Galois Fields with Characteristic 2

November 11, 2023

Abstract

Galois fields are finite fields, that is an algebraic structure where addition and multiplication have inverses that are closed. These algebraic structures have extensive application in cryptography and in computer science in general. In this document we give a formal definition and give some useful properties of Galois Fields with characteristic 2

1 Defintion

- We consider galois fields with characteristic 2 and use the following notation to denote a field of order q, $\mathbf{GF}(2^q)$.
- For every $a \in \mathbf{GF}(2^q)$, a has 2 equvilent representations.
  - The first is the polynomial representation: $a(x) = a_{q-1}x^{q-1} + ... + a_0$
  - The second is the binary representation that are just the coeffiecents of the above polynomial: $a = (a_{q-1}, ..., a_0)$.
  - $a_i \in \{0, 1\}$
  - In this document we will use the binary representation as often as possible.
- For all $a, b, c \in \mathbf{GF}(2^q)$ we have the following which define a Galois Field:
  - Associativity over addition and multiplication: $a+(b+c) = (a+b)+c$ and $a * (b * c) = (a * b) * c$
  - Communitivity over addition and multiplication:: $a + b = b + a$ and $a * b = b * a$
  - There exist an additive and multiplicitive identity i.e. $a + 0 = a$ and $a * 1 = a$
    - In $\mathbf{GF}(2^q)$, 1 is the binary representation of all 1 i.e. $(0, ..., 0, 1)$.
    - In $\mathbf{GF}(2^q)$, 0 is the binary representation of all 0s i.e. $(0, ..., 0)$.
  - Additive inverse: there exist an $-a \in \mathbf{GF}(2^q)$ such that $a + -a = 0$
  - Multiplicitive inverse: there exist an $a^{-1} \in \mathbf{GF}(2^q)$ such that $a * a^{-1} = 1$
  - Distributive: $a * (b + c) = a * b + a * c$

2 Addition and its inverse

Adding in $\mathbf{GF}(2^q)$ is super easy. For $a, b \in \mathbf{GF}(2^q)$, $c = a + b$ is defined as follows: $c = (c_{q-1}, ..., c_0)$ such that $c_i = a_i \otimes b_i$, where $\otimes$ is the XOR operation. For every $a \in \mathbf{GF}(2^q)$, $a^{-1} = a$. Thus the additive inverse is clearly closed and trivial to compute. A neat consequence of this is that addition and suntraction are the same operation.

3 Multiplication and its inverse

Multiplying is significantly more complicated. It requires using the polynomial representation to fully understand and requires a quick aside on polynomial arithmetic

3.1 Polynomial Multiplication

Let $a(x)$ and $b(x)$ be polynomials. Multiplying two polynomials is simply a matter of distrubuting and adding all the terms as you would in grade school. The one caveate is that adding is done modulo 2. As an example $(x^2+x)*(x+1) = x^3 + x^2 + x^2 + x = x^3 + x$. This can be done quickly using on computers with the binary representation using bitshift and XOR operations. See galois.py for the implementation. Now the problem with defining multiplication this way is it is not closed i.e. the product is a polynomial of order $2q$ for two polynomials in $\mathbf{GF}(2^q)$. So we need to define something akin to Euclidean division where we take the remainder. But if you remember from basic modular arithmetic, multiplication "plays nice" only when we take the remainder of some prime number. So with polynomials we need some concept of a prime polynomial.

3.2 Polynomial Euclidean Division and Algorithm

Recall that Eucliden division of two integers m, n requires finding two intergers q, r such $m = qn + r$. Here q is the quotient and r is the remainder. So we want to do somthing similar with polynomials. For two polynomials $a(x), b(x)$ find $q(x), r(x)$ such that $a(x) = q(x)b(x) + r(x)$. This can be done with pen and paper and is typically done in grade school. Several online resources also exists if one needs a review. Again this can be implemented fairly effieciently in code or hardware using the binary representation and the file galois.py contains a python implementation of this algorithm. This also allows us to define a factor. $b(x)$ is a factor of $a(x)$ if its Euclidean division produces $r(x) = 0$.

Now that we have a notion of Euclidean division we can define the notation of a G.C.D. and use the Euclidean Algorithm to compute the G.C.D. Now the G.C.D. of two polynomials is the largest polynomial that divides both polynomials. Moreover, the Euclidean algorithm is exactly the same as that for the integers and its implementation is shown in galois.py. Moreover for more details on the Euclidean Algorithm see Euclidean_Algo.pdf.

Finally, we can again draw an analog from the integer extended Euclidean algorithm to the polynomials. We can use the same exact algorithm as presente in Euclidean_Algo.pdf. Only we replace all integer arithmetic with the arithmetic defined ove the Galois field. This is implemented in galois.py. This algorithm allows us to compute multiplicitive inverses of polynomials in this field. Again the caveat here is the the multiplicitive inverse is modulo some special "prime polynomial". Tables exists that give these polynomials, but we will soon see how to compute these for a given galois field of order q.

3.3 Polynomial Factorization

3.4 Primitive Polynomials

- Euler Totien Function