JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

EN.540.635
Software Carpentry

Lecture 5
Modular Arithmetic | RSA Encryption | Exceptions

Encryption

- Use a key to encode your data so that only those with the keys can decode them. The encoded data is then impossible to comprehend/read unless it is decoded.

- A “secret” encryption key, produced using algorithms can unscramble the data and is only in the possession of the user and the recipient

RSA Encryption

- It is a type of asymmetric encryption and uses 2 different keys

- Implementation:
  - One key you can use to encode a message
  - One key you can use to decode a message
  - Make it such that if key 1 encodes, key 2 decodes, and the other way around.
  - Keep key 1 private and secure (we will call this your **private key**)
  - Share key 2 with websites/servers you trust (we will call this your **public key**)

RSA Encryption

Goal:

Devise a key that encodes and decodes a message.

Challenges:

1. What does encoding and decoding actually mean?
    - Turning letters and symbols into numbers. How?
2. What mathematical property can we take advantage of?

The easiest way of handling problem 1 is to simply enumerate the characters and symbols. This has already been done and is known as ASCII encoding!

ASCII

test.py

```python
letter = "X"

letter_as_number = ord(letter)
number_from_letter = chr(letter_as_number)

print("I have encoded '%s' into '%d' and returned it as '%s'."
      % (letter, letter_as_number, number_from_letter))
```

```
I have encoded 'X' into '88' and returned it as 'X'.
[Finished in 0.1s]
```

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

ord() will return the unicode code of a unicode character
chr() will return the unicode character of a unicode code

Ref : https://www.vertex42.com/ExcelTips/unicode-symbols.html

Modular Arithmetic

$$a = q * b + r$$

- In python we can get the remainder using the '%'.

$$26\%5 \qquad a \bmod b = r \qquad 26 \bmod 5 = 1$$

- Congruence Modulo :

$$a - b = km \quad \text{written as} \quad a \equiv b \pmod{m}$$

- Congruence Modulo asserts that 'a' and 'b' have same remainder when / by m

$$a \bmod m = \mathrm{b} \bmod \mathrm{m}$$

https://en.wikipedia.org/wiki/Modular_arithmetic#Congruence

RSA Encryption

JOHNS HOPKINS WHITING SCHOOL of ENGINEERING

To encrypt a message in RSA do the following operation :

$$C = M^E \bmod N$$

To decrypt a message in RSA do the following operation :

$$M = C^D \bmod N$$

We can simplify and write as follows:

$$(M^E)^D = M \; mod \; N$$

(E, N) -> Public key

(D, N) -> Private key

Challenge : How to find E,D,N such that the above equation holds ?

RSA Encryption: The Keys

Step 1 : Calculate N : $N = P \times Q$

- Choose large P & Q. Can only encode ASCII character encoding < N

Step 2 : Find number of co-primes to N

$$\phi(N) = \phi(P) * \phi(Q) = (P - 1) * (Q - 1)$$

Euler's ϕ function

Step 3 : Find E such that is co-prime to $\boldsymbol{\phi(N)}$

$$GCD\big(E, \phi(N)\big) = 1$$

Step 4 : To find D we want the following condition to be satisfied :

$$D * E = k * \phi(N) + 1 \qquad D * E \bmod \phi(N) = 1$$

For some integer 'k'

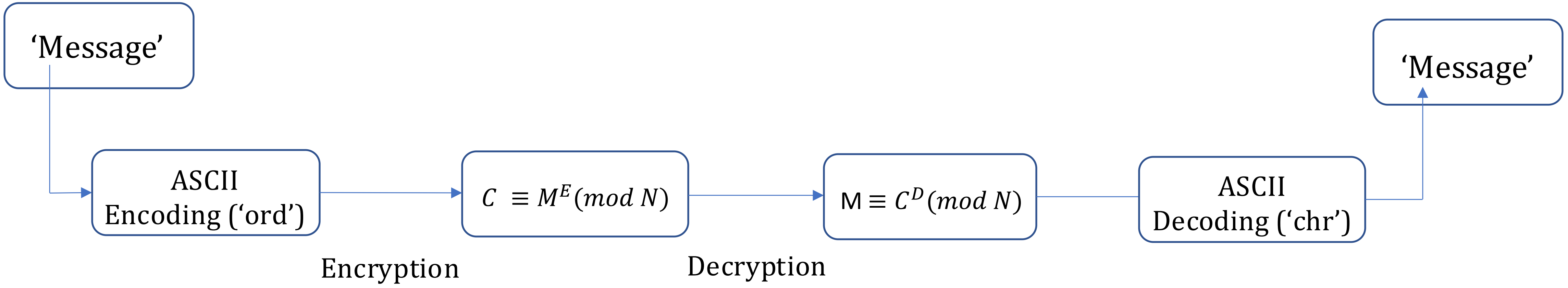
https://www.cryptool.org/en/cto/rsa-step-by-step/

RSA Encryption

If we take M^E or M^D, we can see that this value will be incredibly large.

- To handle this, and to make it even harder to brute force, let's also have an N that is incredibly large. We can generate N as the multiple of two large prime numbers, P and Q

For a particular N, E, D



RSA Encryption: Example

Let M be a letter to encrypt: $M = \text{ord}(\text{‘x’}) = 120$

Encryption method: $C = M^E \% N$

Decryption method: $M = C^D \% N$

Ex.

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

M = 120 encodes to C = 11262

We get M back from C by $11262^{10103} \% 17947$

RSA Encryption: Why it works

How would we try to brute force 11262?

Since the N, E are made public so you can encode any message like the sender but to decode it we need D which comes from knowing $\phi(N)$ which comes from knowing P and Q (which are kept private).

Essentially the task to perform is prime factorization of N which is an extremely difficult task for large P and Q.

Further, how do we know it worked? Many different numbers could give us alternative answers (such as D = 186 would give C = 121, which is the letter ‘y’).

Thus, the ‘attacker’ would have to decode every possible message to some reasonable upper bound of D and read EACH ONE to make sure it makes sense.

Syntax Errors vs Exceptions

- SYNTAX ERRRORS:
  - When the python compiler encounters a wrong statement while parsing through a script and therefore cannot be executed

```
Traceback (most recent call last):
  File "C:\Users\divya\Downloads\temp.py", line 1, in <module>
    for i in rang(1000):
NameError: name 'rang' is not defined
```

- EXCEPTIONS:
  - Our code is syntactically correct
  - Errors caused when an attempt is made to execute it
  - Informs what type of exception that you have run into
  - Built-in exceptions as well as self-defined exceptions

```
>>> print(0/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

```
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
```

Exception Handling

```python
def reciprocal(num):
    return 1 / num


for i in range(-10, 11):
    try:
        print(reciprocal(i))
    except ZeroDivisionError as e:
        print(e)
        print("Can't divide by zero you dumb-dumb")
```

```
-0.1
-0.1111111111111111
-0.125
-0.14285714285714285
-0.16666666666666666
-0.2
-0.25
-0.3333333333333333
-0.5
-1.0
division by zero
Can't divide by zero
1.0
0.5
0.3333333333333333
0.25
0.2
0.16666666666666666
0.14285714285714285
0.125
0.1111111111111111
0.1
[Finished in 0.3s]
```

- The ‘try-except’ block can be used to catch and handle select exceptions

- It is not good programming practice to use an all encompassing ‘except’ clause as it may bypass a critical issue in the logic of your code

‘raise’ and ‘assert’

‘raise’

- Manually throw/raise an exception if a defined condition is met
  - Avoid throwing generic exceptions
  - Try to be as specific as possible

```
import math


def circle_area(radius):
    if radius < 0:
        raise AssertionError('Circle radius cannot be negative')
    return math.pi * (radius ** 2)


if __name__ == '__main__':
    print(circle_area(-4))
```

‘assert’

- Create debug messages when a condition is not specified
  - Throws ‘AssertionError’
  - Useful for debugging

```
import math


def circle_area(radius):
    assert radius >= 0, "Circle radius cannot be negative"
    return math.pi * (radius ** 2)


if __name__ == '__main__':
    print(circle_area(4))
```

https://docs.python.org/3/tutorial/errors.html