MAT246H1-S – LEC0201/9201 Concepts in Abstract Mathematics

## THE RSA ALGORITHM



#### March 4<sup>th</sup>, 2021

How can someone send a secret message in a way that only the recipient could read the content even if the message happens to have been intercepted by a third party?

## Cæsar cipher (monoalphabetic)

**Encryption:** perform a right shift w.r.t. the key **Decryption:** perform a left shift w.r.t. the key



LEON BATTISTA ALBERTI'S cipher disk.

#### Implementation in Julia:

```
function caesar(c::Char, key::Integer)
       (\text{kev} \ge 1 \&\& \text{kev} \le 25) \parallel \text{error}("The key must be between 1 and 25")
       if isletter(c)
          shift = ifelse(islowercase(c), 'a', 'A')
          return (c - shift + kev) % 26 + shift
       and
       return c
    end
   caesar enc(m::AbstractString, kev::Integer) = map(c \rightarrow caesar(c, kev), m)
   caesar dec (m::AbstractString, key::Integer) = map(c -> caesar(c, 26-key), m)
   m = "There is no permanent place in the world for ugly mathematics."
   kev = 6
   println("Original message: $m")
16 c = caesar enc(m, kev)
   println("Encrypted message: $c")
18 println("Decrypted message: $(caesar dec(c,key))")
```

#### [mat246@Pavilion mat246]\$ julia caesar.j

Original message: There is no permanent place in the world for ugly mathematics. Encrypted message: Znkxk oy tu vkxsgtktz vrgik ot znk cuxrj lux amre sgznksgzoly. Decrypted message: There is no permanent place in the world for ugly mathematics.

Weaknesses: bruteforce (only 25 keys), frequency distribution of the letters.

It is an improved version of Cæsar where the shift value changes depending on the position in the text (published in 1553 by GIOVAN BATTISTA BELLASO<sup>1</sup>).

For instance, if the key is MATH, then

- the first letter is shifted by 12,
- 2 the second one by 0,
- 3 the third one by 19,
- 4 the fourth one by 7,
- 5 and then we repeat...

This way Vigenère cipher is robust against frequency distribution of the letters. An efficient attack, when the key is (far) shorter than the message, consists in looking for repetitions in the crypted message (to deduce the length of the key).

<sup>&</sup>lt;sup>1</sup>You will note the great given name.

### Vigenère cipher (polyalphabetic) – 2

Implementation in Julia:

```
function vigenere(m, key::AbstractString, enc::Bool)
      occursin(r"^[a-zA-Z]+$",key) || error("The key can only contain letters")
      kev = lowercase(kev)
      s =
      i = 0
6
      for c in m
         if isletter(c)
8
            shiftcap = islowercase(c) ? 'a' : 'A'
            shiftkey = enc ? key[i%length(key)+1]-'a' : 26-(key[i%length(key)+1]-'a')
            s = s*((c-shiftcap+shiftkey) %26+shiftcap)
            i += 1
         else
            s = s^{*}c
         end
      end
      return s
  end
19 m = "There is no permanent place in the world for ugly mathematics."
20 kev = "MATHEMATICS"
21 println("Original message: $m")
22 c = vigenere(m, key, true)
23 println("Encrypted message: $c")
24 println("Decrypted message: $(vigenere(c,key,false))")
   [mat246@Pavilion mat246]$ julia vigenere.j
   Original message: There is no permanent place in the world for ugly mathematics.
   Encrypted message: Fhxyi us gw rwdmtuizt itcug ig alg whznv rok bkxy fivzgmtamos.
   Decrypted message: There is no permanent place in the world for ugly mathematics.
```

# Enigma (WWII) - 1



Middle Right Left Reflector Rotor Rotor Rotor **Right** rotor advanced one position

https://en.wikipedia.org/wiki/File:Enigma-action.svg

https://commons.wikimedia.org/wiki/File: Enigma\_(crittografia)\_-\_Museo\_scienza\_e\_tecnologia\_Milano.jpg Emulator.https://pi

# Enigma (WWII) – 2

#### **Configuration:**

- You have to pick 3 rotors among 5:  $\frac{5!}{(5-3)!} = 60$ .
- You have to pick the initial position of each rotor:  $26 \times 26 \times 26 = 17576$ .
- Plugboard for  $\ell$  links:  $\frac{26!}{(26-2\ell)!\ell!2^{\ell}}$ .
- In some versions, there are several possible reflector configurations.

Pictures: https://www.cryptomuseum.com/crypto/enigma/working.htm

# Enigma (WWII) – 2

#### **Configuration:**

- You have to pick 3 rotors among 5:  $\frac{5!}{(5-3)!} = 60$ .
- You have to pick the initial position of each rotor:  $26 \times 26 \times 26 = 17576$ .
- Plugboard for  $\ell$  links:  $\frac{26!}{(26-2\ell)!\ell!2^{\ell}}$ .
- In some versions, there are several possible reflector configurations.

#### Strengths:

- There are lots of configurations.
- The rotors move when a key is pressed, thus the Enigma machine is robust against frequency distribution of the letters.

Pictures: https://www.cryptomuseum.com/crypto/enigma/working.htm

# Enigma (WWII) – 2

#### **Configuration:**

- You have to pick 3 rotors among 5:  $\frac{5!}{(5-3)!} = 60$ .
- You have to pick the initial position of each rotor:  $26 \times 26 \times 26 = 17576$ .
- Plugboard for  $\ell$  links:  $\frac{26!}{(26-2\ell)!\ell!2^{\ell}}$ .
- In some versions, there are several possible reflector configurations.

#### Strengths:

- There are lots of configurations.
- The rotors move when a key is pressed, thus the Enigma machine is robust against frequency distribution of the letters.

#### Weaknesses:

- Because of how the reflector is built, a key is never mapped to itself.
- Symmetry of the plugboard.
- Regularity of the stepping: every 26 steps, each rotor induces a stepping to the rotor on its left.
- The position at which a rotor induces a stepping to the next one is fixed (there is a notch on the rotor).

Pictures: https://www.cryptomuseum.com/crypto/enigma/working.htm

In the above examples, there is a common key shared among the participants. Either you share a common key with all the participants (but it increases the risk for the key to be compromised) or each pair of participants has a different key (so there are lots of keys). Big weakness: how to communicate safely the key? In the above examples, there is a common key shared among the participants. Either you share a common key with all the participants (but it increases the risk for the key to be compromised) or each pair of participants has a different key (so there are lots of keys). Big weakness: how to communicate safely the key?

#### Asymmetric cryptographic algorithms:



### Asymmetric ciphers – 2



- One private key: only known by the recipient to decrypt messages.
- One public key: to be used by the sender to encrypt the message.
- The knowledge of the public key is not enough to decrypt message, so it can be widely shared.
- Theoretical approach: mid 70s.
- First concrete algorithm: RSA in Ron Rivest, Adi Shamir, and Leonard Adleman in 1978.
- Actually developped in 1973 by the British secret service (classified until the 90s).
- Asymmetric ciphers can still be vulnerable to frequency distribution.
- In practice, asymmetric ciphers are usually used for authentification purposes or to initialize communications allowing to safely exchange keys that will be used with a symmetric cipher.

### RSA: once upon a time...

#### **Protagonists:**



**Goal:** Find a way for Bob to send a secret message to Alice so that only her can read it, even if the message happens to be intercepted by a third party. Hopefully, Alice and Bob attended MAT246 and understand very well modular arithmetic.



Alice picks two distinct prime numbers p and q.



Alice picks two distinct prime numbers p and q. She sets  $n \coloneqq pq$ .



Alice picks two distinct prime numbers *p* and *q*. She sets  $n \coloneqq pq$ . She chooses  $e \in \mathbb{N}$  such that  $gcd(e, \varphi(n)) = 1$ .



Alice picks two distinct prime numbers p and q. She sets  $n \coloneqq pq$ . She chooses  $e \in \mathbb{N}$  such that  $gcd(e, \varphi(n)) = 1$ . **The public key is** (n, e). She send this key to Bob via e-mail.



Alice picks two distinct prime numbers p and q. She sets  $n \coloneqq pq$ . She chooses  $e \in \mathbb{N}$  such that  $gcd(e, \varphi(n)) = 1$ . **The public key is** (n, e). She send this key to Bob via e-mail.

Since  $gcd(e, \varphi(n)) = 1$ ,  $\exists d \in \mathbb{N}$ ,  $ed \equiv 1 \pmod{\varphi(n)}$ .



Alice picks two distinct prime numbers p and q. She sets  $n \coloneqq pq$ . She chooses  $e \in \mathbb{N}$  such that  $gcd(e, \varphi(n)) = 1$ . **The public key is** (n, e). She send this key to Bob via e-mail.

Since  $gcd(e, \varphi(n)) = 1$ ,  $\exists d \in \mathbb{N}$ ,  $ed \equiv 1 \pmod{\varphi(n)}$ . Alice can easily find such a *d* from a Bézout's relation using Euclid's algorithm:  $\exists u, v \in \mathbb{Z}$ ,  $eu + \varphi(n)v = 1$ . Then she picks  $d = u + k\varphi(n)$  for a suitable  $k \in \mathbb{Z}$  so that d > 0.



Alice picks two distinct prime numbers p and q. She sets  $n \coloneqq pq$ . She chooses  $e \in \mathbb{N}$  such that  $gcd(e, \varphi(n)) = 1$ . **The public key is** (n, e). She send this key to Bob via e-mail.

Since  $gcd(e, \varphi(n)) = 1$ ,  $\exists d \in \mathbb{N}$ ,  $ed \equiv 1 \pmod{\varphi(n)}$ . Alice can easily find such a *d* from a Bézout's relation using Euclid's algorithm:  $\exists u, v \in \mathbb{Z}$ ,  $eu + \varphi(n)v = 1$ . Then she picks  $d = u + k\varphi(n)$  for a suitable  $k \in \mathbb{Z}$  so that d > 0. **The private key is** (n, d). *She keeps this key for her and only her.* 



Alice picks two distinct prime numbers p and q. She sets  $n \coloneqq pq$ . She chooses  $e \in \mathbb{N}$  such that  $gcd(e, \varphi(n)) = 1$ . **The public key is** (n, e). She send this key to Bob via e-mail.

Since  $gcd(e, \varphi(n)) = 1$ ,  $\exists d \in \mathbb{N}$ ,  $ed \equiv 1 \pmod{\varphi(n)}$ . Alice can easily find such a *d* from a Bézout's relation using Euclid's algorithm:  $\exists u, v \in \mathbb{Z}$ ,  $eu + \varphi(n)v = 1$ . Then she picks  $d = u + k\varphi(n)$  for a suitable  $k \in \mathbb{Z}$  so that d > 0. **The private key is** (n, d). *She keeps this key for her and only her.* 

Note that Alice knows *p* and *q*, so she can easily computes  $\varphi(n) = (p-1)(q-1)$ .



Alice picks two distinct prime numbers p and q. She sets  $n \coloneqq pq$ . She chooses  $e \in \mathbb{N}$  such that  $gcd(e, \varphi(n)) = 1$ . **The public key is** (n, e). She send this key to Bob via e-mail.

Since  $gcd(e, \varphi(n)) = 1$ ,  $\exists d \in \mathbb{N}$ ,  $ed \equiv 1 \pmod{\varphi(n)}$ . Alice can easily find such a *d* from a Bézout's relation using Euclid's algorithm:  $\exists u, v \in \mathbb{Z}$ ,  $eu + \varphi(n)v = 1$ . Then she picks  $d = u + k\varphi(n)$  for a suitable  $k \in \mathbb{Z}$  so that d > 0. **The private key is** (n, d). *She keeps this key for her and only her.* 

Note that Alice knows *p* and *q*, so she can easily computes  $\varphi(n) = (p-1)(q-1)$ . There is no efficient algorithm to compute  $\varphi(n)$  directly from *n*, that's why it is difficult to recover the private key from the public key.

If someone finds an efficient prime factorization algorithm then RSA is no longer safe.

Note that p and q should be wisely chosen.

• They need to be large enough so that we can't recover them from *n* using our current computing power.

Note that *p* and *q* should be wisely chosen.

- They need to be large enough so that we can't recover them from *n* using our current computing power.
- But that's not enough: for instance,  $\delta = |p q|$  should be large too.

Note that *p* and *q* should be wisely chosen.

- They need to be large enough so that we can't recover them from *n* using our current computing power.
- But that's not enough: for instance,  $\delta = |p q|$  should be large too. Indeed, assume that p < q then  $q = p + \delta$ .

Thus 
$$\sqrt{n} = p\sqrt{1 + \frac{\delta}{p}} \sim p + \frac{\delta}{2}$$
.

Hence, according to Proposition 3 of Chapter 3, it is enough to check whether each number less than  $\sqrt{n}$  divides *n*, and from the above estimation, *p* could be obtained after less than  $\frac{\delta}{2}$  attempts starting from  $\sqrt{n}$ .

Note that p and q should be wisely chosen.

- They need to be large enough so that we can't recover them from *n* using our current computing power.
- But that's not enough: for instance, δ = |p − q| should be large too. Indeed, assume that p < q then q = p + δ. Thus √n = p√1 + δ/p ~ p + δ/2. Hence, according to Proposition 3 of Chapter 3, it is enough to check whether each number less than √n divides n, and from the above estimation, p could be obtained after less than δ/2 attempts starting from √n.
- They need to satisfy additional properties to avoid known attacks.



Bob knows the public key (n, e).



Bob knows the public key (n, e). A message is an element  $m \in \{0, 1, ..., n - 1\}$ .



Bob knows the public key (n, e). A message is an element  $m \in \{0, 1, ..., n - 1\}$ . There exists a unique  $c \in \{0, 1, ..., n - 1\}$  such that

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



Bob knows the public key (n, e). A message is an element  $m \in \{0, 1, ..., n - 1\}$ . There exists a unique  $c \in \{0, 1, ..., n - 1\}$  such that

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

It is the crypted message that Bob sends to Alice.



Alice received the secret message c from Bob. She knows her private key (n, d).



Alice received the secret message *c* from Bob. She knows her private key (n, d). There exists a unique  $k \in \{0, 1, ..., n - 1\}$  such that

 $k \equiv c^d \pmod{n}$ 



Alice received the secret message *c* from Bob. She knows her private key (n, d). There exists a unique  $k \in \{0, 1, ..., n-1\}$  such that

 $k \equiv c^d \pmod{n}$ 

k is the original message: since  $ed = 1 + l\varphi(n)$  for some  $l \in \mathbb{N}$ , we obtain using Euler's theorem that

 $k \equiv c^d \pmod{n} \equiv m^{ed} \pmod{n} = m^{1+l\varphi(n)} \pmod{n} \equiv m \times \left(m^{\varphi(n)}\right)^l \pmod{n} \equiv m \times 1^l \pmod{n} \equiv m \pmod{n}$ 

We conclude since k has a unique representative in  $\{0, 1, \dots, n-1\}$  and  $m, k \in \{0, 1, \dots, n-1\}$ .



Alice received the secret message *c* from Bob. She knows her private key (n, d). There exists a unique  $k \in \{0, 1, ..., n-1\}$  such that

 $k \equiv c^d \pmod{n}$ 

k is the original message: since  $ed = 1 + l\varphi(n)$  for some  $l \in \mathbb{N}$ , we obtain using Euler's theorem that

 $k \equiv c^{d} \pmod{n} \equiv m^{ed} \pmod{n} = m^{1+l\varphi(n)} \pmod{n} \equiv m \times \left(m^{\varphi(n)}\right)^{l} \pmod{n} \equiv m \times 1^{l} \pmod{n} \equiv m \pmod{n}$ 

We conclude since k has a unique representative in  $\{0, 1, \dots, n-1\}$  and  $m, k \in \{0, 1, \dots, n-1\}$ .

The above proof is not (entirely) correct, why?



Alice received the secret message *c* from Bob. She knows her private key (n, d). There exists a unique  $k \in \{0, 1, ..., n-1\}$  such that

 $k \equiv c^d \pmod{n}$ 

k is the original message: since  $ed = 1 + l\varphi(n)$  for some  $l \in \mathbb{N}$ , we obtain using Euler's theorem that

$$k \equiv c^d \pmod{n} \equiv m^{e^d} \pmod{n} = m^{1+l\varphi(n)} \pmod{n} \equiv m \times \left(m^{\varphi(n)}\right)^l \pmod{n} \equiv m \times 1^l \pmod{n} \equiv m \pmod{n}$$

We conclude since k has a unique representative in  $\{0, 1, \dots, n-1\}$  and  $m, k \in \{0, 1, \dots, n-1\}$ .

The above proof is not (entirely) correct, why? You will fix it in the tutorial questions.

Alice picks the prime numbers p = 13 and q = 17 then n = 221 and  $\varphi(n) = 12 \times 16 = 192$ .

Alice picks the prime numbers p = 13 and q = 17 then n = 221 and  $\varphi(n) = 12 \times 16 = 192$ . Then she picks e = 11, which is a suitable choice since gcd(192, 11) = 1.

Alice picks the prime numbers p = 13 and q = 17 then n = 221 and  $\varphi(n) = 12 \times 16 = 192$ . Then she picks e = 11, which is a suitable choice since gcd(192, 11) = 1.

Using Euclid's algorithm, Alice obtains the Bézout relation  $192 \times (-2) + 11 \times (35) = 1$ . Therefore, she sets d = 35 so that  $ed \equiv 1 \pmod{192}$ . The public key is (n, e) = (221, 11). The private key is (n, d) = (221, 35).

Alice picks the prime numbers p = 13 and q = 17 then n = 221 and  $\varphi(n) = 12 \times 16 = 192$ . Then she picks e = 11, which is a suitable choice since gcd(192, 11) = 1.

Using Euclid's algorithm, Alice obtains the Bézout relation  $192 \times (-2) + 11 \times (35) = 1$ . Therefore, she sets d = 35 so that  $ed \equiv 1 \pmod{192}$ . The public key is (n, e) = (221, 11). The private key is (n, d) = (221, 35).

Later, Bob wants to send the private message  $m = 149 \in \{0, 1, 2, \dots, 220\}$  to Alice.

Alice picks the prime numbers p = 13 and q = 17 then n = 221 and  $\varphi(n) = 12 \times 16 = 192$ . Then she picks e = 11, which is a suitable choice since gcd(192, 11) = 1.

Using Euclid's algorithm, Alice obtains the Bézout relation  $192 \times (-2) + 11 \times (35) = 1$ . Therefore, she sets d = 35 so that  $ed \equiv 1 \pmod{192}$ . The public key is (n, e) = (221, 11). The private key is (n, d) = (221, 35).

Later, Bob wants to send the private message  $m = 149 \in \{0, 1, 2, ..., 220\}$  to Alice. He computes  $m^e = 149^{11} \equiv 89 \pmod{221}$ . So the encrypted message is  $c = 89 \in \{0, 1, 2, ..., 220\}$ . He sends it to Alice by e-mail.

Alice picks the prime numbers p = 13 and q = 17 then n = 221 and  $\varphi(n) = 12 \times 16 = 192$ . Then she picks e = 11, which is a suitable choice since gcd(192, 11) = 1.

Using Euclid's algorithm, Alice obtains the Bézout relation  $192 \times (-2) + 11 \times (35) = 1$ . Therefore, she sets d = 35 so that  $ed \equiv 1 \pmod{192}$ . The public key is (n, e) = (221, 11). The private key is (n, d) = (221, 35).

Later, Bob wants to send the private message  $m = 149 \in \{0, 1, 2, ..., 220\}$  to Alice. He computes  $m^e = 149^{11} \equiv 89 \pmod{221}$ . So the encrypted message is  $c = 89 \in \{0, 1, 2, ..., 220\}$ . He sends it to Alice by e-mail.

After receiving the e-mail, Alice computes  $c^d = 89^{35} \equiv 149 \pmod{221}$  to recover the original message m = 149.

We usually generate prime numbers as follows (it is a little bit tricky):

- **1** Generate a random odd number k of the wanted order of magnitude
- 2 Check whether it is prime or not.
- **3** If not, we repeat (2) with  $k \leftarrow k + 2$ .

We usually generate prime numbers as follows (it is a little bit tricky):

- **()** Generate a random odd number k of the wanted order of magnitude
- 2 Check whether it is prime or not.
- **3** If not, we repeat (2) with  $k \leftarrow k + 2$ .

According to the prime number theorem, for *k* large enough, there are *about*  $\frac{k}{\ln(k)}$  prime numbers less than or equal to *k*, so we could expect a prime number before  $\frac{\ln(k)}{2}$  attempts.

We usually generate prime numbers as follows (it is a little bit tricky):

- **()** Generate a random odd number k of the wanted order of magnitude
- 2 Check whether it is prime or not.
- **3** If not, we repeat (2) with  $k \leftarrow k + 2$ .

According to the prime number theorem, for *k* large enough, there are *about*  $\frac{k}{\ln(k)}$  prime numbers less than or equal to *k*, so we could expect a prime number before  $\frac{\ln(k)}{2}$  attempts.

Nonetheless, we don't know efficient algorithms to check whether a number is prime or not. Instead, we usually use probabilistic primality tests.

We usually generate prime numbers as follows (it is a little bit tricky):

- **()** Generate a random odd number k of the wanted order of magnitude
- 2 Check whether it is prime or not.
- **3** If not, we repeat (2) with  $k \leftarrow k + 2$ .

According to the prime number theorem, for *k* large enough, there are *about*  $\frac{k}{\ln(k)}$  prime numbers less than or equal to *k*, so we could expect a prime number before  $\frac{\ln(k)}{2}$  attempts.

Nonetheless, we don't know efficient algorithms to check whether a number is prime or not. Instead, we usually use probabilistic primality tests.

For example, we know from Fermat's little theorem that if p is prime then

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

We usually generate prime numbers as follows (it is a little bit tricky):

- **()** Generate a random odd number k of the wanted order of magnitude
- 2 Check whether it is prime or not.
- **3** If not, we repeat (2) with  $k \leftarrow k + 2$ .

According to the prime number theorem, for *k* large enough, there are *about*  $\frac{k}{\ln(k)}$  prime numbers less than or equal to *k*, so we could expect a prime number before  $\frac{\ln(k)}{2}$  attempts.

Nonetheless, we don't know efficient algorithms to check whether a number is prime or not. Instead, we usually use probabilistic primality tests.

For example, we know from Fermat's little theorem that if p is prime then

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

Therefore, since  $24^{221} \equiv 176 \pmod{221}$ , we know that 221 is not prime.

We usually generate prime numbers as follows (it is a little bit tricky):

- **()** Generate a random odd number k of the wanted order of magnitude
- 2 Check whether it is prime or not.
- **3** If not, we repeat (2) with  $k \leftarrow k + 2$ .

According to the prime number theorem, for *k* large enough, there are *about*  $\frac{k}{\ln(k)}$  prime numbers less than or equal to *k*, so we could expect a prime number before  $\frac{\ln(k)}{2}$  attempts.

Nonetheless, we don't know efficient algorithms to check whether a number is prime or not. Instead, we usually use probabilistic primality tests.

For example, we know from Fermat's little theorem that if p is prime then

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

Therefore, since  $24^{221} \equiv 176 \pmod{221}$ , we know that 221 is not prime.

But, it is possible for such a congruence to hold even for a non-prime:

$$2^{341} \equiv 2 \pmod{341}$$

although  $341 = 11 \times 31$ .

In the previous example, 149 and 11 are *small* but  $149^{11} = 803616698647447868139149$  is very large.

In the previous example, 149 and 11 are *small* but  $149^{11} = 803616698647447868139149$  is very large.

First, note that we don't need  $m^e$  but only need a representative modulo n,

In the previous example, 149 and 11 are *small* but  $149^{11} = 803616698647447868139149$  is very large.

First, note that we don't need  $m^e$  but only need a representative modulo n,

i.e. given  $m, e, n \in \mathbb{N}$ , we want to find (the unique)  $c \in \{0, 1, \dots, n-1\}$  such that  $m^e \equiv c \pmod{n}$ .

In the previous example, 149 and 11 are *small* but  $149^{11} = 803616698647447868139149$  is very large.

First, note that we don't need  $m^e$  but only need a representative modulo n,

i.e. given  $m, e, n \in \mathbb{N}$ , we want to find (the unique)  $c \in \{0, 1, \dots, n-1\}$  such that  $m^e \equiv c \pmod{n}$ .

Hence we can iteratively multiply by c but reduce to a representative in  $\{0, ..., n\}$  before the next step.

In the previous example, 149 and 11 are *small* but  $149^{11} = 803616698647447868139149$  is very large.

First, note that we don't need  $m^e$  but only need a representative modulo n, i.e. given  $m, e, n \in \mathbb{N}$ , we want to find (the unique)  $c \in \{0, 1, ..., n-1\}$  such that  $m^e \equiv c \pmod{n}$ . Hence we can iteratively multiply by c but reduce to a representative in  $\{0, ..., n\}$  before the next step.

For instance, to compute  $149^{11} \pmod{221}$ , we would do:

**1** 
$$149^1 \equiv 149 \pmod{221}$$

2 
$$149^2 = 149 \times 149 = 22201 \equiv 101 \pmod{221}$$

**3** 
$$149^3 = 101 \times 149 = 15049 \equiv 21 \pmod{221}$$

$$4 \quad 149^4 = 21 \times 149 = 3129 \equiv 35 \pmod{221}$$

**5** 
$$149^5 = 35 \times 149 = 5215 \equiv 132 \pmod{221}$$

$$6 \quad 149^6 = 132 \times 149 = 19668 \equiv 220 \pmod{221}$$

7 
$$149^7 = 220 \times 149 = 32780 \equiv 72 \pmod{221}$$

$$149^8 = 72 \times 149 = 10728 \equiv 120 \pmod{221}$$

9 
$$149^9 = 120 \times 149 = 17880 \equiv 200 \pmod{221}$$

$$149^{10} = 200 \times 149 = 29800 \equiv 186 \pmod{221}$$

(1) 
$$149^{11} = 186 \times 149 = 27714 \equiv 89 \pmod{221}$$

In this example, the largest number we could have obtained is  $149 \times 220 = 32780$ .

We even have more efficient algorithm.

Write the exponent in binary 
$$e = \overline{a_r a_{r-1} \dots a_1 a_0}^2 = \sum_{i=0}^r a_i 2^i$$
 where  $a_i \in \{0, 1\}$ . Then

We even have more efficient algorithm.

Write the exponent in binary 
$$e = \overline{a_r a_{r-1} \dots a_1 a_0}^2 = \sum_{i=0}^r a_i 2^i$$
 where  $a_i \in \{0, 1\}$ . Then  
 $m^e = m^{\sum a_i 2^i} = \prod_{i=0}^r (m^{2^i})^{a_i}$ 

We even have more efficient algorithm.

Write the exponent in binary  $e = \overline{a_r a_{r-1} \dots a_1 a_0}^2 = \sum_{i=0}^r a_i 2^i$  where  $a_i \in \{0, 1\}$ . Then  $m^e = m^{\sum a_i 2^i} = \prod_{i=0}^r (m^{2^i})^{a_i}$ So we just need to compute successive squares:  $m^{2^{i+1}} = (m^{2^i})^2$  (actually we only need it modulo n).

Jean-Baptiste Campesato MAT246H1-S – LEC0201/9201 – Mar 4, 2021

We even have more efficient algorithm.

Write the exponent in binary  $e = \overline{a_r a_{r-1} \dots a_1 a_0}^2 = \sum_{i=0}^{i} a_i 2^i$  where  $a_i \in \{0, 1\}$ . Then

$$m^e = m^{\sum a_i 2^i} = \prod_{i=0}^r \left(m^{2^i}\right)^a$$

So we just need to compute successive squares:  $m^{2^{i+1}} = (m^{2^i})^2$  (actually we only need it modulo *n*).

Implementation in Julia:

```
function fastpowmod(m,e,n::Integer)
     n > 0 || error("n must be positive")
     e >= 0 || error("e must be non-negative")
     r = 1
567
89
     while e > 0
         if (e \& 1) > 0
            r = (r*m) %n
         end
         e >>= 1
        m = (m^2)%n
     end
     return r>0 ? r : r+n
  end
  julia> fastpowmod(149,11,221)
  89
```

```
Basic implementation of RSA in Julia:
   using Primes
   struct PublicKev
       n::Integer
       e::Integer
   end
   struct PrivateKev
       n::Integer
       d::Integer
   end
   function gen_keys(p::Integer, g::Integer, e::Integer)
       isprime(p) || error("p must be a prime number"
       isprime(q) || error("q must be a prime number")
       e>0 || error("e must be positive")
       phi = (p-1)*(q-1)
       (q, u, v) = qcdx(e, phi)
       g == 1 || error("phi(n) and e must be coprime")
       u<0 ? d=(u%phi)+phi : d=u%phi
       n = p * q
       return PublicKev(n, e), PrivateKev(n, d)
   end
   function encrypt(m::Integer, k::PublicKey)
       0 <= m || error("m must be non-negative")
       m < k.n || error("m is too large")</pre>
       return powermod (m, k.e, k.n)
   end
   function decrypt(c::Integer, k::PrivateKey)
       0 <= c || error("c must be non-negative")
       c < k.n || error("c is too large")</pre>
       return powermod(c,k,d,k,n)
   end
    (pbk, pvk) = gen kevs(13, 17, 11)
38 println("The public key is (n,e)=($(pbk,n),$(pbk,e)), give it to people willing to send you a secret message!")
39 println("The private key is (n,d)=($(pvk,n),$(pvk,d)), don't share it!")
40 m = 149
41 println("Original message: $m")
42 c = encrypt(m, pbk)
43 println("Encrypted message: $c")
44 println("Decrypted message: S(decrypt(c,pyk))")
    [mat246@Pavilion mat2461$ julia rsa.j
   The public key is (n,e)=(221,11), give it to people willing to send you a secret message!
    The private key is (n,d)=(221,35), don't share it!
   Original message: 149
    Encrypted message: 89
   Decrypted message: 149
```