EN.540.635 "Software Carpentry"

Weekly Challenge 4 - RSA Encryption

For this assignment, you will generate two functions that use the RSA encryption method, named after Rivest, Shamir, and Adleman. Following this, you will also write a function (as well as several helper functions) that generates RSA encryption key pairs.

In short, the RSA Encryption and Decryption works by using three integers: N, E, and D. These are chosen using some algorithm, but afterwards allow you to encrypt and decrypt messages. Let M be some integer we want to encrypt, we can do so as follows:

$$\textbf{Encrypt: } C \equiv M^E \, mod \, N$$

Now, M is encrypted as C. We can decrypt it back to M as follows:

$$\textbf{Decrypt: } M \equiv C^D \, mod \, N$$

Part 1 -

In class, we went over how RSA encryption can be used to encrypt and decrypt information/messages. An example of the values that can be generated from the RSA encryption algorithm is the following:

$$N = 17947$$
$$E = 7$$
$$D = 10103$$

Using these values, make two functions that will (1) encrypt any string and (2) decrypt an encrypted string. The first function, called "encrypt", will encrypt a string (think of this as password protecting). The second, called "decrypt", will decrypt the given message, allowing you to read it back. Essentially, you should be able to run the following code:

```python
message = "This message will be encrypted"

encrypted_message = encrypt(message, N=17947, E=7)

decrypted_message = decrypt(encrypted_message, N=17947, D=10103)

print("Checking if encryption and decryption worked... "),
print(str(message == decrypted_message))
```

HINTS & NOTES

1. Cast all data types to integers (NOT NUMPY INTEGERS! USE int())
2. Do not use the "np.power()" function (it uses internal variables that lead to buffer overflows).
3. You may use either % or "np.mod()"; however, note that the latter will change your data types to numpy data types, which is an issue.
4. You are welcome to store your encrypted string in any format (for example: a list of encrypted numbers. But there is no one way to do this).

Part 2 -

Make a function to generate the key itself. That is, make a function that will generate for you a valid N, E, and D that you can use with your encrypt and decrypt functions. To get you started, you'll need to write the following functions:

1. generate_key() - A function to call that starts the key generation.
2. get_prime_divisors(N) - A function that finds all prime divisors of a given number that is passed to it (say N).
3. get_primes_in_range(low, high) - A function that finds all prime numbers in the given range between low and high.
4. is_prime(p) - A function that checks if a given number (here p) is prime or not

The algorithm we use to generate our key is as follows (remember: N, E, and D are all integers):

1. $N = P * Q$ where P and Q are large prime numbers (for now let's just say primes > 130).
2. $X = (P - 1)(Q - 1)$
3. Find $E < X$ such that E is relatively prime to X. That is, all the prime divisors of E are not contained in X. Ex. let $X = 24$, it's prime divisors are $[2, 3]$. Thus, from all possible primes less than X we find that E can be comprised of $[5, 7, 11, 13, 17, 19, 23]$.
4. Find D such that $D * E \equiv 1 \, mod \, X$. That is, $D * E = k * X + 1$ where k is some integer.

Please upload a .py file to the appropriate assignment in Blackboard with the following components:

1. Part 1 - A function that can encrypt any string and a function that can decrypt the encrypted message.
2. Part 2 - The four listed functions that are used to generate the key.
3. In your main block, use the generate_key() function to generate keys and then use those keys to encrypt a string and decrypt it back to its original string.

Background (for whoever is interested):

For a quick background, modular arithmatic simply finds the smallest integer remainder of a number N when divided by M. That is, take the following example:

$$R \equiv 44 \, mod \, 7$$

Here, we simply want to find R such that $44 = 7 * k + R$, where k is the largest possible integer for which R is positive. In this example:

$$\begin{aligned} 44 =& 7 * k + R \\ 44 =& 7 * 5 + 9 \\ 44 =& 7 * 6 + 2 \\ 44 =& 7 * 7 - 5 \end{aligned}$$

Thus, $R = 2$ and $44 \, mod \, 7 \equiv 2$. Note the use of the $\equiv$ symbol. This simply means that the two values are *congruent* (that is, the remainder of 44 / 7 is the same as the remainder of 2 / 7). A good way to understand this is to recognize that $1 \, mod \, 10$, $11 \, mod \, 10$, $21 \, mod \, 10$, and $31 \, mod \, 10$ are all equivalent (as the remainder being 1 should be self evident from our decimal system). It is common for the expression $a \equiv b \, mod \, n$ to be such that $b < n$ (as there are a multitude of valid answers here).

Now, the secret behind the RSA encryption is that integers E and D can be chosen in such a way that the following is true:

$$M^{ED} \, mod \, N \equiv (M^E)^D \, mod \, N \equiv (M^D)^E \, mod \, N \equiv M \, mod \, N$$

Why do we care about this? Well, looking back at our encryption, we know that $M^E \equiv C \, mod \, N$. If we want to decrypt it, it would be nice if we had some D such that $C^D \equiv M \, mod \, N$. That is, $(M^E)^D \equiv M \, mod \, N$. Now, all we need to do is show that $(M^E)^D \equiv M \, mod \, N$. Let's first generate N and select an E and D as per the algorithm described above. From this, let's show $(M^E)^D \equiv M \, mod \, N$.

$$\begin{aligned}(M^E)^D \equiv & M \, mod \, N \\ M^{(D*E)} \equiv & M \, mod \, N \\ M^{(k*X+1)} \equiv & M \, mod \, N \\ M^{(k*(P-1)(Q-1)+1)} \equiv & M \, mod \, N\end{aligned}$$

Now, before we continue we need to acknowledge "Fermat's little theorem". That is, that $a^{(p-1)} \equiv 1 \, mod \, p$ if p is prime (don't worry about proving this, it's a well established theorem so we will just accept it for now). We find the following:

$$\begin{aligned}M^{(k*(P-1)(Q-1)+1)} \equiv & M \, mod \, N \\ (M^{(P-1)})^{(k(Q-1)+1)} \equiv & M \, mod \, N \\ (M^{(P-1)})^{k(Q-1)} \cdot M \equiv & M \, mod \, N\end{aligned}$$

To show this is true, we first discern if $(M^{(P-1)})^{k(Q-1)} \cdot M \equiv M \, mod \, P$ is true. This is done as follows:

$$\begin{aligned}(M^{(P-1)})^{k(Q-1)} \cdot M \equiv & M \, mod \, P \\ (1 \, mod \, P)^{k(Q-1)} \cdot M \equiv & M \, mod \, P \\ 1 \cdot M \equiv & M \, mod \, P \\ M \equiv & M \, mod \, P\end{aligned}$$

It can be seen that an equivalent proof can show that $(M^{(P-1)})^{k(Q-1)} \cdot M \equiv M \, mod \, Q$. Now all that's left is to take advantage of a property in modular arithmetic! That is, if we show that $a \equiv b \, mod \, x$ and $a \equiv b \, mod \, y$, then $a \equiv b \, mod \, (x * y)$. In this case, we have shown that $(M^{(P-1)})^{k(Q-1)} \cdot M \equiv M \, mod \, P$ and $(M^{(P-1)})^{k(Q-1)} \cdot M \equiv M \, mod \, Q$. Thus, we have shown that $(M^{(P-1)})^{k(Q-1)} \cdot M \equiv M \, mod \, N$. Therefore:

$$\begin{aligned}M \equiv & M \, mod \, (P * Q) \\ M \equiv & M \, mod \, (N)\end{aligned}$$

As we have just gone through a derivation converting the left hand side from $(M^E)^D$ to M, we find:

$$M^{ED} \equiv M \, mod \, N$$

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Figure 1: ASCII Table for Reference