Faculty of Computing, Engineering and Mathematical Sciences

Internal Moderation of Coursework

General Instructions

The module leader should supply to the moderator a printed copy of the proposed assignment which should include indications of due date, weighting, assessment criteria and effort required. The moderator should use this form to comment on and progress the assignment, recording brief comments on the form and more extensive ones on the assignment specification. When the assignment has been agreed with the module leader, the moderator should sign the form and submit it along with the assignment to the CEMS Programmes Office.

Module Leader to complete this Section

module name **CSA**

module number **ufeEHF-30-1**

assignment number **1**

issue date **25/10/07**

% weighting in module **15%**

estimated time to complete **12 hrs**

module leader **Rob Williams**

internal moderator **Nigel Gunton**

work set by **Rob Williams**

Moderation

The moderator should check the assignment is satisfactory with respect to:

- rubric (including due date, weighting, and estimated effort)
- the task specification
- mark allocation and assessment criteria
- level of work
- effort needed

moderator's comments

setter's response

R Williams

Internal moderation completed

date

signed
(internal moderator)

MODULAR PROGAMME
ASSESSED COURSEWORK SPECIFICATION

Module Details:

Module Code: **ufeEHF-30-1**	Module Title: **Computer Systems**	
Module Leader: **Rob Williams**		
Module Tutors: **John Counsell** **Laurence O'Brien**		
Assignment **CW1**	Element Number: Weighting **15%**	Total Assignment Time: **12 hrs**

Dates:

Date assignment issued to students: **Nov 13th**	Date for return of marked work: **Feb 2nd 2005**
Submission Place: **postbox in N foyer, below the North stairs**	Date of Submission: **Thurs 20th Dec**
	Time of Submission: **10.00am**

Deliverables:

As listed on the Assignment spec sheet

ufeEHF-30-1 CSA, First Coursework Assignment, Nov 2007

The assignment must be delivered, with a completed Official Cover Sheet before the designated hand in date, week beginning 17th December. You are strongly advised NOT to hand in on the final day, but plan your submission for the previous Friday. Demonstrations for your tutor should be arranged before that date.

This assignment is a practical introduction to serial communications using the COM1 serial port on the PC. All code, as far as practicable, should be in asm86. C should only be used to start up the **`__asm`** directive in the VC++ Developer Studio. Calls to intrinsic C functions and Win32 are permissible for i/o operations.

It is better to work in pairs for this assignment, handing in a **single**, joint document. Both partners will gain the same mark.

Develop a secure point-to-point text transfer system. It should be able to take a large text file from disk, "encrypt" it and transfer it safely to the destination computer where it will be received, decrypted and stored on disk as a plain text file.

The encryption technique should use a secret binary "key" which has to be known to both source and destination. This key can be of any length. The key must be read from a USB stick or floppy disk, where it may be encrypted, too.

Deliverables:

1. After you have read this spec, but before you fully start with design and coding, email rob.williams@uwe.ac.uk, with SUBJECT field set to "ESTIMATE", an estimate of how long the program will take you to design and code. If you do not do this you will forfeit the mark for Section 5.

2. Supply a fully commented source listing of your programs. (35 marks)

   Provision of header comments
   Correct use of __asm
   Definition of data in C section
   Access to Win32/libc calls - parameter handling
   Use of user defined subroutines - CALL/RET
   Register parameters
   Stack parameters
   Local variable stack frames using ENTER/LEAVE
   Clear code structure
   en/decryption code structure
   Error handling

3. Demonstrate to your lab tutor a functioning system. You must supply a printed source listing for discussion which will then be dated and signed by the tutor This should then be handed in with the other documents.

   It starts to run from the debugger (25 marks)
   It starts from a desktop icon
   It continues without crashing
   It achieves the basic functionality
   It is fully understood by authors

4. Build a small web site, minimum 2 pages, including some diagrams and links to other useful sites. This should be in two parts:

4.1 An explanation of the various methods of parameter passing used by HLL compilers.

   What is a parameter, where would it be used? (25 marks)
   What is the advantage over alternative methods?
   What is a Stack Frame?
   How is the Base Frame Pointer used?
   Is there any similarity between Local Variables and function Parameters?

4.2 An Outline description of the operation of your encryption scheme. (10 marks)

5. Provide a retrospective total of the time you have spent on designing and coding the programs. Provide a reference list, including useful URLs (5 marks)

NB. Set up the serial link (COM1) using Hyperterminal for HARDWARE FLOWCONTROL.
Check the operation between the 2 PCs before starting.

Rob Williams, 22/10/07

Encryption, A Brief Introduction by Julian Walters

Encryption is the process of transforming data so that, while sender and receiver can still determine its meaning, anyone else will find it incomprehensible. As with most things in life, encryption techniques range from the simple to the intensely complex, with every shade in between. The history of encryption, and the parallel activity of clandestine decryption or code breaking, provides an exciting view of some of the world's most significant events. Without coders and decoders the world would be very different today. Have a look in Simon Singh's excellent text "The Code Book" for an exciting, readable perspective.

Simple Encryption

The simplest and earliest form of encryption is a process termed **Substitution Encryption**. In this, each character is swapped with a different one, consistently throughout the original text. This can be done by applying and algorithm to the numeric representation (ASCII) of the letters, or by using a look-up table. Consider the simple case where we add 1 to the value of the character (except for "Z", which becomes "A"). Examine the table below which illustrates this transformation:

Original letters	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher letter codes	BCDEFGHIJKLMNOPQRSTUVWXYZA

Examining the above table, the message "It is most dire send more money" becomes "ju jt nptu ejsf tfoe npsf npofz". This is not too shabby at first sight, but the lack of encryption of the space character means that the first two words must belong to the very limited subset of words in the English language that have only two characters. Worse, they both start with the same letter. This in itself is enough for someone else to break the cipher without much effort. While we could add the space character to the encryption set, this does not really strengthen the code against attack.

Changing the transformation from an ADD to a logical XOR (conditional bit inversion), or byte ROTATE, would seem to be better, but does not really stop the experienced code breaker, and also requires the use of non-printable ASCII control codes.

The serious problem is that a numerical analysis of English writing reveals the average frequency of use of each letter each word, and even each common phrase. Such information helps decoders to quickly crack substitution codes, as long as they capture messages of sufficient length. All languages have a character which occurs most commonly, second most commonly, and so on. Remember, the code breaker, trying to break the cipher needs, only a few letters before he can start guessing at the rest. I cannot improve on the description of this process given in the Sherlock Holmes story "The Dancing Men" by Arthur Conan Doyle, - read it, it's a good yarn. Encryption of this type, where one cipher character always represents another plain text character, is termed Substitution Encryption.

Before we leave this section it is of interest to note that during the fall of France in 1940, an isolated British unit became aware that the Germans were reading and decrypting their transmissions. As luck should have it, both the isolated unit and Group HQ contained a Welshman in their ranks. Morse communication in Welsh was soon established, and most of the isolated unit escaped. The Germans were alarmed at what seemed an unbreakable code until someone with the relevant linguistic education saw the transcriptions, and revealed the nature of the "code". The Americans used the same technique in their war with the Japanese by exploiting the weird north american Navajo language for battlefield communications. Frequency analysis code breaking techniques do not work if you are unable to reliably identify the language components!

More Complex codes

It is obvious from the above account that straightforward substitution codes are no longer acceptable due to the effectiveness of code breaking techniques. There are, however, some extra methods which can be used to render character frequency analysis less easy. One more secure method of encryption involves having a single password known only to the sender and intended receiver. The encryption method works by combining the ASCII value of each plain text letter with that of a password letter. So, successive password letters are applied to successive text characters of the string to be encrypted, recycling the password over and over again. Let's use the previous example, with the password "seven", but instead of adding the two letter codes together we will use the logical XOR operation. This inverts bits where they have the same value.

```
Plain text message   It is most dire send more money
Password key         sevensevensevensevensevensevens

XOR cypher text      Z1v,=s(96:s!?7+s63+*s(97+s(9++*
```

(because many of the codes resulting from XORing two letters together are unprintable a 32 offset has been added to them all, just to allow you to read something on the page!)

Simply adding two 7 bit ASCII characters will result in overflow into the 8th bit, which may not matter if the full byte is transmitted, however if you want only to send 7 bits there is a problem! Substituting an XOR operation for the addition avoids this problem. In the above example, the first character in the encrypted string "Z" would come from XORing the ASCII values of "I" (0100_1001) and "s" (0111_0011), resulting in 0011_1010, ":", but to make all of the encrypted codes printable a further 32 has been added to make: 0101_1010, "Z".

The code breaker now has to first guess the cipher key. If you look at the cipher text, notice that "s(9", occurs 3 times at positions 5, 20 and 25. In each case they suggest the same three letter sequence has occurred, but more importantly, the cipher key is likely to be 5 characters long! In fact, we know that the text trigram " mo" aligned up with the password letters "sev". But although the code breaker will not yet know this, he can deduce that the password is 5 characters long. Allowing the encrypted text to be divided into subgroups of 5 characters. At this point all the cipher text caracters are placed into five alphabet groups, group 1 = characters 0, 5, 10,...; group 2 = 1, 6, 11,...; group 3 = 2, 7, 12,... etc. Letter frequency analysis can now be applied to the contents of each group as was described in the substitution cipher example above. Because the contents of each group have been encrypted with the same character they will present, as a group, a simple substitution encryption, although each group will represent a separate case. Only a few letters are required for the code-breaker to start to fill in blanks. If the code-breaker has enough of the encrypted text, he will soon find the password and crack the code.

The above scheme of encryption was for some considerable time held to be unbreakable. Charles Babbage took up the task, challenging someone to provide a sample for him to break. To contemporary observers he appeared to fail. It was only many years latter that it was discovered that he had in fact succeeded in breaking the cipher, but had been dissuaded from revealing this fact by official sources who used this form of encryption for sensitive communications. Whenever your own coder breakers are active, it is of the utmost priority to hide their successes from the enemy!

Now we will look at an even stronger code: **Multiple Alphabet Substitution Encryption**. This works by using alternative lookup alphabets rather than the single table explained in the previous paragraph.

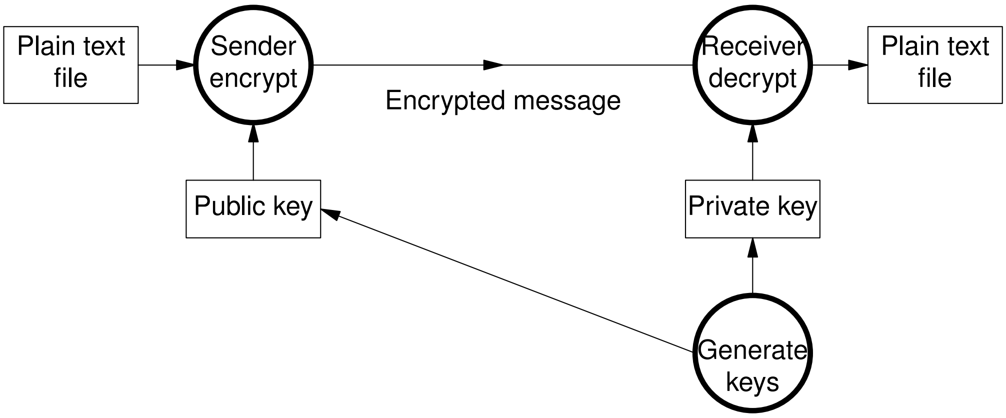
A single key or password is still employed but in a more subtle manner. The extra complexity of this method comes from the use of several substitution alphabets in place of one. The cipher key is employed to select which of the alternative alphabets to use for each of the plain text letters. The technique can be best understood by considering the Vignenere square which holds all the cipher alphabets:

```
abcdefghijklmnopqrstuvwxyz
bcdefghijklmnopqrstuvwxyza
cdefghijklmnopqrstuvwxyzab
defghijklmnopqrstuvwxyzabc
efghijklmnopqrstuvwxyzabcd
fghijklmnopqrstuvwxyzabcde
ghijklmnopqrstuvwxyzabcdef
hijklmnopqrstuvwxyzabcdefg
ijklmnopqrstuvwxyzabcdefgh
jklmnopqrstuvwxyzabcdefghi
klmnopqrstuvwxyzabcdefghij
lmnopqrstuvwxyzabcdefghijk
mnopqrstuvwxyzabcdefghijkl
nopqrstuvwxyzabcdefghijklm
opqrstuvwxyzabcdefghijklmn
pqrstuvwxyzabcdefghijklmno
qrstuvwxyzabcdefghijklmnop
rstuvwxyzabcdefghijklmnopq
stuvwxyzabcdefghijklmnopqr
tuvwxyzabcdefghijklmnopqrs
uvwxyzabcdefghijklmnopqrst
vwxyzabcdefghijklmnopqrstu
wxyzabcdefghijklmnopqrstuv
xyzabcdefghijklmnopqrstuvw
yzabcdefghijklmnopqrstuvwx
zabcdefghijklmnopqrstuvwxy
```

Take a letter, say "t", from the plain text message and use the matching letter, "c", from the key word to select a row. Thus the third row down is the chosen cipher alphabet for use as a look-up table to encode the letter "t". Pass along the top row until you reach "t" then descend to the third row and pick out "v" as the coded letter. In this way you cycle through the alternative cipher alphabets, confusing the code crackers. In terms of code security, the longer the key word the better.

```
Plain text:    It is most dire send more money
Password key   acidrainacidrainacidrainacidrai
```

Obviously there are far more complex and difficult coding stratagems in existence, ones which cannot even now be cracked in a reasonable time using the fastest computers available. An interesting development is the **RSA Public Key Encryption** scheme. This employs not a single key but pairs of Public and Private keys. The sender uses a publically visible key to encrypt messages, while the receiver needs the matched, private key to decrypt the cipher text. Once the system is set up by distributing the private keys, data can be encoded and transmitted with some confidence that no unauthorized decrypting and reading will take place. This brilliantly novel system relies on the time it takes to find the factors for large (10^{129}) numbers.



1. To generate the Public/Private key pairs, you have to select two large prime numbers, which we will call p and q. Ensure that p != q.

2. Compute n = p*q (187 = 17*11)

3. Select a small, odd integer e that is not a factor of (p-1)*(q-1), is > 1, and is < p*q. say 7.

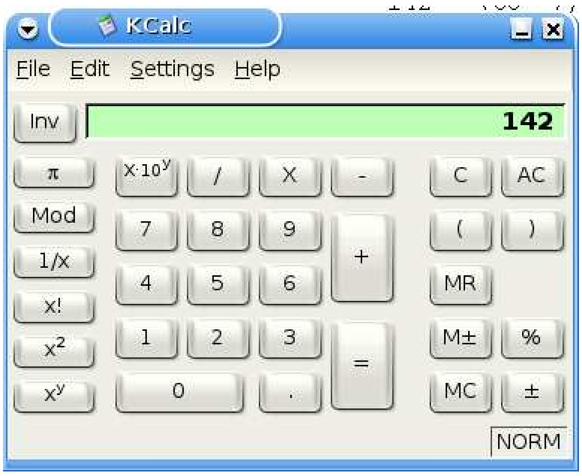
Your Public Key for encryption is now (187, 7)

4. If the data to be sent is the ASCII letter 'A', (data = 65)

5. The encrypted data is $(data)^e$(mod p*q)

cryptdata = (data**e)%(p*q);

Use `bc`, the command line calculator, or `kcalc` for the calculations: 142 = (65**7)%187(cryptdata = 142)



6. To decrypt, select a number e, where 1 = d.e mod(p-1).(q-1)

7*e = 1 (mod 160)

Try out 7*1, 7*2, 7*3..... calculated to mod 160. When you get to 7*23 mod 160, the result will (at last) be 1. (e = 23)

Your Private Key for decryption is now (187, 23)

7. Using whatever method you prefer, teeth, hammer, fire, make sure that p and q are completely erased to prevent their reuse and hide e.

8. To recover the data from the encrypted code:

data = (cryptdata**d)%(p*q)

data = (142**23)%187

Unsurprisingly, this will generate numbers too big for `kcalc` so it has to be broken down into factors:

data = (142**5)%187*(142**5)%187*(142**5)%187*(142**5)%187*(142**3)%187

data = (109*109)%187*(109*109)%187*131%187
= 65 original value of the data recovered!!

The secure remote login tool, ssh, offers an authentication method based on the twin key, RSA method. Each user creates a public/private key pair for authentication purposes. The remote server knows the public key, and only the user knows the private key for local use.

On Linux, the file $HOME/.ssh/authorized_keys holds all the ***public*** keys that are permitted for logging in using ssh from a remote host. When the user logs using ssh, it tells sshd on the remote server which key pair it would like to use for authentication. The server checks if this key is permitted, and if so, sends back to the user (actually the ssh program running on behalf of the user) a challenge, a random number, encrypted by the user's public key. The challenge can only be decrypted using the proper private key. The user's ssh client then decrypts the challenge using the private key, proving that he/she knows the private key but without disclosing it to the server. In this way neither password or key gets transmitted along the vulnerable channel.

To use ssh the user must generate the key pairs beforehand, and store them locally and on the remote server. The RSA key pairs are synthesized using `ssh-keygen`. This stores the local ***private*** key in $HOME/.ssh/identity and stores the companion ***public*** key in $HOME/.ssh/identity.pub in the user's local home directory. The user should then copy the identity.pub to $HOME/.ssh/authorized_keys on the remote machine. This requires a conventional login to the remote server. The authorized_keys file corresponds to the conventional $HOME/.rhosts file, with one key per line. After this, the user can log in without giving the password.

At the moment, even powerful supercomputers find this a time consuming task.

Space and the author's limited knowledge stops the explanation at this point.

If I were developing a simple code from scratch, I would consider the following alternatives:

1. use the single key transformation algorithm with a cipher key of about eight characters length.

2. use the asm ROR instruction, with a variable amount of rotation for each letter in sequence.

3. use a randomly generated look-up table

One nice trick to throw frequency analysis code breakers off the scent is to use bad spelling. Happy Coding!

Read These

The Code Book, Simon Singh, Fourth Estate Books.

Full marking Schedule, tutors' view

1. Develop a secure point-to-point text transfer system. It should be able to take a text file from disk, "encode" it and transfer it safely to the destination computer where it will be received, decoded and stored on disk as a plain text file.

A fully commented source listing of your program.	(35 marks max)
Program banner with title, author name, date, running instructions	5 marks
Function banners	5 marks
Correct use of __asm directive	2 marks
Definition of data in C section	2 marks
fopen, fget, fput, fflush library CALLs	
with stack parameters in correct order	
stack scrubbing using add esp,8	5 marks
EAX return value handling	5 marks
Error trapping and reporting	5 marks
Clear asm program structure	5 marks
minimum use of jmp, jz	
canny use of CPU registers	5 marks
no unnecessary data variables	
Use of user defined subroutines in asm - CALL/RET	5 marks
Stack parameters	
EAX return handling	
Local variable stack frames - ENTER/LEAVE	5 marks extra
Good en/decryption code	5 marks
Good appropriate comments:	5 marks
not line-by-line op code description	
NO mention of CPU registers within a comment	
explains WHY not how	
describes the algorithm not the code	
groups several instructions together for a single comment	

2. Demonstration and oral explanation of the functioning system. A full, printed source listing must be provided which will be dated and signed by the lab tutor, then handed in with the other documents.

	(25 marks max)
It runs once without crashing	5 marks
It achieves the basic functionality	10 marks
It is mostly understood by authors	5 marks
It is fully understood by authors	5 marks

3. A useful 2 page min. web site, including diagrams and reference links, concerning the methods of parameter passing used by HLL compilers.

diag of stack with frames, copied from a book	5 marks
diag of stack with frames, indicating understanding	10 marks
What is a parameter, where would it be used?	(25 marks max)
data required by a function/subroutine to "localise" its activity	5 marks
What is the advantage over alternative methods?	10 marks
passing in a register limits the size/number of parameters, also non-reentrant	
data block indicated by a pointer parameter,	
global data not recommended because of its visibility.	
What is a Stack Frame?	5 marks
data storage area allocated within the stack for every function/subroutine	

Is there any similarity between Local Variables and function Parameters?
both held in stack frame, accessed through base frame pointer EBP
or by POPping into a register
"parameters are pre-initialized local variables" 3 marks

Outline description of the operation of your encryption scheme. (10 marks max)

Reserve 10 marks for the web site operation

6. References, including the useful URLs (5 marks max)