```
01:198:419:01 COMPUTER SECURITY > Assignments > Project 3
Rutgers
Spring 2022
                                     Project 3
                                                                                                                                                                                                                 Submission
(b)
           Home

✓ Submitted!

          Announcements
                                                                                   Submitting a file upload
                                     Due Apr 12 by 11:59pm
                                                                    Points 10
                                                                                                                  File Types zip
                                                                                                                                     Available Mar 10 at 8pm - Apr 14 at 12pm
                                                                                                                                                                                                                  Apr 9 at 12:57pm
                                                                                                                                                                                                                 Submission Details
Assignments
                                                                                                                                                                                                                 Download p3-mrg225.zip
                                     This assignment was locked Apr 14 at 12pm.
           <u>Piazza</u>
                                                                                                                                                                                                                 Grade: 6 (10 pts possible)
                                     \rightarrow Recitation notes \downarrow
           Modules
                                                                                                                                                                                                                  Graded Anonymously: no
                                     → Printable PDF version of this writeup ↓ ☑
                                     → Reference programs ↓ (website link &)
           Quizzes
?
                                                                                                                                                                                                                  Comments:
                                                                                                                                                                                                                   For vencrypt, Additional Test Failed -1 For
           Discussions
                                                                                                                                                                                                                   vdecrypt, Additional Test Failed -1 For
                                                                                                                                                                                                                   scrypt, Additional Test Failed -1 Grading
                                     Introduction
           Grades
                                                                                                                                                                                                                   sbencrypt with key 0 file test-a.txt failed!
                                                                                                                                                                                                                   Grading sbdecrypt with key 0 file test-a.txt
           Pages
                                                                                                                                                                                                                   failed! Total Score: 3.0 Additional test may
                                     This assignment contains three parts:
                                                                                                                                                                                                                   include empty key, empty data file and jpgs
                                                                                                                                                                                                                   If you try to modify anyfiles that you should
           Files
                                                                                                                                                                                                                   not touch, we will catch that and
                                     1. In part 1, you will implement a binary version of the Vigenère polyalphabetic substitution cipher.
                                                                                                                                                                                                                   automatically give you a zero! If your
          Syllabus
                                     2. In part 2, you will implement a stream cipher that uses a linear congruential keystream generator and a hashed key as a seed.
                                                                                                                                                                                                                   program cannot finish in 10 seconds, we will
                                     3. In part 3, you will modify the cipher in part 2 to operate as a block cipher that will shuffle pairs of bytes within each block based on the keystream and will apply
                                                                                                                                                                                                                   kill it and you will not receive any credit.
          Collaborations
                                         cipher block chaining (CBC) between blocks.
                                                                                                                                                                                                                   Grading sbencrypt with key 0 file test-a.txt
          Rutgers Libraries
                                                                                                                                                                                                                   failed! Grading sbdecrypt with key 0 file
                                                                                                                                                                                                                   test-a.txt failed! Total Score:6.0 If you try to
                                                                                                                                                                                                                   modify anyfiles that you should not touch,
          Student Instructional
                                     Environment
          Rating Survey
          Media Gallery
                                     This is an individual project. All work should be your own except for the referenced code for the hash function.
          My Media
                                     You may use Go, Python, Java, C, or C++ in your implementation.
           Zoom
                                     You should be able to implement this on any platform but you are responsible to make sure that the program runs on Rutgers iLab systems with no extra software.
          Cisco Webex
          Office 365
                                     Part 1: Binary Vigenère Cipher
          Reading List
                                     We covered the cipher in class. This was a cipher that was created in the 1500s and used through the late 1800s. It was designed for field use (that is, you could
                                     encrypt and decrypt using only a pencil and paper with no need for special equipment) and used a repeating key. The cipher requires arranging multiple substitution
                                     alphabets in a grid.
                                            ABCDEFGHIJKLMNOPQRSTUVWXYZ
                                           AABCDEFGHIJKLMNOPQRSTUVWXYZ
                                           B B C D E F G H I J K L M N O P Q R S T U V W X Y Z A
                                           CCDEFGHIJKLMNOPQRSTUVWXYZAB
                                           D D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
                                           EEFGHIJKLMNOPQRSTUVWXYZABCD
                                           F F G H I J K L M N O P Q R S T U V W X Y Z A B C D E
                                           G G H I J K L M N O P Q R S T U V W X Y Z A B C D E F
                                           H H I J K L M N O P Q R S T U V W X Y Z A B C D E F G
                                            IIJKLMNOPQRSTUVWXYZABCDEFGH
                                           J J K L M N O P Q R S T U V W X Y Z A B C D E F G H I
                                           K K L M N O P O R S T U V W X Y Z A B C D E F G H I I
                                           LLMNOPQRSTUVWXYZABCDEFGHIJK
                                           M M N O P Q R S T U V W X Y Z A B C D E F G H I J K L
                                           OOPQRSTUVWXYZABCDEFGHIJKLMN
                                           PPQRSTUVWXYZABCDEFGHIJKLMNO
                                           Q Q R S T U V W X Y Z A B C D E F G H I J K L M N O P R R S T U V W X Y Z A B C D E F G H I J K L M N O P Q
                                           SSTUVWXYZABCDEFGHIJKLMNOPQR
                                           TTUVWXYZABCDEFGHIJKLMNOPQRS
                                           UUVWXYZABCDEFGHIJKLMNOPQRST
                                           VVWXYZABCDEFGHIJKLMNOPQRSTU
                                           W W X Y Z A B C D E F G H I J K L M N O P Q R S T U V
                                           XXYZABCDEFGHIIKLMNOPORSTUVW
                                           YYZABCDEFGHIJKLMNOPQRSTUVWX
                                           ZZABCDEFGHIJKLMNOPQRSTUVWXY
                                     Each row of the grid contains the alphabet shifted by one position to the left compared to the previous row. To encrypt, you:
                                     1. Find the row indexed by the plaintext letter.
                                     2. Find the column indexed by the next character of the key
                                     3. The ciphertext letter is the intersection. The key is repeated to make it the same length as the message.
                                     (It is also common to index the row by the key letter and the column by the plaintext letter. Either way produces the same result.)
                                     Your program will implement a binary version of this cipher. The key can be either text or, preferably, binary data and is stored in a key file. A longer binary key
                                     allows the encryption to use a greater variety of substitution alphabets.
                                     The usage of the command is:
                                      vencrypt keyfile plaintext ciphertext
                                     This encrypts the plaintext file using the key in keyfile and produces the file ciphertext. Here's what you need to do (logically):
                                     1. Create a 256x256 grid.
                                         a. Row 0 contains 00 .. 0xff
                                         b. Row 1 contains 01 .. 0xff, 00
                                         c. Row 2 contains 02 .. 0xff, 00, 01
                                         d. Row 255 contains 0xff, 00 .. 0xfe
                                     2. To encrypt a byte n of plaintext, look up its value in the table:
                                         ciphertext_n = table [row=plaintext_n] [column=keydata_{n mod length(ciphertext)}]
                                                                     KEY -
                                                                     ----- 252 253 254 255
                                                                             252 253 254 255
                                                                             243 254 255 0
                                                                             254 255 0 1
                                                      3 4 5 6
                                                                             255 0
                                                    252 253 254 255
                                                                             248 249 250 251
                                                    253 254 255 0
                                                                             249 250 251 252
                                                    254 255 0 1
                                                                             250 251 252 253
                                                                             251 252 253 254
                                                    255 0 1 2
                                     If you think about the operations that take place, you will realize that you don't need a table since you can easily derive the value of the ciphertext from the key
                                     byte and plaintext byte.
                                     Then, create a decryption function
                                      vdecrypt keyfile ciphertext plaintext
                                     This reads the key from keyfile and decrypts the contents in cipherfile into the file message.
                                     Hints & testing
                                     If you find this program getting long, you might be approaching it incorrectly. The entire encryption can be one while loop with one line of code within it! You can
                                     implement a table if you'd like, but it's not necessary.
                                     Test your programs thoroughly. Come up with different test cases and validate them.
                                     With this cipher, a key with bytes of 0 will always produce plaintext. A key with bytes of 1 will produce shifted data. For example, the string ABC will produce the
                                     ciphertext BCD.
                                     Your program should not assume any maximum size for files being encrypted. That is, it should handle multi-gigabyte files without a problem. Hence, reading a
                                     stream of data is a more reasonable implementation than trying to read the file into memory all at once.
                                     You cannot assume a maximum size for the key file but may assume that it's a reasonable size where you can allocate a memory buffer and read in the entire key
                                     file.
                                     There is no distinction between text and binary data: a file is just a stream of bytes. You should not attempt to read lines or take any special actions for, say, newline
                                     characters, carriage returns, or null characters.
                                     Printing input & output of data (as hex #s, for example) can help you see what's going on in your program.
                                     The Linux (and macOS) od command dumps binary data and may be useful for inspecting your output. The command
                                      od -t xC keyfile
                                     shows the contents of keyfile as a series of hexadecimal bytes.
                                     Part 2: Stream cipher
                                     We also covered stream ciphers in class. This type of cipher simulates the one-time pad by using a keystream generator to create a keystream that is the same
                                     length as the message.
                                                                seed
                                                   key
                                                Keystream
                                                                     S_0 | S_1 | S_2 | S_3 | S_4 | S_5 | S_6 | S_7 | S_8
                                                generator
                                                                     \oplus \oplus \oplus \oplus \oplus \oplus \oplus \oplus
                                                                     C_i = S_i \oplus P_i
                                     The keystream generator is a pseudorandom number generator, and the seed will be derived from the password. You will always see the same sequence of numbers
                                     for the same seed.
                                     To implement this cipher, you will:
                                     (a) Implement a linear congruential generator
                                     This is a trivial formula that is described here . This is one of the best-known and widely-used pseudorandom number generators. Each pseudorandom number is
                                     function of the previous one and defined as:
                                     X_{n+1} = aX_n + c \mod m
                                     where:
                                     X_{n+1} is the next pseudorandom # in the sequence
                                     X_n is the number before that in the sequence
                                     m is a modulus. We will be working only on bytes in this assignment, so you will use 256 for the modulus (since that is 2^8 and will produce a range of values that fit
                                     within a byte).
                                     The values a and c are magic parameters. Certain values were found to produce better sequences of data. You will use the same parameters that are used in ANSI C,
                                     C99, and many other places:
                                     Modulus, m = 256 (1 byte)
                                     Multiplier, a = 1103515245
                                     Increment, c = 12345
                                     Implementing this generator is only three lines of code!
                                     By using a well-known formula, your output should be the same regardless of the programming language or operating system you use.
                                     (b) Convert the password to a seed
                                     The seed for a pseudorandom number generator is just a number. For this program, instead of asking users to use a number as a key, you will let them use a textual
                                     password. You will then apply a hash function to this password to create a seed for the keystream generator.
                                     To create the seed, we will use a hash function that works well and is easy to implement. This is the sbdm hash that is used in gawk, the sbdm database, Berkeley
                                     DB, and many other places. You can find the C code for it here 

:: ::
                                      static unsigned long
                                       sdbm(unsigned char *str) {
                                               unsigned long hash = 0;
                                               int c;
                                              while (c = *str++)
                                                       hash = c + (hash << 6) + (hash << 16) - hash;
                                               return hash;
                                     You should be able to translate it to whatever language you're programming in pretty easily. This implementation is also three lines of code!
                                     As with the previous step, this implementation should ensure that your output will be the same regardless of the programmer, programming language, or operating
                                     system.
                                     (c) Apply the stream cipher
                                     The ciphertext is generated byte by byte and is simply:
                                         ciphertext<sub>i</sub> = plaintext<sub>i</sub> ⊕ keytext<sub>i</sub>
                                     Each byte of plaintext is XORed with the next byte from the keystream generator to produce a byte of ciphertext. Because applying an exclusive-or of the same key
                                     a second time undoes the first exclusive-or, you only need to implement one command.
                                             password
                                                                      seed
                                                                      542326702784
                                              "monkey01"
                                                                          8090132
                                                                        keystream
                                                                                                             0xb2 -
                                                                                                                                 \rightarrow 0x80 \rightarrow 0xb9
                                                                                                                        0x03
                                                                                                ▶ 0xdb
                                                                         generator
                                                                     plaintext
                                                                         "ABCDE"
                                                                                                  0xfc
                                     Usage:
                                      scrypt password plaintext ciphertext scrypt password ciphertext plaintext
                                     The password is a text string. The parameters plaintext and ciphertext are files. The same program can be used to encrypt or decrypt messages.
                                     Hints & testing
                                     Before you even start implementing your cipher, you should test that your seed and keystream generation works as expected. It's important that different cipher
                                     implementations are all compatible - even if they are written by different people in different languages and on different platforms.
                                     You are provided with a program called prand-test that lets you enter a password and see the seed and enter either a password or seed to get a pseudorandom
                                     stream of bytes.
                                     The usage is:
                                      prand-test [-p password | -s seed] [-n num]
                                     If you supply a password, then you can see the seed:
                                     $ ./prand-test -p monkey01
                                      using seed=5423267027848090132 from password="monkey01"
                                     It's possible that the seed may be different if you're using python, which implements arbitrary-precision arithmetic instead of 64-bit integers but your sequence
                                     should be the same since we're taking the modulus of the results.
                                     To see a sequence of pseudorandom bytes, use the -n parameter. Here are the first five bytes from the password monkey01:
                                      ./prand-test -p monkey01 -n 5 using seed=5423267027848090132 from password="monkey01"
                                     1891783128
                                     185
                                     You can also test the sequence from a seed number. Here are the first four bytes from the seed 85:
                                      ./prand-test -s 85 -n 5 using seed=85
                                     1069124820954
                                     Part 3: Block encryption with cipher block chaining and padding
                                     This is an enhancement of Part 2 to add the concepts of:

    Processing data in 16-byte blocks

                                      • Padding – adding it and removing it correctly
                                      • Shuffling bits within a block

    Cipher block chaining

                                     We modify the stream cipher above to have it operate on 16-byte blocks instead of bytes. This turns it into a form of block cipher. A block cipher normally uses
                                     multiple iterations (rounds) through an SP-network (substitutions & permutations) to add confusion & diffusion. Confusion refers to changing bit values as a
                                     function of the key so that each bit of the ciphertext is determined by several parts of the key. Diffusion refers to the property that a change in one bit of plaintext
                                     will result in many bits of the ciphertext changing (about half).
                                     In this implementation, we will not use multiple rounds of an SP network. Instead, we will keep the mechanisms of the stream cipher in place but enhance it in two
                                     ways: cipher block chaining and shuffling bytes within the block.
                                     Padding
                                     Block ciphers work on a block (a group of bytes) of data at a time. In our case, we will be processing 16-byte blocks.
                                     Not every file is an exact multiple of 16 bytes so we may encounter a partial block at the end. To support this, every block cipher needs to support padding, which is
                                     the mechanism for adding extra bytes to fill the block. Padding must be added in such a way that we can detect and remove the padding when decrypting a
                                     message.
                                     The way you will implement padding is by adding between 1 and 16 extra bytes at the end of the file. Each byte of the padding data is a number that tells you how
                                     many bytes were added. This is a technique that allows you to know how much padding needs to be removed when decrypting and writing the plaintext output.
                                     In the case that the file was an exact multiple of 16 bytes, we add an entire extra block of padding. Otherwise, we would never know if we had padding. Simply
                                     looking at the last byte of the last block of the file will tell us how much padding to ignore.
                                     Here are a few examples. In the first, the text "I am done." takes up 10 bytes so we have 6 bytes left over. Each of those bytes will contain a pad byte with a value
                                     of 6. Note that this is not the ASCII character 6 but the number 6.
                                                                                                                                        06
                                                                                d
                                                                                                                   06
                                                                                                                          06
                                                                                                                                 06
                                                                                                                                               06
                                                                                                                                                      06
                                                                                                     е
                                                           a
                                                                 m
                                                                                       0
                                                                                              n
                                     In this example, we have the text "This is the end". It takes up 15 bytes, so we need to add one byte of padding. This padding byte contains the value 1.
                                                                                                                                 е
                                                                                                                                               d
                                     In the final example, we have the text "This is the end." with a period at the end. This message takes up exactly 16 bytes. Because of this, we need to add an
                                     extra block filled with bytes containing the number 16.
                                                                                                            h
                                                                                        S
                                                                                                                    е
                                                                                                                                  е
                                                                                                                                        n
                                                                                                                                                d
                                                                  S
                                                    16
                                                          16
                                                                  16
                                                                         16
                                                                                16
                                                                                                            16
                                                                                                                   16
                                                                                                                          16
                                                                                                                                 16
                                                                                                                                        16
                                                                                                                                               16
                                                                                       16
                                                                                              16
                                                                                                     16
                                                                                                                                                      16
                                     When you decode the message, you will need to remove the padding.
                                     We will not use multiple rounds of an SP network. Instead, we will keep the mechanisms of the stream cipher in place but enhance it in two ways: cipher block
                                     chaining and byte shuffling within a block (a simple form of permutations).
                                     Cipher block chaining
                                     Stream ciphers have no diffusion. The change of a bit in plaintext will generally affect only that bit in ciphertext. We will add diffusion across the output by adding
                                     cipher block chaining (CBC). With cipher block chaining, we exclusive-or the previous block with the next block.
                                                                               c_i = E_K(m_i) \oplus c_{i-1}
                                                          Plaintext<sub>0</sub>
                                                                                            Plaintext<sub>1</sub>
                                                                                                                                       Plaintext<sub>N</sub>
                                                         Block cipher
                                             Key
                                                                               Key
                                                                                                                                     Block cipher
                                                                                          Block cipher
                                                                                                                          Key
                                                                                                            -----
                                                         Ciphertext<sub>0</sub>
                                                                                           Ciphertext<sub>1</sub>
                                                                                                                                      Ciphertext<sub>N</sub>
                                                          Block 0
                                                                                                                                       Block N
                                                                                            Block 1
                                     Byte shuffling
                                     Confusion is roughly determined by the seed and the pseudorandom output of the keystream generator in this implementation, but we will enhance the degree of
                                     confusion by shuffling bytes of the block.
                                     We will use the key to determine which sets of bytes in the block to exchange (swap). For each 16-byte block, do the following:
                                      for (i=0; i < blocksize; i=i+1)
                                        first = key[i] & 0xf (lower 4 bits of the keystream)
                                        second = (key[i] >> 4) & 0xf (top 4 bits of the keystream)
                                        swap(block[first], block[second]) (exchange the bytes)
                                     Cipher operation
                                     The flow of the cipher is the following:
                                     Start by creating an initialization vector (IV) for applying CBC to the first block. This will be the first 16 bytes read from the keystream generator. These bytes will
                                     not be used for anything else.
                                     Then, for each 16-byte plaintext_block i:
                                     1. If it is the last block, add padding. This will be an amount from 1 through 16 bytes
                                        (e.g., finish up a block or add a new block). The padding is added before any encryption or shuffling takes place.
                                     2. Apply CBC: temp_block<sub>i</sub> = plaintext_block<sub>i</sub> \oplus ciphertext_block<sub>i-1</sub>. Use the initialization vector if this is the first block (i=0).
                                     3. Read 16 bytes from the keystream.
                                     4. Shuffle the bytes based on the keystream data.
                                     5. ciphertext_block<sub>i</sub> = temp_block<sub>i</sub> ⊕ keystream<sub>i</sub>.
                                     6. Write ciphertext_block<sub>i</sub>.
                                     You will use the same password hashing and keystream generator as in Part 2.
                                     You will need to write two programs for this part, one to encrypt and another to decrypt:
                                               sbencrypt password plaintext ciphertext sbdecrypt password ciphertext plaintext
                                     As with Part 2, the program will take a password string, which will be hashed and used as a seed for the keystream generator. The parameters plaintext and
                                     ciphertext are both file names.
                                     Reference programs
                                     It's important that encryption software works consistently across multiple systems regardless of author, programming language, or operating system. I should be
                                     able to encrypt a message on my Mac and expect you to be able to decrypt it with your program on your Raspberry Pi running Linux.
                                     You are provided with reference versions of the programs that you can use to compare with yours and, perhaps, help debug your code. The linux directory
                                     contains intel architecture linux versions of the executables and the macos directory contains macOS versions. The samples directory contains some sample files
                                     you can use for testing, but you should also create your own files and keys to test your program thoroughly. Be sure to test edge cases, such as empty files, one-
                                     byte files, and files that require different amount of padding (for the block cipher).
                                     Here are the programs provided:
                                     Binary Vigenère Cipher
                                      vencrypt [-d] [keyfile | -k key] plaintext ciphertext
                                     Encrypt a plaintext file to create a ciphertext file using key data stored in keyfile. Alternatively, you can specify a textual keyfile with the -k option. For example,
                                      vencrypt -k monkey01 file.txt file.enc
                                     Will use the bytes in the string monkey01 as the key. This limits the key space but may be useful for debugging.
                                     The -d flag turns on debugging information and shows you what lookups are taking place (note that you do not need to implement a table but can do so if you find
                                     that easier).
                                     Stream Cipher - keystream test
                                     Before you test your cipher, make sure that your password hash and pseudorandom number generator are producing the proper results. You can test this with the
                                      prand-test program:
                                      prand-test [-p password | -s seed] [-n num]
                                     If the program is supplied a password with the password will be hashed and the result shown.
                                     The -n parameter lets you specify the number of pseudorandom numbers to be printed. The default is 0.
                                     If you just want to see the list of pseudorandom numbers generated from a specific seed, you can specify a seed number instead of a password with the -s
                                     parameter.
                                     Stream Cipher
                                     When you are confident that your keystream generator is providing the proper output, you can test the stream cipher with the scrypt command:
                                      scrypt [-d] password plaintext ciphertext
                                     Encrypts a plaintext file into a ciphertext file using a keystream derived from the password string. The same command decrypts a ciphertext file into a plaintext file:
                                      scrypt [-d] password ciphertext plaintext
                                     The -d flag turns on debugging mode and shows the series of xor operations from the source file to the output file.
                                     Block Cipher
                                     The block cipher reference program is:
                                      sbencrypt [-d] password plaintextfile ciphertextfile
                                     This encrypts a plaintext file into a ciphertext file using a keystream derived from the password string. The same command:
                                      sbdecrypt [-d] password ciphertextfile plaintextfile
                                     decrypts a ciphertext file into a plaintext file using a keystream derived from the password string.
                                     For both commands, the -d flag enables debugging, showing the sequence of shuffling per block.
                                     What to submit
                                     Place your source code into a single zip file. If code needs to be compiled, please include a Makefile that will create the necessary executables.
                                     We don't want to figure out how to run your program. We expect to:
                                     1. unzip your submission
                                     2. run make if there's a Makefile
                                     3. Set the mode of the programs to executable: chmod u+x vencrypt vdecrypt scrypt sbencrypt sbdecrypt
                                      4. Run the commands as:
                                         ./vencrypt keyfile plaintext ciphertext
                                         ./sbencrypt password plaintext ciphertext
                                     If you are using python, you can submit either:
                                     A. A vencrypt shell script that runs the program or
                                     B. (preferably) a program named vencrypt that that contains your source and starts with the line:
                                      #!/usr/bin/python3
                                     If you are using Java, you will have a simple makefile that compiles the Java code to produce class files. The vencrypt program will be a script that runs the java
                                     command with the necessary arguments. The file vencrypt will contain content like this:
                                      #!/bin/bash
                                      CLASSPATH=. java Vencrypt "$@"
```

Immersive Reader

Xiaoxiao(Arist) He. Apr 18 at 2:55am

Test your scripts on an iLab machine to make sure they work prior to submitting.

◆ Previous