AEP 2: Public-Key Encryption

Overview

In this AEP, you will apply the concepts of modular arithmetic to learn about the powerful concept of **public key encryption** — a method of encrypting and decrypting messages without sharing private information beforehand. The successful implementation of public-key encryption in the 1980's is widely considered one of the greatest scientific innovations of the last few centuries, and we use it every day without realizing it. And it all boils down to arithmetic!

Learning Targets associated with this AEP:

- A.3: I can compute `a % b` given integers `a` and `b` and perform arithmetic mod n.

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

This AEP focuses on another method for **encrypting** and **decrypting** messages that has a fundamental difference from the other methods we've seen in class and in AEP 1. Before going any farther, please read the "What this AEP is about" section for AEP 1 for an overview of basic terminology and concepts from cryptography such as *encryption*, *decryption*, and *key*.

In all the other cryptosystems we have seen in the course (shift cipher, multiplicative cipher, the XOR cipher in AEP 1), *Alice* is trying to send a message to *Bob*, but she doesn't want an evil eavesdropper *Eve* to be able to read it, even if the message is intercepted. So Alice encrypts the message and sends it to Bob. Simple, right? But there's one problem: In all of those earlier systems, **Alice and Bob have to share the key**. It's a lot like if I wanted to grant you access to my office — I'd need to somehow get you my physical key. But…

- …What if sharing the key for encryption and decryption was problematic? For example, what I believed that "Eve" might be eavesdropping on the channel I am using to send you the key? I could encrypt the key, but that doesn't solve the problem, because how will I get you the *second* key that encrypts the first one, in a way that convinces me Eve hasn't stolen it or tampered with it? The Diffie-Hellman protocol we studied in Module 2 addresses this problem, but it doesn't solve the next one:

- …What if I wanted to send you a message but also wanted to remain anonymous — like a whistleblower who wants to send an encrypted message to a reporter? Traditional cryptosystems fail on this front because they require that I reveal my identity to you in order to send the message — if not by name, then by the fact that I must have information in "the system" (e.g. I have to have my own key) in order to send a message.

What solved both of these problems was a mathematical innovation discovered in the late 1970s and early 1980s called **public key encryption**. It actually began with the Diffie-Hellman protocol. Recall that with Diffie-Hellman, Alice and Bob were able to mutually agree upon a secret key to use without sharing any private information. In fact, what makes that protocol work is that there is **public** information that both Alice and Bob have, as well as **private** information that they have, and through a mathematical combination of the public information and private information, they are able to construct a key without ever sending the key over an open channel.

Public key encryption works this way as well. In a public key system, each user creates two keys: one that is kept private, and another that is made 100% public. If Bob is a user, and Alice wants to send him a message, Alice looks up his public key information and uses it along with an algorithm to encrypt the message. Then, when Bob receives the message, he uses his private key to decrypt it. Public key system are made in such a way that the mathematical combination of the public and private keys will decrypt the message. And, Alice does not have to generate any keys of her own. (Think of a telephone system — you shouldn't need to have your own phone number in order to call someone up.)

Setup (do this first)

This AEP will focus on a simple (but vulnerable) public-key system that use what we learned in class about modular arithmetic. The first of these works as follows.

- Each user chooses four positive integers: a, b, A, and B. The capitalization is important.
- Then the user does four computations:
  - $M = ab - 1$
  - $e = AM + a$
  - $d = BM + b$
  - $n = \frac{ed - 1}{M}$
- The **public key** for the user is the pair of numbers (n, e). The user puts this key next to their name in a public place (a server, Google Doc, skywriting, etc.)
- The **private key** for the user is the number d. The user keeps this private under all circumstances.

If Alice wants to send Bob a message, she does the following:

- Alice looks up Bob's public key (e, n).
- Alice takes her plaintext message — which for this AEP we will assume is just a string of capital letters without punctuation or other symbols — removes all spaces, and converts each letter to numbers 00 through 25 according to its position in the alphabet. This produces a long string of decimal integers; Alice then breaks that string up into equal-sized blocks, each of which is an integer less than n (from Bob's public key), and adds zeroes to the end of the final block if needed to make it the same length as the others.

Example of the previous step: Alice wants to send the message `AM ON MY WAY`. Converting this to integers 00-25 gives `00 12 14 13 12 24 22 00 24` and stripping out the spaces gives `001214131224220024`. Bob's value of n is 237733. So Alice breaks her message up into blocks less than 237733 — for example, into blocks four digits long: `0012 1413 1224 2200 24`. To get that last block to be exactly four digits, she pads it with a couple of zeroes to get

`0012 1413 1224 2200 2400`. (She could also have chosen blocks five digits long, or shorter.)

The encryption step has one more step:

- For each block P, Alice computes: $C = (e \times P)\%n$. Once each block has been encrypted, this is the cipher text that is sent to Bob.

When Bob receives the ciphertext, which is a string of blocks of integers, for each encrypted block C, he computes: $(C \times d)\%n$. This produces the decrypted blocks that Alice sent. Bob can then separate the blocks into two-digit units and convert back to letters, and read the message.

Notice that Bob's private key was never communicated; and Alice needed no key of her own to send the message.

Tasks for this AEP

1. Generate a set of keys for yourself using the method above method. Keep the private key d completely private, but email or DM your public key pair (e, n) to Prof. Talbert when you're done. I will put the class' public keys for all to see in a Google spreadsheet found here: https://docs.google.com/spreadsheets/d/1CO4W3wmT8fE_WrCHlJ9SIQv8HWNQHqJ8S8bL8Lv0_gw/edit?usp=sharing Please note, to prevent tampering, this spreadsheet is comment-only. I have put my own public key information there already. Show all your math work in your writeup.
2. Send me an encrypted message. This message should be sent to me by email and does not go in your writeup. However, please do put your *original decrypted* message in your writeup along with the math work you did to get create the message. (You should never include the unencrypted message in real life!)
3. Once you have posted your public key, I will send *you* an encrypted message by email. It will be something where it's obvious if you've decrypted it correctly. When you get it, put all your math work used to decrypt the message in your writeup.
4. For the fourth task, you have a choice of two options.

Option 1: For coders. Write three Python functions:

- `pk_keys` which takes in four positive integers from the user and generates the public and private keys for the system. The function should use `return` and not `print` for the output, and it should return the keys in a tuple.
- `pk_encrypt` which takes three arguments: (1) The plaintext message to encrypt, (2) the recipient's public key e, and (3) the recipient's public key n; and the output is a string representing the encrypted message. To simplify things, the input string should be a string containing the plaintext message after converting to numbers and breaking it into blocks, and assume that the user is smart enough to check to make sure the blocks are all smaller than n before entering.
- `pk_decrypt` which takes in two arguments: (1) An encrypted message, represented as a string of blocks of integers, and (2) your private key d; and the output is a string representing the decrypted message. Again to keep things simple, the decrypted message can just be blocks of integers — no need to convert these back to letters unless you simply want to.

Some example (fake) outputs to show you what this should look like:

```
print(pk_keys(9,11,5,8))
> '(98, 499, 795, 4048)'   # 4048.0 would be OK

pk_encrypt('538', 499, 4048)
> '1294'

# The messages are strings, but the keys are integers.

pk_decrypt('1294', 795)
> '538'
```

Option 2: For hackers. Choose someone on the public key spreadsheet, and **using only their public key information, figure out their private key**. How you do this is up to you. You can use math, or a computer program you write, or some combination. You may not torture the person to get their information; you may also not engage in social engineering (i.e. some form of asking the person what their private key is and they tell you). Also, in order to deter cheating, *each person choosing this option will need to hack a different person*. So if you choose this option, *you will need to inform me (Talbert) first and I'll assign you a person to hack*. Users whose accounts have been hacked will have a red **HACKED** indicator on the spreadsheet.

If you can do this, you will demonstrate that while this cryptosystem is a nice example of public key encryption, it's highly vulnerable to attack by an adversary (that's you). Any situation where the private information can be recovered relatively straightforwardly from the public information, is bad.

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. **The reader should not have to do any work to fill in gaps.**
- If you choose the programming option for Task 4:
  - Your Python functions *must* use the names given in the specification. Don't pick different names.
  - The three functions must operate precisely as shown above.
  - **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 private 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.

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 2**. Clicking on the text "AEP 2" 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.