Introduction to Elementary Number Theory and Cryptography

CSE 191, Class Note 07
Computer Sci & Eng Dept
SUNY Buffalo

Outline

Introduction to Elementary Number Theory

- Number Theory is one of the oldest branch of mathematics.
- It studies the properties of integers, especially prime numbers.
- There are several simple looking, yet very challenging problems in number theory.
- There are a number of applications in Computer Science. The most important and well known is the RSA Public Key Cryptosystem, which is the basis of virtually all current computer security systems.
- We will study some of the basic topics in number theory, so that we can describe and understand RSA Public Key Cryptosystem.

Some example problems in Number Theory

Fermat's Last Theorem:

For any integer $n \geq 3$, there is no integer solution x, y, z for the equation

$$x^n + y^n = z^n$$

This problem had been open for more than 350 years. It was proved by Andrew Wildes in 1995.

Goldbach's Conjecture:

Every even integer $n \geq 2$ is the sum of two prime numbers.

Example: 4=2+2; 6=3+3; 8=3+5; . . ., 20=3+17; 22 = 3+19= 11+11; . . ..

- It has been verified that this conjecture is true for n up to $1.6 \cdot 10^{18}$.
- British publisher Tony Faber offered a $1,000,000 prize if a proof was submitted before April 2002. The prize was not claimed.
- It remains unsolved today.

Outline

Integer division

- Let a be an integer and d be a positive integer. Then there are unique integers q and r ($0 \leq r < d$), such that $a = dq + r$.
  - This is the the division of a by d.
  - q is the quotient of this division. We write $q = a$ div d.
  - r is the remainder of this division. We write $r = a$ mod d.
  - If $r = 0$, then we say d divides a, and write $d|a$.

Example:

We divide 13 by 3, and get that 13 = 3 $\times$ 4 +1.

- So, 13 div 3 = 4 and 13 mod 3 = 1.
- Since the remainder is not 0, we say that 3 does not divide 13.

Example for integer division

- Divide 27 by 5. What is the quotient and what is the remainder?
- Does 14 divide 98?
- Divide 1000 by 333.
- Does 1111 divide 2345?

Basic properties of division

Theorem:

Let a, b, c be integers. Then:

1. If $a|b$ and $a|c$, then $a|(b+c)$.
2. If $a|b$, then $a|(bc)$ for all integers c.
3. If $a|b$ and $b|c$, then $a|c$.

Proof: (1) Let $d = b$ div a and $d' = c$ div a. Then $b + c = da + d'a = (d + d')a$. So $a|(b + c)$.
(2) Let $d = b$ div a. Then $bc = adc$. So $a|bc$.
(3) Let $d = b$ div a and $d' = c$ div b. Then $c = d'b = d'da$. So $a|c$.

Example for properties of division

Example

Suppose a, b, c are integers such that $a|b$ and $a|c$. Then, for all integers m, n, $a|(mb + nc)$.

Proof: Let $d = b$ div a and $d' = c$ div a. Then:

$$mb + nc = mda + nd'a = (md + nd')a$$

So, $a|mb + nc$.

Note: you should memorize this result.

Outline

Congruence

Definition

Let a and b be integers, and m be a positive integer. Then we say a is congruent to b modulo m if $a \bmod m = b \bmod m$. We write $a \equiv b \pmod{m}$.

Example:

$$13 \bmod 4 = 1 = 21 \bmod 4$$

So, we have $13 \equiv 21 \pmod 4$.

Equivalent definition for congruence

Theorem:

Let a, b be integers and m be a positive integer. Then $a \equiv b \pmod{m}$ if and only if $m|(a-b)$.

Proof: Let $a = mq + r$ and $b = mq' + r'$. With out loss of generality, assume $r \geq r'$. So $a - b = m(q - q') + (r - r')$.

$$
\begin{aligned}
a \equiv b \bmod m \quad &\Leftrightarrow \quad r = r' \text{ (the remainders of } a, b \text{ divided by } m \text{ are the same)} \\
&\Leftrightarrow \quad (a-b) = m(q-q') \\
&\Leftrightarrow \quad m|(a-b)
\end{aligned}
$$

Example for congruence

- Is 101 congruent to 91 mod 9?
- $101 \equiv 91$ (mod ???)
- $100 \equiv 99$ (mod ???)

Addition of congruence

Theorem:

Let m be a positive integer. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then $a + c \equiv b + d \pmod{m}$.

Proof:

$a \equiv b \pmod{m} \Leftrightarrow m|(a-b) \Leftrightarrow (a-b) = k_1 \cdot m$ for some integer k_1.

$c \equiv d \pmod{m} \Leftrightarrow m|(c-d) \Leftrightarrow (c-d) = k_2 \cdot m$ for some integer k_2.

Addition of congruence

Theorem:

Let m be a positive integer. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then $a + c \equiv b + d \pmod{m}$.

Proof:

$a \equiv b \pmod{m} \Leftrightarrow m|(a-b) \Leftrightarrow (a-b) = k_1 \cdot m$ for some integer k_1.

$c \equiv d \pmod{m} \Leftrightarrow m|(c-d) \Leftrightarrow (c-d) = k_2 \cdot m$ for some integer k_2.

Hence: $(a+c) - (b+d) = (a-b) + (c-d) = m \cdot k_1 + m \cdot k_2 = m \cdot (k_1 + k_2)$.

Addition of congruence

Theorem:

Let m be a positive integer. If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then $a + c \equiv b + d \pmod{m}$.

Proof:

$a \equiv b \pmod{m} \Leftrightarrow m|(a-b) \Leftrightarrow (a-b) = k_1 \cdot m$ for some integer k_1.

$c \equiv d \pmod{m} \Leftrightarrow m|(c-d) \Leftrightarrow (c-d) = k_2 \cdot m$ for some integer k_2.

Hence: $(a+c) - (b+d) = (a-b) + (c-d) = m \cdot k_1 + m \cdot k_2 = m \cdot (k_1 + k_2)$.

This means $m|[(a+c) - (b+d)]$. By the equivalent definition of congruence, $(a+c) \equiv (b+d) \pmod{m}$.

Example for addition of congruence

Example

$10001 + 20000005 + 3004 \equiv ?\ (\text{mod } 10)$

Solution:

$$\begin{aligned} 10001 &\equiv 1 (\text{mod } 10) \\ 20000005 &\equiv 5 (\text{mod } 10) \\ 3004 &\equiv 4 (\text{mod } 10) \end{aligned}$$

So $10001+20000005+3004 \equiv 1+5+4\ (\text{mod } 10) \equiv 10\ (\text{mod } 10) \equiv 0\ (\text{mod } 10)$.

Multiplication of congruence

Theorem:

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

Proof:

$a \equiv b \pmod{m} \Leftrightarrow m|(a-b) \Leftrightarrow (a-b) = k_1 \cdot m$ for some integer k_1.

$c \equiv d \pmod{m} \Leftrightarrow m|(c-d) \Leftrightarrow (c-d) = k_2 \cdot m$ for some integer k_2.

Multiplication of congruence

Theorem:

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

Proof:

$a \equiv b \pmod{m} \Leftrightarrow m|(a-b) \Leftrightarrow (a-b) = k_1 \cdot m$ for some integer k_1.

$c \equiv d \pmod{m} \Leftrightarrow m|(c-d) \Leftrightarrow (c-d) = k_2 \cdot m$ for some integer k_2.

Hence:

$ac - bd = ac - ad + ad - bd = a(c-d) + d(a-b) = am \cdot k_2 + dm \cdot k_1 = m \cdot (ak_2 + dk_1)$.

Multiplication of congruence

Theorem:

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

Proof:

$a \equiv b \pmod{m} \Leftrightarrow m|(a-b) \Leftrightarrow (a-b) = k_1 \cdot m$ for some integer k_1.

$c \equiv d \pmod{m} \Leftrightarrow m|(c-d) \Leftrightarrow (c-d) = k_2 \cdot m$ for some integer k_2.

Hence:

$ac - bd = ac - ad + ad - bd = a(c-d) + d(a-b) = am \cdot k_2 + dm \cdot k_1 = m \cdot (ak_2 + dk_1)$.

This means $m|(ac - bd)$. By the equivalent definition of congruence, $ac \equiv bd \pmod{m}$.

Example: 10001 X 20000005 $\equiv$? (mod 13)

Solution: $10001 \equiv 4 \pmod{13}$ and $20000005 \equiv 12 \pmod{13}$

So, 10001 X 20000005 $\equiv$ 4 X 12 (mod 13) $\equiv$ 48 (mod 13) $\equiv$ 9

Addition and residue

Theorem:

Let m be a positive integer and a, b be integers. Then,

$$(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m.$$

Proof: Clearly, we have $a \equiv a \bmod m \pmod m$, and $b \equiv b \bmod m \pmod m$. So, we get that $a + b \equiv (a \bmod m) + (b \bmod m) \pmod m$, which is equivalent to that $(a + b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$.

Theorem:

Let m be a positive integer and a, b be integers. Then,

$$ab \bmod m = ((a \bmod m)(b \bmod m)) \bmod m.$$

The proof is analogous to the previous theorem and so we skip it here.

Example for calculating residue

Example: What is 2008^{2008} mod 3?

$$
\begin{aligned}
2008^{2008} &= (\underbrace{2008 \times \ldots \times 2008}_{\text{2008 times}}) \bmod 3 \\
&= (\underbrace{(2008 \bmod 3) \times \ldots \times (2008 \bmod 3)}_{\text{2008 times}}) \bmod 3 \\
&= (2008 \bmod 3)^{2008} \bmod 3 \\
&= 1^{2008} \bmod 3 = 1
\end{aligned}
$$

Outline

Prime numbers

- A positive integer p (> 1) is called prime if the only positive integers that divide p are 1 and p itself.
- A positive integer (> 1) that is not a prime is called composite.

Example:

Prime: 2, 3, 5, 7, 11, 13, 17, 19
Composite: 4, 6, 8, 9, 10, 12, 14, 15, 16, 18

Fundamental theorem of arithmetic

Theorem:

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

$$n = p_1^{r_1} \times p_2^{r_2} \times \cdots \times p_k^{r_k}$$

which is called the (prime) factorization of n.

Example:

We can write 100= $2 \times 2 \times 5 \times 5 = 2^2 \times 5^2$.
We can write 241= 241.

Example:

625 = ?
891 = ?

Prime factor test

Theorem:

If n is a composite, then n has a prime factor less than or equal to $\sqrt{n}$.

Proof: We prove by contradiction. Suppose that n is a composite and does not have any prime factor $\leq \sqrt{n}$.
By the fundamental theorem of arithmetic, we know that $n = p_1 \cdot p_2 \cdots p_k$, where the prime factors $p_1 > \sqrt{n}, p_2 > \sqrt{n}, \ldots p_k > \sqrt{n}$.
Furthermore, since n is a composite, $k \geq 2$. So we have

$$n = p1 \cdot p2 \cdots p_k > p_1 \cdot p_2 > \sqrt{n} \cdot \sqrt{n} = n$$

Contradiction.

Example: Show 71 is a prime.

Note that $\sqrt{71} = 8.4...$. The only primes $\leq 8.4...$ are 2, 3, 5, 7. So we divide 71 by 2,3,5,7. None of them is a factor of 71. Then we can conclude 71 is a prime.

Number of primes

Theorem: There are infinitely many primes.

Proof: We prove by contradiction. Suppose that there are only a finite number of primes: $p_1, p_2, \ldots, p_n$. Now consider

$$p = p_1 \cdot p_2 \cdots p_n + 1$$

On one hand, p must be a composite since it is greater than any of the above n primes. So by the fundamental theorem of arithmetic, it can be written as the product of two or more primes.

On the other hand, it is easy to verify that any prime p_i $(1 \leq i \leq n)$ cannot divide p since the remainder of the division is 1. Contradiction.

Outline

Greatest common divisor

Definition

Let a and b be integers, not both 0. The largest integer d such that $d|a$ and $d|b$ is called the greatest common divisor of a and b. We write:

$$d = gcd(a, b)$$

Example:

$gcd(30, 6) = 6$ since $6|30$.

Calculating $gcd(a, b)$

How to calculate $gcd(a, b)$?

1. Express a and b as products of powers of increasing primes.
2. Select the prime divisors a and b have in common.
3. For each of the common prime divisor, pick the smaller exponent.
4. Calculate the product of the powers of these common prime divisors, where the exponents are what we just selected.

Example: how can we calculate $gcd(168, 196)$?

- $168 = 2^3 \times 3 \times 7$; and $196 = 2^2 \times 7^2$.
- 2 and 7 are the common prime divisors of 168 and 196.
- For prime divisor 2, we have exponents 3 (for 168) and 2 (for 196). Hence, we select 2; for prime divisor 7, we have exponents 1 (for 168) and 2 (for 196). Hence, we select 1.
- Calculate $2^2 \times 7^1 = 28$. So $gcd(168, 196) = 28$.

Coprime

Definition

The integers a and b are coprime (relatively prime) to each other if $gcd(a, b) = 1$.

Use the method just learned to calculate $gcd(a, b)$, you know whether a and b are coprime to each other.

Example:

- 15 and 25 are not coprime to each other since $gcd(15, 25) = 5$.
- 15 and 24 are not coprime to each other since $gcd(15, 24) = 3$.
- 15 and 28 are coprime to each other since $gcd(15, 28) = 1$.

Coprime: extended definition

Definition

Consider n integers $a_1, a_2, \ldots, a_n$. They are called pairwise coprime if $gcd(a_i, a_j) = 1$ for any $i \neq j$.

Example:

- 15, 17, 25 are not pairwise coprime since $gcd(15, 25) = 5$.
- 15, 17, 28 are pairwise coprime since $gcd(15, 17) = gcd(15, 28) = gcd(17, 28) = 1$.

Least common multiple

Definition

The least common multiple of positive integers a and b is the smallest positive integer that can be divided by both a and b. We denote it by $lcm(a, b)$.

Example:

$lcm(30, 6) = 30$ since $6|30$.

Calculating $lcm(a, b)$

How can we calculate $lcm(a, b)$?

1. Express a and b as products of powers of increasing primes. (Analogous to calculating gcd).
2. Select the prime divisors a and b have in common. (Analogous to calculating gcd)
3. For each of the common prime divisor, pick the larger exponent.
4. Calculate the product of the powers of these common prime divisors, where the exponents are what we just selected, and also all primes that only one of them has. (Different from calculating gcd)

Calculating $lcm(a, b)$

Example: how can we calculate $lcm(168, 196)$?

- $168 = 2^3 \times 3 \times 7$ and $196 = 2^2 \times 7^2$.
- 2 and 7 are the common prime divisors of 168 and 196.
- For prime divisor 2, we (Different from calculating gcd) have exponents 3 (for 168) and 2 (for 196). Hence, we select 3; For prime divisor 7, we have exponents 1 (for 168) and 2 (for 196). Hence, we select 2.
- Calculate $lcm(168, 196) = 2^3 \times 7^2 \times 3 = 1176$.

GCD vs. LCM

Theorem:

Suppose a and b are positive integers. Then $ab = gcd(a, b) \cdot lcm(a, b)$.

Example:

- $gcd(168, 196) = 28$ and $lcm(168, 196) = 1176$.
- $168 \times 196 = 32928 = 28 \times 1176$.

This tells us that if we can calculate the gcd, then we can easily get the lcm, and vice versa.

Outline

Euclidean GCD algorithm

- The gcd (and lcm) algorithms described above rely on finding the factorization of n.
- Although it looks very simple, it is very time consuming even for super computers to find prime factors of large integers.
- This fact (that it is difficult to find prime factors) is the basis of the cryptography (to be discussed later).
- Euclidean algorithm is a much faster algorithm for finding $gcd(a, b)$. It does not rely on prime factorization.

Euclidean GCD algorithm

Euclidean-GCD (a, b: positive integers)

1. $x := a$
2. $y := b$
3. **while** $y \neq 0$
   - $r := x \bmod y$
   - $x := y$
   - $y := r$
4. **return** x ($gcd(a, b)$ is x)

Example of Euclidean GCD algorithm

Example:

$$\begin{aligned} 287 &= 91 \cdot 3 + 14 \\ 91 &= 14 \cdot 6 + 7 \\ 14 &= 7 \cdot 2 \end{aligned}$$

So $gcd(287, 91) = 7$

Example of Euclidean GCD algorithm

Example:

$$\begin{aligned} 287 &= 91 \cdot 3 + 14 \\ 91 &= 14 \cdot 6 + 7 \\ 14 &= 7 \cdot 2 \end{aligned}$$

So $gcd(287, 91) = 7$

Why Euclidean-GCD algorithm work?

Lemma

Let a, b are positive integers and $r = a \bmod b$. Then $gcd(a, b) = gcd(b, r)$.

Proof: We have $a = b \cdot q + r$. Thus $r = a - b \cdot q$.
Let d be a common divisor of a and b. Namely $d|a$ and $d|b$. This implies that $d|(a - b \cdot q)$. Namely $d|r$.
Similarly we can show if $d|b$ and $d|r$, then $d|(b \cdot q + r)$. Namely $d|a$.
Hence $\{a, b\}$ and $\{b, r\}$ have exactly the same set of divisors. So $gcd(a, b) = gcd(b, r)$.

Euclidean GCD algorithm

In the Euclidean-GCD algorithm, let $r_0 = a$ and $r_1 = b$. We have

$$\begin{array}{lcll} r_0 & = & r_1 q_1 + r_2 & 0 \leq r_2 < r_1 \\ r_1 & = & r_2 q_2 + r_3 & 0 \leq r_3 < r_2 \\ \vdots & & & \\ r_{n-2} & = & r_{n-1} q_{n-1} + r_n & 0 \leq r_2 < r_1 \\ r_{n-1} & = & r_n \cdot q_n & \end{array}$$

So

$gcd(a,b) = gcd(r_0, r_1) = gcd(r_1, r_2) = \ldots = gcd(r_{n-2}, r_{n-1}) = gcd(r_{n-1}, r_n) = r_n.$

Bézout's Theorem

Bézout's Theorem

Let a and b be positive integers. Then there exist integers s and t such that $gcd(a, b) = sa + tb$.

By reversing Euclidean GCD algorithm, we can find s and t.

Example: $a = 287, b = 91$. We have:

$$\begin{aligned} 287 &= 91 \cdot 3 + 14 \\ 91 &= 14 \cdot 6 + 7 \\ 14 &= 7 \cdot 2 \end{aligned}$$

So $gcd(287, 91) = 7$. Now going backward:

$$\begin{aligned} 7 &= 91 - 14 \cdot 6 \\ &= 91 - (287 - 91 \cdot 3) \cdot 6 \\ &= -6 \cdot 287 + 19 \cdot 91 \end{aligned}$$

Hence $s = -6$ and $t = 19$.

This process for finding s and t is called Extended Euclidean Algorithm.

Outline

Factoring and Primality Testing Problems

P1: Factoring Problem

Input: an integer X.
Output: Find its prime factorization.

Factoring and Primality Testing Problems

P1: Factoring Problem

Input: an integer X.
Output: Find its prime factorization.

If $X = 117$, the output: $X = 3 \cdot 3 \cdot 13$.

Factoring and Primality Testing Problems

P1: Factoring Problem

Input: an integer X.
Output: Find its prime factorization.

If $X = 117$, the output: $X = 3 \cdot 3 \cdot 13$.

P2: Primality Testing

Input: an integer X.
Output: ”yes” if X is a prime number; ”no” if not.

Factoring and Primality Testing Problems

P1: Factoring Problem

Input: an integer X.
Output: Find its prime factorization.

If $X = 117$, the output: $X = 3 \cdot 3 \cdot 13$.

P2: Primality Testing

Input: an integer X.
Output: ”yes” if X is a prime number; ”no” if not.

- If $X = 117$, output ”no”.

Factoring and Primality Testing Problems

P1: Factoring Problem

Input: an integer X.
Output: Find its prime factorization.

If $X = 117$, the output: $X = 3 \cdot 3 \cdot 13$.

P2: Primality Testing

Input: an integer X.
Output: ”yes” if X is a prime number; ”no” if not.

- If $X = 117$, output ”no”.
- If $X = 456731$, output = ?

Factoring and Primality Testing Problems

- P1 and P2 are related.
- If we can solve P1, we can solve P2 immediately.

Factoring and Primality Testing Problems

- P1 and P2 are related.
- If we can solve P1, we can solve P2 immediately.
- The reverse is not true: even if we know X is not a prime, how to find its prime factors?

Factoring and Primality Testing Problems

- P1 and P2 are related.
- If we can solve P1, we can solve P2 immediately.
- The reverse is not true: even if we know X is not a prime, how to find its prime factors?
- P1 is harder than P2.

Factoring and Primality Testing Problems

- P1 and P2 are related.
- If we can solve P1, we can solve P2 immediately.
- The reverse is not true: even if we know X is not a prime, how to find its prime factors?
- P1 is harder than P2.
- How to solve P1?

Factoring and Primality Testing Problems

- P1 and P2 are related.
- If we can solve P1, we can solve P2 immediately.
- The reverse is not true: even if we know X is not a prime, how to find its prime factors?
- P1 is harder than P2.
- How to solve P1?

Find-Factor(X)

1: **if** X is even **then**
2: return "2 is a factor of X" and stop
3: **end if**
4: **for** $i = 3$ **to** $\sqrt{X}$ **by** $+2$ **do**
5: test **if** X mod $i = 0$, if yes, return "i is a factor of X" and stop
6: **end for**
7: return "X is a prime"

- To solve P1, we call **Find-Factor(X)** to find the smallest prime factor i of X. Then call **Find-Factor(X/i)** ...

- To solve P1, we call **Find-Factor(X)** to find the smallest prime factor i of X. Then call **Find-Factor(X/i)** ...
- The runtime of **Find-Factor**: Suppose the input size is n (Namely X is represented by n bits).

- To solve P1, we call **Find-Factor(X)** to find the smallest prime factor i of X. Then call **Find-Factor(X/i)** ...
- The runtime of **Find-Factor**: Suppose the input size is n (Namely X is represented by n bits).
- Since X is n bits long, the value of X is $\geq 2^{n-1}$.
- In the worst case, we need to perform $\frac{1}{2}\sqrt{2^{n-1}} = \frac{1}{2}(1.414)^{n-1}$ divisions. So this is an exponential time algorithm.
- Minor improvements can be (and had been) made. But basically, we have to perform most of these tests. No poly-time algorithm for Factoring is known.
- It is strongly believed, (but not proven), no poly-time algorithm for solving the Factoring problem exists.

How long does it take?

- Suppose we want to factor a number X with 220 digits.
- Since X is 220 digits long, the value of X is $\geq 10^{219}$.
- So we need to perform $\frac{1}{2}\sqrt{X} > 10^{108}$ divisions.

How long does it take?

- Suppose we want to factor a number X with 220 digits.
- Since X is 220 digits long, the value of X is $\geq 10^{219}$.
- So we need to perform $\frac{1}{2}\sqrt{X} > 10^{108}$ divisions.
- Say we use a super computer with speed of 10^9 divisions/sec.
- This translates into: 10^{99} CPU sec, about $3 \cdot 10^{91}$ years.

How long does it take?

- Suppose we want to factor a number X with 220 digits.
- Since X is 220 digits long, the value of X is $\geq 10^{219}$.
- So we need to perform $\frac{1}{2}\sqrt{X} > 10^{108}$ divisions.
- Say we use a super computer with speed of 10^9 divisions/sec.
- This translates into: 10^{99} CPU sec, about $3 \cdot 10^{91}$ years.
- For comparison: the age of the universe: about $1.5 \cdot 10^{10}$ years.

How long does it take?

- Suppose we want to factor a number X with 220 digits.
- Since X is 220 digits long, the value of X is $\geq 10^{219}$.
- So we need to perform $\frac{1}{2}\sqrt{X} > 10^{108}$ divisions.
- Say we use a super computer with speed of 10^9 divisions/sec.
- This translates into: 10^{99} CPU sec, about $3 \cdot 10^{91}$ years.
- For comparison: the age of the universe: about $1.5 \cdot 10^{10}$ years.
- The number of atoms in the known universe: $\leq 10^{80}$.

How long does it take?

- Suppose we want to factor a number X with 220 digits.
- Since X is 220 digits long, the value of X is $\geq 10^{219}$.
- So we need to perform $\frac{1}{2}\sqrt{X} > 10^{108}$ divisions.
- Say we use a super computer with speed of 10^9 divisions/sec.
- This translates into: 10^{99} CPU sec, about $3 \cdot 10^{91}$ years.
- For comparison: the age of the universe: about $1.5 \cdot 10^{10}$ years.
- The number of atoms in the known universe: $\leq 10^{80}$.
- If every atom in the known universe is a supercomputer and starts at the beginning of the big bang, we have only done $\frac{1.5 \cdot 10^{10} \times 10^{80}}{3 \cdot 10^{91}} = 5\%$ of the needed computations!

How long does it take?

- Suppose we want to factor a number X with 220 digits.
- Since X is 220 digits long, the value of X is $\geq 10^{219}$.
- So we need to perform $\frac{1}{2}\sqrt{X} > 10^{108}$ divisions.
- Say we use a super computer with speed of 10^9 divisions/sec.
- This translates into: 10^{99} CPU sec, about $3 \cdot 10^{91}$ years.
- For comparison: the age of the universe: about $1.5 \cdot 10^{10}$ years.
- The number of atoms in the known universe: $\leq 10^{80}$.
- If every atom in the known universe is a supercomputer and starts at the beginning of the big bang, we have only done $\frac{1.5 \cdot 10^{10} \times 10^{80}}{3 \cdot 10^{91}} = 5\%$ of the needed computations!
- Moore's law: CPU speed doubles every 18 months. If the run time function is $T(n) = 2^n$. Then, instead of solving the problem of size $n =$ say 100, we can solve the problem of size 101.

How long does it take?

- Suppose we want to factor a number X with 220 digits.
- Since X is 220 digits long, the value of X is $\geq 10^{219}$.
- So we need to perform $\frac{1}{2}\sqrt{X} > 10^{108}$ divisions.
- Say we use a super computer with speed of 10^9 divisions/sec.
- This translates into: 10^{99} CPU sec, about $3 \cdot 10^{91}$ years.
- For comparison: the age of the universe: about $1.5 \cdot 10^{10}$ years.
- The number of atoms in the known universe: $\leq 10^{80}$.
- If every atom in the known universe is a supercomputer and starts at the beginning of the big bang, we have only done $\frac{1.5 \cdot 10^{10} \times 10^{80}}{3 \cdot 10^{91}} = 5\%$ of the needed computations!
- Moore's law: CPU speed doubles every 18 months. If the run time function is $T(n) = 2^n$. Then, instead of solving the problem of size $n =$ say 100, we can solve the problem of size 101.
- An exponential time algorithm **cannot** be used to solve problems of realistic input size, no matter how powerful the computers are!

Outline

Encryption

- A customer (Alice) wants to send a message M to her bank (Bob).
- If an intruder (Evil) intercepts M, we must make sure Evil cannot understand it.

Encryption

- A customer (Alice) wants to send a message M to her bank (Bob).
- If an intruder (Evil) intercepts M, we must make sure Evil cannot understand it.
- So M must be encrypted:
  - Alice computes an encrypted message $C = P_A(M)$ ($P_A()$ is the encryption function), and send C to Bob.

Encryption

- A customer (Alice) wants to send a message M to her bank (Bob).
- If an intruder (Evil) intercepts M, we must make sure Evil cannot understand it.
- So M must be encrypted:
  - Alice computes an encrypted message $C = P_A(M)$ ($P_A()$ is the encryption function), and send C to Bob.
  - Bob receives C, and computes $M = S_A(C)$ ($S_A()$ is the decryption function), to retrieve the original M

Encryption

- A customer (Alice) wants to send a message M to her bank (Bob).
- If an intruder (Evil) intercepts M, we must make sure Evil cannot understand it.
- So M must be encrypted:
  - Alice computes an encrypted message $C = P_A(M)$ ($P_A()$ is the encryption function), and send C to Bob.
  - Bob receives C, and computes $M = S_A(C)$ ($S_A()$ is the decryption function), to retrieve the original M
  - Even if Evil sees C, he doesn't know $S_A()$, so cannot recover M.

- 1-1 Encryption:

- 1-1 Encryption:
  - Alice and Bob agree a particular method (secret key) for encryption.

- 1-1 Encryption:
  - Alice and Bob agree a particular method (secret key) for encryption.
  - Only Alice and Bob know this particular secret key, and keep it secret.

- 1-1 Encryption:
  - Alice and Bob agree a particular method (secret key) for encryption.
  - Only Alice and Bob know this particular secret key, and keep it secret.
  - For another customer (Dave), Bob and Dave must use a different key.

- 1-1 Encryption:
  - Alice and Bob agree a particular method (secret key) for encryption.
  - Only Alice and Bob know this particular secret key, and keep it secret.
  - For another customer (Dave), Bob and Dave must use a different key.
- There are many different ways for 1-1 Encryption. It is not hard.

- 1-1 Encryption:
  - Alice and Bob agree a particular method (secret key) for encryption.
  - Only Alice and Bob know this particular secret key, and keep it secret.
  - For another customer (Dave), Bob and Dave must use a different key.
- There are many different ways for 1-1 Encryption. It is not hard.
- However, Bob is dealing with many customers, and Alice is dealing with many banks, on-line accounts ...
- It would be a nightmare if we have to arrange a different key for each (Alice, Bob) pair.

RSA Public-Key Cryptosystem

- Invented by Rivest, Shamir and Aldeman in 1977. Most current computer security systems are based on this.

RSA Public-Key Cryptosystem

- Invented by Rivest, Shamir and Aldeman in 1977. Most current computer security systems are based on this.
- Everyone uses the same public key for encryption.

RSA Public-Key Cryptosystem

- Invented by Rivest, Shamir and Aldeman in 1977. Most current computer security systems are based on this.
- Everyone uses the same public key for encryption.
- Bob: chose a pair of large prime numbers p and q, say 128 digits each.

RSA Public-Key Cryptosystem

- Invented by Rivest, Shamir and Aldeman in 1977. Most current computer security systems are based on this.
- Everyone uses the same public key for encryption.
- Bob: chose a pair of large prime numbers p and q, say 128 digits each.
- Bob: compute $n = p \cdot q$.
- Bob: computes two numbers d and e, such that $d \cdot e \equiv 1 \pmod{[(p-1)\cdot(q-1)]}$. (We will discuss how to do this later. For now, it's enough to know we can calculate d, e easily from p and q. But if we only know n, it is nearly impossible to calculate e and d.)

RSA Public-Key Cryptosystem

- Invented by Rivest, Shamir and Aldeman in 1977. Most current computer security systems are based on this.
- Everyone uses the same public key for encryption.
- Bob: chose a pair of large prime numbers p and q, say 128 digits each.
- Bob: compute $n = p \cdot q$.
- Bob: computes two numbers d and e, such that $d \cdot e \equiv 1 \pmod{[(p-1)\cdot(q-1)]}$. (We will discuss how to do this later. For now, it's enough to know we can calculate d, e easily from p and q. But if we only know n, it is nearly impossible to calculate e and d.)
- The pair (n, e) is the public key. Bob makes it public.

RSA Public-Key Cryptosystem

- Invented by Rivest, Shamir and Aldeman in 1977. Most current computer security systems are based on this.
- Everyone uses the same public key for encryption.
- Bob: chose a pair of large prime numbers p and q, say 128 digits each.
- Bob: compute $n = p \cdot q$.
- Bob: computes two numbers d and e, such that $d \cdot e \equiv 1 \pmod{[(p-1)\cdot(q-1)]}$. (We will discuss how to do this later. For now, it's enough to know we can calculate d, e easily from p and q. But if we only know n, it is nearly impossible to calculate e and d.)
- The pair (n, e) is the public key. Bob makes it public.
- (n, d) is the secret key. Only Bob knows it.

Example

$p = 7$, $q = 29$. Then $n = 7 \cdot 29 = 203$, and $(p-1)\cdot(q-1) = 168$.
Pick $e = 11$ and $d = 107$, then $11 \cdot 107 = 1177 \equiv 1 \pmod{168}$.
Thus $(203, 11)$ is the public key. $(203, 107)$ is the secret key.

RSA Public-Key Cryptosystem

- Alice (and Dave and everyone else): Get public key (n, e) $(= (203, 11)$ in our example).

RSA Public-Key Cryptosystem

- Alice (and Dave and everyone else): Get public key (n, e) $(= (203, 11)$ in our example).
- Treat her message M as an integer. (It can be just the value of the binary string representing M. For example $M = 100$).

RSA Public-Key Cryptosystem

- Alice (and Dave and everyone else): Get public key (n, e) $(= (203, 11)$ in our example).
- Treat her message M as an integer. (It can be just the value of the binary string representing M. For example $M = 100$).
- Compute the encrypted message $C = P_A(M) \stackrel{\text{def}}{=} M^e \pmod{n}$. (In our example $C = 100^{11} \pmod{203} = 4$).

RSA Public-Key Cryptosystem

- Alice (and Dave and everyone else): Get public key (n, e) $(= (203, 11)$ in our example).
- Treat her message M as an integer. (It can be just the value of the binary string representing M. For example $M = 100$).
- Compute the encrypted message $C = P_A(M) \stackrel{\text{def}}{=} M^e \pmod{n}$. (In our example $C = 100^{11} \pmod{203} = 4$).
- Send $C(= 4)$ to Bob.

RSA Public-Key Cryptosystem

- Alice (and Dave and everyone else): Get public key (n, e) $(= (203, 11)$ in our example).
- Treat her message M as an integer. (It can be just the value of the binary string representing M. For example $M = 100$).
- Compute the encrypted message $C = P_A(M) \stackrel{\text{def}}{=} M^e \pmod{n}$. (In our example $C = 100^{11} \pmod{203} = 4$).
- Send $C(= 4)$ to Bob.
- Bob: Receiving $C(= 4)$. Recover the original message $M = S_A(C) \stackrel{\text{def}}{=} C^d \pmod{n}$. (In our example $4^{107} \pmod{203} = 100$).

RSA Public-Key Cryptosystem

- Alice (and Dave and everyone else): Get public key (n, e) $(= (203, 11)$ in our example).
- Treat her message M as an integer. (It can be just the value of the binary string representing M. For example $M = 100$).
- Compute the encrypted message $C = P_A(M) \stackrel{\text{def}}{=} M^e \pmod{n}$. (In our example $C = 100^{11} \pmod{203} = 4$).
- Send $C(= 4)$ to Bob.
- Bob: Receiving $C(= 4)$. Recover the original message $M = S_A(C) \stackrel{\text{def}}{=} C^d \pmod{n}$. (In our example $4^{107} \pmod{203} = 100$).
- Because of the the choice of e, d, the number theory ensures the result M is the same as the original message M. (Namely $(M^e)^d = M \pmod{n}$ for all M.)

RSA Public-Key Cryptosystem

- If Evil intercepts C, he doesn't know the secret key d, so he cannot recover $M = C^d \pmod{n}$.

RSA Public-Key Cryptosystem

- If Evil intercepts C, he doesn't know the secret key d, so he cannot recover $M = C^d \pmod{n}$.
- But Evil knows n (since this is public).

RSA Public-Key Cryptosystem

- If Evil intercepts C, he doesn't know the secret key d, so he cannot recover $M = C^d \pmod{n}$.
- But Evil knows n (since this is public).
- If Evil can factor $n = p \cdot q$, he can calculate d. Then he knows every thing that Bob knows.

RSA Public-Key Cryptosystem

- If Evil intercepts C, he doesn't know the secret key d, so he cannot recover $M = C^d \pmod{n}$.
- But Evil knows n (since this is public).
- If Evil can factor $n = p \cdot q$, he can calculate d. Then he knows every thing that Bob knows.
- But he must factor a 256 digit number n. This will need much much much .... longer time than the previous problem of factoring a number of 220-digits!

RSA Public-Key Cryptosystem

- RSA received 2002 Turing Award (the Nobel prize equivalent in CS) for this (and related) work.

RSA Public-Key Cryptosystem

- RSA received 2002 Turing Award (the Nobel prize equivalent in CS) for this (and related) work.
- This system works because the strong (but not proven) belief: The Factoring (P1) problem cannot be solved in poly-time.

RSA Public-Key Cryptosystem

- RSA received 2002 Turing Award (the Nobel prize equivalent in CS) for this (and related) work.
- This system works because the strong (but not proven) belief: The Factoring (P1) problem cannot be solved in poly-time.
- For long time, it is not known if the problem P2 (Primality Testing) can be solved in poly-time.

RSA Public-Key Cryptosystem

- RSA received 2002 Turing Award (the Nobel prize equivalent in CS) for this (and related) work.
- This system works because the strong (but not proven) belief: The Factoring (P1) problem cannot be solved in poly-time.
- For long time, it is not known if the problem P2 (Primality Testing) can be solved in poly-time.
- In 2001, Agrawal, Kayal and Saxena found a poly-time algorithm for solving P2.

RSA Public-Key Cryptosystem

- RSA received 2002 Turing Award (the Nobel prize equivalent in CS) for this (and related) work.
- This system works because the strong (but not proven) belief: The Factoring (P1) problem cannot be solved in poly-time.
- For long time, it is not known if the problem P2 (Primality Testing) can be solved in poly-time.
- In 2001, Agrawal, Kayal and Saxena found a poly-time algorithm for solving P2.
- Had they found a poly-time algorithm for solving P1 (Factoring), RSA system (and the entire computer security industry) would have collapsed overnight!

Number Theoretic Foundation of RSA Encryption

Fact

Consider the set R of real numbers. For any $a \in R$, the equation

$$a + x = 0$$

has a unique solution $-a \in R$, which is called the additive inverse of a.

Fact

Consider the set R of real numbers. For any $a \in R, a \neq 0$, the equation

$$a \cdot x = 1$$

has a unique solution $a^{-1} \in R$, which is called the multiplicative inverse of a.

Fact:

If we replace R by Q, the set of rational numbers, the above facts are still true. Namely, every $a \in Q$ has an additive inverse $-a \in Q$ and every $a \in Q, a \neq 0$ has a multiplicative inverse $a^{-1} \in Q$.

Number Theoretic Foundation of RSA Encryption

Caution:

Consider Z, the set of integers. Every $a \in Z$ still has an additive inverse $-a \in Z$. However for $a \in Q, a \neq 0$, a **has NO multiplicative inverse in** Z: $a^{-1} \notin Z$ (unless $a = 1$ or $a = -1$).

Definition

Let n be a positive integer. $Z_n = \{0, 1, 2, \ldots, n-1\}$. For elements in $a, b \in Z_n$, define:

$$\begin{aligned} a + b &= (a + b) \mod n \\ a \cdot b &= (a \cdot b) \mod n \end{aligned}$$

Number Theoretic Foundation of RSA Encryption

- For any $a \in Z_n$, there is an additive inverse in Z_n.
  Example: $n = 14$, and $a = 5$. Then $-4 = 10 \bmod 14$.
  Thus -4 = 10 in Z_{14}.
- On the other hand, $a \in Z_n, a \neq 0$ may not have a multiplicative inverse:
- For example:

$$4 \cdot x = 1 \bmod 14$$

  has no solution in Z_{14}.

Number Theoretic Foundation of RSA Encryption

Definition $Z_n^* = \{i \mid 1 \leq i \leq n-1 \text{ and } gcd(i,n) = 1\}$

Example: $Z_{14}^* = \{1, 3, 5, 9, 11, 13\}$

Fact:

For any $a, b \in Z_n^*$, $a \cdot b \in Z_n^*$.

Example: $n = 14$:

- $3 \cdot 5 = 1 \bmod 14$, and $1 \in Z_{14}^*$.
- $3 \cdot 11 = 5 \bmod 14$, and $5 \in Z_{14}^*$.
- $5 \cdot 11 = 13 \bmod 14$, and $13 \in Z_{14}^*$.

Number Theoretic Foundation of RSA Encryption

Fact:

Every $a \in Z_n^*, a \neq 0$ has a unique multiplicative inverse a^{-1} in Z_n^*. In other words, the equation

$$a \cdot x = 1 \bmod n$$

has an unique solution in Z_n^*.

Example: $n = 14$

- Since $3 \cdot 5 = 1 \bmod 14$, so $3^{-1} = 5$ in Z_{14}^*.
- Since $9 \cdot 11 = 1 \bmod 14$, so $9^{-1} = 11$ in Z_{14}^*.

Euler Totient Function

Definition: Euler Totient Function.

We define $\phi(n) = |Z_n^*|$.

Namely $\phi(n)$ is the number of integers in $\{1, \ldots, n-1\}$ that are coprime to n.

Fact:

- If p is a prime, then $\phi(p) = p - 1$.
- If $n = a \cdot b$ and $gcd(a, b) = 1$, then $\phi(n) = \phi(a) \cdot \phi(b)$.

Example:

- $Z_7^* = \{1, 2, 3, 4, 5, 6\}$. So $\phi(7) = 7 - 1 = 6$.
- $\phi(14) = \phi(7) \cdot \phi(2) = (7 - 1) \cdot (2 - 1) = 6$.

(Fermat-)Euler Theorem

(Fermat-)Euler Theorem:

For any n and all $a \in Z_n^*$,

$$a^{\phi(n)} \equiv 1 \bmod n$$

Example: $n = 14$, $\phi(14) = 6$, and $a = 3$

$3^2 \equiv 9 \bmod 14$;
$3^3 \equiv 27 \equiv 13 \bmod 14$;
$3^4 \equiv 13 \cdot 3 \equiv 39 \equiv 11 \bmod 14$;
$3^5 \equiv 11 \cdot 3 \equiv 33 \equiv 5 \bmod 14$;
$3^6 \equiv 5 \cdot 3 \equiv 15 \equiv 1 \bmod 14$;

Number Theoretic Foundation of RSA Encryption

- Bob picks two prime numbers p, q and calculate $n = p \cdot q$.
- Since p and q are primes, $gcd(p, q) = 1$.
- So we have $\phi(n) = \phi(p) \cdot \phi(q) = (p-1) \cdot (q-1)$.
- Bob pick any integer e so that $gcd(e, \phi(n)) = 1$.
- This means that there exists integers d and y such that

$$d \cdot e + y \cdot \phi(n) = 1$$

d and y can be calculated by using the extended Euclidean GCD algorithm.

- Thus

$$d \cdot e \equiv 1 \bmod \phi(n)$$

- Bop publishes (n, e) as the public key. And keep (n, d) as the secrete key.

Number Theoretic Foundation of RSA Encryption

Recall that RSA Encryption works like this:

- Alice wants to send a message, represented by a number M, to Bob.
- She encrypts M by calculating $C \equiv M^e \bmod n$, and send C to Bob.
- Bob receives C, and decrypt C by calculating $C^d \bmod n$.

For the RSA encryption to work, all we need to do is to show:

Theorem

For any $M \in Z_n$, we have $(M^e)^d \equiv M \bmod n$.

If $M \in Z_n^*$ we have:

$$\begin{aligned}
(M^e)^d &= M^{ed} \bmod n = M^{(ed-1)+1} \bmod n \\
&= ((M^{ed-1} \bmod n) \cdot (M \bmod n)) \bmod n \\
&= ((M^{\phi(n)} \bmod n) \cdot (M \bmod n)) \bmod n \\
&= (1 \cdot (M \bmod n)) \bmod n \\
&= (M \bmod n) \bmod n \\
&= M
\end{aligned}$$

If $M \in Z_n - Z_n^*$, we can still show $M^{ed} = M \bmod n$. (The proof is more involved.)