Implementation of finite field arithmetic

Sarbajit Ghosh

January 11, 2021

1 Modular reduction in finite field

Let say we want to compute $r\,(modp)$ where $p = 2^{127} - 1$. Now,

$$r \equiv r_1 2^{127} + r_0$$
$$\equiv r_0 + r_1 (mod 2^{127} - 1)$$

2 Sample Run

Our software specification: Our program is written for handling handling field multiplication in a prime field where $p = 2^{127} - 1$. We have implemented addition, subtraction, multiplication, and inverse under the finite field. Modular reduction is implemented in both the function.We have implemented the the inverse function **without using any if, else condition.**

Input: Two integers, in the range $\{0, 1, ..., 2^{127} - 2\}$ in hexadecimal.

Output:

1. Addition, subtraction, multiplication of two field elements.

2. Also inverse of both the elements in the field.

```
*******************************************************
Program for calculation of finite field arithmatic
Over the prime field of size p=pow(2,127)-1
Please enter two field element in the range
{0x0, 0x1, ..., p-1=0x7ffffffffffffffffffffffffffffffe}
*******************************************************
Enter number one:89a2359acdb123abdee977
Enter number two:4589073677236889
Field addition of two elements over F_p where p=pow(2,127)-1
000000000089a235        e056b85a23025200

Field multiplication of two elements over F_p where p=pow(2,127)-1
222104ccfc7cc37b        91e5e803c5b30d77

Inverse of first element
1fdf8d92240dd4ce        e7aeefb9c36aea67

Inverse of second element
085acd99dce6d7f6        d1878098341038b1

```

```
Field minus of two elements over F_p where p=pow(2,127)-1
Note we have implemented as -a=p-a (mod p)
000000000089a235          5544a9ed34bb80ee
```

And to find inverse of elements we have written a following code in Python. We have used square and multiply algorithm for finding inverse both in C and Python.

```python
a=0x89a2359acdb123abdee977
b=0x4589073677236889
inva=a #step 1
invb=b
p=0x7fffffffffffffffffffffffffffffff # 2**127 -1 in hex
# binary value of p-2 '0b1111111111111111111111111111111111111111111111111111
# 111111111111111111111111111111111111111111111111111111111111111111111111101'
for i in range(124):
    inva=(inva*inva)%p
    inva=((inva*a)%p)
    invb=(invb*invb)%p
    invb=((invb*b)%p)

inva=((inva*inva)%p) # for 126 th bit
inva=((inva*inva*a)%p) # for 127 th bit

invb=((invb*invb)%p) # for 126 th bit
invb=((invb*invb*b)%p) # for 127 th bit

print(hex(inva))
print(hex(invb))
```

Verification of output using python

```
>>> p=2**127-1
>>> hex((0x89a2359acdb123abdee977+0x4589073677236889)%p)
'0x89a235e056b85a23025200'
>>> hex((0x89a2359acdb123abdee977-0x4589073677236889)%p)
'0x89a2355544a9ed34bb80ee'
>>> hex((0x89a2359acdb123abdee977*0x4589073677236889)%p)
'0x222104ccfc7cc37b91e5e803c5b30d77'

$ python3 inverse.py
0x1fdf8d92240dd4cee7aeefb9c36aea67
0x85acd99dce6d7f6d1878098341038b1
```

Screen shot:

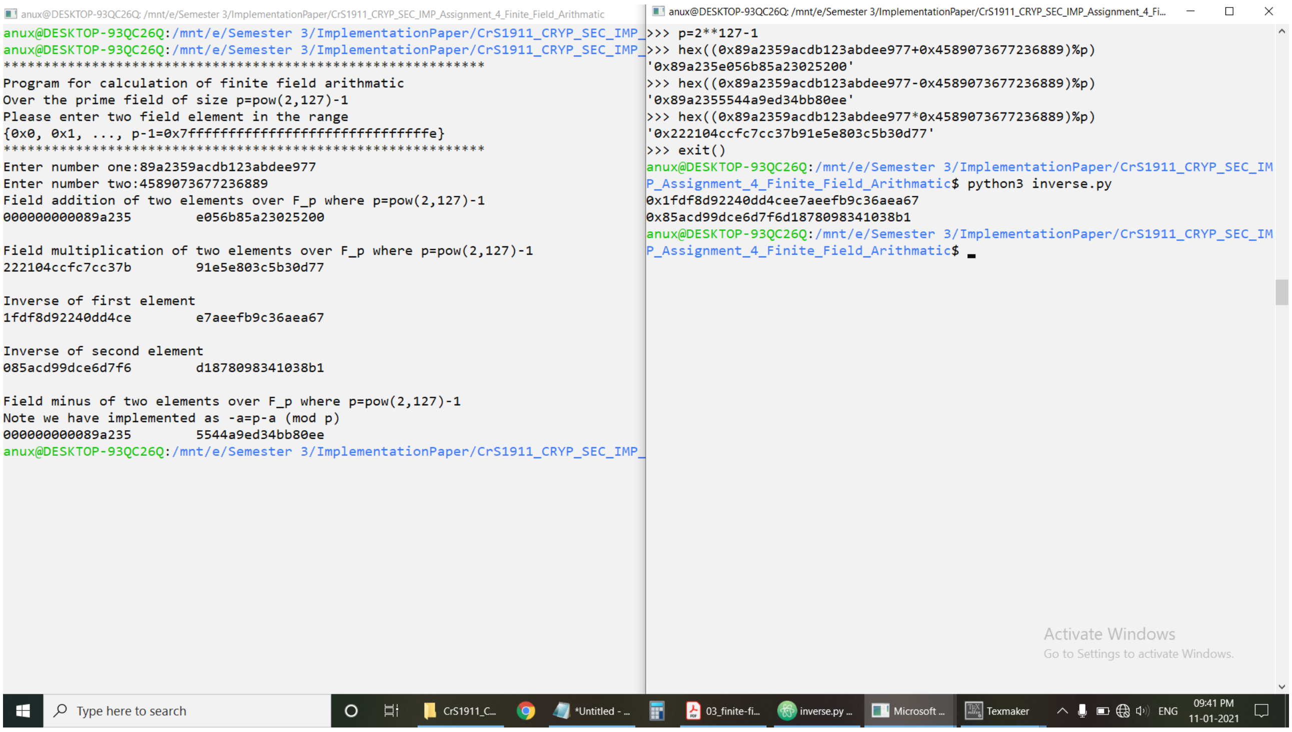

Figure 1: Demo of program & verification of outputs using python3

3 System information:

We have used ubuntu 18.04 under windows subsystem of Linux.

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):             1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 142
Model name:            Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
Stepping:              10
CPU MHz:               1801.000
CPU max MHz:           1801.0000
BogoMIPS:              3602.00
Virtualization:        VT-x
Hypervisor vendor:     Windows Subsystem for Linux
Virtualization type:   container
```