Lecture 2: Number Theory, Groups and Finite Fields

TTM4135

Relates to Stallings Chapters 2 and 5

Spring Semester, 2024

Motivation

- Cryptography makes use of mathematics, computer science and engineering

Motivation

- Cryptography makes use of mathematics, computer science and engineering
- Mostly the mathematics is *discrete mathematics* because cryptology deals with finite objects such as alphabets and blocks of characters

Motivation

- Cryptography makes use of mathematics, computer science and engineering
- Mostly the mathematics is *discrete mathematics* because cryptology deals with finite objects such as alphabets and blocks of characters
- We therefore look at modular arithmetic which only deals with a finite number of values

Motivation

- Cryptography makes use of mathematics, computer science and engineering
- Mostly the mathematics is *discrete mathematics* because cryptology deals with finite objects such as alphabets and blocks of characters
- We therefore look at modular arithmetic which only deals with a finite number of values
- Understanding the algebraic structure of finite objects helps to build useful cryptographic properties

Outline

Outline

- Basic Number Theory
  - Primes and Factorisation
  - GCD and the Euclidean Algorithm
  - Modular arithmetic
- Groups
- Finite Fields
- Boolean Algebra

Factorisation

- Let $\mathbb{Z}$ denote the set of integers

Factorisation

- Let $\mathbb{Z}$ denote the set of integers
- For a and b in $\mathbb{Z}$, we say that a divides b (or a is a factor of b, or write $a|b$) if there exists k in $\mathbb{Z}$ such that $ak = b$

Factorisation

- Let $\mathbb{Z}$ denote the set of integers
- For *a* and *b* in $\mathbb{Z}$, we say that *a* divides *b* (or *a* is a factor of *b*, or write $a|b$) if there exists *k* in $\mathbb{Z}$ such that $ak = b$
- An integer $p > 1$ is said to be a *prime number* (or simply a *prime*) if its only positive divisors are 1 and *p*

Factorisation

- Let $\mathbb{Z}$ denote the set of integers
- For a and b in $\mathbb{Z}$, we say that a divides b (or a is a factor of b, or write $a|b$) if there exists k in $\mathbb{Z}$ such that $ak = b$
- An integer $p > 1$ is said to be a *prime number* (or simply a *prime*) if its only positive divisors are 1 and p
- We can test for prime numbers by trial division (up to the square root of the number being tested)

Factorisation

- Let $\mathbb{Z}$ denote the set of integers
- For *a* and *b* in $\mathbb{Z}$, we say that *a* divides *b* (or *a* is a factor of *b*, or write $a|b$) if there exists *k* in $\mathbb{Z}$ such that $ak = b$
- An integer $p > 1$ is said to be a *prime number* (or simply a *prime*) if its only positive divisors are 1 and *p*
- We can test for prime numbers by trial division (up to the square root of the number being tested)
- In a later lecture we will look at a more efficient way to check for primality

Basic Properties of Factors

1. If a divides b and a divides c, then a divides $b + c$

Basic Properties of Factors

1. If a divides b and a divides c, then a divides $b + c$
2. If p is a prime and p divides ab, then p divides a or b

Basic Properties of Factors

1. If a divides b and a divides c, then a divides $b + c$
2. If p is a prime and p divides ab, then p divides a or b

Euclidean division

For a and b in $\mathbb{Z}$, $a > b$, there exist unique q and r in $\mathbb{Z}$ such that:

$$a = bq + r$$

where $0 \leq r < b$.

Outline

Greatest common divisor (GCD)

The value d is the GCD of a and b, written $\gcd(a, b) = d$, if all of the following hold:

Greatest common divisor (GCD)

The value d is the GCD of a and b, written $\gcd(a, b) = d$, if all of the following hold:

1. d divides a and b

Greatest common divisor (GCD)

The value d is the GCD of a and b, written $\gcd(a, b) = d$, if all of the following hold:

1. d divides a and b
2. if c divides a and b then c divides d

Greatest common divisor (GCD)

The value d is the GCD of a and b, written $\gcd(a, b) = d$, if all of the following hold:

1. d divides a and b
2. if c divides a and b then c divides d
3. $d > 0$

Greatest common divisor (GCD)

The value d is the GCD of a and b, written $\mathrm{gcd}(a, b) = d$, if all of the following hold:

1. d divides a and b
2. if c divides a and b then c divides d
3. $d > 0$

We say that a and b are *relatively prime* if $\mathrm{gcd}(a, b) = 1$

Euclidean algorithm

One can find $d = \gcd(a, b)$.
Let q_i be the quotient and r_i be the remainder in the following.

Euclidean algorithm

One can find $d = \gcd(a, b)$.
Let q_i be the quotient and r_i be the remainder in the following.

$$a = bq_1 + r_1, \text{ for } 0 < r_1 < b$$

Euclidean algorithm

One can find $d = \gcd(a, b)$.
Let q_i be the quotient and r_i be the remainder in the following.

$$
\begin{aligned}
a &= bq_1 + r_1, \text{ for } 0 < r_1 < b \\
b &= r_1 q_2 + r_2, \text{ for } 0 < r_2 < r_1
\end{aligned}
$$

Euclidean algorithm

One can find $d = \gcd(a, b)$.
Let q_i be the quotient and r_i be the remainder in the following.

$$
\begin{aligned}
a &= bq_1 + r_1, \text{ for } 0 < r_1 < b \\
b &= r_1 q_2 + r_2, \text{ for } 0 < r_2 < r_1 \\
r_1 &= r_2 q_3 + r_3, \text{ for } 0 < r_3 < r_2
\end{aligned}
$$

Euclidean algorithm

One can find $d = \gcd(a, b)$.
Let q_i be the quotient and r_i be the remainder in the following.

$$\begin{aligned}
a &= bq_1 + r_1, \text{ for } 0 < r_1 < b \\
b &= r_1 q_2 + r_2, \text{ for } 0 < r_2 < r_1 \\
r_1 &= r_2 q_3 + r_3, \text{ for } 0 < r_3 < r_2 \\
&\vdots \\
r_{k-2} &= r_{k-1} q_k + r_k, \text{ for } 0 < r_k < r_{k-1}
\end{aligned}$$

Euclidean algorithm

One can find $d = \gcd(a, b)$.
Let q_i be the quotient and r_i be the remainder in the following.

$$
\begin{aligned}
a &= bq_1 + r_1, \text{ for } 0 < r_1 < b \\
b &= r_1 q_2 + r_2, \text{ for } 0 < r_2 < r_1 \\
r_1 &= r_2 q_3 + r_3, \text{ for } 0 < r_3 < r_2 \\
&\vdots \\
r_{k-2} &= r_{k-1} q_k + r_k, \text{ for } 0 < r_k < r_{k-1} \\
r_{k-1} &= r_k q_{k+1}, \text{ where } r_{k+1} = 0
\end{aligned}
$$

Euclidean algorithm

One can find $d = \mathrm{gcd}(a, b)$.
Let q_i be the quotient and r_i be the remainder in the following.

$$\begin{aligned}
a &= bq_1 + r_1, \text{ for } 0 < r_1 < b \\
b &= r_1 q_2 + r_2, \text{ for } 0 < r_2 < r_1 \\
r_1 &= r_2 q_3 + r_3, \text{ for } 0 < r_3 < r_2 \\
&\vdots \\
r_{k-2} &= r_{k-1} q_k + r_k, \text{ for } 0 < r_k < r_{k-1} \\
r_{k-1} &= r_k q_{k+1}, \text{ where } r_{k+1} = 0
\end{aligned}$$

Then $d = r_k = \mathrm{gcd}(a, b)$.

Data: a, b

Algorithm: Euclidean algorithm

Data: a, b
Result: $\gcd(a, b)$

Algorithm: Euclidean algorithm

Data: a, b
Result: $\mathrm{gcd}(a, b)$
$r_{-1} \leftarrow a$;

Algorithm: Euclidean algorithm

Data: a, b
Result: $\gcd(a, b)$
$r_{-1} \leftarrow a$;
$r_0 \leftarrow b$;

Algorithm: Euclidean algorithm

Data: a, b
Result: $\gcd(a, b)$
$r_{-1} \leftarrow a$;
$r_0 \leftarrow b$;
$k \leftarrow 0$;

Algorithm: Euclidean algorithm

Data: a, b
Result: $\gcd(a, b)$
$r_{-1} \leftarrow a$;
$r_0 \leftarrow b$;
$k \leftarrow 0$;
while $r_k \neq 0$ **do**

end

Algorithm: Euclidean algorithm

Data: a, b
Result: $\gcd(a, b)$
$r_{-1} \leftarrow a$;
$r_0 \leftarrow b$;
$k \leftarrow 0$;
while $r_k \neq 0$ **do**
    $q_k \leftarrow \lfloor \frac{r_{k-1}}{r_k} \rfloor$;
end

Algorithm: Euclidean algorithm

Data: a, b
Result: $\gcd(a, b)$
$r_{-1} \leftarrow a$;
$r_0 \leftarrow b$;
$k \leftarrow 0$;
while $r_k \neq 0$ **do**
    $q_k \leftarrow \lfloor \frac{r_{k-1}}{r_k} \rfloor$;
    $r_{k+1} \leftarrow r_{k-1} - q_k r_k$;
end

Algorithm: Euclidean algorithm

Data: a, b
Result: $\gcd(a, b)$
$r_{-1} \leftarrow a$;
$r_0 \leftarrow b$;
$k \leftarrow 0$;
while $r_k \neq 0$ **do**
    $q_k \leftarrow \lfloor \frac{r_{k-1}}{r_k} \rfloor$;
    $r_{k+1} \leftarrow r_{k-1} - q_k r_k$;
    $k \leftarrow k + 1$;
end

Algorithm: Euclidean algorithm

Data: a, b
Result: $\gcd(a, b)$
$r_{-1} \leftarrow a$;
$r_0 \leftarrow b$;
$k \leftarrow 0$;
while $r_k \neq 0$ **do**
    $q_k \leftarrow \lfloor \frac{r_{k-1}}{r_k} \rfloor$;
    $r_{k+1} \leftarrow r_{k-1} - q_k r_k$;
    $k \leftarrow k + 1$;
end
$k \leftarrow k - 1$;

Algorithm: Euclidean algorithm

Data: a, b
Result: $\gcd(a, b)$
$r_{-1} \leftarrow a$;
$r_0 \leftarrow b$;
$k \leftarrow 0$;
while $r_k \neq 0$ **do**
  $q_k \leftarrow \lfloor \frac{r_{k-1}}{r_k} \rfloor$;
  $r_{k+1} \leftarrow r_{k-1} - q_k r_k$;
  $k \leftarrow k + 1$;
end
$k \leftarrow k - 1$;
return r_k

Algorithm: Euclidean algorithm

Back substitution (extended Euclidean algorithm)

- By *back substitution* in the Euclidean algorithm we can find integers x and y where

$$ax + by = d = r_k.$$

Back substitution (extended Euclidean algorithm)

- By *back substitution* in the Euclidean algorithm we can find integers x and y where

$$ax + by = d = r_k.$$

- Starting with the penultimate line in the algorithm, $r_{k-2} = r_{k-1}q_k + r_k$, we can compute

$$r_k = r_{k-2} - r_{k-1}q_k.$$

Back substitution (extended Euclidean algorithm)

- By *back substitution* in the Euclidean algorithm we can find integers x and y where

$$ax + by = d = r_k.$$

- Starting with the penultimate line in the algorithm, $r_{k-2} = r_{k-1}q_k + r_k$, we can compute

$$r_k = r_{k-2} - r_{k-1}q_k.$$

  Then we replace r_{k-1} in this equation from the next line up, $r_{k-1} = r_{k-3} - r_{k-2}q_{k-1}$ to get

Back substitution (extended Euclidean algorithm)

- By *back substitution* in the Euclidean algorithm we can find integers x and y where

$$ax + by = d = r_k.$$

- Starting with the penultimate line in the algorithm, $r_{k-2} = r_{k-1}q_k + r_k$, we can compute

$$r_k = r_{k-2} - r_{k-1}q_k.$$

Then we replace r_{k-1} in this equation from the next line up, $r_{k-1} = r_{k-3} - r_{k-2}q_{k-1}$ to get

$$\begin{aligned} r_k &= r_{k-2} - (r_{k-3} - r_{k-2}q_{k-1})q_k \\ &= r_{k-2}(1 + q_{k-1}q_k) - r_{k-3}q_k \end{aligned}$$

- Now we can use this equation to replace r_{k-2} from the line before that, and continue in the same way.

- Now we can use this equation to replace r_{k-2} from the line before that, and continue in the same way.
- Finally replacing r_1 by $r_1 = a - bq_1$ from the first line gives us r_k in terms of a multiple of a and a multiple of b.

- Now we can use this equation to replace r_{k-2} from the line before that, and continue in the same way.
- Finally replacing r_1 by $r_1 = a - bq_1$ from the first line gives us r_k in terms of a multiple of a and a multiple of b.
- We will be particularly interested in the case where $r_k = d = 1$.

Outline

Basic Number Theory
- Primes and Factorisation
- GCD and the Euclidean Algorithm
- Modular arithmetic

Groups

Finite Fields

Boolean Algebra

Modular arithmetic

Definition

b is a residue of a modulo n if $a - b = kn$ for some integer k.

$$a \equiv b \pmod{n} \iff a - b = kn.$$

Modular arithmetic

Definition

b is a residue of a modulo n if $a - b = kn$ for some integer k.

$$a \equiv b \pmod{n} \iff a - b = kn.$$

Given $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then

Modular arithmetic

Definition

b is a residue of a modulo n if $a - b = kn$ for some integer k.

$$a \equiv b \pmod{n} \iff a - b = kn.$$

Given $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then

1. $a + c \equiv b + d \pmod{n}$

Modular arithmetic

Definition

b is a residue of a modulo n if $a - b = kn$ for some integer k.

$$a \equiv b \pmod{n} \iff a - b = kn.$$

Given $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then

1. $a + c \equiv b + d \pmod{n}$
2. $ac \equiv bd \pmod{n}$

Modular arithmetic

Definition

b is a residue of a modulo n if $a - b = kn$ for some integer k.

$$a \equiv b \pmod{n} \iff a - b = kn.$$

Given $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then

1. $a + c \equiv b + d \pmod{n}$
2. $ac \equiv bd \pmod{n}$
3. $ka \equiv kb \pmod{n}$

Modular arithmetic

Definition

b is a residue of a modulo n if $a - b = kn$ for some integer k.

$$a \equiv b \pmod{n} \iff a - b = kn.$$

Given $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then

1. $a + c \equiv b + d \pmod{n}$
2. $ac \equiv bd \pmod{n}$
3. $ka \equiv kb \pmod{n}$

Note

This means we can always reduce the inputs modulo n *before* performing multiplication or addition.

Residue class

Definition

The set $\{r_0, r_1, \ldots, r_{n-1}\}$ is called a *complete set of residues* modulo n if, for every integer a, $a \equiv r_i \pmod{n}$ for exactly one r_i

Residue class

Definition

The set $\{r_0, r_1, \ldots, r_{n-1}\}$ is called a *complete set of residues* modulo n if, for every integer a, $a \equiv r_i \pmod{n}$ for exactly one r_i

- The numbers $\{0, 1, \ldots, n-1\}$ form a complete set of residues modulo n since we can write any a as

$$a = qn + r \text{ for } 0 \leq r \leq n-1$$

Residue class

Definition

The set $\{r_0, r_1, \ldots, r_{n-1}\}$ is called a *complete set of residues* modulo n if, for every integer a, $a \equiv r_i \pmod{n}$ for exactly one r_i

- The numbers $\{0, 1, \ldots, n-1\}$ form a complete set of residues modulo n since we can write any a as

$$a = qn + r \text{ for } 0 \leq r \leq n-1$$

- We usually choose this set as the complete set of residues and denote it:

$$\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$$

Notation: $a \bmod n$

We write

$$a \bmod n$$

to denote the unique value a' in the complete set of residues $\{0, 1, \ldots, n-1\}$ with

$$a' \equiv a \pmod{n}$$

In other words, $a \bmod n$ is the remainder after dividing a by n

Groups

A *group* is a set, G, with a binary operation, $\cdot$, satisfying the following conditions:

Groups

A *group* is a set, G, with a binary operation, $\cdot$, satisfying the following conditions:

- Closure: $a \cdot b \in G$ for all $a, b \in G$

Groups

A *group* is a set, G, with a binary operation, $\cdot$, satisfying the following conditions:

- Closure: $a \cdot b \in G$ for all $a, b \in G$
- Identity: there exists an element, 1, so that $a \cdot 1 = 1 \cdot a = a$ for all $a \in G$

Groups

A *group* is a set, G, with a binary operation, $\cdot$, satisfying the following conditions:

- Closure: $a \cdot b \in G$ for all $a, b \in G$
- Identity: there exists an element, 1, so that $a \cdot 1 = 1 \cdot a = a$ for all $a \in G$
- Inverse: for all $a \in G$ there exists an element, b, so that $a \cdot b = 1$ for all $a \in G$

Groups

A *group* is a set, G, with a binary operation, $\cdot$, satisfying the following conditions:

- Closure: $a \cdot b \in G$ for all $a, b \in G$
- Identity: there exists an element, 1, so that $a \cdot 1 = 1 \cdot a = a$ for all $a \in G$
- Inverse: for all $a \in G$ there exists an element, b, so that $a \cdot b = 1$ for all $a \in G$
- Associative: for all $a, b, c \in G$ that $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

Groups

A *group* is a set, G, with a binary operation, $\cdot$, satisfying the following conditions:

- Closure: $a \cdot b \in G$ for all $a, b \in G$
- Identity: there exists an element, 1, so that $a \cdot 1 = 1 \cdot a = a$ for all $a \in G$
- Inverse: for all $a \in G$ there exists an element, b, so that $a \cdot b = 1$ for all $a \in G$
- Associative: for all $a, b, c \in G$ that $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

We will only be looking at commutative (or abelian) groups which satisfy also:

- Commutative: for all $a, b \in G$ that $a \cdot b = b \cdot a$

Cyclic groups

- The *order of a group G*, often written $|G|$, is the number of elements in G

Cyclic groups

- The *order of a group* G, often written $|G|$, is the number of elements in G
- We write g^k to denote repeated application of g using the group operation — for example $g^3 = g \cdot g \cdot g$. The *order of an element* $g \in G$, often written $|g|$, is the smallest integer k with $g^k = 1$

Cyclic groups

- The *order of a group* G, often written $|G|$, is the number of elements in G
- We write g^k to denote repeated application of g using the group operation — for example $g^3 = g \cdot g \cdot g$. The *order of an element* $g \in G$, often written $|g|$, is the smallest integer k with $g^k = 1$
- A group element g is a *generator* for G if $|g| = |G|$

Cyclic groups

- The *order of a group G*, often written $|G|$, is the number of elements in G
- We write g^k to denote repeated application of g using the group operation — for example $g^3 = g \cdot g \cdot g$. The *order of an element $g \in G$*, often written $|g|$, is the smallest integer k with $g^k = 1$
- A group element g is a *generator* for G if $|g| = |G|$
- A group is *cyclic* if it has a generator

Cyclic groups

- The *order of a group* G, often written $|G|$, is the number of elements in G
- We write g^k to denote repeated application of g using the group operation — for example $g^3 = g \cdot g \cdot g$. The *order of an element* $g \in G$, often written $|g|$, is the smallest integer k with $g^k = 1$
- A group element g is a *generator* for G if $|g| = |G|$
- A group is *cyclic* if it has a generator

Cyclic groups are important in cryptography because if we construct a group G with large order then we can be sure that a generator g can also take on the same large number of values.

Computing inverses modulo n

- The inverse of a, if it exists, is a value x such that

$$ax \equiv 1 \pmod{n}$$

and is written $a^{-1} \bmod n$.

Computing inverses modulo n

- The inverse of a, if it exists, is a value x such that

$$ax \equiv 1 \pmod{n}$$

  and is written $a^{-1} \bmod n$.
- In cryptosystems we often need to find inverses so that we can decrypt, or undo, certain operations.

Computing inverses modulo n

- The inverse of a, if it exists, is a value x such that

$$ax \equiv 1 \pmod{n}$$

  and is written $a^{-1} \bmod n$.
- In cryptosystems we often need to find inverses so that we can decrypt, or undo, certain operations.

Theorem

Let $0 < a < n$. Then a has an inverse modulo n if and only if $\gcd(a, n) = 1$.

Modular inverses using Euclidean algorithm

- To find the inverse of a we can use the Euclidean algorithm which is very efficient

Modular inverses using Euclidean algorithm

- To find the inverse of a we can use the Euclidean algorithm which is very efficient
- Remember that we want to solve for x, given a:

$$ax \equiv 1 \pmod{n}$$

Modular inverses using Euclidean algorithm

- To find the inverse of a we can use the Euclidean algorithm which is very efficient
- Remember that we want to solve for x, given a:

$$ax \equiv 1 \pmod{n}$$

- Since $\gcd(a, n) = 1$ we can find $ax + ny = 1$ for integers x and y by Euclidean algorithm. Therefore:

$$\begin{aligned} ax &= 1 - ny \\ ax &\equiv 1 \pmod{n} \end{aligned}$$

$\mathbb{Z}_p^*$

- A complete set of residues modulo any prime p with the 0 removed forms a group under multiplication denoted $\mathbb{Z}_p^*$.

$\mathbb{Z}_p^*$

- A complete set of residues modulo any prime p with the 0 removed forms a group under multiplication denoted $\mathbb{Z}_p^*$.
- Some useful properties:

$\mathbb{Z}_p^*$

- A complete set of residues modulo any prime p with the 0 removed forms a group under multiplication denoted $\mathbb{Z}_p^*$.
- Some useful properties:
  - The order of $\mathbb{Z}_p^*$ is $p - 1$

$\mathbb{Z}_p^*$

- A complete set of residues modulo any prime p with the 0 removed forms a group under multiplication denoted $\mathbb{Z}_p^*$.
- Some useful properties:
  - The order of $\mathbb{Z}_p^*$ is $p - 1$
  - $\mathbb{Z}_p^*$ is cyclic

$\mathbb{Z}_p^*$

- A complete set of residues modulo any prime p with the 0 removed forms a group under multiplication denoted $\mathbb{Z}_p^*$.
- Some useful properties:
  - The order of $\mathbb{Z}_p^*$ is $p - 1$
  - $\mathbb{Z}_p^*$ is cyclic
  - $\mathbb{Z}_p^*$ has many generators in general

$\mathbb{Z}_p^*$

- A complete set of residues modulo any prime p with the 0 removed forms a group under multiplication denoted $\mathbb{Z}_p^*$.
- Some useful properties:
  - The order of $\mathbb{Z}_p^*$ is $p - 1$
  - $\mathbb{Z}_p^*$ is cyclic
  - $\mathbb{Z}_p^*$ has many generators in general
- $\mathbb{Z}_p^*$ can be represented as the multiplicative group of integers $\{1, 2, \ldots, p - 1\}$

Finding a generator of $\mathbb{Z}_p^*$

- A *generator* of $\mathbb{Z}_p^*$ is an element of order $p - 1$

Finding a generator of $\mathbb{Z}_p^*$

- A *generator* of $\mathbb{Z}_p^*$ is an element of order $p - 1$
- A general theorem of algebraic groups (Lagrange) implies that the order of any element must exactly divide $p - 1$

Finding a generator of $\mathbb{Z}_p^*$

- A *generator* of $\mathbb{Z}_p^*$ is an element of order $p - 1$
- A general theorem of algebraic groups (Lagrange) implies that the order of any element must exactly divide $p - 1$
- To find a generator of $\mathbb{Z}_p^*$ we can choose a value g and test it as follows:

Finding a generator of $\mathbb{Z}_p^*$

- A *generator* of $\mathbb{Z}_p^*$ is an element of order $p-1$
- A general theorem of algebraic groups (Lagrange) implies that the order of any element must exactly divide $p-1$
- To find a generator of $\mathbb{Z}_p^*$ we can choose a value g and test it as follows:
    1. compute all the distinct prime factors of $p-1$ and call them $f_1, f_2, \ldots, f_r$

Finding a generator of $\mathbb{Z}_p^*$

- A *generator* of $\mathbb{Z}_p^*$ is an element of order $p-1$
- A general theorem of algebraic groups (Lagrange) implies that the order of any element must exactly divide $p-1$
- To find a generator of $\mathbb{Z}_p^*$ we can choose a value g and test it as follows:
    1. compute all the distinct prime factors of $p-1$ and call them $f_1, f_2, \ldots, f_r$
    2. then g is a generator as long as $g^{(p-1)/f_i} \neq 1 \bmod p$ for $i = 1, 2, , \ldots, r$

Groups for composite modulus: $\mathbb{Z}_n^*$

- For any n, which may or may not be prime, we can define $\mathbb{Z}_n^*$ to be the group of residues which have an inverse under multiplication

Groups for composite modulus: $\mathbb{Z}_n^*$

- For any n, which may or may not be prime, we can define $\mathbb{Z}_n^*$ to be the group of residues which have an inverse under multiplication
- $\mathbb{Z}_n^*$ is a group but is not cyclic in general

Groups for composite modulus: $\mathbb{Z}_n^*$

- For any n, which may or may not be prime, we can define $\mathbb{Z}_n^*$ to be the group of residues which have an inverse under multiplication
- $\mathbb{Z}_n^*$ is a group but is not cyclic in general
- Finding the order of $\mathbb{Z}_n^*$ is difficult in general

Fields

A *field* is a set, F, with two binary operations, $+$ and $\cdot$, satisfying the following conditions:

Fields

A *field* is a set, F, with two binary operations, $+$ and $\cdot$, satisfying the following conditions:

- F is a commutative group under the + operation, with identity element denoted 0

Fields

A *field* is a set, F, with two binary operations, $+$ and $\cdot$, satisfying the following conditions:

- F is a commutative group under the + operation, with identity element denoted 0
- $F \setminus \{0\}$ is a commutative group under the $\cdot$ operation

Fields

A *field* is a set, F, with two binary operations, $+$ and $\cdot$, satisfying the following conditions:

- F is a commutative group under the + operation, with identity element denoted 0
- $F \setminus \{0\}$ is a commutative group under the $\cdot$ operation
- Distributive: for all $a, b, c \in F$:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

Finite fields

- For secure communications we are usually only interested in fields with a finite number of elements

Finite fields

- For secure communications we are usually only interested in fields with a finite number of elements
- A famous theorem says that finite fields exist of size p^n for any prime p and positive integer n, and that no finite field exists of other sizes

Finite fields

- For secure communications we are usually only interested in fields with a finite number of elements
- A famous theorem says that finite fields exist of size p^n for any prime p and positive integer n, and that no finite field exists of other sizes
- The most interesting cases for us are fields of size p for a prime p and fields of size 2^n for some integer n

Finite field $GF(p)$

- We often write $\mathbb{Z}_p$ instead of $GF(p)$

Finite field $GF(p)$

- We often write $\mathbb{Z}_p$ instead of $GF(p)$
- Multiplication and addition are done modulo p

Finite field $GF(p)$

- We often write $\mathbb{Z}_p$ instead of $GF(p)$
- Multiplication and addition are done modulo p
- Multiplicative group is exactly $\mathbb{Z}_p^*$

Finite field $GF(p)$

- We often write $\mathbb{Z}_p$ instead of $GF(p)$
- Multiplication and addition are done modulo p
- Multiplicative group is exactly $\mathbb{Z}_p^*$
- Later in the course we will see some public key encryption and digital signature schemes using $GF(p)$

Finite field $GF(2)$

- $GF(2)$ is the simplest field. It has only two elements.

Finite field $GF(2)$

- $GF(2)$ is the simplest field. It has only two elements.
- Addition is binary addition modulo 2. This is the same as the logical XOR (exclusive-OR) operation

Finite field $GF(2)$

- $GF(2)$ is the simplest field. It has only two elements.
- Addition is binary addition modulo 2. This is the same as the logical XOR (exclusive-OR) operation
- Since there is only one non-zero element we have a trivial multiplicative group with the single element 1.

Finite field $GF(2)$

- $GF(2)$ is the simplest field. It has only two elements.
- Addition is binary addition modulo 2. This is the same as the logical XOR (exclusive-OR) operation
- Since there is only one non-zero element we have a trivial multiplicative group with the single element 1.
- We often use XOR in cryptography, usually written $\oplus$. For bit strings a and b we write $a \oplus b$ for the bit-wise XOR. For example,

$$101 \oplus 011 = 110$$

Finite field $GF(2^n)$

- Arithmetic in these fields can be considered as polynomial arithmetic where the field elements are polynomials with binary coefficients

Finite field $GF(2^n)$

- Arithmetic in these fields can be considered as polynomial arithmetic where the field elements are polynomials with binary coefficients
- This allow us to equate any n-bit string with a polynomial in a natural way: for example 00101101 $\leftrightarrow x^5 + x^3 + x^2 + 1$

Finite field $GF(2^n)$

- Arithmetic in these fields can be considered as polynomial arithmetic where the field elements are polynomials with binary coefficients
- This allow us to equate any n-bit string with a polynomial in a natural way: for example 00101101 $\leftrightarrow x^5 + x^3 + x^2 + 1$
- The field can be represented in different ways by use of a *primitive polynomial* $m(x)$

Finite field $GF(2^n)$

- Arithmetic in these fields can be considered as polynomial arithmetic where the field elements are polynomials with binary coefficients
- This allow us to equate any n-bit string with a polynomial in a natural way: for example 00101101 $\leftrightarrow x^5 + x^3 + x^2 + 1$
- The field can be represented in different ways by use of a *primitive polynomial* $m(x)$
- Addition and multiplication is defined by polynomial addition and multiplication modulo $m(x)$

Finite field $GF(2^n)$

- Arithmetic in these fields can be considered as polynomial arithmetic where the field elements are polynomials with binary coefficients
- This allow us to equate any n-bit string with a polynomial in a natural way: for example 00101101 $\leftrightarrow x^5 + x^3 + x^2 + 1$
- The field can be represented in different ways by use of a *primitive polynomial* $m(x)$
- Addition and multiplication is defined by polynomial addition and multiplication modulo $m(x)$
- Polynomial division can be done very efficiently in hardware using shift registers

Arithmetic in $GF(2^8)$

- This field is used for calculations in the AES block cipher

Arithmetic in $GF(2^8)$

- This field is used for calculations in the AES block cipher
- To add two strings we add their coefficients modulo 2 (exclusive or)

Arithmetic in $GF(2^8)$

- This field is used for calculations in the AES block cipher
- To add two strings we add their coefficients modulo 2 (exclusive or)
- Multiplication is done with respect to a generator polynomial which for AES is chosen as:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

Arithmetic in $GF(2^8)$

- This field is used for calculations in the AES block cipher
- To add two strings we add their coefficients modulo 2 (exclusive or)
- Multiplication is done with respect to a generator polynomial which for AES is chosen as:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

- To multiply two strings we multiply them as polynomials and then take their remainder after dividing by $m(x)$

Boolean values

- A Boolean variable x takes the values of 1 or 0 representing *true* or *false*

Boolean values

- A Boolean variable x takes the values of 1 or 0 representing *true* or *false*
- A Boolean *function* is any function with range (output) in the set $\{0, 1\}$

Boolean values

- A Boolean variable x takes the values of 1 or 0 representing *true* or *false*
- A Boolean *function* is any function with range (output) in the set $\{0, 1\}$
- Boolean functions are often represented by a *truth table*

Boolean values

- A Boolean variable x takes the values of 1 or 0 representing *true* or *false*
- A Boolean *function* is any function with range (output) in the set $\{0, 1\}$
- Boolean functions are often represented by a *truth table*
- Each row in the table defines one possible input (tuple) and the associated output value

Boolean operations

- Logical AND: equivalent to multiplication modulo 2

x_1	x_2	$z = x_1 \wedge x_2$
1	1	1
1	0	0
0	1	0
0	0	0

Boolean operations

- Logical AND: equivalent to multiplication modulo 2

x_1	x_2	$z = x_1 \wedge x_2$
1	1	1
1	0	0
0	1	0
0	0	0

- Logical OR:

x_1	x_2	$z = x_1 \vee x_2$
1	1	1
1	0	1
0	1	1
0	0	0

Negation

Truth table

x	$\neg x$
1	0
0	1

We can also write $\neg x = x \wedge 1$