# **PEX 2: Modern cryptographic standards**

### *Due: Lesson 38*

#### (75 Points)

|  |
| --- |
| **Help Policy**  **AUTHORIZED RESOURCES:** Any, except another cadet’s program.  **NOTE:**   * Never copy another person’s work and submit it as your own. * Do not jointly create a program. This is neither a team project nor a group project. Although you may discuss the assignment with any cadet enrolled in CS431, you may not jointly implement a program, you may not be coached or walked through a program, and you may not use another cadet's program as a source of help. You must do the assignment yourself and submit your own work for a grade. * You must document all help received from sources other than your instructor or instructor-provided course materials (including your textbook). * **DFCS will recommend a course grade of F for any cadet who egregiously violates this Help Policy or contributes to a violation by others.** |
|  |

|  |
| --- |
| **Documentation Policy** |
| * You must submit a documentation statement documenting all help received on this assignment. |
| * The documentation statement may describe the assistance received or simply refer to comments contained in the program code. |
| * The documentation statement must specify **WHAT** assistance was provided, **WHERE** in the code assistance was provided, and **WHO** provided the assistance. |
| * If **no help** was received on this assignment, the documentation statement must state “NONE.” |

1. Requirements

Learn and implement any reasonably complex standard cryptographic hash function or block cipher, in any programming language that you choose. Your code should be well-structured, well-documented, and should have a user interface that explains what it's doing and what the various inputs and outputs mean. The only requirement is that you use some other pre-existing implementation to verify your code is correct. (For example, many programming languages or environments have built-in cryptographic libraries for SHA-1, AES, etc. There are also web pages that generate test hashes and ciphers for you). You must hand in a screen shot that shows the output of your code and the output of a pre-existing implementation (regardless of whether or not they match). Your documentation statement must name the pre-existing implementation you used for verification.

One important restriction is that *you must develop your source code from scratch*. For reference material, you may use *algorithmic and pseudocode descriptions only.* Source code implementations are not allowed to be used as a source of help. If you have any questions or concerns about this, come see me before proceeding.

1. Recommendations

For possible hash functions to implement, the table in the Wikipedia article on cryptographic [hash functions](http://en.wikipedia.org/wiki/Cryptographic_hash_function) is as good a place as any to start. The same is true concerning the table in the article on [block ciphers](http://en.wikipedia.org/wiki/Block_ciphers). Some of the more obvious choices include DES, AES, SHA-n and MDn, but perhaps you try something you don’t know much about like Skipjack, or Twofish. If you really want to make my eyes water, implement a version of FFX (which requires AES) as that was the basis for my thesis! If you are inclined to choose a more obscure algorithm I recommend locating a pre-existing implementation before coding to guarantee you will be able to provide the necessary verification upon completion. Simple algorithms like Pearson Hashing are not acceptable because such algorithms shouldn’t be used for crypto and are trivial to code. If you have any doubt about the validity of your selection, please discuss your plans with me for final approval.

DES HELP:

If you choose to implement DES, I recommend first starting with your text, as it provides a very good pseudocode breakdown of the algorithm. This site provides another in depth explanation: <http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm>.

You may also look at <http://people.eku.edu/styere/Encrypt/JS-DES.html> which illustrates a walkthrough of the algorithm including intermediate outputs (in binary) for debugging. This may be a useful verification tool for once you get your code working.

Additionally, <http://seit.unsw.adfa.edu.au/staff/sites/lpb/src/DEScalc/index.html> illustrates an encryption iteration and provides a functioning .jar that you can download to verify your code’s functionality.

MD5 HELP:

If you choose to implement MD5, below is a snapshot of intermediate values. These are best viewed in conjunction with the [Wikipedia article on MD5](http://en.wikipedia.org/wiki/MD5).

Input: “The quick brown fox jumps over the lazy dog”

Fully padded message in hex:

54686520 71756963 6B206272 6F776E20 666F6820 6A756D70 73206F76 65722074

6865206C 617A7920 646F6780 00000000 00000000 00000000 58010000 00000000

32 bit little-endian words w[0]..w[15]:

w[0] 20656854

w[1] 63697571

w[2] 7262206B

w[3] 206E776F

w[4] 20786F66

w[5] 706D756A

w[6] 766F2073

w[7] 74207265

w[8] 6C206568

w[9] 20797A61

w[10] 80676F64

w[11] 00000000

w[12] 00000000

w[13] 00000000

w[14] 00000158

w[15] 00000000

Values of a,b,c,d after first iteration of FOR loop (i=0):

a 10325476

b D7D41184

c EFCDAB89

d 98BADCFE

Values of a,b,c,d after FOR loop terminates:

a 3637ED9D

b 92E87FAE

c 9C62FB6D

d C5E74FCC

Final values of h0, h1, h2, h3:

h0 9E107D9D

h1 372BB682

h2 6BD81D35

h3 42A419D6

Final 128-bit hash:

9E107D9D372BB6826BD81D3542A419D6

MD5 IN JAVA HELP:

If you’re implementing MD5 in Java, the lack of an unsigned integer type will make creating the table of k[i]’s as described in the Wikipedia article rather difficult due to conversion problems.  So here is the 64-entry table of constants k:

0xd76aa478 /\* 1 \*/

0xe8c7b756 /\* 2 \*/

0x242070db /\* 3 \*/

0xc1bdceee /\* 4 \*/

0xf57c0faf /\* 5 \*/

0x4787c62a /\* 6 \*/

0xa8304613 /\* 7 \*/

0xfd469501 /\* 8 \*/

0x698098d8 /\* 9 \*/

0x8b44f7af /\* 10 \*/

0xffff5bb1 /\* 11 \*/

0x895cd7be /\* 12 \*/

0x6b901122 /\* 13 \*/

0xfd987193 /\* 14 \*/

0xa679438e /\* 15 \*/

0x49b40821 /\* 16 \*/

0xf61e2562 /\* 17 \*/

0xc040b340 /\* 18 \*/

0x265e5a51 /\* 19 \*/

0xe9b6c7aa /\* 20 \*/

0xd62f105d /\* 21 \*/

0x02441453 /\* 22 \*/

0xd8a1e681 /\* 23 \*/

0xe7d3fbc8 /\* 24 \*/

0x21e1cde6 /\* 25 \*/

0xc33707d6 /\* 26 \*/

0xf4d50d87 /\* 27 \*/

0x455a14ed /\* 28 \*/

0xa9e3e905 /\* 29 \*/

0xfcefa3f8 /\* 30 \*/

0x676f02d9 /\* 31 \*/

0x8d2a4c8a /\* 32 \*/

0xfffa3942 /\* 33 \*/

0x8771f681 /\* 34 \*/

0x6d9d6122 /\* 35 \*/

0xfde5380c /\* 36 \*/

0xa4beea44 /\* 37 \*/

0x4bdecfa9 /\* 38 \*/

0xf6bb4b60 /\* 39 \*/

0xbebfbc70 /\* 40 \*/

0x289b7ec6 /\* 41 \*/

0xeaa127fa /\* 42 \*/

0xd4ef3085 /\* 43 \*/

0x04881d05 /\* 44 \*/

0xd9d4d039 /\* 45 \*/

0xe6db99e5 /\* 46 \*/

0x1fa27cf8 /\* 47 \*/

0xc4ac5665 /\* 48 \*/

0xf4292244 /\* 49 \*/

0x432aff97 /\* 50 \*/

0xab9423a7 /\* 51 \*/

0xfc93a039 /\* 52 \*/

0x655b59c3 /\* 53 \*/

0x8f0ccc92 /\* 54 \*/

0xffeff47d /\* 55 \*/

0x85845dd1 /\* 56 \*/

0x6fa87e4f /\* 57 \*/

0xfe2ce6e0 /\* 58 \*/

0xa3014314 /\* 59 \*/

0x4e0811a1 /\* 60 \*/

0xf7537e82 /\* 61 \*/

0xbd3af235 /\* 62 \*/

0x2ad7d2bb /\* 63 \*/

0xeb86d391 /\* 64 \*/

1. Turn In Requirements

Deliverables

1. E-mail an **electronic copy** of all your PEX files (.zip please) to your instructor. Make sure you include a documentation statement. Include sufficient information to enable your instructor to compile and run your program. Remember turn-ins may be in different languages, so explain in detail what run-time system or IDE is needed.
2. Provide a **hard copy** of your files, including the attached **cut sheet**, and a **screenshot** of your code’s functionality as well as a screenshot of the same inputs verified by your outside source.

**PEX 1 Cut Sheet (100 Points)**

|  |
| --- |
| **Name:**  **Section:**  **Grade: /75** |

|  |  |  |
| --- | --- | --- |
| **Requirements:** |  |  |
|  | **Some semblance of design** | **(5)** |
|  |  |  |
|  | **Implementation** |  |
|  | User interface | **(10)** |
|  | Completion of intermediate steps | **(45)** |
|  | Output verified correct | **(10)** |
|  |  |  |
|  |  |  |
|  | **Adherence to Programming Standards** | **(5)** |

|  |  |  |
| --- | --- | --- |
| **Penalties:** |  |  |
|  | **Vague/Missing Documentation (-5):** |  |
|  | **Missing Grade Sheet (-5):** |  |

|  |  |  |
| --- | --- | --- |
| **Total:** |  | **(75)** |