# ENCRYPTION

David Kauchak
CS52 – Spring 2015

## Admin

Assignment 6

4 more assignments:
- Assignment 7, due 11/13 5pm
- Assignment 8, due 11/20 5pm
- Assignments 9 & 10, due 12/9 11:59pm

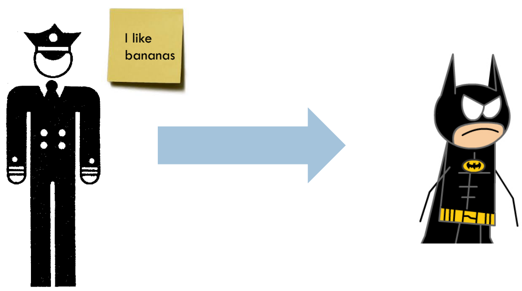
## Admin

Midterm next Thursday
- Covers everything from 9/24 – 10/27 + some minor SML
- Will not have to *write* any assembly
- 2 pages of notes
- Review sessions next week (TBA)

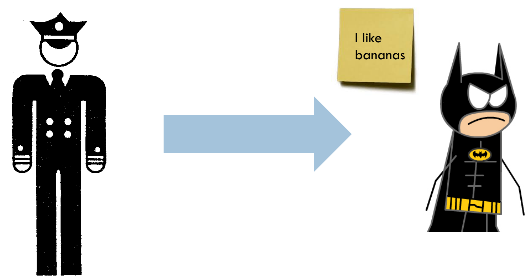
## Encryption

What is it and why do we need it?

## Encryption

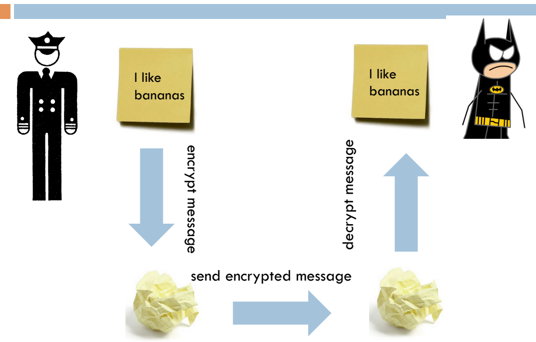



## Encryption




## Encryption: a bad attempt


## Encryption: the basic idea

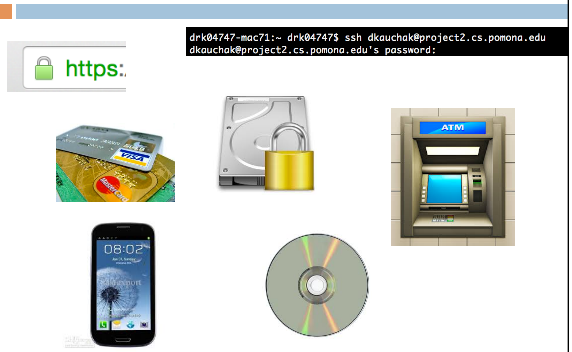


## Encryption: a better approach

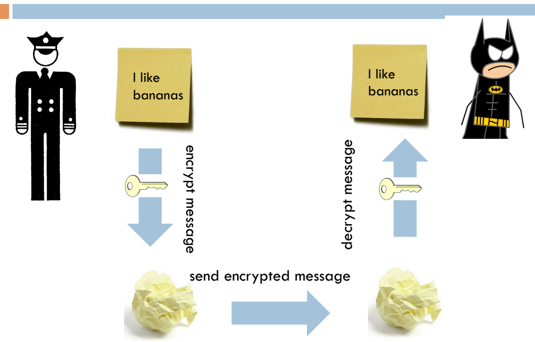

## Encryption uses

Where have you seen encryption used?

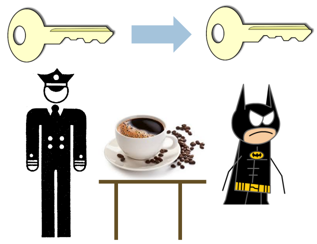
## Encryption uses




## Private key encryption

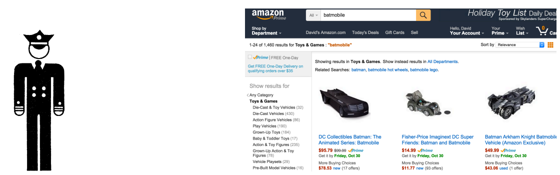


## Private key encryption

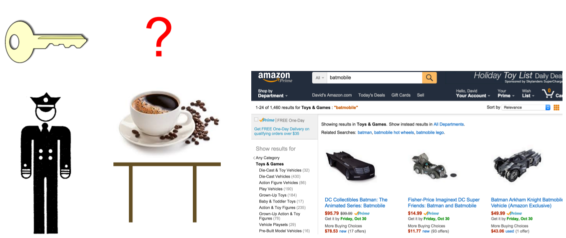


Any problems with this?

## Private key encryption



## Private key encryption

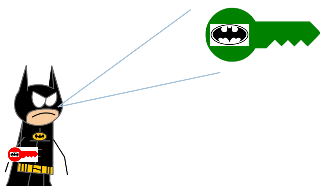



## Public key encryption

private key public key

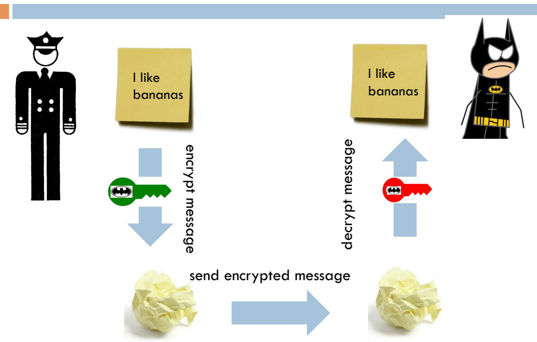
Two keys, one you make publicly available and one you keep to yourself

## Public key encryption



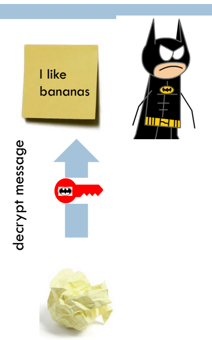
Share your public key with everyone

## Public key encryption




## Public key encryption

Only the person with the private key can decrypt!




## Modular arithmetic

Normal arithmetic:

a = b

a is equal to b *or a-b = 0*

Modular arithmetic:

a ≡ b *(mod n)*

a-b = n*k for some integer k or

a = b + n*k for some integer k or

a % n = b % n (where % is the mod operator)

## Modular arithmetic

Which of these statements are true?

12 ≡ 5 *(mod 7)*

52 ≡ 92 *(mod 10)*

17 ≡ 12 *(mod 6)*

a-b = n*k for some integer k or
a = b + n*k for some integer k or
a % n = b % n (where % is the mod operator)

65 ≡ 33 *(mod 32)*

## Modular arithmetic

Which of these statements are true?

12 ≡ 5 *(mod 7)* — 12-5 = 7 = 1*7; 12 % 7 = 5 = 5 % 7

52 ≡ 92 *(mod 10)* — 92-52 = 40 = 4*10; 92 % 10 = 2 = 52 % 20

17 ≡ 12 *(mod 6)* — 17-12 = 5; 17 % 6 = 5; 12 % 6 = 0

65 ≡ 33 *(mod 32)* — 65-33 = 32 = 1*32; 65 % 32 = 1 = 33 % 32

## Modular arithmetic properties

If:

a ≡ b *(mod n)*

then:

a mod n ≡ b mod n *(mod n)*

"mod"/remainder operator

congruence (mod n)

## Modular arithmetic properties

If:

a ≡ b *(mod n)*

then:

a mod n ≡ b mod n *(mod n)*

More importantly:

(a+b) mod n ≡ (a mod n) + (b mod n) *(mod n)*

and

(a*b) mod n ≡ (a mod n) * (b mod n) *(mod n)*

What do these say?

## Modular arithmetic

Why talk about modular arithmetic and congruence? How is it useful? Why might it be better than normal arithmetic?

We can limit the size of the numbers we're dealing with to be at most n (if it gets larger than n at any point, we can always just take the result % n)

The mod operator can be thought of as mapping a number in the range 0 … number-1

## GCD

What does GCD stand for?

## Greatest Common Divisor

gcd(a, b) is the largest positive integer that divides both numbers without a remainder

gcd(25, 15) = ?

## Greatest Common Divisor

gcd(a, b) is the largest positive integer that divides both numbers without a remainder

gcd(25, 15) = 5

| | 25 | 15 |
|---|---|---|
| Divisors: | 25 | 15 |
| | 5 | 5 |
| | 1 | 3 |
| | | 1 |

## Greatest Common Divisor

gcd(a, b) is the largest positive integer that divides both numbers without a remainder

gcd(100, 52) = ?

## Greatest Common Divisor

gcd(a, b) is the largest positive integer that divides both numbers without a remainder

gcd(100, 52) = 4

Divisors:

| 100 | 52 |
|---|---|
| 100 | 52 |
| 50 | 13 |
| 25 | 4 |
| 20 | 2 |
| 10 | 1 |
| 5 | |
| 4 | |
| 2 | |
| 1 | |

## Greatest Common Divisor

gcd(a, b) is the largest positive integer that divides both numbers without a remainder

gcd(14, 63) = ?

gcd(7, 56) = ?

gcd(23, 5) = ?

gcd(100, 9) = ?

gcd(111, 17) = ?

## Greatest Common Divisor

gcd(a, b) is the largest positive integer that divides both numbers without a remainder

gcd(14, 63) = 7

gcd(7, 56) = 7

gcd(23, 5) = 1

gcd(100, 9) = 1

gcd(111, 17) = 1

Any observations?

## Greatest Common Divisor

When the gcd = 1, the two numbers share no factors/divisors in common

If gcd(a,b) = 1 then a is *relatively prime* to b

This a weaker condition than primality, since any two prime numbers are also relatively prime, but not vice versa

## Greatest Common Divisor

A useful property:

If two numbers are relatively prime (i.e. gcd(a,b) = 1), then there exists a c such that

a*c mod b = 1

## RSA public key encryption

Have you heard of it?

What does it stand for?

## RSA public key encryption

RSA is one of the most popular public key encryption algorithms in use

RSA = Ron Rivest, Adi Shamir and Leonard Adleman

## RSA public key encryption

1. Choose a bit-length *k*
   Security increase with the value of *k*, though so does computation
2. Choose two primes *p* and *q* which can be represented with at most *k* bits
3. Let $n = pq$ and $\varphi(n) = (p-1)(q-1)$
   φ() is called *Euler's totient function*
4. Find d such that $0 < d < n$ and $gcd(d,\varphi(n)) = 1$
5. Find e such that $de \bmod \varphi(n) = 1$
   Remember, we know one exists!

## RSA public key encryption

p: prime number
q: prime number
$n = pq$

$\varphi(n) = (p-1)(q-1)$
d: $0 < d < n$ and $gcd(d,\varphi(n)) = 1$
e: $de \bmod \varphi(n) = 1$

Given this setup, you can prove that given a number *m*:

$$(m^e)^d = m^{ed} = m \pmod{n}$$

What does this do for us, though?

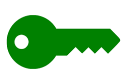
## RSA public key encryption

p: prime number
q: prime number
$n = pq$

$\varphi(n) = (p-1)(q-1)$
d: $0 < d < n$ and $gcd(d,\varphi(n)) = 1$
e: $de \bmod \varphi(n) = 1$

private key

(d, n)



public key

(e, n)

## RSA encryption/decryption

private key — (d, n)

public key — (e, n)

You have a number *m* that you want to send encrypted

$encrypt(m) = m^e \bmod n$ (uses the public key)

How does this encrypt the message?

## RSA encryption/decryption

private key public key

(d, n) (e, n)

You have a number *m* that you want to send encrypted

encrypt(m) = $m^e$ mod n (uses the public key)

- Maps *m* onto some number in the range 0 to *n*-1
- If you vary e, it will map to a different number
- Therefore, unless you know d, it's hard to know original *m* was after the transformation

## RSA encryption/decryption

private key public key

(d, n) (e, n)

You have a number *m* that you want to send encrypted

encrypt(m) = $m^e$ mod n (uses the public key)

decrypt(z) = $z^d$ mod n (uses the private key)

Does this work?

## RSA encryption/decryption

encrypt(m) = $m^e$ mod n

decrypt(z) = $z^d$ mod n

decrypt(z) = decrypt($m^e$ mod n) z is some encrypted message

= $(m^e \bmod n)^d$ mod n definition of decrypt

= $(m^e)^d$ mod n modular arithmetic

= m mod n $(m^e)^d = m^{ed} = m \pmod n$

Did we get the original message?

## RSA encryption/decryption

encrypt(m) = $m^e$ mod n

decrypt(z) = $z^d$ mod n

decrypt(z) = decrypt($m^e$ mod n) z is some encrypted message

= $(m^e \bmod n)^d$ mod n definition of decrypt

= $(m^e)^d$ mod n modular arithmetic

= m mod n $(m^e)^d = m^{ed} = m \pmod n$

If $0 \le m < n$, yes!

## RSA encryption: an example

p: prime number
q: prime number
n = pq

$\varphi(n)$ = (p-1)(q-1)
d: 0 < d < n and gcd(d,$\varphi(n)$) = 1
e: de mod $\varphi(n)$ = 1

p = 3
q = 13
n = ?
$\varphi(n)$ = ?
d = ?
e = ?

## RSA encryption: an example

p: prime number
q: prime number
n = pq

$\varphi(n)$ = (p-1)(q-1)
d: 0 < d < n and gcd(d,$\varphi(n)$) = 1
e: de mod $\varphi(n)$ = 1

p = 3
q = 13
n = ?

## RSA encryption: an example

p: prime number
q: prime number
n = pq

$\varphi(n)$ = (p-1)(q-1)
d: 0 < d < n and gcd(d,$\varphi(n)$) = 1
e: de mod $\varphi(n)$ = 1

p = 3
q = 13
n = 3*13 = 39

## RSA encryption: an example

p: prime number
q: prime number
n = pq

$\varphi(n)$ = (p-1)(q-1)
d: 0 < d < n and gcd(d,$\varphi(n)$) = 1
e: de mod $\varphi(n)$ = 1

p = 3
q = 13
n = 39
$\varphi(n)$ = ?

## RSA encryption: an example

p: prime number
q: prime number
n = pq

$\varphi(n)$ = (p-1)(q-1)
d: 0 < d < n and gcd(d,$\varphi(n)$) = 1
e: de mod $\varphi(n)$ = 1

p = 3
q = 13
n = 39
$\varphi(n)$ = 2*12 = 24

## RSA encryption: an example

p: prime number
q: prime number
n = pq

$\varphi(n)$ = (p-1)(q-1)
d: 0 < d < n and gcd(d,$\varphi(n)$) = 1
e: de mod $\varphi(n)$ = 1

p = 3
q = 13
n = 39
$\varphi(n)$ = 24
d = ?
e = ?

## RSA encryption: an example

p: prime number
q: prime number
n = pq

$\varphi(n)$ = (p-1)(q-1)
d: 0 < d < n and gcd(d,$\varphi(n)$) = 1
e: de mod $\varphi(n)$ = 1

p = 3
q = 13
n = 39
$\varphi(n)$ = 24
d = 5
e = 5

## RSA encryption: an example

n = 39
d = 5
e = 5

encrypt(m) = $m^e$ mod n

decrypt(z) = $z^d$ mod n

encrypt(10) = ?

## RSA encryption: an example

n = 39
d = 5
e = 5

encrypt(m) = $m^e$ mod n

decrypt(z) = $z^d$ mod n

encrypt(10) = $10^5$ mod 39 = 4

## RSA encryption: an example

n = 39
d = 5
e = 5

encrypt(m) = $m^e$ mod n

decrypt(z) = $z^d$ mod n

encrypt(10) = $10^5$ mod 39 = 4

decrypt(4) = ?

## RSA encryption: an example

n = 39
d = 5
e = 5

encrypt(m) = $m^e$ mod n

decrypt(z) = $z^d$ mod n

encrypt(10) = $10^5$ mod 39 = 4

decrypt(4) = $4^5$ mod 39 = 10

## RSA encryption: an example

n = 39
d = 5
e = 5

encrypt(m) = $m^e$ mod n

decrypt(z) = $z^d$ mod n

encrypt(2) = ?

## RSA encryption: an example

n = 39
d = 5
e = 5

encrypt(m) = $m^e$ mod n

decrypt(z) = $z^d$ mod n

encrypt(2) = $2^5$ mod 39 = 32 mod 39 = 32

decrypt(32) = ?

## RSA encryption: an example

n = 39
d = 5
e = 5

encrypt(m) = $m^e$ mod n

decrypt(z) = $z^d$ mod n

encrypt(2) = $2^5$ mod 39 = 32 mod 39 = 32

decrypt(32) = $32^5$ mod 39 = 2

## RSA encryption in practice

For RSA to work: $0 \leq m < n$

What if our message isn't a number?

What if our message is a number that's larger than n?

## RSA encryption in practice

For RSA to work: $0 \leq m < n$

What if our message isn't a number?
- We can always convert the message into a number (remember everything is stored in binary already somewhere!)

What if our message is a number that's larger than n?
- Break it into m sized chunks and encrypt/decrypt those chunks

## RSA encryption in practice

encrypt("I like bananas") =

| | |
|---|---|
| 0101100101011100 … | encode as a binary string (i.e. number) |
| 4, 15, 6, 2, 22, … | break into multiple < n size numbers |
| 17, 1, 43, 15, 12, … | encrypt each number |

## RSA encryption in practice

decrypt((17, 1, 43, 15, 12, …)) =

| | |
|---|---|
| 4, 15, 6, 2, 22, … | decrypt each number |
| 0101100101011100 … | put back together |
| "I like bananas" | turn back into a string (or whatever the original message was) |

Often encrypt and decrypt just assume sequences of bits and the interpretation is done outside