AEP 1: Encrypting messages with binary

Overview

In this AEP, you will learn about a way of **encrypting** a message — encoding it so that only the sender and the intended recipient can read it — that uses base 2 representations of integers and a special operation on binary integers known as **XOR**.

Learning Targets associated with this AEP:

- **A.1:** I can represent an integer in base 2, 8, 10, and 16.

Remember, AEPs do not have fixed deadlines; simply work on this item until you are ready to submit it. But remember the **Two Items Per Week Rule.**

Technology Background

Some of the tasks in this AEP can be done by programming in Python. If you opt to do the programming tasks, you'll need to have completed the **Python/Jupyter Crash Course** that was assigned during the first week of classes. If you do not opt to do those tasks, there's no special tech background needed.

AEP Description and Tasks

What this AEP is about

We keep secrets from each other all the time. Those secrets aren't necessarily incriminating or bad; for example, our credit card numbers, social security numbers, and browsing histories are all information that we'd rather not have leaked into the public eye. To keep that information secret in a digital form, we use **encryption**. Encryption refers to any process that transforms information into a format that is readable only to the owner of that information and the people or machines who the owner decides should see it. When encrypted information is sent, the recipient turns it back into a readable format by **decrypting** it. Both encryption and decryption use a **key**, which is a piece of information (a number, a code phrase, etc.) that, like a physical key, is used to lock and unlock the information being sent.

This AEP is about a method of encryption and decryption that at one time was very common in the early days of digital communications, and which (unfortunately, as you'll see) is still used in some places today. It involves transforming human-readable messages into binary integers, then using a binary string as a key to encrypt it, via a special binary operation called **XOR**. In this AEP you'll learn how to encrypt and decrypt using this **XOR cipher**, and play the role of an adversary by breaking an intercepted message encrypted using the XOR cipher.

Setup (do this first)

Here's a description of how the XOR cipher works.

First, the cipher involves the **XOR** operation. This is an arithmetic operation (denoted `xor`) on single bits (0's and 1's) that works as follows:

- `0 xor 0 = 0`
- `1 xor 0 = 1`
- `0 xor 1 = 1`
- `1 xor 1 = 0`

So this is like regular binary addition except for the last line — **we don't ever carry a 1**. Ordinary addition would say `1 + 1 = 10` but here, `1 xor 1` is just `0`. So *`xor`-ing two bits together always leads to just a single bit*.

We can therefore `xor` *strings* of bits together, just by `xor`-ing the individual bits in corresponding positions. For example:

- `10 xor 11 = 01` (`xor` the two bits in the left position and the two in the right position)
- `1011 xor 1111 = 0100`
- `10110001 xor 01010101 = 11100100`

Now we can describe how the cipher works:

1. Start with a message, in regular text, to send. This is called the **plaintext**. For this AEP, assume that messages only involve capital English letters — no lower-case letters, punctuation marks, numbers or other symbols.
2. Convert each letter in the message into its **ASCII decimal** format. ASCII is a standard means of encoding characters into integers. You can find tables of ASCII values on the web, for example here; in Python, the function `ord` returns the ASCII value of a character. For example, `ord('B')` returns `66`. The letter `A` has an ASCII value of 65; `B` is 66; and so on through `Z` which is 90. (The Python function `chr` reverses this: `chr(66) = 'B'`.)
3. Then convert each of those decimal integers into 8-bit binary.
4. Now choose a random 8-bit binary integer k to use as your key. Share this with the person or machine you want to receive your message.
5. For each 8-bit binary string b that represents a character in the message: Compute `b xor k`. This produces a new 8-bit binary string. Do this for each "letter" in the message. The result is the encrypted version of the message. Send it off to your recipient.
6. When the recipient gets the message, they have the key you sent them (it's the same key as the one you used to encrypt). The recipient breaks the message into 8-bit chunks and for each chunk c, they compute `c xor k` where `k` is the key.
7. This process produces the same string of 8-bit binary integers that you started with. Convert them back to decimal, then from ASCII back to letters, and now the recipient can read the message.

Example: Alice is sending Bob the message `RUN` using this cipher.

- Converting the plaintext to ASCII gives: `82 85 78`.
- Converting those to binary gives: `01010010 01010101 01001110`.
- Alice and Bob agree to use the 8-bit key `11000011`. Alice `xor`s the key with each of the strings above:
  - `01010010 xor 11000011 = 10010001`
  - `01010101 xor 11000011 = 10010110`
  - `01001110 xor 11000011 = 10001101`
- Alice sends Bob the message: `10010001 10010110 10001101`.
- Bob receives this and `xor`s the message with the key:
  - `10010001 xor 11000011 = 01010010`
  - `10010110 xor 11000011 = 01010101`
  - `10001101 xor 11000011 = 01001110`
- This is the same string of binary that Alice had. Bob converts each 8-bit string to decimal – `82 85 78` – and then back into letters using an ASCII table or function, to get `RUN`.

Note, some implementations of this cipher include an extra step: Before sending the encrypted message, convert the 8-bit strings in the ciphertext to ASCII and send those. However this sometimes leads to characters that can't be visualized.

Tasks for this AEP

1. Make up a short (8 characters or less) message in English and a key, and encrypt it using the XOR cipher. Show each step of your process.
2. You've received the following message that was encrypted using the XOR cipher and a key of `10110111`: `11111011 11110110 11111100 11110010 11100101 11100100`. Decrypt the message.
3. In the decryption step of the XOR cipher, Bob `xor`s the encrypted message with the same key as Alice, and the plaintext message magically appears. Explain why this decryption step will always work. That is: Explain why taking a plaintext message, `xor`-ing it with the key, then `xor`-ing *that* with the key will always return the original message. (Hint: What happens when you `xor` a bit string with itself?)
4. For the fourth task, you have a choice of two options.

Option 1: For coders. Write two Python functions: One called `xor_encrypt` that implements the XOR cipher encryption step, and another called `xor_decrypt` that implements the XOR cipher decryption step. The `xor_encrypt` function should take in two strings of characters as input: the first string is the message to encrypt, and the second string is the key (an 8-bit binary integer, formatted as a string); and it should produce the binary string produced by the XOR cipher, formatted also as a string. The `xor_decrypt` function should also accept two strings as input: the binary form of an encrypted message, and the key, and the output should be a string of characters that gives the original message. Here's an example of what this would look like:

```
xor_encrypt('RUN', '11000011')
> '10010001 10010110 10001101'

xor_decrypt('10010001 10010110 10001101', '11000011')
>'RUN'
```

(The `>` is there to show you where the output would go; it's not part of the actual output itself.) Notice, especially that **everything here is a string** including the binary integers.

If you choose this option, you will need to not only provide code for your work but also explain (either in comments within the code or as a separate part of your writeup) why your code works and how you went about coming up with it.

Option 2: For hackers. You get to play the role of an evil eavesdropper, listening in on someone's messages, and breaking the code to find out what they're saying. (If you identify as female, call yourself "Eve". If you identify as male, call yourself "Everett".) You are listening in on a conversation that is encrypted using the XOR cipher. Since you're eavesdropping, you don't have the key. You just have the message:

```
10011110 10000010 10000011 10011001 11101010
10001001 10000011 10011010 10000010 10001111
10011000 11101010 10000011 10011001 11101010
10001111 10001011 10011001 10010011 11101010
10011110 10000101 11101010 10001000 10011000
10001111 10001011 10000001
```

However, you suspect that the binary string `10001111` in the ciphertext was originally the letter `E` in the plaintext, since `E` is the most commonly occurring letter in the alphabet and this string `10001111` happens three times, which is more common than any of the other strings. Using this knowledge, find the key to the cipher, and decrypt the message; and then fully explain what you did.

Assignment Expectations and Grading Criteria

AEPs are graded using the "EMRN" rubric found in the syllabus. **A grade of E or M requires all of the following to be met:**

- All work needs to be shown *and* your thought processes clearly expressed in all of the tasks of the assignment. The results also need to be correct.
- All the information needed for an "outsider" to understand your work needs to be self-contained within the work. For example, you need to clearly state the message and the key in Task 1. **The reader should not have to do any work to fill in gaps.**
- Task 3 needs to be a general explanation that explains why the decryption step ALWAYS works. Do not just give additional examples illustrating that decryption works.
- If you choose the programming option for Task 4:
    - Your Python functions *must* use the names `xor_encrypt` and `xor_decrypt`. Don't pick different names.

    - The two functions must operate precisely as shown above – accepting two string inputs and producing a string output.
    - **Your work must be thoroughly debugged to remove syntax errors before submitting them. If one or both of your functions throws a syntax error when evaluated in Python, you will receive a grade of N**.
    - **You may not use outside libraries or object-oriented programming techniques.** Both functions should be written without loading any additional Python libraries and without setting up classes and objects. If you are unsure about this, let me know.
    - Your functions will be tested against some test cases that I have prepared separately. Your functions will be expected to produce correct output on most, if not all of the test inputs.
- If you choose the "hacker" option for Task 4: You will need to state clearly what the key is and what the original message was, as well as fully and clearly explain your thought processes so that an "outsider" could replicate your steps on a new intercept. The explanation needs also to use the information provided — if you can break the code some other way, then you can provide that as separate information, but your main solution should use the fact that `E` encrypted to `10001111`.

Please note, it is the case with all AEP's that **your grade is primarily based on your explanations and writing, and only secondarily on the precision and correctness of your computations.** Correct computations with insufficient explanation will need to be revised and may receive an "N" grade.

A grade of "E" is given if all of the above expectations are met, and the work is of superior quality in terms of the clarity of explanations and work, appearance of the writeup, and precision of the mathematics.

Submitting your work

AEP submissions must be typewritten and saved as either a PDF or MS Word file. No part of your submission may involve handwriting; work that is submitted that contains handwriting will be graded N and returned without feedback. This includes electronic handwritten docments, for example using a stylus and a note-taking app. To type up your work, you can use MS Word or Google Docs (both of which have equation editors for mathematical notation) or any other computer-based math typesetting tool. Just make sure you save your work as a Word document or PDF (no `.odt`, `.rtf`, or other file extensions are allowed).

When you have your work typed up, double-check it for neatness, correctness, and clarity. Then, go to Blackboard, then **Assignments**, then **AEP**, then **AEP ???**. Clicking on the text "AEP ???" will take you to a place on Blackboard where you can upload your work. All grading and revisions of labs are done entirely on Blackboard. **Do not email your work to the professor** – only Blackboard submissions are accepted.

Getting Help

Please note that according to the syllabus, for AEP's **"no interactions at all with another person or with unauthorized sources on the internet is allowed."** Violations of this rule include *any* consultation with

other students or former students, including Math Center tutors; using work from another student or former student; submitting the problem set to an online help site such as Chegg or Coursehero; or asking for help in an online forum. All such violations will be treated as academic dishonesty and will result in a grade of "N" and being banned from revising the work.

You **may** ask me (Talbert) for help on this assignment in the form of **specific mathematical or technical questions**. If I cannot answer a question because it would give too much away, I'll tell you so. **However please note: I will not "look over your work" before you submit it to give you feedback on the overall submission**; the expectations are clearly laid out above, so just follow those directions and submit your best work, and you'll be allowed to revise it if needed.

You can ask technology related questions to anyone at any time. For example if you need help with Desmos, or with figuring out how to type up your work, there are no restrictions on that. I recommend the `#tech` channel on Campuswire so that you'll reach a large audience.