600.445/600.646 Practical Cryptographic Systems January 24, 2025

Assignment 1

Instructor: Matthew Green Due: 11:59 pm, 6th February 2025

The assignment must be completed individually. You are permitted to use the Internet and any printed references, though your answers and code must be your own!

Problem 1

As part of implementing a cryptographic protocol, you need to sample a number uniformly at random from the set $\{0, \ldots, N\}$, where N is a positive integer. You decide to use Python's `os.urandom` function which generates cryptographically secure random data. However, the `os.urandom` function takes an integer ℓ as input and produces ℓ random *bytes* of data, effectively generating a random number in $\{0, \ldots, 256^\ell - 1\}$. How do you convert this into a random number between 0 and N (both inclusive), for any N of your choice?

You turn to Alice and Bob for help, who both provide functions that they claim generates a random number in the required range. The file `problem_1.py` provided as part of the assignment contains Alice's and Bob's functions. Your task is to compare their approaches and determine which one is more suitable for use in your protocol.

Submission instructions: Answer the questions below in a *PDF* document and include it in your submission.

1. Compare the runtime of Alice's and Bob's functions for $N = 256$. Do you observe a significant difference in their runtimes? If yes, explain the reasons behind the difference.

   You might find the inbuilt `timeit` module helpful for benchmarking the runtimes.

   [5 points]

2. Compare the runtime of Alice's and Bob's functions for $N = 255$. How does the relative performance differ from the case where $N = 256$? Provide an explanation for any observed changes.

   [2 points]

3. Calculate the expected number of iterations in the loop of Alice's function when $N = 256$.

   [3 points]

4. Generate 1,000,000 samples each using Alice's and Bob's functions, with $N = 103$. Plot the frequency of each value in the set $\{0, \ldots, 103\}$ for both set of samples. Which function's output is closer to the uniform distribution on $\{0, \ldots, 103\}$, and why?

   [10 points]

5. **(Bonus Question)** Can you modify Alice's function so that the expected number of iterations for any N is at most 2?

Problem 2

In this problem, you will gradually work through the steps needed to break the Vignère cipher when used to encipher sufficiently long English language texts.

Submission instructions: Complete the functions in `solution_2.py` and include the file in your submission. Your submission for this problem should consist only of `solution_2.py`.

Your solution will be auto-graded by running it on a number of test cases using the test harness implemented in `problem_2.py`. One such test case is provided in `sample.json`. Refer to the comments in the source files for additional details.

1. Implement the `break_caesar_cipher` function in `solution_2.py`. This function takes a ciphertext encrypted using a Caesar cipher as input and uses *frequency analysis* to recover the corresponding plaintext, without knowing the key. In a Caesar cipher, the key (a single byte) is XORed with each byte of the plaintext.

   Your function must reliably recover the plaintext for sufficiently long (at least 2,000 characters) English language texts.

   [40 points]

2. Implement the `find_vigenere_key_length` function in `solution_2.py`. This function takes a ciphertext computed using a Vigenère cipher as input and outputs the most likely guess for the length of the key. Use the *index of coincidence* method to find the length of the key.

   Your implementation must be reliable for keys up to 20 characters in length, when used to encipher English language texts of at least 20,000 characters.

   [25 points]

3. Implement the `break_vigenere_cipher` function in `solution_2.py`. This function takes a Vigenère ciphertext and the key length as inputs, and recovers the corresponding plaintext.

   [15 points]