Number Theory: Applications

Slides by Christopher M. Bourke Instructor: Berthe Y. Choueiry

Spring 2006

Computer Science & Engineering 235 Introduction to Discrete Mathematics Sections 2.4–2.6 of Rosen cse235@cse.unl.edu

Number Theory: Applications

Results from Number Theory have *countless* applications in mathematics as well as in practical applications including security, memory management, authentication, coding theory, etc. We will only examine (in breadth) a few here.

- ► Hash Functions
- Pseudorandom Numbers
- ► Fast Arithmetic Operations
- Cryptography

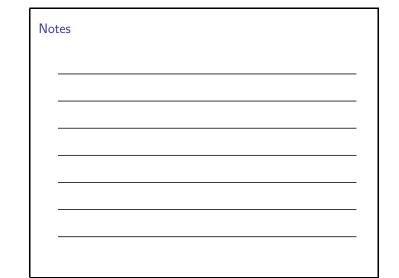
Hash Functions I

Some notation: $\mathbb{Z}_m = \{0, 1, 2, \dots, m-2, m-1\}$

Define a hash function $h: \mathbb{Z} \to \mathbb{Z}_m$ as

 $h(k) = k \bmod m$

That is, h maps all integers into a subset of size m by computing the remainder of k/m.



Notes

Hash Functions II

In general, a hash function should have the following properties

- ▶ It must be easily computable.
- It should distribute items as evenly as possible among all values addresses. To this end, m is usually chosen to be a prime number. It is also common practice to define a hash function that is dependent on each bit of a key
- It must be an onto function (surjective).

Hashing is so useful that many languages have support for hashing (perl, Lisp, Python).

Hash Functions III

However, the function is clearly not one-to-one. When two elements, $x_1\neq x_2$ hash to the same value, we call it a collision.

There are many methods to resolve collisions, here are just a few.

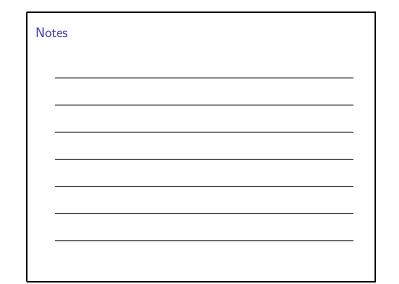
- Open Hashing (aka separate chaining) each hash address is the head of a linked list. When collisions occur, the new key is appended to the end of the list.
- Closed Hashing (aka open addressing) when collisions occur, we attempt to hash the item into an adjacent hash address. This is known as *linear probing*.

Pseudorandom Numbers

Many applications, such as randomized algorithms, require that we have access to a random source of information (random numbers).

However, there is not *truly random* source in existence, only *weak random sources*: sources that *appear* random, but for which we do not know the probability distribution of events.

Pseudorandom numbers are numbers that are generated from weak random sources such that their distribution is "random enough".



Notes

Pseudorandom Numbers I

Linear Congruence Method

One method for generating pseudorandom numbers is the *linear* congruential method.

Choose four integers:

- $\blacktriangleright~m$, the modulus,
- ▶ a, the multiplier,
- $\blacktriangleright\ c$ the increment and
- $\blacktriangleright x_0$ the seed.

Such that the following hold:

- $\blacktriangleright \ 2 \leq a < m$
- $\blacktriangleright \ 0 \leq c < m$
- $\blacktriangleright \ 0 \le x_o < m$

Pseudorandom Numbers II

Linear Congruence Method

Our goal will be to generate a sequence of pseudorandom numbers,

 ${x_n}_{n=1}^{\infty}$

with $0 \le x_n \le m$ by using the congruence

 $x_{n+1} = (ax_n + c) \bmod m$

For certain choices of m, a, c, x_0 , the sequence $\{x_n\}$ becomes *periodic*. That is, after a certain point, the sequence begins to repeat. Low periods lead to poor generators.

Furthermore, some choices are better than others; a generator that creates a sequence $0,5,0,5,0,5,\ldots$ is obvious bad—its not uniformly distributed.

For these reasons, very large numbers are used in practice.

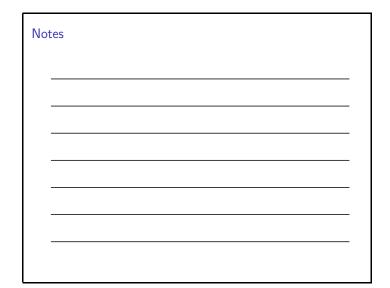
Linear Congruence Method

Example

Example

Let $m = 17, a = 5, c = 2, x_0 = 3$. Then the sequence is as follows.

- x_{n+1} = (ax_n + c) mod m
 x₁ = (5 ⋅ x₀ + 2) mod 17 = 0
- $x_1 = (5 \cdot x_0 + 2) \mod 17 = 5$ • $x_2 = (5 \cdot x_1 + 2) \mod 17 = 2$
- $x_3 = (5 \cdot x_2 + 2) \mod 17 = 12$
- $x_3 = (5 \cdot x_2 + 2) \mod 17 = 12$ • $x_4 = (5 \cdot x_3 + 2) \mod 17 = 11$
- $x_5 = (5 \cdot x_4 + 2) \mod 17 = 6$
- $x_6 = (5 \cdot x_5 + 2) \mod 17 = 15$
- $x_7 = (5 \cdot x_6 + 2) \mod 17 = 9$
- $x_8 = (5 \cdot x_7 + 2) \mod 17 = 13$ etc.



Notes

Representation of Integers I

This should be old-hat to you, but we review it to be complete (it is also discussed in great detail in your textbook).

Any integer $n\ {\rm can}\ {\rm be}\ {\rm uniquely}\ {\rm expressed}\ {\rm in}\ {\rm any}\ {\rm base}\ b\ {\rm by}\ {\rm the}\ {\rm following}\ {\rm expression}.$

 $n = a_k b^k + a_{k-1} b^{k-1} + \dots + a_2 b^2 + a_1 b + a_0$

In the expression, each coefficient $a_i \mbox{ is an integer between } 0 \mbox{ and } b-1 \mbox{ inclusive.}$

Representation of Integers II

For b=2, we have the usual binary representation.

b = 8, gives us the octal representation.

 $b=16\ {\rm gives}$ us the hexadecimal representation.

b=10 gives us our usual decimal system.

We use the notation

 $(a_k a_{k-1} \cdots a_2 a_1 a_0)_b$

For $b=10,\,{\rm we}$ omit the parentheses and subscript. We also omit leading 0s.

Representation of IntegersExampleExample(B9)_{16} = 11 \cdot 16^{1} + 9 \cdot 16^{0} = 176 + 9 = 185(271)_8 = 2 \cdot 8^{2} + 7 \cdot 8^{1} + 1 \cdot 8^{0} = 128 + 56 + 1 = 185(1011 1001)_2 = 1 \cdot 2^{7} + 0 \cdot 2^{6} + 1 \cdot 2^{5} + 1 \cdot 2^{4} + 1 \cdot 2^{3} + 0 \cdot 2^{2} + 0 \cdot 2^{1} + 1 \cdot 2^{0} = 185

You can verify the following on your own:

 $134 = (1000 \ 0110)_2 = (206)_8 = (86)_{16}$

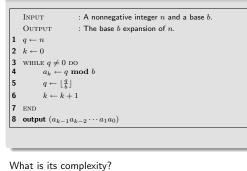
 $44613 = (1010\ 1110\ 0100\ 0101)_2 = (127105)_8 = (AE45)_{16}$

| Notes | | | |
|-------|------|------|-------|
| | | | |
| | | | - |
| | | | |
| | | | |
| | | | - |
| | | | - |

Notes



There is a simple and obvious algorithm to compute the base b expansion of an integer. BASE b EXPANSION



Integer Operations I

You should already know how to add and multiply numbers in binary expansions.

If not, we can go through some examples.

In the textbook, you have 3 algorithms for computing:

- 1. Addition of two integers in binary expansion; runs in O(n).
- 2. Product of two integers in binary expansion; runs in ${\cal O}(n^2)$ (an algorithm that runs in ${\cal O}(n^{1.585})$ exists).
- 3. div and \mod for

$$\begin{array}{rcl} q &=& a \ \mathbf{div} \ d \\ r &=& a \ \mathbf{mod} \ d \end{array}$$

The algorithm runs in $O(q\log a)$ but an algorithm that runs in $O(\log q\log a)$ exists.

Modular Exponentiation I

One useful arithmetic operation that is greatly simplified is modular exponentiation.

Say we want to compute

 $\alpha^n \bmod m$

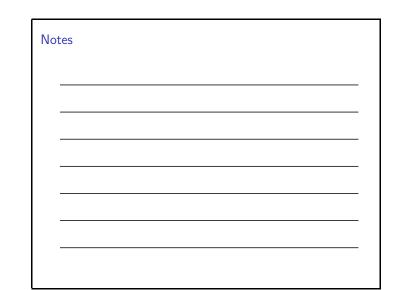
where n is a very large integer. We could simply compute

 $\underbrace{\alpha \cdot \alpha \cdots \alpha}_{n \text{ times}}$

We make sure to ${\rm \bf mod}$ each time we multiply to prevent the product from growing too big. This requires $\mathcal{O}(n)$ operations.

We can do better. Intuitively, we can perform a *repeated squaring* of the base,

 $\alpha, \alpha^2, \alpha^4, \alpha^8, \ldots$



Notes

Modular Exponentiation II

requiring $\log n$ operations instead.

Formally, we note that

$$\begin{aligned} \alpha^n &= \alpha^{b_k 2^k} + b_{k-1} 2^{k-1} + \dots + b_1 2 + b_0 \\ &= \alpha^{b_k 2^k} \times \alpha^{b_{k-1} 2^{k-1}} \times \dots \times \alpha^{2b_1} \times \alpha^{b_0} \end{aligned}$$

So we can compute α^n by evaluating each term as

$$\alpha^{b_i 2^i} = \begin{cases} \alpha^{2^i} & \text{if } b_i = 1\\ 1 & \text{if } b_i = 0 \end{cases}$$

We can save computation because we can simply square previous values:

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

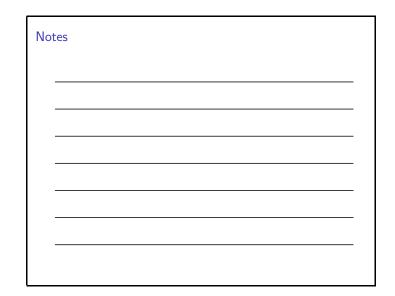
Modular Exponentiation III

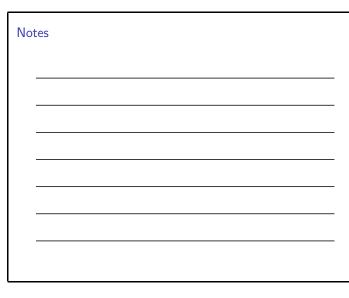
We still evaluate each term independently however, since we will need it in the next term (though the accumulated value is only multiplied by 1).

Modular Exponentiation IV

MODULAR EXPONENTIATION

| | INPUT : Integers α, m and $n = (b_k b_{k-1} \dots b_1 b_0)$ in binary. |
|----------------|---|
| | OUTPUT : $\alpha^n \mod m$ |
| 1 | $term = \alpha$ |
| 2 | |
| 3 | $product = \alpha$ |
| 4 | END |
| 5 | ELSE |
| 6 | product = 1 |
| 7 | END |
| 8 | For $i = 1 \dots k$ do |
| 9 | $term = term \times term \mod m$ |
| 10 | IF $(b_i = 1)$ THEN |
| 11 | $product = product \times term \mod m$ |
| 10 11 12 | END |
| 13 | END |
| 14 | output product |
| _ | |
| | |







Binary Exponentiation Example

Example

Compute $12^{26} \mod 17$ using Modular Exponentiation.

| 1 | 1 | 0 | 1 | 0 | $=(26)_2$ |
|---|----|----|---|----|-----------|
| 4 | 3 | 2 | 1 | - | i |
| 1 | 16 | 13 | 8 | 12 | term |
| 9 | 9 | 8 | 8 | 1 | product |

Thus,

 $12^{26} \mod 17 = 9$

Euclid's Algorithm

Recall that we can find the \gcd (and thus $\operatorname{lcm})$ by finding the prime factorization of the two integers.

However, the only algorithms known for doing this are exponential (indeed, computer security *depends* on this).

We can, however, compute the \gcd in polynomial time using Euclid's Algorithm.

Euclid's Algorithm I

Consider finding the $\gcd(184,1768).$ Dividing the large by the smaller, we get that

 $1768 = 184 \cdot 9 + 112$

Using algebra, we can reason that any divisor of $184~{\rm and}~1768$ must also be a divisor of the remainder, $112.~{\rm Thus},$

gcd(184, 1768) = gcd(184, 112)

| Notes | | | |
|-------|------|------|--|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Notes

Euclid's Algorithm II

Intuition Continuing with our division we eventually get that

 $\begin{array}{rcl} \gcd(184,1768) &=& \gcd(184,112) \\ &=& \gcd(184,72) \\ &=& \gcd(184,40) \\ &=& \gcd(184,24) \\ &=& \gcd(184,16) \\ &=& \gcd(184,8)=8 \end{array}$

This concept is formally stated in the following Lemma.

Lemma

Let a = bq + r, $a, b, q, r \in \mathbb{Z}$, then

 $\gcd(a,b)=\gcd(b,r)$

Euclid's Algorithm III

Intuition

The algorithm we present here is actually the *Extended* Euclidean Algorithm. It keeps track of more information to find integers such that the \gcd can be expressed as a *linear combination*.

Theorem

If a and b are positive integers, then there exist integers $\boldsymbol{s}, \boldsymbol{t}$ such that

gcd(a,b) = sa + tb

| | INPUT : Two positive integers a, b. |
|--------|---|
| | OUTPUT : $r = \gcd(a, b)$ and s, t such that $sa + tb = \gcd(a, b)$. |
| 1 | $a_0 = a, \ b_0 = b$ |
| 2 | $t_0 = 0, t = 1$ |
| 3 | $s_0 = 1, \ s = 0$ |
| 4 | $q = \lfloor \frac{a_0}{b_0} \rfloor$ |
| 5 | $r = a_0 - qb_0$ |
| 6 | while $r > 0$ do |
| 7 | $temp = t_0 - qt$ |
| 8 | $t_0 = t, t = temp$ |
| 9 | $temp = s_0 - qs$ |
| 0 | $s_0 = s, s = temp$ |
| 1 | $a_0 = b_0, \ b_0 = r$ |
| 2 | $q = \lfloor \frac{a_0}{b_0} floor$, $r = a_0 - qb_0$ |
| 3 | IF $r > 0$ then |
| 3 4 | gcd = r |
| 5 | END |
| 6 | END |
| 7 | output gcd, s, t |

Algorithm 1: EXTENDEDEUCLIDIANALGORITHM

| Notes | | | | |
|-------|--|--|--|--|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |



| Euclid's Algor Example | ithm | I | | | | | | | |
|---------------------------|-------|-------|-------|-------|-------|--------|------|-----|--|
| | a_0 | b_0 | t_0 | t | s_0 | s | q | r | |
| | 27 | 58 | 0 | 1 | 1 | 0 | 0 | 27 | |
| | 58 | 27 | 1 | 0 | 0 | 1 | 2 | 4 | |
| | 27 | 4 | 0 | 1 | 1 | -2 | 6 | 3 | |
| | 4 | 3 | 1 | -6 | -2 | 13 | 1 | 1 | |
| | 3 | 1 | -6 | 7 | 13 | -15 | 3 | 0 | |
| Therefore, | gcc | l(27, | 58) = | = 1 = | = (-1 | 5)27 - | + (7 |)58 | |

| Example | | | | | | | | | | | |
|--|-------------------------|------------------------|---|--|-------------|--------------|--------------|----------------------|--|--|--|
| Compute $\gcd(25480,26775)$ and find s,t such that | | | | | | | | | | | |
| gcd(25480, 26775) = 25480s + 26775t | | | | | | | | | | | |
| | | | | <i>'</i> | | | | | | | |
| | | | | | | | | | | | |
| | 0.0 | h_0 | to | + | 80 | 6 | a | r | | | |
| | a ₀ 25480 | b_0 26775 | $\begin{array}{c} t_0 \\ 0 \end{array}$ | $\begin{bmatrix} t \\ 1 \end{bmatrix}$ | s_0 1 | s 0 | q 0 | r 25480 | | | |
| | | | | | - | | | | | | |
| | 25480 | 26775 | 0 | 1 | 1 | 0 | 0 | 25480 | | | |
| | 25480 26775 | 26775 25480 | 0 1 | 1 0 | 1 0 | 0 1 | 0 1 | 25480 1295 | | | |
| | 25480 26775 25480 | 26775 25480 1295 | 0 1 0 | 1 0 1 | 1 0 1 | 0 1 -1 | 0 1 19 | 25480 1295 875 | | | |

gcd(25480, 26775) = 35 = (62)25480 + (-59)26775

Euclid's Algorithm

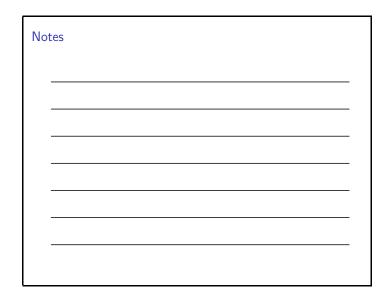
Comments

In summary:

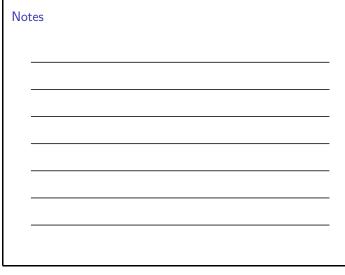
- \blacktriangleright Using the Euclid's Algorithm, we can compute $r=\gcd(a,b),$ where a,b,r are integers.
- ▶ Using the Extended Euclide's Algorithm, we can compute the integers r, s, t such that gcd(a, b) = r = sa + tb.

We can use the Extended Euclide's Algorithm to:

- Compute the inverse of an integer a modulo m, where gcd(a,m)=1. (The inverse of a exists and is unique modulo m when gcd(a,m)=1.)
- \blacktriangleright Solve an equation of linear congruence $ax \equiv b ({\rm mod}\ m),$ where $\gcd(a,m){=}1$







Euclid's Algorithm Computing the inverse

Problem: Compute the inverse of *a* modulo *m* with gcd(a, m)=1, that is find a^{-1} such that $a.a^{-1} \equiv 1 \pmod{m}$

 $gcd(a,m) = 1 \Rightarrow 1 = sa + tm.$ Using the EEA, we can find s and t.

 $1=sa+tm\equiv sa(\mathrm{mod}\ m)\Rightarrow s=a^{-1}.$

Example: Find the inverse of 5 modulo 9.

Problem: Solve $ax \equiv b \pmod{m}$, where gcd(a, m)=1.

Solution:

Euclid's Algorithm

Solving a linear congruence

- Find a^{-1} the inverse of a module m.
- Multiply the two terms of $ax \equiv b \pmod{m}$ by a^{-1} . $ax \equiv b \pmod{m} \Rightarrow$ $a^{-1}ax \equiv a^{-1}b \pmod{m} \Rightarrow$ $x \equiv a^{-1}b \pmod{m}.$

Example: Solve $5x \equiv 6 \pmod{9}$.

Chinese Remainder Theorem

We've already seen an application of linear congruences (pseudorandom number generators).

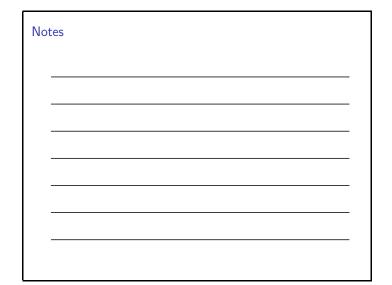
However, systems of linear congruences also have many applications (as we will see).

A system of linear congruences is simply a set of equivalences over a single variable.

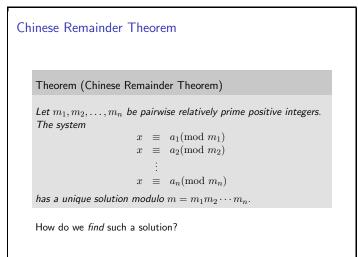
Example

 $x \equiv 5 \pmod{2}$ $x \equiv 1 \pmod{5}$





Notes



Chinese Remainder Theorem

Proof/Procedure

This is a good example of a constructive proof; the construction gives us a procedure by which to solve the system. The process is as follows.

- 1. Compute $m = m_1 m_2 \cdots m_n$.
- 2. For each $k = 1, 2, \ldots, n$ compute

$$M_k = \frac{m}{m_k}$$

- 3. For each k = 1, 2, ..., n compute the inverse, y_k of $M_k \mod m_k$ (note these are *guaranteed* to exist by a Theorem in the previous slide set).
- 4. The solution is the sum

$$x = \sum_{k=1}^{n} a_k M_k y_k$$

Chinese Remainder Theorem I Example

Example

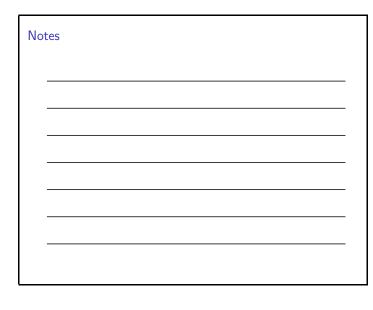
Give the unique solution to the system

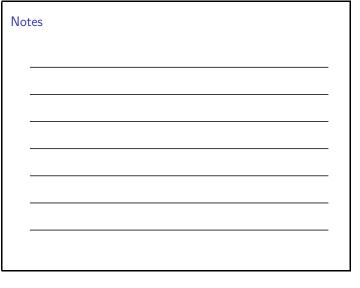
$$\begin{array}{rcl} x &\equiv& 2 \pmod{4} \\ x &\equiv& 1 \pmod{5} \\ x &\equiv& 6 \pmod{7} \\ x &\equiv& 3 \pmod{9} \end{array}$$

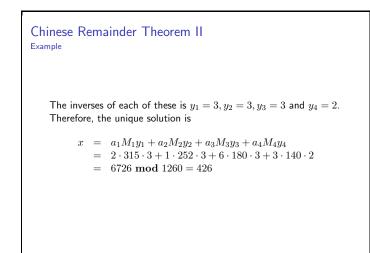
First, $m=4\cdot 5\cdot 7\cdot 9=1260$ and

| M_1 | = | $\frac{1260}{4} = 315$ |
|-------|---|------------------------|
| M_2 | = | $\frac{1260}{5} = 252$ |
| M_3 | = | $\frac{5}{1260} = 180$ |
| M_4 | = | $\frac{1260}{9} = 140$ |
| | | |

| Notes | | | | |
|-------|------|------|------|--|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |







Chinese Remainder Theorem Wait, what?

To solve the system in the previous example, it was necessary to determine the inverses of M_k modulo m_k —how'd we do that?

One way (as in this case) is to try every single element a, $2 \leq a \leq m-1$ to see if

 $aM_k \equiv 1 \pmod{m}$

But there is a more efficient way that we already know how to do—Euclid's Algorithm!

Computing Inverses

Lemma

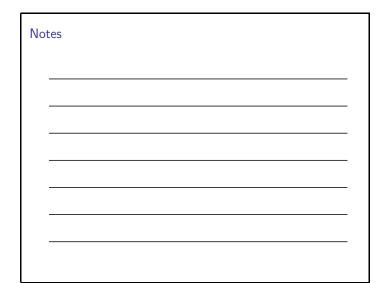
Let a,b be relatively prime. Then the linear combination computed by the Extended Euclidean Algorithm,

gcd(a,b) = sa + tb

gives the inverse of a modulo b; i.e. $s = a^{-1}$ modulo b.

Note that $t = b^{-1} \mod a$.

Also note that it may be necessary to take the modulo of the result.



Notes

Chinese Remainder Representations

In many applications, it is necessary to perform simple arithmetic operations on *very* large integers.

Such operations become inefficient if we perform them bitwise.

Instead, we can use *Chinese Remainder Representations* to perform arithmetic operations of large integers using *smaller* integers saving computations. Once operations have been performed, we can uniquely recover the large integer result.

Chinese Remainder Representations

Lemma

Let m_1,m_2,\ldots,m_n be pairwise relatively prime integers, $m_i\geq 2.$ Let

 $m = m_1 m_2 \cdots m_n$

Then every integer $a, 0 \le a < m$ can be uniquely represented by n remainders over m_i ; i.e.

 $(a \mod m_1, a \mod m_2, \ldots, a \mod m_n)$

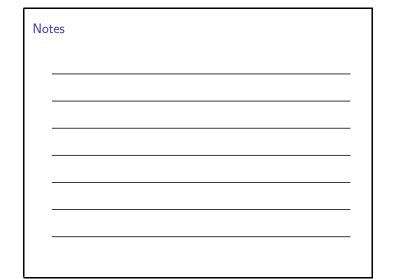
Chinese Remainder Representations I Example

Example

Let $m_1 = 47, m_2 = 48, m_3 = 49, m_4 = 53$. Compute 2, 459, 123 + 789, 123 using Chinese Remainder Representations.

By the previous lemma, we can represent any integer up to 5,858,832 by four integers all less than 53.

First,



Notes

Chinese Remainder Representations II Example

Next,

So we've reduced our calculations to computing (coordinate wise) the addition:

$$\begin{array}{rcl} (36,35,9,29)+(40,3,27,6) &=& (76,38,36,35) \\ &=& (29,38,36,35) \end{array}$$

Chinese Remainder Representations III

 $\mathsf{Exam}\mathsf{N}\mathsf{e}\mathsf{w}$ we wish to recover the result, so we solve the system of linear congruences,

 $\begin{array}{l} x &\equiv 29 ({\rm mod} \ 47) \\ x &\equiv 38 ({\rm mod} \ 48) \\ x &\equiv 36 ({\rm mod} \ 49) \\ x &\equiv 35 ({\rm mod} \ 53) \\ \\ M_1 &= 124656 \\ M_2 &= 122059 \\ M_3 &= 119568 \\ M_4 &= 110544 \\ \\ {\rm uclidean} \ {\rm Algorithm} \ 100 \\ \\ \end{array}$

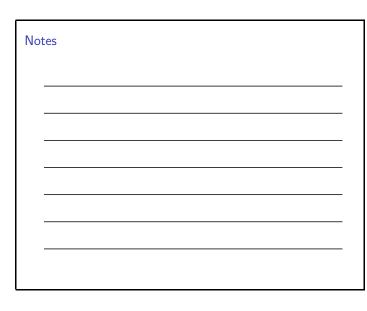
We use the Extended Euclidean Algorithm to find the inverses of each of these w.r.t. the appropriate modulus:

Chinese Remainder Representations IV Example

And so we have that

- $\begin{array}{rcl} x &=& 29(124656 \ {\bf mod} \ 47)4 + 38(122059 \ {\bf mod} \ 48)19 + \\ && 36(119568 \ {\bf mod} \ 49)43 + 35(110544 \ {\bf mod} \ 53)34 \end{array}$
 - = 3,248,246
 - $= \ \ 2,459,123+789,123$

| Notes | | | | |
|-------|------|------|------|--|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |





Caesar Cipher I

Cryptography is the study of secure communication via *encryption*.

One of the earliest uses was in ancient Rome and involved what is now known as a *Caesar cipher*.

This simple encryption system involves a *shift* of letters in a fixed alphabet. Encryption and decryption is simple modular arithmetic.

Caesar Cipher II

In general, we fix an alphabet, Σ and let $m=|\Sigma|.$ Second, we fix an secret key, an integer k such that 0 < k < m. Then the encryption and decryption functions are

$$e_k(x) = (x+k) \mod m$$

 $d_k(y) = (y-k) \mod m$

respectively.

Cryptographic functions must be one-to-one (why?). It is left as an exercise to verify that this Caesar cipher satisfies this condition.

Caesar Cipher

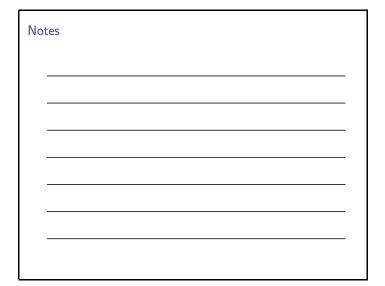
Example

Example

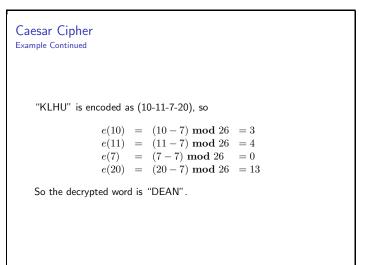
Let $\Sigma=\{A,B,C,\ldots,Z\}$ so m=26. Choose k=7. Encrypt "HANK" and decrypt "KLHU".

"HANK" can be encoded (7-0-13-10), so

so the encrypted word is "OHUR".







Affine Cipher I

Clearly, the Caesar cipher is insecure—the key space is only as large as the alphabet.

An alternative (though still not secure) is what is known as an *affine* cipher. Here the encryption and decryption functions are as follows.

 $e_k(x) = (ax+b) \mod m$ $d_k(y) = a^{-1}(y-b) \mod m$

Question: How big is the key space?

Affine Cipher

Example

Example

To ensure a bijection, we choose m = 29 to be a prime (why?). Let a = 10, b = 14. Encrypt the word "PROOF" and decrypt the message "OBGJLK".

"PROOF" can be encoded as (16-18-15-15-6). The encryption is as follows.

 $\begin{array}{rcl} e(16) &=& (10\cdot 16+14) \ {\bf mod} \ 29 &= 0 \\ e(18) &=& (10\cdot 18+14) \ {\bf mod} \ 29 &= 20 \\ e(15) &=& (10\cdot 15+14) \ {\bf mod} \ 29 &= 19 \\ e(15) &=& (10\cdot 15+14) \ {\bf mod} \ 29 &= 19 \\ e(6) &=& (10\cdot 6+14) \ {\bf mod} \ 29 &= 16 \end{array}$

The encrypted message is "AUPPG".

| Notes | | | | |
|-------|--|------|------|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | _ |
| | | | | |
| | | | | |



```
Affine Cipher
Example Continued
When do we attack? Computing the inverse, we find that a^{-1} = 3.
We can decrypt the message "OBGJLK" (14-1-6-9-11-10) as follows.
e(14) = 3(14-14) \mod 29 = 0 = A
e(1) = 3(1-14) \mod 29 = 19 = T
e(6) = 3(6-14) \mod 29 = 5 = F
e(9) = 3(9-14) \mod 29 = 14 = O
e(11) = 3(11-14) \mod 29 = 20 = U
e(10) = 3(10-14) \mod 29 = 17 = R
```

Public-Key Cryptography I

The problem with the Caesar & Affine ciphers (aside from the fact that they are insecure) is that you still need a secure way to exchange the keys in order to communicate.

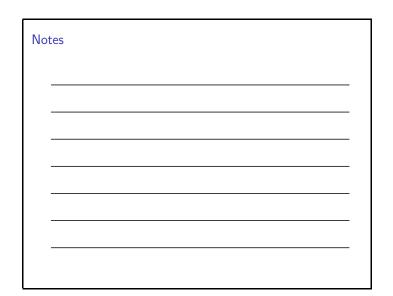
Public key cryptosystems solve this problem.

- One can publish a *public key*.
- Anyone can encrypt messages.
- However, decryption is done with a *private* key.
- ► The system is secure if no one can *feasibly* derive the private key from the public one.
- Essentially, encryption should be computationally easy, while decryption should be computationally hard (without the private key).
- ▶ Such protocols use what are called "trap-door functions".

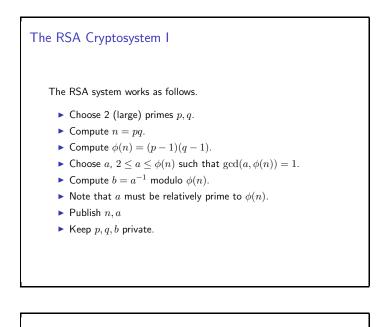
Public-Key Cryptography II

Many public key cryptosystems have been developed based on the (assumed) hardness of *integer factorization* and the *discrete log* problems.

Systems such as the *Diffie-Hellman* key exchange protocol (used in SSL, SSH, https) and the *RSA* cryptosystem are the basis of modern secure computer communication.



Notes



Notes

Notes

Notes

The RSA Cryptosystem II

Then the encryption function is simply

 $e_k(x) = x^a \mod n$

The decryption function is

 $d_k(y) = y^b \bmod n$

The RSA Cryptosystem

Computing Inverses Revisited

Recall that we can compute inverses using the Extended Euclidean Algorithm.

With RSA we want to find $b=a^{-1}\ {\bf mod}\ \phi(n).$ Thus, we compute

 $gcd(a, \phi(n)) = sa + t\phi(n)$

and so $b = s = a^{-1} \mod \phi(n)$.

| The RSA Cryptosystem |
|---|
| Example |
| Let $p = 13, q = 17, a = 47.$ |
| We have |
| • $n = 13 \cdot 17 = 221.$ • $\phi(n) = 12 \cdot 16 = 192.$ |
| \blacktriangleright Using the Euclidean Algorithm, $b=47^{-1}=143$ modulo $\phi(n)$ |
| $e(130) = 130^{47} \text{ mod } 221 = 65$ |
| $d(99) = 99^{143} \ \mathbf{mod} \ 221 = 96$ |

Public-Key Cryptography I Cracking the System

How can we break an RSA protocol? "Simple"—just factor n.

If we have the two factors p and q, we can easily compute $\phi(n)$ and since we already have a, we can also easily compute $b=a^{-1}$ modulo $\phi(n).$

Thus, the security of RSA is contingent on the hardness of *integer factorization*.

Public-Key Cryptography II Cracking the System

If someone were to come up with a polynomial time algorithm for factorization (or build a feasible quantum computer and use Shor's Algorithm), breaking RSA may be a trivial matter. Though this is not likely.

In practice, large integers, as big as 1024 bits are used. 2048 bit integers are considered unbreakable by today's computer; 4096 bit numbers are used by the truly paranoid.

But if you care to try, RSA Labs has a challenge:

http://www.rsasecurity.com/rsalabs/node.asp?id=2091

| Notes | | | |
|-------|------|------|------|
| | | | |
| | | | _ |
| | | | _ |
| | | | |
| | | | |
| | | | |
| | | | _ |
| | | | |
| | | | |

Notes

Public-Key Cryptography Cracking RSA - Example

Example

Let a = 2367 and let n = 3127. Decrypt the message, 1125-2960-0643-0325-1884 (Who is the father of modern computer science?)

Factoring n, we find that $n=53\cdot 59$ so

 $\phi(n) = 52 \cdot 58 = 3016$

 Public-Key Cryptography

 Cracking RSA - Example

 Using the Euclidean algorithm, $b = a^{-1} = 79$. Thus, the decryption function is

 $d(x) = x^{79} \mod 3127$

 Decrypting the message we get that

 $d(1225) = 1225^{79} \mod 3127 = 112$
 $d(2960) = 2960^{79} \mod 3127 = 114$
 $d(0643) = 643^{79} \mod 3127 = 2021$
 $d(0325) = 325^{79} \mod 3127 = 1809$
 $d(1884) = 1884^{79} \mod 3127 = 1407$

 Thus, the message is "ALAN TURING".

