Galois field 2^4

Let's take a quick look at $\mathrm{GF}(2^4)$ before moving on to the main event ($\mathrm{GF}(2^8)$).

These are the polynomial equivalents of the binary numbers with 4 digits, all except 0000. There are 15 elements.

Degree 0:

$$1$$

Degree 1:

$$x \quad x+1$$

Degree 2:

$$x^2 \quad x^2+1 \quad x^2+x \quad x^2+x+1$$

The new ones are degree 3. Write them as binary equivalents:

$$1000 \quad 1001 \quad 1010 \quad 1011$$

$$1100 \quad 1101 \quad 1110 \quad 1111$$

irreducible polynomial

We need an irreducible polynomial. In $\mathrm{GF}(2^3)$ we could choose from x^3+x+1 and x^3+x^3+1.

Now we need a polynomial of degree 4, which may be obtained either by multiplying a degree 3 times a degree 1, or by multiplying two of degree 2.

Start with the degree 3's and multiply them by either 10 or 11:

```
10000   10010   10100   10110
11000   11010   11100   11110
11000   11011   11110   11101
10100   10111   10010   10001
```

Degree 2's as binary equivalents:

$$100 \quad 101 \quad 110 \quad 111$$

And all multiplied together:

```
10000   10100   11000   11100
10001   11110   11011
10100   10011
10101
```

Write them all without the leading 1 (grouping by the next digit) to see the pattern:

```
10 + 000 001 010 011 100 101 110 111
11 + 000 --- 010 011 100 --- 110  ---
```

I find only 13, so 3 are missing. These are

```
11001 11101 11111
```

We shall choose $11001 = x^4 + x^3 + 1$.

generator

I tried **0x03** and found it is not a generator for this field. Rather than guess again, I wrote a short script that calls the gmultiply routine. It's modified to use the irreducible polynomial from above (11001 = decimal 25) and do the mod operation if $n > 15$. I'll put the code at the end

The output is

```
 2   4   8   9  11  15   7  14   5  10  13   3   6  12   1   2   4
 3   5  15   8   1   3   5  15   8   1   3   5  15   8   1   3   5
 4   9  15  14  10   3  12   2   8  11   7   5  13   6   1   4   9
  5   8   3  15   1   5   8   3  15   1   5   8   3  15   1   5   8
 ..
 15   3   8   5   1  15   3   8   5   1  15   3   8   5   1  15   3
```

Unexpectedly, 2 and 4 are generators but 3 and 5 are not. So

```
 0010
 0010
 ----
 0100 => 0100
 0010
 ----
 1000 => 1000
 0010
 ----
10000
11001 mod
-----
 1001 => 1001
 0010
```

```
 ----
10010
11001 mod
-----
 1011 => 1011
..
```

I won't show the whole thing.

I got

```
0100 1000 1001 1011 1111 0111 1110 0101
1010 1101 0011 0110 1100 0001 0010
```

and compare that with

```
 2  4  8  9 11 15  7 14  5 10 13  3  6 12  1  2
```

That's a match. With the powers, we can compute a table of logarithms.

Here they are: the elements are in the first row and their logarithms below.

```
 2  4  8  9 11 15  7 14  5 10 13  3  6 12  1  2
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
```

According to this, 7 and 14 should be multiplicative inverses because their logarithms add to 15, whose anti-logarithm is 1.

Try it:

```
    0111 =  7
    1110 = 14
    ----
    111
   111
  111
```

```
  ----
 101010
 11001 mod
 -----
  11000
  11001 mod
  -----
      1
```

This is pretty tedious. I wrote code to carry out the procedure, and check the above multiplicative inverses. They are all correct.

The last thing is to try an example of the extended Euclidean algorithm. Find the multiplicative inverse of 5.

```
    a      b      q       bq      r
11001    101    110    11110    111
  101    111      1      111     10
  111     10     11      110      1
-----
111 = 11001 - (110)101
 10 = 101 - (1)111
  1 = 111 - (11)10
    = 111 - (11)[101 - (1)111]
    = (10)111 - (11)101
    = (10)[11001 - (110)0101] - (11)101
    = (1100)101 + (11)101
    = (1111)101
    = (15)(5)
```

which is correct.

Here is the code I used. First is the current version of gmultiply:

```
def gmod(n,N):
```

```
    D = { 3:19, 4:25, 8:283 }
    P = D[N]
    b = len(bin(P))
    while True:
        if n < N**2:
            return n
        n = n ^ (P << (len(bin(n)) - b))

def gmultiply(a,b,N=8):  # default N = 8
    s = bin(b)[2:][::-1]
    r = 0
    for c in s:
        if c == '1':
            r = r ^ a
        a = a << 1
    return gmod(r,N)
```