Enviando mensagens criptografadas por email

Equipe:

- Ezequiel Braga;
- Darlan Araújo.

In [ ]:
```
import random
import math
import re
from email.message import EmailMessage
from senha import senha
import ssl
import smtplib
from Crypto.Util import number
```

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 $c^d (mod(n))$.

Gerador de chaves

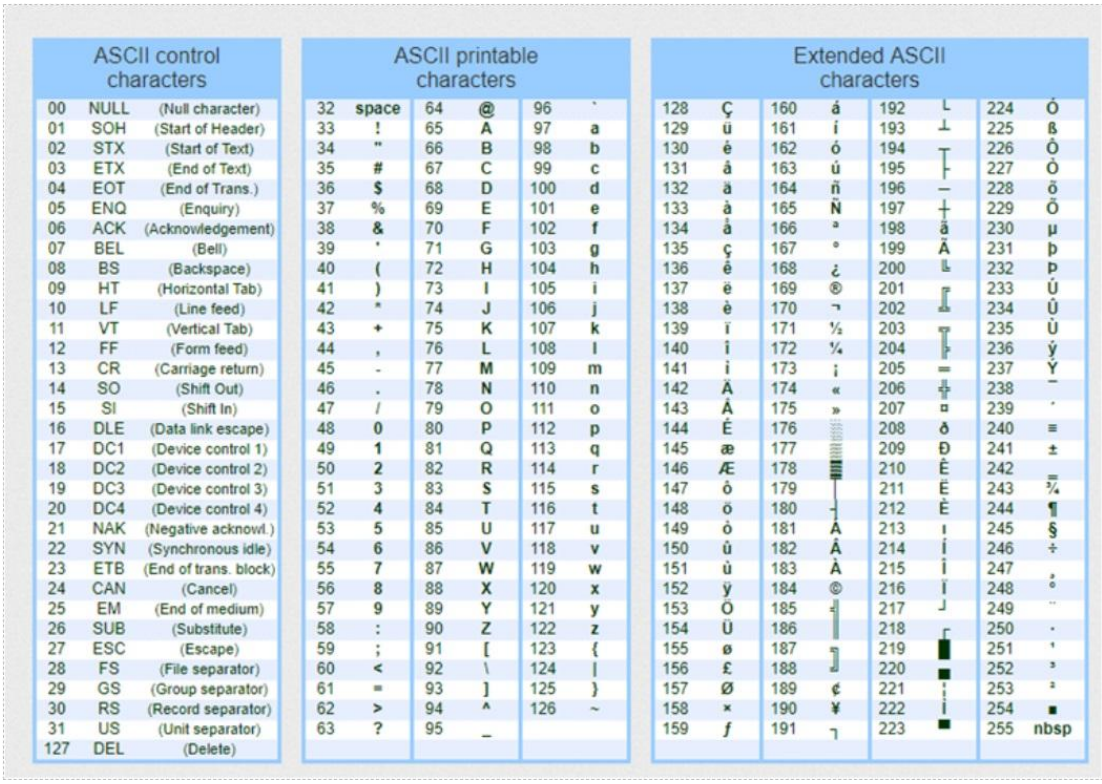
Coml visto anteriormente, para a utilização do RSA, cada usuário precisa de um par de chaves. A função a seguir é responsável por criar tais chaves:

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.

In [ ]:
```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}
```

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 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

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.

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

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.

In [ ]:
```
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)
```

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.

In [ ]:
```
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())
```

Exemplo

Geramos uma chave aleatória de modo que cada primo tenha 2000 bits:

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

```
Chave Pública:  (85493967143774283024141020895801841115083015019300519677700328
10780487216810999597681847872769320330835466134099874497115346487829357089960009
34342596792071209124235812358134627836475011758339197077389708809024616030320048
36408996106731517948467006407132833627827548006805391420358386761800039126793002
45026954426738854529392623840661117037012873315155875302239608176042044171183337
51102271419597019571708362522932936902874115202810479616589789498727702468297400
23403288819348409901258907740535495210615102711309140725962323667154700889004746
46001269501521738781981555358970563920562944535539064335531697244742652338629530
06779281042839863897970667021939903242940225137742789924823156894036934790710310
04354346735445757038394357187367162136122249030332735031578151045595377756794712
12656059164770081930210865490031337083399840153045737102813373013616641073635481
12785618675358314061190447339747298656735494481427991102974181529163025202038318
89041252859824049337051926908243883625316692557798713399610824465808604920595555
51337609196953074674887106475483387173423866997279020362971901755569509921435501
10842128272275227881398673222421065855330417756500315632885762327052265394129242
28620153472693502277567516081175421 3, 361)
Chave Privada:  61337776981267421892666272609453398473148201911354112455746218 7
80945877815901631462592758976378226196298773649082202070716751810188462224367 2
93482896652256043301304027721796914394654516493347702023377689239262480660748 6
82269528854389560904513979486077670626241964924171644816421666795185876283639 7
48808343394054385681791954978704413091034188050564313110250928465232394469154 9
27021836500709918810871631397219685812864150624041945171655276459015926296094 3
64583152471225433917619310382235270347626360171271654980676570165631766011779 2
82945396146651810097873208808126760538110876307607137476529351422172695412156 2
10442722664078965692296284562328246790423606699348240004765370056961211701283 4
89457681004036522248471489028590864435905479400587799341882573865560393154057 1
62179565525856081743207510540838456341046933056236808389378301470718703448246 2
99747159793294681677936107102217490966288657144642394739300995083074564849273 5
16855092925835598033018497467423068192414299975429696075074741514964513036748 8
96285963331589565358170464033558003794129829870179956103795019923027600351718 3
22843227263584874134028613574238937806463937239363254711497462030062658976042 2
52847701735459285361822450263854553
```

Escrevemos uma mensagem de teste, criptografamos e enviamos ao destinatário:

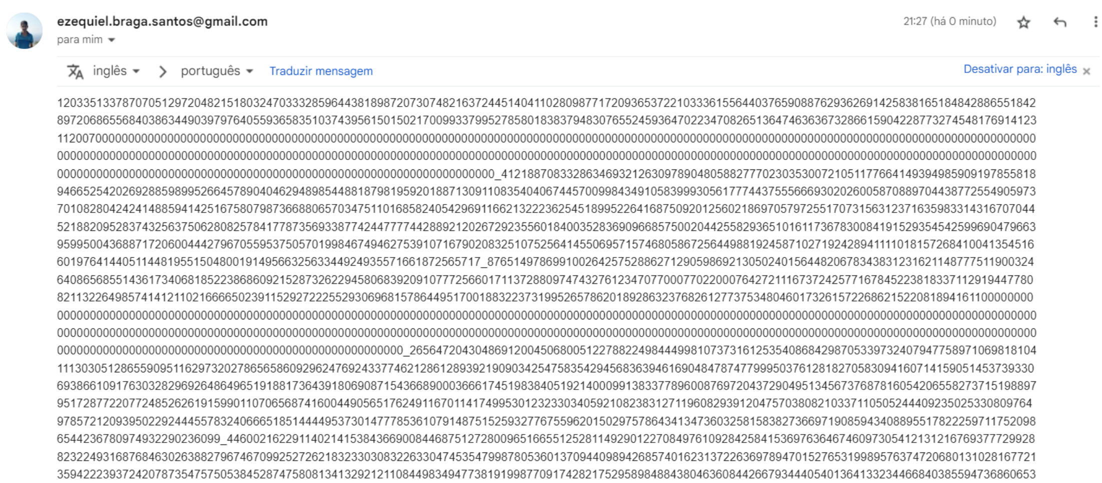
In [ ]:
```
assunto = "Teste"
msg = "Funcionou"

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
```

In [ ]:
```
print(msg_crip)
```

```
1203351337870705129720482151803247033328596443818987207307482163724451404110280
9877172093653722103336155644037659088762936269142583816518484288655184289720686
5568403863449039797640559365835103743956150150217009933799527858018383794830765
5245936470223470826513647463636732866159042287732745481769141231120070000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000_4121887083328634693212630978904805882777023
0353007210511776641493949859091978558189466525420269288598995266457890404629489
8544881879819592018871309110835404067445700998434910583999305617774437555666930
2026005870889704438772554905973701082804242414885941425167580798736688065703475
1101685824054296911662132223625451899522641687509201256021869705797255170731563
1237163598331431670704452188209528374325637506280825784177873569338774244777744
2889212026729235560184003528369096685750020442558293651016117367830084191529354
5425996904796639599500436887172060044427967055953750570199846749462753910716790
2083251075256414550695715746805867256449881924587102719242894111101815726841004
1354516601976414405114481955150480019149566325633449249355716618725657 17_876514
9786991002642575288627129059869213050240156448206783438312316211487775119003246
4086568551436173406818522386860921528732622945806839209107772566017113728809747
4327612347077000770220007642721116737242577167845223818337112919447780821132264
9857414121102166665023911529272225529306968157864495170018832237319952657862018
9286323768261277375348046017326157226862152208189416110000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000_265647204304869120045068005122788224984449981073731612535 4
0868429870533973240794775897106981810411130305128655909511629732027865658609296
2476924337746212861289392190903425475835429456836394616904847874779995037612818
2705830941607141590514537393306938661091763032829692648649651918817364391806908
7154366890003666174519838405192140009913833778960087697204372904951345673768781
6054206558273715198897951728772207724852626191599011070656874160044905651762491
1670114174995301232330340592108238312711960829391204757038082103371105052444092
3502533080976497857212093950229244455783240666518514444953730147778536107914875
1525932776755962015029757864341347360325815838273669719085943408895517822259711
752098654423678097493229023609 9_4460021622911402141538436690084468751272800965 1
6655125281149290122708497610928425841536976364674609730541213121676937772992882
3224931687684630263882796746709925272621832330308322633047453547998780536013709
4409894268574016231372263697894701527653199895763747206801310281677213594222393
7242078735475750538452874758081341329212110844983494773819199877091742821752958
9848843804636084426679344405401364133234466840385594736860653729575639246813022
7819342827722310742859962267997697970470617269057679464760738905223247011330367
4417197939840734992961822617529784666772879544790139307610574892621428374483250
5783924770679137698210541778068649988848642649635308095171321896749750256758214
1309393767639179806394622573861852288246154785156 25_2299298425810292940978545 42
4982357476746205198145429973190379780853771534065572905968431872937291113309430
2700959821124968157658915106508059766010669274491941435655226990162240560538895
3331126511203996033392936580312570717655853606055084404776242342134499082592540
6533873216244891504974232008894340770430207385802714689545572124231014658669511
4356888122693033723813285517864948138136793818386544317292442779647230965432169
3920317627730556970613662241853346421480120096550987272998998568912580790724152
4947690964769363077474176469984170026887106155904216810299334887886725946664029
4742258448196988623248992998045455718060073141030488237817013853692271402649811
3027013972393932398271273242177264557606541476711959003386889335300842126093771
1_8765149786991002642575288627129059869213050240156448206783438312316211487775 1
1900324640865685514361734068185223868609215287326229458068392091077725660171137
2880974743276123470770007702200076427211167372425771678452238183371129194477808
2113226498574141211021666650239115292722255293069681578644951700188322373199526
5786201892863237682612773753480460173261572268621522081894161100000000000000000
```

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000_22992984258102929409785454249823574767462051981454
2997319037978085377153406557290596843187293729111330943027009598211249681576589
1510650805976601066927449194143565522699016224056053889533311265112039960333929
3658031257071765585360605508440477624234213449908259254065338732162448915049742
3200889434077043020738580271468954557212423101465866951143568881226930337238132
8551786494813813679381838654431729244277964723096543216939203176277305569706136
6224185334642148012009655098727299899856891258079072415249476909647693630774741
7646998417002688710615590421681029933488788672594666402947422584481969886232489
9299804545571806007314103048823781701385369227140264981130270139723939323982712
732421772645576065414767119590033868893353008421260937711_412188708332863469321
2630978904805882777023035300721051177664149394985909197855818946652542026928859
8995266457890404629489854488187981959201887130911083540406744570099843491058399
9305617774437555666930202600587088970443877255490597370108280424241488594142516
7580798736688065703475110168582405429691166213222362545189952264168750920125602
1869705797255170731563123716359833143167070445218820952837432563750628082578417
7873569338774244777744288921202672923556018400352836909668575002044255829365101
6117367830084191529354542599690479663959950043688717206004442796705595375057019
9846749462753910716790208325107525641455069571574680586725644988192458710271924
2894111101815726841004135451660197641440511448195515048001914956632563344924935
571661872565717
```

O que resulta na seguinte mensagem criptografada:



Para descriptografar, basta usar a chave privada:

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

```
Funcionou
```

Exemplos frágeis (mais detalhes com Nicole e Wendell)

Exemplo 1: Mesma mensagem para dois destinatários, com mesmo n e e diferentes.

Geramos duas chaves com os mesmos p e q:

In [ ]:
```
p = 637669105899438488175166847640509851379106219477321646294753884090465223147 9
q = 799637232265547981274262922021105133642272366148307382064978461711588143210 1

chaves_11 = {'chave_pub': (509903958942673606831215221020596488980331571676540 50
  473), 'chave_priv': 157391074007675151368620343063439931059468096120031529405 6

chaves_12 = {'chave_pub': (509903958942673606831215221020596488980331571676540 50
  941),
 'chave_priv': 255222916750264897786931316791393141668157460424708370689788334 08
```

Criamos uma mensagem qualquer e criptografamos com cada uma das chaves públicas:

In [ ]:
```
assunto_1 = "Mesmo n, mesma mensagem"
msg_1 = "duvido você descobrir"

msg_11 = criptografa(msg_1, chaves_11["chave_pub"][0], chaves_11["chave_pub"][1]
msg_12 = criptografa(msg_1, chaves_12["chave_pub"][0], chaves_12["chave_pub"][1]
```

Enviamos as mensagens para o ataque:

In [ ]:
```
meu_email = "ezequiel.braga.santos@gmail.com"
email_nicole = 'nicoledossantosouza@gmail.com'

texto_11 = "Mensagem 1: \n" + msg_11 + "\n\n" + "Chave pública 1: \n" + str(chav
texto_12 = "Mensagem 2: \n" + msg_12 + "\n\n" + "Chave pública 2: \n" + str(chav

texto_1 = texto_11 + "\n\n ---------------------------------------------------- \

enviar_email(meu_email, senha, email_nicole, assunto_1, texto_1)
```

Exemplo 2: Mesma mensagem para três destinatários, n diferentes, mas mesmo $e = 3$.

Criamos uma função que gere chaves com mesmo e:

In [ ]:
```
def gera_chaves_mesmo_e(m, e):
    p = 0
    q = 0
    n = 0
    phi = 0

    while ((math.gcd(e, phi) != 1) | (p == q)):
        p = number.getPrime(m)
        q = number.getPrime(m)
        n = p * q
        phi = (p - 1) * (q - 1)

    d = pow(e, -1, phi)

    chave_pub = (n, e)
    chave_priv = d
    return {"chave_pub": chave_pub, "chave_priv": chave_priv}
```

Geramos 3 chaves com o mesmo e:

In [ ]:
```
chaves_21 = {'chave_pub': (704096126886104387811574493196160443189833330686666323),
 'chave_priv': 469397417924069591874382995464106962126555553791110884904573607633
chaves_22 = {'chave_pub': (719193398591625165104116206607077848665593404170440033),
 'chave_priv': 479462265727750110069410804404718565777062269446960021674416467213
chaves_23 = {'chave_pub': (768640344073122548813287417908642088163627143103999223),
 'chave_priv': 512426896048748365875524945272428058775751428735999482376202120393
```

Criptografamos uma mensagem com cada uma das chaves:

In [ ]:
```
assunto_2 = "Mesmo e, n diferentes"
msg_2 = "agora esse é impossível"

msg_21 = criptografa(msg_2, chaves_21["chave_pub"][0], chaves_21["chave_pub"][1]
msg_22 = criptografa(msg_2, chaves_22["chave_pub"][0], chaves_22["chave_pub"][1]
msg_23 = criptografa(msg_2, chaves_23["chave_pub"][0], chaves_23["chave_pub"][1]
```

Enviamos a mensagem para ataque:

In [ ]:
```
texto_21 = "Mensagem 1: \n" + msg_21 + "\n\n" + "Chave pública 1: \n" + str(chav
texto_22 = "Mensagem 2: \n" + msg_22 + "\n\n" + "Chave pública 2: \n" + str(chav
texto_23 = "Mensagem 3: \n" + msg_23 + "\n\n" + "Chave pública 3: \n" + str(chav

texto_2 = texto_21 + "\n\n ---------------------------------------------------- \

enviar_email(meu_email, senha, email_nicole, assunto_2, texto_2)
```

Exemplo 3 (esse não vai ser quebrado, ainda bem!): Tentativa de fatorar o n.

Geramos uma chave aleatória:

In [ ]:
```
chaves_3 = {'chave_pub': (663496675079299035075348441602063535760386791431749353959),
 'chave_priv': 459397072213404128561033748929895920484772502096643973670064013222
```

Criptogramos e enviamos para o ataque:

In [ ]:
```
assunto_3 = "Chave aleatória"
msg_3 = "álgebra é muito legal"

msg_31 = criptografa(msg_3, chaves_3["chave_pub"][0], chaves_3["chave_pub"][1])

texto_3 = "Mensagem: \n" + msg_31 + "\n\n" + "Chave pública: \n" + str(chaves_3[

enviar_email(meu_email, senha, email_nicole, assunto_3, texto_3)
```

Aguardem os resultados!

Referências:

- GeeksforGeeks. (s.d.). RSA Algorithm in Cryptography. Retirado de https://www.geeksforgeeks.org/rsa-algorithm-cryptography/;
- Wikipedia. (n.d.). RSA (cryptosystem). Retirado de https://en.wikipedia.org/wiki/RSA_(cryptosystem);