1.3. Os inteiros. Aplicações: criptografia

A parte da matemática discreta que envolve os números inteiros e as suas propriedades faz parte de uma área clássica da matemática chamada *Teoria dos Números*.

Aplicações motivadoras.

(1) Números aleatórios. Usámos na secção anterior a função `randperm` do `Maple` que permite gerar uma permutação aleatória. É também possível gerar números aleatórios com as instruções `rand()` (gera aleatoriamente um número natural com 12 algarismos) e `rand(n)()` (gera aleatoriamente um número natural entre 1 e $n-1$). A geração destes números é muito útil em simulações computacionais. Diversos métodos têm sido criados para gerar números destes. Em rigor, nenhum destes métodos gera números perfeitamente aleatórios, por isso é habitual chamar-lhes *números pseudo-aleatórios*.

O método mais comum é o chamado *método das congruências lineares*. Escolhemos quatro inteiros: o módulo m, o multiplicador a, o incremento c e a raiz x_0, com $2 \leq a < m$, $0 \leq c < m$ e $0 \leq x_0 < m$. Gera-se uma sequência de números pseudo-aleatórios $\{x_n\}$, com $0 \leq x_n < m$ para qualquer n, usando sucessivamente a *relação de congruência*

$$x_{n+1} = (ax_n + c) \bmod m.$$

Que relação é esta? Dados inteiros a e m, a notação $a \bmod m$ representa simplesmente o resto da divisão inteira de a por m. Por exemplo, $10 \bmod 5 = 0$, $7 \bmod 5 = 2$ e $2 \bmod 5 = 2$.

Por exemplo, a sequência de números pseudo-aleatórios gerada escolhendo $m = 9$, $a = 7$, $c = 4$ e $x_0 = 3$ é a seguinte:

$$\begin{aligned}
x_1 &= (7x_0 + 4) \bmod 9 = 25 \bmod 9 = 7 \\
x_2 &= (7x_1 + 4) \bmod 9 = 53 \bmod 9 = 8 \\
x_3 &= (7x_2 + 4) \bmod 9 = 60 \bmod 9 = 6 \\
x_4 &= (7x_3 + 4) \bmod 9 = 46 \bmod 9 = 1 \\
x_5 &= (7x_4 + 4) \bmod 9 = 11 \bmod 9 = 2 \\
x_6 &= (7x_5 + 4) \bmod 9 = 18 \bmod 9 = 0 \\
x_7 &= (7x_6 + 4) \bmod 9 = \ \ 4 \bmod 9 = 4 \\
x_8 &= (7x_7 + 4) \bmod 9 = 32 \bmod 9 = 5 \\
x_9 &= (7x_8 + 4) \bmod 9 = 39 \bmod 9 = 3
\end{aligned}$$

Como $x_9 = x_0$ e cada termo na sequência só depende do anterior, a sequência terá nove números diferentes antes de se começar a repetir:

$$3, 7, 8, 6, 1, 2, 0, 4, 5, 3, 7, 8, 6, 1, 2, 0, 4, 5, 3, \ldots$$

A maioria dos computadores usa este método para gerar números pseudo-aleatórios. Por exemplo, é muito utilizado o sistema módulo $m = 2^{31} - 1$ com incremento $c = 0$ e multiplicador $a = 7^5 = 16\,807$, que permite gerar $2^{31} - 2$ números antes que a repetição comece.

(2) Cálculo do máximo divisor comum. O algoritmo mais antigo que se conhece, e que aparece no livro VII dos *Elementos* de Euclides (c. 325 a.C. - 265 a.C.), calcula o *máximo divisor comum* $\mathrm{mdc}(a, b)$ de dois inteiros a e b.

Exemplo. Consideremos os inteiros $a = 252$ e $b = 54$. Dividindo a por b obtemos $a = 252 = 54 \times 4 + 36$. Tornando a dividir, agora b pelo resto 36, $54 = 36 \times 1 + 18$. Continuando o processo, chegamos a uma divisão exacta (porquê?), e o processo pára: $36 = 18 \times 2 + 0$. É fácil ver que 18, o último resto não nulo, é o máximo divisor comum de a e b.

$$\begin{aligned} 252 &= 54 \times 4 + 36 \\ 54 &= 36 \times 1 + \boxed{18} \\ 36 &= 18 \times 2 + \boxed{0} \end{aligned}$$

$$\therefore \quad \text{mdc}(252, 54) = 18$$

Não se trata de uma coincidência: não é difícil provar que, seguindo este procedimento para quaisquer outro par de inteiros positivos, o último resto não nulo é sempre igual a $\text{mdc}(a, b)$. Este é o algoritmo de Euclides:

```
procedure mdc(a, b : inteiros positivos)
x := a
y := b
while y ≠ 0
begin
   r := x mod y
   x := y
   y := r
end {x é o mdc(a, b)}
```

O algoritmo de Euclides é um dos resultados básicos dos números inteiros.

(3) Criptografia[8]: cifra de César. As congruências utilizam-se muito na criptografia. O exemplo mais simples (e muito antigo, remonta a Júlio César) é a chamada *cifra de César*. Ele usava um método de escrita de mensagens secretas transladando cada letra do alfabeto para três casas mais à frente:

A	B	C	D	E	F	G	H	I	J	L	M	N	O	P	Q	R	S	T	U	V	X	Z
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
D	E	F	G	H	I	J	L	M	N	O	P	Q	R	S	T	U	V	X	Z	A	B	C

Este sistema de encriptação pode ser descrito matematicamente de forma muito abreviada: substituímos cada letra por um inteiro de 0 até 22, baseado na sua posição no alfabeto:

[8] A *criptografia* (ou *criptologia*) é o estudo de mensagens secretas e dos processos de *encriptação*, ou seja, de escrita de mensagens secretas.

A	B	C	D	E	F	G	H	I	J	L	M
0	1	2	3	4	5	6	7	8	9	10	11

N	O	P	Q	R	S	T	U	V	X	Z	
12	13	14	15	16	17	18	19	20	21	22	

Então o método de César é definido pela função f que aplica cada inteiro n, $0 \leq n \leq 22$, no inteiro

$$f(n) = (n+3) \bmod 23.$$

Por exemplo,

$$\begin{array}{l} \mathrm{X} \longleftrightarrow 21 \\ \qquad \downarrow \\ \qquad f(21) = 24 \bmod 23 = 1 \longleftrightarrow \mathrm{B}. \end{array}$$

Teste. Como fica a mensagem "DESCOBRI A SOLUCAO" depois de encriptada pela cifra de César ?

Para recuperar a mensagem original a partir da mensagem encriptada basta considerar a função inversa f^{-1} que transforma um inteiro n, $0 \leq n \leq 22$, em $f^{-1}(n) = (n-3) \bmod 23$.

Podemos generalizar a cifra de César transladando k casas em vez de três:

$$f(n) = (n+k) \bmod 23.$$

É claro que a cifra de César é um método de encriptação muito pouco seguro. Podemos melhorá-lo um pouco definindo, mais geralmente, $f(n) = (an+b) \bmod 23$, com a e b inteiros escolhidos de modo a garantir que f é uma bijecção.

Teste. Que letra substitui J com a função encriptadora $f(n) = (7n+3) \bmod 23$?

(4) Criptografia: sistema RSA de chave pública. Consideremos agora o problema da criptografia da troca de mensagens confidenciais por intermédio de um canal público, supondo que os agentes comunicantes, a Alice e o Bruno, não partilham segredo nenhum. Uma solução para este problema reside no *protocolo de troca de mensagens por intermédio de chave pública*.

Se a Alice quer comunicar com o Bruno pede-lhe para ele gerar um par de chaves, uma chave pública u (conhecida por toda a gente) e uma chave privada v (conhecida apenas pelo Bruno). As chaves u e v são aplicações do espaço das mensagens para o espaço das mensagens e, para que o sistema funcione bem, e permita manter o secretismo na comunicação difícil de atacar, devem ter as seguintes propriedades:

(P1) $v(u(x)) = x$ para qualquer mensagem x.

(P2) deve ser difícil obter x conhecendo $u(x)$ e não conhecendo v.

O protocolo funciona do seguinte modo:

(1) A Alice envia a mensagem $u(x)$ ao Bruno pelo canal público.

(2) O Bruno obtém a mensagem aplicando v a $u(x)$.

Ao definirmos um *sistema criptográfico* deveremos explicitar o espaço das mensagens bem como as aplicações u e v. Um dos sistemas mais utilizados hoje em dia é o *sistema RSA* cujo nome deriva dos seus criadores (Rivest, Shamir e Adleman, em 1976).

A família de sistemas criptográficos RSA é definida do seguinte modo:

- espaço de mensagens: $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$, onde $n = p \times q$ para algum par de primos p, q.
- $u(x) = x^a \bmod n$, para qualquer $x \in \mathbb{Z}_n$.
- $v(y) = y^b \bmod n$, para qualquer $y \in \mathbb{Z}_n$.

onde

- a e b são tais que $a \times b \bmod (p-1) \times (q-1) = 1$.

Será de facto um bom sistema, isto é, satisfaz as propriedades (P1) e (P2) ?

Os conceitos e resultados relativos aos números inteiros que vamos apresentar nesta secção vão permitir verificar a propriedade (P1). No entanto, vamos apenas ser capazes de confirmar esta propriedade. Quanto à propriedade (P2), a sua confirmação é ainda hoje um problema em aberto[9].

Definição. Dados dois inteiros m e n, diz-se que m *divide* n, o que se denota por $m \mid n$, sse $\frac{n}{m}$ é um inteiro.

No `Maple` esta relação pode ser implementada da seguinte forma, com a ajuda da package `numtheory` de teoria dos números:

```
> with(numtheory);
> divisor := (m,n) -> evalb(n mod m=0);
> divisor(2,6);
```

true

```
> divisor(2,7);
```

false

Definição. Dados dois inteiros m e n, diz-se que r é o *resto* da divisão inteira de n por m, o que se denota por $r = n \bmod m$, se $0 \leq r < |m|$ e $n = q \times m + r$ para algum inteiro q.

```
> 23 mod 7;
```

2

[9] Trata-se de um dos problemas em aberto mais importantes da matemática, com grandes implicações práticas: até hoje, ninguém conseguiu demonstrar se o sistema RSA verifica a propriedade (P2), apesar de todos os especialistas conjecturarem que isso seja verdadeiro. Portanto, toda a criptografia actual assenta, não numa certeza absoluta, mas numa conjectura.

```
> 23 mod (-7);
                    2
> -23 mod 7;
                    5
```

Precisamos de introduzir agora a chamada *relação de congruência módulo n* entre inteiros:

Definição. Dados inteiros a e b e um natural n, diz-se que a é *congruente com b módulo n*, e escreve-se $a \equiv_n b$, se $a \bmod n = b \bmod n$.

Verifique que esta relação tem as seguintes propriedades:

(C1) Trata-se de uma relação de equivalência, isto é, para quaisquer a, b, c, d em $\mathbb{Z}$:

- $a \equiv_n a$ (reflexiva).
- Se $a \equiv_n b$ então $b \equiv_n a$ (simétrica).
- Se $a \equiv_n b$ e $b \equiv_n c$ então $b \equiv_n c$ (transitiva).

(C2) Se $a \equiv_n b$ e $c \equiv_n d$ então $a + c \equiv_n b + d$.

(C3) Se $a \equiv_n b$ e $c \equiv_n d$ então $a \times c \equiv_n b \times d$.

Assim, podem-se definir operações $+_n$ e $\times_n$ sobre o conjunto $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$:

$$a +_n b = (a + b) \bmod n, \qquad a \times_n b = (a \times b) \bmod n.$$

É esta a chamada *aritmética modular*[10] nos inteiros.

No contexto do RSA, é importante identificar quais os elementos de $\mathbb{Z}_n$, chamados *invertíveis*, que têm inverso relativamente à operação $\times_n$. Estes elementos estão intimamente ligados aos números primos.

Definição. Um natural p superior ou igual a dois diz-se *primo* quando, para qualquer natural n, se n divide p então $n = 1$ ou $n = p$.

No `Maple` o predicado `isprime` permite testar se um número é primo e a função `ithprime` lista o i-ésimo número primo:

```
> isprime(23);
                   true
> isprime(1);
```

[10] Para mais informação e manipulação destas operações no caso $1 \leq n \leq 10$ vá a `www.atractor.pt/mat/alg_controlo/arit_modular/mod_texto.htm`.

```
                    false

> ithprime(1);  # o primeiro numero primo

                     2

> ithprime(30000);  # o 30000-esimo numero primo

                    350377

> seq(ithprime(i), i=1..100); # Lista dos primeiros 100 primos

      2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,
101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197,
199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313,
317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439,
443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541
```

Testar se um número é primo é um problema de *complexidade polinomial*. Este resultado só foi provado muito recentemente (em 2002) por uma equipa de investigadores do *Indian Institute of Technology*, depois de muitas tentativas goradas durante o século passado. Os números primos têm um papel fundamental relativamente aos outros números naturais por causa do seguinte resultado:

Teorema Fundamental da Aritmética. *Todo o natural $n \geq 2$ pode escrever-se de maneira única, a menos da ordem dos factores, da seguinte forma:*

$$n = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$$

onde cada p_i é primo e $e_i > 0$.

No `Maple` podemos obter esta decomposição em primos através da função `ifactor`:

```
> ifactor(6600);

                    (2)^3 (3) (5)^2 (11)

> ifactor(27236209);

                    (7)^2 (11) (13)^3 (23)
```

Ao contrário do que acontece com o problema da verificação da primalidade de um número, não se conhece nenhum algoritmo que, em tempo polinomial, consiga realizar esta decomposição em primos. Mais, conjectura-se que um tal algoritmo não existe. Para ter uma ideia do que isto significa vamos medir o tempo que demora a função `ifactor` a ser executada:

```
> ha := time():
> ifactor(12345678998765432139874093811);
> time() - ha;
```

$(7)^2$ (19) (164394780101) (2685678223) (300347)
0.141 (segundos)

```
> ha := time():
> ifactor(12345678998765432139874093881199117);
> time() - ha;
```

$(3)^2$ (26127975507329) (422616703) (12422816017)
0.859 (segundos)

```
> ha := time():
> ifactor(123456789987654321398740938811991178756245);
> time() - ha;
```

$(3)^2$ $(5)^3 *$ (118629278703814742893) (458611871) (201709)
2.797 (segundos)

```
> ha := time():
> ifactor(1234567899876543213987409388119911787562451983);
> time() - ha;
```

(5) (53855317654603124771) (2205594178373) (207869503)
8.328 (segundos)

```
> ha := time():
> ifactor(12345678998765432139874093881199117875624519837);
> time() - ha;
```

(17) (53) (131478232428998933) (117807461794962971) (88463381)
24.625 (segundos)

Teorema "pequeno" de Fermat. *Se p é um primo que não divide a então $a^{p-1} \equiv_p 1$.*

Prova. Consideremos os inteiros

$$a \bmod p, \quad 2a \bmod p, \quad \ldots \quad , \quad (p-1)a \bmod p. \quad (*)$$

Estes números são todos distintos, dois a dois:

Se $na \bmod p = ma \bmod p$, com n, m entre 1 e $p-1$, então $na \equiv_p ma$, isto é, $na - ma \equiv_p 0$, ou seja, p divide $(n-m)a$. Mas p é primo e não divide a logo terá que dividir o outro factor $n-m$. Como $n-m$ é um número entre 0 e $p-1$, o único número destes que é múltiplo de p é o zero. Portanto, $n-m=0$, ou seja, $n=m$.

Assim, como todos estes $p-1$ números são distintos e pertencem ao conjunto $\{1, 2, \ldots, p-1\}$, a lista $(*)$ constitui um rearranjo (permutação) dos números $1, 2, \ldots, p-1$. Portanto, o produto dos números da lista é igual ao produto dos números $1, 2, \ldots, p-1$:

$$a^{p-1}(p-1)! \bmod p = (p-1)! \bmod p.$$

Daqui decorre que

$$a^{p-1}(p-1)! \equiv_p (p-1)! \Leftrightarrow (p-1)!(a^{p-1}-1) \equiv_p 0.$$

Portanto p divide $(p-1)!(a^{p-1}-1)$. Como p não pode dividir $(p-1)!$, terá então que dividir $a^{p-1}-1$, o que significa que $a^{p-1} \equiv_p 1$. □

Definição. Dois naturais m e n dizem-se *primos entre si* se $\text{mdc}(m,n) = 1$. Diz-se também que m é *coprimo* de n.

Um elemento $a \in \mathbb{Z}_n$ diz-se invertível se existe $b \in \mathbb{Z}_n$ tal que $a \times_n b = 1$, ou seja, $ab \equiv_n 1$.

Proposição. Um natural $a \in \mathbb{Z}_n$ é invertível se e só se a e n são primos entre si.

Prova. "⇒": $ab \equiv_n 1$ significa que $ab = nq+1$, isto é, $ab-nq = 1$, para algum inteiro q. Então, se d é um divisor comum de a e n, será um divisor de $ab - nq = 1$, pelo que necessariamente $d = 1$ ou $d = -1$. Isto mostra que $\text{mdc}(a,n) = 1$.
"⇐": Se $\text{mdc}(a,n) = 1$ então, pelo algoritmo de Euclides (usado em ordem inversa — ver exemplo na página seguinte), existem inteiros s e t tais que $1 = sa+tn$, ou seja, $sa = n\times(-t)+1$, o que mostra que $sa \equiv_n 1$. □

Portanto, em $\mathbb{Z}_5 = \{0,1,2,3,4\}$ todos os elementos não nulos são invertíveis:

$$1 \times_5 1 = 1, \quad 2 \times_5 3 = 1, \quad 3 \times_5 2 = 1, \quad 4 \times_5 4 = 1.$$

Mais geralmente, se n é primo todos os elementos $\neq 0$ de $\mathbb{Z}_n$ são invertíveis. Em $\mathbb{Z}_4 = \{0,1,2,3\}$ só o 1 e o 3 são invertíveis; em $\mathbb{Z}_6 = \{0,1,2,3,4,5\}$ só o 1 e o 5 são invertíveis.

Para cada n, o número de coprimos de n menores ou iguais a n é dado pela *função de Euler* (ou função *tociente*)

$$\begin{array}{rcl} \phi : \mathbb{N} & \to & \mathbb{N} \\ n & \mapsto & \#\{m : m \leq n \text{ e } m \text{ é coprimo de } n\} \end{array}$$

cujos valores podemos calcular com o `Maple`, recorrendo à *package* `numtheory`:

```
> with(numtheory);
> phi(4); phi(5); phi(6); phi(10);
```

```
                2
                4
                2
                4
```

```
> phi(107); phi(106);
```

```
                106
                52
```

Portanto, em $\mathbb{Z}_{107}$ há 106 elementos invertíveis e em $\mathbb{Z}_{106}$ só há 52.

Acabámos de obter solução para as congruências do tipo $ax \equiv_n 1$. Mais geralmente:

Resolvendo uma congruência. Sejam a, b, n inteiros. *Se* $mdc(a, n) = 1$ *então a equação* $ax \equiv_n b$ *tem uma solução*:

> (1) Determine inteiros s e t tais que $1 = as + nt$
> (usando o algoritmo de Euclides por ordem inversa)
>
> (2) Então $x = bs$ é uma solução da congruência
> (o conjunto completo de soluções é $\{bs + nk \mid k \in \mathbb{Z}\}$)

Teste. Resolva a equação $10x \equiv_{27} 5$.

Solução. Usemos o algoritmo de Euclides para determinar $\text{mdc}(10, 27)$:

$$\begin{aligned} 27 &= 10 \times 2 + 7 \\ 10 &= 7 \times 1 + 3 \\ 7 &= 3 \times 2 + 1 \end{aligned}$$

Portanto $\text{mdc}(10, 27) = 1$. Agora invertamos o processo para encontrar inteiros s e t tais que $1 = 10s + 27t$:

$$\begin{aligned} 1 &= 7 - 3 \times 2 \\ &= 7 - (10 - 7 \times 1) \times 2 \\ &= 7 \times 3 - 10 \times 2 \\ &= (27 - 10 \times 2) \times 3 - 10 \times 2 \\ &= 10 \times (-8) + 27 \times 3. \end{aligned}$$

Portanto $s = -8$ e $t = 3$. Logo $x = bs = -40$ é uma solução, e o conjunto completo de soluções é $\{-40 + 27k \mid k \in \mathbb{Z}\}$.

Voltemos agora ao algoritmo RSA. O algoritmo permite enviar mensagens encriptadas por uma chave pública a, mas para desencriptar a mensagem o receptor precisa de ter uma chave privada b (só do seu conhecimento).

Sejam p e q primos, $n = pq$, $m = (p-1)(q-1)$ e $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$. Considere ainda a tal que $\text{mdc}(a, m) = 1$ e seja b a solução da congruência $ab \equiv_m 1$.

No sistema RSA, podemos começar por traduzir as mensagens (sequências de letras) em sequências de inteiros (como na cifra de César):

A	B	C	D	E	F	G	H	I	J	L	M
00	01	02	03	04	05	06	07	08	09	10	11

N	O	P	Q	R	S	T	U	V	X	Z	
12	13	14	15	16	17	18	19	20	21	22	

O inteiro x daí resultante é depois transformado, com a ajuda da chave pública a, num inteiro $u(x) = x^a \bmod n$.

Teste. Codifique a mensagem HELP usando o sistema RSA com $p = 43$, $q = 59$ e $a = 13$.

Solução. Neste sistema $n = 43 \times 59 = 2537$. Note que, como 13 é primo, $\text{mdc}(13, 42 \times 58) = 1$. Traduzindo as letras no seu valor numérico, H→07, E→04, L→10 e P→14. A mensagem corresponde então ao número $x = 07041014$. Se aplicarmos já a chave pública u a x obtemos uma mensagem só com quatro algarismos: $u(x) = 07041014^{13} \bmod 2537 = 1507$. Para manter o número de algarismos na mensagem encriptada, como $n = 2537$ tem quatro algarismos, agrupam-se os algarismos de x em blocos de quatro e só depois se aplica u a cada um desses blocos[11]:

$$0704 \xrightarrow{u} 704^{13} \bmod 2537 = 0981,$$

$$1014 \xrightarrow{u} 1014^{13} \bmod 2537 = 1175.$$

A mensagem encriptada é então 0981 1175.

O receptor quando recebe a mensagem desencripta-a com a ajuda da chave privada b que só ele conhece: $v(u(x)) = u(x)^b \bmod n$.

Teste. Descodifique a mensagem seguinte, recebida usando o sistema RSA do exemplo anterior: 2128 2431.

Solução. Temos que resolver a congruência $13b \equiv_{42\times58} 1$ para determinar a chave privada b:

$$\begin{aligned} 42 \times 58 = 2436 &= 13 \times 187 + 5 \\ 13 &= 5 \times 2 + 3 \\ 5 &= 3 \times 1 + 2 \\ 3 &= 2 \times 1 + 1 \end{aligned}$$

donde

$$\begin{aligned} 1 &= 3 - 2 \times 1 \\ &= 3 - (5 - 3 \times 1) \\ &= 3 \times 2 - 5 \\ &= (13 - 5 \times 2) \times 2 - 5 \\ &= 13 \times 2 - 5 \times 5 \\ &= 13 \times 2 - (2436 - 13 \times 187) \times 5 \\ &= 13 \times (2 + 187 \times 5) - 2436 \times 5 \\ &= 13 \times 937 - 2436 \times 5. \end{aligned}$$

[11] Portanto, deveremos ter o cuidado de ter sempre $x < n$.

Portanto $b = 937$. Então $2128^b \bmod 2537 = 2128^{937} \bmod 2537 = 1718$ e $2431^b \bmod 2537 = 2431^{937} \bmod 2537 = 1314$, pelo que a mensagem original é, na versão numérica, 1718 1314, ou seja STOP.

Já sabemos como encriptar e desencriptar mensagens no sistema RSA. Falta assegurar, como tínhamos anunciado, que o RSA satisfaz a propriedade (P1), isto é, que a desencriptação v é de facto inversa da encriptação u:

Proposição. Sejam p e q primos distintos, $n = pq$ e $m = (p-1)(q-1)$. Se a e b são inteiros tais que $ab \equiv_m 1$, então:

(a) Para qualquer inteiro x, $x^{ab} \equiv_n x$.

(b) Para qualquer inteiro $x < n$, $v(u(x)) = x$.

Prova. (a) Como $ab \equiv_m 1$ então $ab = mk + 1$ para algum inteiro k. Consequentemente,

$$x^{ab} = x^{1+mk} = x(x^{mk}) = x(x^{p-1})^{k(q-1)}.$$

Provemos que $x^{ab} \equiv_p x$:

Caso 1: mdc$(x, p) = 1$: Pelo Teorema pequeno de Fermat sabemos que $x^{p-1} \equiv_p 1$. Logo, $x^{ab} \equiv_p x(1)^{k(q-1)} = x$.

Caso 2: mdc$(x, p) \neq 1$: Neste caso, como p é primo, necessariamente p divide x. Portanto, $x \equiv_p 0$, pelo que $x^{ab} \equiv_p x$.

De modo análogo, podemos provar que $x^{ab} \equiv_q x$. Portanto, quer p quer q dividem $x^{ab} - x$. Consequentemente, como são primos distintos, o seu produto $n = pq$ também divide $x^{ab} - x$, ou seja, $x^{ab} \equiv_n x$.

(b) Uma vez que $v(u(x)) = v(x^a \bmod n) = (x^a \bmod n)^b \bmod n = x^{ab} \bmod n$, usando (a) obtemos $v(u(x)) = x \bmod n = x$. □

Observe que, em geral, no sistema RSA assume-se que quase tudo é do conhecimento público, incluindo a forma da função encriptadora. Isto significa que um intruso que intersecta uma mensagem RSA sabe que esta foi formada com a função $u(x) = x^a \bmod n$, e conhece os valores a e n. A vantagem desta informação ser pública reside no facto da Alice e do Bruno para comunicarem entre si numa linha de comunicação insegura não precisarem de pensar numa maneira de trocarem entre si secretamente o expoente de encriptação a e o módulo n. Só a chave privada b nunca pode circular entre ambos, pelo canal de comunicação, para que não possa ser interceptada.

É claro que o exemplo de RSA que apresentamos acima é (matematicamente) inseguro, um intruso facilmente quebraria o sistema: conhecendo $n = 2537$, facilmente obteria a sua factorização prima $pq = 43 \times 49$, calcularia $(p-1)(q-1) = 2436$, e usaria o algoritmo de Euclides para descobrir a chave privada $b = 937$. Contudo, o primeiro passo neste processo requere ao intruso ser capaz de descobrir os dois factores primos p e q de n. É aqui que reside a grande segurança do sistema RSA: a aparente dificuldade em resolver este problema, desde

que p e q sejam muito grandes. Por exemplo, se p e q forem primos com aproximadamente 100 algarismos, o algoritmo de factorização mais rápido que se conhece levaria milhões de anos para encontrar a factorização $n = pq$, mesmo com a ajuda de supercomputadores.

Portanto, mesmo sendo pública a informação de n e a, um intruso não deverá ser capaz de descobrir o expoente b de desencriptação[12].

Como é que o processo de troca de mensagens secretas entre a Alice e o Bruno se desenrola na realidade?

O Bruno, o receptor, inicia o processo escolhendo primos p e q e definindo $n = pq$ e $m = (p-1)(q-1)$. Escolhe ainda um expoente a de encriptação (a chave pública) em $\mathbb{Z}_m$ tal que $\text{mdc}(a, m) = 1$ e, usando o algoritmo de Euclides, determina $b \in \mathbb{Z}_m$ (a chave privada) tal que $ab \equiv_m 1$. Depois envia os valores de n e a para a Alice (a partir daqui, não havendo garantias de segurança no canal de comunicação, os valores de n e a passam a ser eventualmente públicos), mantendo a chave privada b só do seu conhecimento. A Alice tem agora os elementos para encriptar as suas mensagens com a função u e enviá-las ao Bruno. Como só este conhece o valor de b, só ele poderá decifrar a mensagem aplicando a função v.

Simulemos isto com a ajuda do `Maple`. O Bruno começa por procurar dois primos grandes p e q, por exemplo com 80 algarismos. Com `nextprime(r)` calcula o menor primo maior do que o input inteiro r:

```
> r := rand(10^80)();
> s := rand(10^80)();
> p := nextprime(r);
> q := nextprime(s);
```

$r :=$ 19669081321110693270343633073697474256143563558458718976746753830538032062222085
$s :=$ 74121768604305613921745580037409259811952655310075487163797179490457039169594160
$p :=$ 19669081321110693270343633073697474256143563558458718976746753830538032062222257
$q :=$ 74121768604305613921745580037409259811952655310075487163797179490457039169594213

O uso da função `rand` na selecção dos primos tem a intenção de os tornar difíceis de adivinhar. Em seguida, calcula $n = pq$ e $m = (p-1)(q-1)$.

```
> n := p*q;
```

$n :=$ 14579070943426365719341081596858629803265159149118248616433975229804975507362306154960468021868768356118367534405251995876980199548391659324278422783737069987 41

Este valor de n com 160 algarismos permite encriptar mensagens até 80 letras num só bloco.

```
> m := (p-1)*(q-1);
```

[12] A não ser que descubra um algoritmo de factorização que torne o problema realizável em tempo útil, o que os matemáticos acreditam (conjecturam) não existir. Mas, até hoje, ninguém foi capaz de provar isso. Por esta razão a factorização, para a qual se julga não existir algoritmo polinomial, é o calcanhar de Aquiles do RSA.

$m :=$ 14579070943426365719341081596858629803265159149118248616433975229804975507362305217051968767705696435226236423337911314914791514206330253884945212830247518227 2

Em seguida, decide a escolha de a. Como será um valor público, não se preocupa em gerar números aleatoriamente. A única preocupação é que satisfaça $\text{mdc}(a, m) = 1$. Por exemplo, pode fazer $a = 2^{16} + 1 = 65537$. Para verificar que se trata de um expoente de encriptação válido, verifica o $\text{mdc}(a, m)$:

```
> isprime(a);
> gcd(a,m);
```

true
1

Agora calcula b tal que $ab \equiv_m 1$:

```
> b := eval(1/a mod m);
```

$b :=$ 34180298922096847472065507840720943425419102236324807359431775852717312155060777829318324017852209549910908745378489609482547509922679456023648197991886310291 3

Está pronto para definir as função de encriptação u e desencriptação v:

```
> u := (x,a,n) -> Power(x,a) mod n:
> v := (x,b,n) -> Power(x,b) mod n:
```

Verifiquemos num exemplo (mensagem $x =$ "5") que os procedimentos u e v são inversos um do outro:

```
> y := u(5,a,n); x :=v(y,b,n);
```

$y :=$ 44669982652857045772784970284788245601106395884543657540636484577393488318578590496921573836272232443097862919568742270009616466410734910391539516416953887426 1
$x := 5$

Teste. Descodifique a mensagem 14452713420773338508105879307212461196372763000865429239910113230948208915317126596566680325137342547829329336431874588144606598803386096533964035759675077856790 recebida pelo Bruno.

Solução.

```
> x :=v(14452713420773338508105879307212461196372763000865429239910113230948208915317126596566680325137342547829329336431874588144606598803386096533964035759675077856790,b,n);
```

$x := 417181903041104171816191819160017030817021604180017$

Como o primeiro par de algarismos, 41, não corresponde a nenhuma letra (cf. tabela da página 57), o par original deverá ser 04 (o *Maple* não escreveu o 0), ou seja, é a letra E. Continuando, 17→S, 18→T, 19→U, etc.[13]. A mensagem original é

"ESTUDEMESTRUTURASDISCRETAS".

Até agora utilizámos um método muito simples de conversão de letras em números, que representa A pelo número 0, B pelo número 1, etc., até Z. Este método tem uma desvantagem óbvia: não funciona conjuntamente com maiúsculas e minúsculas, com espaços, acentos e outros caracteres. Existe um método muito utilizado de conversão de caracteres no código ASCII, usado pela maioria dos computadores, e que está implementado no `Maple`, permitindo a conversão de cadeias de caracteres alfanuméricos em inteiros e vice-versa. A instrução é

`convert("xxx", bytes)`.

```
> u := convert("Bom dia, a vossa missão para hoje é codificar esta mensagem.",
bytes);
```

$u :=$[66, 111, 109, 32, 100, 105, 97, 44, 32, 97, 32, 118, 111, 115, 115, 97, 32, 109, 105, 115, 115, 227, 111, 32, 112, 97, 114, 97, 32, 104, 111, 106, 101, 32, 233, 32, 99, 111, 100, 105, 102, 105, 99, 97, 114, 32, 101, 115, 116, 97, 32, 109, 101, 110, 115, 97, 103, 101, 109, 46]

Em sentido inverso:

```
> convert([66, 111, 109, 32, 100, 105, 97, 44, 32, 97, 32, 118, 111, 115, 115,
97, 32, 109, 105, 115, 115, 227, 111, 32, 112, 97, 114, 97, 32, 104, 111, 106,
101, 32, 233, 32, 99, 111, 100, 105, 102, 105, 99, 97, 114, 32, 101, 115, 116,
97, 32, 109, 101, 110, 115, 97, 103, 101, 109, 46], bytes);
```

"Bom dia, a vossa missão para hoje é codificar esta mensagem."

Leituras suplementares. É possível implementar um procedimento em `Maple` que traduza mensagens alfanuméricas numa sequência de inteiros, usando a conversão para ASCII, concatene essa sequência num só inteiro grande pronto a ser utilizado pelo algoritmo RSA e, em sentido inverso, converta un inteiro na respectiva mensagem escrita. Se estiver interessado, consulte o texto[14] `conversaoRSA2.mws` de Mike May (2002).

O sistema RSA tem resistido a ataques de criptoanalistas, à custa do aumento da dimensão das chaves, mas é necessário ir acompanhando os desenvolvimentos mais recentes. Apesar da matemática subjacente ser há muito conhecida, a cifra RSA surgiu apenas nos anos 70 porque é aplicável apenas com números primos de grande dimensão e só nos anos 70 apareceram computadores potentes de custo aceitável. Ataques mais conhecidos em 2006:

[13] Não é difícil implementar no `Maple` este procedimento de tradução das letras em inteiros e vice-versa (cf. `conversaoRSA.mws` na página da disciplina, na pasta com ficheiros `Maple`).

[14] Na página da disciplina.

- Números até 100 bits consegue-se quebrar, com PCs.
- Em computação paralela são conhecidos ataques até 640 bits (actualmente recomenda-se o uso de números RSA-1024, ou RSA-2048)[15].

Qual é a complexidade do algoritmo de factorização do número n em primos pq?
Factorização à força bruta: testar todos os primos até $n/2$ (é de complexidade $O(\sqrt{n})$).

Exemplo. $n := 408508091$.

1.	`Divisível por 3?` Não!
2.	`Divisível por 5?` Não!
3.	`etc.`
⋮	
2099.	`Divisível por 18 313 (é o` 2099^o `primo)?` Sim, está identificado o primo p.
2100.	$q = n/18313 = 22307$.

Demorou 2099 passos, e n só tem 9 algarismos! Imagine RSA-640 com 193 algarismos decimais...

A empresa norte-americana *RSA Security*[16] submete a concurso a factorização de números RSA com prémios até 200 mil dólares (para o caso RSA-2048):

- Primeiro prémio (100 dólares) atribuído em Abril 1994, pela factorização de números RSA-129.
- O prémio mais elevado (20 000 dólares) foi ganho em Novembro 2005 na factorização do RSA-640

  310741824049004372135075003588856793003734602284272754572016194882320644051808150455634682967172328678243791627283803341547107310850191954852900733772482278352574238645401469173660247765234660 9

  por F. Bahr, M. Boehm, J. Franke, T. Kleinjung (Univ. Bona, Alemanha). Os factores são

  163473364580925384844313388386509085984178367003309231218111085238933310010450815121211816751157 9

  e

  190087128166482211312685157393541397547189678996851549366663853908802710380210449895719126146557 1

  Os cálculos foram efectuados durante 540 dias por um conjunto de 80 AMD64 Opteron (CPU que equipa cerca de 10% dos supercomputadores mais rápidos do mundo).

[15] A notação RSA-xxxx refere-se a um número RSA com tamanho xxxx em bits.
[16] `www.rsa.com/rsalabs/node.asp?id=2093`.

Desafio. Se conseguir descobrir a factorização do RSA-704

```
74037563479561712828046796097429573142593188889231289084936232638972765034028266
27689199641962511784399589433050212758537011896809828673317327310893090055250511
68770632990723963807867100860969625379346505637963 59
```

terá 20 valores à disciplina e pode candidatar-se ao prémio de 30 000 dólares da *RSA Security*.

Exemplos de chaves públicas usadas em algumas páginas web.[17]

```
>  with(StringTools):
```

Chave pública Departamento de Matemática (tamanho: 140 Bytes / 1120 Bits)

```
> nDMUC := "30 81 89 02 81 81 00 db 35 3c 03 49 fc e0 48 e4 b6 7c 55 66 f3 52 08
39 b9 d8 bf eb 9c c7 e8 7a 32 54 fc 88 66 19 de a0 08 b1 19 ad a0 34 75 0c 2b 0d
f5 6d 3b 9d f4 78 2e 2e fe 45 d3 7e b5 ff 7c f6 9a 3b d7 13 46 a9 e1 ab 0b 01 d8
d8 3c 65 d7 ce a3 e7 32 c3 59 54 97 54 b7 a4 6e be 07 61 25 0d 32 04 3a 99 15 19
23 9d 97 61 e1 66 5d b7 ef 81 e9 1e cd bc 25 fd 39 9b 74 6a 86 09 07 9a 98 bd 45
f2 ba 84 a1 02 03 01 00 01":
> nDMUC := SubstituteAll(nDMUC," ",""):
```

Conversão da representação hexadecimal de um número para a sua representação decimal

```
> nDMUC := convert(nDMUC,decimal,hexadecimal);
```

```
nDMUC := 26986751662544233897390996989562786998209640195002153770034468141230169
50163931124306976585375250715400911398515483295152423812242464904341510349824376
57607908609525327637173783415854826421482431563077009840890804022964491227968726
58542504958657945923684483016763566353046590571882008173675016517836292426828436
78677672254248775317520385
```

```
> ifactor(nDMUC);
```

Warning, computation interrupted

```
ifactor(nDMUC,easy);
```

$(3)^3$ (5) (7) (23) _c324_1 (19421) (18211)

Cálculo de _c324_1 (número com 324 algarismos)

```
> nDMUC/(3^3*5*7*23*19421*18211);
```

```
35106342494443210137514656205211299631137884284193430183859548425279394063722689 75
81540103919906067158525538784012701385666549546945725709737318974523106555931258 37
15950002757151462297064587717616229234945606772955877418526208596227762854261084 89
87305222425436763631372662073167373311228321757390818159517281638334382237296 1
```

[17] Cf. `ChavesPublicas.mws`

```
> isprime(%);

   false

> ifactor(nDMUC/(3^3*5*7*23*19421*18211),easy);

   _c324_2
```

(Trata-se de um número com 324 algarismos, afinal com 2 factores primos, mas não os calcula.)

Chave pública CGD caixa directa (tamanho: 140 Bytes / 1120 Bits)

```
>  nCGD := "30 81 89 02 81 81 00 d1 e2 d6 c0 3f 56 05 f4 b7 cb c2 d7 e0 63 aa bf
0e 35 de c7 c3 ca a0 2e ba 59 fa 02 49 2e 1d 95 44 1b 2b d7 88 27 47 42 2c 4f eb
20 ca 75 bf e8 2a 00 c6 d7 11 e7 3c aa 21 f4 06 97 62 80 4a fd 6f 04 42 56 f2 b1
07 0e 15 00 86 c0 cf 98 35 7c cc dc 84 84 d8 e6 c0 67 ed e2 4b 34 d2 47 3e 1d 0e
29 14 6e 64 94 37 60 f9 02 03 01 00 01":
```

Chave pública VISA eCommerce (tamanho: 270 Bytes / 2160 Bits)

```
> nVISA := "30 82 01 0a 02 82 01 01 00 af 57 de 56 1e 6e a1 da 60 b1 94 27 cb 17
db 07 3f 80 85 4f c8 9c b6 d0 f4 6f 4f cf 99 d8 e1 db c2 48 5c 3a ac 39 33 c7 1f
6a 8b 26 3d 2b 35 f5 48 b1 91 c1 02 4e 04 96 91 7b b0 33 f0 b1 14 4e 11 6f b5 40
af 1b 45 a5 4a ef 7e b6 ac f2 a0 1f 58 3f 12 46 60 3c 8d a1 e0 7d cf 57 3e 33 1e
fb 47 f1 aa 15 97 07 55 66 a5 b5 2d 2e d8 80 59 b2 a7 0d b7 46 ec 21 63 ff 35 ab
a5 02 cf 2a f4 4c fe 7b f5 94 5d 84 4d a8 f2 60 8f db 0e 25 3c 9f 73 71 cf 94 df
4a ea db df 72 38 8c f3 96 bd f1 17 bc d2 ba 3b 45 5a c6 a7 f6 c6 17 8b 01 9d fc
19 a8 2a 83 16 b8 3a 48 fe 4e 3e a0 ab 06 19 e9 53 f3 80 13 07 ed 2d bf 3f 0a 3c
55 20 39 2c 2c 00 69 74 95 4a bc 20 b2 a9 79 e5 18 89 91 a8 dc 1c 4d ef bb 7e 37
0b 5d fe 39 a5 88 52 8c 00 6c ec 18 7c 41 bd f6 8b 75 77 ba 60 9d 84 e7 fe 2d 02
03 01 00 01":
```