

# Números Inteiros e Criptografia, 2021.1

## TRABALHO FINAL<sup>†</sup>

Submeta as soluções de todas as questões do trabalho até **21 de outubro às 9:00** salvando os arquivos na sua pasta chamada “Trabalho Final” no Google Drive

Em todas as questões que envolverem codificação de texto para número ou vice-versa (incluindo a sua implementação do RSA), usaremos a tabela de correspondência entre números e símbolos dada na última página deste PDF.

Lembre-se: você pode usar tudo o que foi visto em aula, em listas anteriores, ou mesmo qualquer questão do trabalho para responder outras questões (mesmo que você não tenha feito a questão que está citando!), desde que você seja claro na sua citação do que está usando. A única exceção é a Questão 3, na qual você não pode usar o item b na solução do item a, a não ser que você faça o item b.

Como sempre, justifique todas as questões.

**Questão 1.** Chamamos um natural ímpar  $n > 1$  de *pseudoprímo de Miller–Rabin para a base b* se  $n$  é composto mas o teste de Miller–Rabin para  $n$  com base  $b$  é inconclusivo.

**a.** Mostre que se um número  $n$  é pseudoprímo de Miller–Rabin para a base  $b$ , então  $n$  é um pseudoprímo de Fermat para esta mesma base.

**b.** Seja  $n \in \mathbb{N}$  e considere o número  $F_n = 2^{(2^n)} + 1$ . Mostre que, se  $F_n$  é composto, então  $F_n$  é um pseudoprímo de Miller–Rabin para a base 2.

**Questão 2.** O expoente  $e = 2$  nunca pode ser usado como expoente público no RSA na prática. Por quê?

**Questão 3** (Teorema Chinês dos Restos).

**a** (TCR, versão simplificada). Sejam  $\ell_1$  e  $\ell_2$  naturais coprimos e  $n = \ell_1 \cdot \ell_2$ . Prove que, para quaisquer inteiros  $a$  e  $b$ , o sistema

$$\begin{cases} x \equiv a \pmod{\ell_1} \\ x \equiv b \pmod{\ell_2} \end{cases}$$

tem *uma única* solução em módulo  $n$ , e esta solução é dada por

$$x \equiv (a \cdot \ell_2 \cdot \ell'_2) + (b \cdot \ell_1 \cdot \ell'_1) \pmod{n},$$

onde  $\ell'_1$  é o inverso de  $\ell_1$  em módulo  $\ell_2$  e  $\ell'_2$  é o inverso de  $\ell_2$  em módulo  $\ell_1$ .

---

<sup>†</sup>Publicado em 6/10. Atualizado em 14/10, corrigindo o enunciado da questão 5.

**b** (TCR, versão completa). Suponha que  $\ell_1, \ell_2, \dots, \ell_k$  sejam primos entre si, i.e.,  $\text{mdc}(\ell_i, \ell_j) = 1$  para todos  $1 \leq i < j \leq k$ , e seja  $n = \prod_{i=1}^k \ell_i$ . Prove que, para quaisquer inteiros  $a_1, a_2, \dots, a_k$ , o sistema

$$\begin{cases} x \equiv a_1 \pmod{\ell_1} \\ x \equiv a_2 \pmod{\ell_2} \\ \vdots \\ x \equiv a_k \pmod{\ell_k} \end{cases}$$

possui *uma única solução* módulo  $n$ , e esta solução é dada por

$$x \equiv \sum_{i=1}^k (a_i \cdot m_i \cdot m'_i) \pmod{n},$$

onde para cada  $i$  com  $1 \leq i \leq k$  definimos

$$m_i = \frac{n}{\ell_i} = \prod_{\substack{j \in \{1, 2, \dots, k\} \\ j \neq i}} \ell_j$$

$m'_i =$  o inverso multiplicativo de  $m_i$  em módulo  $\ell_i$ .

**Questão 4** (Aceleração de descrição no RSA com o TCR). Uma das aplicações práticas do Teorema Chinês dos Restos é para acelerar a etapa de descrição de mensagens. O procedimento é o seguinte: ao gerar seu módulo público  $n = pq$ , expoente público  $e$  e expoente privado  $d$ , o usuário também calcula e guarda os seguintes valores:

- o inverso de  $p$  módulo  $q$
- o inverso de  $q$  módulo  $p$
- a forma reduzida de  $d$  módulo  $p - 1$
- a forma reduzida de  $d$  módulo  $q - 1$ .

Lembrando que a tarefa básica na etapa de descrição é, ao receber um bloco encriptado  $m$ , calcular a forma reduzida da potência modular  $m^d \pmod{pq}$ , explique como usar o Teorema Chinês dos Restos (e o Pequeno Teorema de Fermat) e os dados calculados acima para tornar essa tarefa mais fácil (e explique em que sentido a tarefa fica mais fácil).

**Questão 5.** Três pares  $(n, e)$  de chaves públicas do RSA,

(68397633165470058274884361378645822467587248043012077070501, 5),

(290216330211198637784379813527744010595500946558768017267121, 5),

(245841236512478852752909734912575581815967630033049838269083, 3)

foram geradas usando somente 5 números primos distintos no total. Usando apenas o conhecimento obtido nessa disciplina, explique como a fatoração em primos de dois desses módulos públicos pode ser encontrada rapidamente (e dê a fatoração).

### Questão 6.

**a.** Como vimos, a segurança do RSA está, em parte, baseada no fato de que é difícil calcular *raízes* modulares em geral: dados  $a \bmod n$  e um natural  $e$ , é difícil encontrarmos  $x$  tal que  $x^e \equiv a \pmod n$ . Note que isso é diferente em aritmética comum, onde até mesmo calculadoras simples de bolso são capazes de encontrar  $x$  tal que  $x^e = a$  (assuma esse fato como verdadeiro).

Explique por que se alguma solução  $x$  para a congruência  $x^e \equiv a \pmod n$  satisfaz

$$0 \leq x^e < n,$$

então é **fácil** encontrar  $x$ .

**b.** Usando o item **a**, explique a importância de que, no RSA com expoente público  $e$  pequeno e módulo  $n$  grande, cada bloco  $b$  da mensagem a ser encriptada seja razoavelmente *grande* (apesar de menor do que  $n$ , é claro).

**c.** Considere a situação em que  $k$  pessoas,  $P_1, P_2, \dots, P_k$ , tenham, cada uma, sua própria chave pública para o módulo, mas a mesma chave pública  $e$  para o expoente. Seja  $n_i$  o módulo da chave pública da pessoa  $P_i$  e assumamos que todos esses módulos são coprimos entre si. Agora suponhamos que Maria codifique a mesma mensagem  $m$  para cada pessoa: temos  $0 \leq m < \min\{n_1, n_2, \dots, n_k\}$  e Maria manda  $c_i = m^e \bmod n_i$  para a pessoa  $P_i$ . Finalmente, suponhamos que  $k \geq e$ . Mostre que um invasor que escuta todos os textos codificados pode recuperar a mensagem  $m$  (Dica: use o Teorema Chinês dos Restos).

**Questão 7.** Implemente o RSA em Python! Sua implementação deve ter (pelo menos) os seguintes componentes.

**a.** Uma função para gerar números primos. Sua função deve receber como entrada um natural  $n$  e gerar um número (provavelmente) primo  $p$  satisfazendo  $10^n < p < 10^{n+2}$ , sorteando  $p$  aleatoriamente no intervalo desejado e rodando 10 testes de Miller–Rabin com bases  $b$  aleatórias no intervalo  $1 < b < p - 1$ . (Naturalmente,  $p$  só deve ser aceito como provavelmente primo se todos os testes forem inconclusivos.)

**b.** Uma função chamada `gera_chaves` (por favor use este nome) para gerar chaves do RSA. Sua função deve usar sua função da letra **a** para gerar primos  $p$  e  $q$ , cada um com aproximadamente 75 algarismos, e retornar:

- $n = pq$
- algum número  $e$  inversível módulo  $\phi(n) = (p - 1)(q - 1)$
- o inverso  $d$  de  $e$  módulo  $\phi(n)$

Para uma solução realmente *completa*, sua função deve retornar também:

- $p$
- $q$
- o inverso de  $p$  módulo  $q$
- o inverso de  $q$  módulo  $p$

- a forma reduzida de  $d$  módulo  $p - 1$
- a forma reduzida de  $d$  módulo  $q - 1$ .

**c.** Uma função chamada `encriptar` (por favor use este nome) que recebe como entrada uma string `texto` e números `n` e `e`, e retorna uma lista de números que seja uma sequência válida dos blocos numéricos resultantes da encriptação do `texto` com chave pública de módulo `n` e expoente `e`.

**d.** Uma função chamada `descriptar` (por favor use este nome) que recebe como entrada uma lista `blocos` e números `n` e `d`, e retorna a string resultante da descriptação da sequência de blocos usando a chave privada de módulo `n` e expoente `d`.

Para uma solução realmente *completa*, implemente a versão rápida de descriptação, usando a Questão 4 e os valores adicionais retornados pela função `gera_chaves`.

Use a transformação de símbolos em números dados na tabela ao final deste documento; você encontra a tabela em versões de dicionários de Python, um para conversão de símbolos em números e outra na direção contrária, em

<https://www.hugonobrega.com/codigo.py>

Como todos usaremos a mesma tabela de conversão, faremos uma troca de mensagens encriptadas ao vivo no horário da aula teórica de 21/10 (começando às 9:00)! A participação é completamente livre e não conta para a avaliação de nenhuma forma.

Teste suas funções!

Por exemplo, se você usou `gera_chaves` e obteve `n`, `e`, `d` como chaves pública e privada, então você deve obter, no interpretador do Python:

```
>>> descriptar(encriptar('eu amo pão (MION, 2021)', n, e), n, d)
'eu amo pão (MION, 2021)'
```

cód.	símb.	cód.	símb.	cód.	símb.	cód.	símb.
111	0	141	m	171	B	211	Â
112	1	142	n	172	C	212	Ã
113	2	143	o	173	D	213	É
114	3	144	p	174	E	214	Ê
115	4	145	q	175	F	215	Í
116	5	146	r	176	G	216	Ó
117	6	147	s	177	H	217	Ô
118	7	148	t	178	I	218	Õ
119	8	149	u	179	J	219	Û
121	9	151	v	181	K	221	Ç
122	=	152	w	182	L	222	,
123	+	153	x	183	M	223	.
124	-	154	y	184	N	224	!
125	/	155	z	185	O	225	?
126	*	156	á	186	P	226	;
127	a	157	à	187	Q	227	:
128	b	158	â	188	R	228	_
129	c	159	ã	189	S	229	(
131	d	161	é	191	T	231	)
132	e	162	ê	192	U	232	"
133	f	163	í	193	V	233	#
134	g	164	ó	194	W	234	\$
135	h	165	ô	195	X	235	%
136	i	166	õ	196	Y	236	@
137	j	167	ú	197	Z	237	(espaço)
138	k	168	ç	198	Á	238	(nova linha)
139	l	169	A	199	À		