1									
15		7.v3=t3									
16											
17			iterations	q	t3	t1	u	d	v1	v3	v3!=0
18			1	--	--	--	1	2*17-1	0	51	TRUE
19			2	0xa0a	0x21	0x1	0	0x33	1	0x21	TRUE
20			3	0x1	0x12	-0x1	1	0x21	-1	0x12	TRUE
21			4	0x0	0x12	0x1	-1	0x12	1	0x12	TRUE
22			5	0x1	0x0	-0x2	1	0x0	-0x2	0x0	FALSE
23											
24											
25			v=floor((d-a*u)/b)=-2570		Therefore, 2*17-1*u+51*(-2570)=gcd(2*17-1,51)						
26											

```
---------------------------------------

Question 2
Section A. FLT
Compute a^-1 mod p
Computation of 51^-1 mod 2^17-1= 0x1f5f5
51^(p-2) mod 2^17-1=  0x1f5f5

Question 2
Section B. EEA
The GCD is 1
x = 1, y = -2570
d= 18
v3= 18
q= 0x1
int q = 1
t3= 0x0
0
t1= -0x2
t1= -2
v1= -1
```

```
# Question 2
# A.Fermat's little theorem
print(" \nQuestion 2 ")
print("Section A. FLT ")

# prove thatr a^-1 mod p= a^(p-2) mod p
# steps:
# 1. calculate p-2
# 2. raise a^ (p-2)
# 3. calc a^(p-2) mod p= a^-1 mod p

a2=51
p2=p1-2
ap2=a2**p2
flt_r= ap2 % p1
print("Compute a^-1 mod p")
print("Computation of 51^-1 mod 2^17-1=",hex(pow(51,-1,p1)))
print("51^(p-2) mod 2^17-1= ",hex(flt_r) )



#B. Euclidean Algorithm
print(" \nQuestion 2 ")
print("Section B. EEA ")
# Python program for the extended Euclidean algorithm
def extended_gcd(a, b):
    if a == 0:
        return b, 0, 1
    else:
        gcd, x, y = extended_gcd(b % a, a)
        return gcd, y - (b // a) * x, x


if __name__ == '__main__':

    gcd, x, y = extended_gcd(p1, a2)
    print('The GCD is', gcd)
    print(f'x = {x}, y = {y}')


v3=int(0x12)
d=int(0x12)
```

3. In Exercise 2, you applied two methods to compute the multiplicative inverse of an operand over a finite field. Answer the following questions related to these 2 methods:

(a) How many loop iterations does it take to compute the output for each method for *a* = 51 (exercise 2)?

- It takes one loop iteration to calculate the multiplicative inverse with FLT and 5 total iterations for EEA

(b) Which method is faster in general? Why?

- FLT is very much faster than EEA and is evident from the fact that it runs without loops while EEA needs to do multiple iterations and it grows larger with larger integers

(c) Which method is constant time (aka the number of iterations is the same independent of the input used)? Why?

- FLT is constant time because it is the same operation no matter the input size.