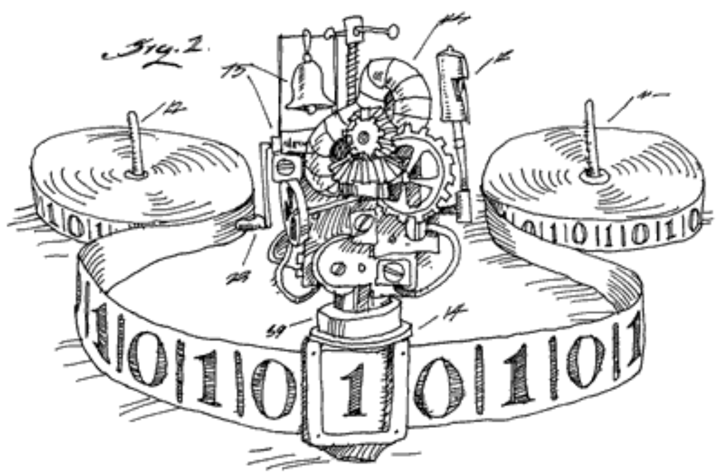
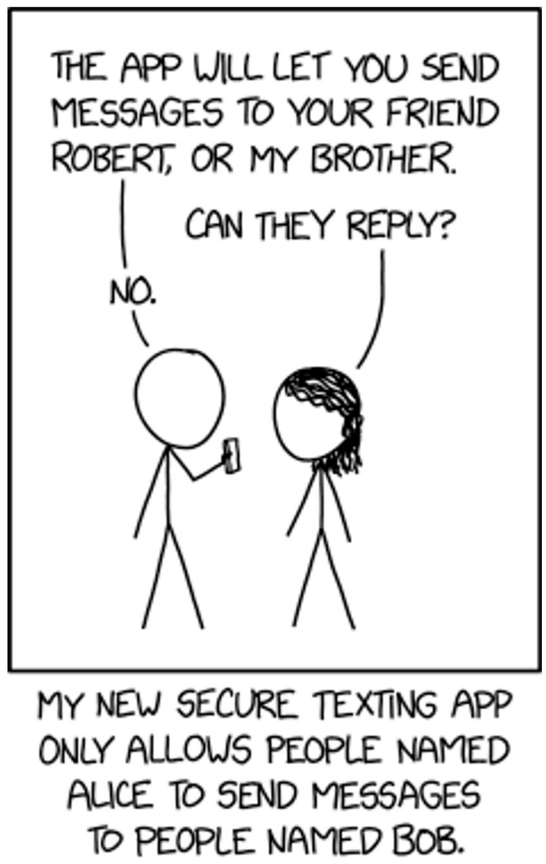
EECS 376: Foundations of Computer Science

Lecture 24 - RSA



RSA:
Public-Key Encryption and Signatures



Modular Multiplicative Inverse in Polynomial Time

Input: Integers $m, x < m$ with $\gcd(x, m) = 1$
Output: $x^{-1} \bmod m$, i.e.,

$$z \text{ where } z \cdot x \equiv 1 \bmod m$$

Algorithm:

1. Run **ExtendedEuclid**(x, m) to find (a, b) such that 1 = ax+bm.
2. Return a.

Correctness: 1 ≡ ax+bm (mod m), so 1 ≡ a·x (mod m).

Running time: poly(log(m))

ExtendedEuclid:
Input: Integers x ≥ y ≥ 0, not both 0
Output: Triple (g,a,b) of integers where g = gcd(x,y) = ax+by.

Extended Euclidean Algorithm (page 241)

Input: integers $x \geq y \geq 0$, not both zero
Output: their greatest common divisor $g = \gcd(x, y)$, and integers a, b such that $ax + by = g$

function $\textsc{ExtendedEuclid}(x, y)$
  if $y = 0$ **then return** $(x, 1, 0)$
  $(g, a', b') = \textsc{ExtendedEuclid}(y, x \bmod y)$
  return $(g, b', a' - b' \cdot \lfloor x/y \rfloor)$

Step	x	y
0	21	13
1	13	8
2	8	5
3	5	3
4	3	2
5	2	1
6	1	0

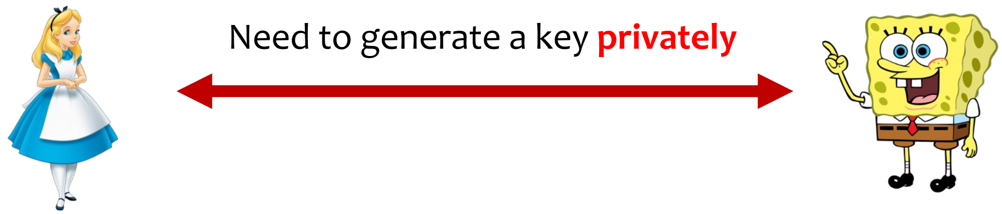
Step	x	y	g	a	b
6	1	0	1	1	0
5	2	1	1	0	$1 - 0 \cdot \lfloor \frac{2}{1} \rfloor = 1$
4	3	2	1	1	$0 - 1 \cdot \lfloor \frac{3}{2} \rfloor = -1$
3	5	3	1	-1	$1 - (-1) \cdot \lfloor \frac{5}{3} \rfloor = 2$
2	8	5	1	2	$-1 - 2 \cdot \lfloor \frac{8}{5} \rfloor = -3$
1	13	8	1	-3	$2 - (-3) \cdot \lfloor \frac{13}{8} \rfloor = 5$
0	21	13	1	5	$-3 - 5 \cdot \lfloor \frac{21}{13} \rfloor = -8$

Thus, we have by = −8·13 ≡ 1 (mod 21). Translating b to an element of Z21, we get b = −8 ≡ 13 (mod 21), which means 13 is its own inverse modulo 13.

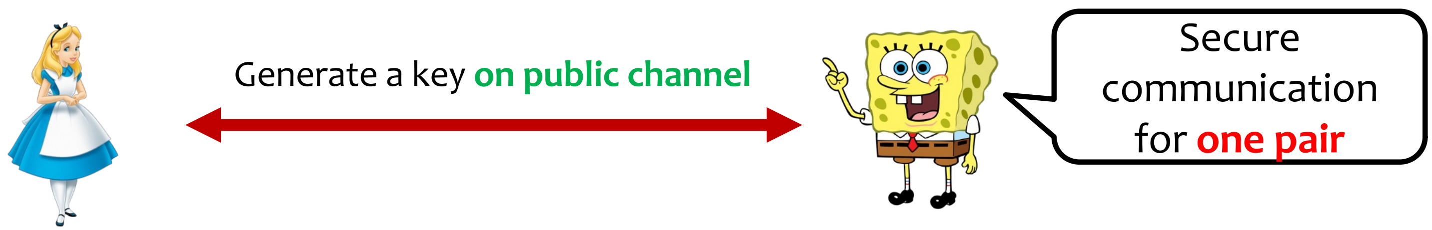
Public Key Encryption

Public-Key Encryption?

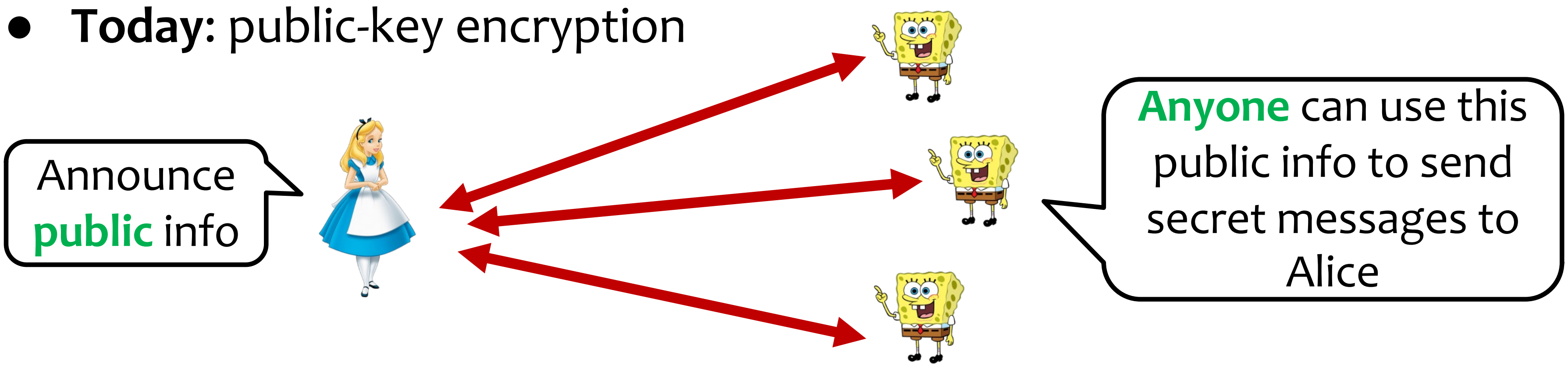
- One-time pad



- Key exchange (using Diffie-Hellman)

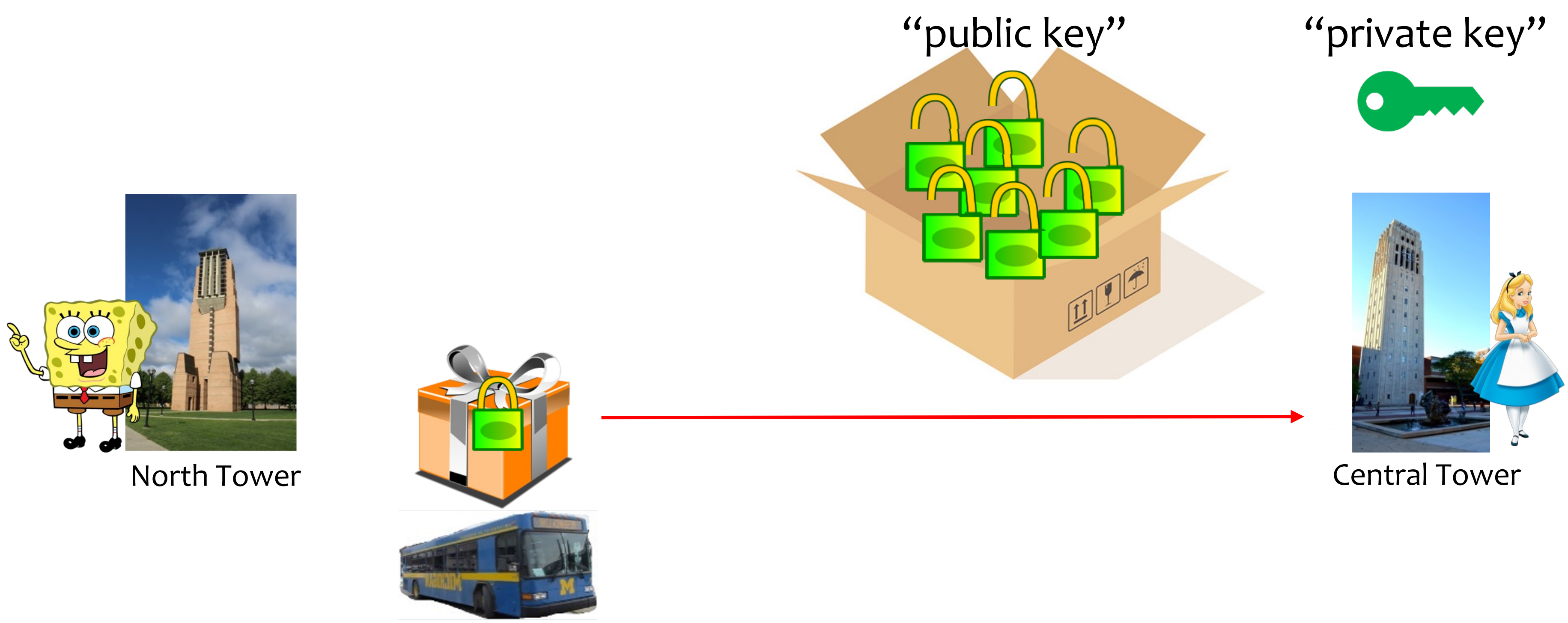


- **Today:** public-key encryption



Analogy

- Central Emperor leaves a box of **open locks out on the street.**
- North Emperor uses one of them to **lock the gift and send it.**



Formal Goal

Create a poly-time encryption algorithm **E** (using **public key**).
Create a poly-time decryption algorithm **D** (using **secret key**).

> **The public key** ≈
> an open lock on the street
>
> **The secret key** ≈
> the Emperor's key to that lock.

Requirement: For any message **m**, **D**(**E**(**m**)) = **m**.
i.e. **D** is the (left) inverse of **E**.

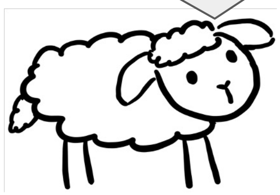
Approach: Find a function **E** that is

- easy to evaluate,
- hard to invert, but…
- easy to invert with a "trapdoor" (**secret key**).

One famous implementation of public-key encryption is **RSA**

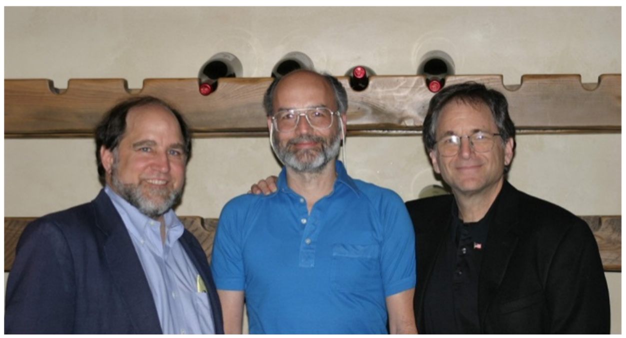
Preview of RSA

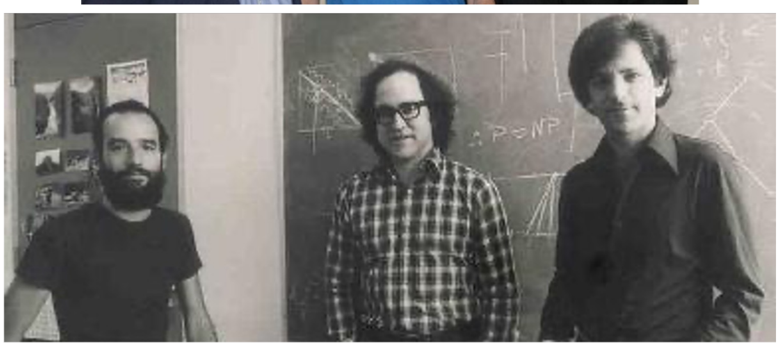
Every time you use a url with "https" you are using RSA!



RSA

First* public-key encryption
First digital signature scheme



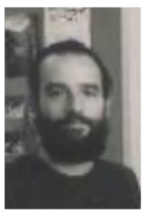


Adi Shamir Ron Rivest Len Adleman (1977)

Also discovered by Clifford Cocks at British Intelligence in 1973; classified until 1997.

*Diffie-Hellman can also be made into PKE but people didn't realize it at the time (see HW)

RSA



Here’s an idea →

← That doesn’t work, try again

Here’s another idea →

← That still doesn’t work, try again

⋮

Apparently this happened 42 times

We believe that RSA is secure because
we believe that it is hard to **factor numbers**

Preview of the Structure of RSA

One-time pad

key = **k**

E is "add **k** (mod 26)"

D is "subtract **k** (mod 26)"

If you know **E**, **D** is easy to find (can invert **E**)

RSA

public key = **(n, e)**, secret key = **d**

E is "take e^{th} power (mod **n**)"

D is "take d^{th} power (mod **n**)"

We'll carefully choose **e**, **n**, **d** so that **E**, **D** are inverses, i.e. $m^{ed} \equiv m \pmod{n}$

Even if you know **E**, **D** is hard to find (cannot invert **E**)

To find such **e**, **n**, **d**, **Fermat's Little Theorem** will be useful…

Fermat's Little Theorem
and
First Wrong Attempt

Fermat's Little Theorem (1640)

Fermat's Little Theorem (FLT):

If p is prime, then for any $a, k \in \mathbb{Z}$,

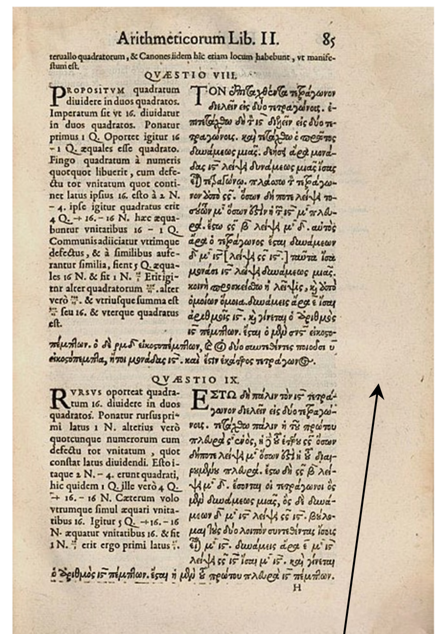
$$a^{\underbrace{1+k(p-1)}_{\text{any number} \equiv 1 \text{ (mod p-1)}}} \equiv a \pmod{p}.$$

E.g. $3^{13} \equiv 3 \pmod{7}$

*proof on last slide and in course notes

(Not to be confused with Fermat's Last Theorem)

"No positive integer solution for $x^n + y^n = z^n$ for any $n \geq 3$"



"It is impossible…for any number which is a power greater than the second to be written as the sum of two like powers. I have a truly marvelous demonstration of this proposition which this margin is too narrow to contain."

Public-Key Encryption: Attempt #1

(Supposedly) secret information is **bold**

I pick:

1. large random prime p
2. e, $\mathbf{d}$ so that gcd(e, p-1) =1 and, $e \cdot \mathbf{d} \equiv 1 \pmod{p-1}$.
   i.e. $e \cdot \mathbf{d} = 1 + k(p-1)$

Announce (p, e)

I compute
$c = \boldsymbol{m}^e \pmod{p}$

$$c = \boldsymbol{m}^e \pmod{p}$$

$\mathbf{d}$ is supposed to be secret.
But I can compute it!
How?

A Bob wants to send Alice a message $\boldsymbol{m} < p$

I decrypt by taking
$c^{\mathbf{d}} \pmod{p}$,
which I claim is $\boldsymbol{m}$.

Why? $c^{\boldsymbol{d}} \bmod p = \boldsymbol{m}^{e\boldsymbol{d}} \bmod p = \boldsymbol{m}^{1+k(p-1)} \bmod p = \boldsymbol{m}$

The Issue with Attempt #1

Eve could calculate d from (p, e) by solving $e \cdot d \equiv 1 \pmod{p-1}$ because the modulus p-1 was **public!**

We would like this modulus to be private to Alice.

Then Alice could still compute d from e and the **private modulus**, but Eve couldn't!

An extension of Fermat's Little Theorem will help…

Euler's Theorem
and
RSA Public Key Encryption

Fermat's Little Theorem

Fermat's Little Theorem: Let p be a prime number. Let a be a positive integer that is coprime to p, then:

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

Euler’s Theorem

Euler's theorem: If *n* and *a* are coprime positive integers, then

$$a^{\varphi(n)} \equiv 1 \, mod \, (n)$$

where φ denotes the Euler’s totient function.

Euler’s totient function counts the positive integers up to a given integer *n* that are relatively prime to *n*, i.e.,

$$\varphi(n) = n \prod_{p|n} (1 - \frac{1}{p})$$

where the product is over the distinct prime numbers dividing *n*.

Euler's Theorem (1763)

(A special case of) Euler's Theorem:

If n = $p.\,q$ is the product of two distinct primes,
then for any $a,\, k \in \mathbb{Z}$ such that a is coprime of n:

$$a^{\underbrace{1+k(p-1)(q-1)}_{\text{any number} \equiv 1 \text{ (mod } (p-1)(q-1))}} \equiv a \text{ (mod n)}$$

E.g.

we have $4^{13} \equiv 4 \text{ (mod 10)}$

because p = 2, q=5, so (p-1)(q-1)=4. We have k = 3.

*proof in course notes

RSA: Public-Key Encryption

I pick:

1. large random primes $\boldsymbol{p}$, $\boldsymbol{q}$.
2. Set $n = \boldsymbol{p} \cdot \boldsymbol{q}$.
3. e coprime to $(\boldsymbol{p}-1)(\boldsymbol{q}-1)$, and pick $\boldsymbol{d}$ so that $e \cdot \boldsymbol{d} \equiv 1 \pmod{(\boldsymbol{p}-1)(\boldsymbol{q}-1)}$.

Secret information is **bold**

I compute $c = \boldsymbol{m}^e \pmod{n}$

(n, e)

$c = \boldsymbol{m}^e \pmod{n}$

Bob wants to send Alice a message $\boldsymbol{m} < n$

I decrypt by taking $c^{\boldsymbol{d}} \pmod{n}$, which I claim is $\boldsymbol{m}$.

I don't how to compute $\boldsymbol{d}$. It seems I need **the factors** $\boldsymbol{p}$ and $\boldsymbol{q}$ of n

Why? $c^d \bmod n = \boldsymbol{m}^{e\boldsymbol{d}} \bmod n = \boldsymbol{m}^{1+k(\boldsymbol{p}-1)(\boldsymbol{q}-1)} \bmod n = \boldsymbol{m}$

RSA: Example

I pick:

1. primes $\boldsymbol{p = 3}$, $\boldsymbol{q = 17}$.
2. Set $n = \boxed{?}$
3. e coprime to $\boxed{?}$ and pick $\boldsymbol{d}$ so that $e \cdot \boldsymbol{d} \equiv 1 \pmod{\boxed{?}}$.
   Set $e = 3$. So $\boldsymbol{d} = \boxed{?}$

Secret information is **bold**

I compute

$c = \boxed{\quad ? \quad}$

(n, e)

c

I decrypt by taking

$\boxed{\quad ? \quad}$

I don't how to compute $\boldsymbol{d}$.
It seems I need **the factors** $\boldsymbol{p}$ and $\boldsymbol{q}$ of n

A Bob wants to send Alice a message $\boldsymbol{m=4}$

Security of
RSA Public Key Encryption

Another "Hard Problem": Factoring

Factoring Assumption:

Given a number n = pq where p and q are randomly chosen secret primes, there is no *efficient* algorithm for finding p,q.

The best known attack for RSA is to solve **factoring.**

(Recall that the best known attack for Diffie-Hellman is to solve **discrete log**.)

Factoring (like discrete log):

- is conjectured to be NP-intermediate.
- has a polynomial-time *quantum* algorithm [Shor, 1994]

RSA Factoring Challenge (193 digits)

In 2005, J. Franke et al. **won $20,000 for showing:**

n=3107418240490043721350750035888567930037346022842727545720161488232064405808150455634682967172328678243791627283803341547107310850191954852900733772482278352574238645401469173660247765234660 9

is the product of

p=16347336458092538484431338838650908598417836700330923121811108523893331001045081512121181675115 79

and

q=19008712816648221131268515739354139754718967899685154936666385390880271038021044989571912614655 71

RSA Factoring Challenge (309 digits)

RSA $100,000 challenge (defunct): factor n into two large primes:

n=135066410865995223349603216278805969938881475605667027524
4851438515265106048595338339402871505719094417982072821644715
513736804197039641917430464965892742562393410208643832021103
729587257623585096431105640735015081875106765946292055636855
294752135008528794163773285339061097505443349998111500569772
36890927563

Real reason for RSA Security

Formally, why can't Eve figure out m?

Well… because we assume that.

RSA assumption: For randomly chosen m,
there is no *efficient* algorithm that given $\underbrace{n, e, m^e \pmod{n}}_{\text{what Eve knows}}$, finds m.

Best known attack: Factor n to find $p, q,$ then what?

- find d where $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$ using Extended Euclid.
- Compute $m^{ed} \bmod n$

One-Time Pad Cons from last time

We didn't fix this one

- **Con 1:** It's insecure to use the same key twice.
- **Con 2:** Alice and Bob must privately agree on the secret key beforehand.

Diffie-Hellman and RSA fix this one

The RSA protocol you just saw suffers from **Con 1** because the encryption algorithm is **deterministic**.
E.g. If two Bobs send the same message, Eve can tell that!

This can be fixed by inserting some **randomization** into the encryption algorithm. (We won't show.)

Efficiency of RSA

Recall: Modular Multiplicative Inverse in Polynomial Time

Input: Integers $m, x < m$ with $\gcd(x, m) = 1$
Output: $x^{-1} \bmod m$, i.e.,

$$z \text{ where } z \cdot x \equiv 1 \bmod m$$

Algorithm:

1. Run **ExtendedEuclid**(x, m) to find (a, b) such that 1 = ax+bm.
2. Return a.

Correctness: 1 ≡ ax+bm (mod m), so 1 ≡ a·x (mod m).

Running time: poly(log(m))

ExtendedEuclid (from HW 2):
Input: Integers x ≥ y ≥ 0, not both 0
Output: Triple (g,a,b) of integers where g = gcd(x,y) = ax+by.

RSA: Public-Key Encryption

I pick:

1. large random primes p, q.
2. Set $n = p \cdot q$.
3. e coprime to $(p-1)(q-1)$, and pick d so that $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.

(n, e)

$c = m^e \pmod{n}$

I compute
$c = m^e \pmod{n}$

A Bob wants to send Alice a message $m < n$

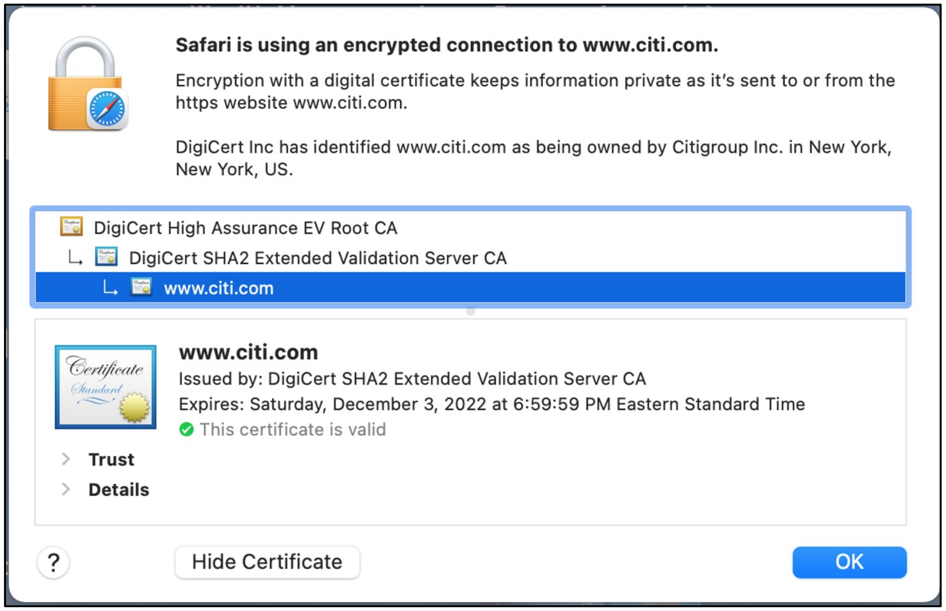
I decrypt by taking
$c^d \pmod{n}$.

Using RSA for Signature

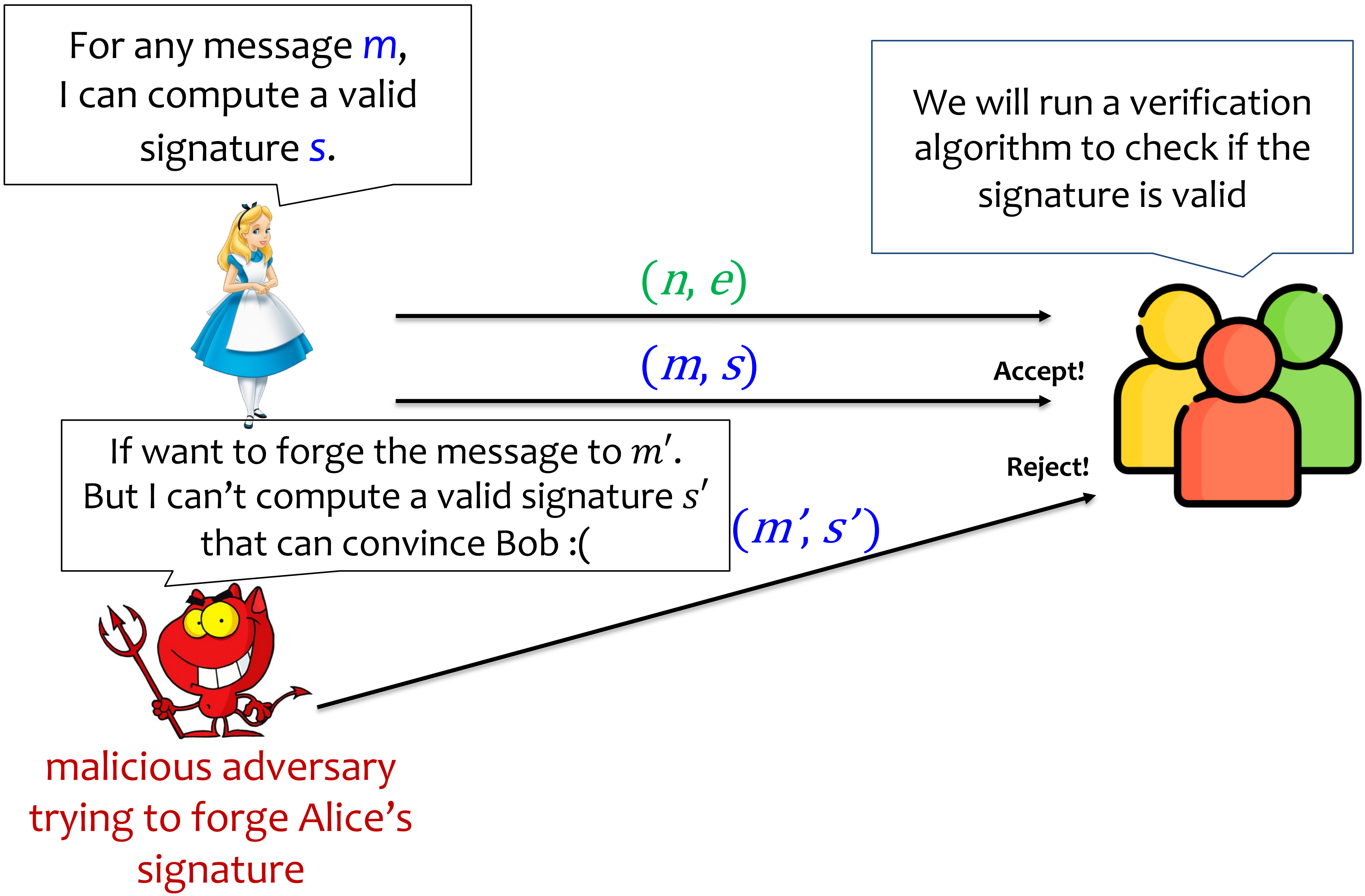
RSA can also be used for "Signatures"

Goal of a signature:

1. Alice publishes a public key (n, e).
2. Alice or a malicious "forger" sends
   a **public message m** with a **signature s**
3. Anyone can check whether the same entity did both 1 and 2.



RSA Signature Goal



RSA Signatures: Run RSA "Backwards"

I pick:

1. large random primes ***p***, ***q***.
2. Set n = ***p*** $\cdot$ ***q***.
3. e coprime to $(\boldsymbol{p}-1)(\boldsymbol{q}-1)$, and pick ***d*** so that $e \cdot \boldsymbol{d} \equiv 1 \pmod{(\boldsymbol{p}-1)(\boldsymbol{q}-1)}$.

I want to sent **public message** m.

I compute signature $s = m^{\boldsymbol{d}} \pmod{n}$

Secret information is **bold**

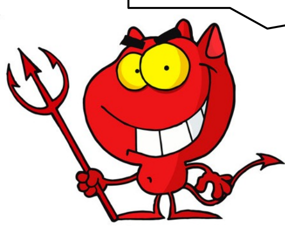
Verification algorithm: check that $s^e \equiv m \pmod{n}$

(n, e)

$(m, s = m^{\boldsymbol{d}} \pmod{n})$

I can't seem to compute a valid signature.
I don't know ***d***

(m', s')



RSA assumption: For randomly chosen m, there is no *efficient* algorithm that given n, e, $\underbrace{m^e \pmod{n}}_{m'}$, finds $\underbrace{m}_{m'^{\boldsymbol{d}} \pmod{n}}$.

What if the forger gets to choose m' based on s'?

Then the forger can successfully forge Alice's signature by first picking s' and then letting $m' = s'^{e} \pmod{n}$!

This can be fixed by Alice applying some wild function H to m and sending $H(m)$, $s = H(m)^{d} \ (mod\ n)$ (We won't show.)

Proof of Fermat's Little Theorem

Theorem: For any prime p and $0 < a < p$, $a^{p-1} \equiv 1 \pmod{p}$

Lemma: $\{a, 2a, 3a, \dots, (p-1)a\} \pmod{p} = \{1, \dots, p-1\}$.

For every $i \in \{1, \dots, p-1\}$,

- ia is not a multiple of p since $\gcd(i, p) = \gcd(a, p) = 1$.
- So, each $ia \not\equiv 0 \pmod{p}$.

For every $i, j \in \{1, \dots, p-1\}$, $i \neq j$,

- $(j-i)a$ is not a multiple of p.
- So, there are no collisions: $ia \not\equiv ja \pmod{p}$.

Then:

$$a \cdot 2a \cdots (p-1)a \equiv 1 \cdot 2 \cdots (p-1) \pmod{p}$$

Since $\{1, \dots, p-1\}$ all have inverses mod p,

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