A little (Computational) Number Theory and Group Theory

Public key cryptographic constructions require some notions of number theory and group theory

Number theory and group theory are huge fields

We will only see what's needed for the following lectures

A little (Computational) Number Theory and Group Theory

Public key cryptographic constructions require some notions of number theory and group theory

Number theory and group theory are huge fields

We will only see what's needed for the following lectures

Differently from the pure mathematics approach, we will also be interested in **how quickly** we can solve various problems

In particular, we are interested in whether the problems at hand can be solved in **polynomial time**

Representing Integers

In the the word-RAM model we assume that each integer is stored in a single memory word

This is not a good model for problems that deal with large numbers

Representing Integers

In the the word-RAM model we assume that each integer is stored in a single memory word

This is not a good model for problems that deal with large numbers

Instead we use the logarithmic cost model

- Storing an integer n requires $\approx \log n$ bits
- An elementary operation involving integers with b bits requires time $\Theta(b)$

Representing Integers

In the the word-RAM model we assume that each integer is stored in a single memory word

This is not a good model for problems that deal with large numbers

Instead we use the logarithmic cost model

- Storing an integer n requires $\approx \log n$ bits
- An elementary operation involving integers with b bits requires time $\Theta(b)$

How do we store big (non-negative) integers in practice?

Representing Integers

In the the word-RAM model we assume that each integer is stored in a single memory word

This is not a good model for problems that deal with large numbers

Instead we use the logarithmic cost model

- Storing an integer n requires $\approx \log n$ bits
- An elementary operation involving integers with b bits requires time $\Theta(b)$

How do we store big (non-negative) integers in practice?

- Arrays of digits
- E.g., each entry in the array is a byte and stores a digit in base 256

74	241	176	81	206	92	108	31	42

Representing Integers

In the the word-RAM model we assume that each integer is stored in a single memory word

This is not a good model for problems that deal with large numbers

Instead we use the logarithmic cost model

- Storing an integer n requires $\approx \log n$ bits
- An elementary operation involving integers with b bits requires time $\Theta(b)$

How do we store big (non-negative) integers in practice?

- Arrays of digits
- E.g., each entry in the array is a byte and stores a digit in base 256

74	241	176	81	206	92	108	31	42

Encodes: $74 \cdot 256^8 + 241 \cdot 256^7 + 176 \cdot 256^6 + 81 \cdot 256^5 + 206 \cdot 256^4 + 92 \cdot 256^3 + 108 \cdot 256^2 + 31 \cdot 256 + 42$

$= 1\,382\,474\,571\,160\,304\,230\,186$

Representing Integers

In the the word-RAM model we assume that each integer is stored in a single memory word

This is not a good model for problems that deal with large numbers

Instead we use the logarithmic cost model

- Storing an integer n requires $\approx \log n$ bits
- An elementary operation involving integers with b bits requires time $\Theta(b)$

How do we store big (non-negative) integers in practice?

- Arrays of digits
- E.g., each entry in the array is a byte and stores a digit in base 256

74	241	176	81	206	92	108	31	42

Encodes: $74 \cdot 256^8 + 241 \cdot 256^7 + 176 \cdot 256^6 + 81 \cdot 256^5 + 206 \cdot 256^4 + 92 \cdot 256^3 + 108 \cdot 256^2 + 31 \cdot 256 + 42$

$= 1\,382\,474\,571\,160\,304\,230\,186$ Requires 71 bits to represent (does not fit in a 64-bit word)

Representing Integers

Recall the difference between polynomial-time and pseudopolynomial-time algorithms

Running times are measured as a function of the input length

An algorithm that takes an integer n and runs in time $\Theta(n)$ is **not** a polynomial-time algorithm

Representing Integers

Recall the difference between polynomial-time and pseudopolynomial-time algorithms

Running times are measured as a function of the input length

An algorithm that takes an integer n and runs in time $\Theta(n)$ is **not** a polynomial-time algorithm

- The running time is polynomial w.r.t. the **value** of the integer n
- It is not polynomial in the length of the input, i.e., the number of bits needed to represent n
- As a function of the input length η, the time complexity is $\Theta(2^\eta)$
- This is an **exponential-time** algorithm!

Representing Integers

The grade-school algorithms for addition and multiplication (over big integers) run in polynomial-time

- Adding n and m requires time $O(\log n + \log m)$
- Multiplying n and m requires time $O((\log n) \cdot (\log m))$ (can be improved)

Representing Integers

The grade-school algorithms for addition and multiplication (over big integers) run in polynomial-time

- Adding n and m requires time $O(\log n + \log m)$
- Multiplying n and m requires time $O((\log n) \cdot (\log m))$ (can be improved)

What about exponentiation?

- Given m and n, compute m^n

Representing Integers

The grade-school algorithms for addition and multiplication (over big integers) run in polynomial-time

- Adding n and m requires time $O(\log n + \log m)$
- Multiplying n and m requires time $O((\log n) \cdot (\log m))$ (can be improved)

What about exponentiation?

- Given m and n, compute m^n

Fix $m = 2$. Given n, compute 2^n.

- What's the size of the input?
- What's the size of the output?

Representing Integers

The grade-school algorithms for addition and multiplication (over big integers) run in polynomial-time

- Adding n and m requires time $O(\log n + \log m)$
- Multiplying n and m requires time $O((\log n) \cdot (\log m))$ (can be improved)

What about exponentiation?

- Given m and n, compute m^n

Fix $m = 2$. Given n, compute 2^n.

- What's the size of the input? $\Theta(\log n)$
- What's the size of the output? $\Theta(n)$

Representing Integers

The grade-school algorithms for addition and multiplication (over big integers) run in polynomial-time

- Adding n and m requires time $O(\log n + \log m)$
- Multiplying n and m requires time $O((\log n) \cdot (\log m))$ (can be improved)

What about exponentiation?

- Given m and n, compute m^n

Fix $m = 2$. Given n, compute 2^n.

- What's the size of the input? $\Theta(\log n)$
- What's the size of the output? $\Theta(n)$
- We cannot even write out the result in polynomial-time

Reminder: Modular arithmetic

Proposition: Let a be an integer and let N be a positive integer. There exist unique integers q, r for which $a = qN + r$ and $0 \leq r < N$.

Reminder: Modular arithmetic

Proposition: Let a be an integer and let N be a positive integer. There exist unique integers q, r for which $a = qN + r$ and $0 \leq r < N$.

- $a \bmod N = r$ by definition
- $a = b \pmod{N}$ is a shorthand for $(a \bmod N) = (b \bmod N)$

Reminder: Modular arithmetic

Proposition: Let a be an integer and let N be a positive integer. There exist unique integers q, r for which $a = qN + r$ and $0 \leq r < N$.

- $a \bmod N = r$ by definition
- $a = b \pmod{N}$ is a shorthand for $(a \bmod N) = (b \bmod N)$

We can reduce intermediate values during computation of additions and products:

- $(a + b) \bmod N = ((a \bmod N) + (b \bmod N)) \bmod N$
- $(a \cdot b) \bmod N = ((a \bmod N) \cdot (b \bmod N)) \bmod N$

Reminder: Modular arithmetic

Proposition: Let a be an integer and let N be a positive integer. There exist unique integers q, r for which $a = qN + r$ and $0 \leq r < N$.

- $a \bmod N = r$ by definition
- $a = b \pmod N$ is a shorthand for $(a \bmod N) = (b \bmod N)$

We can reduce intermediate values during computation of additions and products:

- $(a + b) \bmod N = ((a \bmod N) + (b \bmod N)) \bmod N$
- $(a \cdot b) \bmod N = ((a \bmod N) \cdot (b \bmod N)) \bmod N$

Example:

$(7236782 \cdot 23392301) \bmod 100$

Reminder: Modular arithmetic

Proposition: Let a be an integer and let N be a positive integer. There exist unique integers q, r for which $a = qN + r$ and $0 \leq r < N$.

- $a \bmod N = r$ by definition
- $a = b \pmod{N}$ is a shorthand for $(a \bmod N) = (b \bmod N)$

We can reduce intermediate values during computation of additions and products:

- $(a + b) \bmod N = ((a \bmod N) + (b \bmod N)) \bmod N$
- $(a \cdot b) \bmod N = ((a \bmod N) \cdot (b \bmod N)) \bmod N$

Example:

$(7236782 \cdot 23392301) \bmod 100 = (82 \cdot 1) \bmod 100 = 82$

Reminder: Modular arithmetic

There are polynomial-time algorithms for:

- Modular reduction (given a and N, compute $a \bmod N$)
- Modular addition
- Modular multiplication

Reminder: Modular arithmetic

There are polynomial-time algorithms for:

- Modular reduction (given a and N, compute $a \bmod N$)
- Modular addition
- Modular multiplication

What about modular exponentiation?

Given an integer $N > 0$ and $a, b \in \{0, \ldots, N-1\}$ compute $a^b \bmod N$.

Reminder: Modular arithmetic

There are polynomial-time algorithms for:

- Modular reduction (given a and N, compute $a \bmod N$)
- Modular addition
- Modular multiplication

What about modular exponentiation?

Given an integer $N > 0$ and $a, b \in \{0, \ldots, N-1\}$ compute $a^b \bmod N$.

- We cannot simply compute a^b and then perform modular reduction.

Reminder: Modular arithmetic

There are polynomial-time algorithms for:

- Modular reduction (given a and N, compute $a \bmod N$)
- Modular addition
- Modular multiplication

What about modular exponentiation?

Given an integer $N > 0$ and $a, b \in \{0, \ldots, N-1\}$ compute $a^b \bmod N$.

- We cannot simply compute a^b and then perform modular reduction.

Divide and conquer:

- If $b = 0$ return 1

Reminder: Modular arithmetic

There are polynomial-time algorithms for:

- Modular reduction (given a and N, compute $a \bmod N$)
- Modular addition
- Modular multiplication

What about modular exponentiation?

Given an integer $N > 0$ and $a, b \in \{0, \ldots, N-1\}$ compute $a^b \bmod N$.

- We cannot simply compute a^b and then perform modular reduction.

Divide and conquer:

- If $b = 0$ return 1
- If b is even: recursively compute $x = a^{b/2} \mod N$ and return $(x \cdot x) \mod N$

Reminder: Modular arithmetic

There are polynomial-time algorithms for:

- Modular reduction (given a and N, compute $a \bmod N$)
- Modular addition
- Modular multiplication

What about modular exponentiation?

Given an integer $N > 0$ and $a, b \in \{0, \ldots, N-1\}$ compute $a^b \bmod N$.

- We cannot simply compute a^b and then perform modular reduction.

Divide and conquer:

- If $b = 0$ return 1
- If b is even: recursively compute $x = a^{b/2} \mod N$ and return $(x \cdot x) \mod N$
- If b is odd: recursively compute $x = a^{(b-1)/2} \mod N$ and return $(x \cdot x \cdot a) \mod N$

Reminder: Modular arithmetic

There are polynomial-time algorithms for:

- Modular reduction (given a and N, compute $a \bmod N$)
- Modular addition
- Modular multiplication

What about modular exponentiation?

Given an integer $N > 0$ and $a, b \in \{0, \ldots, N-1\}$ compute $a^b \bmod N$.

- We cannot simply compute a^b and then perform modular reduction.

Divide and conquer:

- If $b = 0$ return 1
- If b is even: recursively compute $x = a^{b/2} \mod N$ and return $(x \cdot x) \mod N$
- If b is odd: recursively compute $x = a^{(b-1)/2} \mod N$ and return $(x \cdot x \cdot a) \mod N$

Recusion depth: $O(\log b)$

The non-recursive part of each call involves a constant number of polynomial-time operations

Reminder: Modular arithmetic

A non-negative integer b is invertible modulo $N \geq 1$ if there exists an integer a such that $ab = ba = 1 \pmod{N}$

Reminder: Modular arithmetic

A non-negative integer b is invertible modulo $N \geq 1$ if there exists an integer a such that $ab = ba = 1 \pmod{N}$

If b is invertible and $xb = yb \pmod{N}$ then $x = y \pmod{N}$

Reminder: Modular arithmetic

A non-negative integer b is invertible modulo $N \geq 1$ if there exists an integer a such that $ab = ba = 1 \pmod N$

If b is invertible and $xb = yb \pmod N$ then $x = y \pmod N$

Proof: Let a be an inverse of b. $\quad x = xba = yba = y \pmod N$ $\square$

Reminder: Modular arithmetic

A non-negative integer b is invertible modulo $N \geq 1$ if there exists an integer a such that $ab = ba = 1 \pmod{N}$

If b is invertible and $xb = yb \pmod{N}$ then $x = y \pmod{N}$

Proof: Let a be an inverse of b. $x = xba = yba = y \pmod{N}$ $\square$

- This is not necessarily true if b is not invertible: $1 \cdot 2 = 3 \cdot 2 \pmod{4}$ but $1 \neq 3 \pmod{4}$

Reminder: Modular arithmetic

A non-negative integer b is invertible modulo $N \geq 1$ if there exists an integer a such that $ab = ba = 1 \pmod{N}$

If b is invertible and $xb = yb \pmod{N}$ then $x = y \pmod{N}$

Proof: Let a be an inverse of b. $x = xba = yba = y \pmod{N}$ $\square$

- This is not necessarily true if b is not invertible: $1 \cdot 2 = 3 \cdot 2 \pmod{4}$ but $1 \neq 3 \pmod{4}$
- If b is invertible, then it has a unique inverse $a \in \{0, \ldots, N-1\}$.

Reminder: Modular arithmetic

A non-negative integer b is invertible modulo $N \geq 1$ if there exists an integer a such that $ab = ba = 1 \pmod{N}$

If b is invertible and $xb = yb \pmod{N}$ then $x = y \pmod{N}$

Proof: Let a be an inverse of b. $x = xba = yba = y \pmod{N}$ $\square$

- This is not necessarily true if b is not invertible: $1 \cdot 2 = 3 \cdot 2 \pmod{4}$ but $1 \neq 3 \pmod{4}$
- If b is invertible, then it has a unique inverse $a \in \{0, \ldots, N-1\}$.

  Proof: Let a and a' be inverses of b. $ab = 1 = a'b \pmod{N} \implies a = a' \pmod{N}$ $\square$

Reminder: Modular arithmetic

A non-negative integer b is invertible modulo $N \geq 1$ if there exists an integer a such that $ab = ba = 1 \pmod{N}$

If b is invertible and $xb = yb \pmod{N}$ then $x = y \pmod{N}$

Proof: Let a be an inverse of b. $\quad x = xba = yba = y \pmod{N}$ $\square$

- This is not necessarily true if b is not invertible: $\quad 1 \cdot 2 = 3 \cdot 2 \pmod{4}$ but $1 \neq 3 \pmod{4}$

- If b is invertible, then it has a unique inverse $a \in \{0, \ldots, N-1\}$.

  Proof: Let a and a' be inverses of b. $\quad ab = 1 = a'b \pmod{N} \implies a = a' \pmod{N}$ $\square$

- We denote the unique inverse of an invertible element b with $b^{-1} \pmod{N}$

Reminder: Modular arithmetic

A non-negative integer b is invertible modulo $N \geq 1$ if there exists an integer a such that $ab = ba = 1 \pmod{N}$

If b is invertible and $xb = yb \pmod{N}$ then $x = y \pmod{N}$

Proof: Let a be an inverse of b. $x = xba = yba = y \pmod{N}$ □

- This is not necessarily true if b is not invertible: $1 \cdot 2 = 3 \cdot 2 \pmod{4}$ but $1 \neq 3 \pmod{4}$
- If b is invertible, then it has a unique inverse $a \in \{0, \ldots, N-1\}$.

  Proof: Let a and a' be inverses of b. $ab = 1 = a'b \pmod{N} \implies a = a' \pmod{N}$ □
- We denote the unique inverse of an invertible element b with $b^{-1} \pmod{N}$

Two integers a, b are coprime if $\gcd(a, b) = 1$

Theorem: b is invertible modulo N if and only if b and N are coprime

Bézout's identity

Bézout's identity: Let a, b be positive integers. Then there exist integers X, Y such that $Xa + Yb = \text{gcd}(a, b)$. Furthermore, $\text{gcd}(a, b)$ is the smallest positive integer that can be expressed in this way.

Bézout's identity

Bézout's identity: Let a, b be positive integers. Then there exist integers X, Y such that $Xa + Yb = \text{gcd}(a, b)$. Furthermore, $\text{gcd}(a, b)$ is the smallest positive integer that can be expressed in this way.

The **extended Euclidean algorithm** is able to compute $\text{gcd}(a, b)$ and the integers X and Y in polynomial time.

Bézout's identity

Bézout's identity: Let a, b be positive integers. Then there exist integers X, Y such that $Xa + Yb = \text{gcd}(a, b)$. Furthermore, $\text{gcd}(a, b)$ is the smallest positive integer that can be expressed in this way.

The **extended Euclidean algorithm** is able to compute $\text{gcd}(a, b)$ and the integers X and Y in polynomial time.

If b is invertible modulo N how do we (efficiently) find b^{-1}?

Bézout's identity

Bézout's identity: Let a, b be positive integers. Then there exist integers X, Y such that $Xa + Yb = \text{gcd}(a, b)$. Furthermore, $\text{gcd}(a, b)$ is the smallest positive integer that can be expressed in this way.

The **extended Euclidean algorithm** is able to compute $\text{gcd}(a, b)$ and the integers X and Y in polynomial time.

If b is invertible modulo N how do we (efficiently) find b^{-1}?

- Let X and Y be such that $XN + Yb = \text{gcd}(N, b) = 1$

Bézout's identity

Bézout's identity: Let a, b be positive integers. Then there exist integers X, Y such that $Xa + Yb = \text{gcd}(a, b)$. Furthermore, $\text{gcd}(a, b)$ is the smallest positive integer that can be expressed in this way.

The **extended Euclidean algorithm** is able to compute $\text{gcd}(a, b)$ and the integers X and Y in polynomial time.

If b is invertible modulo N how do we (efficiently) find b^{-1}?

- Let X and Y be such that $XN + Yb = \text{gcd}(N, b) = 1$
- Since $XN + Yb = 1$ we have $0 + Yb = 1 \pmod{N} \implies Y$ is an inverse for b.

Group Theory: Some Definitions

A group is a pair $(G, \circ)$, where G is a set, $\circ : G \times G \to G$ is a binary operation, and the following conditions are satisfied:

Group Theory: Some Definitions

A group is a pair $(G, \circ)$, where G is a set, $\circ : G \times G \to G$ is a binary operation, and the following conditions are satisfied:

- **Existence of an identity:** There is an element $e \in G$ such that $e \circ g = g \circ e = g$ for all $g \in G$.

Group Theory: Some Definitions

A group is a pair $(G, \circ)$, where G is a set, $\circ : G \times G \to G$ is a binary operation, and the following conditions are satisfied:

- **Existence of an identity:** There is an element $e \in G$ such that $e \circ g = g \circ e = g$ for all $g \in G$.

- **Associativity:** For all $a, b, c \in G$, it holds that $(a \circ b) \circ c = a \circ (b \circ c)$

Group Theory: Some Definitions

A group is a pair $(G, \circ)$, where G is a set, $\circ : G \times G \to G$ is a binary operation, and the following conditions are satisfied:

- **Existence of an identity:** There is an element $e \in G$ such that $e \circ g = g \circ e = g$ for all $g \in G$.
- **Associativity:** For all $a, b, c \in G$, it holds that $(a \circ b) \circ c = a \circ (b \circ c)$
- **Existence of inverses:** For all $g \in G$, there is some $h \in G$ such that $g \circ h = h \circ g = e$

Group Theory: Some Definitions

A group is a pair $(G, \circ)$, where G is a set, $\circ : G \times G \to G$ is a binary operation, and the following conditions are satisfied:

- **Existence of an identity:** There is an element $e \in G$ such that $e \circ g = g \circ e = g$ for all $g \in G$.

- **Associativity:** For all $a, b, c \in G$, it holds that $(a \circ b) \circ c = a \circ (b \circ c)$

- **Existence of inverses:** For all $g \in G$, there is some $h \in G$ such that $g \circ h = h \circ g = e$

Some consequences:

- Exactly one element e satisfies the first condition. This element is called **the** identity element.

Group Theory: Some Definitions

A group is a pair $(G, \circ)$, where G is a set, $\circ : G \times G \to G$ is a binary operation, and the following conditions are satisfied:

- **Existence of an identity:** There is an element $e \in G$ such that $e \circ g = g \circ e = g$ for all $g \in G$.
- **Associativity:** For all $a, b, c \in G$, it holds that $(a \circ b) \circ c = a \circ (b \circ c)$
- **Existence of inverses:** For all $g \in G$, there is some $h \in G$ such that $g \circ h = h \circ g = e$

Some consequences:

- Exactly one element e satisfies the first condition. This element is called **the** identity element.

  Proof: Let $e, f \in G$ be identity elements. We must have $e = f$. Indeed: $e = e \circ f = f$. $\square$

Group Theory: Some Definitions

A group is a pair $(G, \circ)$, where G is a set, $\circ : G \times G \to G$ is a binary operation, and the following conditions are satisfied:

- **Existence of an identity:** There is an element $e \in G$ such that $e \circ g = g \circ e = g$ for all $g \in G$.
- **Associativity:** For all $a, b, c \in G$, it holds that $(a \circ b) \circ c = a \circ (b \circ c)$
- **Existence of inverses:** For all $g \in G$, there is some $h \in G$ such that $g \circ h = h \circ g = e$

Some consequences:

- Exactly one element e satisfies the first condition. This element is called **the** identity element.

  Proof: Let $e, f \in G$ be identity elements. We must have $e = f$. Indeed: $e = e \circ f = f$. $\square$

- Each element has a unique inverse.

Group Theory: Some Definitions

A group is a pair $(G, \circ)$, where G is a set, $\circ : G \times G \to G$ is a binary operation, and the following conditions are satisfied:

- **Existence of an identity:** There is an element $e \in G$ such that $e \circ g = g \circ e = g$ for all $g \in G$.
- **Associativity:** For all $a, b, c \in G$, it holds that $(a \circ b) \circ c = a \circ (b \circ c)$
- **Existence of inverses:** For all $g \in G$, there is some $h \in G$ such that $g \circ h = h \circ g = e$

Some consequences:

- Exactly one element e satisfies the first condition. This element is called **the** identity element.

  Proof: Let $e, f \in G$ be identity elements. We must have $e = f$. Indeed: $e = e \circ f = f$. $\square$

- Each element has a unique inverse.

  Proof: If g has inverses h and h' then: $h = h \circ e = h \circ (g \circ h') = (h \circ g) \circ h' = e \circ h' = h'$. $\square$

Group Theory: Some Definitions

A group is a pair $(G, \circ)$, where G is a set, $\circ : G \times G \to G$ is a binary operation, and the following conditions are satisfied:

- **Existence of an identity:** There is an element $e \in G$ such that $e \circ g = g \circ e = g$ for all $g \in G$.
- **Associativity:** For all $a, b, c \in G$, it holds that $(a \circ b) \circ c = a \circ (b \circ c)$
- **Existence of inverses:** For all $g \in G$, there is some $h \in G$ such that $g \circ h = h \circ g = e$

Some consequences:

- Exactly one element e satisfies the first condition. This element is called **the** identity element.

  Proof: Let $e, f \in G$ be identity elements. We must have $e = f$. Indeed: $e = e \circ f = f$. □
- Each element has a unique inverse.

  Proof: If g has inverses h and h' then: $h = h \circ e = h \circ (g \circ h') = (h \circ g) \circ h' = e \circ h' = h'$. □

The **order** of a group is the cardinality $|G|$ of G. If G is a finite set, then the group is **finite**.

Group Theory: Some Definitions

A group is a pair $(G, \circ)$, where G is a set, $\circ : G \times G \to G$ is a binary operation, and the following conditions are satisfied:

- **Existence of an identity:** There is an element $e \in G$ such that $e \circ g = g \circ e = g$ for all $g \in G$.
- **Associativity:** For all $a, b, c \in G$, it holds that $(a \circ b) \circ c = a \circ (b \circ c)$
- **Existence of inverses:** For all $g \in G$, there is some $h \in G$ such that $g \circ h = h \circ g = e$

Some consequences:

- Exactly one element e satisfies the first condition. This element is called **the** identity element.

  Proof: Let $e, f \in G$ be identity elements. We must have $e = f$. Indeed: $e = e \circ f = f$. □

- Each element has a unique inverse.

  Proof: If g has inverses h and h' then: $h = h \circ e = h \circ (g \circ h') = (h \circ g) \circ h' = e \circ h' = h'$. □

The **order** of a group is the cardinality $|G|$ of G. If G is a finite set, then the group is **finite**.

If the operation $\circ$ is commutative (i.e., $a \circ b = b \circ a$ for all $a, b \in G$) then the group is **Abelian**.

Examples

Which of these are groups?

- $(\{0\}, +)$
- $(\mathbb{Z}, +)$
- $(\mathbb{Z}, \cdot)$
- $(\mathbb{Q} \setminus \{0\}, +)$
- $(\mathbb{Q} \setminus \{0\}, \cdot)$
- $(\{1, \ldots, N-1\}, \circ)$ where $a \circ b = ab \bmod N$
- $(\{0,1\}^n, \oplus)$

Examples

Which of these are groups?

- $(\{0\}, +)$ Group
- $(\mathbb{Z}, +)$
- $(\mathbb{Z}, \cdot)$
- $(\mathbb{Q} \setminus \{0\}, +)$
- $(\mathbb{Q} \setminus \{0\}, \cdot)$
- $(\{1, \ldots, N-1\}, \circ)$ where $a \circ b = ab \bmod N$
- $(\{0,1\}^n, \oplus)$

Examples

Which of these are groups?

- $(\{0\}, +)$ Group
- $(\mathbb{Z}, +)$ Group
- $(\mathbb{Z}, \cdot)$
- $(\mathbb{Q} \setminus \{0\}, +)$
- $(\mathbb{Q} \setminus \{0\}, \cdot)$
- $(\{1, \ldots, N-1\}, \circ)$ where $a \circ b = ab \bmod N$
- $(\{0,1\}^n, \oplus)$

Examples

Which of these are groups?

- $(\{0\}, +)$ Group
- $(\mathbb{Z}, +)$ Group
- $(\mathbb{Z}, \cdot)$ Not a group. No inverse for 0, no inverse for 2, ...
- $(\mathbb{Q} \setminus \{0\}, +)$
- $(\mathbb{Q} \setminus \{0\}, \cdot)$
- $(\{1, \ldots, N-1\}, \circ)$ where $a \circ b = ab \bmod N$
- $(\{0,1\}^n, \oplus)$

Examples

Which of these are groups?

- $(\{0\}, +)$ Group
- $(\mathbb{Z}, +)$ Group
- $(\mathbb{Z}, \cdot)$ Not a group. No inverse for 0, no inverse for 2, ...
- $(\mathbb{Q} \setminus \{0\}, +)$ Not a group. Not closed. $1 + (-1) = 0 \notin \mathbb{Q} \setminus \{0\}$
- $(\mathbb{Q} \setminus \{0\}, \cdot)$
- $(\{1, \ldots, N-1\}, \circ)$ where $a \circ b = ab \bmod N$
- $(\{0, 1\}^n, \oplus)$

Examples

Which of these are groups?

- $(\{0\}, +)$ Group
- $(\mathbb{Z}, +)$ Group
- $(\mathbb{Z}, \cdot)$ Not a group. No inverse for 0, no inverse for 2, ...
- $(\mathbb{Q} \setminus \{0\}, +)$ Not a group. Not closed. $1 + (-1) = 0 \notin \mathbb{Q} \setminus \{0\}$
- $(\mathbb{Q} \setminus \{0\}, \cdot)$ Group
- $(\{1, \ldots, N-1\}, \circ)$ where $a \circ b = ab \bmod N$
- $(\{0, 1\}^n, \oplus)$

Examples

Which of these are groups?

- $(\{0\}, +)$ Group
- $(\mathbb{Z}, +)$ Group
- $(\mathbb{Z}, \cdot)$ Not a group. No inverse for 0, no inverse for 2, ...
- $(\mathbb{Q} \setminus \{0\}, +)$ Not a group. Not closed. $1 + (-1) = 0 \notin \mathbb{Q} \setminus \{0\}$
- $(\mathbb{Q} \setminus \{0\}, \cdot)$ Group
- $(\{1, \ldots, N-1\}, \circ)$ where $a \circ b = ab \bmod N$

  Depends on N.
  In general not a group (no inverses).
- $(\{0,1\}^n, \oplus)$

Examples

Which of these are groups?

- $(\{0\}, +)$ Group
- $(\mathbb{Z}, +)$ Group
- $(\mathbb{Z}, \cdot)$ Not a group. No inverse for 0, no inverse for 2, ...
- $(\mathbb{Q} \setminus \{0\}, +)$ Not a group. Not closed. $1 + (-1) = 0 \notin \mathbb{Q} \setminus \{0\}$
- $(\mathbb{Q} \setminus \{0\}, \cdot)$ Group
- $(\{1, \ldots, N-1\}, \circ)$ where $a \circ b = ab \bmod N$ Depends on N. In general not a group (no inverses).
- $(\{0,1\}^n, \oplus)$ Group

Examples

Which of these are groups?

- $(\{0\}, +)$ Group
- $(\mathbb{Z}, +)$ Group
- $(\mathbb{Z}, \cdot)$ Not a group. No inverse for 0, no inverse for 2, ...
- $(\mathbb{Q} \setminus \{0\}, +)$ Not a group. Not closed. $1 + (-1) = 0 \notin \mathbb{Q} \setminus \{0\}$
- $(\mathbb{Q} \setminus \{0\}, \cdot)$ Group
- $(\{1, \ldots, N-1\}, \circ)$ where $a \circ b = ab \bmod N$

  Depends on N.
  In general not a group (no inverses).
- $(\{0,1\}^n, \oplus)$ Group

In the following we will only consider finite Abelian groups!

Group Theory: Additive and Multiplicative Notations

Depending on the context, it might be convenient to write the group operation as $+$ or as $\cdot$

Keep in mind that it is still **not** a regular addition or multiplication, but the group operation instead!

Additive notation	Multiplicative notation

Group Theory: Additive and Multiplicative Notations

Depending on the context, it might be convenient to write the group operation as $+$ or as $\cdot$

Keep in mind that it is still **not** a regular addition or multiplication, but the group operation instead!

	Additive notation	Multiplicative notation
Group operation applied to $a, b \in G$:	$a + b$	$a \cdot b$ or just ab

Group Theory: Additive and Multiplicative Notations

Depending on the context, it might be convenient to write the group operation as $+$ or as $\cdot$

Keep in mind that it is still **not** a regular addition or multiplication, but the group operation instead!

	Additive notation	Multiplicative notation
Group operation applied to $a, b \in G$:	$a + b$	$a \cdot b$ or just ab
Identity element e:	0	1

Group Theory: Additive and Multiplicative Notations

Depending on the context, it might be convenient to write the group operation as $+$ or as $\cdot$

Keep in mind that it is still **not** a regular addition or multiplication, but the group operation instead!

	Additive notation	Multiplicative notation
Group operation applied to $a, b \in G$:	$a + b$	$a \cdot b$ or just ab
Identity element e:	0	1
Inverse of an element $g \in G$	$-g$ $a - b$ is a shorthand for $a + (-b)$	g^{-1}

Group Theory: Additive and Multiplicative Notations

Depending on the context, it might be convenient to write the group operation as $+$ or as $\cdot$

Keep in mind that it is still **not** a regular addition or multiplication, but the group operation instead!

		Additive notation	Multiplicative notation
Group operation applied to $a, b \in G$:		$a + b$	$a \cdot b$ or just ab
Identity element e:		0	1
Inverse of an element $g \in G$		$-g$ $a - b$ is a shorthand for $a + (-b)$	g^{-1}
Group exponentiation:			
for $m \in \mathbb{N}$ and $g \in G$:	$\underbrace{g \circ g \circ \cdots \circ g}_{m \text{ times}}$	mg or $m \cdot g$	g^m

Group Theory: Additive and Multiplicative Notations

Depending on the context, it might be convenient to write the group operation as $+$ or as $\cdot$

Keep in mind that it is still **not** a regular addition or multiplication, but the group operation instead!

		Additive notation	Multiplicative notation
Group operation applied to $a, b \in G$:		$a + b$	$a \cdot b$ or just ab
Identity element e:		0	1
Inverse of an element $g \in G$		$-g$ $a - b$ is a shorthand for $a + (-b)$	g^{-1}
Group exponentiation:			
for $m \in \mathbb{N}$ and $g \in G$:	$\underbrace{g \circ g \circ \cdots \circ g}_{m \text{ times}}$	mg or $m \cdot g$ $0g = 0$	g^m $g^0 = 1$

Group Theory: Additive and Multiplicative Notations

Depending on the context, it might be convenient to write the group operation as $+$ or as $\cdot$

Keep in mind that it is still **not** a regular addition or multiplication, but the group operation instead!

	Additive notation	Multiplicative notation
Group operation applied to $a, b \in G$:	$a + b$	$a \cdot b$ or just ab
Identity element e:	0	1
Inverse of an element $g \in G$	$-g$ $a - b$ is a shorthand for $a + (-b)$	g^{-1}

Group exponentiation:

	Additive notation	Multiplicative notation
for $m \in \mathbb{N}$ and $g \in G$: $\underbrace{g \circ g \circ \cdots \circ g}_{m \text{ times}}$	mg or $m \cdot g$	g^m
	Integer $0g = 0$ Group element	Integer $g^0 = 1$ Group element

Group Theory: Additive and Multiplicative Notations

Depending on the context, it might be convenient to write the group operation as $+$ or as $\cdot$

Keep in mind that it is still **not** a regular addition or multiplication, but the group operation instead!

	Additive notation	Multiplicative notation
Group operation applied to $a, b \in G$:	$a + b$	$a \cdot b$ or just ab
Identity element e:	0	1
Inverse of an element $g \in G$	$-g$ $a - b$ is a shorthand for $a + (-b)$	g^{-1}
Group exponentiation:		
for $m \in \mathbb{N}$ and $g \in G$: $\underbrace{g \circ g \circ \cdots \circ g}_{m \text{ times}}$	mg or $m \cdot g$	g^m
	$0g = 0$	$g^0 = 1$
$\underbrace{g^{-1} \circ g^{-1} \circ \cdots \circ g^{-1}}_{m \text{ times}}$	$(-m)g = m(-g) = -(mg)$	$g^{-m} = (g^{-1})^m = (g^m)^{-1}$

Group Theory: Efficient Group Exponentiation

Given $g \in G$ and an integer b, how do we compute g^b?

Group Theory: Efficient Group Exponentiation

Given $g \in G$ and an integer b, how do we compute g^b?

(Essentially) the same approach of modular exponentiation works

Group Theory: Efficient Group Exponentiation

Given $g \in G$ and an integer b, how do we compute g^b?

(Essentially) the same approach of modular exponentiation works

If $b < 0$ then compute $h = g^{-1}$ and then $h^{|b|}$. For $b \geq 0$:

Divide and conquer:

- If $b = 0$ return 1
- If b is even: recursively compute $x = g^{b/2}$ and return $x \cdot x$
- If b is odd: recursively compute $x = g^{(b-1)/2}$ and return $x \cdot x \cdot g$

Group Theory: Efficient Group Exponentiation

Given $g \in G$ and an integer b, how do we compute g^b?

(Essentially) the same approach of modular exponentiation works

If $b < 0$ then compute $h = g^{-1}$ and then $h^{|b|}$. For $b \geq 0$:

Divide and conquer:

- If $b = 0$ return 1
- If b is even: recursively compute $x = g^{b/2}$ and return $x \cdot x$
- If b is odd: recursively compute $x = g^{(b-1)/2}$ and return $x \cdot x \cdot g$

If the group operation can be computed in polynomial-time, then group exponentiation can be performed in polynomial-time

The group $\mathbb{Z}_N$ under addition modulo N

Let $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$. The set $\mathbb{Z}_N$ is an Abelian group under addition modulo N.

The group $\mathbb{Z}_N$ under addition modulo N

Let $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$. The set $\mathbb{Z}_N$ is an Abelian group under addition modulo N.

- **Closure:** follows from the fact that addition is performed modulo N.

The group $\mathbb{Z}_N$ under addition modulo N

Let $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$. The set $\mathbb{Z}_N$ is an Abelian group under addition modulo N.

- **Closure:** follows from the fact that addition is performed modulo N.
- **Existence of the identity:** The identity element is 0. Indeed $g + 0 = 0 + g = g \pmod{N}$.

The group $\mathbb{Z}_N$ under addition modulo N

Let $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$. The set $\mathbb{Z}_N$ is an Abelian group under addition modulo N.

- **Closure:** follows from the fact that addition is performed modulo N.
- **Existence of the identity:** The identity element is 0. Indeed $g + 0 = 0 + g = g \pmod{N}$.
- **Associativity, Commutativity:** Trivial from addition over the integers.

The group $\mathbb{Z}_N$ under addition modulo N

Let $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$. The set $\mathbb{Z}_N$ is an Abelian group under addition modulo N.

- **Closure:** follows from the fact that addition is performed modulo N.
- **Existence of the identity:** The identity element is 0. Indeed $g + 0 = 0 + g = g \pmod{N}$.
- **Associativity, Commutativity:** Trivial from addition over the integers.
- **Existence of inverses:** The inverse of g is $-g \bmod N$ (recall that $-g \bmod N$ is an integer between 0 and $N-1$).

The group $\mathbb{Z}_N^*$ under multiplication modulo N

Let $\mathbb{Z}_N^* = \{0 < x < N \mid \gcd(x, N) = 1\}$. The set $\mathbb{Z}_N^*$ is an Abelian group under multiplication modulo N.

Intuition: We are removing the “problematic” elements (i.e., those without an inverse) from $\{1, \ldots, N\}$,

The group $\mathbb{Z}_N^*$ under multiplication modulo N

Let $\mathbb{Z}_N^* = \{0 < x < N \mid \text{gcd}(x, N) = 1\}$. The set $\mathbb{Z}_N^*$ is an Abelian group under multiplication modulo N.

Intuition: We are removing the "problematic" elements (i.e., those without an inverse) from $\{1, \ldots, N\}$,

- **Existence of the identity:** The identity element is $1 \in \mathbb{Z}_N^*$. Indeed $g \cdot 1 = 1 \cdot g = g \pmod{N}$.

The group $\mathbb{Z}_N^*$ under multiplication modulo N

Let $\mathbb{Z}_N^* = \{0 < x < N \mid \text{gcd}(x, N) = 1\}$. The set $\mathbb{Z}_N^*$ is an Abelian group under multiplication modulo N.

Intuition: We are removing the "problematic" elements (i.e., those without an inverse) from $\{1, \ldots, N\}$,

- **Existence of the identity:** The identity element is $1 \in \mathbb{Z}_N^*$. Indeed $g \cdot 1 = 1 \cdot g = g \pmod N$.
- **Associativity, Commutativity:** Trivial from multiplication over the integers.

The group $\mathbb{Z}_N^*$ under multiplication modulo N

Let $\mathbb{Z}_N^* = \{0 < x < N \mid \mathrm{gcd}(x, N) = 1\}$. The set $\mathbb{Z}_N^*$ is an Abelian group under multiplication modulo N.

Intuition: We are removing the "problematic" elements (i.e., those without an inverse) from $\{1, \ldots, N\}$,

- **Existence of the identity:** The identity element is $1 \in \mathbb{Z}_N^*$. Indeed $g \cdot 1 = 1 \cdot g = g \pmod N$.
- **Associativity, Commutativity:** Trivial from multiplication over the integers.
- **Existence of inverses:** Since $g \in \mathbb{Z}_N^*$ we have $\mathrm{gcd}(g, N) = 1$ and hence there is some $h \in \{1, \ldots, N-1\}$ such that $gh = 1 \pmod N$.

The group $\mathbb{Z}_N^*$ under multiplication modulo N

Let $\mathbb{Z}_N^* = \{0 < x < N \mid \text{gcd}(x, N) = 1\}$. The set $\mathbb{Z}_N^*$ is an Abelian group under multiplication modulo N.

Intuition: We are removing the "problematic" elements (i.e., those without an inverse) from $\{1, \ldots, N\}$,

- **Existence of the identity:** The identity element is $1 \in \mathbb{Z}_N^*$. Indeed $g \cdot 1 = 1 \cdot g = g \pmod{N}$.
- **Associativity, Commutativity:** Trivial from multiplication over the integers.
- **Existence of inverses:** Since $g \in \mathbb{Z}_N^*$ we have $\text{gcd}(g, N) = 1$ and hence there is some $h \in \{1, \ldots, N-1\}$ such that $gh = 1 \pmod{N}$.

  Since h is invertible modulo N (the inverse is g), then $\text{gcd}(h, N) = 1$ and $h \in \mathbb{Z}_N^*$.

The group $\mathbb{Z}_N^*$ under multiplication modulo N

Let $\mathbb{Z}_N^* = \{0 < x < N \mid \gcd(x, N) = 1\}$. The set $\mathbb{Z}_N^*$ is an Abelian group under multiplication modulo N.

Intuition: We are removing the "problematic" elements (i.e., those without an inverse) from $\{1, \ldots, N\}$,

- **Existence of the identity:** The identity element is $1 \in \mathbb{Z}_N^*$. Indeed $g \cdot 1 = 1 \cdot g = g \pmod N$.
- **Associativity, Commutativity:** Trivial from multiplication over the integers.
- **Existence of inverses:** Since $g \in \mathbb{Z}_N^*$ we have $\gcd(g, N) = 1$ and hence there is some $h \in \{1, \ldots, N-1\}$ such that $gh = 1 \pmod N$.

  Since h is invertible modulo N (the inverse is g), then $\gcd(h, N) = 1$ and $h \in \mathbb{Z}_N^*$.
- **Closure:** Pick $a, b \in \mathbb{Z}_N^*$, and let a', b' be their inverses. Notice that $ab \bmod N$ is invertible modulo N (the inverse is $a'b'$)

  Then $\gcd(ab \bmod N, N) = 1$ hence $ab \bmod N \in \mathbb{Z}_N^*$.

The group $\mathbb{Z}_N^*$ under multiplication modulo N

Let $\mathbb{Z}_N^* = \{0 < x < N \mid \text{gcd}(x, N) = 1\}$. The set $\mathbb{Z}_N^*$ is an Abelian group under multiplication modulo N.

Intuition: We are removing the "problematic" elements (i.e., those without an inverse) from $\{1, \ldots, N\}$,

- **Existence of the identity:** The identity element is $1 \in \mathbb{Z}_N^*$. Indeed $g \cdot 1 = 1 \cdot g = g \pmod N$.
- **Associativity, Commutativity:** Trivial from multiplication over the integers.
- **Existence of inverses:** Since $g \in \mathbb{Z}_N^*$ we have $\text{gcd}(g, N) = 1$ and hence there is some $h \in \{1, \ldots, N-1\}$ such that $gh = 1 \pmod N$.

  Since h is invertible modulo N (the inverse is g), then $\text{gcd}(h, N) = 1$ and $h \in \mathbb{Z}_N^*$.
- **Closure:** Pick $a, b \in \mathbb{Z}_N^*$, and let a', b' be their inverses. Notice that $ab \bmod N$ is invertible modulo N (the inverse is $a'b'$)

  Then $\text{gcd}(ab \bmod N, N) = 1$ hence $ab \bmod N \in \mathbb{Z}_N^*$.

Consequence: If p is a prime number then $\{1, 2, \ldots, p-1\}$ is an Abelian group under multiplication modulo p.

Order of $\mathbb{Z}_N^*$

What's the order of $\mathbb{Z}_N^*$?

Order of $\mathbb{Z}_N^*$

What's the order of $\mathbb{Z}_N^*$?

Euler's totient function (or Euler's phi function): $\phi(N)$ is the number of positive integers $a \leq N$ such that a and N are coprime.

$\phi(N) = |\{a \in \{1, \ldots, N-1\} : \text{gcd}(a, N) = 1\}| = |\mathbb{Z}_N^*|$

Order of $\mathbb{Z}_N^*$

What's the order of $\mathbb{Z}_N^*$?

Euler's totient function (or Euler's phi function): $\phi(N)$ is the number of positive integers $a \leq N$ such that a and N are coprime.

$$\phi(N) = |\{a \in \{1, \ldots, N-1\} \, : \, \text{gcd}(a, N) = 1\}| = |\mathbb{Z}_N^*|$$

What's the order of $\mathbb{Z}_p^*$ when p is prime?

Order of $\mathbb{Z}_N^*$

What's the order of $\mathbb{Z}_N^*$?

Euler's totient function (or Euler's phi function): $\phi(N)$ is the number of positive integers $a \leq N$ such that a and N are coprime.

$$\phi(N) = |\{a \in \{1, \ldots, N-1\} \, : \, \text{gcd}(a, N) = 1\}| = |\mathbb{Z}_N^*|$$

What's the order of $\mathbb{Z}_p^*$ when p is prime?

$|\mathbb{Z}_p^*| = \phi(p) = p - 1$ All integers $a = 1, \ldots, p-1$ are such that $\gcd(a, p) = 1$

Order of $\mathbb{Z}_N^*$

What's the order of $\mathbb{Z}_N^*$?

Euler's totient function (or Euler's phi function): $\phi(N)$ is the number of positive integers $a \leq N$ such that a and N are coprime.

$$\phi(N) = |\{a \in \{1, \ldots, N-1\} : \mathsf{gcd}(a, N) = 1\}| = |\mathbb{Z}_N^*|$$

What's the order of $\mathbb{Z}_p^*$ when p is prime?

$|\mathbb{Z}_p^*| = \phi(p) = p - 1$ All integers $a = 1, \ldots, p-1$ are such that $\gcd(a, p) = 1$

What's the order of $\mathbb{Z}_N^*$ when $N = pq$ and p, q are distinct prime numbers?

Order of $\mathbb{Z}_N^*$

What's the order of $\mathbb{Z}_N^*$?

Euler's totient function (or Euler's phi function): $\phi(N)$ is the number of positive integers $a \leq N$ such that a and N are coprime.

$\phi(N) = |\{a \in \{1, \ldots, N-1\} : \text{gcd}(a, N) = 1\}| = |\mathbb{Z}_N^*|$

What's the order of $\mathbb{Z}_p^*$ when p is prime?

$|\mathbb{Z}_p^*| = \phi(p) = p - 1$ All integers $a = 1, \ldots, p-1$ are such that $\gcd(a, p) = 1$

What's the order of $\mathbb{Z}_N^*$ when $N = pq$ and p, q are distinct prime numbers?

$\phi(pq) =$ [$pq - 1$] $-$ [# multiples of p] $-$ [# multiples of q] $+$ [# multiples of both p and q]

Order of $\mathbb{Z}_N^*$

What's the order of $\mathbb{Z}_N^*$?

Euler's totient function (or Euler's phi function): $\phi(N)$ is the number of positive integers $a \leq N$ such that a and N are coprime.

$$\phi(N) = |\{a \in \{1, \ldots, N-1\} : \text{gcd}(a, N) = 1\}| = |\mathbb{Z}_N^*|$$

What's the order of $\mathbb{Z}_p^*$ when p is prime?

$|\mathbb{Z}_p^*| = \phi(p) = p - 1$ All integers $a = 1, \ldots, p-1$ are such that $\gcd(a, p) = 1$

What's the order of $\mathbb{Z}_N^*$ when $N = pq$ and p, q are distinct prime numbers?

$\phi(pq) =$ $pq - 1$ $-$ $|\{p, 2p, \ldots, (q-1)p\}|$ $-$ # multiples of q $+$ # multiples of both p and q

Order of $\mathbb{Z}_N^*$

What's the order of $\mathbb{Z}_N^*$?

Euler's totient function (or Euler's phi function): $\phi(N)$ is the number of positive integers $a \leq N$ such that a and N are coprime.

$$\phi(N) = |\{a \in \{1, \ldots, N-1\} \,:\, \text{gcd}(a, N) = 1\}| = |\mathbb{Z}_N^*|$$

What's the order of $\mathbb{Z}_p^*$ when p is prime?

$|\mathbb{Z}_p^*| = \phi(p) = p - 1$ All integers $a = 1, \ldots, p-1$ are such that $\gcd(a, p) = 1$

What's the order of $\mathbb{Z}_N^*$ when $N = pq$ and p, q are distinct prime numbers?

$\phi(pq) =$ $pq - 1$ $-$ $(q-1)$ $-$ # multiples of q $+$ # multiples of both p and q

Order of $\mathbb{Z}_N^*$

What's the order of $\mathbb{Z}_N^*$?

Euler's totient function (or Euler's phi function): $\phi(N)$ is the number of positive integers $a \leq N$ such that a and N are coprime.

$$\phi(N) = |\{a \in \{1, \ldots, N-1\} : \text{gcd}(a, N) = 1\}| = |\mathbb{Z}_N^*|$$

What's the order of $\mathbb{Z}_p^*$ when p is prime?

$|\mathbb{Z}_p^*| = \phi(p) = p - 1$ All integers $a = 1, \ldots, p-1$ are such that $\gcd(a, p) = 1$

What's the order of $\mathbb{Z}_N^*$ when $N = pq$ and p, q are distinct prime numbers?

$\phi(pq) =$ $pq - 1$ $-$ $(q-1)$ $-$ $(p-1)$ $+$ # multiples of both p and q

Order of $\mathbb{Z}_N^*$

What's the order of $\mathbb{Z}_N^*$?

Euler's totient function (or Euler's phi function): $\phi(N)$ is the number of positive integers $a \leq N$ such that a and N are coprime.

$$\phi(N) = |\{a \in \{1, \ldots, N-1\} : \text{gcd}(a, N) = 1\}| = |\mathbb{Z}_N^*|$$

What's the order of $\mathbb{Z}_p^*$ when p is prime?

$|\mathbb{Z}_p^*| = \phi(p) = p - 1$ All integers $a = 1, \ldots, p-1$ are such that $\gcd(a, p) = 1$

What's the order of $\mathbb{Z}_N^*$ when $N = pq$ and p, q are distinct prime numbers?

$$\phi(pq) = \boxed{pq - 1} - \boxed{(q-1)} - \boxed{(p-1)} + \boxed{0}$$

Order of $\mathbb{Z}_N^*$

What's the order of $\mathbb{Z}_N^*$?

Euler's totient function (or Euler's phi function): $\phi(N)$ is the number of positive integers $a \leq N$ such that a and N are coprime.

$$\phi(N) = |\{a \in \{1, \ldots, N-1\} : \text{gcd}(a, N) = 1\}| = |\mathbb{Z}_N^*|$$

What's the order of $\mathbb{Z}_p^*$ when p is prime?

$|\mathbb{Z}_p^*| = \phi(p) = p - 1$ All integers $a = 1, \ldots, p-1$ are such that $\gcd(a, p) = 1$

What's the order of $\mathbb{Z}_N^*$ when $N = pq$ and p, q are distinct prime numbers?

$$\phi(pq) = \boxed{pq - 1} - \boxed{(q-1)} - \boxed{(p-1)} + \boxed{0}$$

$$= pq - q - p + 1 = p(q-1) - (q-1) = (p-1)(q-1)$$

Order of $\mathbb{Z}_N^*$

What's the order of $\mathbb{Z}_N^*$?

Euler's totient function (or Euler's phi function): $\phi(N)$ is the number of positive integers $a \leq N$ such that a and N are coprime.

$$\phi(N) = |\{a \in \{1, \ldots, N-1\} \, : \, \text{gcd}(a, N) = 1\}| = |\mathbb{Z}_N^*|$$

What's the order of $\mathbb{Z}_p^*$ when p is prime?

$|\mathbb{Z}_p^*| = \phi(p) = p - 1$ All integers $a = 1, \ldots, p-1$ are such that $\gcd(a, p) = 1$

What's the order of $\mathbb{Z}_N^*$ when $N = pq$ and p, q are distinct prime numbers?

$$\phi(pq) = \boxed{pq - 1} - \boxed{(q-1)} - \boxed{(p-1)} + \boxed{0}$$

$$= pq - q - p + 1 \;\; = p(q-1) - (q-1) \; = (p-1)(q-1) \; = \phi(p)\phi(q)$$

Fermat's little theorem

Theorem: Let G be a finite group of order m and let $g \in G$. Then $g^m = 1$.

Fermat's little theorem

Theorem: Let G be a finite group of order m and let $g \in G$. Then $g^m = 1$.

Proof in the Abelian case:

Let $G = \{g_1, g_2, \ldots, g_m\}$.

Since $gg_i = gg_j \implies g^{-1}gg_i = g^{-1}gg_j \implies g_i = g_j$ we have $g_i \neq g_j \implies gg_i \neq gg_j$

Fermat's little theorem

Theorem: Let G be a finite group of order m and let $g \in G$. Then $g^m = 1$.

Proof in the Abelian case:

Let $G = \{g_1, g_2, \ldots, g_m\}$.

Since $gg_i = gg_j \implies g^{-1}gg_i = g^{-1}gg_j \implies g_i = g_j$ we have $g_i \neq g_j \implies gg_i \neq gg_j$

Then:

$$g_1g_2 \ldots g_m = (gg_1)(gg_2) \ldots (gg_m)$$

(Each side of the equation contains only distinct elements, since the order of G in m, all elements are multiplied)

Fermat's little theorem

Theorem: Let G be a finite group of order m and let $g \in G$. Then $g^m = 1$.

Proof in the Abelian case:

Let $G = \{g_1, g_2, \ldots, g_m\}$.

Since $gg_i = gg_j \implies g^{-1}gg_i = g^{-1}gg_j \implies g_i = g_j$ we have $g_i \neq g_j \implies gg_i \neq gg_j$

Then:

$$g_1 g_2 \ldots g_m = (gg_1)(gg_2)\ldots(gg_m) \ = g^m(g_1 g_2 \ldots g_m)$$

(Each side of the equation contains only distinct elements, since the order of G in m, all elements are multiplied)

Fermat's little theorem

Theorem: Let G be a finite group of order m and let $g \in G$. Then $g^m = 1$.

Proof in the Abelian case:

Let $G = \{g_1, g_2, \ldots, g_m\}$.

Since $gg_i = gg_j \implies g^{-1}gg_i = g^{-1}gg_j \implies g_i = g_j$ we have $g_i \neq g_j \implies gg_i \neq gg_j$

Then:

$$g_1 g_2 \ldots g_m = (gg_1)(gg_2)\ldots(gg_m) \; = g^m(g_1 g_2 \ldots g_m)$$

(Each side of the equation contains only distinct elements, since the order of G in m, all elements are multiplied)

Multiplying both sides by $(g_1 g_2 \ldots g_m)^{-1}$

$$(g_1 g_2 \ldots g_m)^{-1}(g_1 g_2 \ldots g_m) = g^m(g_1 g_2 \ldots g_m)(g_1 g_2 \ldots g_m)^{-1}$$

Fermat's little theorem

Theorem: Let G be a finite group of order m and let $g \in G$. Then $g^m = 1$.

Proof in the Abelian case:

Let $G = \{g_1, g_2, \dots, g_m\}$.

Since $gg_i = gg_j \implies g^{-1}gg_i = g^{-1}gg_j \implies g_i = g_j$ we have $g_i \neq g_j \implies gg_i \neq gg_j$

Then:

$$g_1 g_2 \dots g_m = (gg_1)(gg_2)\dots(gg_m) \ = g^m (g_1 g_2 \dots g_m)$$

(Each side of the equation contains only distinct elements, since the order of G in m, all elements are multiplied)

Multiplying both sides by $(g_1 g_2 \dots g_m)^{-1}$

$$1 = \ (g_1 g_2 \dots g_m)^{-1}(g_1 g_2 \dots g_m) = g^m (g_1 g_2 \dots g_m)(g_1 g_2 \dots g_m)^{-1} \ = g^m$$

□

Fermat's little theorem: examples

Theorem: Let G be a finite group of order m and let $g \in G$. Then $g^m = 1$.

In $\mathbb{Z}_N$ (under addition modulo N):

- For all $a \in \mathbb{Z}_N$, we have $N \cdot a = 0$. $\underbrace{a + a + \cdots + a}_{N \text{ times}} = Na = 0 \pmod{N}$.

Fermat's little theorem: examples

Theorem: Let G be a finite group of order m and let $g \in G$. Then $g^m = 1$.

In $\mathbb{Z}_N$ (under addition modulo N):

- For all $a \in \mathbb{Z}_N$, we have $N \cdot a = 0$. $\underbrace{a + a + \cdots + a}_{N \text{ times}} = Na = 0 \pmod{N}$.

In $\mathbb{Z}_N^*$ (under multiplication modulo N):

- For all $a \in \mathbb{Z}_N^*$, we have $a^{\phi(N)} = 1$
- For all $a \in \mathbb{Z}_p^*$ where p is prime, we have $a^{p-1} = 1$

Fermat's little theorem: corollaries

Theorem: Let G be a finite group of order m and let $g \in G$. Then $g^m = 1$.

Corollary: Let G be a finite group of order $m > 1$ and let $g \in G$. For any integer x, $g^x = g^{x \bmod m}$.

Fermat's little theorem: corollaries

Theorem: Let G be a finite group of order m and let $g \in G$. Then $g^m = 1$.

Corollary: Let G be a finite group of order $m > 1$ and let $g \in G$. For any integer x, $g^x = g^{x \bmod m}$.

Proof: Write x as $qm + r$ with $r \in \{0, \ldots, m-1\}$. $g^x = g^{qm+r} = (g^m)^q \cdot g^r = 1^q \cdot g^r = g^r$. $\square$

Fermat's little theorem: corollaries

Theorem: Let G be a finite group of order m and let $g \in G$. Then $g^m = 1$.

Corollary: Let G be a finite group of order $m > 1$ and let $g \in G$. For any integer x, $g^x = g^{x \bmod m}$.

Proof: Write x as $qm + r$ with $r \in \{0, \ldots, m-1\}$. $g^x = g^{qm+r} = (g^m)^q \cdot g^r = 1^q \cdot g^r = g^r$. $\square$

Corollary: Let G be a finite group of order $m > 1$. Let $e > 0$ be an integer, and define the function $f : G \to G$ as $f_e(g) = g^e$. If $\gcd(e, m) = 1$ then

- 1) f_e is a permutation;
- 2) $f_e^{-1}(g) = f_d(g) = g^d$, where d is the inverse of e modulo m.

Fermat's little theorem: corollaries

Theorem: Let G be a finite group of order m and let $g \in G$. Then $g^m = 1$.

Corollary: Let G be a finite group of order $m > 1$ and let $g \in G$. For any integer x, $g^x = g^{x \bmod m}$.

Proof: Write x as $qm + r$ with $r \in \{0, \ldots, m-1\}$. $g^x = g^{qm+r} = (g^m)^q \cdot g^r = 1^q \cdot g^r = g^r$. $\square$

Corollary: Let G be a finite group of order $m > 1$. Let $e > 0$ be an integer, and define the function $f : G \to G$ as $f_e(g) = g^e$. If $\gcd(e, m) = 1$ then

- 1) f_e is a permutation;
- 2) $f_e^{-1}(g) = f_d(g) = g^d$, where d is the inverse of e modulo m.

Proof: We just need to show $2)$ since this implies that f_e injective and surjective, i.e., a bijection.

$$f_d(f_e(g)) = (g^e)^d = g^{ed} = g^{ed \bmod m} = g^{1 \bmod m} = g. \qquad \square$$

Roadmap

Use the tools from number theory and group theory to...

- Find some problem that is easy to solve given some secret information but “hard” to solve otherwise
- Use the “hardness” of this problem to build secure public-key schemes

Roadmap

Use the tools from number theory and group theory to...

- Find some problem that is easy to solve given some secret information but "hard" to solve otherwise
- Use the "hardness" of this problem to build secure public-key schemes

The "hard" problems will be:

- Related to prime numbers and factoring
- Related to cyclic groups

Roadmap

Use the tools from number theory and group theory to...

- Find some problem that is easy to solve given some secret information but "hard" to solve otherwise
- Use the "hardness" of this problem to build secure public-key schemes

The "hard" problems will be:

- Related to prime numbers and factoring
- Related to cyclic groups

Generating Prime numbers

We will be interested in working with prime numbers

The security parameter n will be related to the number of bits of the prime numbers

A n-bit number is an integer between 2^n and $2^{n+1} - 1$ (i.e., its binary representation has n digits and the most significant bit is 1).

Generating Prime numbers

We will be interested in working with prime numbers

The security parameter n will be related to the number of bits of the prime numbers

A n-bit number is an integer between 2^n and $2^{n+1} - 1$ (i.e., its binary representation has n digits and the most significant bit is 1).

How do we efficiently generate a random prime number with n bits?

Generating Prime numbers

We will be interested in working with prime numbers

The security parameter n will be related to the number of bits of the prime numbers

A n-bit number is an integer between 2^n and $2^{n+1} - 1$ (i.e., its binary representation has n digits and the most significant bit is 1).

How do we efficiently generate a random prime number with n bits?

- Suppose that we can check whether a number is prime in polynomial time

Generating Prime numbers

We will be interested in working with prime numbers

The security parameter n will be related to the number of bits of the prime numbers

A n-bit number is an integer between 2^n and $2^{n+1} - 1$ (i.e., its binary representation has n digits and the most significant bit is 1).

How do we efficiently generate a random prime number with n bits?

- Suppose that we can check whether a number is prime in polynomial time

- Repeat up to t times:
  - Choose a number p u.a.r. among all n-bit numbers

    Pick r u.a.r. in $\{0, 1\}^{n-1}$ and let $p \leftarrow 1 \| r$.
  - If p is prime: return p
- Return "failure"

Generating Prime numbers

- Repeat up to t times:
  - Choose a number p u.a.r. among all n-bit numbers
  - If p is prime: return p
- Return "failure"

Running time?

Generating Prime numbers

- Repeat up to t times:
  - Choose a number p u.a.r. among all n-bit numbers
  - If p is prime: return p
- Return "failure"

Running time? $O(t \cdot \text{poly}(n))$

Generating Prime numbers

- Repeat up to t times:
  - Choose a number p u.a.r. among all n-bit numbers
  - If p is prime: return p
- Return "failure"

Running time? $O(t \cdot \text{poly}(n))$ The output size is $\Theta(n)$. We allow time $O(\text{poly}(n))$

What's the probability that an iteration selects a prime number p?

Generating Prime numbers

- Repeat up to t times:
  - Choose a number p u.a.r. among all n-bit numbers
  - If p is prime: return p
- Return "failure"

Running time? $O(t \cdot \text{poly}(n))$ The output size is $\Theta(n)$. We allow time $O(\text{poly}(n))$

What's the probability that an iteration selects a prime number p?

For $n > 1$, the fraction of n-bit numbers that are prime is at least $\frac{1}{3n}$.

Generating Prime numbers

- Repeat up to t times:
    - Choose a number p u.a.r. among all n-bit numbers
    - If p is prime: return p
- Return "failure"

Running time? $O(t \cdot \text{poly}(n))$ The output size is $\Theta(n)$. We allow time $O(\text{poly}(n))$

What's the probability that an iteration selects a prime number p?

For $n > 1$, the fraction of n-bit numbers that are prime is at least $\frac{1}{3n}$.

What's the probability that the algorithm fails?

Generating Prime numbers

- Repeat up to t times:
  - Choose a number p u.a.r. among all n-bit numbers
  - If p is prime: return p
- Return "failure"

Running time? $O(t \cdot \text{poly}(n))$ The output size is $\Theta(n)$. We allow time $O(\text{poly}(n))$

What's the probability that an iteration selects a prime number p?

For $n > 1$, the fraction of n-bit numbers that are prime is at least $\frac{1}{3n}$.

What's the probability that the algorithm fails?

At most $(1 - \frac{1}{3n})^t$

Generating Prime numbers

- Repeat up to t times:
  - Choose a number p u.a.r. among all n-bit numbers
  - If p is prime: return p
- Return "failure"

Running time? $O(t \cdot \text{poly}(n))$ The output size is $\Theta(n)$. We allow time $O(\text{poly}(n))$

What's the probability that an iteration selects a prime number p?

For $n > 1$, the fraction of n-bit numbers that are prime is at least $\frac{1}{3n}$.

What's the probability that the algorithm fails?

At most $(1 - \frac{1}{3n})^t = \left((1 - \frac{1}{3n})^{3n}\right)^{\frac{t}{3n}}$

Generating Prime numbers

- Repeat up to t times:
  - Choose a number p u.a.r. among all n-bit numbers
  - If p is prime: return p
- Return "failure"

Running time? $O(t \cdot \text{poly}(n))$ The output size is $\Theta(n)$. We allow time $O(\text{poly}(n))$

What's the probability that an iteration selects a prime number p?

For $n > 1$, the fraction of n-bit numbers that are prime is at least $\frac{1}{3n}$.

What's the probability that the algorithm fails? How do we pick t?

At most $(1 - \frac{1}{3n})^t = \left((1 - \frac{1}{3n})^{3n}\right)^{\frac{t}{3n}} \leq e^{-\frac{t}{3n}}$

Generating Prime numbers

- Repeat up to t times:
  - Choose a number p u.a.r. among all n-bit numbers
  - If p is prime: return p
- Return "failure"

Running time? $O(t \cdot \text{poly}(n))$ The output size is $\Theta(n)$. We allow time $O(\text{poly}(n))$

What's the probability that an iteration selects a prime number p?

For $n > 1$, the fraction of n-bit numbers that are prime is at least $\frac{1}{3n}$.

What's the probability that the algorithm fails? How do we pick t? E.g., $t = 3n^2$.

At most $(1 - \frac{1}{3n})^t = \left((1 - \frac{1}{3n})^{3n}\right)^{\frac{t}{3n}} \leq e^{-\frac{t}{3n}} = e^{-n}$

Generating Prime numbers

- Repeat up to t times:
  - Choose a number p u.a.r. among all n-bit numbers
  - If p is prime: return p
- Return "failure"

Running time? $O(t \cdot \text{poly}(n))$ The output size is $\Theta(n)$. We allow time $O(\text{poly}(n))$

What's the probability that an iteration selects a prime number p?

For $n > 1$, the fraction of n-bit numbers that are prime is at least $\frac{1}{3n}$.

What's the probability that the algorithm fails? How do we pick t? E.g., $t = 3n^2$.

At most $(1 - \frac{1}{3n})^t = \left((1 - \frac{1}{3n})^{3n}\right)^{\frac{t}{3n}} \leq e^{-\frac{t}{3n}} = e^{-n}$ ← Negligible

The algorithm has a polynomial running time and fails with negligible proability!

Testing Primality

Can we check whether a number N is prime in polynomail time?

I.e., in time $O(\log^k N)$ for some constant k.

Testing Primality

Can we check whether a number N is prime in polynomail time? **Yes!**

I.e., in time $O(\log^k N)$ for some constant k.

Testing Primality

Can we check whether a number N is prime in polynomail time? **Yes!**

I.e., in time $O(\log^k N)$ for some constant k.

- For a long time no polynomial-time deterministic algorithm was known
- Breakthrough in 2002: deterministic algorithm running in time $O(\log^{12} N \cdot \log^k \log N)$ for some constant k.
- Can be improved to $O(\log^6 N \cdot \log^k \log N)$ for some constant k.

Testing Primality

Can we check whether a number N is prime in polynomail time? **Yes!**

I.e., in time $O(\log^k N)$ for some constant k.

- For a long time no polynomial-time deterministic algorithm was known
- Breakthrough in 2002: deterministic algorithm running in time $O(\log^{12} N \cdot \log^k \log N)$ for some constant k.
- Can be improved to $O(\log^6 N \cdot \log^k \log N)$ for some constant k.

In practice randomized algorithm are used, since they are faster and fail with negligible probability.

- The Miller-Rabin primality test is a probabilistic polynomial-time algorithm with one-sided error
- If n is prime, the Miller-Rabin primality test reports n as prime with certainty
- If n is composite, the Miller-Rabin primality test might report n as prime, but only with negligible probability.

Factoring

Given a composite N can we find $p, q > 1$ such that $pq = N$ in polynomial time?

Factoring

Given a composite N can we find $p, q > 1$ such that $pq = N$ in polynomial time?

- Not known to be solvable in polynomial time.
- Not known to be hard.

Factoring

Given a composite N can we find $p, q > 1$ such that $pq = N$ in polynomial time?

- Not known to be solvable in polynomial time.
- Not known to be hard.

Conjectured not to be solvable in polynomial-time.

Factoring

Given a composite N can we find $p, q > 1$ such that $pq = N$ in polynomial time?

- Not known to be solvable in polynomial time.
- Not known to be hard.

Conjectured not to be solvable in polynomial-time.

A fist attempt to formalize the hardness of factoring. Define a factoring experiment w-Factor$_{\mathcal{A}}(n)$ for a given algorithm $\mathcal{A}$:

- Two n-bit integers x_1, x_2 are chosen u.a.r., and $N = x_1 \cdot x_2$ is computed
- N is sent to $\mathcal{A}$
- $\mathcal{A}$ outputs two integers x_1', x_2'
- The outcome of the experiment is 1 if $x_1', x_2' > 1$ and $x_1' \cdot x_2' = N$. Otherwise the outcome is 0.

Factoring

Given a composite N can we find $p, q > 1$ such that $pq = N$ in polynomial time?

- Not known to be solvable in polynomial time.
- Not known to be hard.

Conjectured not to be solvable in polynomial-time.

A fist attempt to formalize the hardness of factoring. Define a factoring experiment w-Factor$_{\mathcal{A}}(n)$ for a given algorithm $\mathcal{A}$:

- Two n-bit integers x_1, x_2 are chosen u.a.r., and $N = x_1 \cdot x_2$ is computed
- N is sent to $\mathcal{A}$
- $\mathcal{A}$ outputs two integers x_1', x_2'
- The outcome of the experiment is 1 if $x_1', x_2' > 1$ and $x_1' \cdot x_2' = N$. Otherwise the outcome is 0.

We could hope that, for all probabilistic polynomial-time algorithms $\mathcal{A}$:

$$\Pr[\text{w-Factor}_{\mathcal{A}}(n) = 1] \leq \varepsilon(n) \text{ for some negligible } \varepsilon(n)$$

Factoring

We could hope that, for all probabilistic polynomial-time algorithms $\mathcal{A}$:

$$\Pr[\text{w-Factor}_{\mathcal{A}}(n) = 1] \leq \varepsilon(n) \text{ for some negligible } \varepsilon(n)$$

Is this true?

Factoring

We could hope that, for all probabilistic polynomial-time algorithms $\mathcal{A}$:

$$\Pr[\text{w-Factor}_{\mathcal{A}}(n) = 1] \leq \varepsilon(n) \text{ for some negligible } \varepsilon(n)$$

Is this true?

There is a trivial algorithm that wins the above experiment with probability $\geq \frac{3}{4}$.

Factoring

We could hope that, for all probabilistic polynomial-time algorithms $\mathcal{A}$:

$$\Pr[\text{w-Factor}_{\mathcal{A}}(n) = 1] \le \varepsilon(n) \text{ for some negligible } \varepsilon(n)$$

Is this true?

There is a trivial algorithm that wins the above experiment with probability $\ge \frac{3}{4}$.

$\mathcal{A}(N)$

- If N is even
  - Return $x'_1 = 2$, $x'_2 = N/2$
- Otherwise
  - Return some arbitrary pair of numbers

Factoring

We could hope that, for all probabilistic polynomial-time algorithms $\mathcal{A}$:

$$\Pr[\text{w-Factor}_{\mathcal{A}}(n) = 1] \leq \varepsilon(n) \text{ for some negligible } \varepsilon(n)$$

Is this true?

There is a trivial algorithm that wins the above experiment with probability $\geq \frac{3}{4}$.

$\mathcal{A}(N)$

- If N is even
  - Return $x'_1 = 2$, $x'_2 = N/2$
- Otherwise
  - Return some arbitrary pair of numbers

With probability $1 - (\frac{1}{2})^2 = \frac{3}{4}$ at least one of x_1 and x_2 is even $\implies$ N is even $\implies$ $\mathcal{A}$ wins the experiment.

Factoring

- It is easy to factor most integers!

- The "hardest" integers N to factor are those that have exactly two prime factors p, q

Factoring

- It is easy to factor most integers!

- The "hardest" integers N to factor are those that have exactly two prime factors p, q

- If N is composite then its smallest (non-trivial) factor is at most $\sqrt{N}$

  Proof: let x be a (non-trivial) factor of N. If $x \leq \sqrt{N}$ we are done.
  Otherwise N/x is a (non-trivial) factor of N and $N/x < N/\sqrt{N} = \sqrt{N}$.

Factoring

- It is easy to factor most integers!

- The "hardest" integers N to factor are those that have exactly two prime factors p, q

- If N is composite then its smallest (non-trivial) factor is at most $\sqrt{N}$

  Proof: let x be a (non-trivial) factor of N. If $x \leq \sqrt{N}$ we are done.
  Otherwise N/x is a (non-trivial) factor of N and $N/x < N/\sqrt{N} = \sqrt{N}$.

- The two prime factors should be roughly $\sqrt{N}$, i.e., the two primes should have (roughly) the same number of bits

Factoring

- It is easy to factor most integers!

- The “hardest” integers N to factor are those that have exactly two prime factors p, q

- If N is composite then its smallest (non-trivial) factor is at most $\sqrt{N}$

  Proof: let x be a (non-trivial) factor of N. If $x \leq \sqrt{N}$ we are done.
  Otherwise N/x is a (non-trivial) factor of N and $N/x < N/\sqrt{N} = \sqrt{N}$.

- The two prime factors should be roughly $\sqrt{N}$, i.e., the two primes should have (roughly) the same number of bits

Let GenModulus be a polynomial-time algorithm that, on input 1^n , outputs a triple (N, p, q) where $N = pq$, and p and q are n-bit primes, except with probability negligible in n.

The Factoring Assumption

We can now revise the previous experiment. For an algorithm $\mathcal{A}$, define $\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(n)$ as:

- Run $\mathsf{GenModulus}(1^n)$ to obtain (N, p, q).
- N is sent to $\mathcal{A}$
- $\mathcal{A}$ outputs two integers p', q'
- The outcome of the experiment is 1 if $p, q > 1$ and $pq = N$. Otherwise the outcome is 0.

The Factoring Assumption

We can now revise the previous experiment. For an algorithm $\mathcal{A}$, define $\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(n)$ as:

- Run $\mathsf{GenModulus}(1^n)$ to obtain (N, p, q).
- N is sent to $\mathcal{A}$
- $\mathcal{A}$ outputs two integers p', q'
- The outcome of the experiment is 1 if $p, q > 1$ and $pq = N$. Otherwise the outcome is 0.

Definition: Factoring is hard relative to GenModulus if for any probabilistic polynomial-time algorithm $\mathcal{A}$ there exists a negligible function ε such that

$$\Pr[\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1] \leq \varepsilon(n).$$

The Factoring Assumption

We can now revise the previous experiment. For an algorithm $\mathcal{A}$, define $\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(n)$ as:

- Run $\mathsf{GenModulus}(1^n)$ to obtain (N, p, q).
- N is sent to $\mathcal{A}$
- $\mathcal{A}$ outputs two integers p', q'
- The outcome of the experiment is 1 if $p, q > 1$ and $pq = N$. Otherwise the outcome is 0.

Definition: Factoring is hard relative to GenModulus if for any probabilistic polynomial-time algorithm $\mathcal{A}$ there exists a negligible function ε such that

$$\Pr[\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1] \leq \varepsilon(n).$$

The factoring assumption: there exists a GenModulus algorithm relative to which the factoring problem is hard.

The Factoring Assumption

We can now revise the previous experiment. For an algorithm $\mathcal{A}$, define $\text{Factor}_{\mathcal{A},\text{GenModulus}}(n)$ as:

- Run $\text{GenModulus}(1^n)$ to obtain (N, p, q).
- N is sent to $\mathcal{A}$
- $\mathcal{A}$ outputs two integers p', q'
- The outcome of the experiment is 1 if $p, q > 1$ and $pq = N$. Otherwise the outcome is 0.

Definition: Factoring is hard relative to GenModulus if for any probabilistic polynomial-time algorithm $\mathcal{A}$ there exists a negligible function ε such that

$$\Pr[\text{Factor}_{\mathcal{A},\text{GenModulus}}(n) = 1] \leq \varepsilon(n).$$

The factoring assumption: there exists a GenModulus algorithm relative to which the factoring problem is hard.

Recall: this is just an assumption. We don't currently know whether the factoring problem is hard.

Are we there yet?

Are we there yet?

Almost...

- The factoring assumption is still too weak
- We need a stronger, but related assumption called the **RSA** assumption

Are we there yet?

Almost...

- The factoring assumption is still too weak
- We need a stronger, but related assumption called the **RSA** assumption

Let $N = pq$ where p and q are distinct odd primes

The order of Z_N^* is $\phi(N) = (p-1) \cdot (q-1)$

- Trivial to compute if we know p and q
- "Hard" to compute if we know N but not p and q (can be shown to be equivalent to factoring N)

Are we there yet?

Almost...

- The factoring assumption is still too weak
- We need a stronger, but related assumption called the **RSA** assumption

Let $N = pq$ where p and q are distinct odd primes

The order of Z_N^* is $\phi(N) = (p-1) \cdot (q-1)$

- Trivial to compute if we know p and q
- "Hard" to compute if we know N but not p and q (can be shown to be equivalent to factoring N)

Pick $e \in \mathbb{Z}_N^*$ such that gcd$(e, \phi(N)) = 1$.

- By the corollary of Fermat's little theorem, $f_e(x) = x^e$ is a permutation of $\mathbb{Z}_N^*$
- Let d be the inverse of e modulo $\phi(N)$. Then $f_d(x) = x^d$ is the inverse of f_e.

$$(x^e)^d = (x^d)^e = x$$

(All the operations are performed modulo N)

e-th roots of x

Since $(x^e)^d = x$ we can think of x^d as the e-th root of x

- We define $x^{1/e} = x^d$

e-th roots of x

Since $(x^e)^d = x$ we can think of x^d as the e-th root of x

- We define $x^{1/e} = x^d$

Given N, e, and x, how do we compute $x^{1/e}$?

e-th roots of x

Since $(x^e)^d = x$ we can think of x^d as the e-th root of x

- We define $x^{1/e} = x^d$

Given N, e, and x, how do we compute $x^{1/e}$?

- If p and q are also known: easy!
  - Compute $\phi(N) = (p-1)(q-1)$
  - Compute the inverse d of e modulo $\phi(N)$
  - Compute x^d via modular exponentiation

e-th roots of x

Since $(x^e)^d = x$ we can think of x^d as the e-th root of x

- We define $x^{1/e} = x^d$

Given N, e, and x, how do we compute $x^{1/e}$?

- If p and q are also known: easy!
  - Compute $\phi(N) = (p-1)(q-1)$
  - Compute the inverse d of e modulo $\phi(N)$
  - Compute x^d via modular exponentiation
- If p and q are not known:
  - Computing $\phi(N)$ is as hard as factoring N
  - We don't know how to compute d without knowing $\phi(N)$
  - ???

The RSA problem

Informally: given a random $y \in \mathbb{Z}_N^*$, computing $y^{1/e}$ is hard

Let GenRSA be a polynomial-time algorithm that, on input 1^n , outputs a triple (N, e, d) where:

- $N = pq$, for two n-bit primes p and q
- $ed = 1 \pmod{\phi(N)}$

The algorithm is allowed to fail with negligible probability.

The RSA problem

Informally: given a random $y \in \mathbb{Z}_N^*$, computing $y^{1/e}$ is hard

Let GenRSA be a polynomial-time algorithm that, on input 1^n , outputs a triple (N, e, d) where:

- $N = pq$, for two n-bit primes p and q
- $ed = 1 \pmod{\phi(N)}$

The algorithm is allowed to fail with negligible probability.

A possible implementation:

- Generate two n-bit primes p, q chosen u.a.r.
- $N \leftarrow p \cdot q$
- $\phi(N) \leftarrow (p-1) \cdot (q-1)$
- Pick some e with $\gcd(e, \phi(N)) = 1$
- $d \leftarrow e^{-1} \pmod{\phi(N)}$
- Output (N, e, d)

The RSA problem

Informally: given a random $y \in \mathbb{Z}_N^*$, computing $y^{1/e}$ is hard

Let GenRSA be a polynomial-time algorithm that, on input 1^n , outputs a triple (N, e, d) where:

- $N = pq$, for two n-bit primes p and q
- $ed = 1 \pmod{\phi(N)}$

The algorithm is allowed to fail with negligible probability.

A possible implementation:

- Generate two n-bit primes p, q chosen u.a.r.
- $N \leftarrow p \cdot q$
- $\phi(N) \leftarrow (p-1) \cdot (q-1)$
- Pick some e with $\gcd(e, \phi(N)) = 1$
- $d \leftarrow e^{-1} \pmod{\phi(N)}$
- Output (N, e, d)

The choice of e is not believed to affect the hardness of the RSA problem

Common choices: $e = 3$ or $e = 2^{16} + 1$ for efficiency reasons

The RSA assumption

For an algorithm $\mathcal{A}$, define the experiment RSA-inv$_{\mathcal{A},\mathsf{GenRSA}}(n)$ as:

- Run GenRSA(1^n) to obtain (N, e, d).
- Choose $y \in \mathbb{Z}_N^*$ u.a.r.
- Send N, e and y to $\mathcal{A}$
- $\mathcal{A}$ outputs $x \in \mathbb{Z}_N^*$
- The outcome of the experiment is 1 if x is the e-th root of y, i.e., if $x^e = y$ (or equivalently $y^{1/e} = y^d = x$). Otherwise the outcome is 0.

The RSA assumption

For an algorithm $\mathcal{A}$, define the experiment $\text{RSA-inv}_{\mathcal{A},\text{GenRSA}}(n)$ as:

- Run $\text{GenRSA}(1^n)$ to obtain (N, e, d).
- Choose $y \in \mathbb{Z}_N^*$ u.a.r.
- Send N, e and y to $\mathcal{A}$
- $\mathcal{A}$ outputs $x \in \mathbb{Z}_N^*$
- The outcome of the experiment is 1 if x is the e-th root of y, i.e., if $x^e = y$ (or equivalently $y^{1/e} = y^d = x$). Otherwise the outcome is 0.

Definition: The RSA problem is hard relative to GenRSA if for any probabilistic polynomial-time algorithm $\mathcal{A}$ there exists a negligible function ε such that

$$\Pr[\text{RSA-inv}_{\mathcal{A},\text{GenRSA}}(n) = 1] \leq \varepsilon(n).$$

The RSA assumption

For an algorithm $\mathcal{A}$, define the experiment $\text{RSA-inv}_{\mathcal{A},\text{GenRSA}}(n)$ as:

- Run $\text{GenRSA}(1^n)$ to obtain (N, e, d).
- Choose $y \in \mathbb{Z}_N^*$ u.a.r.
- Send N, e and y to $\mathcal{A}$
- $\mathcal{A}$ outputs $x \in \mathbb{Z}_N^*$
- The outcome of the experiment is 1 if x is the e-th root of y, i.e., if $x^e = y$ (or equivalently $y^{1/e} = y^d = x$). Otherwise the outcome is 0.

Definition: The RSA problem is hard relative to GenRSA if for any probabilistic polynomial-time algorithm $\mathcal{A}$ there exists a negligible function ε such that

$$\Pr[\text{RSA-inv}_{\mathcal{A},\text{GenRSA}}(n) = 1] \leq \varepsilon(n).$$

The RSA assumption: there exists a GenRSA algorithm relative to which the RSA problem is hard.

The RSA assumption and the factoring assumption

The RSA assumption: there exists a GenRSA algorithm relative to which the RSA problem is hard.

$$\Downarrow$$

The factoring assumption: there exists a GenModulus algorithm relative to which the factoring problem is hard.