Problem 4: Cryptonite

By Nalin Ranjan

By almost all counts, this problem was at least the most tedious, if not the hardest. No one was able to solve part (ii).

(i) The idea for this one was to turn it in to a normal crib-dragging problem by observing that either $c_1 \oplus c_2 \oplus 1 = m_1 \oplus m_2$ holds, or $c_2 \oplus c_3 \oplus 1 = m_2 \oplus m_3$ holds. (Here 1 denotes the number, represented in $42 \times 8 = 336$ bits.) More precisely, if s's last bit was one, then the former must be true, while if s's last bit was two, then the latter must be true. Intuitively, this is saying that out of three consecutive numbers, there exists a consecutive pair of these numbers that share all but their least significant bit.

Once you get the XOR of two plaintexts, you can proceed as if you were just cracking a normal OTP. In this part, it happened that $c_2 \oplus c_3 \oplus 1 = m_2 \oplus m_3$ held. However, even if you chose to crib-drag on $c_1 \oplus c_2 \oplus 1$, you would still be able to recover the vast majority of the plaintext, because $(s+1) \oplus (s+2)$ was all zeroes, except in the eight least significant place values. So you would have been able to recover all but the last character (byte) in the plaintexts, and used what you had recovered already to guess the last byte of any plaintext. And once you recover any one of the plaintexts fully, you can XOR it with the corresponding ciphertext to get the secret key, which will suffice to crack all of the ciphertexts.

A Note on Crib-Dragging: It seemed like many people thought that words like `princeton` or `eisgruber` were in the plaintexts, but this was not the case. The idea was to use the structure of the English language (common words, the use of spaces, etc.) to guess what might be in the ciphertexts. In this case, the plaintexts were

```
m1 = "you miss 100% of the shots you don't take."
m2 = "when you reach a fork in the road, take it"
m3 = "why aren't you responding to me? so sad :("
```

So crib-dragging words like `you`, `the`, `of`, `when`, `why`, `to`, `me`, and so on would have yielded something intelligible. Another indispensible trick: when you crib drag a word, try doing it with spaces at the beginning and end. This gets you two more characters as long as the word was not at the very beginning or very end of the plaintext.

(ii) The second part was more of an open-ended problem. The best solution we know of requires knowing most of the ciphertexts, and is a brute-force method. It depends on the observation that if we know what the last k bits of s_1 and s_2 are, then we know what the last k bits of $s_1 + i \cdot s_2$ are, for any i.

The idea is to brute-force guess the last bytes (8 bits) of s_1 and s_2, calculate the last byte of $s_1 + i \cdot s_2$ for $1 \leq i \leq 10$, use this to decode all ciphertexts, then eliminate all choices where any of the last bytes of the decoded plaintexts were not allowed characters (lowercase letters or the six additional punctuations we gave you). With access to 10 ciphertexts, there was almost always only one choice, so you didn't even have to determine which of two options made more sense. Once you recovered the last byte, you would enumerate all possibilities for the second-to-last byte and do the same procedure as before, eliminating any choices where any of the second-to-last bytes weren't allowable characters. This method works without not much if you know all ciphertexts, takes a decent bit of effort if you only know 6-8, and becomes nearly impossible if you know fewer, since the number of valid possibilities per byte you are trying to crack grows too large.

We've attached some code in `SolutionCryptoniteII.java` for you to play with. It takes one command-line argument n, which is the index of the byte that you want to brute-force. Start off with $n = 41$ (the last byte), and you'll notice that there is only one possible choice of the last bit of s_1 and s_2 that decodes the last bit of all the ciphertexts to an allowable value:

```
s1[n] = 0xdb, s2[n] = 0xa6
The Last 1 characters of the resulting Decoded Plaintexts:
n
u
?
s
.
!
!
?
e
!
```

which suggests that the last byte of s_1 is 0xdb and the last byte of s_2 is 0xa6. We can now fill this information in on lines 72 and 77 of `SolutionCryptoniteII.java`, recompile, and then run `java SolutionCryptoniteII 40` to try and figure out the next bytes of s_1 and s_2. The more bytes we recover, the easier it becomes, since words start to be revealed. For example, once you've revealed fifteen characters, the program prints (see next page)

```
s1[n] = 0x19, s2[n] = 0x39
The Last 15 characters of the resulting Decoded Plaintexts:
u cooler, dolan
 u. i'll show u
sity president?
n't you jealous
books than you.
physics than u!
n paul stevens!
economic forum?
ing your coffee
't do it again!
```

so we could guess that the second to last word in plaintext 3 is `university`, the fourth to last word in the last plaintext is `won't`, etc. You can find the decoded plaintexts under the `decoded_plaintexts` directory of the solution files.

This one was definitely a hard one though! It would probably have been more reasonable to solve this in a 24-hour period.

Plaudits:

- Congratulations to Alex Valtchanov '22 and Brandon Huynh '22, as well as Devon Ulrich '23 and Andrew Chen '23, for solving part (i).
- Since no one solved part (ii), Alex Valtchanov '22 and Brandon Huynh '22 will win the "best code-breaker" prize for solving part (i) the fastest!