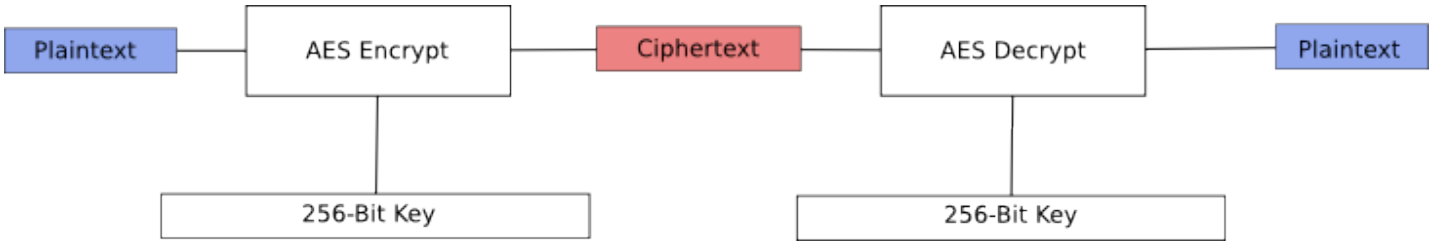
Assignment 1 - AES Partial Key Search

Aims

- Revise some C programming fundamentals, including pointers, file I/O, command line arguments and system calls.
- Develop a parallelisation strategy for a computationally intensive problem
- To explore and implement Interprocess communication using pipes
- To implement a the ring IPC topology

The problem

The *Advanced Encryption Standard (AES)* is a specification for the encryption for digital information. The figure below summarises the process of encrypting and decrypting a plain-text message.



Essentially, this process starts by feeding the plain-text massage into the AES encryption function. This function uses a 256-bit key (which is normally generated using a randomised generation process) to produce a stream of cipher-text, that contains the original message in an encrypted form. The encrypted message can then be transmitted through an un-secure channel, to a destination where the AES decrypt function can be used to recover the plain-text from the cipher-text. The AES decrypt function uses the same 256-bit key, that wae used for the original encryption process, to convert the cipher-text back to the plain-text.

Clearly, the security of this system is reliant on the key that is used to encrypt and decrypt the plain text remaining secret. If a potential attacker is able to get access to a sample of plain-text and its corresponding cipher-text, an ***exhaustive key search*** can be used to determine the key used to produce cipher text from the given plain text. This basically involves trying every possible value for the 256-bit key on the sample of cipher-text until a value is found that produces the corresponding cipher-text. The problem with this approach is that it will require (worst-case) 2^256 encryption operations. The universe will end long before the key is every found. If (through coercion, espionage or a phishing attack) you are able to obtain part of the 256-bit key, the search space can be drastically cut-down resulting in a more achievable ***partial key search***.

In this assignment you will write a multi-process program to carry out a ***partial key search***. The input for

this program will be a stream of cipher-text, a stream of corresponding plain-text and 236 bits of the 256 key that was used for the original encryption.

Your program will use a **ring** of processes to search for the full encryption key used to generate a piece of cipher-text.

Getting Started

For the encryption and decryption, you will need to use the AES encryption functions provided within the OpenSSL libraries (these are already installed on `turing`). The following program uses the OpenSSL functions to encrypt a segment of plain-text that is read is from a text file. The key is read from the command line as an argument.

- generate_ciphertext.c

This program can be compiled using the line:

```
[mwelch8@turing a1]$ gcc generate_ciphertext.c -lcrypto -o generate_ciphertext -Wall
```

Please notice that libcrypto has been included for linking using the `-lcrypto` flag.

The program linked below provides an example for the decryption process.

- decrypt_ciphertext.c

This program reads the cipher-text from a text file and decrypts it using a key that is passed in from a command line argument. A full example of both programs running (please not that your web browser can't display the cipher-text bytes correctly):

```
[mwelch8@turing a1]$ cat plain.txt
This is my super secret text
[mwelch8@turing a1]$ generate_ciphertext 12345678123456781234567812345678 > cipher.txt
[mwelch8@turing a1]$ cat cipher.txt
!\ufffd\ufffd\ufffdh\ufffd\ufffd\ufffd\ufffd.\ufffdd\ufffdE}G_\ufffd\ufffdx=t\ufffd\ufffd
?\ufffdm
                              [mwelch8@turing a1]$
[mwelch8@turing a1]$ decrypt_ciphertext 12345678123456781234567812345678
This is the key: 12345678123456781234567812345678
It is 32 bytes in length
Ciphertext: !\ufffd\ufffd\ufffdh\ufffd\ufffd\ufffd\ufffd.\ufffdd\ufffdE}G_\ufffd\ufffdx=t
\ufffd\ufffd?\ufffdm

Plaintext: This is my super secret text

[mwelch8@turing a1]$
```

The final program contains a single process, single threaded, implementation for a simple key-search program. This program reads in the cipher-text from the cipher.txt file, the plain-text from the plain.txt file and the partial key from the command line. It then sequentially tries all possible variations, using the partial key information that passed in.

- search_keyspace.c

An example of this program in action:

```
[mwelch8@turing a1]$ search_keyspace 12345678123456781234567812345

Plain:This is my super secret text

Cipher:!\ufffd\ufffd\ufffdh\ufffd\ufffd\ufffd\ufffd.\ufffdd\ufffdE}G_\ufffd\ufffdx=t\ufff
d\ufffd?\ufffdm

OK: enc/dec ok for "This is my super secret text"
Key No.:3553080:12345678123456781234567812345678
[mwelch8@turing a1]$
```

In this example, all but 3 bytes (24 bits) of the encryption key was provided resulting in a search of the remaining 16777216 (2 ^ 24) possible keys. The key was found after 3553080 trials.

Your Program

You will implement a multiprocess version of this key-search program that uses pipes arranged using the ring topology (i.e. a ring of processes) for inter-process communication. Your program should take the

number of processes and the partial key as command line arguments and read the supplied plain-text and cipher-text from the file plain.txt and cipher.txt. Your program will need to divide the search space up amongst the processes and execute a partial search in each process.

```
parallel_search_keyspace <num. procs.> <partial key>
```

Example:

```
[mwelch8@turing a1]$ parallel_search_keyspace 5 12345678123456781234567812345

Plain:This is my super secret text

Cipher:!\ufffd\ufffd\ufffdh\ufffd\ufffd\ufffd\ufffd.\ufffdd\ufffdE}G_\ufffd\ufffdx=t\ufff
d\ufffd?\ufffdm

OK: enc/dec ok for "This is my super secret text"
Key No.:3553080:123456781234567812345678 12345678
[mwelch8@turing a1]$
```

Within the ring topology, you will need to develop a system/protocol that communicates the full key (from the process that discovers it) around the ring and back to the parent node. How this is actually done is up to you.

The Resources

- The partial key you are to process is

```
L17hhOCMtHI8L6m67Twgo8Dx7n0jD
```

The last 3 bytes are missing (i.e. the 24 least significant bits)

The cipher-text for your search:

- cipher.txt

The corresponding plain-text for your search:

- plain.txt

Hints

- Review all of the supplied code as most of the hard parts are done for you.
- Review lectures 4 and 5 - you are free to use any code from the lectures in your assignment.

- Make use of the ***Data Display Debugger*** (available on turing) to debug your code - it will allow you to view the contents or variables and easily spot errors.
- The supplied code produced numerous compiler warnings and uses several hardwired constants throughout. You should re-factor the any code you use to remove these.
- Pay attention to the marking scheme!

Submission

- Your assignment will need to be sumbitted through the `submit` program on turing.
- Make sure that you record a script of your program compiling and working correctly.
- Confirm that the file sizes listed in the submission receipt are not 0Kb!

Tentative Marking Guide

item	Marks
The makefile	....
targets	[/1]
uses -Wall	[/1]
compiles without warnings etc.	[/2]
The Program	....
Correct Output (i.e. finds full key)	[/6]
Ring Structure	[/5]
Command line error checking	[/2]
Checks for fork failure	[/2]
Doesn't use hardwired constants	[/2]
Consistent Use of Style	[/3]
Correct number of children	[/1]
Total	25 Marks