In this assignment, you will be asked to
1. implement a cipher,
2. analyze a newspaper article about secure email, and
3. securely submit your assignment encrypted with your own cipher.

Let's look at each part in detail.

1. A Simplified Enigma Cipher (50 points)

In the first part of this assignment, you will implement a simple but rather strong cipher, which we will call the Enigma-349. If you recall from recitation, the German Enigma machine was a polyalphabetic cipher which encrypted a text message one letter at a time. This is good for confusion, but the diffusion is poor. So, you will get the output of a simplified Enigma machine from a transposition cipher that will apply a given permutation on blocks of 16 characters. Then, you will build this up into a block cipher by implementing a variant of one of the cipher chaining algorithms we saw in class. Each of the three parts will have its own key. The overall cipher is summarized in the figure below.

Additionally, you will write a simple utility that will allow you to encrypt arbitrary binary files, not just text. Let's look at each part in detail.

1.1 The Initial Permutation (10 points)

The Enigma-349 takes as input strings drawn from the 26 uppercase letters "A−Z". These strings will have a length that is a multiple of 16 characters. The first
transformation you will implement divides this input into blocks of 16 characters and applies to them a permutation provided as a key.

For uniformity, this key will itself be represented as a string, and more specifically it will be a permutation of the 16 uppercase letters "A-P". It describes a permutation of a 16-character block as follows: if the \( i \)th element of this key is the \( j \)th letter of the alphabet, your permutation will move the character in the \( i \)th position in the block to the \( j \)th position. For example, assume that your permutation key \( k_1 \) is "EIOPADFGHJLKCBNM", then the block \( b = "KVGFTGMGPCARHG" \) will be permuted into "THRMKGMGVPCAPGGF". Indeed, the "e" in \( k_1 \) specifies that the first letter in \( b \) (i.e., "K") should be moved to the 5th place in the output block, the "i" specifies that the second letter ("V") should be moved to the 9th place, etc.

The purpose of this initial permutation is to add diffusion to the Enigma-based middle substitution cipher in part (1.2), which will output one character for each character of input. Indeed, although this output will depend on all the previous inputs (due to the stepping mechanism — see below), the diffusion is rather poor.

1.2 The Middle Substitution (20 points)

The middle substitution will be a simplified Enigma machine inspired to the device used by the Germans in World War II (it is actually very similar to the model known as the 1928 Army Enigma, or Enigma G). It will consist of three components, described next.

- **Entry device**: The entry device is a set of 26 contacts corresponding to the 26 uppercase letters "A-Z" of the alphabet. This is where the letter to encipher is input, and this is where the result of the encryption is read off. In an implementation, it is the input of the initial permutation (1.1) and the output of the final transformation (1.3).

- **Rotors**: The variant of the enigma you will be implementing consists of 3 rotors. A rotor is a thick disc with 26 contacts on each face and the 26 letters "A-Z" engraved on the edge of the disc — the picture on the right shows two rotors of a later Enigma (with numbers instead of letters on the edge). Inside each rotor are 26 circuits that connects pairs of contacts on opposite faces. The exact wiring is specified by the following table:

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotor I:</td>
<td>E K M F L G D Q V Z N T O W Y H X U S P A I B R C J</td>
</tr>
<tr>
<td>Rotor II:</td>
<td>A J D K S I R U X B L H W T M C Q G Z N P Y F V O E</td>
</tr>
<tr>
<td>Rotor III:</td>
<td>B D F H J L C P R T X V Z N Y E I W G A K M U S Q O</td>
</tr>
</tbody>
</table>

This table is read as follows: the first row lists the 26 input characters engraved on the edge of each rotor and they label the contacts on the right face of the disc. For each input character, a circuit maps the corresponding contact on the right face (corresponding to the engraved letter) to the contact on the left face specified by the appropriate row of the table. For example, rotor I connects "A" to "E", "B" to "K", and so on until "Z" to "J". Similarly, rotor II connects "A" to itself, "B" to "J", etc., and rotor III connects "A" to "E", "B" to "D", and so on up to "Z" to "O". This is a variant of the actual wiring of the Enigma G.
When the rotors are in place, they are aligned. There is a cover hiding them except for a little window that shows only one letter for each of them. Therefore, when an Enigma machine is operational, there are always three letters showing, one for each rotor. This is important as these letters will initially be the key for the machine.

As the name implies, the rotors rotate. This allows to set the initial 3-letter key, one for each rotor. The Enigma also contains a stepping mechanism that causes the rotors to rotate with respect to each other when the machine operates. The stepping mechanism functions as follows:

- After entering each input, the right rotor (rotor I) rotates by one notch upward, so that if "F" was showing through the window before the input, then "G" will show after.
- When the right rotor (rotor I) goes from "Z" to "A", the middle rotor (rotor II) advances by one step.
- When the middle rotor (rotor II) goes from "Z" to "A", the left rotor (rotor III) advances by one step.

This stepping mechanism has the effect of changing the way the output from one rotor is connected to the input of the next. This configuration changes after encoding every character between the input device and rotor I and between rotors I and II, every 26 characters between rotors II and III, and every 676 (= 26x26) characters between rotor III and the reflector (see next).

**Reflector:** The reflector is a fixed set of circuits that maps each of its 26 contacts to another one of these 26 contacts. The mapping is given by the following table, which is read as for the rotors.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reflector:</td>
<td>R Y U L W P H G S X N D O K M F V A I Z C Q E J B T</td>
</tr>
</tbody>
</table>

The reflector does not move and its contacts touch the output contacts of rotor III.  

We will now see how all these pieces fit together to allow us to encrypt a message. Let us begin by encrypting a single character. In short, the alignment of the rotors and the contacts between them will form a circuit that will transform an input letter through rotors I, II and III, travel through the reflector, and then back through rotors III, II and I. Two examples are depicted graphically in the figure on the right of this text.

Let's describe this process in more detail. As we start, the rotors will be in some initial configuration, given by the three letters showing through the window. A letter to be enciphered will first enter the left face of rotor I from the input device through a contact that depends on the configuration of rotor I. Rotor I will transform this letter to the letter on its left face, which will be mapped to some right contact of rotor II based on its own configuration. Rotor II will transform it into another letter and pass it on via a left contact to a right contact of rotor III who will transform it similarly. Once a letter comes out of the left face of rotor III, it is passed to the reflector, which sends it back to rotor III via the left face. Rotor III will produce a transformed letter via its right face. This character will flow through the circuits of rotor II and then rotor I, being transformed at each intermediate step. The final output will eventually emerge through the entry device where the original letter was entered. Only at this point will the stepping mechanism take effect.

---

1For your information, the original Enigma G allowed the operator to choose the order of the rotors, and the stepping mechanism was more complicated than what we are implementing. Later Enigma machines had more rotors and also a plugboard that further strengthened the cipher. You are not asked to implement any of this. The original Enigma machines received their input from a mechanical keyboard and produced their output on a lit display. You can find a video describing the basic operation of the Enigma on YouTube. Here is a mode detailed video with some commentary about how it was cracked.
move the appropriate rotors.

An example will help — its step-by-step simulation is depicted on the top half of the above figure. Assume that the initial configuration of the rotors is "AYX" and that we want to encrypt the letter "I". The following will happen:

- Because rotor I is in position "X", the input "I" will be connected to "F" on its right face (we need to shift "I" by 23 positions up).
- The table for rotor I above tells us that "F" becomes "G" on the left face.
- Now, rotor II is in position "Y", which is one notch up from rotor I. Therefore "G" on the left face of rotor I touches "H" on the right face of rotor II.
- The table for rotor II maps "H" to "U".
- Because rotor III is in position "A", which is 2 notches up from rotor II, "U" on the left face of rotor II touches "W" on the right face of rotor III.
- The table for rotor III maps "W" to "U".
- Rotor III is in position "A", which happens to be aligned with the reflector, so that "U" on the left face of rotor III arrives as "U" to the reflector.
- The table for the reflector changes "U" to "C".
- Again, the reflector and rotor III are aligned, so that "C" remains "C" when it enters the left face of rotor III.
- Going from left to right, rotor III transform "C" into "G".
- Rotors III and II are still 2 notches apart, but this time we need to go down, which maps "G" to "E".
- In the reverse direction, rotor II maps "E" to "Z".
- Rotors II and I are still 1 notch apart, which rewrites "Z" to "Y".
- Rotor I takes "Y" to "O" going from left to right.
- Now, rotor I is still in configuration "X", which means that we need to shift this letter by 23 positions down as it goes through the entry device. This produces "R", which is our final result.
- Finally, the initial configuration advances to "AYY" in preparation for the next input character. Were we to encrypt "I" again, it would go through the sequence of transformations shown on the bottom half of the figure.

This all looks very complicated when done by hand, as we just did, but it is very simple to implement.
To decrypt a ciphered message with an Enigma machine, all you need to do is to reset it to the initial configuration of the original message, and enter the ciphertext letters one at a time. The cleartext will come out as if by magic!

To test your code, the encryption of the cleartext "HELLOWORLD" starting with configuration "TST" is "ABEIYLIIXV" with final configuration "TTD". You may also want to try to encrypt and decrypt your own test suites.

1.3 Final Text-Block Cipher Chaining (10 points)

So far, each 16-characters block appears to be encrypted separately from the previous blocks. This is not completely true because of the stepping mechanism in the middle substitution — see (1.2). Just to be sure, we will scramble it a little bit more: we will group the output produced by (1.2) into 16-character blocks and combine them with an initialization vector which will act as a third component of our key. The result of this combination will be combined with the second block output by (1.2), and so on. This will produce a block cipher similar to (but somewhat different from) the ones we have seen in class. We call this phase text-block cipher chaining.

The chaining is based on a block-level transformation that takes as input a 16-character block initialization vector (either $k_3$ or the previous block output by the chaining) and a 16-character block produced by the final permutation in (1.2), and it returns as output a 16-character block (which will be used as the initialization vector for the next block). The block-level transformation is shown in the following figure:

![Diagram of text-block cipher chaining]

The transformation combines the characters in corresponding positions in the two inputs by simply adding them modulo 26. For example, if our initialization vector is "QWERTYUIOPASDDFG" and we use the block "THRMKGMVPCAPGGF", the result would be "JDVDDEGOJECSSJLL" (which will be used as the initialization vector of the next block produced by the final permutation, and so on). To understand how it is computed, notice that "Q" is the 16th letter of our alphabet (counting from zero) and "T" is the 19th. Since $16 + 19 \mod 26 = 35 \mod 26 = 9$, the first letter of the output is "J", the 9th letter of our alphabet. All other characters are computed in the same way, position by position.

1.4 Encoding Binary Files (10 points)

Since the Enigma-349 works exclusively on the alphabet "A-Z", we need some mechanism to express an arbitrary string into it. To do so, you are asked to implement the following simple transformation from a generic byte into a pair of uppercase letters, and vice versa:

- **Encoding**: To encode an 8-bit byte $b_1b_2b_3b_4b_5b_6b_7b_8$, you will break it into two 4-bit half-bytes, i.e., $b_1b_2b_3b_4$ and $b_5b_6b_7b_8$, and map each of them to a character in the range A-P so that "0000" corresponds to "A", "0001" to "B" and so on up to "1111" which corresponds to "P". Then, the encoding of $b_1b_2b_3b_4b_5b_6b_7b_8$ is the 2-character string given by the encoding of $b_1b_2b_3b_4$ followed by the encoding of $b_5b_6b_7b_8$. 
- **Decoding:** A pair of characters in the range A-P are decoded by performing the inverse operation. Because (1.2) and (1.3) operate on blocks of 16 characters, when encoding a file or a string you will need to add padding so that the output has length a multiple of 16 letters, and you will need to record its original length so that you know how many trailing characters to discard when decoding it. This is how you are going to do it:

  - **Encoding:** your input is going to be pre-processed as follows:
    1. Compute the length in bytes of your input string or file. Write this number in binary format over 64 bits (8 bytes). You will then treat these 8 bytes as if they were a prefix of original input.
    2. Pad your input with additional bytes so that its length becomes a multiple of 8 bytes. You can choose whatever you want as padding, for example the zero byte ("00000000").
    3. The encoding of your original input will be the concatenation of all the encoded bytes in its length and padded input, as just described.
    
    For example, the encoding of "Hello world!" is computed as follows. This string has 12 characters, so the number 12 in 64-bit binary is
    
    $\begin{align*}
    00000000 & 00000000 & 00000000 & 00000000 \\
    00000000 & 00000000 & 00000000 & 0001100
    \end{align*}$
    
    (the spaces are added for clarity). The encoding of this bit string is the 16-byte character string
    
    $\text{AA AA AA AA}$
    
    $\text{AA AA AA AM}$
    
    since each zero byte "00000000" is encoded as "AA" and the last byte containing the number 12, i.e., "0001100", is mapped to "AM".

    because it is only 12-bytes long, we need to pad "Hello world!" with four more bytes. For display purpose, we write "Ø" for the zero byte, obtaining the 16-byte padded string "Hello world!ØØØØ". If we line up each input letter and padding with its encoding, we obtain the following 32 character encoding:

    $\text{H e l l o w o r l d ! Ø Ø Ø Ø}$
    
    $\text{EI GF GM GM GP CA HH GP HC GM GE CB AA AA AA AA}$
    
    where the encoding of the ASCII for "H" is indeed "EI" and so on up to "!", which is mapped to "CB".

    Therefore, the overall encoding of the 12-byte string is "Hello world!" is the 48 uppercase character string (formatting spaces removed):

    "AAAAAAAAAAAAAAAAAAAAMEIGFGMGMGPAHHPHPCGMCBAAAAAAAAA"

  - **Decoding:** A string containing a multiple of 16 uppercase characters in the range A-P is decoded by performing the inverse operations. Specifically,
    1. The first 16 characters are decoded to the length $l$ of the string to be output. In our example, that’s "AAAAAAAAAAAAAAAAAM", from which the number 12 is recovered.
    2. The remaining characters are decoded to the padded output. In our example, that’s "EIGFGMGMGPAHHPHPCGMCBAAAAAAAA", from which we recover the byte string "Hello world!ØØØØ".
    3. The first $l$ bytes of this string are kept, discarding the rest. In our example, this yields Hello world!".

**Implementation Guidelines**

To complete part (1) of this assignment, you will need to give a working implementation of the Enigma-349. You are free to do so in whatever programming language you want.
Your overall program should take 4 input arguments:

1. the 16-character key corresponding to the initial substitution,
2. the 3-character key for the middle substitution,
3. the 16-character key corresponding to the initialization vector, and
4. the binary message you want to encrypt; you may want this input to reside in a file, in which case you would be passing a file name,

and return the encrypted message. Just like for the input, it will be convenient to have your program write the output to a file.

Although not strictly necessary, you may want to implement the decryption and decoding algorithms for this cipher: that will be the only way for you to test that it works!!! Said this, part (3) of the assignment (below) points to a series of files that demonstrate the intermediate outputs of this exercise on an example.

To help us give you credit for partial solutions, we will also ask you to provide the code of each of parts (1.1), (1.2), (1.3) and (1.4) as separate programs that just implement the functionality of that part. For example, the program that implements part (1.2) will take as input the 3-character key for the initial substitution and (a file containing) an uppercase input text and produce (a file containing) its initial substitution. Your overall implementation (above) is just the chaining of these four pieces of code.

So, altogether, you will need to submit 5 programs:

1. a program that implements part (1.1),
2. a program that implements part (1.2),
3. a program that implements part (1.3),
4. a program that implements part (1.4), and
5. a program that implements the Enigma-349 by implementing all parts at once.

See part (3) for details.

2. Security in the Press (50 points)

Security, or more specifically insecurity, is a very popular subject in the press. Sometimes journalists get it right, and other times they don't. In this part of the assignment, you will be asked to read a brief article, explain it in your own words, and criticize it.


Your task will be to write an essay that describes the contents of this article to somebody who is just starting to learn about computer security. Think about one of your classmates. Examples of questions that this person may want explained are:

- What is PGP? What is it used for?
- How does it work? What kind of cryptographic methods does it use?
- How does the attack work? How serious is it? Is it really an attack?
- What are those 'scrambled forms' the article refers to? What do 'sniffers' have to do with all this?

These are just examples of questions you may want to answer. The more inquisitive the better! While you have the whole Internet, and more, at your disposal to do the necessary research to answers such questions, the answers should be your own. Categorically.

In case of doubt, upload your paper on turnitin.com (the class ID is "5555549" and the enrollment password is "enigma" — see also this manual).
You will notice that there is vast disagreement among the people interviewed in this article about the severity of the attack. On the basis of your research, evaluate the pros and cons of each position and justify your own position.

As you write this essay, aim for clarity, thoughtfulness and coherence. There is no minimum or maximum number of words. Notice however that it is worth as many points as part 1 (and part 3). This means that you are expected to spend enough time on it to produce an essay of very high quality.

3. Submitting the Assignment (50 points)

You are asked to submit your assignment by sending an encrypted and signed email to iliano@cmu.edu. To do this, you will rely on PGP, the very same cryptographic suite that the article you criticized in part (2) of this assignment was about. This will involve a number of steps.

3.1 Installing PGP on your own Computer (20 points)

Yes! You need to install stuff! There are a number of free distributions of PGP on the Internet and they run on all kinds of platforms and OS's. You can use some from the command line, so that you may produce your signed and encrypted message and then email it. Others work directly with an email client such as GMail or Thunderbird (none works with WebMail however).

Once you have installed whatever version of PGP on your system, take a screenshot and include it as part of your submission.

3.2 Configuring PGP (10 points)

Once you have installed PGP on your computer, you will need to create your own set of keys. In some distributions, you need to run a special utility. In others, key generation happens automatically during the installation process.

Once you have generated your keys, upload the public key to pgp.mit.edu. This is how we will know you have done it.

3.3 Submitting your Assignment (20 points)

You have implemented an encryption algorithm in part (1), haven't you? and also a little utility to encrypt arbitrary binary files, right? Well, it's time to use them! The submission process is as follows:

1. Use the cipher you implemented in part (1) to encrypt the source code of your overall program (yes, you are encrypting it with itself!). The key for the encryption is as follows:
   - The initial permutation will shift every letter A-Z by as many positions as there are letters in your email address: therefore, iliano@cmu.edu would shift the alphabet by 12, so that "A" becomes "M".
   - The 3-letter initialization code for the middle substitution performed by the Enigma-349 will be the first 3 letters of your email address in uppercase: if you were iliano@cmu.edu, this would be "ILI".
   - The initialization vector for part 3 will be your email address stripped of non-letter symbols ("@", ".", "-", etc) and concatenated with itself as many times as it takes to get a string of at least 16 characters, and then you would take just the first 16 of them: iliano@cmu.edu would produce "ILIANOCMUEDUILIA".

Put the encrypted program in a file called program.349.

8
For testing purposes, here are 4 files and the intermediate results of each of the encryption steps in part (1) with the above keys:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>input ⇔ (1.4) ⇔ tmp-1 ⇔ (1.1) ⇔ tmp-2 ⇔ (1.2) ⇔ tmp-3 ⇔ (1.3) ⇔ output</td>
</tr>
<tr>
<td>b.</td>
<td>input ⇔ (1.4) ⇔ tmp-1 ⇔ (1.1) ⇔ tmp-2 ⇔ (1.2) ⇔ tmp-3 ⇔ (1.3) ⇔ output</td>
</tr>
<tr>
<td>c.</td>
<td>input ⇔ (1.4) ⇔ tmp-1 ⇔ (1.1) ⇔ tmp-2 ⇔ (1.2) ⇔ tmp-3 ⇔ (1.3) ⇔ output</td>
</tr>
<tr>
<td>d.</td>
<td>input ⇔ (1.4) ⇔ tmp-1 ⇔ (1.1) ⇔ tmp-2 ⇔ (1.2) ⇔ tmp-3 ⇔ (1.3) ⇔ output</td>
</tr>
</tbody>
</table>

2. Use again the cipher you implemented in part (1), with the same keys as for the program, to encrypt a PDF of your essay from part (2). Put the encrypted essay in a file called `essay.349`.

3. Sign and encrypt both files as well as the individual programs that implement parts (1.1) through (1.4), the screenshot of your PGP installation, and a readme file with your new PGP key and email them to `iliano@cmu.edu`. Therefore, your secure email will contain:
   - **README.txt**: instruction file containing the following
     - your name,
     - your implementation language and environment (compiler/interpreter, version, any non-standard libraries, etc.)
     - the command(s) you use to compile your program,
     - the keys you used to generate the *.349 files,
     - the exact commands you ran to generate the *.349 files,
     - the PGP installation you used, and
     - any other message you would like to add.
   - **pgp.png**: a screenshot of your PGP installation,
   - **program.349**: the overall program from part (1) encrypted with itself,
   - **essay.349**: a PDF of the essay from part (2) encrypted with the overall program from part (1),
   - **pg_1-1.<ext>**: unencrypted program for part (1.1),
   - **pg_1-2.<ext>**: unencrypted program for part (1.2),
   - **pg_1-3.<ext>**: unencrypted program for part (1.3),
   - **pg_1-4.<ext>**: unencrypted program for part (1.4).

(The last four are so that we can give you partial credit in case we are not able to decrypt the first two).

As you take each step, make sure that you have all the information you need to proceed ...