Lecture 10
Number Theoretic Algorithms 1
El-Gamal Encryption

COMP26120

Giles Reger

April 2019

Adverts

Summer Placement to work on COMPjudge
To apply send me an email with

1. a CV
2. a description of how you think COMPjudge could be improved, ideally with some thoughts on how that could be done

Third Year Projects - see project book but also happy to receive suggestions related to COMP24412, COMP26120 (or COMP11212)

Lucas and Milan have some exciting projects up as well.

Today

Number theory forms the basis of modern cryptography

Today we will look at some standard number-theoretic algorithms and see how they fit together to build a very simple cryptographic system

Your final lab involves implementing this system

Next time (week 11, after Easter) we look at the problem of finding prime numbers (and see an example of a randomized algorithm)

Reminder

Integer Division

An integer a divides an integer b, $a \mid b$, if b can be written as $a.c$ for some other integer c. Furthermore, for any integer a and any positive integer n we can write $a = qn + r$ for unique integers q and r such that $0 \leq r < n$.

Modular Arithmetic

Given $a = qn + r$ we can write $r = a \bmod n$. It follows that $0 \leq (a \bmod n) < n$. If $(a \bmod n) = 0$ then $n \mid a$.

Prime Numbers

An integer a is prime if its only divisors are 1 and a. These are very important in number theory (and cryptography) and play nicely with modular arithmetic (as we see later).

Highest Common Factor (or greatest common divisor)

The highest common factor (hcf) of two non-negative integers a and b is the largest number that divides both a and b. We write this $hcf(a, b)$.

Important property: $hcf(a, b) \mid (ax + by)$ for any x, y. Why?

Highest Common Factor (or greatest common divisor)

The highest common factor (hcf) of two non-negative integers a and b is the largest number that divides both a and b. We write this $hcf(a, b)$.

Important property: $hcf(a, b) \mid (ax + by)$ for any x, y. Why?

Second important property (Bézout's identity): There exist integers x, y such that $ax + by = hcf(a, b)$. *(proof not one line)*

Corollary of that: if $d \mid a$ and $d \mid b$ then $d \mid hcf(a, b)$. Why?

Highest Common Factor (or greatest common divisor)

The highest common factor (hcf) of two non-negative integers a and b is the largest number that divides both a and b. We write this $hcf(a, b)$.

Important property: $hcf(a, b) \mid (ax + by)$ for any x, y. Why?

Second important property (Bézout's identity): There exist integers x, y such that $ax + by = hcf(a, b)$. *(proof not one line)*

Corollary of that: if $d \mid a$ and $d \mid b$ then $d \mid hcf(a, b)$. Why?

If we assume $a \geq b$ then we can write $a = qb + r$ for positive q and $r = a \bmod b$. We can rearrange to get $(a \bmod b) = r = a - qb$.

Highest Common Factor (or greatest common divisor)

The highest common factor (hcf) of two non-negative integers a and b is the largest number that divides both a and b. We write this $hcf(a, b)$.

Important property: $hcf(a, b) \mid (ax + by)$ for any x, y. Why?

Second important property (Bézout's identity): There exist integers x, y such that $ax + by = hcf(a, b)$. *(proof not one line)*

Corollary of that: if $d \mid a$ and $d \mid b$ then $d \mid hcf(a, b)$. Why?

If we assume $a \geq b$ then we can write $a = qb + r$ for positive q and $r = a \bmod b$. We can rearrange to get $(a \bmod b) = r = a - qb$.

It follows that $hcf(a, b) \mid a \bmod b$.

Highest Common Factor (or greatest common divisor)

The highest common factor (hcf) of two non-negative integers a and b is the largest number that divides both a and b. We write this $hcf(a, b)$.

Important property: $hcf(a, b) \mid (ax + by)$ for any x, y. Why?

Second important property (Bézout's identity): There exist integers x, y such that $ax + by = hcf(a, b)$. *(proof not one line)*

Corollary of that: if $d \mid a$ and $d \mid b$ then $d \mid hcf(a, b)$. Why?

If we assume $a \geq b$ then we can write $a = qb + r$ for positive q and $r = a \bmod b$. We can rearrange to get $(a \bmod b) = r = a - qb$.

It follows that $hcf(a, b) \mid a \bmod b$.Therefore, $hcf(a, b) \mid hcf(b, a \bmod b)$.

Highest Common Factor (or greatest common divisor)

The highest common factor (hcf) of two non-negative integers a and b is the largest number that divides both a and b. We write this $hcf(a, b)$.

Important property: $hcf(a, b) \mid (ax + by)$ for any x, y. Why?

Second important property (Bézout's identity): There exist integers x, y such that $ax + by = hcf(a, b)$. *(proof not one line)*

Corollary of that: if $d \mid a$ and $d \mid b$ then $d \mid hcf(a, b)$. Why?

If we assume $a \geq b$ then we can write $a = qb + r$ for positive q and $r = a \bmod b$. We can rearrange to get $(a \bmod b) = r = a - qb$.

It follows that $hcf(a, b) \mid a \bmod b$.Therefore, $hcf(a, b) \mid hcf(b, a \bmod b)$.

Similarly, $hcf(b, a \bmod b) \mid hcf(a, b)$

Highest Common Factor (or greatest common divisor)

The highest common factor (hcf) of two non-negative integers a and b is the largest number that divides both a and b. We write this $hcf(a, b)$.

Important property: $hcf(a, b) \mid (ax + by)$ for any x, y. Why?

Second important property (Bézout's identity): There exist integers x, y such that $ax + by = hcf(a, b)$. *(proof not one line)*

Corollary of that: if $d \mid a$ and $d \mid b$ then $d \mid hcf(a, b)$. Why?

If we assume $a \geq b$ then we can write $a = qb + r$ for positive q and $r = a \bmod b$. We can rearrange to get $(a \bmod b) = r = a - qb$.

It follows that $hcf(a, b) \mid a \bmod b$.Therefore, $hcf(a, b) \mid hcf(b, a \bmod b)$.

Similarly, $hcf(b, a \bmod b) \mid hcf(a, b)$, thus $hcf(a, b) = hcf(b, a \bmod b)$.

Euclid's Algorithm

Based on $hcf(a, b) = hcf(b, a \bmod b)$

Given a, b such that $a \neq 0$ and $a \geq b$

```
hcf(a,b)
  if b = 0
    return a
  else
    r := a mod b
    return hcf(b,r)
```

$$hcf(30, 21) =$$

Euclid's Algorithm

Based on $hcf(a, b) = hcf(b, a \bmod b)$

Given a, b such that $a \neq 0$ and $a \geq b$

```
hcf(a,b)
  if b = 0
    return a
  else
    r := a mod b
    return hcf(b,r)
```

$$hcf(30, 21) = hcf(21, 9) =$$

Euclid's Algorithm

Based on $hcf(a, b) = hcf(b, a \bmod b)$

Given a, b such that $a \neq 0$ and $a \geq b$

```
hcf(a,b)
  if b = 0
    return a
  else
    r := a mod b
    return hcf(b,r)
```

$$hcf(30, 21) = hcf(21, 9) = hcf(9, 3) =$$

Euclid's Algorithm

Based on $hcf(a, b) = hcf(b, a \bmod b)$

Given a, b such that $a \neq 0$ and $a \geq b$

```
hcf(a,b)
  if b = 0
    return a
  else
    r := a mod b
    return hcf(b,r)
```

$$hcf(30, 21) = hcf(21, 9) = hcf(9, 3) = hcf(3, 0) =$$

Euclid's Algorithm

Based on $hcf(a, b) = hcf(b, a \bmod b)$

Given a, b such that $a \neq 0$ and $a \geq b$

```
hcf(a,b)
  if b = 0
    return a
  else
    r := a mod b
    return hcf(b,r)
```

$$hcf(30, 21) = hcf(21, 9) = hcf(9, 3) = hcf(3, 0) = 3$$

Euclid's Algorithm: Complexity

Let a_i be the value of the first argument on the ith recursive call to `hcf`

Clearly, $a = a_1$ and $a_i > a_{i+1}$ so it terminates

Note that $a_{i+2} = a_i \texttt{ mod } a_{i+1}$

Let a_n be the last step. Given $n > 2$ then for $1 \leq h \leq n - 2$

1. If $a_{h+1} \leq a_h/2$ then $a_{h+2} < a_h/2$
2. If $a_{h+1} > a_h/2$ then $a_{h+2} = a_h \texttt{ mod } a_{h+1} = a_h - a_{h+1} < a_h/2$

In either case, $a_{h+2} < a_h/2$, therefore $n = \texttt{max}(2, 2\lceil log_2 a \rceil)$.

So, what is the complexity?

Euclid's Algorithm: Complexity

Let a_i be the value of the first argument on the ith recursive call to `hcf`

Clearly, $a = a_1$ and $a_i > a_{i+1}$ so it terminates

Note that $a_{i+2} = a_i \texttt{ mod } a_{i+1}$

Let a_n be the last step. Given $n > 2$ then for $1 \leq h \leq n-2$

1. If $a_{h+1} \leq a_h/2$ then $a_{h+2} < a_h/2$
2. If $a_{h+1} > a_h/2$ then $a_{h+2} = a_h \texttt{ mod } a_{h+1} = a_h - a_{h+1} < a_h/2$

In either case, $a_{h+2} < a_h/2$, therefore $n = \texttt{max}(2, 2\lceil log_2 a \rceil)$.

So, what is the complexity? The size of a is the number of bits required, which is $n = log_2\ a$. Therefore, this is linear in the size of a.

Relatively Prime

We say that a and b are relatively prime or coprime if their highest common factor is 1 e.g. if $hcf(a, b) = 1$.

If p is prime then every number $k < p$ is relatively prime to p.

Modular Exponentiation

Something we may want to compute is $a^b \bmod k$ (assuming $a < k$)

A trivial algorithm for this would be

```
pow1(a, b, k)
  s := 1
  for i from 1 to b
    s := s · a mod k
  return s
```

What is the complexity?

Modular Exponentiation

Something we may want to compute is $a^b \bmod k$ (assuming $a < k$)

A trivial algorithm for this would be

```
pow1(a, b, k)
  s := 1
  for i from 1 to b
    s := s · a mod k
  return s
```

What is the complexity? Cleary $O(b)$ but....

Modular Exponentiation

Something we may want to compute is $a^b \mathtt{~mod~} k$ (assuming $a < k$)

A trivial algorithm for this would be

```
pow1(a, b, k)
  s := 1
  for i from 1 to b
    s := s · a mod k
  return s
```

What is the complexity? Cleary $O(b)$ but.... this is $O(2^n)$ where b is n bits, so exponential in the size of b.

Modular Exponentiation (Better)

Let's think of b in terms of its binary representation $b = b_{n-1}, \ldots, b_0$ i.e.

$$b = \sum_{i=0}^{n-1} b_i \cdot 2^i$$

This means that

$$a^b = \prod_{i=0}^{n-1} a^{(b_i \cdot 2^i)}$$

And we can observe that

$$a^{(b_i \cdot 2^i)} = \begin{cases} 1 & if \;\; b_i = 0 \\ a^{(2^i)} & if \;\; b_i = 1 \end{cases}$$

So, we can iterate over n and if the ith bit is set we multiply by

$$a^{(2^i)} = a^{2 \cdot 2^{i-1}} = (a^{2^{i-1}})^2$$

Modular Exponentiation (Better)

We increase c from 0 to b and maintain that $d = a^c \texttt{ mod } k$

```
pow2(a,b,k)
  c = 0; d = 1
  for i = n downto 0
    c = 2c
    d = d^2 mod k
    if b_i == 1
      c = c+1
      d = (d · a) mod k
  return d
```

Clearly there are n iterations and this is linear in the size of b.

(Course textbook gives alternative presentation)

Modular Exponentiation (Better)

We increase c from 0 to b and maintain that $d = a^c \bmod k$

```
pow2(a,b,k)
  c = 0; d = 1
  for i = n downto 0
    c = 2c
    d = d^2 mod k
    if b_i == 1
      c = c+1
      d = (d · a) mod k
  return d
```

`pow2(7,11,10)`
e.g. $7^{11} \bmod 10$
$(7^{11} = 1977326743)$

i	c	d	b_i
3	1	7	1
2	2	9	0
1	5	7	1
0	11	3	1

Clearly there are n iterations and this is linear in the size of b.

(Course textbook gives alternative presentation)

Fermat's Little Theorem

Raising positive numbers to powers modulo k produces 1 quite often.

For example, consider the following modulo 7 (which is prime)

$$
\begin{array}{llllll}
1^1 = 1 & & & & & \\
2^1 = 2 & 2^2 = 4 & 2^3 = 1 & & & \\
3^1 = 3 & 3^2 = 2 & 3^3 = 6 & 3^4 = 4 & 3^5 = 5 & 3^6 = 1 \\
4^1 = 4 & 4^2 = 2 & 4^3 = 1 & & & \\
5^1 = 5 & 5^2 = 4 & 5^3 = 6 & 5^4 = 2 & 5^5 = 3 & 5^6 = 1
\end{array}
$$

Fermat's Little Theorem

Raising positive numbers to powers modulo k produces 1 quite often.

For example, consider the following modulo 7 (which is prime)

$$
\begin{array}{llllll}
1^1 = 1 \\
2^1 = 2 & 2^2 = 4 & 2^3 = 1 \\
3^1 = 3 & 3^2 = 2 & 3^3 = 6 & 3^4 = 4 & 3^5 = 5 & 3^6 = 1 \\
4^1 = 4 & 4^2 = 2 & 4^3 = 1 \\
5^1 = 5 & 5^2 = 4 & 5^3 = 6 & 5^4 = 2 & 5^5 = 3 & 5^6 = 1
\end{array}
$$

Fermat's little theorem says that if p is prime then

$$a^{p-1} \equiv 1 \pmod{p}$$

To get the proof for this go and look up Euler's totient function. Briefly, we have a special function $\phi(n)$ that gives the number of $k < n$ that are relatively prime to n. For prime p we clearly have $\phi(p) = p - 1$. Euler's theorem shows that $a^{\phi(n)} \equiv 1 \pmod{n}$ for all a relatively prime to n.

Primitive Roots

Let $A_n = \{k = a \bmod n \mid a \in \mathbb{Z} \textit{ and } hcf(k, n) = 1\}$ be the set of integers from 0 to n that are relatively prime to n.

If p is prime then $A_p = \{1, 2, \ldots, p-1\}$

If $\{g^i \bmod n \mid 1 \leq i < n\} = A_n$ then g is a primitive root modulo n or generator for A_n

Primitive Roots

Let $A_n = \{k = a \bmod n \mid a \in \mathbb{Z} \ and \ hcf(k, n) = 1\}$ be the set of integers from 0 to n that are relatively prime to n.

If p is prime then $A_p = \{1, 2, \ldots, p - 1\}$

If $\{g^i \bmod n \mid 1 \leq i < n\} = A_n$ then g is a primitive root modulo n or generator for A_n

There are primitive roots modulo n if and only if $n = 1, 2, 3, 4, p^k$, or $2p^k$ where p is an odd prime.

Therefore, if p is prime then A_p has primitive roots

In books you will see A_p written as $\mathbb{Z}_n^*$

Modular Arithmetic and Primes

In other words...

If the modulus is a prime number p then for $1 \leq a < p$ there is a b such that

$$a \cdot b \equiv b \cdot a \equiv 1 \pmod{p}$$

i.e. b is the inverse of a (modulo p), or $b = a^{-1}$

E.g. $3 \cdot 5 \equiv 5 \cdot 3 \equiv 1 \pmod{7}$

Discrete Logarithm

Let g be a primitive root modulo some prime p

For every y (in A_p) we can find an x (in A_p) such that

$$g^x = y \texttt{ mod } p$$

We call x the discrete logarithm of y with base a, modulo p

This is an inverse of exponentiation.

Discrete Logarithm

Let g be a primitive root modulo some prime p

For every y (in A_p) we can find an x (in A_p) such that

$$g^x = y \texttt{ mod } p$$

We call x the discrete logarithm of y with base a, modulo p

This is an inverse of exponentiation.

We can compute exponentiation quickly but there is no fast algorithm for computing discrete logarithms (e.g. recovering x from y)

Therefore, modular exponentiation can be considered a one-way function – easy to compute, hard to invert.

One-Way Functions and Cryptography

A one-way function (or trapdoor function) is one that is easy to compute but hard to invert.

One-Way Functions and Cryptography

A one-way function (or trapdoor function) is one that is easy to compute but hard to invert.

Formally, a function f is one-way if it can be computed in polynomial time but any polynomial-time randomized algorithm F that attempts to compute a pseudo-inverse for f succeeds with negligible probability e.g.

$$Pr[f(F(f(x))) = f(x)] < n^{-c}$$

for positive c and sufficiently large $n = |x|$.

One-Way Functions and Cryptography

A one-way function (or trapdoor function) is one that is easy to compute but hard to invert.

Formally, a function f is one-way if it can be computed in polynomial time but any polynomial-time randomized algorithm F that attempts to compute a pseudo-inverse for f succeeds with negligible probability e.g.

$$Pr[f(F(f(x))) = f(x)] < n^{-c}$$

for positive c and sufficiently large $n = |x|$.

This makes it 'hard' in the average case, not worst-case as in NP-hardness e.g. NP-hardness is not a sufficient condition for a function to be one-way.

If one-way functions exist then $P \neq NP$. Do one-way functions exist?

One-Way Functions and Cryptography

A one-way function (or trapdoor function) is one that is easy to compute but hard to invert.

Formally, a function f is one-way if it can be computed in polynomial time but any polynomial-time randomized algorithm F that attempts to compute a pseudo-inverse for f succeeds with negligible probability e.g.

$$Pr[f(F(f(x))) = f(x)] < n^{-c}$$

for positive c and sufficiently large $n = |x|$.

This makes it 'hard' in the average case, not worst-case as in NP-hardness e.g. NP-hardness is not a sufficient condition for a function to be one-way.

If one-way functions exist then $P \neq NP$. Do one-way functions exist?

In practice, we look for functions with a very computationally difficult inverse. Finding good functions to use with cryptography is difficult.

El Gamal Encryption

Fix a prime p and find g, a primitive root modulo p.
Choose $x \in A_p$ as a private key

Broadcast the public key (p, g, y) where $y = g^x \bmod p$

Assume Alice wants to send a message M (such that $1 \leq M < p$). She picks k relatively prime to $p - 1$ and sets

$$a \leftarrow g^k \bmod p \qquad b \leftarrow My^k \bmod p$$

and sends ciphertext $C = (a, b)$

To decode $C = (a, b)$ we can do

$$\begin{aligned} M' = b/(a^x) \bmod p \quad &= My^k(a^x)^{-1} \bmod p \\ &= M(g^{xk})(g^{xk})^{-1} \bmod p \\ &= M \end{aligned}$$

This requires us to have x.

Demonstrating El Gamal Encryption

Giles wants to get a date from Lucas. He picks some things

$$p = 4093082899 \quad g = 2 \quad x = 23$$

He produces

$$y = 2^{23} \bmod 4093082899 = 8388608$$

Lucas takes the date as a number and picks $k = 1607207821$ to get

$$a = 3826697734 \qquad b = 1137681086$$

and sends this to Giles.

Giles uses $x = 23$ to get

$$(1137681086/(3826697734^{23}) \bmod 4093082899) = 30052019$$

and now knows the date 30/05/2019

Summary

El Gamal Encryption based on modular exponentiation (probably) being a one-way function

How to find a k relatively prime to $p-1$? Randomly generate and use *hcf* to check relatively primeness.

How to find a good p for the public key? next time: primality testing

Hint for lab: if you follow the algorithm for fast modular exponentiation from the Goodrich book then it admits a nice optimisation using Fermat's Little Theorem.

Also, if you manage to implement discrete logarithm in polynomial time then you may have broken modern cryptography... double check

Finally, if you're interested in this stuff check out the RSA crytosystem