Cryptography Summative 2021 Assignment

Alisa Hussain

Thursday 21st January, 2021

1 Introduction

The objective of this assignment was to decrypt a 16 byte long ciphertext that was encoded using data encryption standard (DES) encoding with the electronic code book (ECB) mode of operation. More precisely, the original plaintext consisted of three words separated by a dot, representing a 3m x 3m square area on Earth *e.g. heavy.bravo.goals*. Further information included the 16 byte ciphertext in hexadecimal and the fact that the key was 8 bytes long. We are provided an *.exe* file, acting as a black box oracle which takes in as input a hexadecimal string and returns an encrypted hexadecimal string. Unfortunately a Windows OS is inaccessible, and consequently it is not possible to implement the attack plan on the given oracle. Rather, the approach has been tested using self-made test cases, by means of the *des* python library. This report will detail an analysis of the problem, a proposed solution and finally an analysis of the approach.

2 Problem Analysis

ECB We know encryption was performed in ECB mode of operation. This differs from CBC and CFB modes in that it does not use an initialisation vector. This means that any two identical 8 byte block of plaintext always produces the same ciphertext. This allows us to reduce the problem into two subproblems, significantly decreasing computation required.

Ciphertext The given ciphertext is 16 bytes, which means the input was a maximum of 16 characters, or less if padding was used.

Plaintext We know the plaintext is of the format $w_1.w_2.w_3$ where w_i is a word in the English dictionary. There is a finite number of words, allowing a brute force dictionary attack. Given the oracle means access to infinite plaintext-ciphertext pairs, facilitating a chosen plaintext attack. We know the input had a maximum length of 16 including two full stops. Research into what3words revealed candidate words were a minimum length of 4, and excluded proper nouns, compound and offensive words[1].

3 Attack Plan

Research revealed several existing chosen plaintext attack methods. DES is known to have been designed with differential cryptanalysis in mind. However, theoretically, the method can successfully cryptanalyse with 2^{47} known plaintexts. Linear cryptanalysis is not particularly more practical, also requiring 2^{47} known plaintexts[2].

These times seem too impractical, so after analysing the problem, we propose an alternate attack method - an informed dictionary-based brute force analysis.

The first step entails creating the dictionary to select words from for the plaintext. We orignally used a dataset containing all 370,103 words in the English language[3]. Since the maximum length of the plaintext is 16 and the minimum length word used by what3words is 4[1], potential words could only be between length 4 and 6. We represent the structure of the plaintext as $w_1.w_2.w_3$, where w_i is a word of length, λ, $4 \leq \lambda \leq 6$. The final dataset had filtered out words that did not fit this condition, along with offensive, and disused words[4]. This process automatically reduced the dataset to 18,000.

[1] https://support.what3words.com/en/articles/2212810-what-are-the-shortest-and-longest-words-used
[2] Matsui, M. "Linear cryptanalysis method for DES cipher". Advances in Cryptology - EUROCRYPT, 1993
[3] https://github.com/dwyl/english-words/
[4] https://github.com/RobertJGabriel/Google-profanity-words

For the attack, we consider each 8 bytes separately. We exploit the ECB vulnerability, that the encoding of a block of 8 bytes will always result in the same ciphertext. We iterate through the word dataset once for each word, using the entirety of a word for w_1 of length λ, a full stop, and the first $7 - \lambda$ characters of a word, w_2, to make up 8 bytes. For example, using the two words *"heavy"* and *"bravo"*, the byte tested would be *"heavy.br"*. To avoid duplicate calls to the oracle we store test cases in a dictionary. This proves effective when two words have the same initial characters, such as ***br**ass* and ***br**eak*. After the first byte's corresponding ciphertext matches with the first 8 bytes of the given encrypted message, we proceed to the second stage.

For the second stage, test cases are limited, since the first 1 - 3 characters of w_2 are already determined. This significantly reduces the scope of w_2 candidates, and in turn the length of w_3. We iterate through possible test cases, comparing corresponding ciphertext to that given, until a match is achieved.

We also looked into the What3Words API for tools that may help this endeavour as an heuristic. The Autosuggest[5] tool is free and allows unlimited calls. This tool can be used to correct addresses of that do not exist by suggesting similar ones. The API can be called once for each word to determine whether it exists in the application's dictionary. If no addresses are suggested with the word in question, it can be completely eliminated from our dictionary, preventing unnecessary calls to the oracle. In addition, for the second stage, the API can be called with the known initial letters of w_2, and words suggested would be the first to be tested.

4 Performance Analysis

With no access to the *.exe* oracle, this approach was tested using the python *des* library. Different 8 byte keys, and plaintexts of a maximum 16 bytes, with and without padding were used. The ciphertext was successfully cracked each time. The distribution of 4-, 5- and 6- length words in the English language is approximately 2.6%, 5.2% and 8.5% respectively [6]. Therefore the maximum number of tests for the initial 8 bytes is approximately $(\frac{8.5}{8.5+5.2+2.6} \times 18000 \times 26) + (\frac{5.2}{8.5+5.2+2.6} \times 18000 \times 26^2) + (\frac{2.6}{8.5+5.2+2.6} \times 18000 \times 26^3) = 54{,}589{,}471$ assuming a uniform distribution of letters. This is an overestimate, since we are disregarding the use of a dictionary saving previous calls, the use of an API and the fact that letters are not naturally uniformly distributed.

The number of tests required for the second half is more difficult to calculate, since the initial characters of w_2 have been determined. Given this information, the the pool of potential final 3-5 characters of w_2 decreases significantly, and the size of this pool is variable. For example there is an abundance of candidate words that begin with *"pla"*, and very few that begin with *"xra"*. In addition the possible lengths of w_3 are influenced by the length of w_2. Although the maximum number of tests is evidently less than that of the first half, let us assume the maximum is 54,589,471 again. This would result in a total number of tests less than $54{,}589{,}471 \times 2 = 109{,}178{,}942 \equiv 2^{26}$. Recall linear and differential cryptanalysis use 2^{47} known plaintexts. Our informed brute force method requires 2^{21} times less known plaintexts - a significant reduction.

For future attacks, //mention saving updated dictionary wihtout non existing words

[5] https://developer.what3words.com/public-api/autosuggest-component
[6] http://www.ravi.io/language-word-lengths