M1C
An Introduction To Python

Phil Ramsden

The RSA Cryptosystem

This sheet, together with all the other resources you'll need, are available in Blackboard, at

`http://learn.imperial.ac.uk/`

Cryptography summary

A secret code or **cipher** consists of an **encryption algorithm** (which converts the message to its secret form) and a **decryption algorithm** (which converts it back again). The unencrypted message is called the **plaintext** and the encrypted version is called the **ciphertext**.

Quite often, the general type of cipher being used is public knowledge, and encryption and decryption each depend on knowing a small but crucial additional piece of information, known as the **key**. For example, among the oldest types of secret communication system is the **Caesar cipher**, an example of which is shown below.

plain	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**
cipher	L	M	N	O	P	Q	R	S	T	U	V	W	X

plain	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
cipher	Y	Z	A	B	C	D	E	F	G	H	I	J	K

The role of mathematics

Many ciphers can be represented mathematically. To do this, we replace every letter by a number, so that the Caesar cipher above becomes

plain	**00**	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**
cipher	11	12	13	14	15	16	17	18	19	20	21	22	23

plain	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**
cipher	24	25	00	01	02	03	04	05	06	07	08	09	10

(confusingly, replacing letters by numbers in this way is often called **encoding**.)

Then, if p is the plaintext and c is the ciphertext, the encryption algorithm can simply be written

$$c = p + 11 \mod 26,$$

We say that the modulus is 26 and the **encryption key** is 11.

The **decryption key** is 15. That's because the decryption algorithm can be written

$$p = c + 15 \mod 26.$$

Can you see why?

This is a crude cryptosystem, but one of the key modern cryptosystems, RSA, is based on more or less exactly the same idea, with a few tweaks:

- We use a larger character set: capital and lower-case letters, numbers, punctuation, special characters. About a hundred in all should be fine.
- We encrypt not single characters, but blocks of many (what's called a **Block cipher**). This makes things much more secure, because it means the enemy can't just try one encryption key after another (a variety of what's known as "brute force" cryptanalysis).
- We don't use modular *addition*, but something else. More about why a little later.

A ninety-five-character numerical encoding

For this project, the numerical encoding at the end of this handout is *required* (just so we're all using the same system).

Block Caesar cipher example

Using our numerical encoding, the 16-character text string

```
"My name is Bond."
```

would be encoded as the 32-digit integer

$$p = 45890078657769007383003479786814$$

Now, suppose our encryption algorithm was

$$c = p + e \mod n,$$

where

$$\begin{aligned} e &= 367203625032225669459547639019 9, \\ n &= 995894547254793823589941314865433 0 \end{aligned}$$

(with 31 and 34 digits respectively).

Then,

$$p + e = 49562114908091264077598956177013,$$

and since this is less than n, there's no need to "wrap round", and we have the numerically encoded ciphertext

$$c = 49562114908091264077598956177013.$$

Decryption is now a matter of setting up the decryption key

$$d = n - e = 995527343629761597920481767226413 1.$$

Then

$$c + d = 100048355512057072432824166284411 44,$$

which, reduced mod n, is

$$458900786577690073830034797868 14,$$

which was indeed our plaintext, decoding once again to

```
"My name is Bond."
```

Inverting modular multiplication

As you've seen, the encryption algorithm

$$c = p + e \mod n$$

is inverted by the decryption algorithm

$$p = c + d \mod n,$$

where d has been chosen so that

$$e + d \equiv 0 \pmod{n}.$$

What if we'd used *multiplication* instead of addition? That is, suppose the encryption algorithm is of the form

$$c = p\,e \mod n,$$

for some modulus n and encryption key e? This is called a **linear cipher**.

Let's use the same example as before, with the (numerically encoded) plaintext

$$p = 458900786577690073830034797868 14.$$

Let's also use the same modulus as before,

$$n = 995894547254793823589941314865 4330.$$

Let's make the encryption key fairly small,

$$e = 331.$$

Then, using Python,

$$p\,e = 151896160357215414437741518094 35434.$$

Reducing mod n, we have

$$c = 523067056317360320787473866078110 4$$

If we receive this from our agent in the field, how do we decrypt it? The answer, perhaps surprisingly, is that just as we can undo a modular addition with another modular addition, so we can undo a modular *multiplication* with another modular multiplication. That is, our decryption algorithm is

$$p = c\,d \mod n.$$

But for what value of d? The answer turns out to be

$$d = 192559670768298503654852701363709 1.$$

We have that

$$c\,d = 292490746293540750364109013672869503866755785287593059788036720824 94,$$

which really does reduce mod n to

$$p = 458900786577690073830034797868 14.$$

Now, how did we find this value of d? The answer is that we selected a d such that

$$e\,d \equiv 1 \pmod{n},$$

so that

$$c\,d \equiv (p\,e)\,d \equiv p\,(e\,d) \equiv p \pmod{n}.$$

And how did we select such a d? The answer to *that* is that we used the extended Euclid's algorithm to find the HCF of n and e (which is 1), and the values of λ and μ such that

$$\lambda\,n + \mu\,e = 1.$$

Then set d equal to μ, giving

$$e\,d = 1 - \lambda\,n \equiv 1 \pmod{n}.$$

Very Important: For this to work, we had to choose e, in the first place, *coprime* to n: that is, we had to make sure that $\mathrm{HCF}\,(n, e) = 1$. This is most easily done by making sure that e is prime, but not one of n's prime factors.
Question: What would we have done if the extended Euclid's algorithm had given us a negative d?

The RSA cryptosystem

The RSA cryptosystem works in exactly this way, except that instead of addition (like a Caesar cipher) or multiplication (like a linear cipher) it uses raising to a power: the encryption algorithm is

$$c = p^e \mod n$$

for some modulus n and encryption key e.

As you might by now suspect, the decryption algorithm is

$$p = c^d \mod n$$

for some suitably chosen decryption key d. But how is d chosen? That's the ingenious bit. It works as follows:

- We make sure that our modulus n is the product of two primes: $n = p_1\, p_2$.
- We set $\phi(n) = (p_1 - 1)\,(p_2 - 1)$. **This is the key step.**
- We choose d such that
$$e\, d \equiv 1 \pmod{\phi(n)}.$$
Notice: modulo $\phi(n)$, not modulo n.

We're making use, here, of **Euler's Totient Theorem**, which says that

$$p^{\phi(n)} \equiv 1 \pmod{n},$$

and thus

$$\begin{aligned}(p^e)^d &= p^{e\,d} \\ &= p^{k\,\phi(n)+1} \quad \text{for some integer } k \\ &= (p^{\phi(n)})^k \times p \\ &\equiv 1^k \times p = p \pmod{n}.\end{aligned}$$

Exercise: Encode and encrypt

```
"My name is Bond."
```

using a "toy" RSA cipher with

$$\begin{aligned}n &= 99158679366511727 \times 97369398007206967 \\ &= 9655020917106901547888376921602009, \quad \text{and} \\ e &= 43.\end{aligned}$$

Then find a decryption key, and decrypt.

Discussion question: The RSA cryptosystem is *public key*: that is, the modulus and encryption key are not kept secret. Why is this useful, and how is it possible to make it secure?

STANDARD NUMERICAL ENCODING: YOU MUST USE THIS

There are 95 characters in the standard US English set, which is quite a convenient number (though it would be more convenient if it were 100). Here's the numerical encoding we'll use.

char:	*sp*	!	"	#	$	%	&	'	(	)
num:	**00**	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**

char:	*	+	,	-	.	/	0	1	2	3
num:	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**

char:	4	5	6	7	8	9	:	;	<	=
num:	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**

char:	>	?	@	A	B	C	D	E	F	G
num:	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**

char:	H	I	J	K	L	M	N	O	P	Q
num:	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**

char:	R	S	T	U	V	W	X	Y	Z	[
num:	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**

char:	\	]	^	_	`	a	b	c	d	e
num:	**60**	**61**	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**

char:	f	g	h	i	j	k	l	m	n	o
num:	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**

char:	p	q	r	s	t	u	v	w	x	y
num:	**80**	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**

char:	z	{	|	}	~
num:	**90**	**91**	**92**	**93**	**94**

Note that the encoding for the space character is 00.

You might find the following command helpful:

```
[ord(char)-32 for char in 'My name is Bond']
```