Security, Runtimes, and a computational Review of Modular Arithmetic

FA 2018 Cryptography Seminar

J. David Taylor

September 10, 2018

The Problem

Alice wants to send an encrypted message to Bob so that Carol cannot read the message, if she intercepts it and knows what type of encryption is being used.

The Problem

Alice wants to send an encrypted message to Bob so that Carol cannot read the message, if she intercepts it and knows what type of encryption is being used.

Definition

A Cipher is a tuple (K, M, C, enc, dec) where

- K, M, C are *finite* sets, and for every $k \in K$
- $enc_k : M \to C$,
- $dec_k : C \to M$, and
- $dec_k \circ enc_k = \mathrm{id}_M$.

The Problem

Alice wants to send an encrypted message to Bob so that Carol cannot read the message, if she intercepts it and knows what type of encryption is being used.

Definition

A Cipher is a tuple (K, M, C, enc, dec) where

- K, M, C are *finite* sets, and for every $k \in K$
- $enc_k : M \to C$,
- $dec_k : C \to M$, and
- $dec_k \circ enc_k = \mathrm{id}_M$.

Problem: How do we relate the ABC situation with the notion of a cipher?

What is Cryptographic Security?

What is Cryptographic Security?

Some possibilities:

What is Cryptographic Security?

Some possibilities:

- "It should be impossible for the Carol to recover the key."

What is Cryptographic Security?

Some possibilities:

- "It should be impossible for the Carol to recover the key."
- What about a cipher with $M \subseteq C$ and $enc_k = \mathrm{id}_M$ for all $k \in K$?

What is Cryptographic Security?

Some possibilities:

- "It should be impossible for the Carol to recover the key."
- What about a cipher with $M \subseteq C$ and $enc_k = \mathrm{id}_M$ for all $k \in K$?
- Clearly, the ciphertext yields no information on the key, yet Carol can read the message.

What is Cryptographic Security?

Some possibilities:

What is Cryptographic Security?

Some possibilities:

- "It should be impossible for Carol to recover the entire plaintext from the ciphertext"

What is Cryptographic Security?

Some possibilities:

- "It should be impossible for Carol to recover the entire plaintext from the ciphertext"
- What about a cipher that performs a Caesar cipher, but only on the first character?

What is Cryptographic Security?

Some possibilities:

What is Cryptographic Security?

Some possibilities:

- "It should be impossible for Carol to recover any character of the plaintext from the ciphertext"

What is Cryptographic Security?

Some possibilities:

- "It should be impossible for Carol to recover any character of the plaintext from the ciphertext"
- What if Alice is sending Bob accounting information and Carol can discern whether a specific account is above $100,000 without knowing the value?

What is Cryptographic Security?

Definition

A secure encryption scheme is one which has the following property:
"Regardless of any information Carol already has, a ciphertext should leak no additional information about the underlying plaintext."

Perfect Secrecy

Perfect Secrecy

Definition (Shannon)

A cipher is *perfectly secret* if for every probability distribution over M, every plaintext $m \in M$, and every ciphertext $c \in C$ for which $P(X_C = c) > 0$,

$$P(X_M = m | X_C = c) = P(X_m = m).$$

Perfect Secrecy

Definition (Shannon)

A cipher is *perfectly secret* if for every probability distribution over M, every plaintext $m \in M$, and every ciphertext $c \in C$ for which $P(X_C = c) > 0$,

$$P(X_M = m | X_C = c) = P(X_m = m).$$

Are any ciphers perfectly secure?

Perfect Secrecy

Definition (Shannon)

A cipher is *perfectly secret* if for every probability distribution over M, every plaintext $m \in M$, and every ciphertext $c \in C$ for which $P(X_C = c) > 0$,

$$P(X_M = m | X_C = c) = P(X_m = m).$$

Are any ciphers perfectly secure? Yes!

One-Time Pad: Algorithm

- Frank Miller (1882) and Gilbert S. Vernam (1917) independently invented the One-Time Pad.

One-Time Pad: Algorithm

- Frank Miller (1882) and Gilbert S. Vernam (1917) independently invented the One-Time Pad.
- Joseph Mauborgne suggested the use of a random key.

One-Time Pad: Algorithm

- Frank Miller (1882) and Gilbert S. Vernam (1917) independently invented the One-Time Pad.
- Joseph Mauborgne suggested the use of a random key.
- Claude Shannon proved that the One-Time Pad has perfect secrecy.

One-Time Pad: Algorithm

- Frank Miller (1882) and Gilbert S. Vernam (1917) independently invented the One-Time Pad.
- Joseph Mauborgne suggested the use of a random key.
- Claude Shannon proved that the One-Time Pad has perfect secrecy.

- Fix a positive integer $l > 0$.

One-Time Pad: Algorithm

- Frank Miller (1882) and Gilbert S. Vernam (1917) independently invented the One-Time Pad.
- Joseph Mauborgne suggested the use of a random key.
- Claude Shannon proved that the One-Time Pad has perfect secrecy.

- Fix a positive integer $l > 0$.
- $M = K = C = 2^l$, the set of binary sequences of length l.

One-Time Pad: Algorithm

- Frank Miller (1882) and Gilbert S. Vernam (1917) independently invented the One-Time Pad.
- Joseph Mauborgne suggested the use of a random key.
- Claude Shannon proved that the One-Time Pad has perfect secrecy.

- Fix a positive integer $l > 0$.
- $M = K = C = 2^l$, the set of binary sequences of length l.
- Pick $k \in K$ randomly using uniform distribution.

One-Time Pad: Algorithm

- Frank Miller (1882) and Gilbert S. Vernam (1917) independently invented the One-Time Pad.
- Joseph Mauborgne suggested the use of a random key.
- Claude Shannon proved that the One-Time Pad has perfect secrecy.

- Fix a positive integer $l > 0$.
- $M = K = C = 2^l$, the set of binary sequences of length l.
- Pick $k \in K$ randomly using uniform distribution.
- $enc_k(m) = m \oplus k$.

One-Time Pad: Algorithm

- Frank Miller (1882) and Gilbert S. Vernam (1917) independently invented the One-Time Pad.
- Joseph Mauborgne suggested the use of a random key.
- Claude Shannon proved that the One-Time Pad has perfect secrecy.

- Fix a positive integer $l > 0$.
- $M = K = C = 2^l$, the set of binary sequences of length l.
- Pick $k \in K$ randomly using uniform distribution.
- $enc_k(m) = m \oplus k$.
- $dec_k(c) = c \oplus k$.

One-Time Pad: Algorithm

- Frank Miller (1882) and Gilbert S. Vernam (1917) independently invented the One-Time Pad.
- Joseph Mauborgne suggested the use of a random key.
- Claude Shannon proved that the One-Time Pad has perfect secrecy.

- Fix a positive integer $l > 0$.
- $M = K = C = 2^l$, the set of binary sequences of length l.
- Pick $k \in K$ randomly using uniform distribution.
- $enc_k(m) = m \oplus k$.
- $dec_k(c) = c \oplus k$.

One-Time Pad: Example (a very short message)

- $l = 8$ (Alice and Bob will trade a letter in ASCII)

One-Time Pad: Example (a very short message)

- $l = 8$ (Alice and Bob will trade a letter in ASCII)
- $k = 11010011$ (random)
- Alice encodes the letter 'E' using ASCII in binary as 'm=01000101'

One-Time Pad: Example (a very short message)

- $l = 8$ (Alice and Bob will trade a letter in ASCII)
- $k = 11010011$ (random)
- Alice encodes the letter 'E' using ASCII in binary as 'm=01000101'
- $enc_k(m) = 10010110 = c$, which is a non-printable Unicode character. This is what Carol receives.

One-Time Pad: Example (a very short message)

- $l = 8$ (Alice and Bob will trade a letter in ASCII)
- $k = 11010011$ (random)
- Alice encodes the letter 'E' using ASCII in binary as 'm=01000101'
- $enc_k(m) = 10010110 = c$, which is a non-printable Unicode character. This is what Carol receives.
- $dec_k(10010110) = 01000101$, which is the 'E' that Bob receives.

One-Time Pad: Example (a very short message)

- $l = 8$ (Alice and Bob will trade a letter in ASCII)
- $k = 11010011$ (random)
- Alice encodes the letter 'E' using ASCII in binary as 'm=01000101'
- $enc_k(m) = 10010110 = c$, which is a non-printable Unicode character. This is what Carol receives.
- $dec_k(10010110) = 01000101$, which is the 'E' that Bob receives.
- Aside: a common means of encoding text as numbers is ASCII, or its Unicode extensions.

One-Time Pad: Limitations

- Requires truly random key

One-Time Pad: Limitations

- Requires truly random key
- Secure generation and exchange of the key

One-Time Pad: Limitations

- Requires truly random key
- Secure generation and exchange of the key
- Preserved security of the key

One-Time Pad: Limitations

- Requires truly random key
- Secure generation and exchange of the key
- Preserved security of the key
- Can only be used once

One-Time Pad: Limitations

- Requires truly random key
- Secure generation and exchange of the key
- Preserved security of the key
- Can only be used once
- It's a theorem that perfect secrecy $\Rightarrow |K| \geq |M|$

One-Time Pad: Limitations

- Requires truly random key
- Secure generation and exchange of the key
- Preserved security of the key
- Can only be used once
- It's a theorem that perfect secrecy $\Rightarrow |K| \geq |M|$
- These features hold in general for perfect secrecy.

One-Time Pad: Benefits

- Theoretically optimal

One-Time Pad: Benefits

- Theoretically optimal
- One of the most practical forms of encryption when at least one party must do everything by hand

One-Time Pad: Benefits

- Theoretically optimal
- One of the most practical forms of encryption when at least one party must do everything by hand
- When two parties meet securely and then depart

One-Time Pad: Historical Uses

- KGB spies used one-time pads during the Cold War

One-Time Pad: Historical Uses

- KGB spies used one-time pads during the Cold War
- British Special Operations Executive used one-time pads during WWII to send messages between offices

One-Time Pad: Historical Uses

- KGB spies used one-time pads during the Cold War
- British Special Operations Executive used one-time pads during WWII to send messages between offices
- The hotline between D.C. and Moscow established after the Cuban missile crisis used a one-time pad system with the keys stored on tapes.

One-Time Pad: Historical Uses

- KGB spies used one-time pads during the Cold War
- British Special Operations Executive used one-time pads during WWII to send messages between offices
- The hotline between D.C. and Moscow established after the Cuban missile crisis used a one-time pad system with the keys stored on tapes.
- U.S. Army Special Forces used one-time pads via Morse code in Vietnam.

Big-O Notation

Definition

Let $f, g : \mathbb{Z}_+ \to \mathbb{R}$. We say that "$f = O(g)$" if for all $N > 0$ there is $C > 0$ such that

$$n \geq N \Rightarrow f(n) \leq C \cdot g(n).$$

Big-O Notation

Definition

Let $f, g : \mathbb{Z}_+ \to \mathbb{R}$. We say that "$f = O(g)$" if for all $N > 0$ there is $C > 0$ such that

$$n \geq N \Rightarrow f(n) \leq C \cdot g(n).$$

Examples:

- $n^5 - 900n^4 + 3 = O(n^5)$

Big-O Notation

Definition

Let $f, g : \mathbb{Z}_+ \to \mathbb{R}$. We say that "$f = O(g)$" if for all $N > 0$ there is $C > 0$ such that

$$n \geq N \Rightarrow f(n) \leq C \cdot g(n).$$

Examples:

- $n^5 - 900n^4 + 3 = O(n^5)$
- the number of digits of $n = O(\log n)$ (in any base)

Big-O Notation

Definition

Let $f, g : \mathbb{Z}_+ \to \mathbb{R}$. We say that "$f = O(g)$" if for all $N > 0$ there is $C > 0$ such that

$$n \geq N \Rightarrow f(n) \leq C \cdot g(n).$$

Examples:

- $n^5 - 900n^4 + 3 = O(n^5)$
- the number of digits of $n = O(\log n)$ (in any base)
- any bounded function is $= O(1)$.

Runtimes

- The runtime of an algorithm is the number of elementary operations required to run the algorithm, where elementary operations are those that take a fixed amount of time.

Runtimes

- The runtime of an algorithm is the number of elementary operations required to run the algorithm, where elementary operations are those that take a fixed amount of time.
- The one-time pad encryption and decryption algorithms each have a runtime $O(l)$, where l is the length of the sequences.

Runtimes

- The runtime of an algorithm is the number of elementary operations required to run the algorithm, where elementary operations are those that take a fixed amount of time.
- The one-time pad encryption and decryption algorithms each have a runtime $O(l)$, where l is the length of the sequences.
- Adding two l-digit numbers has runtime $O(l)$.

Runtimes

- The runtime of an algorithm is the number of elementary operations required to run the algorithm, where elementary operations are those that take a fixed amount of time.
- The one-time pad encryption and decryption algorithms each have a runtime $O(l)$, where l is the length of the sequences.
- Adding two l-digit numbers has runtime $O(l)$.
- Multiplying a k-digit number and an l-digit number with the schoolbook algorithm has runtime of $O(l \cdot k)$.

Runtimes

- The runtime of an algorithm is the number of elementary operations required to run the algorithm, where elementary operations are those that take a fixed amount of time.
- The one-time pad encryption and decryption algorithms each have a runtime $O(l)$, where l is the length of the sequences.
- Adding two l-digit numbers has runtime $O(l)$.
- Multiplying a k-digit number and an l-digit number with the schoolbook algorithm has runtime of $O(l \cdot k)$.
- Half a kilobyte $= 8^{500} \approx 10^{451}$.

Runtimes

- The runtime of an algorithm is the number of elementary operations required to run the algorithm, where elementary operations are those that take a fixed amount of time.
- The one-time pad encryption and decryption algorithms each have a runtime $O(l)$, where l is the length of the sequences.
- Adding two l-digit numbers has runtime $O(l)$.
- Multiplying a k-digit number and an l-digit number with the schoolbook algorithm has runtime of $O(l \cdot k)$.
- Half a kilobyte $= 8^{500} \approx 10^{451}$.
  We will assume that we are doing arithmetic with large integers (e.g. on the order of 10^{100} to 10^{500}).

Runtimes

- The runtime of an algorithm is the number of elementary operations required to run the algorithm, where elementary operations are those that take a fixed amount of time.
- The one-time pad encryption and decryption algorithms each have a runtime $O(l)$, where l is the length of the sequences.
- Adding two l-digit numbers has runtime $O(l)$.
- Multiplying a k-digit number and an l-digit number with the schoolbook algorithm has runtime of $O(l \cdot k)$.
- Half a kilobyte $= 8^{500} \approx 10^{451}$.
  We will assume that we are doing arithmetic with large integers (e.g. on the order of 10^{100} to 10^{500}). This will affect runtime calculations.

Computational Secrecy: Definitions

- We say that an algorithm is *efficient* if its runtime is $O(n^k)$ for some $k \geq 0$. (where n is the size of input to the algorithm)

Computational Secrecy: Definitions

- We say that an algorithm is *efficient* if its runtime is $O(n^k)$ for some $k \geq 0$. (where n is the size of input to the algorithm)
  This is also called a polynomial time algorithm.

Computational Secrecy: Definitions

- We say that an algorithm is *efficient* if its runtime is $O(n^k)$ for some $k \geq 0$. (where n is the size of input to the algorithm)
  This is also called a polynomial time algorithm.
- A function is negligible if it is $O(n^{-k})$ for some $k > 0$.

Computational Secrecy: Definitions

- We say that an algorithm is *efficient* if its runtime is $O(n^k)$ for some $k \geq 0$. (where n is the size of input to the algorithm)
  This is also called a polynomial time algorithm.
- A function is negligible if it is $O(n^{-k})$ for some $k > 0$.

Definition

A cipher is *computationally secure* if any probabilistic polynomial time (PPT) algorithm succeeds in breaking the cipher with negligible probability.

Computational Secrecy: Definitions

- We say that an algorithm is *efficient* if its runtime is $O(n^k)$ for some $k \geq 0$. (where n is the size of input to the algorithm)
  This is also called a polynomial time algorithm.
- A function is negligible if it is $O(n^{-k})$ for some $k > 0$.

Definition

A cipher is *computationally secure* if any probabilistic polynomial time (PPT) algorithm succeeds in breaking the cipher with negligible probability.

Note: This is an asymptotic condition.

Computational vs Perfect Secrecy

- Computational Secrecy is only a guarantee against PPT algorithms.

Computational vs Perfect Secrecy

- Computational Secrecy is only a guarantee against PPT algorithms. Carol could carry out a brute-force attack trying all keys. This has a runtime of $O(|K|)$.

Computational vs Perfect Secrecy

- Computational Secrecy is only a guarantee against PPT algorithms. Carol could carry out a brute-force attack trying all keys. This has a runtime of $O(|K|)$.
- Computational Secrecy allows a small probability of success by Carol.

Computational vs Perfect Secrecy

- Computational Secrecy is only a guarantee against PPT algorithms. Carol could carry out a brute-force attack trying all keys. This has a runtime of $O(|K|)$.
- Computational Secrecy allows a small probability of success by Carol. If Carol had a collection of pairs $(m_1, c_1), \ldots, (m_i, c_i)$, she could guess $k \in K$ (uniformly) and check if it works. This has a runtime of $O(1)$ and probability $1/|K|$ of success.

Modular Arithmetic

Division Algorithm

- For every pair of integers a, b there are *unique* integers q, r such that

$$a = qb + r \text{ and } 0 \leq r < |b|.$$

Division Algorithm

- For every pair of integers a, b there are *unique* integers q, r such that

$$a = qb + r \text{ and } 0 \leq r < |b|.$$

- Using Newton's Method, q, r can be computed with runtime $O(\log(a) \times \log(b))$.

Division Algorithm

- For every pair of integers a, b there are *unique* integers q, r such that

$$a = qb + r \text{ and } 0 \leq r < |b|.$$

- Using Newton's Method, q, r can be computed with runtime $O(\log(a) \times \log(b))$.
- At most 4 iterations are required for double precision (53 bit integer) floating point numbers.

Division Algorithm

- For every pair of integers a, b there are *unique* integers q, r such that

$$a = qb + r \text{ and } 0 \leq r < |b|.$$

- Using Newton's Method, q, r can be computed with runtime $O(\log(a) \times \log(b))$.
- At most 4 iterations are required for double precision (53 bit integer) floating point numbers.
- See Wikipedia: "Division algorithm"

Extended Euclidean Algorithm

- Given integers a, b, $\gcd(a, b)$ can be computed as follows:

Extended Euclidean Algorithm

- Given integers a, b, $\gcd(a, b)$ can be computed as follows:
- $r_0 := a, r_1 := b,$

Extended Euclidean Algorithm

- Given integers a, b, $\gcd(a, b)$ can be computed as follows:
- $r_0 := a, r_1 := b$, $s_0 := 1, s_1 := 0, t_0 := 0, t_1 = 1$,

Extended Euclidean Algorithm

- Given integers a, b, $\gcd(a, b)$ can be computed as follows:
- $r_0 := a, r_1 := b$, $s_0 := 1, s_1 := 0, t_0 := 0, t_1 = 1$,
- Given sequences r, s, t up to index i, compute

$$r_{i-1} = q_i r_i + r_{i+1}$$
$$s_{i-1} = q_i s_i + s_{i+1}$$
$$t_{i-1} = q_i t_i + t_{i+1}$$

using the division algorithm in the first line, and muliplication/subtraction on the following two lines (since q_i is known).

Extended Euclidean Algorithm

- Stop when $r_{k+1} = 0$.

Extended Euclidean Algorithm

- Stop when $r_{k+1} = 0$.
- $\gcd(a, b) = r_k$.

Extended Euclidean Algorithm

- Stop when $r_{k+1} = 0$.
- $\gcd(a, b) = r_k$.
- (Bezout's Identity)

$$\gcd(a, b) = a \cdot s_k + b \cdot t_k.$$

Extended Euclidean Algorithm

- Stop when $r_{k+1} = 0$.
- $\gcd(a, b) = r_k$.
- (Bezout's Identity)

$$\gcd(a, b) = a \cdot s_k + b \cdot t_k.$$

- The total runtime is at most $O(\min(\log |a|, \log |b|)^2)$.

Extended Euclidean Algorithm: Example

- Compute $\gcd(273, 186)$

Extended Euclidean Algorithm: Example

- Compute $\gcd(273, 186)$
- $273 = 1 \cdot 186 + 87$, so $r_2 = 87$ and $q_1 = 1$

Extended Euclidean Algorithm: Example

- Compute $\gcd(273, 186)$
- $273 = 1 \cdot 186 + 87$, so $r_2 = 87$ and $q_1 = 1$
  $s_2 = s_0 - 1 \cdot s_1 = 1$
  $t_2 = t_0 - 1 \cdot t_1 = -1$

Extended Euclidean Algorithm: Example

- Compute $\gcd(273, 186)$
- $273 = 1 \cdot 186 + 87$, so $r_2 = 87$ and $q_1 = 1$
  $s_2 = s_0 - 1 \cdot s_1 = 1$
  $t_2 = t_0 - 1 \cdot t_1 = -1$
- $186 = 2 \cdot 87 + 12$, so $r_3 = 12$ and $q_2 = 2$

Extended Euclidean Algorithm: Example

- Compute $\gcd(273, 186)$
- $273 = 1 \cdot 186 + 87$, so $r_2 = 87$ and $q_1 = 1$
  $s_2 = s_0 - 1 \cdot s_1 = 1$
  $t_2 = t_0 - 1 \cdot t_1 = -1$
- $186 = 2 \cdot 87 + 12$, so $r_3 = 12$ and $q_2 = 2$
  $s_3 = s_1 - 2s_2 = -2$
  $t_3 = t_1 - 2t_2 = 3$
- $87 = 7 \cdot 12 + 3$, so $r_4 = 3$ and $q_3 = 7$

Extended Euclidean Algorithm: Example

- Compute $\gcd(273, 186)$
- $273 = 1 \cdot 186 + 87$, so $r_2 = 87$ and $q_1 = 1$
  $s_2 = s_0 - 1 \cdot s_1 = 1$
  $t_2 = t_0 - 1 \cdot t_1 = -1$
- $186 = 2 \cdot 87 + 12$, so $r_3 = 12$ and $q_2 = 2$
  $s_3 = s_1 - 2s_2 = -2$
  $t_3 = t_1 - 2t_2 = 3$
- $87 = 7 \cdot 12 + 3$, so $r_4 = 3$ and $q_3 = 7$
  $s_4 = s_2 - 7s_3 = 15$
  $\mathrm{j}t_4 = t_2 - 7t_3 = -22$
- $12 = 4 \cdot 3 + 0$, so $r_5 = 0$,

Extended Euclidean Algorithm: Example

- Compute $\gcd(273, 186)$
- $273 = 1 \cdot 186 + 87$, so $r_2 = 87$ and $q_1 = 1$
  $s_2 = s_0 - 1 \cdot s_1 = 1$
  $t_2 = t_0 - 1 \cdot t_1 = -1$
- $186 = 2 \cdot 87 + 12$, so $r_3 = 12$ and $q_2 = 2$
  $s_3 = s_1 - 2s_2 = -2$
  $t_3 = t_1 - 2t_2 = 3$
- $87 = 7 \cdot 12 + 3$, so $r_4 = 3$ and $q_3 = 7$
  $s_4 = s_2 - 7s_3 = 15$
  $\mathrm{j}t_4 = t_2 - 7t_3 = -22$
- $12 = 4 \cdot 3 + 0$, so $r_5 = 0$, and the gcd is

$$3 = 273 \cdot (15) + 186 \cdot (-22)$$

Modular Arithmetic: Runtimes

- Let $a, a', m \in \mathbb{Z}$ with $m > 0$.

Modular Arithmetic: Runtimes

- Let $a, a', m \in \mathbb{Z}$ with $m > 0$. We say that $a \equiv a' \mod m$, if $m|(a - a')$.

Modular Arithmetic: Runtimes

- Let $a, a', m \in \mathbb{Z}$ with $m > 0$. We say that $a \equiv a' \mod m$, if $m|(a - a')$.
- For each integer m, $\equiv$ is an equivalence relation on $\mathbb{Z}$.

Modular Arithmetic: Runtimes

- Let $a, a', m \in \mathbb{Z}$ with $m > 0$. We say that $a \equiv a' \mod m$, if $m|(a - a')$.
- For each integer m, $\equiv$ is an equivalence relation on $\mathbb{Z}$.
- We let $\mathbb{Z}/m\mathbb{Z}$ be the set of equivalence classes.

Modular Arithmetic: Runtimes

- Let $a, a', m \in \mathbb{Z}$ with $m > 0$. We say that $a \equiv a' \mod m$, if $m|(a - a')$.
- For each integer m, $\equiv$ is an equivalence relation on $\mathbb{Z}$.
- We let $\mathbb{Z}/m\mathbb{Z}$ be the set of equivalence classes. It is a ring under addition and multiplication of representatives.

Modular Arithmetic: Runtimes

- Let $a, a', m \in \mathbb{Z}$ with $m > 0$. We say that $a \equiv a' \mod m$, if $m|(a - a')$.
- For each integer m, $\equiv$ is an equivalence relation on $\mathbb{Z}$.
- We let $\mathbb{Z}/m\mathbb{Z}$ be the set of equivalence classes. It is a ring under addition and multiplication of representatives.
- The operations $+, -$ can be calculated in $O(\log(m))$ operations.

Modular Arithmetic: Runtimes

- Let $a, a', m \in \mathbb{Z}$ with $m > 0$. We say that $a \equiv a' \mod m$, if $m|(a - a')$.
- For each integer m, $\equiv$ is an equivalence relation on $\mathbb{Z}$.
- We let $\mathbb{Z}/m\mathbb{Z}$ be the set of equivalence classes. It is a ring under addition and multiplication of representatives.
- The operations $+, -$ can be calculated in $O(\log(m))$ operations.
- The operation $\times$ can be calculated in $O(\log(m)^2)$ operations.

Modular Arithmetic: Runtimes

- An integer u is a unit in $\mathbb{Z}/m\mathbb{Z}$ iff $\gcd(u, m) = 1$.

Modular Arithmetic: Runtimes

- An integer u is a unit in $\mathbb{Z}/m\mathbb{Z}$ iff $\gcd(u, m) = 1$.
- This can be checked by the Euclidean algorithm in $O(\log(m)^2)$ operations.

Modular Arithmetic: Runtimes

- An integer u is a unit in $\mathbb{Z}/m\mathbb{Z}$ iff $\gcd(u, m) = 1$.
- This can be checked by the Euclidean algorithm in $O(\log(m)^2)$ operations.
- Furthermore, if $\gcd(u, m) = u \cdot s_k + m \cdot t_k$, then $s_k = u^{-1}$ in $\mathbb{Z}/m\mathbb{Z}$.

Modular Arithmetic: Runtimes

- An integer u is a unit in $\mathbb{Z}/m\mathbb{Z}$ iff $\gcd(u, m) = 1$.
- This can be checked by the Euclidean algorithm in $O(\log(m)^2)$ operations.
- Furthermore, if $\gcd(u, m) = u \cdot s_k + m \cdot t_k$, then $s_k = u^{-1}$ in $\mathbb{Z}/m\mathbb{Z}$. So reciprocals can also be computed in $O(\log(m)^2)$ time.

Modular Arithmetic: Runtimes

- An integer u is a unit in $\mathbb{Z}/m\mathbb{Z}$ iff $\gcd(u, m) = 1$.
- This can be checked by the Euclidean algorithm in $O(\log(m)^2)$ operations.
- Furthermore, if $\gcd(u, m) = u \cdot s_k + m \cdot t_k$, then $s_k = u^{-1}$ in $\mathbb{Z}/m\mathbb{Z}$. So reciprocals can also be computed in $O(\log(m)^2)$ time.
- Let $a, n \in \mathbb{Z}$ and $n > 0$. Then $a^n \mod m$ can be computed in $O(n \log(m)^2)$ operations via repeated squaring or "Montgomery reduction".

Modular Arithmetic: Examples

- $1 = 273 \cdot (20) + 53 \cdot (-103)$

Modular Arithmetic: Examples

- $1 = 273 \cdot (20) + 53 \cdot (-103)$
- Therefore, $-103 \equiv 170 \mod 273$ is the reciprocal of 53 mod 273.

Modular Arithmetic: Examples

- $1 = 273 \cdot (20) + 53 \cdot (-103)$
- Therefore, $-103 \equiv 170 \mod 273$ is the reciprocal of 53 mod 273.
- Let's compute $53^{10} \mod 273$.

Modular Arithmetic: Examples

- $1 = 273 \cdot (20) + 53 \cdot (-103)$
- Therefore, $-103 \equiv 170 \mod 273$ is the reciprocal of 53 mod 273.
- Let's compute $53^{10} \mod 273$.
- $53^2 \equiv 79 \mod 273$

Modular Arithmetic: Examples

- $1 = 273 \cdot (20) + 53 \cdot (-103)$
- Therefore, $-103 \equiv 170 \mod 273$ is the reciprocal of 53 mod 273.
- Let's compute $53^{10} \mod 273$.
- $53^2 \equiv 79 \mod 273$
- $53^4 \equiv 79^2 \equiv 235 \mod 273$

Modular Arithmetic: Examples

- $1 = 273 \cdot (20) + 53 \cdot (-103)$
- Therefore, $-103 \equiv 170 \mod 273$ is the reciprocal of 53 mod 273.
- Let's compute $53^{10} \mod 273$.
- $53^2 \equiv 79 \mod 273$
- $53^4 \equiv 79^2 \equiv 235 \mod 273$
- $53^8 \equiv 235^2 \equiv 79 \mod 273$

Modular Arithmetic: Examples

- $1 = 273 \cdot (20) + 53 \cdot (-103)$
- Therefore, $-103 \equiv 170 \mod 273$ is the reciprocal of 53 mod 273.
- Let's compute $53^{10} \mod 273$.
- $53^2 \equiv 79 \mod 273$
- $53^4 \equiv 79^2 \equiv 235 \mod 273$
- $53^8 \equiv 235^2 \equiv 79 \mod 273$
- $53^{10} = 53^2 \cdot 53^8 \equiv 79 \cdot 79 \equiv 235 \mod 273$.

Modular Arithmetic: Totient

- For every positive integer n, Euler's Totient function $\phi(n)$ is defined as the number of positive integers less than n and coprime to it.

Modular Arithmetic: Totient

- For every positive integer n, Euler's Totient function $\phi(n)$ is defined as the number of positive integers less than n and coprime to it.
- $\phi(m) =$ the number of units in $\mathbb{Z}/m\mathbb{Z}$.

Modular Arithmetic: Totient

- For every positive integer n, Euler's Totient function $\phi(n)$ is defined as the number of positive integers less than n and coprime to it.
- $\phi(m) =$ the number of units in $\mathbb{Z}/m\mathbb{Z}$.
- $$\phi(m) = m \cdot \prod_{p|m} \left(1 - \frac{1}{p}\right)$$

Modular Arithmetic: Totient

- For every positive integer n, Euler's Totient function $\phi(n)$ is defined as the number of positive integers less than n and coprime to it.
- $\phi(m) =$ the number of units in $\mathbb{Z}/m\mathbb{Z}$.
- $$\phi(m) = m \cdot \prod_{p|m} \left(1 - \frac{1}{p}\right)$$
- (Euler's Theorem) If $\gcd(a, m) = 1$, then $$a^{\phi(m)} \equiv 1 \mod m$$

Modular Arithmetic: Chinese Remainder Theorem

- Suppose that $m = \prod_{i=1}^{s} p_i^{e_i}$ is the prime factorization of m into distinct primes.

Modular Arithmetic: Chinese Remainder Theorem

- Suppose that $m = \prod_{i=1}^{s} p_i^{e_i}$ is the prime factorization of m into distinct primes.
- The map $\mathbb{Z}/m\mathbb{Z} \to \prod_{i=1}^{s} \mathbb{Z}/p_i^{e_i}\mathbb{Z}$ given by

$$m \mapsto (m \mod p_i^{e^i})_i$$

is an isomorphism of rings.

Modular Arithmetic: Chinese Remainder Theorem

- Suppose that $m = \prod_{i=1}^{s} p_i^{e_i}$ is the prime factorization of m into distinct primes.
- The map $\mathbb{Z}/m\mathbb{Z} \to \prod_{i=1}^{s} \mathbb{Z}/p_i^{e_i}\mathbb{Z}$ given by

$$m \mapsto (m \mod p_i^{e^i})_i$$

  is an isomorphism of rings.
- The inverse map is

$$(a_i)_i \mapsto \sum_{i=1}^{s} a_i \left(\frac{m}{p_i^{e_i}}\right) M_i,$$

  where M_i is the reciprocal of $\frac{m}{p_i^{e_i}} \mod p_i^{e_i}$.

CRT Example

- Note that $792 = 2^3 \cdot 3^2 \cdot 11$

CRT Example

- Note that $792 = 2^3 \cdot 3^2 \cdot 11$
- Let's solve $5x \equiv 7 \mod 792$

CRT Example

- Note that $792 = 2^3 \cdot 3^2 \cdot 11$
- Let's solve $5x \equiv 7 \mod 792$
- This is equivalent to solving the system

$$\begin{aligned} 5x_2 &\equiv 7 \mod 8 \\ 5x_3 &\equiv 7 \mod 9 \\ 5x_{11} &\equiv 7 \mod 11 \end{aligned}$$

CRT Example

- $5^{-1} \equiv 5 \mod 8$

CRT Example

- $5^{-1} \equiv 5 \mod 8$
  So $x_2 = 7 \cdot 5 \equiv 3 \mod 8$.

CRT Example

- $5^{-1} \equiv 5 \mod 8$
  So $x_2 = 7 \cdot 5 \equiv 3 \mod 8$.
- $5^{-1} \equiv 2 \mod 9$

CRT Example

- $5^{-1} \equiv 5 \mod 8$
  So $x_2 = 7 \cdot 5 \equiv 3 \mod 8$.
- $5^{-1} \equiv 2 \mod 9$
  So $x_3 = 7 \cdot 2 \equiv 5 \mod 9$.

CRT Example

- $5^{-1} \equiv 5 \mod 8$
  So $x_2 = 7 \cdot 5 \equiv 3 \mod 8$.
- $5^{-1} \equiv 2 \mod 9$
  So $x_3 = 7 \cdot 2 \equiv 5 \mod 9$.
- $5^{-1} \equiv 9 \mod 11$

CRT Example

- $5^{-1} \equiv 5 \mod 8$
  So $x_2 = 7 \cdot 5 \equiv 3 \mod 8$.
- $5^{-1} \equiv 2 \mod 9$
  So $x_3 = 7 \cdot 2 \equiv 5 \mod 9$.
- $5^{-1} \equiv 9 \mod 11$
  So $x_{11} = 7 \cdot 9 \equiv 8 \mod 11$.

CRT Example

- $5^{-1} \equiv 5 \mod 8$
  So $x_2 = 7 \cdot 5 \equiv 3 \mod 8$.
- $5^{-1} \equiv 2 \mod 9$
  So $x_3 = 7 \cdot 2 \equiv 5 \mod 9$.
- $5^{-1} \equiv 9 \mod 11$
  So $x_{11} = 7 \cdot 9 \equiv 8 \mod 11$.
- Now we need to figure out which element of $\mathbb{Z}/792\mathbb{Z}$ corresponds to $(3 \mod 8, 5 \mod 9, 8 \mod 11)$.

CRT Example

- Compute $M_2 = 3$, $M_3 = 4$, and $M_{11} = 2$.

CRT Example

- Compute $M_2 = 3$, $M_3 = 4$, and $M_{11} = 2$.
- So (mod 792)

$$\begin{aligned} x &\equiv 3 \cdot \left(\frac{792}{8}\right) \cdot M_2 + 5 \cdot \left(\frac{792}{9}\right) \cdot M_3 + 8 \cdot \left(\frac{792}{11}\right) \cdot M_{11} \\ &\equiv 3 \cdot 99 \cdot 3 + 5 \cdot 88 \cdot 4 + 8 \cdot 72 \cdot 2 \\ &\equiv 335 \mod 792 \end{aligned}$$

Modular Arithmetic: Primitive Roots

- Let p be a prime number and let $e \in \mathbb{Z}$ be positive.

Modular Arithmetic: Primitive Roots

- Let p be a prime number and let $e \in \mathbb{Z}$ be positive.
- It is a theorem that $(\mathbb{Z}/p^e\mathbb{Z})^\times$ is a cyclic group.

Modular Arithmetic: Primitive Roots

- Let p be a prime number and let $e \in \mathbb{Z}$ be positive.
- It is a theorem that $(\mathbb{Z}/p^e\mathbb{Z})^\times$ is a cyclic group.
- A generator of $(\mathbb{Z}/p^e\mathbb{Z})^\times$ is called a primitive root mod p^e.

Modular Arithmetic: Primitive Roots

- Let p be a prime number and let $e \in \mathbb{Z}$ be positive.
- It is a theorem that $(\mathbb{Z}/p^e\mathbb{Z})^\times$ is a cyclic group.
- A generator of $(\mathbb{Z}/p^e\mathbb{Z})^\times$ is called a primitive root mod p^e.
- Primitive roots are not easy to find in general.

Modular Arithmetic: Primitive Roots

- Let p be a prime number and let $e \in \mathbb{Z}$ be positive.
- It is a theorem that $(\mathbb{Z}/p^e\mathbb{Z})^\times$ is a cyclic group.
- A generator of $(\mathbb{Z}/p^e\mathbb{Z})^\times$ is called a primitive root mod p^e.
- Primitive roots are not easy to find in general.
- Corollary: Let g be a primitive root mod p^e. Then

$$\mathbb{Z}/\phi(p^e)\mathbb{Z} \to (\mathbb{Z}/p^e\mathbb{Z})^\times$$

given by

$$n \mapsto g^n$$

is an isomorphism of groups.

Primitive Roots Example

- $13^2 = 169$

Primitive Roots Example

- $13^2 = 169$
- $\mathbb{Z}/169\mathbb{Z}$ is a cyclic group of order $\phi(13^2) = 13^2(1 - 1/13) = 156$.

Primitive Roots Example

- $13^2 = 169$
- $\mathbb{Z}/169\mathbb{Z}$ is a cyclic group of order $\phi(13^2) = 13^2(1 - 1/13) = 156$.
- $156 = 4 \cdot 3 \cdot 13$

Primitive Roots Example

- $13^2 = 169$
- $\mathbb{Z}/169\mathbb{Z}$ is a cyclic group of order $\phi(13^2) = 13^2(1 - 1/13) = 156$.
- $156 = 4 \cdot 3 \cdot 13$
- $2^{4 \cdot 3} \equiv 128 \mod 169$

Primitive Roots Example

- $13^2 = 169$
- $\mathbb{Z}/169\mathbb{Z}$ is a cyclic group of order $\phi(13^2) = 13^2(1 - 1/13) = 156$.
- $156 = 4 \cdot 3 \cdot 13$
- $2^{4 \cdot 3} \equiv 128 \mod 169$
  $2^{4 \cdot 13} \equiv 146 \mod 169$

Primitive Roots Example

- $13^2 = 169$
- $\mathbb{Z}/169\mathbb{Z}$ is a cyclic group of order $\phi(13^2) = 13^2(1 - 1/13) = 156.$
- $156 = 4 \cdot 3 \cdot 13$
- $2^{4 \cdot 3} \equiv 128 \mod 169$
  $2^{4 \cdot 13} \equiv 146 \mod 169$
  $2^{3 \cdot 13} \equiv 99 \mod 169$

Primitive Roots Example

- $13^2 = 169$
- $\mathbb{Z}/169\mathbb{Z}$ is a cyclic group of order $\phi(13^2) = 13^2(1 - 1/13) = 156.$
- $156 = 4 \cdot 3 \cdot 13$
- $2^{4 \cdot 3} \equiv 128 \mod 169$
  $2^{4 \cdot 13} \equiv 146 \mod 169$
  $2^{3 \cdot 13} \equiv 99 \mod 169$
  $2^{156} \equiv 1 \mod 169$

Primitive Roots Example

- $13^2 = 169$
- $\mathbb{Z}/169\mathbb{Z}$ is a cyclic group of order $\phi(13^2) = 13^2(1 - 1/13) = 156$.
- $156 = 4 \cdot 3 \cdot 13$
- $2^{4 \cdot 3} \equiv 128 \mod 169$
  $2^{4 \cdot 13} \equiv 146 \mod 169$
  $2^{3 \cdot 13} \equiv 99 \mod 169$
  $2^{156} \equiv 1 \mod 169$
- So 2 is a primitive root mod 169 and

$$\mathbb{Z}/156\mathbb{Z} \to (\mathbb{Z}/169\mathbb{Z})^\times$$

given by $n \mapsto 2^n \mod 169$, is an isomorphism of groups.

That's all folks!

Thank You