**CS-591 Project Report**

Autumn Semester 2020

# Development of cryptanalytic tools for Post-quantum code-based cryptosystems

*Submitted in partial fulfillment of the requirements for the award of degree of*

**Bachelor of Technology**
**in**
**Computer Science and Engineering**

Submitted by

| Roll No | Name of Student |
|---|---|
| 39/CSE/18078/389 | Sayan Medya |
| 39/CSE/18079/390 | Sayon Seal |
| 39/CSE/18085/395 | Shivam Gupta |

Under the guidance of

**Dr. Bhaskar Biswas**

*Department of Computer Science and Engineering*

**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY**

# Table of Contents

# 1. Introduction

## 1.1 Purpose

Code based cryptography is the oldest member of the post quantum cryptography family. The McEliece cryptosystem, dating back to 1978, relies on the hardness of decoding in a random binary (Goppa) code.

During our project, we shall implement the necessary modules for code based cryptosystems in Python. Our module will include a Polynomial Library and implementation of the Galois Field in order to support the complete implementation of McEliece Cryptosystem. Our library will support all basic operations of addition, multiplication, division between different polynomials and finding inverses using the extended euclidean algorithm with respect to an irreducible polynomial.

## 1.2 References

- https://www.rocq.inria.fr/secret/CBCrypto/index.php
- https://en.wikipedia.org/wiki/Finite_field
- Cryptography and Secure Communications - Richard E. Blahut

# 2. Acknowledgement

Firstly, we would like to thank our mentor professor Dr. Bhaskar Biswas sir for giving us the opportunity to work in a project that we were looking forward to do and then giving us the creative freedom to design and work on it how we wished to but simultaneously giving us his constructive criticism and advice on what problems we might face along the way specially his advice on how to plan out a timeline or what we could work on really helped us complete this project in a timely manner. This project would not be possible without his constant support throughout.

# 3. Requirements

## 3.1 Mathematical Background

In order to implement the polynomial libraries required for McEliece cryptosystem, we first need to understand some basic mathematical concepts of Galois Field or Finite Field.

**Definition of Finite Field -** A finite field is a finite set which is a field; this means that multiplication, addition, subtraction and division (excluding division by zero) are defined and satisfy the rules of arithmetic known as the field axioms.

The number of elements of a finite field is called its *order* or, sometimes, its *size*. A finite field of order $q$ exists if and only if the order $q$ is a prime power $p^k$ (where $p$ is a prime number and $k$ is a positive integer). In a field of order $p^k$, adding $p$ copies of any element always results in zero; that is, the characteristic of the field is $p$.

$GF(p^q)$ is defined as the set of polynomials of degree at most $q-1$ and all coefficients of these polynomials are in the integer field $Z(p)/p$

In our implementation we are only interested in the case when $p = 2$. We can represent the polynomials of this $GF(2^n)$ as binary strings.

**Addition in $GF(2^n)$:** The addition of two polynomials is the same as normal polynomial addition where all the coefficients are added in the modulo 2 field.

**Multiplication in $GF(2^n)$:** In order to define multiplication of two polynomials we need an irreducible polynomial of degree at most $n$ to bind the result in the GF. An irreducible polynomial is a polynomial which cannot be factored into two or more polynomials. We modulo the multiplication result using the irreducible polynomial of the GF.

**Finding inverse $GF(2^n)$:** Inverse of a polynomial $p$ in $GF(2^n)$ with respect to an irreducible polynomial $IP$ is a polynomial q ∈ $GF(2^n)$ such that,

$$p * q \equiv 1 \ (mod\, IP)$$

We can find the inverse of a polynomial using the Extended Euclidean algorithm or Bezout's Identity method.

## 3.2 Modules

**BitArray Module** : We used the bitarray module to use bitarray data structure to store coefficients of polynomial in the form of '1' and '0' bits.

**Ex** : 1101 : $1+x+x^3$

# 4. Implementation

## 4.1 Description

We have constructed two modules i.e. poly.py and gf.py.

**Poly.py :**

We have implemented a polynomial class inherited from the bitarray class to perform basic functionalities like addition, multiplication, division (finding quotient and remainder) and some utility functions (i.e calculate degree of the polynomial, right / left shifting a polynomial and remove trailing zeros etc). We have used operator overloading concept to increase the readability of the code and make it user friendly.

**Addition:** In a GF($2^n$) field addition and subtraction yield the same result and is equal to the bitwise xor operation of the bits of representing the input polynomials

**Multiplication:** We have used Shift-and-Add Multiplication algorithm for multiplying two binary polynomials.

**Division:** We have used long division process for producing both the quotient and remainder for the division. We have also included the case where the divisor may be a zero polynomial and then our program throws an exception.

**Gf.py :**

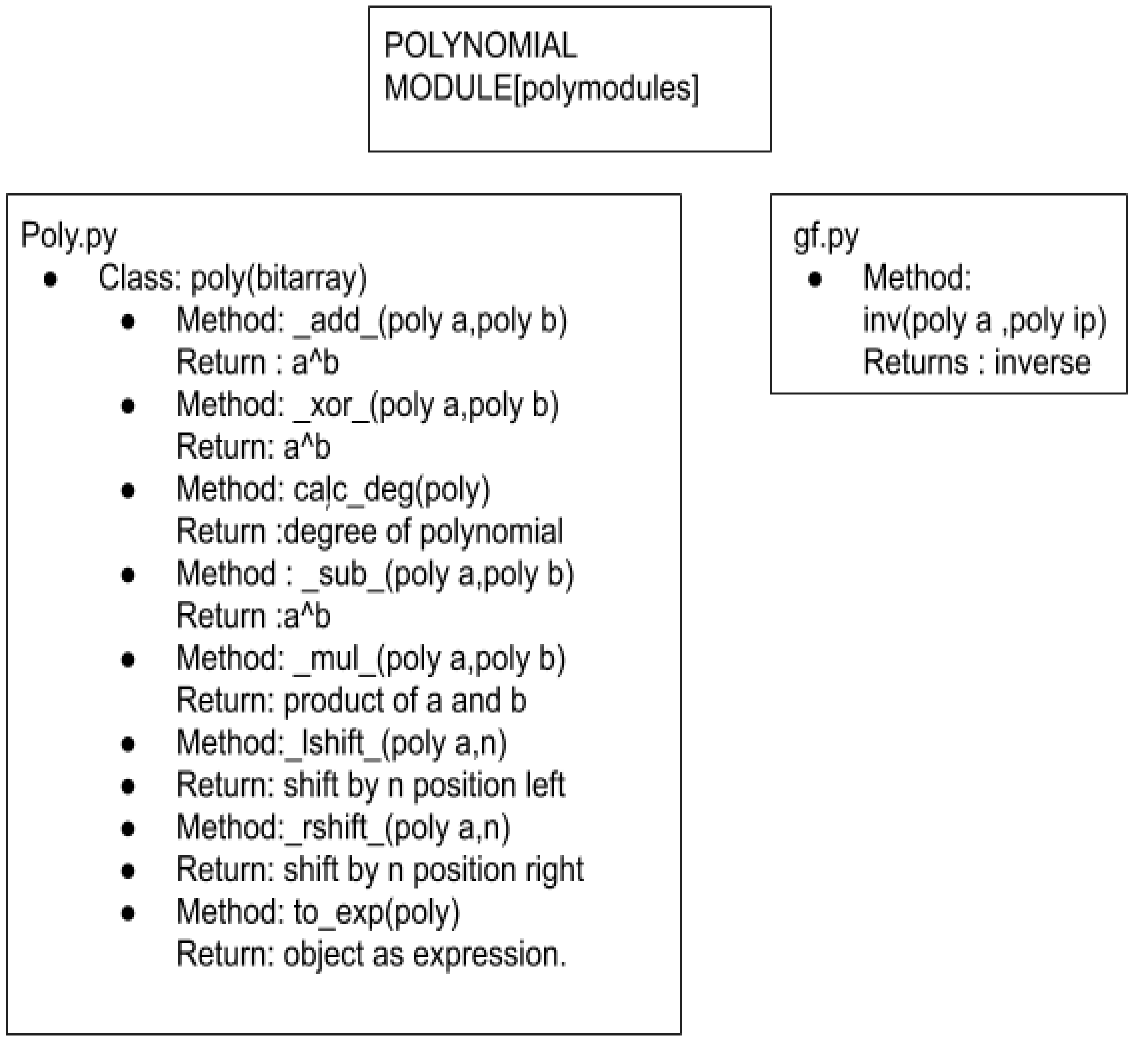
This module includes necessary functions for operating in the Galois Field which includes finding inverse in the GF, finding GCD of two polynomials and power of a polynomial.

**<u>Inverse</u> :** We have used 'Extended Euclidean Algorithm' to find the inverse of a polynomial within the Galois Field($2^n$) under some irreducible polynomial(IP) of degree at most n.

**<u>Power</u> <u>:</u>** This is a function to calculate power of polynomials within the Galois Field($2^n$) under some irreducible polynomial(IP) of degree at most n with optimal complexity of O(log k). This function uses binary exponential to reduce the time complexity.

**<u>GCD</u> :** This is a simple implementation of 'Euclid's Algorithm' to find gcd of two polynomials.

## 4.2 Flow Chart of the Implementation



POLYNOMIAL
MODULE[polymodules]

Poly.py

- Class: poly(bitarray)
  - Method: _add_(poly a,poly b)
    Return : a^b
  - Method: _xor_(poly a,poly b)
    Return: a^b
  - Method: calc_deg(poly)
    Return :degree of polynomial
  - Method : _sub_(poly a,poly b)
    Return :a^b
  - Method: _mul_(poly a,poly b)
    Return: product of a and b
  - Method:_lshift_(poly a,n)
  - Return: shift by n position left
  - Method:_rshift_(poly a,n)
  - Return: shift by n position right
  - Method: to_exp(poly)
    Return: object as expression.

gf.py

- Method:
  inv(poly a ,poly ip)
  Returns : inverse

## 5. Conclusion and Future Scope