

# CST8134 — Lab assignment 3: Crypto

You find that you have a need for a pair of cryptographic routines, to be named encrypt and decrypt. Since you are highly skilled in M68K assembler and since that addresses the problem space nicely, you decide to use that.

In your investigation, you discard the simple substitution cypher many of us played with in public school (replace a with q, b with s, c with b, and so on). It's just too easy, as is the rotate-13 (ROT-13 aka EGB-13) commonly used on the Internet to obscure contentious material. Public key encryption, you decide, is much too complex. An approach using EOR (XOR; exclusive OR) looks OK, and you imagine that combining it with a bit rotation will be an advantage.

You need two programs, since this is not a symmetric algorithm, one to encrypt the cleartext into cyphertext and one to decrypt cyphertext back into cleartext. You also realize that the cyphertext will not be in printable ASCII characters, so you will have to convert it to an ASCII representation of the hex in the encrypt program, and read in that form and convert it to hex in the decrypt program. That is, "xyz" as cleartext input might become 0x039BF1 as cyphertext, so it is displayed as "039BF1" which is actually 0x303339424631. Reading in "039BF1", you first have to convert it to 0x039BF1.

Each of the two programs will first have to read in the key string, a word or phrase at least 17 and not more than (for convenience) 53 characters long.

In encrypt, you then read in the cleartext message, up to perhaps 80 characters. First, use your StrCat to make a copy of the key string the same length as the cleartext. Now for each character (byte) of the cleartext, rotate it 4 bits (this reverses the nibbles: 0x39 becomes 0x93, for example); then use EOR (exclusive OR) with the corresponding byte of the length-extended key string. Convert the resulting byte to 2 bytes of displayable characters as above.

Decrypt is the reverse. After reading in the key string, read in the ASCII cyphertext and convert each pair of characters into one hex byte. Length-extend the key-string as in encrypt. Now first EOR (XOR) each byte against the corresponding key string byte, and rotate the nibbles back to their normal position. You should be able to display this, an exact representation of the original input to the encrypt program.

You will notice several opportunities above to use your StrXxx functions, and to create some new functions specifically for encrypt and decrypt. You will of course do so wherever you can.

For example (your input is **red**; I would use copy/paste for the cyphertext input):

```
[run encrypt]
Enter key string: the bear came over the mountain
Enter cleartext: I have a secret message that I have to hide here
Cyphertext: 3165 ... 9F37
```

```
[run decrypt]
Enter key string: the bear came over the mountain
Enter cyphertext: 3165 ... 9F37
Cleartext: I have a secret message that I have to hide here
```

Use the stack for your functions to pass arguments and return values; use of a

frame pointer in each function where it's necessary is highly desirable.

You will also need to design and write your main functions to prompt and receive the input, process each line, and produce the output. You know that you can use the provided ReadLine function as well as all the functions in bdblib.

I suggest that you determine what functions you can write (first do over-all PDL), then write each function separately, PDL first, with a simple test driver for each. Once they are all working correctly you can combine them with the final encrypt and decrypt programs. Start now – don't wait until the last week or you will not finish. You have already given me your StrXxx functions, so do not submit source or PDL for them unless you have made significant changes to them.

You will hand in your assignment at your lab start in 3 weeks time (week 14; April 17 or 18, 2008). No disk, email, or Blackboard submission is required: use paper. This must include:

- The standard cover page and a table of contents;
- A Design Document (write it in MS Word or OpenOffice) containing:
  - The problem statement in your own words, and some observations on requirements that may be implied or unstated;
  - The PDL for the main program and for each new function that you write, (do not also include PDL in your source code);
  - A data dictionary for each function that lists the registers you expect to use as well as each variable in the DATA and TEXT 1 sections (the name, type, size in bytes, and a brief purpose for each);
  - For each function, a description of the arguments and return value, plus a memory map of the stack frame if it's used;
  - A test plan to confirm that all aspects of your program work correctly, and that incorrect input will be dealt with reasonably (use a table to show the test number, the input to use, the expected result, and the purpose of the test);
- A cleanly printed copy of your .lis files, including reasonable comments, both in-line and block (each function, including your main programs, must have a block comment header at its start, much like a C++ function);
- Your printed test results, consisting of:
  - an updated test plan showing the actual results of the tests (briefly);
  - a screen-shot (black text on white) showing the required data results:
    - Key string: Assembler is my favourite language
    - Cleartext: The External Shareholder Focus Group (XFG) sessions have been convened to gather feedback.

You may work singly or in teams of two. Your lab instructor requires in advance an email from each member of a pair (separately), stating who your partner is. Partners do not have to be in the same lab section, but the assignment must be handed in for the earlier of the two sections.

Be sure to remember that you will not be able to write your tests and examinations in teams, and act accordingly – make sure both partners in a pair understand all the details.