# Chapter 5
# SOLUTIONS

This section describes how to complete the Cyber Warfare Simulations.

## 5.1 Simulation 1

This simulation was about breaking encryption.

---

**Ex1** The file for this task contains encrypted passwords that we need to decrypt. There are two things to note at this point:

1. We know that all passwords are dictionary words, but we do not know what dictionary they come from – although it is probably safe to assume that all words are English words.
2. We do not know how the words were encrypted.

The first point is easy to address: we can download dictionaries from the web and try them with whatever program we write until we have decrypted all words.

The second point is trickier, since there are basically infinitely many possible encryption methods that could have been used. Aside from contextual clues (e.g. this being the first exercise of a the first Cyber Warfare Simulation), we can look at the file contents for additional clues. We can note that each encrypted password has a different length and that we exclusively have alphabetical characters. This suggests that e.g. a hashing function was not used. Based on the given strings, it seems likely that whatever encryption method was used simply maps characters of the passwords to other alphabetical characters.

At this point, we need to experiment. It makes sense to start with known encryption methods. The simplest well-known method is probably a Caesar cipher which maps characters from the alphabet to an alphabet where all characters are shifted by some offset (the key). To test whether this is the correct cipher, we can write a program in a language of our choice which reads in the encrypted strings one-at-a-time, tries all possible keys and for each checks whether the result is in the dictionary.

---

**Ex2** There are two encrypted emails that need to be decrypted. The words in both files look similar to what we observed for **Ex1**. Unlike for **Ex1**, it is probably easier to start doing this by hand since emails tend to have certain formats (e.g. a greeting at the start) so we can speculate what some of the words might be and how they map to characters in the alphabet. If we do this, we quickly realise that the first email is not encrypted using a Caesar cipher but that there seems to be an arbitrary mapping of characters. A substitution cipher has been used. However, once you have made a start working out what the mappings are, it should be possible to gradually decrypt the entire first email.

The second email is trickier and it does not appear to use the same substitution cipher as the first email. However, like with the first email, we can try to guess what some of the words in the email are, particularly since it seems to be a response to the first email as the first word in the second email is the same as the last word in the first email. We can then also guess that the last three words of the second email are the same as the first three of the first email. If we write down the character mappings, we will see that (except for the first word), they do not agree with those of the first email. We can conclude that the mappings in the second email must be different. However, since the key for the first word is the same as for the first email, it seems sensible to conclude that the mapping changes from word to word, so the question is: how do we determine the mapping for each word? It seems reasonable to assume that there is some connection between the mappings for each word, rather than completely random mappings for each word (which would be very difficult to figure out) so we should continue by looking for similarities between the mappings. Perhaps after some experimentation, we can work out that the characters in each mapping are shifted from the mapping used for the first email. Indeed, the mapping is shifted by 1 for each word so that, knowing the key for the first word, we can then work out the subsequent keys.

---

**Ex3** Similarly to **Ex1** we have a file with passwords. However, this time they are hashed (one-way encryption) and do not use symmetric key encryption. Solving this task is similar to **Ex1**, except that we need to work out what hashing algorithm was used and then hash all words in the dictionary in order to figure out the hash values of words in the dictionary. Looking at the strings in the file, we can see that they appear to be in hexadecimal notation (as is usual for representing hash values as text). Based on the length of the strings, we can determine how large the hashes are: 16 bytes (128 bits). Based on this information, we can narrow down the search space to hashing functions which produce 128 bit hashes. An obvious candidate is MD5 which turns out to be solution.