Project 1 : Designing and attacking symmetric-key cryptosystems

Part 1 Due: 11:59pm, Thur, March 5

Part 2 Due: 11:59pm, Thur, April 2

Summary: In Part 1 you will design and implement a symmetric-key cryptosystem. You can combine and modify the systems we have used in class, or you may do something completely different. You will use your cryptosystem to encrypt several messages with secret keys.

In Part 2, we will randomly assign each student a different target cryptosystem designed by a classmate; your goal will be to break the one assigned to you. You will have access to the encryption and decryption algorithms that your classmate designed, but you will not be given the secret keys. You will receive (small amounts of) bonus points if (i) you exploit weaknesses in the target cryptosystem, or (ii) your own cryptosystem resists attacks from the classmate assigned to attack it. In this way you will be incentivized to make your cryptosystem strong and to try hard to attack the one you've been assigned. However, you will still be able to achieve a high grade without these extra points.

This back and forth between the cryptographer designing a cryptosystem and the cryptanalyst attacking it is meant to mimic how cryptology develops in the real world. You will get to play both roles - cryptographer and cryptanalyst.

Collaboration Policy: You can discuss topics related to programming or cryptographic algorithm design with your classmates or others. You may also research cryptography on the internet or from books (this is certainly not required). But you must design and write your own algorithm and report. In particular, you should understand how your code works, and be able to justify your design choices.

Part 1 - Due 11:59pm, Thur March 5

For this part, you will submit four files on Blackboard as *Project 1, Part 1.* The files are described below.

A "symmetric-key" cryptosystem is a system for two people, Alice and Bob, who have agreed on a single secret key before-hand, to encrypt and decrypt messages sent over an insecure channel. The shift cipher, Vigenere cipher, and one-time pad are all examples of symmetric-key cryptosystems. There are various notions of security, but essentially what we desire is that an eavesdropper Eve who sees the ciphertexts cannot gain much useful information about the plaintexts.

Your task is to design, implement, and describe a symmetric-key cryptosystem with the following specifications:

- A plaintext to be encrypted consists of a string containing only the characters A,B,...,Z. A plaintext of any length that is a multiple of 12 can be encrypted. The encrypted ciphertext must be a string of characters (but not necessarily just capital letters).

- The key can have a variety of forms, but you must explicitly specify which keys are allowable. There can be at most $26^{12} = 95428956661682176$ valid choices of key (so a one-time pad will not work). The key must be describable with a string (but not necessarily just capital letters). So, for instance, one example of the set of valid possible choices of key is

  $$\{s : s \text{ is a string consisting of capital letters of length 12}\}.$$

  Another example is

  $$\{s : s \text{ is a binary (0,1) string of length 56}\}.$$

  (the last example satisfies the condition on number of possible keys since $2^{56} < 26^{12}$.

- The encrypted message can be any length (as long as it's not many orders of magnitude bigger than the plaintext). But the encrypted message needs to be uniquely decryptable.

What to turn in:

1. **Code Portion:** A Sage/Jupyter notebook named `cryptosystem.ipynb` that contains the following:

   - A function called `my_encrypt(plaintext, key)` that takes in a string of capital letters and a key. It returns the ciphertext corresponding to encrypting `plaintext` using your algorithm with key equal to `key`.
   - A function called `my_decrypt(ciphertext, key)` that takes in a string and key. It returns the plaintext corresponding to `ciphertext` with encryption key equal to `key`. Because it is decrypting, it must have the property that

     `my_decrypt(my_encrypt(plaintext,key),key) = plaintext`,

     for any valid `plaintext` and `key`.

   You want to make your cryptosystem as strong as possible. But it essential that your cryptosystem satisfies the two bullet points above; otherwise the student who is assigned to target your cryptosystem will be given something that does not work properly. Make sure to comment your code to make it easier to read.

2. **Cryptosystem Report**: A written description in a pdf document called `cryptosystem_report.pdf` of how your encryption algorithm works and why you believe it to be resistant against attack. You should describe your algorithm in sufficient detail so that someone who has not seen

your code has a good idea of what you are doing. You must explicitly specify which keys are allowable, as described above.

It should be written for someone who understands programming and math in general, but does not know anything about specific sage/python syntax. The exact format of this report is up to you. It can be hand-written (legibly) and then scanned (legibly) into a `pdf` document. Or it can be be made on a computer. It must be in `pdf` form (it is generally easy to convert other forms such as `.doc` to `pdf`). It may be useful to include diagrams indicating what your algorithm does.

3. **Ciphertext Messages and Tests**: A text file named `ciphertexts.txt` which will be given to the classmate who is assigned to attack your cryptosystem. Use the following format:

```
plaintext_test = "THEWORLDISNO"
key_test = "$1"
ciphertext_test = "$2"
plaintext_known = "TOMORROWNEVERDIESFORYOUREYESONLYONHERMAJESTYSSECRETSERVICEIN"
ciphertext_known = "$3"
ciphertext_960_english = "$4"
ciphertext_96_english = "$5"
ciphertext_960_unknown = "$6"
ciphertext_96_unknown = "$7"
```

You replace each $\$i$ with the following

- $1 is a simple key of your choice (this is just for test purposes, so it doesn't need to be hard to guess)
- $2 is `my_encrypt(plaintext_test, key_test)`, where `plaintext_test` is as above.
- $3 is `my_encrypt(plaintext_known, key_known)`, where `plaintext_known` is as above, and `key_known` is a key of your choice (make it hard to guess).
- $4 is `my_encrypt(plaintext_960_english, key_960_english)`, where `plaintext_960_english` is a plaintext of your choice of exactly 960 characters consisting of English text in capital letters with all punctuation and spaces removed (make it hard to guess), and `key_960_english` is a key of your choice (make it hard to guess).

- $5 is `my_encrypt(plaintext_96_english, key_96_english)`,
  where `plaintext_96_english` is a plaintext of your choice of exactly 96 characters consisting of English text in capital letters with all punctuation and spaces removed (make it hard to guess), and `key_96_english` is a key of your choice (make it hard to guess).
- $6 is `my_encrypt(plaintext_960_unknown, key_960_unknown)`,
  where `plaintext_960_unknown` is a plaintext of your choice of exactly 960 capital letters (doesn't have to be any language; make it hard to guess), and `key_960_unknown` is a key of your choice (make it hard to guess).
- $7 is `my_encrypt(plaintext_96_unknown, key_96_unknown)`,
  where `plaintext_96_unknown` is a plaintext of your choice of exactly 96 capital letters (doesn't have to be any language; make it hard to guess), and `key_96_known` is a key of your choice (make it hard to guess).

4. **Plaintext Messages/Keys:** A text file named `secret_plaintexts.txt` with the secret messages and keys you chose above. The student who is assigned to attack your cryptosystem will not be given this file - that student will try to guess the contents. The format should be as follows:

```
key_known = "*"
plaintext_960_english = "*"
key_960_english = "*"
plaintext_96_english = "*"
key_96_english = "*"
plaintext_960_unknown = "*"
key_960_unknown = "*"
plaintext_96_unknown = "*"
key_96_unknown = "*"
```

   where you replace the `*` character on each line with the value chosen in the previous part whose name is given at the beginning of the line. For instance, the first `*` is replaced with the value of `key_known` you chose above.

Part 2 - Due 11:59pm, Thur April 2

You will submit one or two files as *Project 1, Part 2* in Blackboard.

For this part, you will be emailed the "Code Portion", "Cryptosystem Report", and "Ciphertext Message and Tests" files submitted by some other student in the course. Your goal is to find weaknesses in this target cryptosystem, and in particular to determine some or all of the contents of the "Plaintext Messages/Keys" file written by that student (which you will not be given).

What to turn in:

1. **Cryptanalysis Report:** A written description in a pdf document called `cryptanalysis_report.pdf` that describes any weaknesses in the target cryptosystem. Serious weaknesses may allow you to decrypt some of the ciphertexts you were given and/or recover the keys. Somewhat less serious weaknesses may allow you to guess part of the plaintext or part of the key. If you can distinguish, with probability greater than 1/2, encrypted English from encrypted random text, that is also a weakness. Another sort of weakness might be if you find an improvement of a brute force attack that is still too computationally intensive to allow you to actually decrypt the messages, but would be feasible if you had more computational resources. If you can find no weaknesses, give some justification for why you think the target cryptosystem is so strong, and give some statistical evidence that there aren't statistical patterns in the encryptions of English text.

   This can be hand-written and then scanned, or made with a computer. You may also include computer code that you used to analyze/break the target cryptosystem (by, say, saving a Jupyter notebook as a pdf and appending it to your report).

2. **Decrypts (Optional, Bonus):** Call this `decrypts.txt`. The submission format is exactly the same as for "Plaintext Messages/Keys" above, but replace each `*` with your guess for the value for the target cryptosystem.

Grading Rubric

Total is 100 points (but it is possible to earn higher via the two sources of bonus points).

- Part 1: 60 points
- Part 2: 40 points
- Design bonus points: 0.5 points for each entry in your submitted `plaintexts.txt` file that the student assigned to target your cryptosystem was *unable* to correctly guess.
- Attack bonus points: In your `decrypts.txt` file, if you guess the first `*` correctly, you will receive 2 bonus points. For each of the other `*`'s that you correctly guess, you will receive 1 bonus point.

If you are taking MAT 459 concurrently you will receive a PASS/FAIL grade for writing on this assignment based on your Cryptosystem Report and Cryptanalysis Report.