

# Final Project

---

**Due** Mar 15 by 11:59pm    **Points** 125    **Submitting** a file upload    **File Types** pdf and asm

---

**Due: Tuesday, March 15th, 11:59 PM Pacific USA Time Zone**

Note: This is an individual project and you must complete the work independently. Since this is the final project (due on the final's week), **you are not allowed to use grace days on the submission**. Late submissions will lose 50 points per day. Please submit your work on time!

**This project is worth 125 points (100 points for the code and 25 points for the documentation).**

## Objectives

1. Working on a project that utilizes a broad spectrum of knowledge from this course
2. Using various forms of indirect addressing
3. Passing parameters on the stack
4. Extensive interaction with arrays

## scription

Alex and Sila are espionage agents working for the Top Secret Agency (TSA). Sometimes they discuss sensitive intelligence information that needs to be kept secret from the Really Bad Guys (RBG). Alex and Sila have decided to use a simple obfuscation algorithm that should be good enough to confuse the RBG. As the TSA's resident programmer you've been assigned to write a MASM procedure that will implement the requested behavior. Your code must be capable of encrypting and decrypting messages (and meet the other requirements given in this document). This final project is written less like an academic assignment and more like a real-life programming task. This might seem daunting at first but don't despair!

## Algorithm Details

Alex and Sila's algorithm is a simple substitution cipher. If you are not familiar with substitution ciphers, I suggest reading the Wikipedia page ([available here](http://en.wikipedia.org/wiki/Substitution_cipher) [\\_ \(http://en.wikipedia.org/wiki/Substitution\\_cipher\)](http://en.wikipedia.org/wiki/Substitution_cipher)). For this implementation, the key consists of 26 lowercase English letters. The following example demonstrates the obfuscation algorithm that Alex and Sila are requesting.

Suppose that you are provided the following 26 character secret key: **efbcdghijklmnopqrstuvwxyz**a. Each character listed in the key corresponds to a character from the English alphabet. In other words, the key is a map that tells you how to take the original text (also known as plaintext) and replace each letter with the encrypted character.

In order to understand the key, consider the alphabetical arrangement of the English alphabet beginning with a, b, c, d, e, f, (and so on). Now reconsider the key, which starts with e, f, b, c, d, g, (and so on).

Using the key from above, we can map each letter as follows:

```
Original Character --> Encrypted Replacement
a --> e
b --> f
c --> b
d --> c
e --> d
f --> g
g --> h
h --> i
i --> j
(some entries skipped)
x --> y
> z
▶ > a
```

For example, every “a” in the original message is to be enciphered as an “e”. Every “b” in the original message should be replaced with the letter “f”. Each character in the original message maps to a specific replacement character.

Now suppose that you are trying to encrypt the following plaintext message:  
**the contents of this message will be a mystery.**  
Using the secret key from above, the corresponding ciphertext is:  
**uid bpoudout pg uijt ndttehd xjmm fd e nztudsz.**

Note that only the letters have been changed! Other aspects of the message such as punctuation and spacing should be left unchanged. In order to decrypt the message, you can use the same key (in reverse) to process the ciphertext and convert it to plaintext.

The key will always consist of exactly 26 characters (containing every letter of the English alphabet arranged in a random order). A character can NEVER appear twice within the same key.

## General Overview

You are going to be writing a MASM procedure named **compute** which will be called from a MAIN procedure (that you do not control). Your MASM procedure has several modes of operation, depending on how it is called.

Please read this overview and then look at the Example Usage section below.

The **compute** procedure will be capable of several operations, namely:

- A default "decoy" mode of operation where the procedure accepts two 16-bit operands by value and one operand by OFFSET. The procedure will calculate the sum of the two operands and will store the result into memory at the OFFSET specified.
  - Accepts 3 parameters on the stack
    - 16 bit signed WORD operand
    - 16 bit signed WORD operand
    - 32 bit OFFSET of a signed DWORD (the sum will be placed into the specified DWORD)
- An encryption mode
  - Accepts 3 parameters on the stack
    - 32 bit OFFSET of a BYTE array (this array contains a 26 character key)
    - 32 bit OFFSET of a BYTE array (this array contains the plaintext message to be encrypted)
    - 32 bit OFFSET of a signed DWORD (the dereferenced value initially contains the integer -1)
  - Note that the plaintext message will be in a BYTE array that ends with a NULL character (indicating the end of the message)
  - This operational mode will encrypt the requested message. By the time your function returns, the original plaintext message array will be overwritten with the correctly encrypted message.
- A decryption mode
  - Accepts 3 parameters on the stack
    - 32 bit OFFSET of a BYTE array (this array contains a 26 character key)
    - 32 bit OFFSET of a BYTE array (this array contains the encrypted message that is to be decoded)
    - 32 bit OFFSET of a signed DWORD (the dereferenced value initially contains the integer -2)
  - Note that the encrypted message will be in a BYTE array that ends with a NULL character (indicating the end of the message)

- This operational mode will decrypt the requested message. By the time your function returns, the encrypted characters (inside the array) will be overwritten with the decrypted message.
- A key generation mode (OPTIONAL) - **THIS SPECIFIC MODE IS EXTRA CREDIT**
  - Accepts 2 parameters on the stack
    - 32 bit OFFSET of a BYTE array (this array contains space for exactly 26 characters)
    - 32 bit OFFSET of signed DWORD (the dereferenced value initially contains the integer -3)
  - In this mode, your code will utilize the Irvine random number library. You must generate a 26 character string containing **all** the lowercase letters of the English alphabet.
    - The letters must be arranged in a random order.
    - If this function is called multiple times, the order of the characters must be different each time.
  - Your randomly generated character string will be placed into BYTE array specified by the stack parameter.

## Example Usage of Your Procedure

From the description above, it may not be clear how the various modes of operation are selected. This section provides some examples that should provide clarification.

Recall that you will not be providing a .data segment in your submitted code. However, it is fine to write a .data segment while you are testing your code.

 decoy mode (as called from the **main** procedure):

```
.data
operand1  WORD    46
operand2  WORD   -20
dest       DWORD    0
.code
;; inside the MAIN procedure
push  operand1
push  operand2
push  OFFSET dest
call  compute
;; currently dest holds a value of +26
mov   eax, dest
call  WriteInt ; should display +26
```

An example of encryption mode (as called from the **main** procedure):

```
.data
myKey      BYTE    "efbcdghijklmnopqrstuvwxyz"
message    BYTE    "the contents of this message will be a mystery.",0
dest       DWORD    -1
.code
;; inside the MAIN procedure
push  OFFSET myKey
push  OFFSET message
push  OFFSET dest
call  compute
;; message now contains the encrypted string
mov   edx, OFFSET message
call  WriteString
;; should display "uid bpoudout pg uijt ndttehd xjmm fd e nztudsz."
```

An example of decryption mode (as called from the **main** procedure):

```
.data
myKey      BYTE    "efbcdghijklmnopqrstuvwxyz"
message    BYTE    "uid bpoudout pg uijt ndttehd xjmm fd e nztudsz.",0
dest       DWORD    -2
.code
;; inside the MAIN procedure
push  OFFSET myKey
push  OFFSET message
push  OFFSET dest
call  compute
;; message now contains the decrypted string
mov   edx, OFFSET message
call  WriteString
;; should display "the contents of this message will be a mystery."
```

An example of key generation mode (as called from the **main** procedure). As noted above, you are not required to implement this mode unless you want the extra credit.

```
.data
newKey     BYTE    26    DUP(?)
dest       DWORD    -3
.code
```

```
;; inside the MAIN procedure
push    OFFSET newKey
push    OFFSET dest
call    compute
;; newKey now contains a randomly generated key.
;; The key will contain all 26 letters of the alphabet
;; (arranged in a random order).

;; Note that newKey is not NULL terminated.
```

## Example Output

```
Decoy:
Should display '+26':
+26

Encryption:
Should display 'uid bpoudout pg uijt ndttehd xjmm fd e nztudsz.':
uid bpoudout pg uijt ndttehd xjmm fd e nztudsz.

Decryption:
Should display 'the contents of this message will be a mystery.':
the contents of this message will be a mystery.

Key generation: (Extra credit)
Should display a 26 character long string containing a-z with no repeats:
bqjlygehvdtnupisrfkomz
```




## Additional Requirements

1. Your code must operate in the manner shown in the examples (accepting the parameters on the stack).
2. I will always interact with your code by calling the **compute** procedure.
3. The **main** procedure from your code will not be used! Instead, I will verify the functionality of your procedure by writing my own **main** procedure and calling your **compute** procedure (in multiple ways) to test it.
4. **Your code cannot include a .data segment!** Instead, any memory that you want to utilize must be allocated from the stack.
5. Your procedures are not allowed to reference .data segment variables by name (this is implied by requirement #4).

6. Procedures are allowed to use local variables if you choose (section 8.2.9 in the textbook). The alternative is for you to manually adjust ESP (allocating an area of memory on the stack that you can use within your procedure).
7. All procedure parameters must be passed on the system stack.
8. Each procedure will implement a section of the program logic. Your code must be modularized into at least 4 procedures (not including **main**). In other words, your **compute** procedure must divide the work into various sub-procedures (that are called from inside **compute**). The specifics of how you divide the work and achieve this task are up to you.
9. Parameters must be passed by value or by reference on the system stack as listed above.
10. You are never allowed to corrupt memory by accessing arrays at invalid indices. In particular, the encrypted messages and decrypted messages are stored in arrays that terminate with a NULL character. The NULL character must be left intact.
11. The program must use appropriate addressing modes for array elements (no hard-coded array names).
12. The random number generator must be properly seeded using Randomize
13. Each procedure must have a procedure header that follows the format discussed during lecture.
14. The code and the output must be well-formatted.
15. The usual requirements regarding documentation, readability, user-friendliness, etc., apply.
16. Your code must not enter any infinite loops, cause a segmentation fault, or result in any other undefined behavior.

## Notes

1. **DO NOT** put this off - it is the final project for this course and serves as the final exam.
2. Assume that the messages (to be encrypted or decrypted) could be up to 50,000 bytes long (including the NULL character).
-  The key is exactly 26 characters long and is never NULL terminated.
4. The parameters will ALWAYS be pushed to the stack in the exact order shown in the examples above.
5. The Irvine library provides procedures for generating random numbers. Call Randomize once at the beginning of the program (to set up so you don't get the same sequence every time), and call RandomRange to get a pseudo-random number. (See the lecture slides)
6. For this program you can assume that the plaintext message will not contain any capital letters. That makes the program easier to implement.
7. If you choose to use the LOCAL directive while working on this program be sure to read section 8.2.9 in the Irvine textbook. LOCAL variables will affect the contents of the system stack!

## Extra Credit Options

- (1 pt) Add a comment that contains EXACTLY ONE TA name at the beginning of the program. If that TA happens to be your grader, then you get the points :)
- (2 pts) Ensure that the "decoy" mode correctly returns the sum of ANY signed 16 bit numbers  
For example, if I sum these two 16-bit numbers:  $(-32768) + (-32768)$  I should receive the value  $(-65,536)$  (stored in the 32 bit variable).
- (8 pts) Implement the key generation mode as described in the documentation.

## Project Write-Up

**Part 1:** Summarize your work in a well-written report. You should proof-read your report and ensure that the writing meets professional standards. Use images, charts, diagrams or other visual techniques to help convey your information to the reader.

Be sure to address the following points and answer these questions:

- How did you break up your code into multiple procedures (i.e. what was your strategy to separate the code into pieces)?
- Did you run out of general purpose registers when implementing your design? If so, what steps did you take to work around the problem?
- Is there anything you would implement differently if you were to re-implement this project?
- What were the primary challenges that you encountered while working on the project?
- Besides the textbook, were there any particular resources that you would recommend to future students in this course?

**Part 2:** If you chose to implement any extra credit tasks, be sure to include a thorough description of this work in the report.



## What to Submit

- Your source code (contained in an .asm file)
- a PDF file containing your typed write-up

## Errata

This section of the assignment will be updated as changes and clarifications are made. Each entry here should have a date and brief description of the change so that you can look over the errata and easily see if any updates have been made since your last review.

**Feb 24th** - Posted documentation for final project.

---






Criteria	Ratings		Pts
Student submitted a .asm file with the source code and a .pdf file containing the typed project report.	<b>10 pts Full Marks</b>	<b>0 pts No Marks</b>	10 pts
All procedures contain a header that follows the format discussed during lecture.	<b>6 pts Full Marks</b>	<b>0 pts No Marks</b>	6 pts
Code is written according to the standards that have been explained throughout the term (proper indentation, suitable comments, easy to understand, etc).	<b>8 pts Full Marks</b>	<b>0 pts No Marks</b>	8 pts
Your code must be modularized into at least 4 procedures (not including main).	<b>8 pts Full Marks</b>	<b>0 pts No Marks</b>	8 pts
<div data-bbox="79 979 149 1039" data-label="Image"></div> e enters correct mode based on value that is contained within the dereferenced DWORD variable (passed by OFFSET).	<b>4 pts Full Marks</b>	<b>0 pts No Marks</b>	4 pts
Student code is written in such a way that all features are available solely by pushing parameters onto the stack and calling the "compute" procedure.	<b>5 pts Full Marks</b>	<b>0 pts No Marks</b>	5 pts

Criteria	Ratings		Pts
Punctuation, spacing, and non-lowercase letters within plaintext are left intact by the encryption & decryption process.	<b>3 pts Full Marks</b>	<b>0 pts No Marks</b>	3 pts
Decoy mode functionality: Utilizes 3 parameters from the stack (as shown in the documentation): two 16 bit *signed* WORD operands and a 32 bit OFFSET of a DWORD variable.	<b>4 pts Full Marks</b>	<b>0 pts No Marks</b>	4 pts
Decoy mode functionality: Code enters decoy mode by derefencing the OFFSET of a DWORD value (the last operand pushed onto the stack) and verifying that the value is not -1 or -2. If you specifically check for the value 0, that's okay as well.	<b>3 pts Full Marks</b>	<b>0 pts No Marks</b>	3 pts
Decoy mode functionality: Code correctly sums the two signed 16 bit operands and stores the sum into the DWORD at the provided OFFSET.	<b>5 pts Full Marks</b>	<b>0 pts No Marks</b>	5 pts
<div data-bbox="79 979 149 1039" data-label="Image"></div> yption mode functionality: Utilizes 3 parameters from the stack (as shown in the documentation): OFFSET of a BYTE array (containing the 26 character key), OFFSET of BYTE array containing a NULL-terminated message, and OFFSET of DWORD memory variable that contains the value -1.	<b>3 pts Full Marks</b>	<b>0 pts No Marks</b>	3 pts
Encryption mode functionality: Code utilizes the key (provided via OFFSET on the stack) and correctly encrypts the message (also provided via OFFSET on the stack).	<b>14 pts Full Marks</b>	<b>0 pts No Marks</b>	14 pts

Criteria	Ratings		Pts
Encryption mode functionality: After execution of the procedure, each character of the plaintext array (inside the array) has been replaced with its corresponding encrypted character.	<b>5 pts Full Marks</b>	<b>0 pts No Marks</b>	5 pts
Decryption mode functionality: Utilizes 3 parameters from the stack (as shown in the documentation): OFFSET of a BYTE array (containing the 26 character key), OFFSET of BYTE array containing a NULL-terminated encrypted message, and OFFSET of DWORD memory variable that contains the value -2.	<b>3 pts Full Marks</b>	<b>0 pts No Marks</b>	3 pts
Decryption mode functionality: Code utilizes the key (provided via OFFSET on the stack) and correctly decrypts the encrypted text (also provided via OFFSET on the stack).	<b>14 pts Full Marks</b>	<b>0 pts No Marks</b>	14 pts
Decryption mode functionality: After execution of the procedure, each character of the encrypted text (inside the array) has been replaced with its corresponding plaintext character.	<b>5 pts Full Marks</b>	<b>0 pts No Marks</b>	5 pts
Project report is written professionally (with correct grammar and spelling).	<b>7.5 pts Full Marks</b>	<b>0 pts No Marks</b>	7.5 pts
Report includes thoughtful answers to all 5 questions that are listed in the "project write-up" section of the documentation.	<b>17.5 pts Full Marks</b>	<b>0 pts No Marks</b>	17.5 pts

Criteria	Ratings		Pts
Extra credit: Successfully guess the grader (1 pt)	0 pts Full Marks	0 pts No Marks	0 pts
Extra credit (MUST be documented in the report): (2 pts) Decoy mode handles any combination of signed 16 bit operands and returns the correct 32 bit signed sum (stored into the appropriate memory address).	0 pts Full Marks	0 pts No Marks	0 pts
Extra credit for key generation mode (MUST be documented in the report): (3 pts) Utilizes 2 parameters from the stack (as shown in the documentation): OFFSET of a BYTE array (containing space for exactly 26 characters), and the OFFSET of a DWORD memory variable that contains the value -3. Places a randomly generated key into the provided array.	0 pts Full Marks	0 pts No Marks	0 pts
Extra credit for key generation mode (MUST be documented in the report): (1 pt) Characters in key are random each time the procedure is called (utilizing the Irvine random number procedure). (2 pts) The random number generator must be properly seeded using Randomize. 	0 pts Full Marks	0 pts No Marks	0 pts
Extra credit for key generation mode (MUST be documented in the report): (2 pts) Generated key contains all 26 characters of the English alphabet (stored into the provided array).	0 pts Full Marks	0 pts No Marks	0 pts
Deduction: (-10 pts) Global variables are not allowed within your procedures. They must operate correctly without any ".data" segment.	0 pts Full Marks	0 pts No Marks	0 pts

Criteria	Ratings		Pts
Deduction: (-5 pts) Within your procedures, only memory from the stack is utilized (registers are permitted of course).	0 pts Full Marks	0 pts No Marks	0 pts
Deduction: (-3 pts) Code must not corrupt the NULL character that is present at the end of each plaintext or encrypted message.	0 pts Full Marks	0 pts No Marks	0 pts
Deduction: (-6 pts) Code must not impose any arbitrary length on the encrypted or decrypted message. The program must function with messages that are up to 50,000 bytes in length (including the NULL character).	0 pts Full Marks	0 pts No Marks	0 pts
Deduction: (-3 pts) Code must treat the key as a 26 character array and should not require a NULL terminator to function correctly.	0 pts Full Marks	0 pts No Marks	0 pts
<div data-bbox="79 976 149 1040">▶</div> Deduction: (-10 pts) Code does not crash, halt abruptly, or enter any infinite loops.	0 pts Full Marks	0 pts No Marks	0 pts
Deduction: (-10 pts) The code does not corrupt any memory or produce unexpected output.	0 pts Full Marks	0 pts No Marks	0 pts

Criteria	Ratings		Pts
Deduction: (-2 pts) Your code should accept parameters from the stack in the exact order that is demonstrated in the project documentation.	0 pts Full Marks	0 pts No Marks	0 pts
Late Penalty Remove points here for late assignments. (-50 pts per day)	0 pts Full Marks	0 pts No Marks	0 pts
Total Points: 125			

