Activity 2.3.2

Security by Encryption

Introduction

Data and processing power should only be available to people who are authorized to access them. But how does a computer know who someone is? A user proves identity with a password, but an eavesdropper can capture a password en route.

Ultimately, cybersecurity depends on encryption, and encryption depends on mathematics – on one-way mathematical functions. One-way functions are functions that are faster to do than to undo – so much faster that it is impractical to undo them. Cybersecurity comes down to speed. Systems are secure because it would take a malicious hacker "forever" to break in.

Why can't we depend on secret passwords? Why is time the anchor of cybersecurity?

Materials

- Computer with Canopy distribution of *Python*®

Resources

2.3.2 sourceFiles.zip

Procedure

Part I: Compare inverse functions and identify prime numbers

1. Form pairs as directed by your teacher. Meet or greet each other to practice professional skills. Review the class expecations for engineering notebooks or electronic documents for written work.

2. Encrypting and decrypting a message are inverses of each other. If you do one and then the other, you get back to where you started. Another example of a pair of actions that are inverses of each other is multiplying by ten and dividing by ten. Tying shoes and untying shoes is another example of a pair of functions that are inverses of each other.

   Create a group of four by combining partnerships in your class. Brainstorm some pairs of actions that are inverses of each other.

3. Encryption depends on one-way functions. One-way functions are quick to do but very slow to undo.

   Tying shoes takes longer than untying shoes. Suppose it takes 8 seconds to tie a shoe and 1 second to untie a shoe. That's an 8:1 ratio.

   time to do a task : time to do the inverse task = 8:1

   Of the inverses you thought of in the previous step, which has the largest ratio?

4. Recall from Lesson 2.1 that symmetric key encryption is when the sender and recipient agree on a secret key beforehand. Symmetric key encryption doesn't work for exchanging information on the Internet since the sender and recipient never meet beforehand.

   Also recall that public key encryption, also called paired key encryption, *does* allow people to exchange encrypted messages without exchanging a secret key beforehand. Refer to Activity 2.1.5. To review, explain how Jane Doe sends Company Q her credit card without agreeing on a secret shared key.

5. Return to working with just your partner. Most paired key encryption today uses the RSA algorithm. To create a pair of RSA keys, you start by picking two prime numbers: ***p*** and ***q***. Challenge yourself: what are the largest two prime numbers you and your partner can identify in the next 60 seconds?

   Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

6. Check whether your numbers were really prime using the following steps.
   - Obtain a copy of `paired_keys.py`. Open Canopy. Open a new code editor window. Select **File** > **Open** and open `paired_keys.py`.

   - Execute `paired_keys.py` using the green arrow. This will define thirteen functions, including `prime_factors()`.

     On lines 41 to 67 in the code editor, read the code that defines `prime_factors()`. You can use a function without worrying about the details of how it does what it does. Discuss with your partner whether the docstring correctly abstracts what this algorithm does. Write a summary of what the function does.

     ```
     def prime_factors(unfactored):
     ```

   - At the IPython prompt, test the numbers you thought were primes. In the example shown here, a student has discovered that 111 is not prime.

```
In[]:  prime_factors(111)
Out[]: [3,37]
```

- Challenge yourself again: what are the two largest prime numbers you and your partner can identify in the next 60 seconds?

Part II: Pick prime numbers for RSA encryption

7. The encryption used for HTTPS and other secure protocols uses the RSA algorithm. RSA creates a pair of keys by starting with two prime numbers. We won't be doing encryption in this activity. Our aim is to understand the fact that encryption is done with algorithms, and that the encryption only works for secrecy and authentication because of the time algorithms take to execute.

   Use two prime numbers to create a pair of RSA keys. To do this, pass the prime numbers to the `make_keys_from_primes()` function that was defined when you executed `paired_keys.py`. Do this in the IPython window. In the example shown here, the prime numbers 3329 and 7411 are used to create a pair of keys. Each key has two numbers in it. Either key can be the public key, and the other will be the private key.

   ```
   In []:  make_keys_from_primes(3329,7411)
   Out[]: [(24671219, 55667), (24671219, 443)]
   ```

   Record information about your keys:

   - prime numbers are ________ and ________
   - keys are (________ , ________) and (________ , ________)

8. In the example above, the first number in each key, 24671219, is called the modulus. It is the product of 3329*7411. Both keys include the product of the prime numbers, so this product is known by anyone with the public key.

   Record information about your keys:

   - The modulus is ________ and it is the product of ________*________
   - If you use RSA encryption, who has the modulus?

9. ***This step optional, as directed by teacher.*** This step shows you the arithmetic for making RSA keys. The other numbers, 55667 and 443, in the RSA keys shown in Step 7 are factors of

   $$3328^{*}7410 + 1.$$

   This is ***(p-1)(q-1)+1*** where ***p*** and ***q*** were the prime numbers. So the other numbers 55667 and 443 in the keys in Step 7 work because

   $$55667^{*}443 = 3328^{*}7410 + 1.$$

   The RSA algorithm will also work if the keys both contain the product ***pq*** (that's 3329*7411 = 24671219 in the keys in Step 7) and each contain another number, as long as the other numbers are factors of:

$$2 * 3328^* 7410 + 1 \quad \text{or}$$
$$3 * 3328^* 7410 + 1 \quad \text{or}$$
$$4 * 3328^* 7410 + 1 \quad \text{or}$$
$$\text{any multiple of } 3328^* 7410 + 1.$$

You can use the IPython session as a calculator to determine what multiple was used to create your keys. Multiply the keys' other numbers (like 55667*443) and estimate what multiple of ***(p-1)(q-1)*** is needed. Using numbers, and not the letters ***p*** and ***q***, record information about your keys:

- The other numbers in the keys are ______ and ______ and they are factors of ______ * __________ * __________ +1

10. The public key tells everyone what the product ***pq*** is of two primes ***p*** and ***q***. If someone can factor that product ***pq*** into ***p*** and ***q***, they can figure out your private key. What do you suppose prevents people from factoring the product that is published in the public key? Confirm your answer with your teacher or as a class before moving on to Part III.

11. In Activity 2.1.5, each student sent a message using the recipient's public key. The senders put their messages where anyone could get them, but only the recipients could read the messages. Suppose Chris managed to factor the number ***pq*** in Allie's public key. What would that enable Chris to do?

Part III: Analyze and measure the efficiency of algorithms

12. If two algorithms accomplish the same task, but one is faster, we say the faster one is more time efficient. Below are two algorithms. Both algorithms return True if the number is prime. We will find out which algorithm is more efficient and why. Consider which lines of code are executed in each of the following cases.
    - `is_prime_versionA(number=10)`
    - `is_prime_versionB(number=10)`
    - `is_prime_versionA(number=11)`
    - `is_prime_versionB(number=11)`
    - With your partner, list as many reasons as you can why one algorithm is more efficient than the other. For each reason, specify the lines of code in each algorithm contributing to that particular difference in efficiency.

Algorithm A	Algorithm B

<pre>def is_prime_versionA(number):
  """ Return True if number is prime.
  Returns False otherwise.
  """
  # Initialize a one-way flag</pre>
	<pre>def is_prime_versionB(number):
	  """ Returns True if number is prime.
	  Returns False otherwise.
	  """
	  # Check is number is even
	  if number%2 == 0:
	    return False</pre>

<pre>   prime = True
 	
    # Check all possible divisors
    # from 2 to number-1
    for divisor in range(2, number-1):
      if number % divisor == 0:
        # No remainder; it's a factor
        prime = False
 	
  return prime</pre>
	<pre> # If there are divisor pairs,
	    # one divisor <= number's square root
	    max_divisor = int(number**0.5)
	    max_divisor += 1 # So range() includes it
	    # Check all possible divisors
	    # from the odd numbers
	    for divisor in range(3, max_divisor, 2):
	      if number % divisor == 0:
	        # No remainder, so it's a factor
	        return False
	
	  return True</pre>

13. You analyzed the efficiency of these algorithms theoretically. To theoretically analyze an algorithm, you think about how many instructions the processor will have to execute. You can also analyze the efficiency of algorithms **empirically**. To empirically analyze an algorithm, you actually measure the time with a clock. We will do that in a moment, but first think about contraints on a program.

    Speed can trade off against other important criteria. Before diving deeper into the speed required for encryption, consider the other important factors. Match the criteria on the left with the circumstance on the right in which that criterion might be the top priority.

Criteria	Circumstance
Program must execute as **fast** as possible. Program must be as **readable** as possible. **Program** must fit in **memory.** **Data** must be handled without overflowing the **memory.**	Program executes on a **coffee pot's** electronics. Program **simulates** positions of molecules during a chemical reaction. Program calculates position of a softball to be **caught by a robot.** Program is to be **reused** by other programmers.

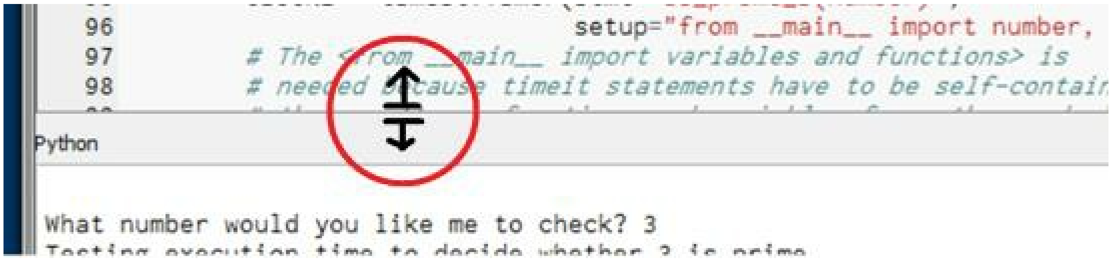
14. The `timeit` library of *Python* allows us to time the execution of code on the processor. The operating system and other programs will be using the processor too, so it's not a perfect measurement. Like any measurement in science, there will be noise in the data. Noise is variation in the data that is not caused by the changes you are trying to compare.

    Compare the two algorithms using the following steps.

    - From the Canopy editor, select **File** > **Open** and open `efficiency.py`.
    - Execute the program `efficiency.py`. When the program asks for a number to check, enter 50, as shown here.

```
What number would you like me to check? 50
0.00702 seconds for 10000 executions of is_prime_versionA(50)
```

- Adjust the Canopy interface so that you can see more of the IPython session. Do this by mousing over the dividing line between them until the mouse pointer appears as shown below. Then drag upward.



- Record the minimum time shown for 10,000 executions for each algorithm to determine if 30 is prime. Round the measurement if desired.

  Algorithm A: ______________ Algorithm B: ______________

- The times vary because the operating system and other programs (including the Canopy environment itself) vary in how much they are using the processor. Why was the minimum time the most relevant time to use for comparing the algorithms?
- Which algorithm was faster? Does this agree with your theoretical analysis in Step 12?
- Execute the `efficiency.py` program again, but provide 3 as the number when prompted. Record the minimum time shown for 10,000 executions for each algorithm to determine if 3 is prime. Round the measurement if desired.

  Algorithm A: ______________ Algorithm B: ______________

- Which algorithm was faster? Does this agree with your theoretical analysis in Step 12?
- Explain why the results in Step 14f and Step 14h are different. Refer to specific lines of code from Step 12.
- Use the following paragraph to generalize what you have observed by filling in the blanks and completing the sentence. Use your engineering notebook or an electronic document as directed by your teacher.

  Two algorithms that accomplish the same task can be compared using ______________ analysis or ______________ analysis. Two kinds of analysis should provide the same answer about which algorithm will be faster. In either kind of analysis, which algorithm is faster might depend on

  ____
  _________________________________.

Part IV: Measure the effect of input length on decision time

15. You've measured the time for each of two algorithms to determine if a number is prime. If the input to the algorithms is larger, do the algorithms use more time to solve the problem? Let's explore.

    - The program `efficiency_with_mins.py` is similar to the program you used in Step 14, but it will determine the minimum times for you. Open it in Canopy and execute it. Collect data for 3 and 30 again using this program and record the times in the table below. Are your results consistent with the results from Step 14d and Step 14g?
    - Complete the table. In the second column, record the number of characters provided as input to the program. For example, 3000 has 4 characters.

Input data x	# Input digits n	Time t (10000 executions)	
		Algorithm A	**Algorithm B**
3	1 (like Step 14g)		
30	2 (like Step 14d)		
300	3		
3000	4		
30000	5	(too slow)	
300000	6	(too slow)	
3000000		(too slow)	
30000000		(too slow)	

16. Examine what happens to the running time of Algorithm B when you increase the length of the input data by one character. Which of the following describes the pattern for algorithm A?
    - The running time stays the same for input with more characters.
    - The running time adds the same number of seconds for each character.
    - The running time adds more seconds for each character.
    - The running time multiplies by the same factor for each character.

17. Examine what happens to the running time of Algorithm B when you increase the length of the input data by one character. Which of the following describes the pattern for algorithm B?
    - The running time stays the same for input with more characters.
    - The running time adds the same number of seconds for each character.
    - The running time adds more seconds for each character.
    - The running time multiplies by the same factor for each character.

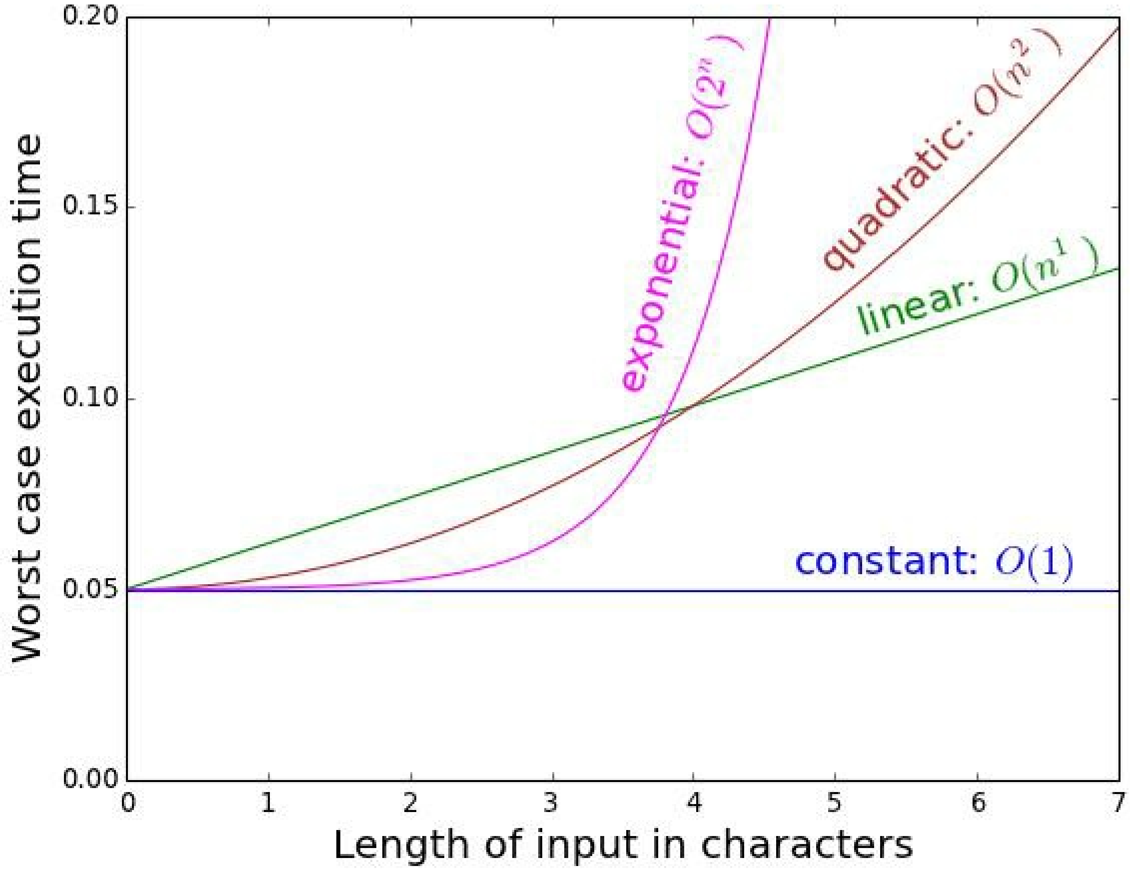
18. Refer to your downloadable resources for this material. Interactive content may not be available in the PDF edition of this course.

The time complexity of an algorithm describes how the worst-case running time of the algorithm increases with longer input. We'll ignore the “worst-case” part of that definition for now. Based on your data, find the time complexity of Algorithms A and B as follows.

- Graph your results with *Python*, Excel® software, or another tool. For *Python*, you can select **File** > **New** > ***Python* file** and paste the following code, placing your data in the lists on lines 9-10. Paste a screenshot of your graph in your work. The code is also available in `graph_data.py`.

```
import matplotlib.pyplot as plt

# Column 2 from data table
A_input_chars = [1, 2, 3, 4]
B_input_chars = [1, 2, 3, 4, 5, 6, 7, 8]

# Column 3 and 4 from data table
# Replace list elements with your times
A_time = [0.0014, 0.019, 0.17, 1.6]
B_time = [0.075, 0.047, 0.046, 0.052, 0.047, 0.039, 0.062, 0.057

fig, ax = plt.subplots(1,1)
# plot(x_list, y_list, "color and style")
ax.plot(A_input_chars, A_time, 'ro-', label='Algo. A') # red dot
ax.plot(B_input_chars, B_time, 'bo-', label='Algo. B') # blue do

# Label and show
ax.set_xlabel ("Length of input in characters")
ax.set_ylabel("Execution time")
ax.set_title("Execution time vs. input length")
ax.legend(loc='center left') # Show and place the legend
ax.margins(.1) # Extend the graph area beyond data by 10%
fig.show()
```

- Time complexity is described by big-O notation. The graph below shows how worst-case running time depends on input length for algorithms of four types of time complexity. Judging from the shape of the curves, which time complexity shown below most closely resembles your data for Algorithm A?
    - Constant: ***O(1)***
    - Linear: ***O(n)***
    - Quadratic: $\boldsymbol{O(n^2)}$
    - Exponential: $\boldsymbol{O(2^n)}$
- Which time complexity shown below most closely resembles your data for Algorithm B?



19. Repeat Steps 13-16, but use the following data as inputs. Record your results in the following steps.

Input data x	# Input digits n	Time t (10000 executions)	
		Algorithm A	**Algorithm B**
3	1		
37	2		
317	3		
3167	4		
33769	5	(too slow)	
321983	6	(too slow)	
3221983		(too slow)	
33149399		(too slow)	

- (Like Step 15) Execute `efficiency_with_mins.py` to measure the time for the computer to determine whether the numbers in the "Input Data" column from the table above are prime. Record the minimum times for each algorithm.
- (Like Step 16) According to what pattern does the running time of Algorithm A increase with the number of input characters?
- (Like Step 17) According to what pattern does the running time of Algorithm B increase with the number of input characters?
- (Like Step 18a) Graph your data and paste your graph here.
- (Like Step 18b) Comparing to the graph shown in Step 16b, which time complexity most closely resembles your new data for Algorithm A?
- (Like Step 18c) Which most closely resembles your new data for Algorithm B?

20. For Algorithm A, both Steps 15 and 19 correctly measured the algorithm's time complexity. Only Step 19 measured Algorithm B's time complexity, however, because Step 15 did not result in Algorithm B's **worst-case running time**. Examine Algorithms A and B again. Why did Algorithm A show its worst-case running time with both sets of input but Algorithm B only had its worst-case running time for the inputs in Step 19?

21. In this activity you measured the time complexity of two algorithms. You measured the complexity of the solutions to a problem. That problem was "Decide if a number is prime." Problems are also categorized by their time complexity, based on the fastest algorithm that can solve the problem. The problem of deciding whether a number is prime is known to be solvable in **polynomial time**. Polynomial time means that the worst-case running time grows with the length n of input data like $t=n^1$, or $t=n^2$, $t=n^{100}$, or maybe even a much higher polynomial power–but not as bad as the exponential $t=2^n$ which has the n in the exponent.

    So we know that the problem of deciding whether a number is prime isn't as complex as $O(2^n)$. Considering your answer to Step 19d, explain why this increases or decreases your confidence that one of our algorithms is the best one possible for solving this problem.

Part V: Connect cybersecurity and time complexity

22. Using an RSA key to encrypt or decrypt data takes time. If we use bigger numbers for the RSA key, it takes more time for the sender's and receiver's computers to handle the message. However, breaking RSA encryption only requires that the eavesdropper is able to factor the first number in the public key. Using a key with larger numbers makes it take longer to crack.

    - Explain why people don't want to use longer keys than necessary.
    - Explain why people don't want to use shorter keys than necessary.
    - The key length that is necessary keeps changing. In 2014, the National Institute of Standards and Technology required RSA keys be at least 2048 bits long (~617 decimal digits) for commercial uses like sending a credit card by HTTPS, although 1024 bit keys were allowed through 2013. Why does the necessary key length keep getting longer?
    - To be useful in sending a credit card, an RSA key has to be able to be used by our function `use_keys()` in, at most, a few seconds, but the first number has to take months to factor with our function `prime_factors()`. Think about who uses a key to encrypt a credit card and who breaks a key by factoring it. Why is there such a big

difference between the maximum time `use_keys()` and the minimum time for `prime_factors()` for a useful key?

23. The algorithm to encrypt a message with an RSA key of length n has complexity ***O(n)***. That means that using a key that is twice as long takes 2 times as long to encrypt. We don't know for sure yet, but we think the problem of factoring an integer is at least as complex as $O(n^{100})$. That would mean that factoring an RSA key that is twice as long would take 2^{100} times as much time, which is equal to 1,000,000,000,000,000,000,000,000,000,000 times as much time. If the RSA algorithm really does have such a lower complexity than the factoring-an-integer problem, why does that guarantee that RSA will stay secure no matter how fast computers get?
24. Encryption algorithms like the RSA algorithm can be combined with protocols to make Internet communication secure. When encryption is applied to an HTTP transmission, for example, the resulting protocol is called HTTPS. Not everything in an HTTPS transaction gets encrypted, though. Think back to what you know about the HTTP protocol. Why would the client and server addresses still be visible and unencrypted for third parties along the way?

25. Problems for which a possible solution can be *checked* in polynomial time are known as **NP problems**. It is fast to *check* whether you have found factors of a given integer, so factoring an integer is an NP problem. But it is not so fast to *find* the factors of an integer. Problems that can be *solved* in polynomial time are known as **P problems**. Our best known algorithms for factoring a number are not as fast as polynomial time. We don't know whether factoring can be done in polynomial time.

    The Clay Foundation offers a $1,000,000 prize to any person who proves - or disproves - that all NP problems are P problems. Another of their seven $1,000,000 prizes is offered for proving the Riemann Hypothesis, a proof that many believe would reveal the mysterious pattern of prime numbers. Of the seven million dollar prizes, only one has been won, and that mathematician, Grigory Perelman, refused to accept it, saying that it would cheapen his work. Can you explain why solving these problems would be worth much more than a million dollars?

Conclusion

1. What is the difference between theoretically and empirically analyzing an algorithm?
2. Creating a new and more efficient algorithm for a common problem like searching, sorting, aligning, encrypting, or factoring can be a tremendous discovery. Why?
3. Why does Internet security depend on processing speed and the time complexity of algorithms?