50.042 FCS Summer 2024
Lecture 10 – Modular Arithmetic II

Felix LOH
Singapore University of Technology and Design

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

With selected materials adapted from: *Understanding Cryptography: A Textbook for Students and Practitioners, by C. Paar and J. Pelzl*

Last lecture…

- We discussed several algebraic structures and their properties
  - Groups
  - Rings
  - Fields
  - Finite fields or Galois fields
  - Prime fields
  - Extension fields

Finite fields (recap)

- In cryptography, we are usually interested in fields with a finite number of elements
  - These fields are known as finite fields or Galois fields
- The number of elements in the field is called the *order* of the field (similar to the order of a finite group)

- The following theorem regarding finite fields is fundamental:

  A field with order m only exists if m is a prime power, i.e. $m = p^n$ for some positive integer n and prime integer p. p is called the characteristic of the finite field.

Prime fields (recap)

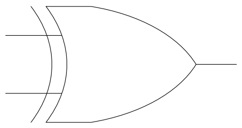
- A prime field *GF*(*p*) is a Galois (finite) field of prime order (i.e. $n = 1$)
- The two operations of the field are integer addition *modulo p* and integer multiplication *modulo p*

- The following important theorem defines a prime field:

  Let *p* be a prime integer. The integer ring $\mathbb{Z}_p$ is denoted as *GF*(*p*) and is referred to as a prime field, or as a Galois field with a prime number of elements. All non-zero elements of *GF*(*p*) have an inverse. Arithmetic in *GF*(*p*) is done *modulo p*.

Prime fields: example (recap)

- Prime field *GF*(2):
  - This is a very important prime field (the smallest possible finite field)
  - Addition and multiplication are calculated *modulo* 2, as shown in the tables below:

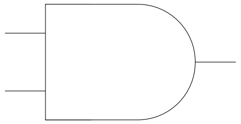


addition

+	0	1
0	0	1
1	1	0

multiplication

$\times$	0	1
0	0	0
1	0	1



  - Addition *modulo* 2 is the XOR operation (likewise for subtraction *modulo* 2)
    - i.e. XOR operation is addition or subtraction in the Galois field *GF*(2)
  - Multiplication *modulo* 2 is the AND operation
    - i.e. AND operation is multiplication in the Galois field *GF*(2)

Extension fields (recap)

- An extension field is a Galois (finite) field $GF(p^n)$ with $n > 1$
- The order of an extension field is **not** prime (since $m = p^n$ is not prime)
  - This means that **if** we were to represent the elements of $GF(p^n)$ as **integers**, not all non-zero integers of $GF(p^n)$ will have an **inverse**
  - This consequently implies that we **cannot** perform **integer** addition and multiplication ***modulo*** $\boldsymbol{p^n}$ on the elements of an extension field
  - Rather, we need to represent the elements of an extension field using a different notation (i.e. not integers, but *polynomials* instead) and implement different rules for performing arithmetic on these elements

Extension fields (recap)

- We can roughly think of an extension field $GF(p^n)$ as an algebraic structure that contains n "instances" of a prime field $GF(p)$

- The elements of an extension field $GF(p^n)$ are not represented by integers
- Rather, the elements of $GF(p^n)$ are represented by *polynomials* of degree n-1 with coefficients in the prime field $GF(p)$
- Arithmetic in the extension field is achieved by performing a certain kind of *polynomial arithmetic*:
  - Addition and subtraction: bitwise XOR of the corresponding coefficients
  - Multiplication: polynomial multiplication and reduction by a fixed *irreducible polynomial*

Extension fields: example (recap)

- The extension field $GF(2^8)$ is used in the layers of AES (this field was chosen because each of the field elements can be represented by one byte)
- Each element in $GF(2^8)$ is represented by a polynomial of the form:

  $A(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0,\ a_i \in GF(2) = \{0, 1\}$

- E.g. byte 0xC3 = 11000011_2 can be represented as: $A(x) = x^7 + x^6 + x + 1$
- Note that the factors x^7, x^6, etc. are placeholders (and are not the variables to be evaluated); we do **not** need to store these factors since each polynomial $A(x)$ can be stored in digital form as an 8-bit vector:

  $A = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$

Extension fields: polynomial arithmetic

- In this lecture, we will focus on *polynomial* arithmetic in extension fields $GF(2^n)$, particularly for $GF(2^8)$:
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Inversion (multiplicative inverse)

Overview of polynomial arithmetic in $GF(2^n)$

- $GF(2^n)$ addition and subtraction: Do addition *modulo* 2 (XOR) of each of the corresponding coefficients of the two input polynomials
- $GF(2^n)$ multiplication: Do polynomial multiplication, followed by polynomial division by an *irreducible* polynomial, and keep only the remainder
  - An *irreducible* polynomial is analogous to a prime number
  - $GF(2^n)$ multiplication is analogous to $a \cdot b \bmod 2$ computation in $GF(2)$, where we multiply the two integers a and b, then do an integer divide by the modulus 2 (a prime number), and keep only the remainder
- $GF(2^n)$ inversion: Use the Extended Euclidean algorithm with the input polynomial and an *irreducible* polynomial

Addition and subtraction in $GF(2^n)$

- Formal definition: Let $A(x)$ and $B(x) \in GF(2^n)$. The sum of the two elements is computed as

  $C(x) = A(x) + B(x) = \sum_{i=0}^{n-1} c_i x^i$, $\quad c_i \equiv a_i + b_i \bmod 2$

  and the difference of the two elements is computed as

  $C(x) = A(x) - B(x) = \sum_{i=0}^{n-1} c_i x^i$, $\quad c_i \equiv a_i - b_i \bmod 2 \equiv a_i + b_i \bmod 2$

- Practically, this means that we just do a bitwise XOR of the corresponding coefficients of $A(x)$ and $B(x)$:

  $A(x) + B(x) = A(x) - B(x) = (a_{n-1} \oplus b_{n-1})x^{n-1} + \ldots + (a_1 \oplus b_1)x + (a_0 \oplus b_0)$

Addition and subtraction in $GF(2^n)$

- Example addition/subtraction in $GF(2^4)$:

  $A(x) = x^3 + x^2 + 1$

  $B(x) = x^2 + x + 0x^3$

  Then $C(x) = A(x) + B(x)$

  $= A(x) - B(x)$

  $= (1 \oplus 0)x^3 + (1 \oplus 1)x^2 + (0 \oplus 1)x + (1 \oplus 0)$

  $= x^3 + x + 1$

Addition/subtraction in $GF(2^n)$ – application

- Recall that addition in $GF(2^8)$ is used in the Key Addition layer of AES
  - The two inputs to this layer are the 16-byte state matrix C and the round key k_i (also 16 bytes), where i is the ith round
  - The round keys are derived using the key schedule
  - The two inputs are combined using addition in $GF(2^8)$, i.e. bitwise XOR operation
- Addition in $GF(2^8)$ is also used in the MixColumn sublayer of AES

The AES algorithm: Key Addition layer

- The two inputs to this layer are the 16-byte state matrix *C* and the round key k_i (also 16 bytes), where *i* is the *i*th round
- The round keys are derived using the key schedule
- The two inputs are combined using a bitwise XOR operation
  - This is equivalent to addition in the Galois extension field $GF(2^8)$
  - Recall that bitwise XOR operation is also equivalent to addition modulo 2, which means that it is also addition in the Galois field $GF(2)$

Irreducible polynomials

- Irreducible polynomials are analogous to prime numbers
  - They cannot be factorized, as their only factors are 1 and the polynomial itself
- Irreducible polynomials are necessary for reduction of the product after polynomial multiplication in $GF(2^n)$

- E.g. the irreducible polynomial used in AES is $x^8 + x^4 + x^3 + x + 1$, with arithmetic performed in the extension field $GF(2^8)$
  - This irreducible polynomial is part of the AES specification
- E.g. an irreducible polynomial for $GF(2^4)$ is $x^4 + x + 1$

Multiplication in $GF(2^n)$

- Formal definition: Let $A(x)$ and $B(x) \in GF(2^n)$, and let

  $P(x) = \sum_{i=0}^{n} p_i x^i, \quad p_i \in GF(2) = \{0, 1\}$

  be an *irreducible* polynomial. The product of the two elements $A(x)$ and $B(x)$ is computed as

  $C(x) \equiv A(x) \cdot B(x) \bmod P(x)$

- Note that every extension field $GF(2^n)$ requires an irreducible polynomial $P(x)$ of degree n (and **not** n-1), with coefficients from $GF(2)$, in order to perform a polynomial division of the product

Multiplication in $GF(2^n)$

- Practically, this means that we perform the multiplication computation in three steps:

1. Do a regular polynomial multiplication of $A(x)$ and $B(x)$ to obtain $C'(x)$
    - The coefficients are in $GF(2)$, so use AND for the multiplication of the terms of the polynomials $A(x)$ and $B(x)$; and XOR for the addition of the corresponding terms after multiplication
2. Do a polynomial division of the product $C'(x)$ by the irreducible polynomial $P(x)$
3. Keep the remainder of the polynomial division, this is our desired result $C(x)$

Multiplication in $GF(2^n)$

- Example multiplication in $GF(2^4)$:

  $A(x) = x^3 + x^2 + 1$

  $B(x) = x^2 + x$

- Then $C'(x) = A(x) \cdot B(x)$

  $$= (x^3 + x^2 + 1)\,(x^2 + x)$$
  $$= x^5 + x^4 + x^2 + x^4 + x^3 + x$$
  $$= x^5 + x^3 + x^2 + x$$

- Note that the degree of the resulting polynomial $C'(x)$ is 5, which is greater than the maximum degree allowed by the extension field (which is 3); we will reduce $C'(x)$ after discussing polynomial division

$A(x) = x^6 + x + 1$
$B(x) = x^4 + x^2 + 1$
$P(x) = x^8 + x^4 + x^3 + x + 1$

$A(x) \cdot B(x) \bmod P(x) = x^3$

$A(x) \cdot B(x) = (x^6 + x + 1)(x^4 + x^2 + 1)$
$= x^{10} + x^8 + x^6 + x^5 + x^3 + x + x^4 + x^2 + 1$
$= x^{10} + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1x^0$

$\Rightarrow$ convert to binary string

$\Rightarrow$ 10101111111

$P(x) \rightarrow$ convert to binary $\rightarrow$ 100011011

Division (bitwise XOR)

```
                 101        → quotient
100011011 ) 10101111111
            100011011
XOR →       00100010001
            000000000
XOR →       01000100111
            100011011
XOR →       00000001000  → remainder = x^3
```

Polynomial division in $GF(2^n)$

- We can do polynomial division using the traditional long division method

- Example division in $GF(2^4)$:

  $C'(x) = x^5 + x^3 + x^2 + x$ (the product $A(x) \times B(x)$ we obtained earlier, to be reduced by an irreducible polynomial $P(x)$ via polynomial division)

  $P(x) = x^4 + x + 1$ (the irreducible polynomial)

Polynomial division in $GF(2^n)$

1. To divide $C'(x)$ by $P(x)$, we first rewrite each polynomial as a binary string:

   $C'(x) = 101110_2$

   $P(x) = 10011_2$

2. Then we divide $C'(x)$ by $P(x)$ using long division:

$$10011 \overline{)\,101110}$$

Polynomial division in $GF(2^n)$

3. Check the leading bit of the current dividend. If it is ‘1’, place a ‘1’ at the appropriate position in the partial quotient, then write a copy of the divisor string below the current dividend. If it is ‘0’, place a ‘0’ instead, and write a string of all 0’s below the current dividend. Left justify the partial quotient.

- In this case, the leading bit is a ‘1’, so we write the left-justified partial quotient as shown below and copy the bit string 10011_2 below the current dividend:

```
        00001
10011 ) 101110
        10011
```

Left justify the partial quotient

Leading bit

Polynomial division in $GF(2^n)$

4. Then subtract the copied bit string from the current dividend. This is actually addition/subtraction in $GF(2)$, so this is just a bitwise XOR operation.

- In this case, the copied bit string is 10011_2 and so we subtract this value from the partial string 10111_2 of the current dividend to obtain 00100_2:

$$
\begin{array}{r|l}
 & 00001 \\
\hline
10011 & 101110 \\
 & \underline{10011} \\
 & 00100
\end{array}
$$

Polynomial division in $GF(2^n)$

5. Bring down the next trailing bit value of the **original** dividend, then repeat steps 3, 4 and 5 until you complete the rightmost position of the quotient. When you repeat step 3, make sure that you check the **next** leading bit of the current dividend.

- In this case, the trailing bit is a ‘0’. We bring this down to the current dividend. Then, we repeat step 3 and check the **next** leading bit of the current dividend 001000_2 (which is also ‘0’):

$$
\begin{array}{r}
00001 \\
10011 \overline{\smash{)}\,101110} \\
\underline{10011} \\
001000
\end{array}
$$

Bring down the next trailing bit value of original dividend

Next leading bit of current dividend

Polynomial division in $GF(2^n)$

6. After the rightmost position of the quotient is completed, we keep the remainder as the desired result of the polynomial division.

- In this case, after repeating steps 3, 4, and 5, we obtain the remainder 01000_2 (which we keep) and the quotient 000010_2 (which we throw away):

$$
\begin{array}{r@{\,}l}
 & 000010 \quad \longleftarrow \text{quotient} \\
10011 & \overline{)101110} \\
 & \underline{10011} \\
 & 001000 \\
 & \underline{00000} \\
 & 01000 \quad \longleftarrow \text{remainder}
\end{array}
$$

- Thus $C'(x)$ *mod* $P(x) = (x^5 + x^3 + x^2 + x)$ *mod* $(x^4 + x + 1) = 01000_2 = x^3$
  - Aside: we also have $C'(x) / P(x) = (x^5 + x^3 + x^2 + x) / (x^4 + x + 1) = 000010_2 = x$

Multiplication in $GF(2^n)$ – application

- Multiplication in $GF(2^8)$ is used in the MixColumn layer of AES
- Recall that the MixColumn sublayer is a linear transformation which mixes each column of the input state matrix

- The operation performed is $C = MixColumn(B_{SR})$, where B_{SR} is the input state matrix after the ShiftRows operation is executed
- Each 4-byte column of B_{SR} is treated as a vector and multiplied by a fixed constant 4×4 matrix. Multiplication and addition of the coefficients is done in $GF(2^8)$
  - The irreducible polynomial used in AES is $x^8 + x^4 + x^3 + x + 1$, as defined in the AES specification, for polynomial reduction of the product

Multiplication in $GF(2^n)$ – application

- E.g. For the first column of C and B_{SR} respectively:

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

- Each state byte C_i and B_i with $i = 0, \ldots, 15$, is an 8-bit value that represents an element from the Galois extension field $GF(2^8)$
- Here, column (C_0, C_1, C_2, C_3) is the result of a matrix-vector multiplication of the fixed constant 4×4 matrix with the first column of B_{SR}
  - Additions in the matrix-vector multiplication are $GF(2^8)$ additions
  - Multiplications in the matrix-vector multiplication are multiplications in $GF(2^8)$

Multiplication in $GF(2^n)$ – application

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

- E.g. for the computation $C_3 = 03 \cdot B_0 + 01 \cdot B_5 + 01 \cdot B_{10} + 02 \cdot B_{15}$, the operation $03 \cdot B_0$ is a multiplication operation in $GF(2^8)$:

  $00000011_2 \cdot B_0(x) = (x + 1) \cdot B_0(x)$,

  where $x + 1$ and $B_0(x)$ are polynomial elements in $GF(2^8)$, each representing an 8-bit binary vector

Additional practice for multiplication in $GF(2^n)$

- Let's practice the multiplication of two polynomials in $GF(2^8)$:

  $A(x) = x^6 + x + 1$

  $B(x) = x^4 + x^2 + 1$

  $P(x) = x^8 + x^4 + x^3 + x + 1$ (AES irreducible polynomial)

- We should get the answer:

  $A(x) \cdot B(x) \ mod \ P(x) \equiv x^3$

$(x^6+x+1)(x^4+x^2+1) = x^{10}+x^8+x^6+x^5+x^3+x+x^4+x^2+1$

$= x^{10}+x^8+x^6+x^5+x^4+x^3+x^2+x$

1 0 1 0 1 1 1 1 1 1 1

101

100011011) 10101111111

100011011

00100010011

000000000

01000100111

100011011

00000001000

x^3

Inversion in $GF(2^n)$

- Formal definition: Let $A(x) \in GF(2^n)$ be a non-zero element and let $P(x)$ be an *irreducible* polynomial of $GF(2^n)$. The inverse $A^{-1}(x)$ of $A(x)$ is defined as

  $A^{-1}(x) \cdot A(x) \equiv 1 \bmod P(x)$

- For small finite fields with 2^{16} elements or less, like $GF(2^8)$, lookup tables are often used
- The tables contain the precomputed inverses of all field elements
- E.g. a 16×16 lookup table is used in the S-boxes of AES (to save on computation time)

Inversion in $GF(2^n)$

- To compute the inverse of $A(x)$, we can use the *extended Euclidean algorithm* (EEA)
- But before we discuss the EEA, we need to briefly cover the *Euclidean algorithm* first
- So let’s discuss the Euclidean algorithm and its application towards finding the greatest common divisor of two *integers*
- Then we’ll talk about the EEA and its application towards finding the modular inverse of an *integer*, then how we can use EEA to find the inverse of a *polynomial* in $GF(2^n)$

The Euclidean algorithm

- The Euclidean algorithm is an efficient way to compute the greatest common divisor of two positive integers r_0 and r_1 (with $r_1 \leq r_0$)
- The basic idea behind this algorithm: $gcd(r_0, r_1) = gcd(r_0 - r_1, r_1)$
  - i.e. the problem of finding the greatest common divisor of r_0 and b can be <u>reduced</u> to finding greatest common divisor of the difference $r_0 - b$ and b

- Proof: Let $gcd(r_0, r_1) = g$. Since g divides both r_0 and r_1, we can rewrite r_0 and r_1 as $r_0 = gx$ and $r_1 = gy$.

  Then $gcd(r_0 - r_1, r_1) = gcd(gx - gy, gy) = gcd(g(x - y), gy) = g = gcd(r_0, r_1)$

The Euclidean algorithm

- We can keep subtracting r_1 from r_0,

  i.e. $gcd(r_0, r_1) = gcd(r_0 - r_1, r_1) = gcd(r_0 - 2r_1, r_1) = \ldots = gcd(r_0 - nr_1, r_1)$, as long as $r_0 - nr_1 > 0$

- To minimize the number of subtractions needed, we make n as large as possible
- This can be done by taking the modulus: $gcd(r_0, r_1) = gcd(r_0 \ mod \ r_1, r_1)$

- Since ($r_0 \ mod \ r_1$) is smaller than r_1, we swap their positions:

  $gcd(r_0, r_1) = gcd(r_1, r_0 \ mod \ r_1)$

The Euclidean algorithm

- Then we can keep doing this process iteratively :

  $gcd(r_0, r_1) = gcd(r_1, r_0 \; mod \; r_1) = gcd(r_1, r_2)$

  $gcd(r_1, r_2) = gcd(r_2, r_1 \; mod \; r_2) = gcd(r_2, r_3)$

  $gcd(r_2, r_3) = \ldots$

  $\ldots = gcd(r_{last}, 0) = r_{last}$

- So the Euclidean algorithm is as follows:

  Given two two positive integers r_0 and r_1 (with $r_1 \leq r_0$), we iteratively compute $r_i = r_{i\text{-}2} \; mod \; r_{i\text{-}1}$, for $i = 2, 3, 4, \ldots, n$ until we obtain $r_n = 0$. Then we have $r_{n\text{-}1} = gcd(r_0, r_1)$.

The extended Euclidean algorithm (EEA)

- The EEA is able to compute the modular inverse of an integer, in addition to finding the greatest common divisor
- The fundamental idea behind EEA is that the algorithm computes a linear combination of the form:

  45 = 4(10) + 5

  10 = 5(2) + 0

  10 45

  $gcd(r_0, r_1) = sr_0 + tr_1$, where s and t are integer coefficients

  5 = 10 + 45

- The above relation is known as the *Diophantine equation*
- Recall from Lecture 9 that for an integer ring $\mathbb{Z}_m$, the multiplicative inverse of $a \in \mathbb{Z}_m$ exists if and only if $gcd(a, m) = 1$

The extended Euclidean algorithm (EEA)

- So using the equation $gcd(r_0, r_1) = sr_0 + tr_1$, and setting $r_0 = m$ and $r_1 = a$, we get:

  $sm + ta = gcd(m, a)$

  $sm + ta = 1$, since we are assuming that $a^{-1} \in \mathbb{Z}_m$ exists

  $s \cdot 0 + t \cdot a \equiv 1 \; mod \; m$

  $t \cdot a \equiv 1 \; mod \; m$,
- This implies that t is the multiplicative inverse of a, i.e. $t \equiv a^{-1} \; mod \; m$

- Now let's briefly discuss the EEA, without going into too much detail about its derivation

The extended Euclidean algorithm (EEA)

- The EEA follows a similar procedure as the Euclidean algorithm:

  We start by setting $r_0 = m$, $r_1 = a$, $t_0 = 0$ and $t_1 = 1$,

  then we iteratively compute $r_i = r_{i-2} \bmod r_{i-1}$,

  as well as the quotient $q_{i-1} = (r_{i-2} - r_i) / r_{i-1}$ and the value $t_i = t_{i-2} - q_{i-1}\, t_{i-1}$, for $i = 2, 3, 4, \ldots, n$ until we obtain $r_n = 0$.

  Then we have $r_{n-1} = gcd(a, m) = 1$ and $t_{n-1} \equiv a^{-1} \bmod m$

- Note: The above strategy is just for your knowledge; we will use the EEA in a slightly different way to obtain the inverse of a polynomial

Applying the EEA towards polynomial inverse

- Now let's discuss how we can use the EEA to compute the multiplicative inverse of a *polynomial* $A(x)$ in an extension field $GF(2^n)$

- To compute the inverse, we set the field element $A(x)$ and the irreducible polynomial $P(x)$ as inputs to the EEA
- The EEA computes the auxiliary polynomials $s(x)$ and $t(x)$, as well as $gcd(P(x), A(x))$, such that:

  $s(x)\, P(x) + t(x)\, A(x) = gcd(P(x), A(x)) = 1$
- Note that since $P(x)$ is irreducible, the greatest common divisor of $P(x)$ and $A(x)$ is always 1

Applying the EEA towards polynomial inverse

- As such, we have:

  $s(x)\ P(x) + t(x)\ A(x) = 1$

  mod $p(x)$

  $s(x)\ (0) + t(x)\ A(x) \equiv 1 \ mod\ P(x)$

  $t(x)\ A(x) \equiv 1 \ mod\ P(x)$

- i.e. $t(x) \equiv A^{-1}(x) \ mod\ P(x)$

- So, similar to the integer case we discussed earlier, $t(x)$ is the multiplicative inverse of $A(x)$

Applying the EEA towards polynomial inverse

- To find the inverse of $A(x)$, we start with the polynomials $P(x)$ and $A(x)$
- Then we iteratively calculate the remainder after polynomial division (similar to how we computed the remainder in the Euclidean algorithm), and we **also** keep track of the **quotient** each time
- After we have computed the final quotient, we then "backtrack" and calculate the $t(x)$ polynomial values
- The final $t(x)$ we compute is the inverse of $A(x)$

Applying the EEA towards polynomial inverse

- Let's illustrate this with an example
- Suppose we want to find the inverse of $A(x) = x^2$ in the extension field $GF(2^3)$, with $P(x) = x^3 + x + 1$
- We will use a table to keep track of the relevant polynomials, as we apply the EEA to find the inverse of $A(x)$:

1. First, initialize the table with $P(x)$ and $A(x)$ in the "Remainder" column and '0' in the top entry of the "Quotient" column.

Remainder	Quotient	$t(x)$
$x^3 + x + 1$ ⟵ $P(x)$	0	
x^2 ⟵ $A(x)$		

$A(x) = x^2$, $P(x) = x^3+x+1$

r	q	$t(x)$ x^2+x+1
$P(x) = x^3+x+1$	1.multiply	
	x	$x+1$
$A(x) = x^2$	1.multiply	2.add
	$x+1$	1
$x+1$		2.add
1	$x+1$	0
0		

$$\begin{array}{r} 10 \Rightarrow x \quad q \\ 100 \overline{)1011} \\ 100\downarrow \\ \hline 0011 \\ 000 \\ \hline 011 \Rightarrow x+1 \quad r \end{array}$$

$$\begin{array}{r} 11 \Rightarrow x+1 \quad q \\ 11 \overline{)100} \\ 11\downarrow \\ \hline 010 \\ 11 \\ \hline 01 \Rightarrow 1 \quad r \end{array}$$

verify $A^{-1}(x) = x^2+x+1$:

$A^{-1}(x) \cdot A(x) \equiv 1 \bmod P(x)$

$A^{-1}(x) \cdot A(x) = x^2 \cdot (x^2+x+1)$

$= x^4+x^3+x^2$

reduce with $P(x)$ $\Rightarrow$

$$\begin{array}{r} 11 \\ 1011 \overline{)11100} \\ 1011\downarrow \\ \hline 01010 \\ 1011 \\ \hline 0001 \end{array} \leftarrow 1 \bmod P(x)$$

Applying the EEA towards polynomial inverse

2. Next, do a polynomial division of the last remainder ($P(x)$ in this case) by the current remainder ($A(x)$ in this case) to obtain the current quotient and the new remainder.

- $(x^3 + x + 1) / x^2 = 1011_2 / 100_2$

$$
\begin{array}{rl}
\underline{0010} & \leftarrow \text{quotient} \\
100\,\overline{)1011} & \\
\underline{100} & \\
0011 & \\
\underline{000} & \\
011 & \leftarrow \text{remainder}
\end{array}
$$

Remainder	Quotient	$t(x)$
$x^3 + x + 1$	0	
x^2	x ← current quotient	
$x + 1$ ← new remainder		

Applying the EEA towards polynomial inverse

3. Repeat step 2 and divide the last remainder by the current remainder to obtain the current quotient and the new remainder Continue to repeat step 2 until the new remainder is 0.

- Repeating step 2: $x^2 / (x + 1) = 100_2 / 11_2$

```
     011   ← quotient
11 ) 100
     11
     ---
     010
      11
     ---
      01   ← remainder
```

Remainder	Quotient	$t(x)$
$x^3 + x + 1$	0	
x^2	x	
$x + 1$	$x + 1$ ← current quotient	
1 ← new remainder		

Applying the EEA towards polynomial inverse

- Computing step 2 yet again, we get $(x + 1) / 1 = x + 1$ (with a remainder of 0)

<u>Remainder</u>	<u>Quotient</u>	<u>$t(x)$</u>
$x^3 + x + 1$	0	
x^2	x	
$x + 1$	$x + 1$	
1	$x + 1$ ← current quotient	
0 ← new remainder		

Applying the EEA towards polynomial inverse

4. Now that we have computed all of the remainder values, we can begin calculating the values for $t(x)$. First, initialize the value of $t(x)$ that is in the last complete row to '0' and the value of $t(x)$ in the row above to '1'.

Remainder	Quotient	$t(x)$
$x^3 + x + 1$	0	
x^2	x	
$x + 1$	$x + 1$	1 ← initialize
1	$x + 1$	0 ← initialize
0		

Applying the EEA towards polynomial inverse

5. Compute the next value of $t(x)$ for the row above by using the following formula:

   next $t(x)$ = current quotient $\cdot$ current $t(x)$ + previous $t(x)$

- Here, next $t(x) = (x + 1) \cdot 1 + 0 = x + 1$

Remainder	Quotient	$t(x)$
$x^3 + x + 1$	0	
x^2	x	$x + 1$ ← next $t(x)$
$x + 1$	$x + 1$ ← current quotient	1 ← current $t(x)$
1	$x + 1$	0 ← previous $t(x)$
0		

Applying the EEA towards polynomial inverse

6. Repeat step 5 and keep computing the next value of $t(x)$ until you reach the top row, which will be the final value of $t(x)$. Then we have $A^{-1}(x)$ = final $t(x)$.

- Here, we repeat step 5 to obtain: final $t(x) = x \cdot (x + 1) + 1 = x^2 + x + 1$

  Therefore, $A^{-1}(x) = x^2 + x + 1$

Remainder	Quotient	$t(x)$
$x^3 + x + 1$	0	$x^2 + x + 1$ ← final $t(x)$
x^2	x ← current quotient	$x + 1$ ← current $t(x)$
$x + 1$	$x + 1$	1 ← previous $t(x)$
1	$x + 1$	0
0		

Applying the EEA towards polynomial inverse

- Let’s check that $A^{-1}(x) = x^2 + x + 1$ is indeed the inverse of $A(x) = x^2$, i.e. that $A^{-1}(x) \cdot A(x) \equiv 1 \; mod \; P(x)$:

  $A^{-1}(x) \cdot A(x) = (x^2 + x + 1) \cdot x^2 = x^4 + x^3 + x^2$

- Then, doing a polynomial division of $x^4 + x^3 + x^2$ by $P(x)$:

  $(x^4 + x^3 + x^2) \,/\, (x^3 + x + 1) = 11100_2 \,/\, 1011_2$

$$\begin{array}{rr} & 00011 \\ 1011 & \overline{)\,11100} \\ & 1011 \\ \hline & 01010 \\ & 1011 \\ \hline & 0001 \end{array} \quad \longleftarrow \text{remainder} = 1$$

- We obtain a remainder of 1, as expected.
- Thus, $A^{-1}(x) = x^2 + x + 1$

Inversion in $GF(2^n)$ – application

- Inversion in $GF(2^8)$ is utilized within the S-boxes of the Byte Substitution layer of AES
  - Recall that the S-boxes do an inversion in $GF(2^8)$, then an affine mapping
- The S-boxes use a precomputed multiplicative inverse lookup table
  - The table is shown below, which contains all inverses in $GF(2^8)$, modulo $P(x) = x^8 + x^4 + x^3 + x + 1$, for all field elements XY:

X \ Y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	8D	F6	CB	52	7B	D1	E8	4F	29	C0	B0	E1	E5	C7
1	74	B4	AA	4B	99	2B	60	5F	58	3F	FD	CC	FF	40	EE	B2
2	3A	6E	5A	F1	55	4D	A8	C9	C1	0A	98	15	30	44	A2	C2
3	2C	45	92	6C	F3	39	66	42	F2	35	20	6F	77	BB	59	19
4	1D	FE	37	67	2D	31	F5	69	A7	64	AB	13	54	25	E9	09
5	ED	5C	05	CA	4C	24	87	BF	18	3E	22	F0	51	EC	61	17
6	16	5E	AF	D3	49	A6	36	43	F4	47	91	DF	33	93	21	3B
7	79	B7	97	85	10	B5	BA	3C	B6	70	D0	06	A1	FA	81	82
8	83	7E	7F	80	96	73	BE	56	9B	9E	95	D9	F7	02	B9	A4
9	DE	6A	32	6D	D8	8A	84	72	2A	14	9F	88	F9	DC	89	9A
A	FB	7C	2E	C3	8F	B8	65	48	26	C8	12	4A	CE	E7	D2	62
B	0C	E0	1F	EF	11	75	78	71	A5	8E	76	3D	BD	BC	86	57
C	0B	28	2F	A3	DA	D4	E4	0F	A9	27	53	04	1B	FC	AC	E6
D	7A	07	AE	63	C5	DB	E2	EA	94	8B	C4	D5	9D	F8	90	6B
E	B1	0D	D6	EB	C6	0E	CF	AD	08	4E	D7	E3	5D	50	1E	B3
F	5B	23	38	34	68	46	03	8C	DD	9C	7D	A0	CD	1A	41	1C

Inversion in $GF(2^n)$ – application

- According to the table, if byte XY = 0x23, then the inverse should be 0xF1

- As extra practice, check that this is the case, i.e. if

  $A(x) = x^5 + x + 1$, then we should have

  $A^{-1}(x) = x^7 + x^6 + x^5 + x^4 + 1$