Trabalho A2: Enviando emails criptografados

Disciplina: Álgebra e Criptografia
Professor: Luciano Castro
Alunos: Ezequiel Braga
Darlan Araújo

1 Funcionamento do RSA

O RSA é um algoritmo de criptografia que utiliza um par de chaves: uma pública (n, e) e outra privada (d). A chave pública é usada para criptografar, enquanto a chave privada é usada para descriptografar. A segurança do RSA está baseada na dificuldade de fatorar o produto de dois números primos grandes, que compõem a chave pública. O algoritmo funciona da seguinte forma:

1. Escolhe-se dois primos grandes p e q;
2. Calcula-se $n = pq$ e $\phi(n) = (p-1)(q-1)$;
3. Escolhe-se e, com $2 < e < \phi(n)$ e $mdc(e, \phi(n)) = 1$;
4. Dada uma mensagem m, com $0 \leq m \leq n-1$, a mensagem codificada é $c = m^e \ (mod(n))$;
5. Para decodificar, basta calcular $c^d (mod(n))$, onde $d = e^{-1} \ mod(\phi(n))$.

2 Gerador de chaves

Como visto anteriormente, para a utilização do RSA, cada usuário precisa de um par de chaves. A função a seguir é responsável por criar tais chaves:

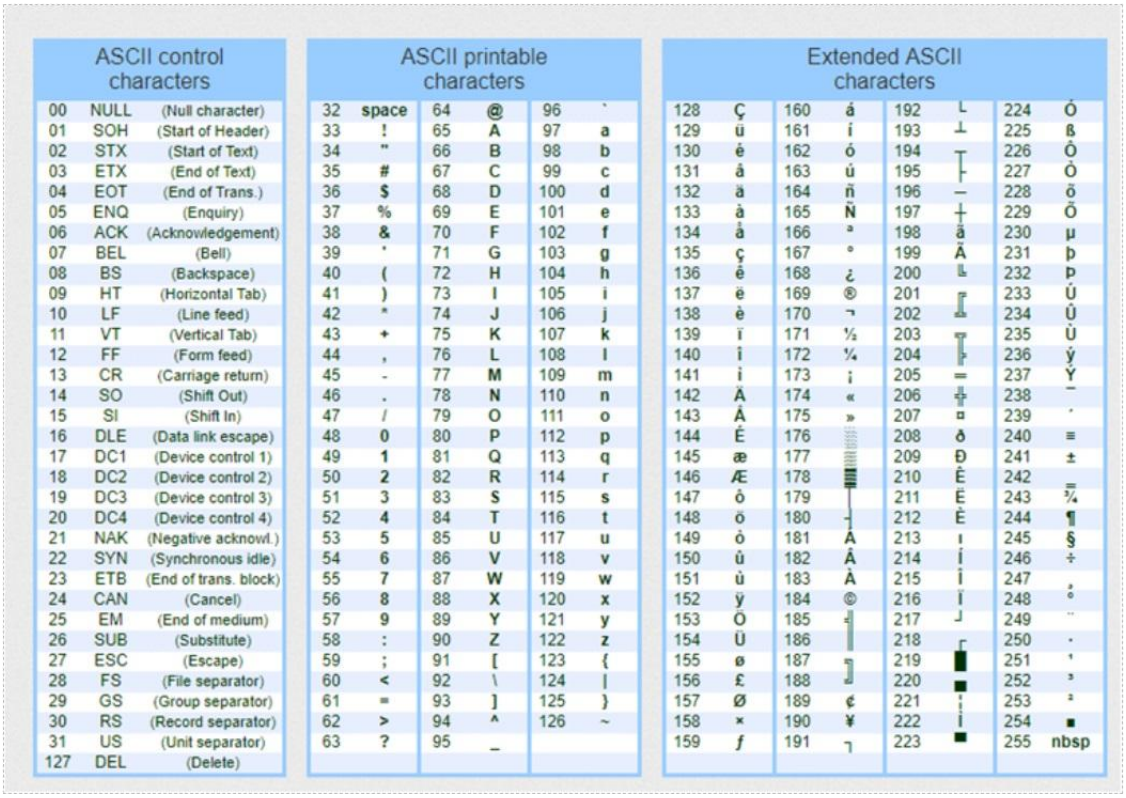
Dado um número de bits m, se a variável *cond* for $True$, geramos p e q aleatoriamente com a quantidade de bits especificada, usando a função $number.getPrime()$, da biblioteca $PyCrypto$. Caso $cond = False$, usamos p e q fornecidos (útil para testes futuros). Uma vez tendo p e q, calculamos n e $\phi(n)$ e continuamos o algortimo padrão: escolhe-se um número e aleatório entre 2 e 2^{2000}, de modo que $mdc(e, \phi(n)) = 1$, e daí calcula-se seu inverso d módulo $\phi(n)$. A partir disso, temos a chave pública (n, e) e a chave privada, d. Em python, temos o seguinte código:

```python
def gera_chaves(m, cond, p, q):
    if cond:
        p = 0
        q = 0
        while (p == q):
            p = number.getPrime(m)
            q = number.getPrime(m)

    n = p * q
    phi = (p - 1) * (q - 1)

    e = random.randint(2, 2^2000)
    while True:
        if math.gcd(e, phi) == 1:
            break
        e += 1

    d = pow(e, -1, phi)

    chave_pub = (n, e)

    return {"chave_pub": chave_pub, "chave_priv": d}
```

Figura 1: Gerador de chaves

3 Criptografador

Por outro lado, queremos ser capazes de criptografar mensagens de texto em geral e não apenas números. Para isso, dada uma mensagem com k caracteres, associamos cada um deles ao seu valor numérico na tabela ASCII, conforme exemplo a seguir:



ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	▒	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	▒	251	¹
156	£	188	╝	220	▒	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▒	255	nbsp

Figura 2: Exemplo de tabela ASCII

Uma vez que cada caractere j tem um número associado $f(j)$, podemos então criptografar $f(j)$ usando o algoritmo RSA padrão. Assim, ao ler uma mensagem m, associamos cada caractere ao seu valor numérico e criamos uma sequência de números criptografados com a chave correspondente, separados por _, conforme o algoritmo abaixo:

1. Recebe uma mensagem (msg) e a chave pública do RSA (n e e);
2. Separa a mensagem pelos caracteres e intera sobre cada caractere na mensagem;
3. Converte o caractere para o valor ASCII usando $ord(carac)$;
4. Eleva o valor ASCII à potência e e calcula o resto da divisão pelo módulo n;
5. Une tudo em uma lista msg_crip, junta os resultados da lista, separando-os com "_"e retorna esse valor.

Em python, temos a seguinte função:

```python
def criptografa(msg, n, e):
    msg_crip = []
    for carac in msg:
        msg_crip.append(str( pow(ord(carac), e, n) ) )
    return "_".join(msg_crip)
```

Figura 3: Função de criptografia

4 Descriptografador

Ao receber uma mensagem criptografada, separamos os números correpondes a cada caractere e utilizamos a chave privada para descriptografar cada um deles e depois associá-los aos seus respectivos caracteres, conforme o algoritmo a seguir:

1. Recebe uma mensagem criptografada (msg_crip) o n e a chave privada d;
2. Usa $re.split('\_', msg\_crip)$ para obter uma lista de valores criptografados como strings;
3. Converte cada caractere para inteiro e calcula o valor original usando a chave privada d e o módulo n;
4. Então, converte os valores resultantes de volta para caracteres usando $chr(int(carac)^{**}d \,\%\, n)$.
5. Junta os caracteres resultantes em uma mensagem e a retorna.

Em python, temos a seguinte função:

```python
def descriptografa(msg_crip, n, d):
    msg_crip = re.split('_', msg_crip)
    msg_desc = []
    for carac in msg_crip:
        msg_desc.append( chr( pow(int(carac), d, n) )  )
    return "".join(msg_desc)
```

Figura 4: Função de descriptografia

5 Enviar email

Tendo esse mecanismo de criptografia, podemos então criar chaves para destinatários e ser capaz de enviar mensagens criptografas por email, por exemplo. O código a seguir demonstra como enviar um e-mail usando o protocolo SMTP (Simple Mail Transfer Protocol) com autenticação SSL. Depois da definição das credencias (remetente, senha e destinatário) e das configurações do e-mail (assunto e corpo), é feita uma conexão SSL segura com o servidor SMTP do Gmail e feito um login no servidor SMTP com as credenciais para, então, enviar um e-mail utilizando o método sendmail e passando o remetente, destinatário e a mensagem formatada como uma string.

Em python, criamos a função abaixo:

```python
def enviar_email(meu_email, senha, email_dest, assunto, texto):
    em = EmailMessage()
    em['From'] = meu_email
    em['To'] = email_dest
    em['subject'] = assunto
    em.set_content(texto)

    context = ssl.create_default_context()

    with smtplib.SMTP_SSL('smtp.gmail.com', 465, context=context) as smtp:
        smtp.login(meu_email, senha)
        smtp.sendmail(meu_email, email_dest, em.as_string())
```

Figura 5: Função para enviar email

6 Exemplo

Geramos um exemplo com primos com 2000 bits:

```python
chaves = gera_chaves(2000, True, 1, 1)
print("Chave Pública: ", chaves["chave_pub"])
print("Chave Privada: ", chaves["chave_priv"])
```

```
Chave Pública:  (725237103450510881856785036858414736055767541663961054720736771391449446816008188262935768022651822181876159183718991022862551159408275391643671765352338862799
Chave Privada:  583768996314453530049233549057595880373913943642553845456762266043646146967057119812116335012357812214553665147881914043079726843942031308197315110771271587565
```

Figura 6: Geração de chaves

Mensagem de teste:

```
assunto = "Teste 1"
msg = "Funcionou"

msg_crip = criptografa(msg, chaves["chave_pub"][0], chaves["chave_pub"][1])

meu_email = "ezequiel.braga.santos@gmail.com"

enviar_email(meu_email, senha, "ezequielsantos2306@gmail.com", assunto, msg_crip)
```

Figura 7: Mensagem de teste

No email, temos

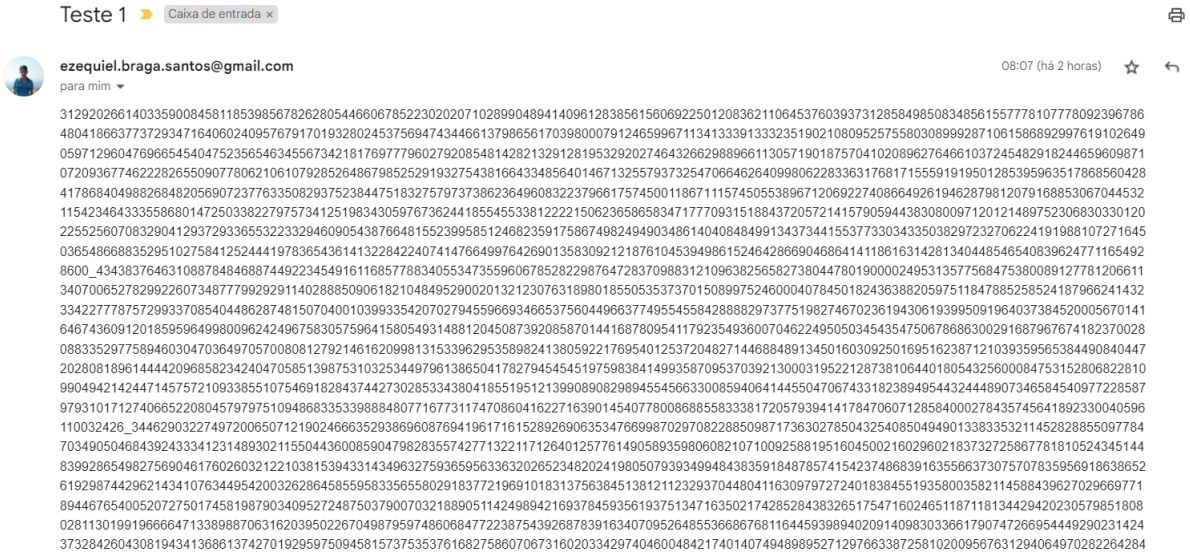


Figura 8: Mensagem no email

Por fim, a mensagem descriptografada:

```
print(descriptografa(msg_crip, chaves["chave_pub"][0], chaves["chave_priv"]))

Funcionou
```

Figura 9: Mensagem descriptografada

Observação final: este sistema ainda é frágil por possuir um conjunto de mensagens pequeno, uma vez que o intervalo de números da tabela ASCII é pequeno. Para tornar mais seguro, deveríamos criar uma identificação de caractere de modo que o intervalo seja maior, para dificultar a testagem de possíveis mensagens m.

Referências

GeeksforGeeks (s.d.). *RSA Algorithm in Cryptography*. https://www.geeksforgeeks.org/rsa-algorithm-cryptography/.

Wikipedia (s.d.). *RSA (cryptosystem)*. https://en.wikipedia.org/wiki/RSA_(cryptosystem).