CSCI 104
Number Theory

CSCI 104 Teaching Team

Introduction

- Number Theory is the study of integers and division.
- **Prime numbers** come into play a lot.
- **Modular arithmetic** is a centerpiece of Number Theory.
- The primary application of Number Theory is **Cryptography**, which uses divisibility, prime numbers, and modular arithmetic in lots of creative ways.
  - Why is it safe to send your credit card information over the web to Amazon? This unit will start to lay the foundations to answer this question!

MODULAR ARITHMETIC AND CONGRUENCE CLASSES

Congruence

- If a and b are integers, and m is a positive integer, then:
  - "a is **congruent** to b **modulo m**" means (a % m) = (b %m)
  - This is written: $a \equiv b \pmod{m}$
  - Alternatively, b = a + mf, for some (possibly negative) integer **f**.
- Are 24 and 14 congruent **modulo** 6?
  - No, they have remainders of 0 and 2 respectively.
- Are 17 and 5 congruent **modulo** 6?
  - Yes, they both have a remainder of 5.

Ways of Showing Congruence

- $a \equiv b(mod\ m)$ if and only if:
  - Method 1: $\boldsymbol{a\ (mod\ m) = b\ (mod\ m)}$
    - 7 mod 4 = 3 and 15 mod 4 = 3 (i.e. $7 = 3 + 4 \cdot 1$ and $15 = 3 + 4 \cdot 3$)
  - Method 2: There exists some integer, f, such that $\boldsymbol{b = a + m \cdot f}$
    - 7 and 15 are congruent mod 4 because $15 = 7 + 4 \cdot 2$
  - Method 3: There exists some integer, f, such that $\boldsymbol{(b - a) = m \cdot f}$
    - 7 and 15 are congruent mod 4 because 15-7 is a multiple of 4 (i.e. 15-7=8=2*4)

$$4 \overline{)\ 7}\quad 1 \text{ r. } 3 \qquad 7 = 4 \bullet 1 + 3$$

$$4 \overline{)\ 15}\quad 3 \text{ r. } 3 \qquad 15 = 4 \bullet 1 + 3$$

Number Theory Proofs (Solution)

- Given $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$:
- **Prove**: $a+c \equiv b + d \pmod{m}$ [or $(b+d)\%m = ((b\%m)+(d\%m))\%m$]
  - $b = a + m \cdot f$, and $d = c + m \cdot g$

- **Prove**: $a \cdot c \equiv b \cdot d \pmod{m}$ [or $(b \bullet d)\%m = ((b\%m) \bullet (d\%m))\%m$]
  - $b = a + m \cdot f$, and $d = c + m \cdot g$

- **Also holds for subtraction $a-c \equiv b-d \pmod{m}$**
- **Does NOT hold for division $a/c \neq b/d \pmod{m}$**
  - More on this later

Number Theory Proofs (Solution)

- Given $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$:
- **Prove**: $a+c \equiv b + d \pmod{m}$
  - $b = a + m \cdot f$, and $d = c + m \cdot g$
  - $b + d = a + m \cdot f + c + m \cdot g = a + c + m \cdot (f+g)$. Proven!
- **Prove**: $a \cdot c \equiv b \cdot d \pmod{m}$
  - $b = a + m \cdot f$, and $d = c + m \cdot g$
  - $b \cdot d = a \cdot c + a \cdot m \cdot g + c \cdot m \cdot f + m^2 \cdot f \cdot g$ *[by distribution]*
  - $b \cdot d = a \cdot c + m \cdot (a \cdot g + c \cdot f + m \cdot f \cdot g)$. Proven!
- **Also holds for subtraction $a-c \equiv b-d \pmod{m}$**
- **Does NOT hold for division $a/c \neq b/d \pmod{m}$**
  - More on this later

Modular Arithmetic

- Applying these proofs, the modulo m result of a sum or product is equivalent to the sum or product of the inputs modulo m:

$$(a + b \cdot c)(mod\ m)$$
$$= (a\ mod\ m) + ((b\ mod\ m) \cdot (c\ mod\ m))$$

- Example:

$$(18327 + 2642 \cdot 7985)(mod\ 4) =$$

Or likely more easily computed as:

$$=$$

Modular Arithmetic (Solution)

- Applying these proofs, the modulo m result of a sum or product is equivalent to the sum or product of the inputs modulo m:

$$(a + b \cdot c)(mod\ m)$$
$$= (a\ mod\ m) + ((b\ mod\ m) \cdot (c\ mod\ m))$$

- Example:

$$(18327 + 2642 \cdot 7985)(mod\ 4) = 21{,}114{,}697(mod\ 4) = 1$$

- Or likely more easily computed as:

$$= (18327\ mod\ 4) + \big((2642\ mod\ m) \cdot (7985\ mod\ 4)\big)$$
$$= ((3) + (2 \cdot 1))(mod\ m)$$
$$= 1\ (mod\ m)$$

Alternate number bases

- In base-b, all digits must be a value between 0 and b-1.
  - If b > 10 (most commonly for b=16 known as hexadecimal) we typically use letters for a digit: 10 = A, 11 = B, …, 15 = F etc.
  - FACE is a number in base-16.
- To disambiguate the base, we will usually write it as a subscript:
  - $FACE_{16} = 64206_{10} = 1111101011001110_2$

Base Conversion (Base b to Decimal)

- Given the binary: 11011, we know this is:
  - $1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 27\ decimal$

- We could implement it as below
  - But how would pow( ) be implemented and consider the work it has to do (and then re-do, and then re-do)?

```
// suppose the binary number is
// given as a vector of bits
// in reverse order (i.e. the bit
// multiplied by 2^0 is at index 0)
unsigned bin2dec(vector<int> b)
{
  unsigned val = 0;
  for(int i=0; i < b.size(); i++)
  {
    val += b[i] * pow(2,i);
  }
  return val;
}
```

Base Conversion (Base b to Decimal)

- If `pow(r,n)` simply multiplied r by itself n-1 times then each call would take $\Theta(n)$ and to compute a polynomial of degree n would be $\Theta(n^2)$
  - $a_4 \cdot r^4 + a_3 \cdot r^3 + a_2 \cdot r^2 + a_1 \cdot r^1 + a_0 \cdot r^0$
- We can achieve the computation in linear time by factoring to the following form:
  - $\Big(\big(\big(\big((a_4) \cdot r + a_3\big) \cdot r + a_2\big) \cdot r + a_1\big) \cdot r + a_0\Big)$

Base Conversion (Decimal to Base b)

Example: Convert 23 to base 2:

- $23\ decimal = a_4 \cdot 2^4 + a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$ and we just need to find the coefficients a_4, a_3, …, a_0
- $= \frac{a_4 \cdot 2^4 + a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0}{2} = a_4 \cdot 2^3 + a_3 \cdot 2^2 + a_2 \cdot 2^1 + a_1 \cdot 2^0 + a_0 \cdot 2^{-1}$
- So, we see that the remainder of dividing by 2 is a_0 and then we can repeat again on the quotient to find a_1 and so on

```
// n is the value to convert, b is the base to
// convert to, res is the array to store the
// converted result in.
void convert(int n, int b, int res[]) {
    for (int k = 0; n != 0; k++) {
        res[k] = n % b;
        n /= b;
    }
}
```

Modular Exponentiation

- Suppose we want to calculate $b^n \% m$ (e.g. $52^{23} \% 7$)
  - We want to do this in Cryptography
  - If b and n are large, then calculating b^n may be impractical
- Let us find an alternate, more efficient approach
  - Let a = n in binary (if n=23, a=10111 bin)
    $n = (a_{k-1}a_{k-2} \ldots a_0)_2$, or $a_{k-1} \cdot 2^{k-1} + a_{k-2} \cdot 2^{k-2} + \cdots + a_0 \cdot 2^0$
  - $b^n = b^{a_{k-1} \cdot 2^{k-1} + a_{k-2} \cdot 2^{k-2} + \cdots + a_0 \cdot 2^0}$
  - $b^n = b^{a_{k-1} \cdot 2^{k-1}} \cdot b^{a_{k-2} \cdot 2^{k-2}} \cdot \ldots \cdot b^{a_0 \cdot 2^0}$
- If n = 23:
  - $b^{23} = b^{1 \cdot 2^4} \cdot b^{0 \cdot 2^3} \cdot b^{1 \cdot 2^2} \cdot b^{1 \cdot 2^1} \cdot b^{1 \cdot 2^0} = b^{2^4} \cdot b^{2^2} \cdot b^{2^1} \cdot b^{2^0}$
    $= b^{16} \cdot b^4 \cdot b^2 \cdot b^1$
  - Note: Any $a_k = 0$, leads to $b^0 = 1$ and can be left out of the product

Modular Exponentiation

- Let's use our example of n=23 so $b^{23} = b^{16} \cdot b^4 \cdot b^2 \cdot b^1$
- But how can we quickly calculate those powers and what if that value overflows the range we can make with an **int** or **long int**
  - Our naïve approach had us multiply by b a total of n times (i.e. b*b*b*b…)
  - Notice we can calculate our powers with **log(n)** multiplications: $b^2 = b^1 \cdot b^1$ which we can use to compute $b^4 = b^2 \cdot b^2$ which we can use to compute $b^8 = b^4 \cdot b^4$
  - We can perform mod m operation anytime to ensure the value does not overflow. So we start by letting $r_0 = b\%m$, and include that in our answer only if a_0 = 1.
  - Then calculate $b^2 \,\%\, m$ by calculating $r_1 = \big((b\%m) \cdot (b\%m)\big) = (r_0 \cdot r_0)\%m$, and include that in our answer only if a_1 = 1.
  - Then Calculate $b^4 \,\%\, m$ by calculating $r_2 = \big((b^2\%m) \cdot (b^2\%m)\big)\%m = (r_1 \cdot r_1)\%m$, and include that in our answer only if a_2 = 1.

Modular Exponentiation: Example 1

```
// suppose N is given in binary as a vector of bools
// in reverse order
int modularExponentiation(vector<bool> N, int b, int m) {
   int x = 1, r = b % m;
   for (int i = 0; i < N.size(); i++) {
     if ( N[i] == true )  x = (x * r) % m;
     r = (r * r) % m;
   }
   return x;
}
```

81*9*3 =2187 9*3 =27 =3

$$52^{23} mod\ 7 = 3^{23} mod\ 7 = (3^{16} \cdot 3^{4} \cdot 3^{2} \cdot 3^{1})\ mod\ 7$$

I	N[i]	x (after iter i)	r (after iter i)
initially		1	52%7=3
0	1 (a0 = LSB)		
1	1		
2	1		
3	0		
4	1 (a1 = MSB)		

$\equiv 3^{1}\ mod\ 7$ 3

$\equiv 3^{2}\ mod\ 7$ 9

$\equiv 3^{4}\ mod\ 7$ 81

$\equiv 3^{8}\ mod\ 7$ 6561

$\equiv 3^{16}\ mod\ 7$ 43046721

Modular Exponentiation: Example 1 (Sol)

```
// suppose N is given in binary as a vector of bools
// in reverse order
int modularExponentiation(vector<bool> N, int b, int m) {
   int x = 1, r = b % m;
   for (int i = 0; i < N.size(); i++) {
     if ( N[i] == true )  x = (x * r) % m;
     r = (r * r) % m;
   }
   return x;
}
```

81*9*3 =2187 9*3 =27 =3

$$52^{23} mod\ 7 = 3^{23} mod\ 7 = (3^{16} \cdot 3^{4} \cdot 3^{2} \cdot 3^{1})\ mod\ 7$$

I	N[i]	x (after iter i)	r (after iter i)	
initially		1	52%7=3	$\equiv 3^{1}\ mod\ 7$ 3
0	1 (a0 = LSB)	1*3=3 mod 7	3*3=2 mod 7	$\equiv 3^{2}\ mod\ 7$ 9
1	1	3*2=6 mod 7	2*2=4 mod 7	$\equiv 3^{4}\ mod\ 7$ 81
2	1	6*4=3 mod 7	4*4=2 mod 7	$\equiv 3^{8}\ mod\ 7$ 6561
3	0	3 (no change)	2*2=4 mod 7	$\equiv 3^{16}\ mod\ 7$ 43046721
4	1 (a4 = MSB)	3*4=5 mod 7	4*4=2 mod 7	

Modular Exponentiation: Example 2

```
// suppose N is given in binary as a vector of bools
// in reverse order
int modularExponentiation(vector<bool> N, int b, int m) {
   int x = 1, r = b % m;
   for (int i = 0; i < N.size(); i++) {
     if ( N[i] == true )  x = (x * r) % m;
     r = (r * r) % m;
   }
   return x;
}
```

$$117^{27} mod\ 5 = 2^{27} mod\ 5 = (2^{16} \cdot 2^{8} \cdot 2^{2} \cdot 2^{1})\ mod\ 5$$

I	N[i]	x (after iter i)	r (after iter i)	
initially		1	117%5=2	$\equiv 2^{1}\ mod\ 5$
0	1 (a0 = LSB)			$\equiv 2^{2}\ mod\ 5$
1	1			$\equiv 2^{4}\ mod\ 5$
2	0			$\equiv 2^{8}\ mod\ 5$
3	1			$\equiv 2^{16}\ mod\ 5$
4	1 (a1 = MSB)	3*1=3 mod 5		

Modular Exponentiation: Example 2 (Sol)

```
// suppose N is given in binary as a vector of bools
// in reverse order
int modularExponentiation(vector<bool> N, int b, int m) {
   int x = 1, r = b % m;
   for (int i = 0; i < N.size(); i++) {
     if ( N[i] == true )  x = (x * r) % m;
     r = (r * r) % m;
   }
   return x;
}
```

$$117^{27} mod\ 5 = 2^{27} mod\ 5 = (2^{16} \cdot 2^{8} \cdot 2^{2} \cdot 2^{1})\ mod\ 5$$

I	N[i]	x (after iter i)	r (after iter i)	
initially		1	117%5=2	$\equiv 2^{1}\ mod\ 5$
0	1 (a0 = LSB)	1*2=2 mod 5	2*2=4 mod 5	$\equiv 2^{2}\ mod\ 5$
1	1	2*4=3 mod 5	2*2=1 mod 5	$\equiv 2^{4}\ mod\ 5$
2	0	3 (no change)	1*1=1 mod 5	$\equiv 2^{8}\ mod\ 5$
3	1	3*1=3 mod 5	1*1=1 mod 5	$\equiv 2^{16}\ mod\ 5$
4	1 (a4 = MSB)	3*1=3 mod 5		

Modular Exponentiation: Alternate

- Or alternatively:

$$x^n = \begin{cases} \left(x^{\frac{n}{2}}\right)^2, & \text{if n is even} \\ x \cdot \left(x^{\frac{n-1}{2}}\right)^2, & \text{if n is odd} \end{cases}$$

$$3^{23} mod\ 7 = 3 \cdot \left(3^{11}\right)^2 mod\ 7$$
$$= 3 \cdot \left(3 \cdot (3^5)^2\right)^2 mod\ 7$$
$$= 3 \cdot \left(3 \cdot (3 \cdot (3^2)^2)^2\right)^2 mod\ 7$$
$$= 3 \cdot \left(3 \cdot (3 \cdot (2)^2)^2\right)^2 mod\ 7$$
$$= 3 \cdot \left(3 \cdot (12)^2\right)^2 mod\ 7$$
$$= 3 \cdot \left(3 \cdot (5)^2\right)^2 mod\ 7$$
$$= 3 \cdot (5)^2\ mod\ 7$$
$$= 3 \cdot 4\ mod\ 7 = 5 \text{ mod } 7$$

Modular Exponentiation: Alternate

- Or alternatively:

$$x^n = \begin{cases} \left(x^{\frac{n}{2}}\right)^2, & \text{if n is even} \\ x \cdot \left(x^{\frac{n-1}{2}}\right)^2, & \text{if n is odd} \end{cases}$$

$$\mathbf{3^{27} mod\ 7 = 3 \cdot \left(3^{13}\right)^2 mod\ 7}$$

$$=$$

Modular Exponentiation: Alternate

- Or alternatively:

$$x^n = \begin{cases} \left(x^{\frac{n}{2}}\right)^2, & \text{if n is even} \\ x \cdot \left(x^{\frac{n-1}{2}}\right)^2, & \text{if n is odd} \end{cases}$$

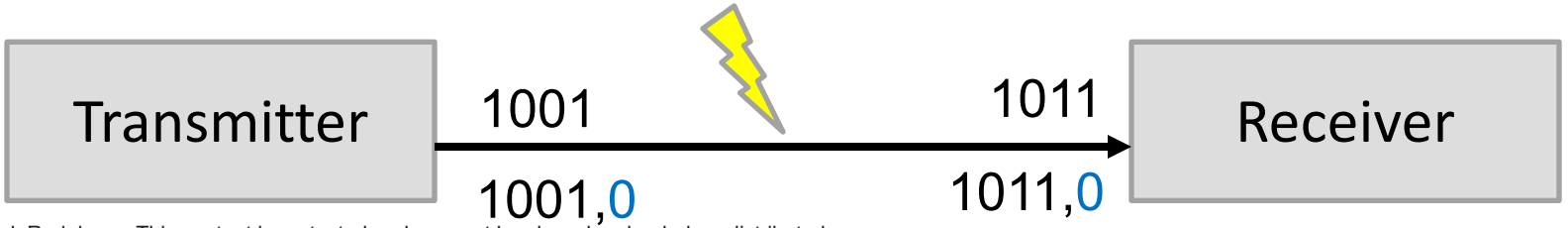
$$3^{27} mod\ 7 = 3 \cdot \left(3^{13}\right)^2 mod\ 7$$
$$= 3 \cdot \left(3 \cdot (3^6)^2\right)^2 mod\ 7$$
$$= 3 \cdot \left(3 \cdot ((3^3)^2)^2\right)^2 mod\ 7$$
$$= 3 \cdot \left(3 \cdot ((27)^2)^2\right)^2 mod\ 7$$
$$= 3 \cdot \left(3 \cdot ((6)^2)^2\right)^2 mod\ 7$$
$$= 3 \cdot \left(3 \cdot (1)^2\right)^2 mod\ 7$$
$$= 3 \cdot (3)^2\ mod\ 7$$
$$= 3 \cdot 2\ mod\ 7 = 6 \text{ mod } 7$$

XKCD #953

- If you get an 11/100 on a CS test, but you claim it should be counted as a ‘C’, they’ll probably decide you deserve the upgrade.

Applications

- When sending digital information, the probability that any given bit gets corrupted is not zero.
  - Even very small probabilities are significant, when considering the size of most transmissions: when taking a networking class, you may be surprised by how frequent errors are.
  - To counteract this, we send extra information, to help us identify when a transmission error occurred.
  - The first and simplest method was **parity** checking, which adds a single bit to the end of the message.
  - If the message contained an even number of 1s (in binary), the added bit is a 0. If it contains an odd number of 1s, the added bit is a 1.



Parity Checking

- Where previously we sent the message $x_1 x_2 \ldots x_n$, now we send $x_1 x_2 \ldots x_n x_{n+1}$, where $x_{n+1} = (x_1 + x_2 + \cdots + x_n)\%2$
- When you receive the message, you check if there are an:
  - **Odd number of 1s**: The message is corrupted, request the transmitter to send again
  - **Even number of 1s**: Accept the message as correct (is it really though?), remove the parity bit and process the message
  - If a single bit is corrupted, this system will catch it.
  - If exactly two bits are corrupted, this system will not catch it (but better systems will).
  - This system will catch an odd number of errors, which is more likely to occur than an even number of errors.

Pseudorandom Number Generators

- Your computer's random number generator is not truly random, it is instead pseudorandom.
- A common method to produce pseudorandom numbers is as follows:
  - Choose the modulus m, the multiplier a, the increment c, and the seed x_0.
  - When producing the "next" random number, generate

$$x_{n+1} = (a \cdot x_n + c)\%m$$

- When using the random library, you have the option of choosing the seed x_0.
  - We often choose the current time so different runs of the program yield different sequences.
- While the sequence of numbers looks random, if you use the same seed you will always get the same sequence.

```
// gcc rand() implementation
int rand() {
  int32_t val;
  val = ((state[0] * 1103515245) + 12345) % 2147483647;
  state[0] = val;
  return val;
}
```

```
// gcc sran() implementation
void srand(int seed)
{
  state[0] = seed;
}
```

PRIME NUMBERS

Primes

- An integer p > 1 is **prime** if its only positive factors are 1 and p.
- A positive integer > 1 that is not prime is called **composite**.
- The **Fundamental Theorem of Arithmetic** states that every integer n > 1 can be factored into a unique product of primes $(n = p_1 p_2 \ldots p_n)$, where $p_i \leq p_{i+1}$.
- Prime numbers are at the center of the field of **cryptography**.

"Divides"

- Let $a|b$ be read as "a divides b" and mean:
  - (b % a) = 0 OR
  - a is a factor of b (i.e. $b = a \cdot f$ for some integer, f)
- Examples:
  - $4 \mid 28$ (since 28 % 4 = 0 OR 28 = 4*7)
  - $4 \nmid 22$ (since 22 % 4 != 0)

The Sieve of Erastosthenes

- Used to find primes
  - List all integers from 2 to n. (let n = 31 for our example)
  - Remove all numbers that are a multiple of the first number, then the second number, then the third number, until the next number is $> \sqrt{n}$

- 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
- 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 ~~22~~ 23 ~~24~~ 25 ~~26~~ 27 ~~28~~ 29 ~~30~~ 31
- 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ ~~22~~ 23 ~~24~~ 25 ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~ 31
- 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ ~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~ 31
- 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ ~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~ 31
- Stop after crossing out multiples of 5 (since the next number 7 > sqrt(31)

Basic Divisibility Proof 1

- Prove that if $a|(b+c)$ and $a|b$ then $a|c$

Basic Divisibility Proof 1 (Sol)

- Prove that if $a|(b+c)$ and $a|b$ then $a|c$
  - Solution:
    - If $a|(b+c)$ then $\mathrm{f} \cdot a = (b+c)$ for some f and
    - If $a|b$ then $g \cdot a = b$ for some g
    - Since $\mathrm{c} = (b+c) - b = \mathrm{f} \cdot a - g \cdot a = a \cdot (f-g)$

Basic Divisibility Proof 2

- Prove that for a prime number, p, if $p|(b \cdot c)$ then $p|b$ or $p|c$

Basic Divisibility Proof 2 (Sol)

- Prove that for a prime number, p, if $p|(b \cdot c)$ then $p|b$ or $p|c$
  - Solution:
    - By the fundamental theorem of arithmetic, both b and c must be the product of some primes: $b = p_{b1}^{be1} \cdot \cdots \cdot p_{bk}^{bek}$ and $c = p_{c1}^{ce1} \cdot \cdots \cdot p_{ck}^{cek}$
    - So $b \cdot c = p_{b1}^{be1} \cdot \cdots \cdot p_{bk}^{bek} \cdot p_{c1}^{ce1} \cdot \cdots \cdot p_{ck}^{cek}$
    - Since $p|(b \cdot c)$ and p is prime, p must appear in $p_{b1}^{be1} \ldots p_{bk}^{bek} \cdot p_{c1}^{ce1} \ldots p_{ck}^{cek}$ and thus must divide either b or c

Modulo by Primes

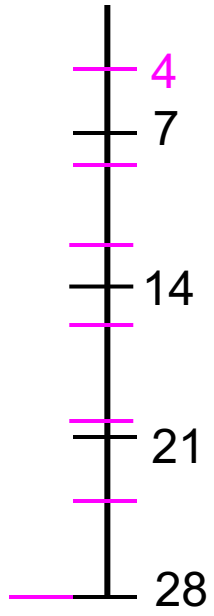
- Theorem: If p is prime and $0 < a < p$, then:
  $[0 \cdot a], [1 \cdot a], [2 \cdot a], \ldots, [(p-1) \cdot a]$ **are all distinct (i.e. a permutation of 0,1,…,(p-1))**
  - Where the notation [x] mean x mod p
- Proof by contradiction:
  - Suppose this was not true and for some distinct integers i and j, (i.e. $0 < i < j < p$), $i \cdot a \equiv j \cdot a \ (mod\ p)$ [i.e. $i \cdot a$ and $j \cdot a$ yield the same remainder mod p]
  - Then it must be that $j \cdot a - i \cdot a = a \cdot (j - i) = f \cdot p$ (i.e. the difference $j \cdot a - i \cdot a$ is a multiple of p…from def. of modular congruence)
  - Then p must divide $a \cdot (j - i)$ which means p must divide a OR p must divide$(j - i)$ [since if $p \mid b \cdot c$ then $p \mid b$ or $p \mid c$]
  - p cannot divide a since $0 \leq a < p$
  - But p cannot divide$(j - i)$ since $0 \leq (j - i) < p$
  - Contradiction!

Double Hashing

- Assume
  - m=13,
  - h1(k) = k % 13
  - h2(k) = 5 – (k % 5)
- What sequence would I probe if k = 31
  - h1(31) = ___, h2(31) = _____
  - Seq: _____
  - Notice we _____ in the table. Why? **A _____ table size!**

Cicadas (suh-kay-duh)

- **Corollary**: If p is prime and $0 < a < p$, then: $\boldsymbol{p \cdot a}$ is the first common multiple
- Cicadas are insects that make use of prime numbers.
  - They spend most of their lives as grubs, but emerge from their burrows after a prime number of years (7, 13, or 17 are common, depending on the type).
- Some predators cycle through many hunting areas, with a cycle length of > 1 year.
- If cicadas emerged every 12 years, then any predator that appears every 2, 3, 4, 6, or 12 years could evolve to prey on them.
- The prime number intervals make it difficult for predators to specialize in preying on cicadas.
  - Therefore, the predator that comes every a years can snack on cicadas only every pth circuit,
  - And the cicadas that emerge every pth year will get munched on only every ath emergence



Suppose cicadas emerged every 7th year and a predator visited every 4th year

GREATEST COMMON DIVISOR AND EUCLID'S ALGORITHM

Greatest Common Divisors

- You are given two integers a,b that are not both zero. The largest integer d that divides evenly into both a and b is the greatest common divisor of a and b, denoted $\gcd(a, b)$.
- If $\gcd(a, b) = 1$, we say that a and b are relatively prime.

- What is $\gcd(24,36)$?
  - 12
- The greatest common divisor can be calculated by the Euclidean Algorithm.

GCD and LCM

- What is gcd(24,36)?
  - $24 = 2 \cdot 2 \cdot 2 \cdot 3$ and $36 = 2 \cdot 2 \cdot 3 \cdot 3$
- So gcd(24,36) is $12 = 2 \cdot 2 \cdot 3$
- In general, given 2 numbers
  $a = p_1^{j1} \cdot p_2^{j2} \cdot \cdots \cdot p_n^{jn}$ and $b = p_1^{k1} \cdot p_2^{k2} \cdot \cdots \cdot p_n^{kn}$
  - Note: ji and ki may be 0
- Then, $\gcd(a,b) = p_1^{\min(j1,k1)} \cdot p_2^{\min(j2,k2)} \cdot \cdots \cdot p_n^{\min(jn,kn)}$
  - $\gcd(24,36) = 2^{\min(3,2)} \cdot 3^{\min(1,2)} = 2^2 \cdot 3^1 = 12$
- What is the LCM(24,36)?
  - 72
- In general, given 2 numbers a and b with prime factors as shown above, then, $\mathrm{lcm}(a,b) = p_1^{\max(j1,k1)} \cdot p_2^{\max(j2,k2)} \cdot \cdots \cdot p_n^{\max(jn,kn)}$
  - $\mathrm{lcm}(24,36) = 2^{\max(3,2)} \cdot 3^{\max(1,2)} = 2^3 \cdot 3^2 = 8 \cdot 9 = 72$

The Euclidean Algorithm

- Suppose we want to calculate $\gcd(287, 91)$.
- First divide the smaller by the larger to obtain $287 = 91 \cdot 3 + 14$.
  - If a number divides both 91 and 287, it must also divide 14
    - Since if a | (b+c) and a | b, then a | c
- So, we have reduced the problem to $\gcd(91,14)$. We repeat the above process.
- $91 = 14 \cdot 6 + 7$, so we have $\gcd(14,7)$.
- $14 = 2 \cdot 7 + 0$.
- If your remainder is 0, then the gcd is the smaller number. In this case, 7.

The Euclidean Algorithm

```
// assume: 0 <= b <= a, but a != 0
int euclid_gcd(int a, int b)
{
   if(b == 0) return a;
   else {
      int r = a % b;
      return euclid_gcd(b, r);
   }
}
```

Recursive Approach

```
// assume: 0 <= b <= a, but a != 0
int euclid_gcd(int a, int b)
{
   while (b != 0) {
     int r = (a % b);
     a = b;
     b = r;
   }
   return a;
}
```

Non-Recursive Approach

Euclid Example 1

- Calculate gcd(414, 248).

```
// assume: 0 <= b <= a,
// but a != 0
int Euclid_gcd(
    int a, int b)
{
    while (b != 0) {
      int r = (a % b);
      a = b;
      b = r;
    }
    return a;
}
```

Euclid Example 1 (Solution)

- Calculate gcd(414, 248).
  - 414 = 248 · 1 + 166, so we have gcd(248, 166).
  - 248 = 166 · 1 + 82, so we have gcd(166, 82).
  - 166 = 82 · 2 + 2, so we have gcd(82, 2).
  - 82 = 2 · 41 + 0, so 2 is the gcd.

```
// assume: 0 <= b <= a,
// but a != 0
int Euclid_gcd(
    int a, int b)
{
    while (b != 0) {
      int r = (a % b);
      a = b;
      b = r;
    }
    return a;
}
```

OTHER APPLICATIONS OF EUCLID'S ALGORITHM

Multiplicative Inverses 1

- Suppose you were asked to solve for x in the given equation:

$$8x + 5 \equiv 9 \; mod \; 11$$

- Algebraically, we would subtract 5 and divide by 8, but how does division work in modular arithmetic
- We've seen that modular arithmetic works for addition, subtraction, and multiplication
- What about division?

Multiplicative Inverses 2

- Division may lead to non-integral results, so how do we do division in modular arithmetic where we must get an integer between 0 and m-1?
  - If $b \cdot x = 1$, what is the relationship between x and b?
  - $b = x^{-1} = \frac{1}{x}$ … So, **dividing by x is the same as multiplying by b**!
  - If we work in a $(mod\ m)$ system, then we just have to find a number, b, such that $\boldsymbol{b \cdot x \equiv 1\ (mod\ m)}$, and then multiplying by b would be equivalent to dividing by x.
- A multiplicative inverse will exist for x mod m if gcd(x,m)=1
  - If m is prime, then a multiplicative inverse exists for all values of x < m
- The multiplicative inverse for 8 mod 11 is 7
  - Because $7 \cdot 8 = 56 \equiv 1\ (mod\ 11)$
  - Examples: $\left(\frac{48}{8}\right) mod\ 11 \equiv 48 \cdot 7 = 336 = 6\ (mod\ 11)$

Using Multiplicative Inverse

- To solve:

$$8x + 5 \equiv 9 \; mod \; 11$$

- First we subtract 5 from both sides

$$8x \equiv 9 - 5 \; mod \; 11 \equiv 4 \; mod \; 11$$

- Now use the multiplicative inverse of 8 (which was stated to be 7)

$$7 \cdot 8x \equiv 7 \cdot 4 \; mod \; 11$$

$$x \equiv 28 \; mod \; 11 \equiv 6 \; mod \; 11$$

- Plug back in to check:

$$8 \cdot 6 + 5 = 53 \equiv 9 \; mod \; 11$$

- But how can we find that 7 was the multiplicative inverse of 8 mod 11? Using Euclid's algorithm.
  - But to understand, let's look at another application: Diophantine equations which will lead us back to multiplicative inverses.

Diophantine Eqn Example 1

- Diophantine equations take the form: $ax + by = s$ where all unknowns are integers
- Give an ***integer*** solution to the equation: $68x + 18y = gcd(68,18)$ and give the smallest magnitude x and y if possible.
- It turns out there is always a solution to this kind of problem (where the right side of the equation is the gcd of the coefficients of the two variables)
- Start by using Euclid's algorithm but keep the **quotient** and remainder:

$$68 = 3 * 18 + 14$$
$$18 = 1 * 14 + 4$$
$$14 = 3 * 4 + 2$$
$$4 = 2 * 2 + 0$$

So gcd(18,68) = 2

Diophantine Eqn Example Workspace

$68x + 18y = \gcd(68,18)$

$$68 = 3 * 18 + 14$$
$$18 = 1 * 14 + 4$$
$$14 = 3 * 4 + 2$$
$$4 = 2 * 2 + 0$$

So gcd(18,68) = 2

Diophantine Eqn Example (Sol)

- Use that work and repeated substitution to find the solutions to the original equation: 68x + 18y = gcd(68,18)
- To solve 68x + 18y = gcd(68,18), start with a=r0=68 and b=r1=18, (the textbook uses r0 and r1, we'll use a and b) rearrange the lines from Euclid's to solve for the remainder instead :

$68 = 3 * 18 + 14$ => $14 = 68 - 3 * 18$

$18 = 1 * 14 + 4$ => $4 = 18 - 1 * 14$

$14 = 3 * 4 + 2$ => $\gcd(68,18) = 2 = 14 - 3 * 4$

- Then put each remainder in terms of r0 and r1, substituting an earlier remainder's expression into later occurrences of that remainder until you reach an equation for the gcd in terms of a=r0=68 and b=r1=18.

$$14 = 68 - 3 * 18 = 1 \cdot a - 3 \cdot b$$

But now substitute that expression in for 14 in the next line of Euclid's

$$4 = 18 - 1 * 14 = b - 1 * (1 \cdot a - 3 \cdot b) = 4 \cdot a - 1 \cdot b$$

But now substitute the expressions for 14 and 4 into the next line of Euclid's

$$2 = 14 - 3 * 4 = (1 \cdot a - 3 \cdot b) - 3 * (4 \cdot b - 1 \cdot a) = 4 \cdot a - 15 \cdot b$$

$2 = \gcd(68,18) = 4 \cdot a - 15 \cdot b = 4 \cdot 68 - 15 \cdot 18$ (you can verify)

So for 68x + 18y = gcd(68,18) we have solved x=4, y=-15

Multiplicative Inverses

- Now we can understand how to find a multiplicative inverse if it exists.
  - Recall: A multiplicative inverse will exist for x mod m, if gcd(x,m)=1
  - If m is prime, then a multiplicative inverse exists for all values of x < m
- If a multiplicative inverse exists for values of x and m, we can find the inverse of x using the process we just practiced of solving Diaphontine equation: $\boldsymbol{a \cdot m + b \cdot x = 1 (\mathrm{mod}\ m)}$
  - We put our multiplicative inverse problem into the form of a Diophantine equation since we now know how to solve that form of problem
- Why? Because $\boldsymbol{a \cdot m + b \cdot x = 1 (\mathrm{mod}\ m) = (a \cdot m) mod\ m + (b \cdot x) mod\ m = 1 (\mathrm{mod}\ m)}$
  - $a \cdot m \equiv 0 (mod\ m)$ so that implies $b \cdot x \equiv 1 (mod\ m)$ and thus b must be the multiplicative inverse

Using Euclid's Algorithm

- Find the multiplicative inverse of x=8 mod 11 (i.e. x=8, m=11)
  - Note that for any x >= m, we will find its congruence such that 0 <= x <= (m-1)
- Perform Euclid's (then rearrange solving for the remainders)

$11 = 1 \cdot 8 + 3$ => $3 = 11 - 1 \cdot 8$

$8 = 2 \cdot 3 + 2$ => $2 = 8 - 2 \cdot 3$

$3 = 1 \cdot 2 + 1$ => $1 = 3 - 1 \cdot 2$

$2 = 2 \cdot 1 + 0$

- Now substitute back in with r0=11 and r1=8

- So 11z+8y=1 mod 11 has a solution of x=__nd y=____ mod 11
- Thus __ is the multiplicative inverse of 8 mod 11

Using Euclid's Algorithm

- Find the multiplicative inverse of x=8 mod 11 (i.e. x=8, m=11)
  - Note that for any x >= m, we will find its congruence such that 0 <= x <= (m-1)
- Perform Euclid's (then rearrange solving for the remainders)

$$11 = 1 \cdot 8 + 3 \quad \Rightarrow \quad 3 = 11 - 1 \cdot 8$$
$$8 = 2 \cdot 3 + 2 \quad \Rightarrow \quad 2 = 8 - 2 \cdot 3$$
$$3 = 1 \cdot 2 + 1 \quad \Rightarrow \quad 1 = 3 - 1 \cdot 2$$
$$2 = 2 \cdot 1 + 0$$

- Now substitute back in with r0=11 and r1=8

$$3 = r0 - 1 \cdot r1$$
$$2 = r1 - 2 \cdot 3 = r1 - 2 \cdot (r0 - r1) = -2 \cdot r0 + 3 \cdot r1$$
$$1 = 3 - 1 \cdot 2 = (r0 - r1) - 1 \cdot (-2 \cdot r0 + 3 \cdot r1) = 3 \cdot r0 - 4 \cdot r1$$

- So 11z+8y=1 mod 11 has a solution of x=3 and y=-4=7 mod 11
- Thus 7 is the multiplicative inverse of 8 mod 11

OTHER INTERESTING APPLICATIONS ABOUT PRIMES

Interesting Facts about Primes

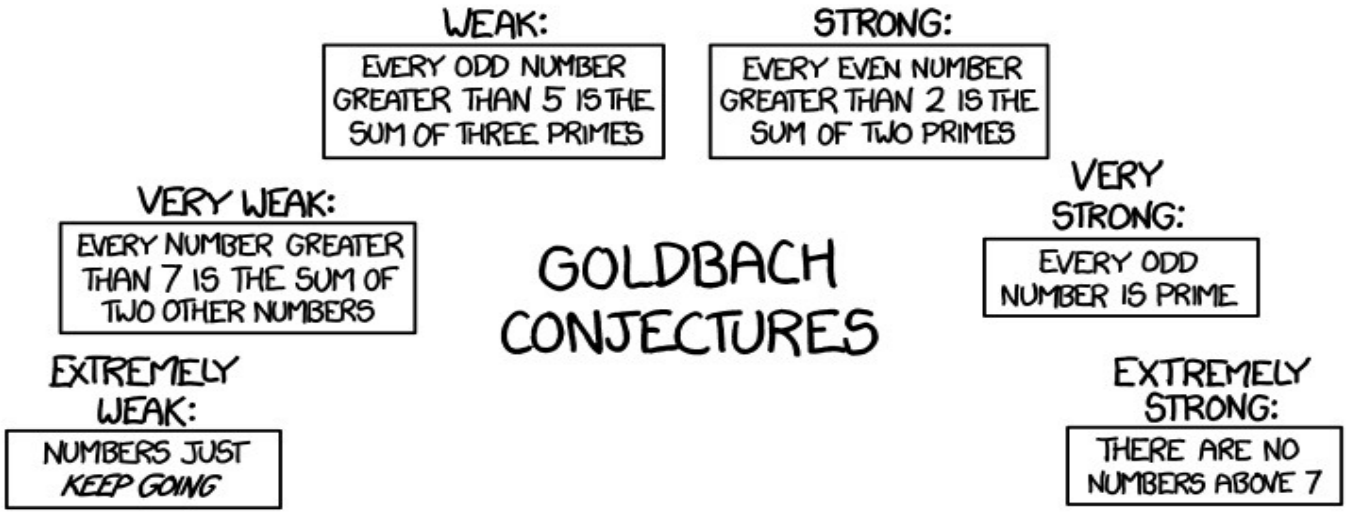
- Twin Primes are pairs of primes that differ by 2.
  - 3 and 5
  - 5 and 7
  - 11 and 13
  - 17 and 19
  - 4967 and 4969
  - 8675309 and 8675311
  - $2996863034895 \cdot 21290000 \pm 1$
- The Twin Prime Conjecture claims there are infinitely many twin primes.

Goldbach's Conjecture

- Goldbach's Conjecture says that every even integer n > 2 is the sum of two primes.
  - 4 = 2 + 2
  - 6 = 3 + 3
  - 8 = 3 + 5
  - 10 = 3 + 7 and also 5 + 5
  - 12 = 5 + 7
  - 100 = 3 + 97 or 11 + 89 or 17 + 83 or 29 + 71 or 41 + 59 or 47 + 53
- Verified by computer up to 4×10^{18}, but unproven!

XKCD #1310

- The weak twin primes conjecture states that there are infinitely many pairs of twin primes. The tautological prime conjecture states that the tautological prime conjecture is true.



APPLICATIONS TO CRYPTOGRAPHY

Private Key Cryptography

- One of the earliest known uses of cryptography was by Julius Caesar. He made messages secret by shifting each character three letters forward in the alphabet:
- $f(p) = (p + 3) \% 26$
- This way, if a message was intercepted on the battlefield, they would not know what information was being conveyed.
- This is an example of a private key cryptosystem.
- UNIVERSITY OF SC would be encrypted as XQLYHUVLWB RI VF
- Knowing the encryption key allows you to easily find the decryption key:
- $f(c) = (c - 3) \% 26$

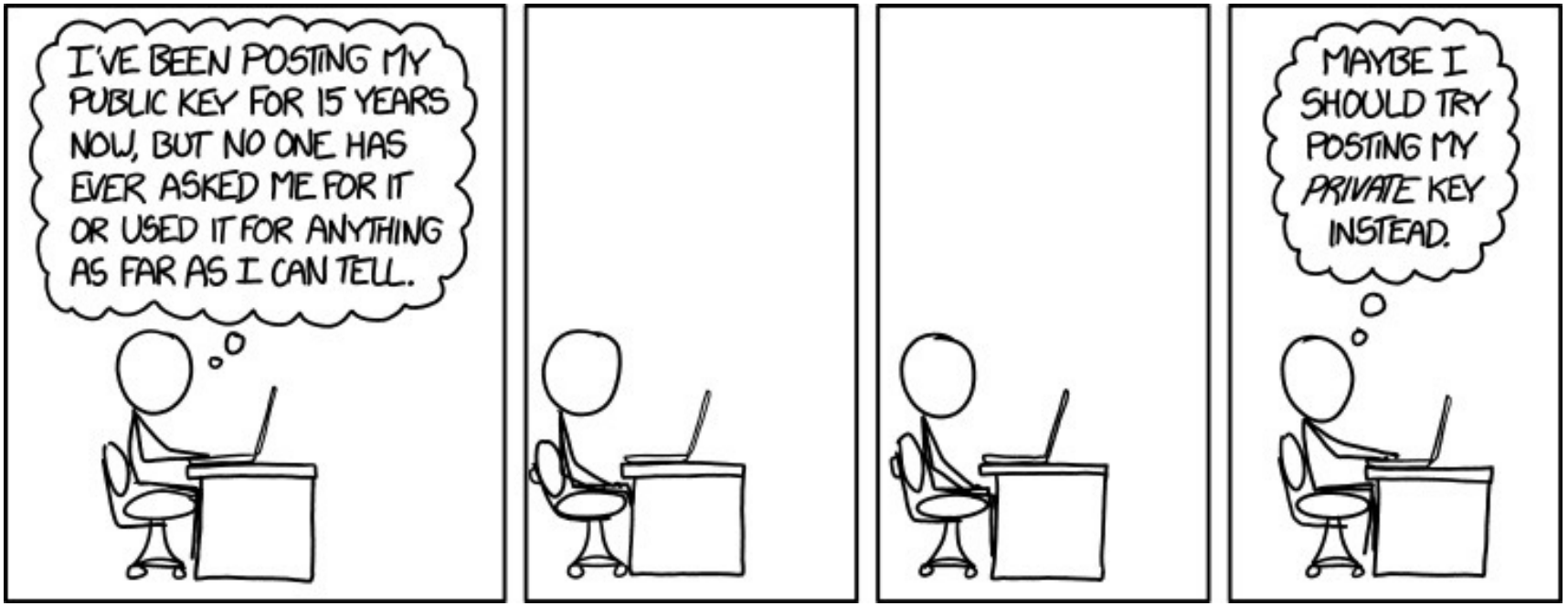
Private Key Cryptography drawbacks

- Any pair of people who want to communicate secretly must have the same key.
- This was not a problem for Caesar, as his generals met up beforehand and agreed upon the encryption key.
- Today however, we will want to communicate securely with people we have never talked to before. You shouldn't have to go to Amazon headquarters to be able to securely convey your credit card number
- If you simply send the encryption key via a digital communication, it can be intercepted. Then they will know how to decrypt your future messages!
- As you haven't agreed on an encryption key yet, it cannot be sent encrypted!

Public key cryptography

- Much more interesting is **public key cryptography**.
- In these systems, knowing the encryption key does not help you identify the decryption key.
- Therefore, you make your encryption key public and keep your decryption key private.
  - Alice and Bob want to communicate securely.
  - Alice sends her encryption key to Bob.
  - Bob uses it to encrypt his message and sends it to Alice.
  - Alice uses her decryption key to read the message.
  - Bob sends his encryption key to Alice.
  - Alice uses it to encrypt her reply and sends it to Bob.
  - Bob uses his decryption key to read the message

XKCD #1553



RSA

- The RSA public key cryptosystem was first discovered in secret government research.
- It was then discovered independently, and introduced to the world, by Rivest, Shamir, and USC's very own Leonard Adleman.
- Encryption isn't too complicated, and it will be explained thoroughly in the coming slides.
- If you want to understand RSA Decryption, you'll need to delve much deeper into Number Theory!

RSA encryption

- I will select two values, e and n, and publish them.
- These two values are specifically engineered, but we won't talk about that until we get to RSA decryption.
- If you want to send me a secure message, you will break it up into consecutive blocks of letters (how long the blocks are depends on n).
- To send one block of your message, translate it into a number M. This is nothing more than a base-conversion algorithm from ASCII / Unicode / whatever to base 10.
- Calculate $C = M^e \% n$, and send the message C.

An Example

- We want to encrypt the message STOP, using the public key (n=2537, e=13).
- Translate STOP into base-10 using a straightforward conversion: 18-19-14-15, so 18191415
- Break it up into 2 blocks, each of length 4 (a block must always have value $\leq$ n): 1819-1415
- Calculate $1819^{13}\ \%\ 2537 = 2081$ and $1415^{13} \%\ 2537 = 2182$.
- Send the message 2081-2182. You're done!
- Note that we need the Modular Exponentiation algorithm for this, as calculating 1819^{13} is not a good use of time and space (it has 43 digits!)
- We chose a relatively small n in this example, but n is usually around 400 digits, making the Modular Exponentiation algorithm essential.

RSA Decryption

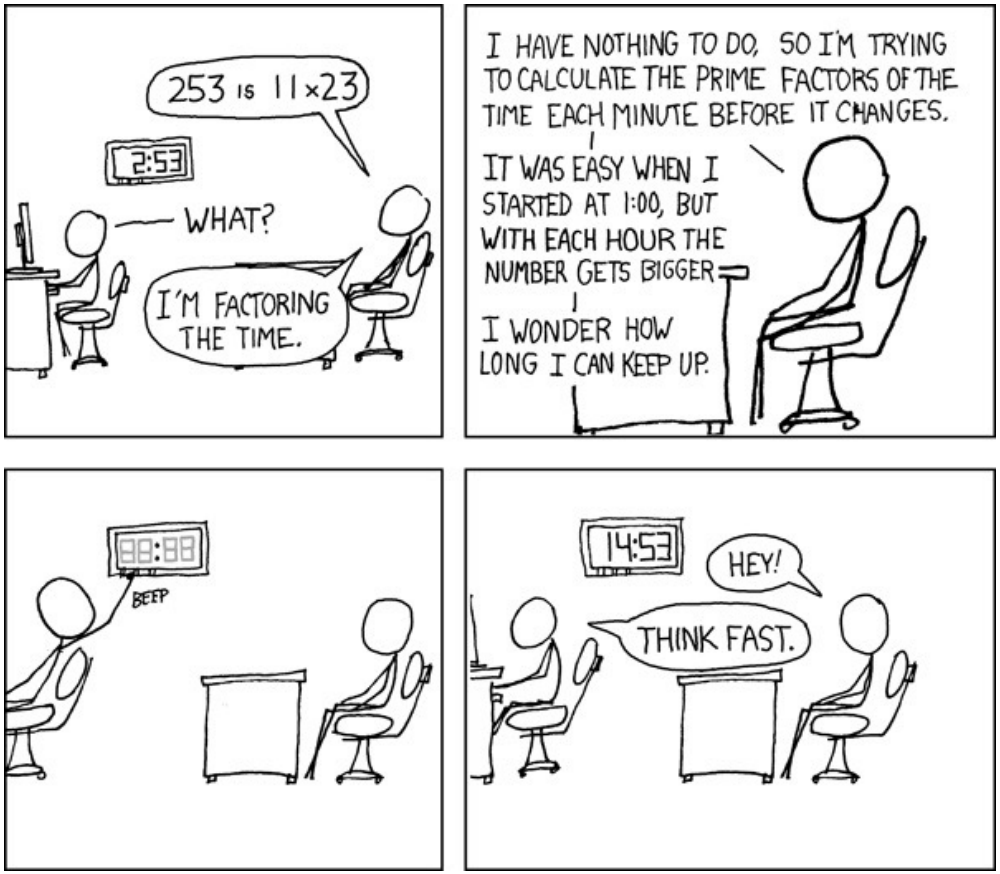
- e and n were engineered very carefully, they were not arbitrary numbers.
- $n = p \cdot q$, where p and q are prime numbers of around 200 digits.
  - In the prior example, $2537 = 43 \cdot 57$
- e is chosen to be a number that is relatively prime to $(p-1) \cdot (q-1)$. A common choice of e is $2^{16} + 1 = 65537$.
- If you know p and q, you can decrypt the message.
- The most efficient known factorization method would require billions of years to factor 400-digit integers, making RSA quite secure…
  - …We think.

P vs NP, and one-way Functions

- We believe that multiplying large primes together is a one-way function. That is, it is easy to calculate $f(a, b) = a \cdot b$, but hard to calculate $f^{-1}(a \cdot b) = \{a, b\}$.
- All modern cryptography depends on this assumption, that factoring large numbers is very hard. But we're **actually not sure**.
- If P = NP, that is, if Hamiltonian Cycle has a polynomial-time solution, then it follows that not only is factoring large numbers easy, but there is no such thing as a one-way function!
- If you prove that P = NP, you could crack all existing cryptosystems on the planet!

XKCD #247

- I occasionally do this with mile markers on the freeway.



You want more detail?

- Using e, p, and q, you can calculate decryption key d.
- Compute λ(n), which is the least common multiple of $p-1$ and $q-1$.
- Solve the equation $d \cdot e \equiv 1 \ (mod\ \lambda(n))$
- Now you can convert a block of letters C back to the original message M:
- $C^d \ \% \ n \ = \ M$

You still want more detail? You're sure?

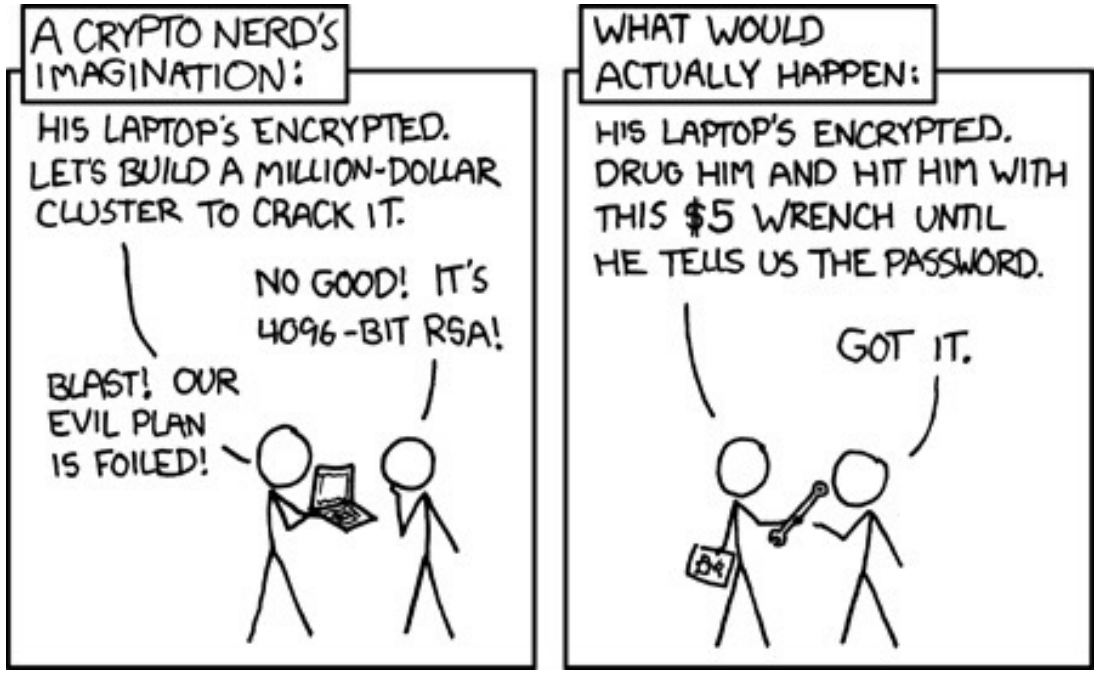
- We can calculate the least common multiple of $p-1$ and $q-1$ using the Euclidean Algorithm.
- It turns out the least common multiple of a and b is $\frac{a \cdot b}{\gcd(a,b)}$
- Since we chose e to be relatively prime to $(p-1) \cdot (q-1)$, we know it is also relatively prime with λ(n).
- It turns out that if x and y are relatively prime, then $d \cdot x \equiv 1 \ (mod \ y)$ has a unique solution for $d \ (mod \ y)$. The solution d is known as the modular multiplicative inverse.
- d can be calculated using a variant of the Euclidean algorithm.

Far Too Much Detail, Cont.

- $C^d \equiv (M^e)^d \equiv M^{d \cdot e} \equiv M^{1+k(p-1)(q-1)}$, for some k
- Since $M < p \cdot q$, either $\gcd(M, p) = 1$, or $\gcd(M, q) = 1$ (usually both).
- We can use a result called Fermat's Little Theorem to assert that, if $\gcd(M, p) = 1$, then $M^{p-1} \equiv 1 (mod\ p)$, and if $\gcd(M, q) = 1$, then $M^{q-1} \equiv 1 (mod\ q)$ (at least one is true, usually both)
- Therefore, $M^{k(p-1)(q-1)} \equiv 1\ (mod\ n)$
- Finally, $M^{1+k(p-1)(q-1)} \equiv M\ (mod\ n)$
- So, $C^d \equiv M\ (mod\ n)$.
- Easy, right?

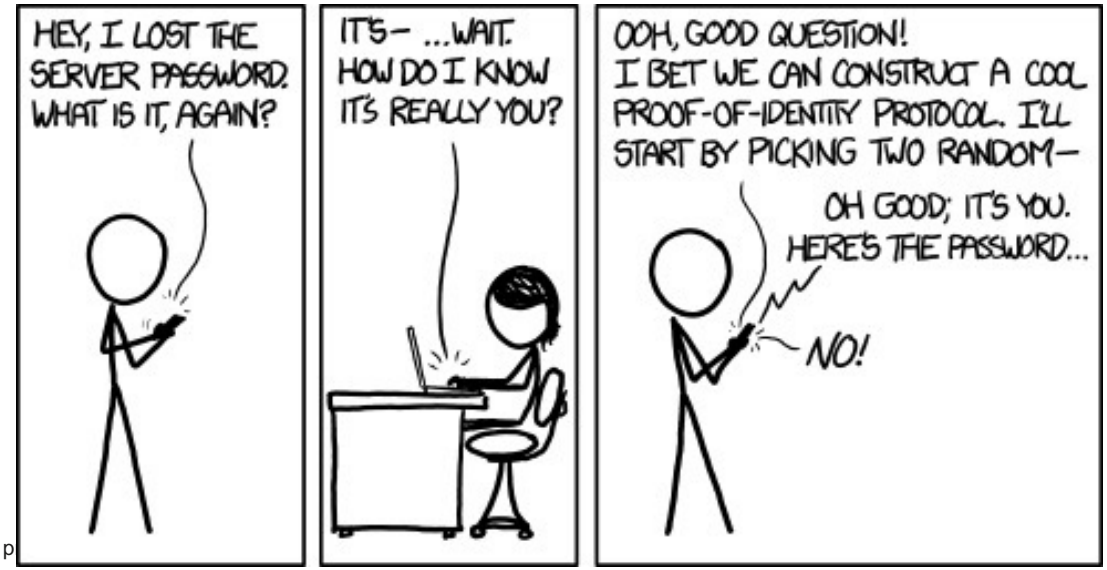
XKCD #538

- Actual actual reality: nobody cares about his secrets. (Also, I would be hard-pressed to find that wrench for $5.)



XKCD #1121

- Not sure why I just taught everyone to flawlessly impersonate me to pretty much anyone I know. Just remember to constantly bring up how cool it is that birds are dinosaurs and you'll be set.



Modern Cryptography

- The only drawback to private-key cryptosystems is that you have to agree on the private key away from prying eyes.
- Most modern systems will use a public-key system such as RSA to securely send the private key, and then they switch to the private-key system.
- Alice posts in the Daily Trojan:
  "Dear Bob, lets use n=2537 and e=13. - Alice"
- Bob uses her public key, encrypts and sends:
  "Dear Alice, let's use the Caesar cipher for future communications."
- All future communications then use the Caesar cipher, which they agreed upon privately.
- This still raises an interesting question: how is Alice sure she's communicating with Bob? How is Bob sure he is communicating with Alice?

Digital Signatures

- Suppose Alice's public RSA key is (n, e), and her private key is d.
  - She sends her message as normal, but she attaches a signature S, which has been run through her **decryption** protocol $S^d \ (mod\ n)$
  - Anyone can then run the resulting signature through her **public encryption** protocol to extract the signature. Alice has produced a message which only she could write!
- We've over-simplified it, as someone could copy-paste her signature and re-use it. This is why we have trusted 3rd party organizations that verify a specific signature hasn't already been used.

Beyond Polynomial

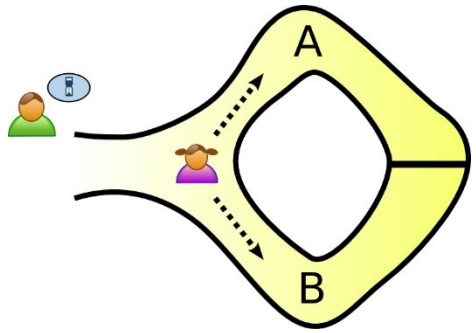
- In this class, we have referred to any runtime that is greater than polynomial as "exponential".
  - This is not actually correct, as exponential means a specific thing: $\Theta(d^{\overline{f}})$, where d > 1, and f > 0
  - Any time greater than polynomial is called **super-polynomial**
  - Any time less than exponential is called **sub-exponential**
  - The best known algorithm to factor large numbers falls between polynomial and exponential.

Zero-Knowledge Proofs

- Suppose Peggy has some information she wants to sell.
- Victor wishes to buy this information.
- Victor does not wish to hand over his money or credit card number until he is sure Peggy actually has this information and isn't trying to scam him.
- Of course, as soon as Peggy reveals the information, Victor may opt to not hand over the money.
- Peggy needs to prove to Victor she has the information, while also giving him no additional clues about what the information actually is.
- Such a proof is known as a zero-knowledge proof.

A simple zero-knowledge proof

- There is a cave which consists of two pathways that meet up at a magic door.
- The door has a magic password: speak it, and the door opens.
- Peggy wants to convince Victor that she knows the password, but she doesn't want to reveal what that password is to Victor.
- Victor can watch Peggy go down one tunnel, and wait. If Peggy then appears from the other tunnel, she has proven she has the information (and Victor has verified she has it), while simultaneously giving no additional information about what the password is to Victor.



Probability

- Most zero-knowledge proofs are probabilistic: there is a low chance that Peggy will be able to fool Victor, but most of the time the verification process works.
- We could modify the magic door proof to make it probabilistic:
  - Peggy chooses a tunnel (left or right) and walks down it, unknown to Victor.
  - Victor then approaches the split, and yells out “come back via the right tunnel!”
  - If Peggy has the password, she will be able to do so.
  - If Peggy doesn’t have the password, she’ll be able to do so only if she originally chose the right tunnel. 50% odds.
  - Victor can repeat the experiment as many times as he likes until he is satisfied.

Another example

- Peggy wants to prove that a given graph G is 3-colorable, while not revealing to Victor how to actually do it. You could have a page with the same graph mirrored on both sides.
  - Peggy can color the graph on one side, but not reveal the coloring (Victor only sees the other, uncolored side).
  - Victor chooses a single edge from the uncolored side.
  - Peggy cuts out that edge and its two nodes, flips it over, and reveals to Victor that the two nodes have different colors.
  - If Peggy didn't actually 3-color the graph, at least one edge will reveal the lie, so Victor has a $\frac{1}{m}$ chance of catching the lie.
  - Victor can repeat as many times as he likes, until he is satisfied. Peggy will keep swapping the 3 colors, so that Victor is unable to construct a larger solution.

Sudoku

- How would you convince someone that a given Sudoku problem has a solution, without spoiling the answer?

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

BACKUP

Diophantine Eqn Example (Sol)

- Use that work and repeated substitution to find the solutions to the original equation: 68x + 18y = gcd(68,18)
- To solve 68x + 18y = gcd(68,18), start with r0=68 and r1=18, rearrange the lines from Euclid's to solve for the remainder instead :

$68 = 3 * 18 + 14$ => $14 = 68 - 3 * 18$

$18 = 1 * 14 + 4$ => $4 = 18 - 1 * 14$

$14 = 3 * 4 + 2$ => $\gcd(68,18) = 2 = 14 - 3 * 4$

- Then put each remainder in terms of r0 and r1, substituting an earlier remainder's expression into later occurrences of that remainder until you reach an equation for the gcd in terms of r0=68 and r1=18.

$$14 = 68 - 3 * 18 = 1 \cdot r0 - 3 \cdot r1$$

But now substitute that expression in for 14 in the next line of Euclid's

$$4 = 18 - 1 * 14 = r1 - 1 * (1 \cdot r0 - 3 \cdot r1) = 4 \cdot r1 - 1 \cdot r0$$

But now substitute the expressions for 14 and 4 into the next line of Euclid's

$$2 = 14 - 3 * 4 = (1 \cdot r0 - 3 \cdot r1) - 3 * (4 \cdot r1 - 1 \cdot r0) = 4 \cdot r0 - 15 \cdot r1$$

$2 = \gcd(68,18) = 4 \cdot r0 - 15 \cdot r1 = 4 \cdot 68 - 15 \cdot 18$ (you can verify)

So for 68x + 18y = gcd(68,18) we have solved x=4, y=-15