Introduction to Number Theory

Let $a,b \in \mathbf{Z}$ with $a \neq 0$.

$a|b \equiv$ "*a divides b*" $:\equiv \quad (\exists\, c \in \mathbf{Z}: b=ac)$

"There is an integer *c* such that *c* times *a* equals *b*."

If *a* divides *b*, then we say *a* is a *factor* or a *divisor* of *b*, and *b* is a *multiple* of *a*.

We will go through some useful basics of *number theory*.

Vital in many important algorithms today (hash functions, cryptography, digital signatures; in general, on-line security).

Introduction to Number Theory

Common facts:

- $a \mid 0$
- If $a \mid b$ and $a \mid c$, then $a \mid (b+c)$
- If $a \mid b$, then $a \mid bc$ for all integers c
- If $a \mid b$ and $b \mid c$, then $a \mid c$

Corollary: If a, b, c are integers, such that $a \mid b$ and $a \mid c$, then $a \mid mb + nc$ whenever m and n are integers.

Division Algorithm --- Let a be an integer and d a positive integer. Then there are unique integers q and r, with $0 \leq r < d$, such that $a = dq+r$.

r is called the **remainder,** *d* is called the **divisor,** *a* is called the **dividend,** *q* is called the **quotient**

It's really just a theorem, not an algorithm… Only called an "algorithm" for historical reasons.

- If $a = 7$ and $d = 3$, then $q = 2$ and $r = 1$, since $7 = (2)(3) + 1$.
- If $a = -7$ and $d = 3$, then $q = -3$ and $r = 2$, since $-7 = (-3)(3) + 2$.

Proof of Division Algorithm : (we'll use the well-ordering property directly that states that every set of nonnegative integers has a least element.)

Existence: We want to show the existence of q and *r*, with the property that $a = dq+r$, $0 \leq r < d$

Consider the set of non-negative numbers of the form $a - dq$, where q is an integer. By the well-ordering property, S has a least element, $r = a - d\,q_0$.

r is non-negative; also, $r < d$. Otherwise if $r \geq d$, there would be a smaller nonnegative element in S, namely $a-d(q_0+1) \geq 0$. But then $a-d(q_0+1)$, which is smaller than $a-dq_0$, is an element of S, contradicting that $a-dq_0$ was the smallest element of S.

So, it cannot be the case that $r \geq d$, proving the existence of $0 \leq r < d$ and q.

QED

Introduction to Number Theory

b) **Uniqueness**

Suppose ∃ q, Q, R $0 \leq r, R < d$ such that $a = dq + r$ and $a = dQ + R$.

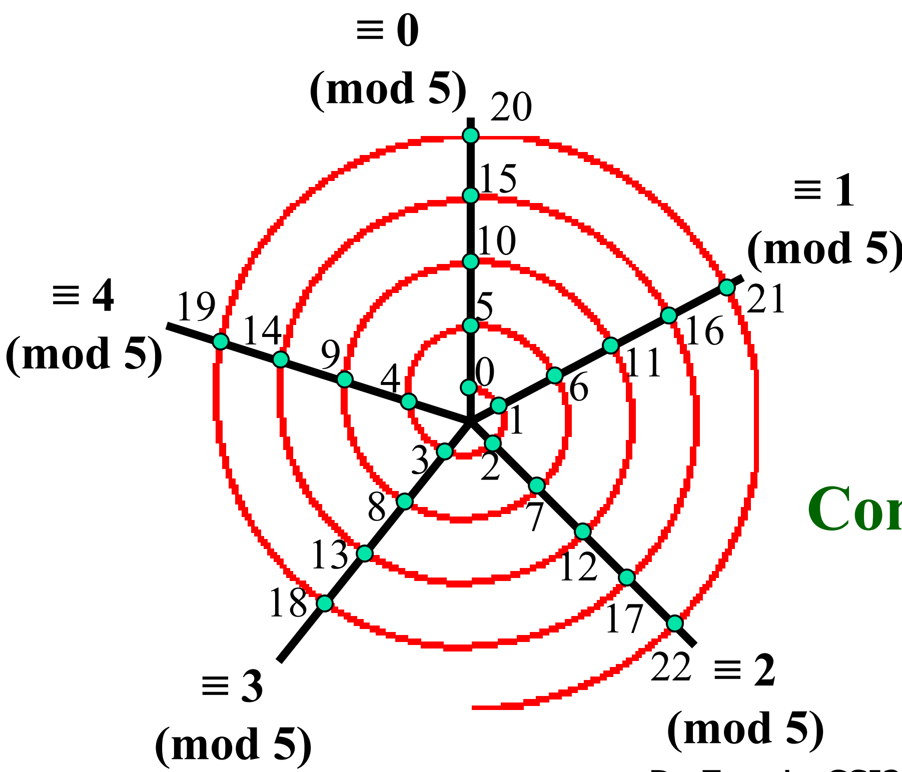
Without loss of generality we may assume that $q \leq Q$. Subtracting both equations we have: $d(q-Q) = (R - r)$. So d divides (R-r); so, either $|d| \leq |(R - r)|$ or $(R - r) = 0$; Since $0 \leq r, R < d$ then $-d < R - r < d$ i.e., $|R-r| < d$, thus we must have $R - r = 0$.

So, $R = r$. Substituting into the original two equations, we have $dq = d Q$ (note $d \neq 0$) and thus $q=Q$, proving uniqueness.

Modular Arithmetic

If *a* and *b* are integers and *m* is a positive integer, then **"*a* is *congruent to b modulo m*" if *m* divides *a-b***

(denoted**: $a \equiv b \pmod{m}$; **a mod m = b mod m)



As 6 divides 17-5, 17 is congruent to 5 modulo 6, $17 \equiv 5 \pmod{6}$

Congruence classes modulo 5.

Modular Arithmetic

Theorem: Let m be a positive integer. The integers a and b are congruent modulo m if and only if there is an integer k such that $a = b + km$

Theorem: Let m be a positive integer. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then $a+c \equiv (b+d) \pmod{m}$ and $ac \equiv bd \pmod{m}$

Hashing Functions

- Also known as:
  - *hash functions*, *hash codes*, or just *hashes*.
- Two major uses:
  - Indexing *hash tables*
    - Data structures which support O(1)-time access.
  - Creating short unique IDs for long documents.
    - Used in *digital signatures* – the short ID can be signed, rather than the long document.

Hash Functions

- Example: Consider a record that is identified by the SSN (9 digits) of the customer.
- How can we assign a memory location to a record so that later on it's easy to locate and retrieve such a record?
- Solution to this problem → a good hashing function.
- Records are identified using a key (k), which uniquely identifies each record.
- If you compute the hash of the same data at different times, you should get the same answer – if not then the data has been modified.

Hash Function Requirements

- A hash function h: $A \rightarrow B$ is a map from a set A to a smaller set B (*i.e.*, $|A| \geq |B|$).

- An effective hash function should have the following properties:
  - It should cover (be *onto*) its codomain B.
  - It should be efficient to calculate.
  - The cardinality of each pre-image of an element of B should be about the same.
    - $\forall b_1, b_2 \in B$: $|h^{-1}(b_1)| \approx |h^{-1}(b_2)|$
    - That is, elements of B should be generated with roughly uniform probability.
  - Ideally, the map should appear random, so that clearly "similar" elements of A are not likely to map to the same (or similar) elements of B.

Hash Function Requirements

Why are these important?

- To make computations fast and efficient.
- So that any message can be hashed.
- To prevent a message being replaced with another with the same hash value.
- To prevent the sender claiming to have sent x_2 when in fact the message was x_1.

Hash Function Requirements

- Furthermore, for a *cryptographically secure* hash function:
  - Given an element $b \in B$, the problem of finding an $a \in A$ such that $h(a)=b$ should have average-case time complexity of $\Omega(|B|^c)$ for some $c>0$.
    - This ensures that it would take exponential time in the length of an ID for an opponent to "fake" a different document having the same ID.

A Simple Hash Using **mod**

- Let the domain and codomain be the sets of all natural numbers below certain bounds:

  $$A = \{a \in \mathbf{N} \mid a < a_{\text{lim}}\}, B = \{b \in \mathbf{N} \mid b < b_{\text{lim}}\}$$

- Then an acceptable (although not great!) hash function from A to B (when $a_{\text{lim}} \geq b_{\text{lim}}$) is $h(a) = a \bmod b_{\text{lim}}$.

- It has the following desirable hash function properties:
  - It covers or is *onto* its codomain B (its range is B).
  - When $a_{\text{lim}} \gg b_{\text{lim}}$, then each $b \in B$ has a preimage of about the same size,
    - Specifically, $|h^{-1}(b)| = \lfloor a_{\text{lim}}/b_{\text{lim}} \rfloor$ or $\lceil a_{\text{lim}}/b_{\text{lim}} \rceil$.

A Simple Hash Using **mod**

- However, it has the following limitations:
  - It is *not* very random. Why not?

For example, if all a's encountered happen to have the same residue mod $b_{\lim}$, they will all map to the same b! (see also "spiral view")

  - It is definitely *not* cryptographically secure.
    - Given a b, it is easy to generate a's that map to it. How?

We know that for any $n \in \mathbf{N}$, $h(b + n\, b_{\lim}) = b$.

Hash Function: Collision

- Because a hash function is not one-to-one (there are more possible keys than memory locations) more than one record may be assigned to the same location → we call this situation a **collision**.
- What to do when a collision happens?
- One possible way of solving a collision is to assign the first free location following the occupied memory location assigned by the hashing function.
- There are other ways… for example chaining (At each spot in the hash table, keep a linked list of keys sharing this hash value, and do a sequential search to find the one we need.)

Digital Signature Application

- Many digital signature systems use a cryptographically secure (but public) hash function *h* which maps arbitrarily long documents down to fixed-length (*e.g.*, 1,024-bit) "fingerprint" strings.
- Document signing procedure:
  - Given a document a to sign, quickly compute its hash $b = h(a)$.
  - Compute a certain function $c = f(b)$ that is known only to the signer
    - This step is generally slow, so we don't want to apply it to the whole document.
  - Deliver the original document together with the digital signature c.
- Signature verification procedure:
  - Given a document a and signature c, quickly find a's hash $b = h(a)$.
  - Compute $b' = f^{-1}(c)$. (Possible if f's inverse f^{-1} is made public (but not f ☺).)
  - Compare b to b'; if they are equal then the signature is valid.

What if h was not cryptographically secure?

Note that if h were not cryptographically secure, then an opponent could easily forge a different document a′ that hashes to the <u>same</u> value b, and thereby attach someone's digital signature to a different document than they actually signed, and fool the verifier!

Pseudorandom numbers

Computers cannot generate truly random numbers – that's why we call them pseudo-random numbers!

- **Linear Congruential Method:** Algorithm for generating pseudorandom numbers.
- Choose 4 integers
  - **Seed** x_0: starting value
  - **Modulus** m: maximum possible value
  - **Multiplier** a: such that $2 \leq a < m$
  - **Increment** c: between 0 and m
- In order to generate a sequence of pseudorandom numbers, $\{x_n \mid 0 \leq x_n < m\}$, apply the formula:

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

Pseudorandom numbers

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

  Let $x_0 = 3$, $m = 9$, $a = 7$, and $c = 4$

- $x_1 = 7x_0+4 = 7*3+4 = 25 \bmod 9 = 7$
- $x_2 = 7x_1+4 = 7*7+4 = 53 \bmod 9 = 8$
- $x_3 = 7x_2+4 = 7*8+4 = 60 \bmod 9 = 6$
- $x_4 = 7x_3+4 = 7*6+4 = 46 \bmod 9 = 1$
- $x_5 = 7x_4+4 = 7*1+4 = 46 \bmod 9 = 2$
- $x_6 = 7x_5+4 = 7*2+4 = 46 \bmod 9 = 0$
- $x_7 = 7x_6+4 = 7*0+4 = 46 \bmod 9 = 4$
- $x_8 = 7x_7+4 = 7*4+4 = 46 \bmod 9 = 5$

Pseudorandom numbers

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

Let $x_0 = 3$, $m = 9$, $a = 7$, and $c = 4$

This sequence generates:

3, 7, 8, 6, 1, 2, 0, 4, 5, 3 , 7, 8, 6, 1, 2, 0, 4, 5, 3

- Note that it repeats!
- But it selects all the possible numbers before doing so

- The common algorithms today use $m = 2^{32}$-1
  - You have to choose 4 billion numbers before it repeats
- Multiplier 7^5 = 16,807 and increment c=0 (pure multiplicative generator)

Cryptology
(secret messages)

- The Caesar cipher: Julius Caesar used the following procedure to encrypt messages
- A function f to encrypt a letter is defined as: $f(p) = (p+3) \bmod 26$
  - Where p is a letter (0 is A, 1 is B, 25 is Z, etc.)
- Decryption: $f^{-1}(p) = (p-3) \bmod 26$
- This is called a substitution cipher
  - You are substituting one letter with another

The Caesar cipher

- Encrypt "go cavaliers"
  - Translate to numbers: g = 6, o = 14, etc.
    - Full sequence: 6, 14, 2, 0, 21, 0, 11, 8, 4, 17, 18
  - Apply the cipher to each number: $f(6) = 9$, $f(14) = 17$, etc.
    - Full sequence: 9, 17, 5, 3, 24, 3, 14, 11, 7, 20, 21
  - Convert the numbers back to letters 9 = j, 17 = r, etc.
    - Full sequence: jr wfdydolhuv

- Decrypt "jr wfdydolhuv"
  - Translate to numbers: j = 9, r = 17, etc.
    - Full sequence: 9, 17, 5, 3, 24, 3, 14, 11, 7, 20, 21
  - Apply the cipher to each number: $f^{-1}(9) = 6$, $f^{-1}(17) = 14$, etc.
    - Full sequence: 6, 14, 2, 0, 21, 0, 11, 8, 4, 17, 18
  - Convert the numbers back to letters 6 = g, 14 = 0, etc.
    - Full sequence: "go cavaliers"

Rot13 encoding

A Caesar cipher, but translates letters by 13 instead of 3

- Then, apply the same function to decrypt it, as 13+13=26 (Rot13 stands for "rotate by 13")

- Example:

```
>echo Hello World | rot13
Uryyb Jbeyq
> echo Uryyb Jbeyq | rot13
Hello World
```

Fundamental Theorem of Arithmetic

A positive integer *p* is prime if the only positive factors of *p* are 1 and *p*. *(If there are other factors, it is composite, note that 1 is not prime! It's not composite either – it's in its own class)*

Fundamental Theorem of Arithmetic:

Every positive integer greater than 1 can be uniquely written as a prime or as the product of two or more primes where the prime factors are written in order of non-decreasing size

primes are the *building blocks* of the natural numbers.

Fundamental Theorem of Arithmetic

Proof of Fundamental theorem of arithmetic: (use Strong Induction)

Show that if n is an integer greater than 1, then *n* can be written as the product of primes.

- Base case – P(2) 2 can be written as 2 (the product of itself)

- Inductive Hypothesis - Assume P(j) is true for $\forall\ 2 \leq j \leq k$, j integer and prove that P(k+1) is true.

a) If k+1 is prime then it's the product of itself, thus P(k+1) true;

b) If k+1 is a composite number and it can be written as the product of two positive integers *a* and *b,* with $2 \leq a \leq b \leq k+1$. By the inductive hypothesis, a and b can be written as the product of primes, and so does k+1 ,

Missing Uniqueness proof, it needs more knowledge, soon…

Fundamental Theorem of Arithmetic

Theorem: If n is a composite integer, then n has a prime divisor less than or equal to the square root of n

Proof:

Since n is composite, it has a factor a such that $1<a<n$. Thus, $n = ab$, where a and b are positive integers greater than 1.

Either $a \le \sqrt{n}$ or $b \le \sqrt{n}$ (Otherwise, $ab > \sqrt{n} * \sqrt{n} > n$. Contradiction.) Thus, n has a divisor not exceeding $\sqrt{n}$. This divisor is either prime or a composite. If the latter, then it has a prime factor (by the FTA). In either case, n has a prime factor less than $\sqrt{n}$ •

- E.g., show that 113 is prime.
- Solution
  - The only prime factors less than $\sqrt{113} = 10.63$ are 2, 3, 5, and 7
  - None of these divide 113 evenly
  - Thus, by the fundamental theorem of arithmetic, 113 must be prime

Mersenne numbers

Mersenne number: any number of the form 2^n-1

Mersenne prime: any prime of the form 2^p-1, where p is also a prime.

- Example: 2^5-1 = 31 is a Mersenne prime
- But 2^{11}-1 = 2047 is not a prime (23*89)

If M is a Mersenne prime, then M(M+1)/2 is a perfect number

- A perfect number equals the sum of its divisors
- Example: 2^3-1 = 7 is a Mersenne prime, thus 7*8/2 = 28 is a perfect number
  - 28 = 1+2+4+7+14
- Example: 2^5-1 = 31 is a Mersenne prime, thus 31*32/2 = 496 is a perfect number

496 = 2*2*2*2*31 ➔ 1+2+4+8+16+31+62+124+248 = 496

The largest primes found are Mersenne primes.

- Since, 2^p-1 grows fast, and there is an extremely efficient test – Lucas-Lehmer test – for determining if a Mersenne prime is prime

GCD and LCM of Two Integers

- The greatest common divisor of two integers *a* and *b* is the largest integer *d* such that *d* | a and *d* | b, denoted by gcd(a,b)
- Two numbers are *relatively prime* if they don't have any common factors (other than 1), that is gcd (a,b) = 1
- The least common multiple of the positive integers *a* and *b* is the smallest positive integer that is divisible by both *a* and *b*. Denoted by lcm (*a*, *b*).

Given two numbers *a* and *b*, rewrite them as:

$$a = p_1^{a_1} p_2^{a_2} \ldots p_n^{a_n}, \quad b = p_1^{b_1} p_2^{b_2} \ldots p_n^{b_n}$$

The gcd and the lcm are computed by the following formulas:

$$\gcd(a,b) = p_1^{\min(a_1,b_1)} p_2^{\min(a_2,b_2)} \ldots p_n^{\min(a_n,b_n)}$$

$$\operatorname{lcm}(a,b) = p_1^{\max(a_1,b_1)} p_2^{\max(a_2,b_2)} \ldots p_n^{\max(a_n,b_n)}$$

GCD and LCM of Two Integers

lcm(10, 25) = 50

What is lcm (95256, 432)?

- $95256 = 2^3 3^5 7^2$, $432 = 2^4 3^3$
- lcm $(2^3 3^5 7^2, 2^4 3^3) = 2^{\max(3,4)} 3^{\max(5,3)} 7^{\max(2,0)} = 2^4\, 3^5\, 7^2 = 190512$

What is gcd (95256, 432)?

- gcd $(2^3 3^5 7^2, 2^4 3^3) = 2^{\min(3,4)} 3^{\min(5,3)} 7^{\min(2,0)} = 2^3\, 3^3\, 7^0 = 216$

Theorem: Let a and b be positive integers.

Then $a*b = \gcd(a,b) * \mathrm{lcm}(a, b)$.

Finding GCDs by comparing prime factorizations is not necessarily a good algorithm (can be difficult to find prime factors are! And, no fast algorithm for factoring is known. (except …)

Euclid: For all integers a, b, $\gcd(a, b) = \gcd((a \bmod b), b)$.

Sort a,b so that a>b, and then (given b>1) (a mod b) < a, so problem is simplified.

GCD and LCM of Two Integers

Theorem: Let a =bq+r, where a,b,q,and r are integers. Then gcd(a,b) = gcd(b,r)

Proof: Suppose *a* and *b* are the natural numbers whose gcd has to be determined. And suppose the remainder of the division of *a* by *b* is *r*. Therefore $a = qb + r$ where *q* is the quotient of the division.

- Any common divisor of *a* and *b* is also a divisor of *r*. To see why this is true, consider that *r* can be written as $r = a - qb$. Now, if there is a common divisor *d* of *a* and *b* such that $a = sd$ and $b = td$, then $r = (s-qt)d$. Since all these numbers, including $s-qt$, are whole numbers, it can be seen that *r* is divisible by *d*.
- The above analysis is true for any divisor *d*; thus, the greatest common divisor of *a* and *b* is also the greatest common divisor of *b* and *r*.

GCD and LCM of Two Integers

Before we get to two Additional Applications:

1 - Performing arithmetic with large numbers
2 - Public Key System

We require additional key results in Number Theory

- **Theorem 1:**
  - $\forall a,b$ *integers*, $a,b > 0$: $\exists s,t$: $\gcd(a,b) = sa + tb$
- **Lemma 1:**
  - $\forall a,b,c > 0$: $\gcd(a,b)=1$ and $a \mid bc$, then $a \mid c$
- **Lemma 2:**
  - If p is prime and $p \mid a_1 a_2 \ldots a_n$ (integers a_i), then $\exists i$: $p \mid a_i$.
- **Theorem 2:**
  - If $ac \equiv bc \pmod{m}$ and $\gcd(c,m)=1$, then $a \equiv b \pmod{m}$.

GCD and LCM of Two Integers

Theorem 1: $\forall a \geq b \geq 0 \ \exists s,t$: $\gcd(a,b) = sa + tb$

Proof: By induction over the value of the larger argument a.

Base case: If b=0 or a=b then gcd(a,b)= a and thus $\gcd(a,b) = sa + tb$ where $s = 1$, $t = 0$. Therefore Theorem true for base case.

Inductive step: From Euclid theorem, we know that if $c = a \bmod b$, (i.e. $a = kb + c$ for some integer k, and thus $c = a - kb$.) then $\gcd(a,b) = \gcd(b,c)$.

Since $b < a$ and $c < b$, *then* by the strong inductive hypothesis, we can deduce that $\exists uv$: $\gcd(b,c) = ub + vc$.

Substituting for $c = a - kb$, we obtain $ub + v(a - kb)$, which we can regroup to get $va + (u - vk)b$.

So, for $s = v$, and let $t = u - vk$, we have $\gcd(a,b) = sa + tb$. *This finishes the* induction step.

GCD and LCM of Two Integers

Lemma 2: If p is a prime and $p|a_1 \ldots a_n$ then $\exists i$: $p|a_i$.

Proof: We use strong induction on the value n.

Base case: $n=1$ Obviously the lemma is true, since $p|a_1$ implies $p|a_1$.

Inductive case: Suppose the lemma is true for all $n \le k$ and suppose $p|a_1 \ldots a_{k+1}$. If $p|m$ where $m=a_1 \ldots a_k$ then by induction p divides one of the a_i's for i=1, …k, and we are done.

Otherwise, we have $p|ma_{k+1}$ but $\neg(p|m)$. Since m is not a multiple of p, and p has no factors, m has no common factors with p, thus $\gcd(m,p)=1$. So, by applying lemma 1, $p|a_{k+1}$. This end the proof of the inductive step ■

the Fundamental Theorem of Arithmetic: Uniqueness

The “other” part of proving the Fundamental Theorem of Arithmetic.

“The prime factorization of any number *n* is unique.”

Theorem: If $p_1 \ldots p_s = q_1 \ldots q_t$ are equal products of two non decreasing sequences of primes, then $s = t$ and $p_i = q_i$ for all i.

Proof:

We proceed with a proof by contradiction. We assume that $p_1 \ldots p_s = q_1 \ldots q_t$ however there i such that *for every j,* $p_i \neq q_j$. In fact, and without loss of generality we may assume that all primes in common have already been divided out, and thus may assume that $\forall ij$: $p_i \neq q_j$.

But since $p_1 \ldots p_s = q_1 \ldots q_t$, we clearly have $p_1 / q_1 \ldots q_t$. According to Lemma 2, $\exists j$: $p_1 | q_j$. Since q_j is prime, it has no divisors other than itself and 1, so it must be that $p_i = q_j$. This contradicts the assumption $\forall ij$: $p_i \neq q_j$. The only resolution is that after the common primes are divided out, both lists of primes were empty, so we couldn’t pick out p_1. *In other words, the two lists must have been identical to begin with!*

(primes are the building blocks of numbers)

GCD and LCM of Two Integers

Theorem 2: If $ac \equiv bc \pmod{m}$ and $\gcd(c,m)=1$, then $a \equiv b \pmod{m}$.

Proof: Since $ac \equiv bc \pmod{m}$, this means $m \mid ac-bc$. Factoring the right side, we get $m \mid c(a-b)$. Since $\gcd(c,m)=1$ (c and m are relative prime), lemma 1 implies that $m \mid a-b$, in other words, $a \equiv b \pmod{m}$.

An Application of Theorem 2

Suppose we have a pure-multiplicative pseudo-random number generator $\{x_n\}$ using a multiplier a that is relatively prime to the modulus m.

Then the transition function that maps from x_n to x_{n+1} is bijective. Because if $x_{n+1} = ax_n \bmod m = ax_n' \bmod m$, then $x_n = x_n'$ (by theorem 2). This in turn implies that the sequence of numbers generated cannot repeat until the original number is re-encountered. And this means that on average, we will visit a large fraction of the numbers in the range 0 to $m-1$ before we begin to repeat!

- Intuitively, because the chance of hitting the first number in the sequence is $1/m$, so it will take $\Theta(m)$ tries on average to get to it.
- Thus, the multiplier a ought to be chosen relatively prime to the modulus, to avoid repeating too soon.

GCD and LCM of Two Integers

- A congruence of the form $ax \equiv b \pmod{m}$ is called a *linear congruence*.
  - *Solving* the congruence is to find the x's that satisfy it.
- An *inverse of a, modulo m* is any integer a' such that $a'a \equiv 1 \pmod{m}$.
- If we can find such an a', notice that we can then solve $ax \equiv b$. Enough to multiply both sides by a', giving $a'ax \equiv a'b$,

  thus $1 \cdot x \equiv a'b$, therefore $x \equiv a'b \pmod{m}$.

Theorem 3: If $\gcd(a,m)=1$ and $m>1$, then a has a unique (modulo m) inverse a'.

Proof:

By theorem 1, $\exists st$: $sa+tm = 1$, so $sa+tm \equiv 1 \pmod{m}$. Since $tm \equiv 0 \pmod{m}$, $sa \equiv 1 \pmod{m}$. Thus s is an inverse of $a \pmod{m}$. Theorem 2 guarantees that if $ra \equiv sa \equiv 1$ then $r \equiv s$. Thus this inverse is unique mod m. (All inverses of a are in the same congruence class as s.)

Pseudoprimes

- Ancient Chinese mathematicians noticed that whenever n is prime, $2^{n-1} \equiv 1 \pmod{n}$.
  - Then some also claimed that the converse was true.
- It turns out that the converse is not true!
  - If $2^{n-1} \equiv 1 \pmod{n}$, it doesn't follow that n is prime.
    - $341 = 11 \cdot 31$ do it is not prime, but $2^{340} \equiv 1 \pmod{341}$.

    (not so easy to find the counter example)

If converse was true, what would be a good test for primality?

- Composites n with this property are called *pseudoprimes*.
  - More generally, if $b^{n-1} \equiv 1 \pmod{n}$ and n is composite, then n is called a *pseudoprime to the base b*.

Fermat's Little Theorem

Fermat generalized the ancient observation that $2^{p-1} \equiv 1 \pmod{p}$ for primes p to the following more general theorem:

Theorem: (Fermat's Little Theorem.)

- If p is prime and a is an integer not divisible by p, then
  $a^{p-1} \equiv 1 \pmod{p}$.
- Furthermore, for every integer a
  $a^{p} \equiv a \pmod{p}$.

Carmichael numbers

These are sort of the "ultimate pseudoprimes."

A *Carmichael number* is a composite n such that $a^{n-1} \equiv 1 \pmod{n}$ for all a relatively prime to n.

The smallest few are 561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341.

These numbers are important since they fool the Fermat primality test: They are "Fermat liars".

The Miller-Rabin ('76 / '80) randomized primality testing algorithm eliminates problems with Carmichael problems.

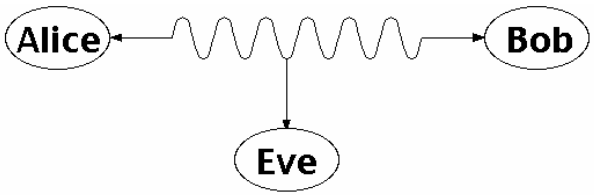
Carmichael numbers

Carmichael numbers have at least three prime factors.

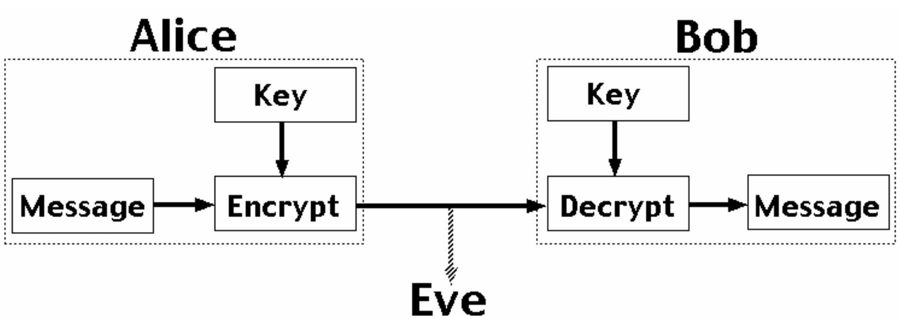
k	
3	$561 = 3 \cdot 11 \cdot 17$
4	$41041 = 7 \cdot 11 \cdot 13 \cdot 41$
5	$825265 = 5 \cdot 7 \cdot 17 \cdot 19 \cdot 73$
6	$321197185 = 5 \cdot 19 \cdot 23 \cdot 29 \cdot 37 \cdot 137$
7	$5394826801 = 7 \cdot 13 \cdot 17 \cdot 23 \cdot 31 \cdot 67 \cdot 73$
8	$232250619601 = 7 \cdot 11 \cdot 13 \cdot 17 \cdot 31 \cdot 37 \cdot 73 \cdot 163$
9	$9746347772161 = 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 31 \cdot 37 \cdot 41 \cdot 641$

The first Carmichael numbers with k=3, 4, 5, … prime factors

RSA and Public-key Cryptography



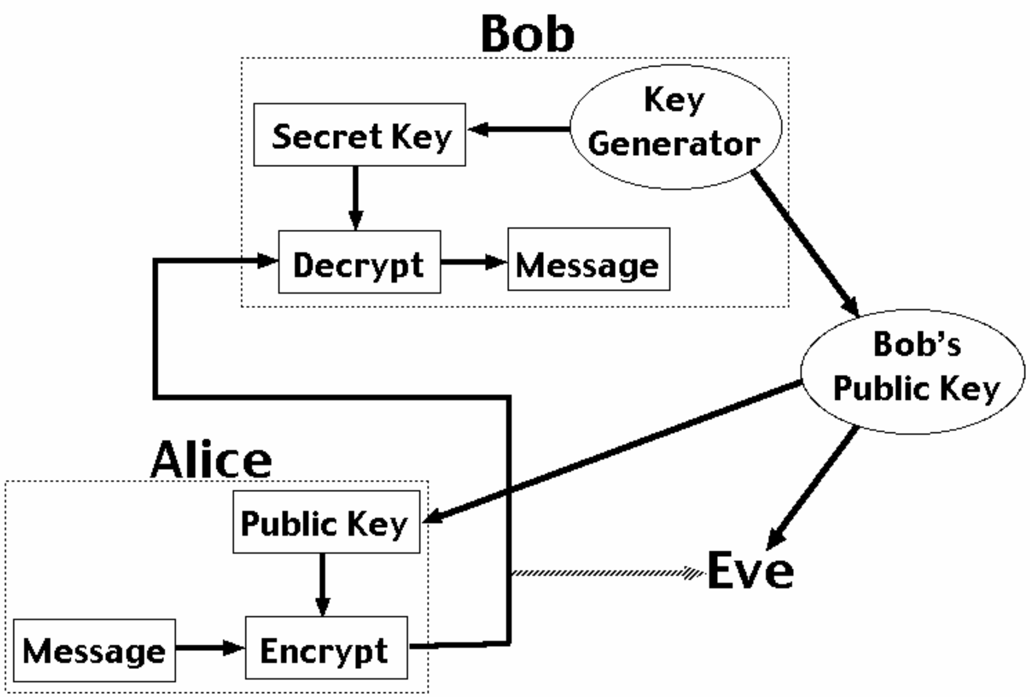
Alice and Bob have never met but they would like to exchange a message. Eve would like to eavesdrop.



They could come up with a good encryption algorithm and exchange the encryption key – but how to do it without Eve getting it? (If Eve gets it, all security is lost.)

CS folks found the solution:
public key encryption. Quite remarkable.

Public Key Encryption: RSA



RSA – Public Key Cryptosystem (why RSA?)

Uses modular arithmetic and large primes → Its security comes from the computational difficulty of factoring large numbers.

Public Key Encryption: RSA

RSA stands for its inventors **R**ivest, **S**hamir, **A**dleman

Normal cryptography:

- communicating parties both need to know a secret key **k**
- sender encodes the message **m** using the key **k** and gets the *ciphertext* **c = f(m,k)**
- the receiver decodes the ciphertext using the key **k** and recovers the original message **m = g(c,k)**

Problem: How to securely distribute the key **k**

- for security reasons, we don't want to use the same k everywhere/for long time

Public Key Encryption: RSA

- In *private key cryptosystems*, the same secret “key” string is used to both encode and decode messages.
  - This raises the problem of how to securely communicate the key strings.
- In *public key cryptosystems*, instead there are two *complementary* keys.
  - One key decrypts the messages that the other one encrypts.
- This means that one key (the *public key*) can be made public, while the other (the *private key*) can be kept secret from everyone.
  - Messages to the owner can be encrypted by anyone using the public key, but can *only* be decrypted by the owner using the private key.
  - Or, the owner can encrypt a message with the private key, and then anyone can decrypt it, and know that *only* the owner could have encrypted it.
    - This is the basis of digital signature systems.
- The most famous public-key cryptosystem is RSA.
  - It is based entirely on number theory

Public Key Encryption: RSA

RSA brings the idea of **public key cryptography**

- the receiver publishes (lets everybody know) its **public key k**
- everybody can send an encoded message **c** to the receiver: **c=f(m,k)**
    - **f** is a known encoding function
- only the receiver that know the secret key **k'** can decode the ciphertext using **m = g(c, k')**
    - the decoding function **g** is also known, just **k'** is not publicly known

So how does it works? What are the keys k and **k'** and the functions **f()** and **g()**?

Public Key Encryption: RSA

Let **p** and **q** be two really large primes (each of several hundred digits)

The public key is a pair **(n,e)** where

n = pq, and **e** is relatively prime to **(p-1)(q-1)**

The encoding function is $f(m,k) = m^e \bmod n$

- assumes you message is represented by an integer **m<n**
- every message **m** can be split into integers $m_1, m_2, \ldots$ and encode those integers separately

The secret (private) key is the number **d** which is an inverse of **e** modulo **(p-1)(q-1)**

The decoding function is $g(c, d) = c^d \bmod n$

The basic idea is that from the knowledge of **n** it is very difficult (exponential in the number of digits) to figure **p** and **q**, and therefore very difficult to figure **d**.

Public Key Encryption: RSA

Hmm, how come that we actually recover the original message?

We want to show that **$g(f(m, k), k') = m$**

$g(f(m,k), k') = (m^e \bmod n)^d \bmod n = m^{ed} \bmod n$

By choice of **e** and **d**, we have **$ed \equiv 1 \bmod (p-1)(q-1)$**, ie **$ed = 1+k(p-1)(q-1)$** for some **$k$**

Let us assume that **$\gcd(m,p) = \gcd(m,q) = 1$**

- that can be checked by the encoding algorithm and handled separately if not true

Then, by Fermat's Little Theorem **$m^{p-1} \equiv 1 \pmod p$ and $m^{p-1} \equiv 1 \pmod q$**

We get **$m^{ed} \equiv m^{1+k(p-1)(q-1)} \equiv m*(m^{p-1})^{k(q-1)} \equiv m*1^{k(q-1)} \equiv m \pmod p$**

Analogously, we get **$m^{ed} \equiv m \pmod q$**

Since **p** and **q** are relatively prime, by the Chinese Remainder Theorem we get **$m^{ed} \equiv m \pmod{pq}$**