Lab 5: Modular Arithmetics

50.042 FCS

Hand-out: 15 Oct
Hand-in: 11:59pm, 22 Oct

1 Objective

By the end of this lab, you should be able to:

- Implement a class to do Number Theory computation in GF(2^n)
- Implement Byte Substitution layer of AES encryption.

2 Part I: Algebraic Structures

- Create a class `Polynomial2` where the coefficient is in GF(2).
  - The constructor takes in a list containing the values of the coefficient of the polynomial starting from the lowest power to the highest power. For example:

    $$x^5 + x^2 + x \tag{1}$$

    is created as follows:

    ```
    x = Polynomial([0,1,1,0,0,1])
    ```
  - Write the method `add(p2)` and `sub(p2)` to perform Addition and Subtraction. To do addition, you simply XOR the coefficient of the same powers. Addition and subtraction operations on polynomials are the same operation. The method should return a Polynomial2 object.
  - Overwrite the method `mul(p2,modp)` to perform Multiplication. The method takes in another polynomial `p2` to be multiplied and the modulus polynomial `modp` and returns a `Polynomial2` object. The default value for the modulus polynomial is `None`. If `modp` is given, the multiplication is done with modulus with respect to `modp`. To do modulus multiplication, we first generate the partial results. For example, instead of finding the result of $(x^2 \oplus P_2)$, the program finds the result of $(x \oplus (x \oplus P_2))$.
    - Generate partial results as follows:
      - If the most significant bit of the previous result is 0, just shift the previous result one bit to the left. This is basically multiplying the polynomial with an x.
      - If the most significant bit of the previous result is 1, we need to do the following: 1) shift it one bit to the left, and 2) exclusive-or it with the modulus without the most significant bit. This basically means that if we multiplying the polynomial with an x and the result has the same degree as the modulus, we need to reduce the result with the irreducible polynomial or the modulus.

* According to the coefficients in the polynomial, add up the partial results to get the final result.
* If the multiplication is done without any modulus, simply generate partial results by shifting the previous partial result one bit to the left, then add up the partial results to get the final result.
* An example written in Polynomial is shown here: $P_1 = x^5 + x^2 + x$, $P_2 = x^7 + x^4 + x^3 + x^2 + x$, in GF($2^8$) with irreducible polynomial ($x^8 + x^4 + x^3 + x + 1$).

Powers	Operation	New Result	Reduction
$x^0 \otimes P_2$		$x^7 + x^4 + x^3 + x^2 + x$	No
$x^1 \otimes P_2$	$x \otimes (x^7 + x^4 + x^3 + x^2 + x)$	$x^5 + x^2 + x + 1$	Yes
$x^2 \otimes P_2$	$x \otimes (x^5 + x^2 + x + 1)$	$x^6 + x^3 + x^2 + x$	No
$x^3 \otimes P_2$	$x \otimes (x^6 + x^3 + x^2 + x)$	$x^7 + x^4 + x^3 + x^2$	No
$x^4 \otimes P_2$	$x \otimes (x^7 + x^4 + x^3 + x^2)$	$x^5 + x + 1$	Yes
$x^5 \otimes P_2$	$x \otimes (x^5 + x + 1)$	$x^6 + x^2 + x$	No

$P_1 \otimes P_2 = (x^6 + x^2 + x) + (x^6 + x^3 + x^2 + x) + (x^5 + x^2 + x + 1) = x^5 + x^3 + x^2 + x + 1$

– Write the method `div` to perform Division. Use the Euclidian Division algorithm
  * `http://en.wikipedia.org/wiki/Polynomial_greatest_common_divisor#Euclidean_division`

  In this algorithm, deg() stands for the degree of its argument, and lc() stands for the leading coefficient, or the coefficient of the highest degree of the variable.

```
Euclidean division
Input: a and nonzero b (two polynomials in the variable x);
Output: q, the quotient and r, the remainder;
Begin
q:=0; r:=a;
d:=deg(b); c:=lc(b);
while deg(r) >= d do

   s:=(lc(r)/c)x^{deg(r)-d};
   q:=q+s;
   r:=r-sb;

end do;
return (q, r);
end.
```

  The method should return two polynomials, one for the quotient and the other for the remainder.

– Overwrite the `__str__` method so that it can print the polynomial. For example,

```
x^5+x^3+x^1+x^0
```

– Write a method `getInt()` that returns the equivalent integer value of the polynomial represented.

- Create a class with the name `GF2N` that implements Galois Field. The object belongs to a finite field with the number of element equals to 2^n.

- The constructor takes in `x`, `n`, and `ip`. Where `x` is the number to be represented in Galois Field, and `n` is the power in 2^n, and `ip` is the irreducible polynomial. The default for n is 8. The constructor stores the number as a Polynomial of degree =n-1. The default irreducible polynomial is $P(x) = x^8 + x^4 + x^3 + x + 1$.
- Implement methods to do addition, subtraction, multiplication, and division.
- Implement a method `getPolynomial2()` that returns the polynomial representation of the number.
- Implement a method `getInt()` that returns the integer value.
- Overwrite `__str__` method so that it can print the integer value.

- Test Cases are provided at the end of the handout.
- **Hand-in**:
  - Create a table for addition and multiplication for GF(2^4), using ($x^4 + x^3 + 1$) as the modulus.
  - You should be able to show the workings of addition and multiplication of GF(2^n) by hand.

3 Part II: AES Byte Substitution Layer (optional)

- Inversion in GF(2^8) is the core operation of the Byte Substituion transformation which contains the AES S-Boxes. In this exercise, you need to implement multiplicative inverse as a method in GF2N class using the Extended Euclidean Algorithm.
  - `http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm`

  The algorithm can be written as follows. Assume that the Polynomial we want to find the inverse is `b` and the modulus polynomial is `n`.

```
r1 <- n; r2 <- b;
t1 <- 0; t2 <- 1;
while (r2 > 0)
{
  q <- r1/r2;

  r <- r1 - q x r2;
  r1 <- r2; r2 <- r;

  t <- t1 - q x t2;
  t1 <- t2; t2 <- t;
}
if (r1 = 1) then return t1;
```

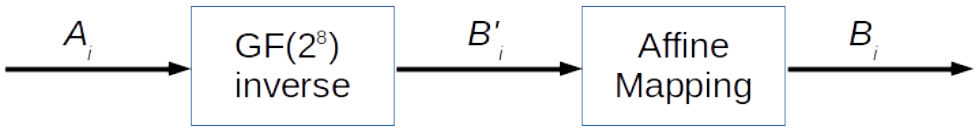
- Implement affine mapping method in GF2N class. Affine mapping is described as the following transformation.

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \text{ mod } 2 \tag{2}$$

  - Implement Byte Substitution layer of AES S-box as follows:

$$S(A_i) = B_i \tag{3}$$

    where B_i is the output of `affineMap(Bi_prime)`, and `Bi_prime` is the output of `Ai.mulInv()`. `Ai` is in GF(2^8).



- **Hand-in**:
  - Generate Table 4.3 on page 101 of *Understanding Cryptography*:
    * `https://app.box.com/s/n8pd9mgza1avb2sg4pln`
  - You should be able to show the working of finding the inverse multiplication by hand.

4 Test Cases:

```
Test 1
======
p1=x^5+x^2+x
p2=x^3+x^2+1
p3= p1+p2 =  x^5+x^3+x^1+x^0

Test 2
======
p4=x^7+x^4+x^3+x^2+x
modp=x^8+x^7+x^5+x^4+1
p5=p1*p4 mod (modp)= x^7+x^6+x^4+x^3

Test 3
======
p6=x^12+x^7+x^2
p7=x^8+x^4+x^3+x+1
```

```
q for p6/p7= x^4+x^0
r for p6/p7= x^5+x^3+x^2+x^1+x^0

Test 4
======
g1 =  x^6+x^5+x^2
g2 =  x^2+x^0
g1+g2 =  97

Test 5
======
irreducible polynomial x^4+x^1+x^0
g4 =  x^3+x^2+x^0
g5 =  x^2+x^1
g4 x g5 =  x^3

Test 6
======
g7 =  x^12+x^7+x^2
g8 =  x^8+x^4+x^3+x^1+x^0
g7/g8 =
q =  x^4+x^0
r =  x^5+x^3+x^2+x^1+x^0

Test 7
======
irreducible polynomial x^4+x^1+x^0
g9 =  x^2+x^0
inverse of g9 = x^3+x^1+x^0

Test 8
======
irreducible polynomial x^8+x^4+x^3+x^1+x^0
g10 = 0xc2
inverse of g10 = g11 = 0x2f
affine map of g11 = 0x25
```