Solutions Report

Modern Cryptology (CS641)

Computer Science and Engineering

Levels 1-4

Team: **team58**
Ayush Bansal (160177)
Gunjan Jalori (170283)
Siddharth Nohria (160686)

Date: February 29, 2020

1 Chapter 1 (The Entry)

There are 5 sub-levels in the chapter, first 4 of these don't have any cipher which needs to be decrypted.

The last sub-level is a **Substitution Cipher**, the answer to - "how it was recognised and solved" is explained in the subsection after the following list of commands.

Below is the solution to each of the sub-levels:

1. go
2. read
3. enter
4. read
5. cyLe70Lecy

1.1 Substitution Cipher

The ciphertext given was:

```
Nwy dejp pmcplpz cdp sxlrc adegipl ws cdp aejpr. Er nwy aem rpp cdplp xr mwcdxmv ws xmcplprc
xm cdp adegipl. Rwgp ws cdp qecpl adegiplr fxqq ip gwlp xmcplprcxmv cdem cdxr wmp, x
eg rplxwyr. Cdp awzp yrpz swl cdxr gprrevp xr e rxgbqp ryircxcycxwm axbdpl xm fdxad zxvxcr
dejp ippm rdxscpz in 2 bqeapr. Swl cdxr lwymz berrfwlz xr vxjpm ipqwf, fxcdwyc cdp hywcpr.
```

For identifying what kind of cipher is applied in the above text, we will use the **Index of Coincidence**.

The detailed explanation on *Index of Coincidence* can be found in section 5.1.

The *Index of Coincidence* of the above ciphertext is about 0.07, which is approximately same as a valid English text, this suggests that the cipher used is *Mono-alphabetic* such as *Substitution Cipher*.

For Solving the *Substitution Cipher*, the following steps were employed:

1. Calculate the frequency of each of the characters in the ciphertext, ignoring anything which is not an english alphabet.
2. The Character with the highest frequency is most probably 'e' or 'a', which can be placed in its place and identified further.
3. As the places get revealed, played hangman to find out what the other characters might be looking at one-letter, 2-letter, 3-letter words with highest number of characters revealed.
4. Built the decryption key by keeping a map of characters as they are being replaced.
5. Finally used the decryption key to decrypt the code given for the solution.

The code used in this part is in the file - `break_substitution.py`.

The Steps employed in the hangman game and building the key are mentioned below:

```
key = {}
key['p'] = 'e'    # Because 'p' has very high frequency
key['r'] = 's'    # 'r' has very high frequency,
                  #  _ee word exists, matches with "see" not "bee"
key['i'] = 'b'    # _e word exists, matches with "be"
key['n'] = 'y'    # b_ word exists, matches with "by"
key['m'] = 'n'    # bee_ word exists, matches with "been"
key['w'] = 'o'    # _ne word exists, matches with "one"
```

```
key['s'] = 'f'    # o_ word exists, 'n' is already taken, matches with "of"
key['l'] = 'r'    # fo_ word exists, matches with "for"
key['y'] = 'u'    # yo_ word exists, matches with "you"
key['g'] = 'm'    # so_e word exists, matches with "some"
key['z'] = 'd'    # use_ word exists, 'r' is already taken, matches with "used"
key['c'] = 't'    # en_ered word exists, matches with "entered"
key['d'] = 'h'    # t_e word exists, matches with "the"
key['x'] = 'i'    # f_rst word and _ (single letter word) exist,
                  # matches with "first" and "i"
key['e'] = 'a'    # single letter word exists, 'i' is already taken, matches with "a"
key['j'] = 'v'    # ha_e word exists, matches with "have"
key['a'] = 'c'    # _hamber word exists, matches with "chamber"
key['v'] = 'g'    # nothin_ word exists, matches with "nothing"
key['f'] = 'w'    # _hich word exists, matches with "which"
key['q'] = 'l'    # be_ow and wi__ word exists, matches with "below" and "will"
key['b'] = 'p'    # sim_le and ci_her word exists, matches with "simple" and "cipher"
key['h'] = 'q'    # _uotes word exists, matches with "quotes"
```

The plaintext revealed after using the above decryption key is:

```
You have entered the first chamber of the caves. As you can see there is nothing of interest
in the chamber. Some of the later chambers will be more interesting than this one, i
am serious. The code used for this message is a simple substitution cipher in which digits
have been shifted by 2 places. For this round password is given below, without the quotes.
```

```
# For the case of integer digits, "1" must be subtracted from each digit, as mentioned
# text after decryption, it was "2" but it itself was shifted so
# x+x = 2, this gives x = 1
```

So, final plaintext is:

```
You have entered the first chamber of the caves. As you can see there is nothing of interest
in the chamber. Some of the later chambers will be more interesting than this one, i
am serious. The code used for this message is a simple substitution cipher in which digits
have been shifted by 1 places. For this round password is given below, without the quotes.
```

Using the above decryption key and the logic for digit, we can decipher the code for the answer as well:

Code: `anQp81Qpan`
Solution: `cyLe70Lecy`

2 Chapter 2 (The Caveman)

There are 2 sub-levels in the chapter, first one doesn't have any cipher which needs to be decrypted.

The second sub-level is a **Vigenere Cipher**, the answer to - "how it was recognised and solved" is explained in the subsection after the following list of commands.

The detailed explanation on *Vigenere Cipher* can be found in section 5.3.

Below is the solution to each of the sub-levels:

1. read
2. the_cave_man_be_pleased

2.1 Vigenere Cipher

The ciphertext given was:

```
Lg ccud qh urg tgay ejbwdkt, wmgtf su bgud nkudnk lrd vjfbg. Yrhfm qvd vng sfuuxytj
"vkj_ecwo_ogp_ej_rnfkukf" wt iq urtuwjm. Ocz iqa jdag vio uzthsivi pqx vkj pgyd encpggt.
Uy hopg yjg fhkz arz hkscv ckoq pgfn vu wwygt nkioe zttft djkth.
```

For identifying what kind of cipher is applied in the above text, we will use the **Index of Coincidence**.

The detailed explanation on *Index of Coincidence* can be found in section 5.1.

The *Index of Coincidence* of the above ciphertext is about 0.042, which is closer to the uniform distribution of English text, this suggests that the cipher is *Poly-alphabetic* such as *Vigenere Cipher*, it may be some other Poly-alphabetic cipher as well but we still have to give it a shot.

For solving the *Vigenere Cipher*, the following steps were employed:

1. Remove all characters from the text which are not part of the English alphabets and capitilize all characters.
2. Partition the text according to different key lengths and sort them according to the *Index of Coincidences* achieved, since higher the IC, closer it is to valid English Text.
3. For each keylen, perform frequency analysis to get the best key possible with the given length.
4. Try out all the keys retrieved and see which one gives some valid English text.

The code used in this part is in the file - `break_vigenere.py`.

The plaintext revealed after using the above decryption key is:

```
Be wary of the next chamber, there is very little joy there. Speak out the password
"the_cave_man_be_pleased" to go through. May you have the strength for the next chamber.
To find the exit you first will need to utter magic words there.
```

From the above, the solution is revealed: `the_cave_man_be_pleased`.

3 Chapter 3 (The Holes)

There are 4 sub-levels in the chapter, first 3 of these don't have any cipher but there are different tricks which need to be employed to get to the final sub-level.

The last sub-level is a **Permutation-Substitution Cipher**, the answer to - "how it was recognised and solved" is explained in the subsection after the following list of steps/commands.

Below are the solution steps to get out of the final chamber:

1. Type `enter` to go to sub-level 2.
2. At sub-level 2, you try to `put` your hand in the small hole, it is bitten, denoting there is someone there.
3. Type `enter` to go to sub-level 3, here there are a lot of mushrooms growing on the ground.
4. Type `pick` to pluck some mushrooms and come back to sub-level 2.
5. Type `give` to give the mushrooms to whatever is there in the small hole.
6. There is a spirit here, who gives you the code `thrnxxtzy` which can reveal a hidden door at the entrance chamber (sub-level 1).
7. Go back to sub-level 1 and type `thrnxxtzy`, this reveals a hidden door with a glass panel beside it.
8. Type `read` to get the ciphertext and code.
9. Type `jyg_izuqo_rr`, which is the decoded plaintext from the cipher provided.

3.1 Permutation-Substitution Cipher

The ciphertext given was:

```
cpiftgt ef oldo ukuq vtyp vv ptttqkk dp txe tkcnmbi uxkfft ueukwuqe ad uwv ttdo. da tocwc,
qqc qgcu woyg cx cpifteud wat tvkbd vu owk zelc dp txe vthr uccfgg. keb dteuof ut gle
dzcc rtc wv ukkyyc xxuo edw. mqgu zec dtyac uldw cqev evyu xvo tee moo mt gle dkcur.
tm evyoi qtzc cxz o mlcuauoc, vw wetd kkcc gwhego! cf da foedokm, aibet ccd ktbfkqyo:
```

For identifying what kind of cipher is applied in the above text, we used the following techniques:

- The **Index of Coincidence** of the above ciphertext is about 0.057, which is very close to that of valid English text, this suggests that the cipher used is *Mono-alphabetic* such as *Substitution Cipher*, the detailed explanation on *Index of Coincidence* can be found in section 5.1.
- The **Chi-squared Statistic** of the above ciphertext is about 157 against *uniform distribution*, this suggests that the cipher used is *not Poly-alphabetic* since it is not closer to uniform distribution, the detailed explanation on *Chi-squared Statistic* can be found in section 5.2.
- The **Chi-squared Statistic** of the above ciphertext is about 958 against *valid English text*, this suggests that the cipher used is **not** a *Simple Permutation* of letters.
- Based on the above, we try out different forms of *Mono-alphabetic* ciphers first instead of *Poly-alphabetic*.

Firstly, we will try to solve the cipher assuming it is *Simple Substitution Cipher*. This doesn't seem to work, since we are not able to get any valid English text from it.

The code for solving the Substitution Cipher uses the **n-gram** approach and it is in the file: `ngram_score.py`.

Since a *Simple Substitution Cipher* doesn't work here, it could be some other form of *Mono-alphabetic* cipher.

Lets make some observations about the ciphertext:

- The occurrences of double letter phrases in words is very frequent and at very odd places, see the below text:
  `cpiftgt ef oldo ukuq vtyp vv ptttqkk dp txe tkcnmbi uxkfft ueukwuqe ad uwv ttdo. da tocwc, qqc qgcu woyg cx cpifteud wat tvkbd vu owk zelc dp txe vthr uccfgg. keb dteuof ut gle dzcc rtc wv ukkyyc xxuo edw. mqgu zec dtyac uldw cqev evyu xvo tee moo mt gle dkcur. tm evyoi qtzc cxz o mlcuauoc, vw wetd kkcc gwhego! cf da foedokm, aibet ccd ktbfkqyo:`
- The character 'o' appears as a single letter, if we assume it to be 'a' or 'i' (according to english text), then the word 'moo' will coincide to '_aa' or '_ii' which will not come out to be a valid English word.
- A simple substitution solver doesn't give us a valid result for the ciphertext.

The above observations suggest the following things:

- The letters in the ciphertext needs to be permuted before applying Substitution, this permutation can be a block permutation or matrix permutation or maybe something entirely different.
- The cipher is *Poly-alphabetic* (less-likely).

Firstly, we will try out **block-permutation** along with **Substitution**, i.e. **Simple Permutation-Substitution Cipher**.

For solving the *Simple Permutation-Substitution Cipher*, the following steps were employed:

1. Remove all characters from the text which are not part of the English alphabets, noting their position in the text since they will have to be added back at the end.
2. Calculate the *block length* (for permutation) using the idea that block length will be a *factor of the total number of characters*.
3. For each permutation get the permutated text from the ciphertext, and insert all the special characters at their designated places in the text.
4. Apply a Simple Substitution Cipher Solver to the text and see whether it gives a valid English Text.
5. If no permutation gives a successful result, try other block length till we reach some valid English Text or run out of factors.

The code used in this part is in the file - `break_perm-subs.py`.

The *total number of characters* in the ciphertext is 270.
Another observation to make here is that the code to be deciphered: `uhs_xafmf_no` has total of 10 characters.
This suggests that the block length will be a factor of both 270 and 10, so it can be 2, 5 or 10, and we will try each of these one by one.

Using the method described above, we got each permutation of ciphertext corresponding to block-length 2 first, but none of them returned a valid English text after solving the Substitution, so we changed block-length to 5.

Using block-length 5 revealed the following plaintext after a certain permutation of ciphertext was solved using Substitution Cipher Solver:

```
breaker of this code will be blessed by the squeaky spirit residing in the hole. go ahead,
and find away of breaking the spell on him cast by the evil jaffar. the spirit of the
cave man is always with you. find the magic wand that will let you out of the caves.
it would make you a magician, no less than jaffar! to go through, speak the password:
```

At this point, we got a valid English text from the ciphertext, so we won't be moving forward and trying out different block-length or other ciphers.

Using the permutation and decryption key retrieved from solving above cipher, we can decipher the code for the answer as well:

The decryption key retrieved from the Solver gave us 2 possible solutions, since we did not have mapping for all 26 characters:
Code: `uhs_xafmf_no`
Solution 1: `jyg_izuqo_rr`
Solution 2: `jyg_ixuqo_rr`

Finally, Solution 1 was the answer.

4 Chapter 4 (The Spirit)

This level is tricky as we had to go back to the previous level and perform some task before we could proceed further.

There are 3 sub-tasks here, firstly we have to retrieve a *Magic Wand*, secondly we have to *free the Spirit*, finally solve the **DES Cipher** to advance to next level.

Below are the solution steps for each of the tasks listed above:

- Retrieving the Magic Wand:
  1. Type `enter` to go ahead in the chamber.
  2. Type `dive` to take a dive into the lake.
  3. At this point, we see an object looking like a wand, but trying to pull it directly causes us to drown, so first go back to surface and take a deep breath.
  4. Type `dive` and `pull` the wand.
- Freeing the Spirit:
  1. At this point, we can't figure out any way out, the screen in the chamber door is also blank and wand does not help us here.
  2. We can recall that there was an old man/spirit before who helped us in chamber 3 and mentioned that he was trapped by someone, he could be freed by the magic wand.
  3. We go back to chamber 3, `wave` our wand infront of the hole where the old man's spirit was.
  4. The spirit is freed and says that he will help us along the way.
- Solving the **DES Cipher**:
  1. After entering the 4th chamber, type `read`.
  2. The screen is still blank, but the spirit tells us what is supposed to be there:
     ```
     This is a magical screen. You can whisper something close to the screen and
     the corresponding coded text would appear on it after a while. So go ahead and
     try to break the code! The code used for this is a 4-round DES, so it should
     be easy for you!! Er wait ... maybe it is a 6-round DES ... sorry, my memory
     has blurred after so many years. But I am sure you can break even 6-round DES
     easily. A 10-round DES is a different matter, but this one surely is not 10-round
     ...(long pause) ... at least that is what I remember. One thing that I surely
     remember is that you can see the coded password by whispering 'password'. There
     was something funny about how the text appears, two letters for one byte or
     something like that. I do not recall more than that. I am sure you can figure
     it out though ...
     ```

4.1 3-Round DES Cipher

Earlier, we were trying to figure out how many rounds of DES Cipher is applied here, so that we can devise the algorithm for the same.

The task became easier when a hint was revealed stating that the DES Cipher is *3 Round*.

We will break *3 Round DES* using **Differential Cryptanalysis** as we had discussed in class.

The idea behind *Differential Cryptanalysis* was to get rid of the unknown value (i.e. the Key) so that an equation can be formed over the non-linear step (i.e. sBoxes).

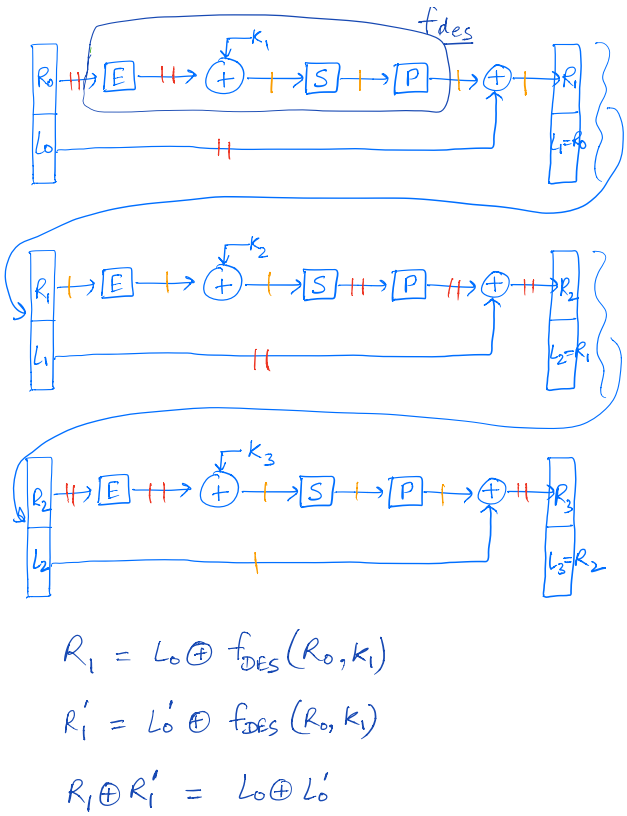

Figure 1: 3 Round DES

Important points about the figure above:

- *Double Red Line* implies we know both individual values of *Differential Cryptanalysis*.
- *Single Golden Line* implies we know the differential value of 2 inputs, not the individual values themselves.
- The pairs of inputs taken during Differential Cryptanalysis must have equal 32 bits on the right side.

Solving 3 Round DES:

1. We will start by figuring out the 48-bit Key for the 3rd Round.
2. We know the differential value, just after and before the S-Boxes, thus, there will be 4 possible pairs of input values to the S-Box as studied in class.
3. This narrows down the search for key over each 6 bits to 4 possibilities.
4. We will pick another input here, take the intersection of possibilities and narrow down the key.
5. After sufficient tries, we get the 48-bit 3rd round key.
6. Now only 8 bits of the key remain, these can be easily brute-forced and figured out using the other round values.

The code used in this part is as follows:

- `constants.py`: Contains the constants for the DES.
- `des.py`: Defines the DES Class encoding all the functions related to it.
- `utils.py`: Defines the common utility functions related to DES and Key generation.

- `break_des.py`: Defines the main function which uses all the utilities and DES class to break the 3 Round DES according the steps described above.
- `generate_input.py`: Generates a pair of input which have equal 32 bits on the right side.

The key retrieved for the 3rd round of the DES is as follows:

```
[61, 28, 9, 54, 55, 9, 28, 51]
```

here, each value represents the 6-bits of the 48-bit key.

After doing brute force on the remaining part of the key, we get the following value:

147

The Final Key (64-bit including the parity bits) for the DES is as follows:

0111101001011100001010000011011011110010011010100110101011101000

The encrypted password: `gnushmilfrplulktkrtrogltjojfqjpt`
The decrypted password: `rirfiirqnujpopirgkholonsqntpkqqi`

5 Appendix

This section explains each of the things used in between the solutions without proper explanation.

5.1 Index of Coincidence

The **Index of Coincidence** is a measure of how similar a frequency distribution is to the uniform distribution.

$$I.C. = \frac{\sum_{i=A}^{i=Z} f_i(f_i - 1)}{N(N-1)}$$

where f_i is the count of letter i (where $i = A, B, ..., Z$) in the ciphertext, and N is the total number of letters in the ciphertext.

Important facts about the *Index of Coincidence*:

- The *Index of Coincidence* of valid English text is about 0.066.
- The *Index of Coincidence* for uniform distribution of English text is about 0.038.
- The *Index of Coincidence* remains the same for the ciphertext and plaintext if cipher is **Mono-alphabetic** (i.e. Substitution Cipher).
- The *Index of Coincidence* of ciphertext is closer to uniform distribution if cipher is **Poly-alphabetic** (such as Vigenere Cipher).

We can get an approximate idea of what kind of cipher is used to generate the ciphertext by using the *Index of Coincidence*.

5.2 Chi-squared Statistic

The **Chi-squared Statistic** is a measure of how similar two categorical probability distributions are. If the two distributions are identical, the chi-squared statistic is 0, if the distributions are very different, some higher number will result. The formula for the chi-squared statistic is:

$$\chi^2(C, E) = \sum_{i=A}^{i=Z} \frac{(C_i - E_i)^2}{E_i}$$

where C_A is the count (not the probability) of letter A, and E_A is the expected count of letter A.

Important facts about the *Chi-squared Statistic*:

- If the *Chi-squared Statistic* of a ciphertext against *uniform distribution* is very low (~50 or less), then it is highly probable that the cipher is *Poly-alphabetic*.
- If the *Chi-squared Statistic* of a ciphertext against *valid English text* is high and the cipher is *Mono-alphabetic*, then it can be solved by trying keys and lowering it.

We can get an approximate idea of whether the cipher is *Poly-alphabetic* or not by using *Chi-squared Statistic*.

5.3 Vigenere Cipher

The Vigenere Cipher is a polyalphabetic substitution cipher.

Suppose, the length of the encryption key is k, then the string formed by picking out each letter with a multiple of k letters in between them will be a *Caesar Cipher*.

Since each such string is a *Caesar Cipher*, the *Index of Coincidence* of this string will be closer to that of valid English text rather than closer to uniform distribution.

Using the above principle, we can crack the *Vigenere Cipher*.