## The RSA algorithm

March $4^{\text {th }}, 2021$

## Introduction

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)
    (key >= 1 && key <= 25) || error("The key must be between 1 and 25")
    if isletter(c)
        shift = ifelse(islowercase(c), 'a', 'A')
        return (c - shift + key) % 26 + shift
    end
    return c
end
caesar_enc(m: :AbstractString, key::Integer) = map(c -> caesar(c, key), 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."
key = 6
println("Original message: $m")
c = caesar_enc(m,key)
println("Encrypted message: $c")
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 sgznksgzoiy.
Decrypted message: There is no permanent place in the world for ugly mathematics.

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

## Vigenère cipher (polyalphabetic) - 1

It is an improved version of Cæsar where the shift value changes depending on the position in the text (published in 1553 by Gıovan Battista Bellaso ${ }^{1}$ ).

For instance, if the key is MATH, then
(1) 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).

[^0]
## 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")
    key = lowercase(key)
    s = ""
    i}=
    for c in m
            if isletter(c)
                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
m = "There is no permanent place in the world for ugly mathematics."
key = "MATHEMATICS"
println("Original message: $m")
c = vigenere(m,key,true)
println("Encrypted message: $c")
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 itcuq ig alq whznv rok bkxy fivzqmtamos.
Decrypted message: There is no permanent place in the world for ugly mathematics.
```


## Enigma (WWII) - 1


https://commons.wikimedia.org/wiki/File:

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

## 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: $\frac{26!}{(26-2 e)!!122^{2}}$.
- 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: $\frac{26!}{(26-2 e)!!122^{2}}$.
- 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: $\frac{26!}{(26-2 e)!!1^{2}}$.
- 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

## Asymmetric ciphers - 1

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 ciphers - 1

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.

## RSA: key generation - 1



## Alice picks two distinct prime numbers $p$ and $q$.

## RSA: key generation - 1



Alice picks two distinct prime numbers $p$ and $q$. She sets $n:=p q$.

## RSA: key generation - 1



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

## RSA: key generation - 1



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

## RSA: key generation - 1



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

## RSA: key generation - 1



Alice picks two distinct prime numbers $p$ and $q$.
She sets $n:=p q$.
She chooses $e \in \mathbb{N}$ such that $\operatorname{gcd}(e, \varphi(n))=1$.
The public key is $(n, e)$.
She send this key to Bob via e-mail.
Since $\operatorname{gcd}(e, \varphi(n))=1, \exists d \in \mathbb{N}$, ed $\equiv 1(\bmod \varphi(n))$.
Alice can easily find such a $d$ from a Bézout's relation using
Euclid's algorithm: $\exists u, v \in \mathbb{Z}, e u+\varphi(n) v=1$.
Then she picks $d=u+k \varphi(n)$ for a suitable $k \in \mathbb{Z}$ so that $d>0$.

## RSA: key generation - 1



Alice picks two distinct prime numbers $p$ and $q$.
She sets $n:=p q$.
She chooses $e \in \mathbb{N}$ such that $\operatorname{gcd}(e, \varphi(n))=1$.
The public key is $(n, e)$.
She send this key to Bob via e-mail.
Since $\operatorname{gcd}(e, \varphi(n))=1, \exists d \in \mathbb{N}$, ed $\equiv 1(\bmod \varphi(n))$.
Alice can easily find such a $d$ from a Bézout's relation using
Euclid's algorithm: $\exists u, v \in \mathbb{Z}, e u+\varphi(n) v=1$.
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.

## RSA: key generation - 1



Alice picks two distinct prime numbers $p$ and $q$.
She sets $n:=p q$.
She chooses $e \in \mathbb{N}$ such that $\operatorname{gcd}(e, \varphi(n))=1$.
The public key is $(n, e)$.
She send this key to Bob via e-mail.
Since $\operatorname{gcd}(e, \varphi(n))=1, \exists d \in \mathbb{N}, e d \equiv 1(\bmod \varphi(n))$.
Alice can easily find such a $d$ from a Bézout's relation using
Euclid's algorithm: $\exists u, v \in \mathbb{Z}, e u+\varphi(n) v=1$.
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 $\varphi(n)=(p-1)(q-1)$.

## RSA: key generation - 1



Alice picks two distinct prime numbers $p$ and $q$.
She sets $n:=p q$.
She chooses $e \in \mathbb{N}$ such that $\operatorname{gcd}(e, \varphi(n))=1$.
The public key is $(n, e)$.
She send this key to Bob via e-mail.
Since $\operatorname{gcd}(e, \varphi(n))=1, \exists d \in \mathbb{N}$, ed $\equiv 1(\bmod \varphi(n))$.
Alice can easily find such a $d$ from a Bézout's relation using
Euclid's algorithm: $\exists u, v \in \mathbb{Z}, e u+\varphi(n) v=1$.
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.

## RSA: key generation - 2

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.


## RSA: key generation - 2

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.


## RSA: key generation - 2

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 $\sqrt{n}$.


## RSA: key generation - 2

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 $\sqrt{n}$.
- They need to satisfy additional properties to avoid known attacks.


Bob knows the public key $(n, e)$.

## RSA: how to encrypt a message



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

## RSA: how to encrypt a message



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

$$
c \equiv m^{e}(\bmod n)
$$

## RSA: how to encrypt a message



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

$$
c \equiv m^{e}(\bmod n)
$$

It is the crypted message that Bob sends to Alice.

## RSA: how to decrypt a message

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

## RSA: how to decrypt a message

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

$$
k \equiv c^{d}(\bmod n)
$$

## RSA: how to decrypt a message

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

$$
k \equiv c^{d}(\bmod n)
$$

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

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

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

## RSA: how to decrypt a message



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

$$
k \equiv c^{d}(\bmod n)
$$

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

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

We conclude since $k$ has a unique representative in $\{0,1, \ldots, n-1\}$ and $m, k \in\{0,1, \ldots, n-1\}$.
The above proof is not (entirely) correct, why?

## RSA: how to decrypt a message



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

$$
k \equiv c^{d}(\bmod n)
$$

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

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

We conclude since $k$ has a unique representative in $\{0,1, \ldots, n-1\}$ and $m, k \in\{0,1, \ldots, n-1\}$.
The above proof is not (entirely) correct, why? You will fix it in the tutorial questions.

## RSA: en example

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

## RSA: en example

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 $\operatorname{gcd}(192,11)=1$.

## RSA: en example

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 $\operatorname{gcd}(192,11)=1$.

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

## RSA: en example

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 $\operatorname{gcd}(192,11)=1$.

Using Euclid's algorithm, Alice obtains the Bézout relation $192 \times(-2)+11 \times(35)=1$. Therefore, she sets $d=35$ so that $e d \equiv 1(\bmod 192)$.
The public key is $(n, e)=(221,11)$.
The private key is $(n, d)=(221,35)$.
Later, Bob wants to send the private message $m=149 \in\{0,1,2, \ldots, 220\}$ to Alice.

## RSA: en example

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 $\operatorname{gcd}(192,11)=1$.

Using Euclid's algorithm, Alice obtains the Bézout relation $192 \times(-2)+11 \times(35)=1$.
Therefore, she sets $d=35$ so that $e d \equiv 1(\bmod 192)$.
The public key is $(n, e)=(221,11)$.
The private key is $(n, d)=(221,35)$.
Later, Bob wants to send the private message $m=149 \in\{0,1,2, \ldots, 220\}$ to Alice.
He computes $m^{e}=149^{11} \equiv 89(\bmod 221)$.
So the encrypted message is $c=89 \in\{0,1,2, \ldots, 220\}$.
He sends it to Alice by e-mail.

## RSA: en example

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 $\operatorname{gcd}(192,11)=1$.
Using Euclid's algorithm, Alice obtains the Bézout relation $192 \times(-2)+11 \times(35)=1$.
Therefore, she sets $d=35$ so that $e d \equiv 1(\bmod 192)$.
The public key is $(n, e)=(221,11)$.
The private key is $(n, d)=(221,35)$.
Later, Bob wants to send the private message $m=149 \in\{0,1,2, \ldots, 220\}$ to Alice.
He computes $m^{e}=149^{11} \equiv 89(\bmod 221)$.
So the encrypted message is $c=89 \in\{0,1,2, \ldots, 220\}$.
He sends it to Alice by e-mail.
After receiving the e-mail, Alice computes $c^{d}=89^{35} \equiv 149(\bmod 221)$ to recover the original message $m=149$.

## RSA in practice: how to generate the prime numbers $p$ and $q$ ?

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 $k \leftarrow k+2$.

## RSA in practice: how to generate the prime numbers $p$ and $q$ ?

## 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 $k \leftarrow k+2$.

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.

## RSA in practice: how to generate the prime numbers $p$ and $q$ ?

## 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 $k \leftarrow k+2$.

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.

## RSA in practice: how to generate the prime numbers $p$ and $q$ ?

## 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 $k \leftarrow k+2$.

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

$$
a^{p} \equiv a(\bmod p)
$$

## RSA in practice: how to generate the prime numbers $p$ and $q$ ?

## 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 $k \leftarrow k+2$.

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

$$
a^{p} \equiv a(\bmod p)
$$

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

## RSA in practice: how to generate the prime numbers $p$ and $q$ ?

## 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 $k \leftarrow k+2$.

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

$$
a^{p} \equiv a(\bmod p)
$$

Therefore, since $24^{221} \equiv 176(\bmod 221)$, we know that 221 is not prime.
But, it is possible for such a congruence to hold even for a non-prime:

$$
2^{341} \equiv 2(\bmod 341)
$$

although $341=11 \times 31$.

RSA in practice: how to efficiently compute large powers? - 1
In the previous example, 149 and 11 are small but $149^{11}=803616698647447868139149$ is very large.

RSA in practice: how to efficiently compute large powers? - 1
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 $n$,

## RSA in practice: how to efficiently compute large powers? -1

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) $c \in\{0,1, \ldots, n-1\}$ such that $m^{e} \equiv c(\bmod n)$.

## RSA in practice: how to efficiently compute large powers? - 1

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) $c \in\{0,1, \ldots, n-1\}$ such that $m^{e} \equiv c(\bmod n)$. Hence we can iteratively multiply by $c$ but reduce to a representative in $\{0, \ldots, n\}$ before the next step.

## RSA in practice: how to efficiently compute large powers? - 1

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) $c \in\{0,1, \ldots, n-1\}$ such that $m^{e} \equiv c(\bmod n)$. Hence we can iteratively multiply by $c$ but reduce to a representative in $\{0, \ldots, n\}$ before the next step.

For instance, to compute $149^{11}(\bmod 221)$, we would do:$149^{1} \equiv 149(\bmod 221)$
(2) $149^{2}=149 \times 149=22201 \equiv 101(\bmod 221)$
(3) $149^{3}=101 \times 149=15049 \equiv 21(\bmod 221)$
(4) $149^{4}=21 \times 149=3129 \equiv 35(\bmod 221)$
(5) $149^{5}=35 \times 149=5215 \equiv 132(\bmod 221)$
(6) $149^{6}=132 \times 149=19668 \equiv 220(\bmod 221)$
(7) $149^{7}=220 \times 149=32780 \equiv 72(\bmod 221)$
(8) $149^{8}=72 \times 149=10728 \equiv 120(\bmod 221)$
(9) $149^{9}=120 \times 149=17880 \equiv 200(\bmod 221)$
(10) $149^{10}=200 \times 149=29800 \equiv 186(\bmod 221)$
(11) $149^{11}=186 \times 149=27714 \equiv 89(\bmod 221)$

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

## RSA in practice: how to efficiently compute large powers? - 2

We even have more efficient algorithm.
Write the exponent in binary $e=\overline{a_{r} a_{r-1} \ldots a_{1} a_{0}}{ }^{2}=\sum_{i=0}^{r} a_{i} i^{i}$ where $a_{i} \in\{0,1\}$. Then

## RSA in practice: how to efficiently compute large powers? -2

We even have more efficient algorithm.
Write the exponent in binary $e=\overline{a_{r} a_{r-1} \ldots a_{1} a_{0}}{ }^{2}=\sum_{i=0}^{r} a_{i} i^{i}$ where $a_{i} \in\{0,1\}$. Then

$$
m^{e}=m^{\Sigma a_{i} i^{i}}=\prod_{i=0}^{r}\left(m^{2^{i}}\right)^{a_{i}}
$$

## RSA in practice: how to efficiently compute large powers? - 2

We even have more efficient algorithm.
Write the exponent in binary $e=\overline{a_{r} a_{r-1} \ldots a_{1} a_{0}}{ }^{2}=\sum_{i=0}^{r} a_{i} i^{i}$ where $a_{i} \in\{0,1\}$. Then

$$
m^{e}=m^{\Sigma a_{i} 2^{i}}=\prod_{i=0}^{r}\left(m^{2^{i}}\right)^{a_{i}}
$$

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

## RSA in practice: how to efficiently compute large powers? -2

We even have more efficient algorithm.
Write the exponent in binary $e=\overline{a_{r} a_{r-1} \ldots a_{1} a_{0}}{ }^{2}=\sum_{i=0}^{r} a_{i} 2^{i}$ where $a_{i} \in\{0,1\}$. Then

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

So we just need to compute successive squares: $m^{2^{i+1}}=\left(m^{2^{i}}\right)^{2} \quad$ (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
    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:

struct PublicKey

## n: :Integer

end
struct PrivateKey
n: Integer
end
function gen_keys(p::Integer, $q::$ 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 $=(p-1)^{\star}(q-1)$
$(\mathrm{g}, \mathrm{u}, \mathrm{v})=\mathrm{gcdx}(\mathrm{e}, \mathrm{phi})$
$g==1$ || error("phi(n) and e must be coprime")
$\mathrm{u}<0$ ? $\mathrm{d}=(\mathrm{d}=\mathrm{phi})+\mathrm{phi}: \mathrm{d}=\mathrm{u} \% \mathrm{phi}$
$n=p^{*} q$
end
function encrypt ( $\mathrm{m}:$ : Integer, $\mathrm{k}:$ : PublicKey)
$0<=m$ | error ("m must be non-negative")
$m<k . n$ || error("m is too large")
return powermod (m,k.e,k.n)
end
function decrypt (c::Integer, k::Privatekey)
$0<=c$ | error("c must be non-negative")
$c<k \cdot n$ || error("c is too large")
return powermod (c, k.d,k.n)
end
$($ pbk, pvk) $=$ gen_keys $(13,17,11)$
println("The public key is $(n, e)=(\$(p b k . n), \$(p b k . e))$, give it to people willing to send you a secret message!")
println("The private key is $(n, d)=(\$(p v k . n), \$(p v k . d))$, don't share it!")
$m=149$
println("Original message: \$m")
$c=$ encrypt (m, pbk)
println("Encrypted message: \$c")
println("Decrypted message: \$(decrypt (c, pvk))")
[mat246@Pavilion mat246]\$ julia rsa.j
The public key is $(\mathrm{n}, \mathrm{e})=(221,11)$, give it to people willing to send you a secret message!
The private key is $(\mathrm{n}, \mathrm{d})=(221,35)$, don't share it!
Original message: 149
$\begin{array}{ll}\text { Encrypted message: } & 89 \\ \text { Decrypted message: } & 149\end{array}$


[^0]:    ${ }^{1}$ You will note the great given name.

