SageMath 2: Number Theory and RSA Cryptosystem

Cheng-Hsin Hsu

National Tsing Hua University

Department of Computer Science

Divisibility

- a and b are integers, b divides a if there exists an integer q such that $a=qb$
  - b is a divisor of a
  - b is a factor of a
  - a is a multiple of b
- Example
  - 12 divides 144, because 144 = 12 x 12
  - Every integer divides 0
  - 0 does not divide any integer, except 0 itself

Modular Arithmetic

- Example: $$16 \div 3 = 5 \cdots 1$$

  Dividend Divisor Quotient Remainder

- For any $a, b \in \mathbb{Z}, b > 0$, there exist unique $q, r \in \mathbb{Z}$ such that $a = qb + r,\ 0 \leqslant r < b$
  - SageMath/Python: (i) $b < 0$ is possible, and $br \geq 0, |r| < |b|$
- Modulo
  - $16 \mod 3 = 1$
  - $-12 \mod 5 = 3$

Modular Arithmetic in SageMath

```
25 // 4
```

6

$$q = a \text{ div } b = \lfloor a/b \rfloor$$

Quotient

```
25 % 4
```

1

$$r = a \bmod b = a - qb$$

Modulo

```
4 * 6 + 1
```

25

$$a = qb + r$$

Validation

Integers in Base Other than 10

- Write *6137* in the octal system (base *8*). In other words, finds $r_0, r_1, \ldots, r_k$ so that $(6137)_{10}=(r_k \ldots r_2 r_1 r_0)_8$
- Write *3387* into binary (base *2*) and hexadecimal (base *16*)

		Remainders
8	6137	
8	767	$1(r_0)$
8	95	$7(r_1)$
8	11	$7(r_2)$
8	1	$3(r_3)$
	0	$1(r_4)$

		Remainders	
16	13,874,945		
16	867,184	1	(r_0)
16	54,199	0	(r_1)
16	3,387	7	(r_2)
16	211	11 (= B)	(r_3)
16	13	3	(r_4)
	0	13 (= D)	(r_5)

Convert Integers into Other Bases

```
123.digits(base=16)
```

```
[11, 7]
```

```
123.digits(base=20)
```

```
[3, 6]
```

```
123.digits(base=60)
```

```
[3, 2]
```

```
3 + 2*60
```

```
123
```

Validation

Exact Fractions: a Toy Problem

- Start from our familiar 10-based (decimal) system, for a fraction number *p/q* , we sometime can represent it exactly in two-decimal-place (or fewer)
  - 1/2 = 0.50
  - 1/3 = 0.33333333333333333333333333333
  - 1/4 = 0.25
  - 1/5 = 0.20
- Any rules?? $\frac{p}{q} = 0.01z \rightarrow z = \frac{100p}{q} \rightarrow q|100$

Divisors of 100

- Turns out that if *q | 100*, we can write *p/q* as a two-decimal-place exact decimal!
- Let’s use SageMath to find all divisors of 100

```
divisors(100)
```

```
[1, 2, 4, 5, 10, 20, 25, 50, 100]
```

- When q is in {3, 6, 7, 8, 9}: two decimals are not enough! ← What does this mean?
  - Splitting 10k TA salary among 3 students!

How About Other Bases

- Consider binary?

```
divisors(2^2)
```

```
[1, 2, 4]
```

- Base-20?

```
divisors(20^2)
```

```
[1, 2, 4, 5, 8, 10, 16, 20, 25, 40, 50, 80, 100, 200, 400]
```

- Base-60?

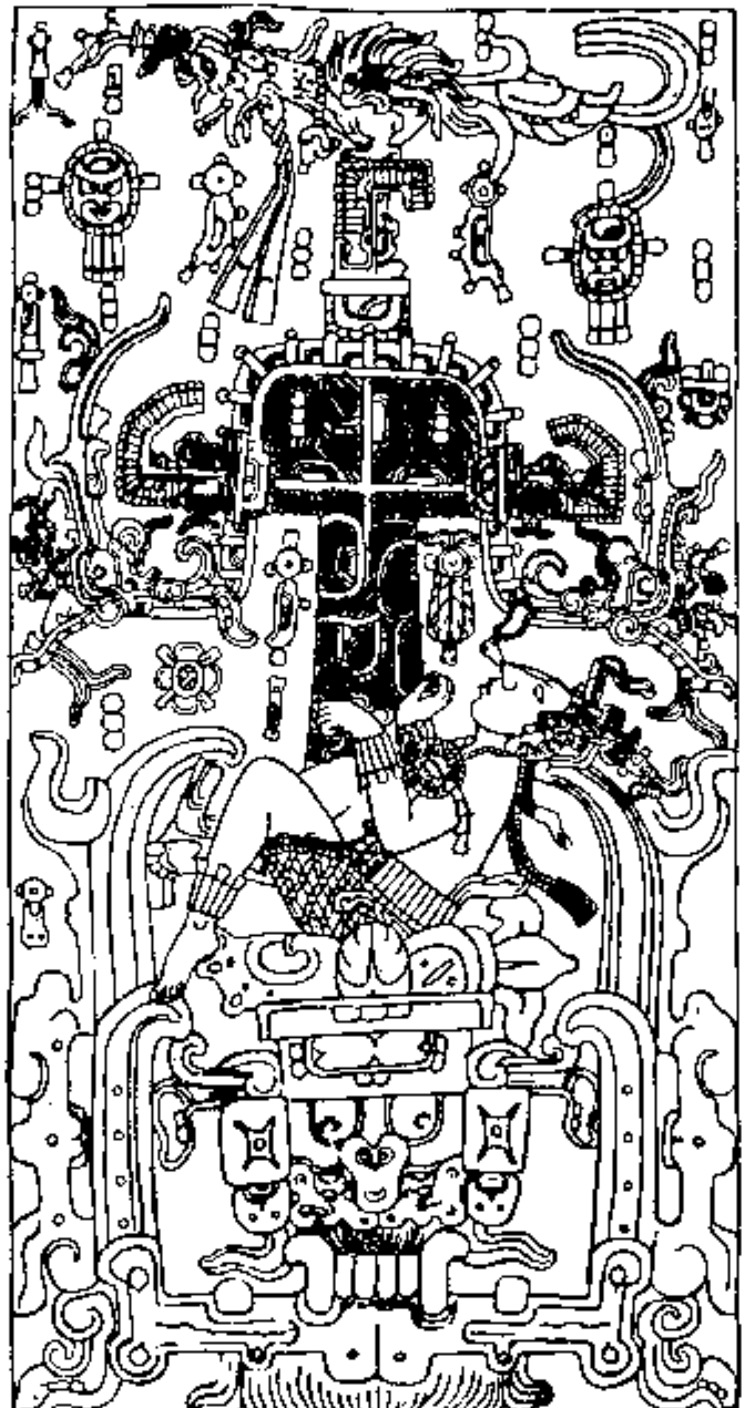
```
len(divisors(60^2))
```

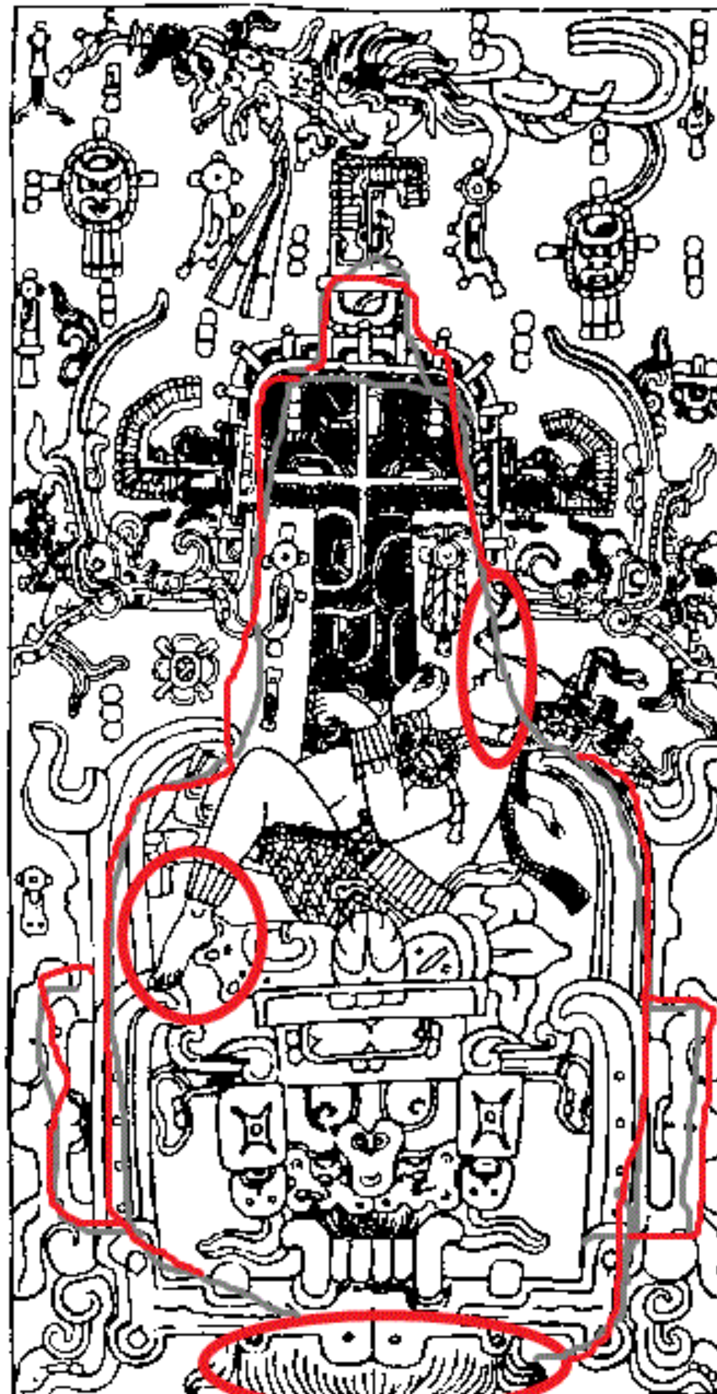
```
45
```

$\boxed{\{1, 2, 3, 4, 5, 6, 8, 9, 10,} 12, \ldots\}$

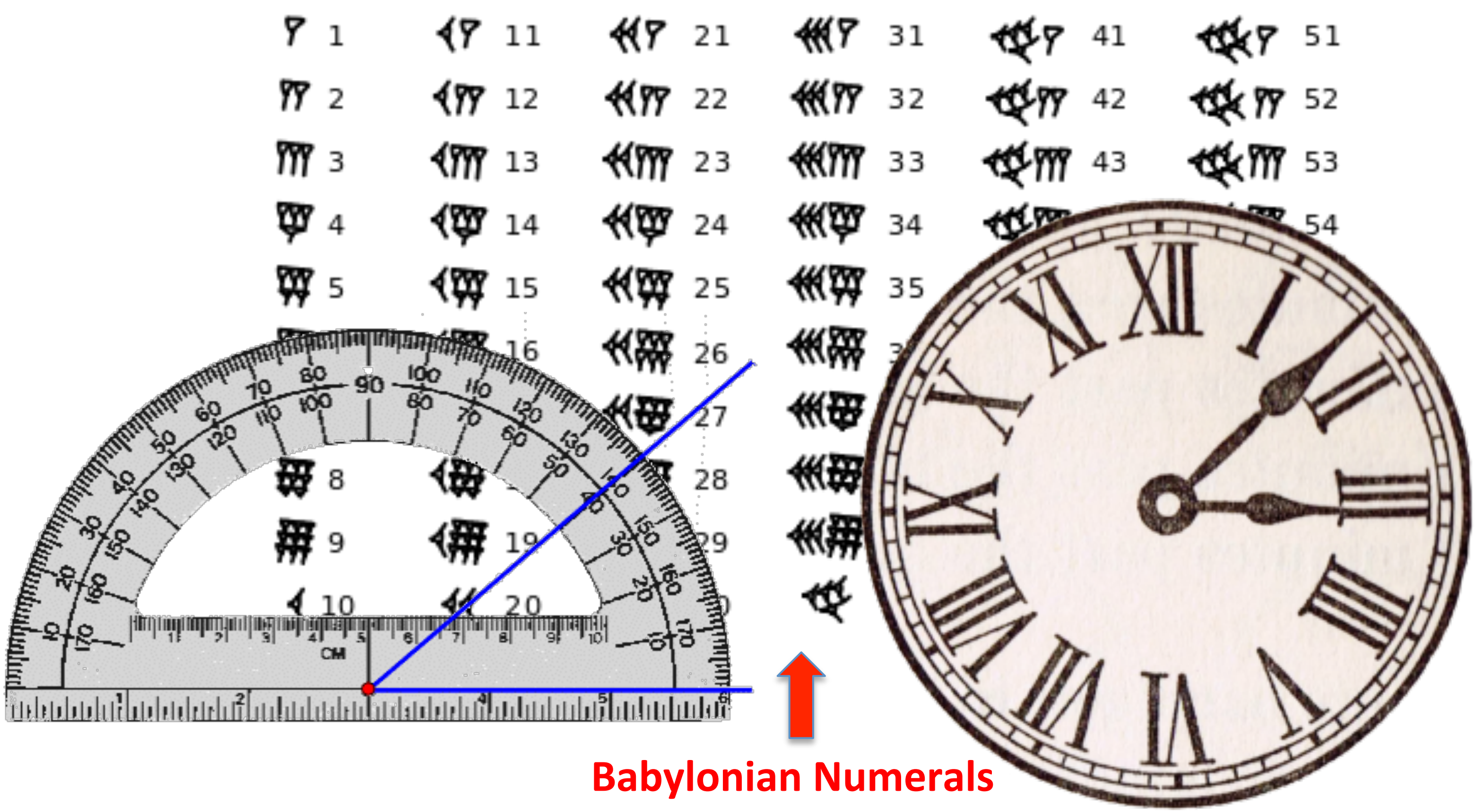
- 20 and 60 are highly composite numbers
  - Simplify counting, e.g., with base-60 system, 1/3 can be easily written!

se-





How About Base-60 System?



- We still see base-60 systems in trigonometry and time metrics

Prime and Composite

- Primes are integers ($n>1$) with exactly two positive divisors
- All other integers ($n>1$) are called composite
- If $n \in \mathbb{Z}^+$ is composite, then there is a prime p such that $p|n$
- 0 and 1 are neither prime nor composite

```
is_prime(123)
```

```
is_prime(179424673)
```

Fundamental Theorem of Arithmetic

- If $a, b \in \mathbb{Z}^+$ and *p* is a prime, then $p|ab \Rightarrow p|a \text{ or } p|b$
  - Can be generalized to *n* positive integers
- Any integer *n>1* can be written as a (unique) product of primes $$a = p_1^{t1} p_2^{t2} \cdots p_k^{t_k} = \prod_{i=1}^{k} p_i^{t_i}$$
  - Factorization: canonical representation
- Exercise: What is the prime factorization of 980?
- Prove that *17*|*n* given

$$10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot n = 21 \cdot 20 \cdot 19 \cdot 18 \cdot 17 \cdot 16 \cdot 15 \cdot 14$$

Prime Related Fun Functions

```
next_prime(123)
```

127

```
next_prime(10^100)
```

100000000000000000000000000000000000000000000000000000000000000000000000\
00000000000000000000000000267

```
nth_prime(1)
```

evaluate

2

```
nth_prime(999)
```

7907

Prime Related Fun Functions (cont.)

```
prime_range(1,10)
```

```
[2, 3, 5, 7]
```

```
prime_range(1050, 1100)
```

```
[1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097]
```

```
factor(2015)
```

```
5 * 13 * 31
```

```
factor(-9999)
```

```
-1 * 3^2 * 11 * 101
```

Common Divisors

- For $a, b \in \mathbb{Z}$, $c > 0$ is a common divisor of *a* and *b* if $c|a \text{ and } c|b$
- Let $a, b \in \mathbb{Z}$, where $a \neq 0 \text{ or } b \neq 0$. Then $c \in \mathbb{Z}^+$ is a greatest common divisor (gcd) of *a* and *b* if
  - $c|a, c|b$
  - For any common divisor *d* of *a* and *b*, we know $d|c$
- For all $a, b \in \mathbb{Z}^+$, there exists a unique greatest common divisor of *a* and *b*, written as *gcd*(*a*,*b*)
  - it is actually the smallest positive integer that is a linear combination of *a* and *b* $\text{gcd}(a, b) = ax + by$

Common Multiples

- Let $a, b \in \mathbb{Z}^+$. *c* is a common multiple of a and *b*. *c* is the least common multiple if it is the smallest positive common multiple of *a*, b, we write *c*=*lcm*(*a*,*b*)

- If $a, b \in \mathbb{Z}^+$ and $c = \mathrm{lcm}(a, b)$. For any *d* that is a common multiple of *a* and *b*, we know *c*|*d*

- For all $a, b \in \mathbb{Z}^+$, *ab*=lcm(*a*,*b*)gcd(*a*,*b*)

Systematic Way to Find GCD and LCM

- Exercise: Count the # of positive divisors of 360
  - Find 2 ways to do this in SageMath, hint: *factor*(.) and *divisors*(.)

```
sage: len(divisors(360))
24
```

```
sage: factor(360)
2^3 * 3^2 * 5
```

- Let $m = p_1^{e_1} p_2^{e_2} \cdots p_t^{e_t}, n = p_1^{f_1} p_2^{f_2} \cdots p_t^{f_t}$, with $e_i, f_i \geqslant 0,\ \forall e_i, f_i$ we have

$$gcd(m,n) = \prod_{i=1}^{t} p_i^{a_i},\ \text{ and } lcm(m,n) = \prod_{i=1}^{t} p_i^{b_i},$$

$$\text{where } a_i = \min(e_i, f_i), b_i = \max(e_i, f_i)$$

  - Find the gcd and lcm of $491891400 = 2^3 3^3 5^2 7^2 11^1 13^2$ and $1138845708 = 2^2 3^2 7^1 11^2 13^3 17^1$

GCD and LCM Related Functions

```
gcd(120, 64)
```

8

```
lcm(120, 64)
```

960

```
gcd(gcd(120, 55), gcd(25, 35))
```

5

```
gcd([120, 55, 25, 35])
```

5

Euclidean Algorithm

- Compute gcd(a,b)
  - $r_0 = a$, $r_1 = b$
  - For $i >= 1$, stop when $r_n = 0$
    - $r_{i+1} = r_i \bmod r_{i-1}$
  - gcd(a,b) = r_{n-1}
- Example: gcd (1650, 2420) = 110

i	r_i
0	2420
1	1650
2	770
4	110
5	0

Extended Euclidean Algorithm

- Compute *x*, *y* for gcd(*a*,*b*) = *ax* + *by*
  - $r_0=a,\ x_0=1,\ y_0=0$
  - $r_1= b,\ x_0=0,\ y_0=1$
  - For *i* >= *1*, stop when $r_n = 0$
    - $q_i=r_{i-1}$ div r_i
    - $x_{i+1} = x_{i-1} - q_i x_i$
    - $y_{i+1} = y_{i-1} - q_i y_i$
    - $r_{i+1} = r_{i-1} - q_i r_i$
  - $x=x_{i-1},\ y=y_{i-1},$

Example: a=2420, b=1650

i	x_i	y_i	r_i	q_i
0	1	0	2420	—
1	0	1	1650	1
2	1	−1	770	2
4	−2	3	110	7
5	—	—	0	

Linear Diophantine Equations

- For two non-zero integers a and b
- There exist integer solutions for $gcd(x,y)=ax+by$ ← extended Euclidean algorithm
- $ax+by=c$ has integer solutions iff gcd(a,b) | c
- $ax+by=1$ has integer solutions iff gcd(a,b) = 1
  - Relative prime (or co-prime)

```
gcd(1234,8765)
```

1

Congruence

- If *a* and *b* have the same remainder upon division by *n*, *a* is congruent to *b* modulo *n*
  - Written as $a \equiv b \pmod{n}$
  - That is $n|(a-b)$
- Example
  - $5|(23-8)$
  - $23 \equiv 8 \pmod{5}$
- Congruence is compatible with additions and multiplications: $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$ imply $a + c \equiv b + d \pmod{n}$ and $ac \equiv bd \pmod{n}$

Why Congruence is Useful?

- Compute $12^{2016} \pmod{19}$
- Steps:
  1. $12^2 = 144 \equiv 11 \pmod{19}$
  2. $12^3 = 11 \times 12 = 132 \equiv 18 \pmod{19}$
  3. $12^4 = 18 \times 12 = 216 \equiv 7 \pmod{19}$
  4. $12^5 = 7 \times 12 = 84 \equiv 8 \pmod{19}$
  5. $12^6 = 8 \times 12 = 96 \equiv 1 \pmod{19}$
- We know 2016 = 6 x 336, what's the answer?

Linear Congruence

- Linear congruence refers to an equation

$$ax \equiv b \pmod{m}$$

- It is the same as: $m|ax - b$, meaning that there exists an *y*, so that $ax - b = my$
  - Equivalent to $ax - my = b$
- The linear congruence has a solution iff $\gcd(a, m)|b$

Slide 22

Solving Linear Congruence $ax \equiv b \pmod{m}$

- First, we apply the extended Euclidean algorithm to find *u*, *v* so that $au + mv = \gcd(a, m)$
- If $\gcd(a, m) | b$, we have a solution $x_0 = ub/\gcd(a,m)$
  ← validate: show $m \overset{?}{|} \dfrac{aub}{\gcd(a, m)} - b$
- There are more solutions:

$$\frac{ub}{\gcd(a, m)} + i \frac{m}{\gcd(a, m)}, \text{ where } i = 0, 1, \ldots, \gcd(a, m) - 1$$

m | (a/gcd(a,m)) im

$$35x \equiv 10 \pmod{240}$$

Example of Linear Congruence

- Use extended Euclidean algorithm to get:

$$35 \times (-41) + 240 \times 6 = 5 = \gcd(35, 240)$$

- One solution: $x = ub / \gcd(35, 240) = (-41 \times 10)/5 = -82 \leftarrow 158 \pmod{240}$
- Step size: $240/5 = 48$, then we have the following solutions $\{158, 158 + 48, 158 + 2 \times 48, 158 + 3 \times 48, 158 + 4 \times 48\} = \{158, 206, 14, 62, 110\}$ under $\pmod{240}$

Multiplicative Inverse Modulo m

- *a* is invertible modulo *m* is there is an inverse *x* so that $ax \equiv 1 \pmod{m}$
  – a is invertible iff $\gcd(a, m) = 1$
- *x* is unique, and is denoted as a^{-1}, where $0 <= a^{-1} < |m|$
- Example: Find the inverse of 65 mod 321 (if exists)
  – First, we have: $65 \times (-79) + 321 \times 16 = 1$
  – Then, we have $a^{-1} = -79$ mod *321* ← what's the ans?

Congruence Classes

- The congruence classes of mod m are:

$$[a] = \{\forall x \equiv a \pmod{m}, x \in \mathbb{Z}\}, \text{where } a = 1, 2, \ldots, m-1$$

  - Equivalent classes: reflective, symmetric, and transitive
  - Note that [10] = [4] (mod 6)
- The set of congruence classes mod m is written as: $\mathbb{Z}/m\mathbb{Z} = \{[0], [1], \ldots, [m-1]\}$
  - Additions and multiplications are well defined ← see the next slide

Operations on $\mathbb{Z}/m\mathbb{Z}$

- Two operations: + and •
  - [a] + [b] = [a+b]
  - [a]•[b] = [a•b]
- ($\mathbb{Z}/m\mathbb{Z}$, +, •) is a commutative ring
  - + and • are: commutative, associative, • is distributive w.r.t +, 1 is the identity of multiplication
  - [a] may not have inverse, only if gcd(a,m)=1

Hence, it's not a field

Euler's Phi Function

- We define $\phi(n)$ as the number of $1 \leq z \leq n$, where $\gcd(z, n) = 1$
  - What is $\phi(10)$?
- Write code to compute phi function

```
x = 10
cnt = 0
for n in xrange(1, x):
    if gcd(n, x) == 1:
        print n,
        cnt = cnt + 1
print
print 'phi(', x, ')=', cnt
```

```
1 3 7 9
phi( 10 )= 4
```

Euler's Phi Function (cont.)

- Try other x values for $\phi(x)$
- Suppose *x* and *y* are two distinct primes, what are the relation among $\phi(x), \phi(y),$ and $\phi(x \times y)$?
  - Why?
  - In fact, as long as x and y are co-prime, the above discussion holds!
- SageMath has phi built-in

```
euler_phi(10)
```

4

```
euler_phi(123)
```

80

Divisors of an Integer

- We define $\tau(x)$ be the number of divisors of *x*
- We define $\sigma(x)$ be the sum of all the divisors of *x*

```
divisors(250)
```

```
[1, 2, 5, 10, 25, 50, 125, 250]
```

```
len(divisors(250))
```

```
8
```

$\tau(250)$

```
sum(divisors(250))
```

```
468
```

$\sigma(250)$

Built-in Sigma Function

- Again, actually SageMath has a sigma function

```
divisors(45)
```

[1, 3, 5, 9, 15, 45]

```
sigma(45)
```

78

$$1 + 3 + 5 + 9 + 15 + 45 = 78$$

```
sigma(45, 2)
```

2366

$$1^2 + 3^2 + 5^2 + 9^2 + 15^2 + 45^2 = 2366$$

```
sigma(45, 0)
```

6

$$1^0 + 3^0 + 5^0 + 9^0 + 15^0 + 45^0 = 6$$

What is this?

Amicable Pairs of Numbers

- An interesting Arab tradition: put two numbers 220 and 284 on their rings and give them to their spouses

$$\text{divisors}(\boxed{220}) = \{1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110, 220\}$$

$$1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = \boxed{284}$$

$$\text{divisors}(\boxed{284}) = \{1, 2, 4, 71, 142, 284\}$$

$$1 + 2 + 4 + 71 + 142 = \boxed{220}$$

Summary (So Far)

- We introduced various number theory functions in SageMath
- We will use them to introduce some cryptography results
- References:
  - http://www.sagemath.org ← Official Web and resources
  - http://www.gregorybard.com/SAGE.html ← Our textbook

How to Secretly Send Messages

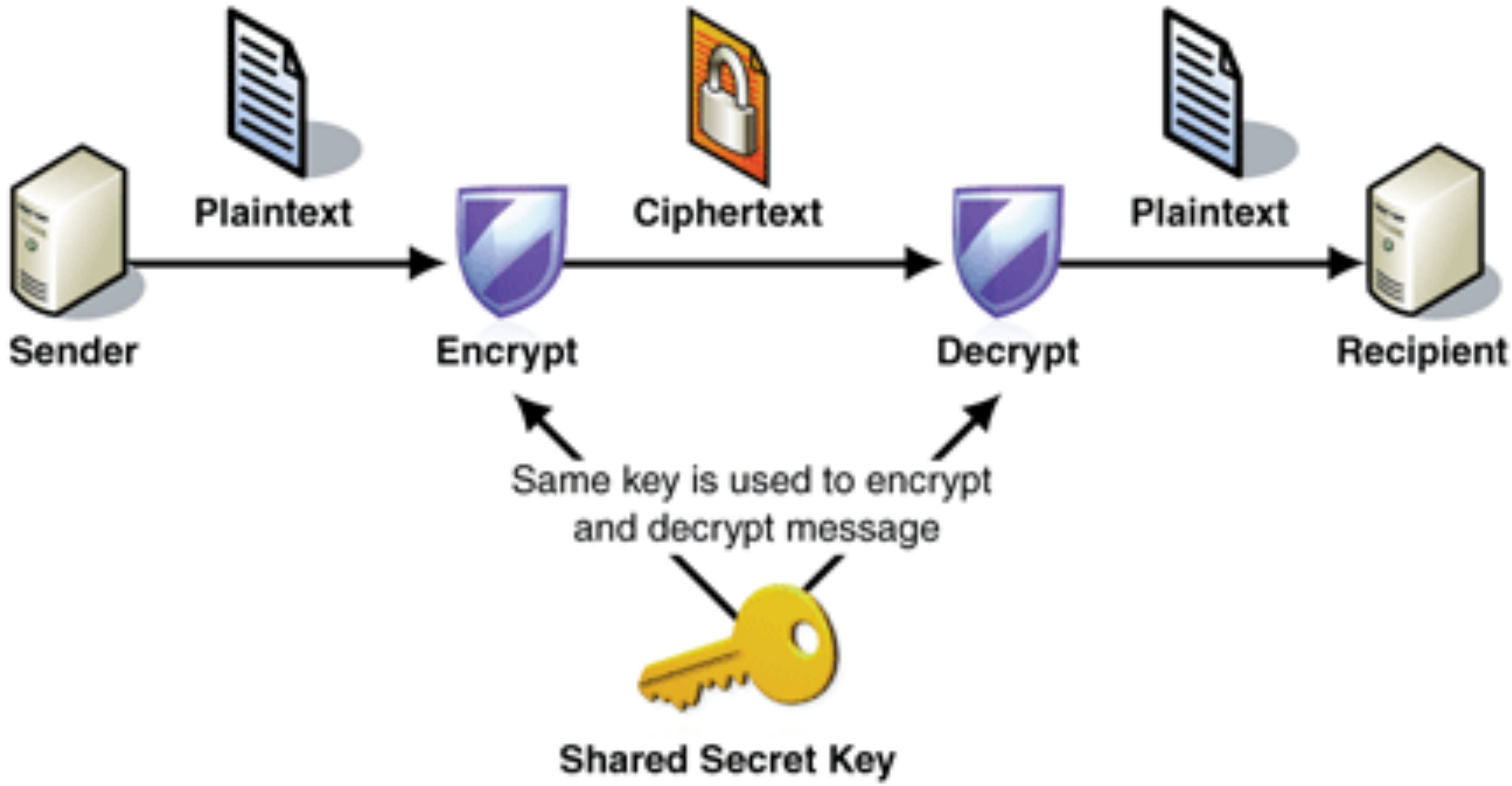
- Plaintext: human-readable messages
- Ciphertext: scrambled message
- Encryption: plaintext → ciphertext
- Decryption: ciphertext → plaintext

Naïve Way: ASCII Encoding

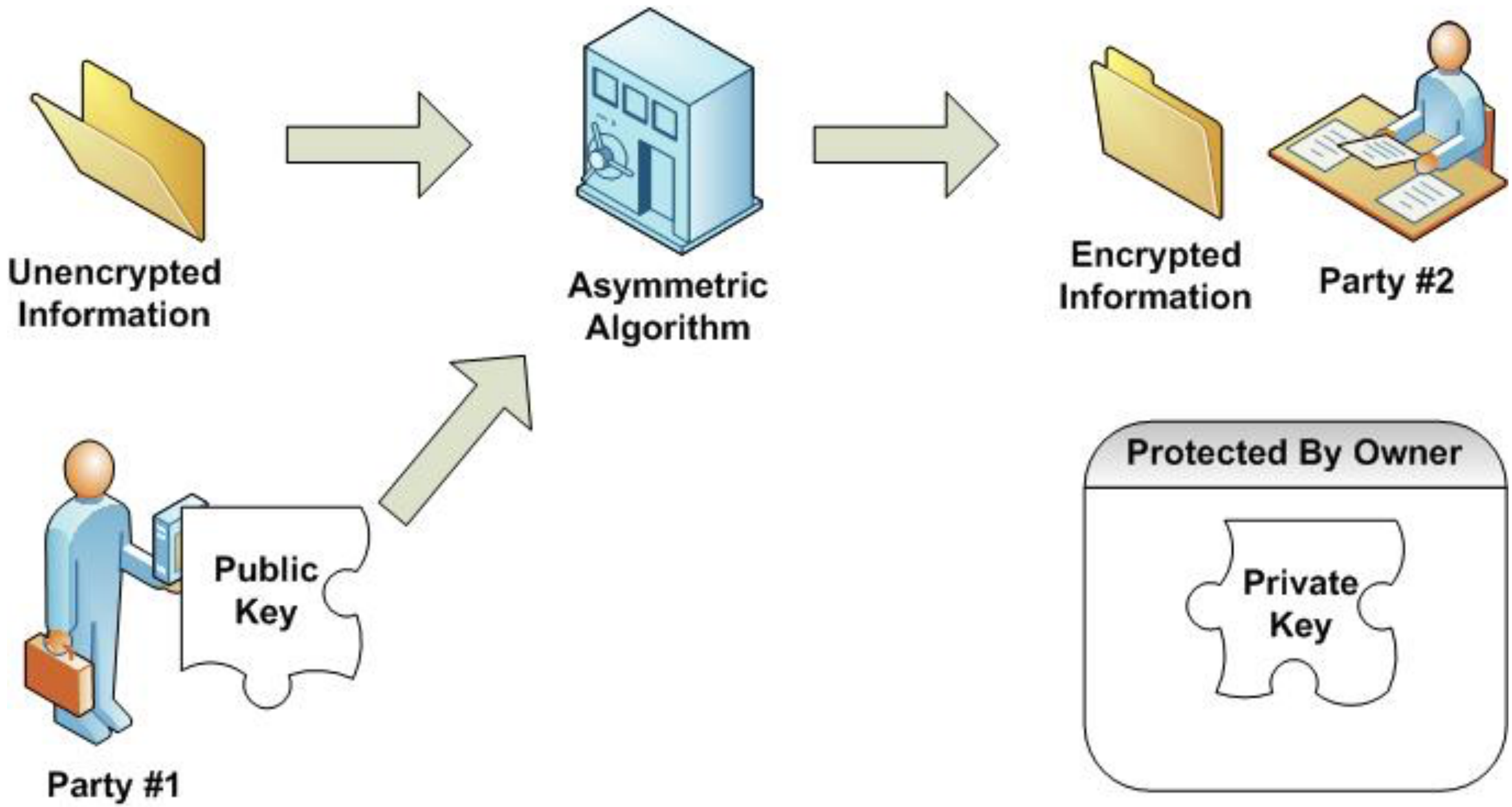
- Let $\Sigma = \{A, B, \ldots, Z\}$ be the English (uppercase) alphabet ← plaintext
- Let $\Phi = \{65, 66, \ldots, 90\}$ be the ASCII encodings, where $f : \Sigma \rightarrow \Phi$

- Example: "SCIENCE" → 83677369786769
- But it's too weak

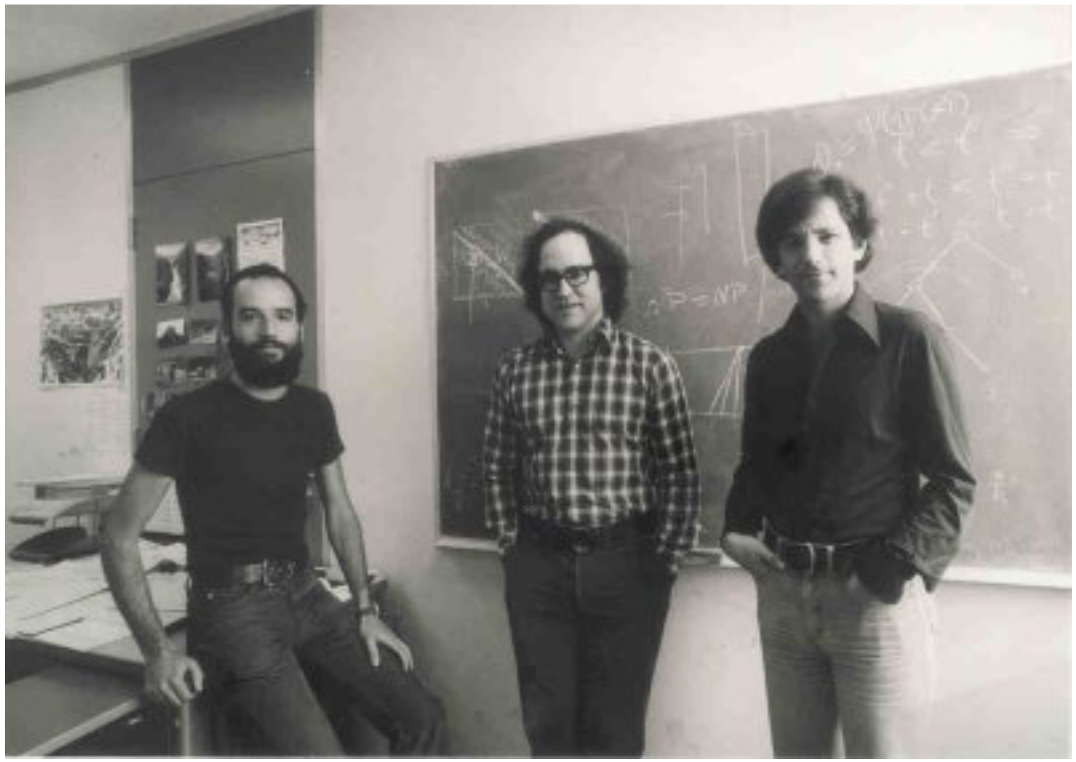
Symmetric Cryptography



Asymmetric Cryptography



A Popular Asymmetric Algorithm: RSA



R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. ACM Communications, 21, 2 (February 1978), 120-126.

RSA Pseudocode

1. Choose two huge primes *p* and *q*, and let *n=pq*
2. Let $e \in \mathbb{Z}$ be positive s.t. $\gcd(e, \phi(n)) = 1$
3. Find a $d \in \mathbb{Z}$ so that $de \equiv 1 \pmod{\phi(n)}$
4. Public key (*n*, *e*), private key (*p*, *q*, *d*)
5. For any integer *m* < *n*, encrypt *m* by $c \equiv m^e \pmod{n}$
6. Decrypt *c* using $m \equiv c^d \pmod{n}$

Let's try to walk through this in Sage!

Mersenne Primes

- Studied by Marin Mersenne in 17^{th} century
- Power of two minus 1: $M_m = 2^m - 1$
- If M_m is a prime, the it's called Mersenne primes
  - Sounds like a good way to create huge primes
  - is_prime(.) tells us if a number is prime
- Alternatively, we may use random_prime(…)

Generate the Primes for Keys

```
sage: p = 2^31 - 1
sage: is_prime(p)
True
sage: q = 2^61 -1
sage: is_prime(q)
True
sage: n = p*q
sage: n
4951760154835678088235319297
```

BTW, far-apart p and q is very bad choices in the sense of security

RSA Pseudocode, Step 2

1. Choose two huge primes *p* and *q*, and let *n*=*pq*
2. Let $e \in \mathbb{Z}$ be positive s.t. $\gcd(e, \phi(n)) = 1$
3. Find a $d \in \mathbb{Z}$ so that $de \equiv 1 \pmod{\phi(n)}$
4. Public key (*n*, *e*), private key (*p*, *q*, *d*)
5. For any integer *m* < *n*, encrypt *m* by $c \equiv m^e \pmod{n}$
6. Decrypt *c* using $m \equiv c^d \pmod{n}$

Find a Coprime of Euler Phi

- We learned how to calculate euler_phi(.)
- Let's randomly pick a number < phi, and wish they are coprime
- We stop only when we find a coprime *e*
  – Usage of while loop….

While-Loop to Find *e*

```
sage: phi=euler_phi(n); phi
4951760152529835076874141700
sage: e=int(random() * (phi-1)) + 1
sage: while gcd(e, phi) !=1 :
....:     e=int(random() * (phi-1)) + 1
....:
sage: e
3093458420861290024932474881
```

What does this do?

RSA Pseudocode, Step 3

1. Choose two huge primes *p* and *q*, and let *n*=*pq*
2. Let $e \in \mathbb{Z}$ be positive s.t. $\gcd(e, \phi(n)) = 1$
3. Find a $d \in \mathbb{Z}$ so that $de \equiv 1 \pmod{\phi(n)}$
4. Public key (*n*, *e*), private key (*p*, *q*, *d*)
5. For any integer *m* < *n*, encrypt *m* by $c \equiv m^e \pmod{n}$
6. Decrypt *c* using $m \equiv c^d \pmod{n}$

How to Find *d*?

- Sounds tricky: $de \equiv 1 \pmod{\phi(n)}$
  - $\phi(n) | de - 1$
  - or $de - 1 = k \times \phi(n)$ for some integer *k*
  - or $de - k\phi(n) = 1$
- Think again
  - What are given? ← *e* and *phi*
  - What do we want to determine? ← *d* and *k*
- How can we find two integers *d* and *k?*
  - *Recall that e and phi are coprime*

Extended Euclidean Algorithm!

- We know $\gcd(a, b) = xa + yb$ for some *x* and *y*
- Sage command xgcd(a, b) returns *(gcd(a,b), x, y)* as a 3-tuple

```
sage: tuple=xgcd(e, phi); tuple
(1, -16522784699765489228624745 79,
103220967678441436335607125 3)
sage: d = Integer(mod(tuple[1], phi)); d
32994816825532861540116671 21
sage: mod(d*e, phi)
1
```

Found our d

Validate d

RSA Pseudocode, Step 4

1. Choose two huge primes *p* and *q*, and let *n*=*pq*
2. Let $e \in \mathbb{Z}$ be positive s.t. $\gcd(e, \phi(n)) = 1$
3. Find a $d \in \mathbb{Z}$ so that $de \equiv 1 \pmod{\phi(n)}$
4. Public key (*n*, *e*), private key (*p*, *q*, *d*)
5. For any integer *m* < *n*, encrypt *m* by $c \equiv m^e \pmod{n}$
6. Decrypt *c* using $m \equiv c^d \pmod{n}$

Public and Private Keys

sage: (n,e) ← Public Key

(4951760154835678088235319297, 3093458420861290024932474881)

sage: (p,q,d) ← Private Key

(2147483647, 2305843009213693951, 3299481682553286154011667121)

RSA Pseudocode, Step 5

1. Choose two huge primes *p* and *q*, and let *n*=*pq*
2. Let $e \in \mathbb{Z}$ be positive s.t. $\gcd(e, \phi(n)) = 1$
3. Find a $d \in \mathbb{Z}$ so that $de \equiv 1 \pmod{\phi(n)}$
4. Public key (*n*, *e*), private key (*p*, *q*, *d*)
5. For any integer *m* < *n*, encrypt *m* by $c \equiv m^e \pmod{n}$
6. Decrypt *c* using $m \equiv c^d \pmod{n}$

Encrypt the Message (and Fail)

- "SCIENCE" → m=83677369786769
- $c \equiv m^e \pmod{n}$

e=3093458420861290024932474881

```
sage: m=83677369786769
sage: c=mod(m^e, n)
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-19-c5605db94841> in <module>()
----> 1 c=mod(m**e, n)
/usr/lib/sagemath/local/lib/python2.7/site-packages/sage/rings/
integer.so in sage.rings.integer.Integer.__pow__ (sage/rings/integer.c:
14001)()
RuntimeError: exponent must be at most 9223372036854775807
```

Repeated Squaring

- Start from $d = 1$
- Convert b into binary $(b_1, b_2, \ldots, b_k)$
- Iterate i from 1 to k
  - $d = d * d \bmod n$ Move 1 digit toward left
  - If $b_i = 1$, let $d = d * a \bmod n$

If there is a 1, multiply by a

Example of Repeated Squaring

- Derive $3^6 \leftarrow 6 = (110)_2$
- Step 1: d=1
- Step 2: i=1, d=1*1 = 1, d = 1*3 = 3
- Step 3: i=2, d=3*3 = 9, d=9*3 = 27
- Step 4: I=3, d=27*27=729

- Note that I ignore modulus here for brevity

Repeated Square Function

- Save the following code as rsmod.sage ← Uah, pay attentions to indents, like all python sources
- Load it using %runfile rsmod.sage
- Test it

```
def rsmod(a, b, n):
   d=1
   for i in list(Integer.binary(b)):
      d=mod(d*d, n)
      if Integer(i) == 1:
          d = mod(d*a, n)
   return Integer(d)
```

```
sage: %runfile rsmod.sage
sage: rsmod(3,6,100000)
729
```

Now We are Back on Track

- Use *e* and *n* (=*pq*) to encrypt *m* into *c*

sage: c=rsmod(m, e, n)

sage: c

1406082576299748012744893983

- Last step, decode c using *d* and *n*

sage: m2=rsmod(c, d, n); m2==m

True

Recap: RSA Pseudocode

1. Choose two huge primes *p* and *q*, and let *n*=*pq*
2. Let $e \in \mathbb{Z}$ be positive s.t. $\gcd(e, \phi(n)) = 1$
3. Find a $d \in \mathbb{Z}$ so that $de \equiv 1 \pmod{\phi(n)}$
4. Public key (*n*, *e*), private key (*p*, *q*, *d*)
5. For any integer *m* < *n*, encrypt *m* by $c \equiv m^e \pmod{n}$
6. Decrypt *c* using $m \equiv c^d \pmod{n}$

We have done this!

Naïve Way to Break It

- Figure out the *p* and *q* values. But, how hard is factorization?

```
sage: time factor(random_prime(2^32)*random_prime(2^32))
CPU times: user 0.01 s, sys: 0.00 s, total: 0.01 s
sage: time factor(random_prime(2^64)*random_prime(2^64))
CPU times: user 0.05 s, sys: 0.00 s, total: 0.05 s
sage: time factor(random_prime(2^96)*random_prime(2^96))
CPU times: user 3.54 s, sys: 0.04 s, total: 3.58 s
sage: time factor(random_prime(2^128)*random_prime(2^128))
CPU times: user 534.39 s, sys: 0.12 s, total: 534.51 s
```

Growing into something

- Well there are many primes between 2^{511} and 2^{512} ← Attackers cannot be that lucky

Flawed Random Number Generators

- 1995 Goldberg-Wagner: During any particular second, the Netscape browser generates only about 2^{47} possible keys
- 2008 Bello: Debian and Ubuntu generate $<2^{20}$ possible keys for SSH, OpenVPN, etc
- What we can do is:
  - Generate many private keys on a device
  - Check if any of these private keys divide n
  - Finding p (and q) is no longer impossible

Pollard's *p-1* Attack

- Due to John Pollard in 1974
- Only work on special primes ← Smooth primes
- A number is *k*-smooth if all of its prime factors are smaller than *k*
- Example: 10, 100, and 2^{1024} are all 6-smooth, but 14 is not

Background of Pollard's *p-1* Attack

- RSA's n can be readily factorized if $p-1$ or $q-1$ are smooth ← only have small factors
  - Wait, but we don't know p nor q, right? Indeed …
- Checking if an integer k is B-smooth may be to computationally demanding
  - Compare it against if $k|B!$

Integer k Divides $B!$

Lemma: $k \mid B!$ implies k is B-smooth

Proof:

- Assume k is not B-smooth, then there existing an integer $f|k$, where $f > B$.
- f does not divide any $b' <= B$.
- Since we know $p|ab$ iff $p|a$ or $p|b$, k does not divide $B!$ for sure.

Note: the converse is false, proof is left as exercise

Fermat's Little Theorem

Theorem: Given a prime number *p*, and any $a \not\equiv 0 \pmod{p}$, we know $a^p \equiv a \pmod{p}$

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

Proof:

The first *p-1* positive multiples of *a* are: *a*, *2a*, *3a*, ..., (*p-1*)*a*. These multiples are all distinct, because if *xa* = *ya* (mod *p*), we know *x*=*y* (since *p* is a prime).

Fermat's Little Theorem (cont.)

The *p-1* multiples are congruent to *1, 2, …, p-1,* in some order (the precise permutation is not important). Let's multiply all of them together and we have $a \cdot 2a \cdots (p-1)a \equiv 1 \cdot 2 \cdots (p-1) \pmod p$, and then $a^{p-1}(p-1)! \equiv (p-1)! \pmod p$. Getting rid of (*p-1*)*!* at both sides yields the theorem.

How Fermat's Little Theorem Helps?

- Say p-1 | $B!$, there is a k so that $k(p\text{-}1) = B!$
- Then we have

$$2^{B!} = 2^{k(p-1)} = (2^{p-1})^k \equiv 1^k \equiv 1 \pmod p$$

- Or $c = (2^{B!} - 1)$ is a multiple of p
  - Both in ordinary integers and under mod p
  - I skip some technical details
- OK. What I'm talking about? Since n=pq, a multiple of p; gcd(c, n) is a multiple of p

The Pollard's *p-1* Attack

- We compute $c=2^{B!}-1$ mod n
- We compute gcd(c,n)
- If it is between 1 and n, it is p (because q is a prime) ← we can then compute $q = n/p$
- We have broken the public key!
- But, if gcd(c,n)=n, we fail ← this only happens if both $p-1$ and $q-1$ divide B*!*
  - Well, we ignore these corner cases for brevity. Just remember *p-1* does not always work

There is a Catch…..

- How to compute $c=2^{B!}-1$ mod n? Is it realistic?
- Remember p-1 must be *B*-smooth, and B is not going to be small!
  – Say $2^{(10000!)}$ mod n
- Can we really do this? NO….
- Need the last twist….

$$c_1 = 2^1 \pmod{n}$$
$$c_2 = (c_1)^2 \pmod{n}$$
$$c_3 = (c_2)^3 \pmod{n}$$
$$c_4 = (c_3)^4 \pmod{n}$$
$$\vdots$$
$$c_B = (c_{B-1})^B \pmod{n}$$

The Last Twist

$$c_1 = 2^1 \pmod{n}$$
$$c_2 = (c_1)^2 \pmod{n}$$
$$c_3 = (c_2)^3 \pmod{n}$$
$$c_4 = (c_3)^4 \pmod{n}$$
$$\vdots$$
$$c_B = (c_{B-1})^B \pmod{n}$$

$$c_4 \equiv (c_3)^4 \equiv ((c_2)^3)^4 \equiv (((c_1)^2)^3)^4 \equiv ((((2^1)^2)^3)^4)$$

Similarly $C_B \equiv 2^{B!} \pmod{n}$, and we can recursively calculate C_B

Put Pollard's *p-1* Together

Sage:n=4442660146065829115772553600812801729789078746371942790312811803660 57

sage: B_fac=factorial(2^25)

Well, I did not apply the last optimization

sage: c=Integer(pow(2,B_fac,n)) - 1

sage: p=gcd(c,n); p

1267650600228229401496703217601

sage: q=n/p; q

35046409044148024855564290035771968 2857

sage: p*q == n

True

Yeah, it works

Summary (Second Half)

- We introduced symmetric and asymmetric cryptography systems
- We walked through RSA algorithm
- We discussed one of the RSA attacks
- There are many other attacks ← out of scope
- References:
  - http://www.gregorybard.com/SAGE.html ← Our textbook
  - http://doc.sagemath.org/html/en/thematic_tutorials/numtheory_rsa.html ← Introduction on RSA
  - https://www.hyperelliptic.org/tanja/vortraege/facthacks-29C3.pdf ← Many more attacks

SageMath #2 Homework (S2)

1. (2%) Write a SageMath program to find out at least 3 amicable pairs, including (220, 284)
2. (1%) Use Pollard's *p-1* attack to factorize this number:
   n=862021547643631582396998212208722914288258644234791307950582916442747222039795609417741932278317121

You need to explain and run your code in front of the TA, or you get zero point