Cryptanalysis of a Shift Cipher with Opaque Scheduling Algorithm by Plaintext Search

Prashanth Ramakrishna [*] Hao Shu [†] Dov Salomon [‡] Jonathan Alter [§]

March 15, 2020

1 Introduction

The following is a report detailing out team's attempt to solve the crytanalytical challenges presented in Project 1. We begin by detailing these challenges.

Our goal is to decrypt messages encrypted with the following scheme. Let M, the message space, be such that $\forall m \in M$, $|m| = L \in \mathbb{N}$. That is, m can be represented as $m = m_1 m_2 ... m_L$, where $\forall i = 1, ..., L, m_i \in E$. Here, E represents the lower case English alphabet and a space character, $\{a, b, ..., z, space\}$. The ciphertext space, denoted C, is such that $\forall c \in C$, $c = c_1 c_2 ... c_L$, where $\forall i = 1, 2, ..., L, c_i \in E$. Formally, then, $M, C = E^L$ The key space is denoted K and defined $K = \{0, 1, ..., 26\}^t$. Each key in the key space can be represented as $k = k_1 k_2 ... k_t$. Ciphertext c is generated from message m with key k by encryption scheme E as follows:

$$E(m_i, k) = m_i + k_{j(i)} \pmod{26} = c_i, \tag{1}$$

where $j(i)$ is a "key scheduling function" that determines by which key character each m_i is lexicographically shifted.

The core task is to decrypt a given ciphertext. There are two challenges. In the first challenge, the plaintext is taken from a known, small, fixed set of several choices. In the second challenge, the plaintext is generated as randomly selected words from a small dictionary, separated by a space. In both cases, the key and the scheduling function $j(i)$ are unknown.

We believe we have finished the first challenge with complete success and the second challenge with partial success. This has been a collaborative effort. We consulted each other for the solutions for all challenges. The C-code for the first challenge was primarily written by Jonathan Alter, but developed in discussion with Prashanth Ramakrishna. The C-code for the second challenge was written and developed mainly by Hao Shu and Dov Salomon, with input from Jonathan Alter and Prashanth Ramakrishna. This report was primary written by Prashanth Ramakrishna, and edited by Dov Salomon. We did not consider any modification to the supplied encryption scheme.

2 Substitution/Permutation Cipher Survey

Before discussing our approaches to Project 1, we will travel through a brief survey of substitution and permutation ciphers as well as the various cryptanalysis techniques used to break them. This will provide an entertaining but useful background that will inform the original approaches that we have developed and detail later on.

[*] pmr347@nyu.edu
[†] hs3812@nyu.edu
[‡] dms833@nyu.edu
[§] ja3943@nyu.edu

2.1 Substitution Ciphers

As suggested by its name, a substitution cipher is a cipher in which plaintext letters or groups of letters are replaced with letters, numbers, or symbols according to a particular deterministic scheme. There have been many notable substitution ciphers over the centuries. Here, we will discuss the famous **Affine Cipher**, **Hill Cipher**, and **Vigenere Cipher**. For each cipher the corresponding cryptanalysis techniques used to recover the original message will also be discussed.

2.1.1 Affine Cipher

The Affine Cipher is the simplest class of substitution cipher. It is simply a monoalphabetic substitution. Characters in an alphabet are mapped to an integer equivalent. The plaintext is then converted to a sequence of integers. A key character is chosen. Each constituent of the plaintext is then forward shifted using a modular arithmetic encryption function that takes as input a plaintext constituent and the integer that the key character maps to. Once the entire message is shifted, the resulting integer sequence is then converted back to characters in the original alphabet, producing the final ciphertext.

Formally, let the given alphabet with which the plaintext is written be denoted by $A = \{a_0, a_1, ..., a_{|A|-1}\}$. Further define the mapping $\phi : A \to \mathbb{N}$ by $\phi(a_i) = i$. Then, each message m can be written $m = m_1m_2...m_n$ such that $\forall 1 \leq i \leq n, m_i \in A$. To construct the ciphertext from the plaintext, first we compute $\phi(m) = \phi(m_1)\phi(m_2)...\phi(m_n)$. Then we pass each constituent of $\phi(m)$ through a modular arithmetic encryption function $E : \mathbb{N} \to \mathbb{N}$, along with a chosen character key $\phi(k)$, where $k \in A$, and constant $a \in \mathbb{Z}$. E is defined by,

$$E(\phi(m_i), \phi(k), a) = (a\phi(m_i) + \phi(k)) \pmod{|A|} \tag{2}$$

Finally, we apply ϕ^{-1} to each constituent $E(\phi(m_i), \phi(k), a)$ of $E(\phi(m), \phi(k), a) = E(\phi(m_1))...E(\phi(m_n))$ to produce ciphertext $c = c_1...c_n$, where $c_i = \phi^{-1}(E(\phi(m_i), \phi(k), a))$.

Decryption is given by the function $D : \mathbb{N} \to \mathbb{N}$ such that $D = E^{-1}$. D, then, is defined,

$$D(\phi(c_i), \phi(k), a) = a^{-1}(\phi(C_i) - \phi(k)) \pmod{|A|}, \tag{3}$$

where a^{-1} is the modular multiplicative inverse of $a \pmod{|A|}$. Recall that $a^{-1} \pmod{|A|}$ only exists if $gcd(a, |A|) = 1$. Thus, Without a and $|A|$ co-prime, decryption may not be possible.

The first and most famous Aphine Cipher is the **Ceasar Cipher**, named after Julius Ceasar, in which $a = 1$ and $\phi(k) = 3$. If $A =$English, then there are 26 such Ceasar Ciphers. These are trivial to cryptanalyze. One can either brute force or use frequency analysis. In general, not including these 26 Ceasar Ciphers, there are only 286 non-trivial Affine Ciphers. This is because there are only 12 numbers less that 26 that are co-prime with 26. That is, there are only 12 possible values for a. For each value of a, there are 26 possible forward shifts, one for each character in the English alphabet. We exclude $a = 1$. There are thus $11 \times 26 = 286$ Affine Ciphers. Brute forcing through all possible encryption schemes is therefore computationally feasible.

2.1.2 Hill Cipher

The Hill Cipher was invented in 1929 by the American Mathematician Lester S. Hill. It works by the principles of linear algebra and is significant as the first substitution encryption scheme to operate on more than three characters simultaneously.

We retain the mapping ϕ that takes alphabet characters in A to integers. Applying ϕ to message $m = m_1m_2...m_n$, we obtain an $n-$vector of integers. Then, we define key K as an $n \times n$ invertible character matrix. That is,

$$\phi(K) = \begin{pmatrix} \phi(k_{11}) & \dots & \phi(k_{1n}) \\ \vdots & \ddots & \vdots \\ \phi(k_{n1}) & \dots & \phi(k_{nn}) \end{pmatrix} \tag{4}$$

Encryption amounts to matrix modular multiplication. That is E is defined

$$E(\phi(m), \phi(K)) = \phi(K)\phi(m)^t \pmod{|A|} = \phi(c_i)^t \tag{5}$$

Or, more explicitly,

$$E(\phi(m), \phi(K)) = \begin{pmatrix} \phi(k_{11}) & \dots & \phi(k_{1n}) \\ \vdots & \ddots & \vdots \\ \phi(k_{n1}) & \dots & \phi(k_{nn}) \end{pmatrix} \begin{pmatrix} \phi(m_1) \\ \vdots \\ \phi(m_n) \end{pmatrix} \pmod{|A|} = \phi(c_i)^t \tag{6}$$

$(\phi^{-1}(\phi(c)^t)^t$ gives the character ciphertext. Decryption is done simply by multiplying the resultant ciphertext vector $\phi(c)^t$ by $\phi(K)^{-1}$. That is,

$$D(\phi(c)^t, \phi(K)) = \phi(K)^{-1}\phi(c)^t = \phi(m)^t \tag{7}$$

$(\phi^{-1}(\phi(m)^t)^t$ retrieves the original character plaintext. Note that matrix multiplication places more constraints of $\phi(K)$ than just invertibility. If $\exists \phi(k_{ij}) \in K$ such that $k_{ij}||A|$ or if $\det(\phi(K)) = 0$, decryption may not be possible.

The Hill Cipher is, by modern standards, extremely insecure. It is deeply susceptible to a known plaintext attack. If an adversary collects n ciphertext/plaintext pairs, then a linear system can be set up and easily solved to retrieve K.

2.1.3 Vigenere Cipher

The Vigenere Cipher was invented by Giovan Battista Bellaso in the 16th century. However, in the 19th century, the cipher was misattributed to Blaise de Vigenere, who subsequently became its namesake. The Vigenere Cipher, in many ways, marked a new era of cryptography. It was considered unbreakable for centuries. Eventually, in 1863, Freidrich Kasiski, the German cryptographer, published the first successful attack on the Vigenere Cipher known as the **Kasiski Examination**.

The Vigenere Cipher is a polyalphabetic substitution cipher that uses a key word to interweave multiple monoalphabetic Ceasar encryptions. Let $k = k_1k_2....k_t$, where $\forall 1 \leq i \leq t, k_i \in A$, where A, as previously used, denotes the alphabet. Ciphertext c is generated from plaintext m and key k in the following way. First, a **keystream**, k_s is generated by repeating the key until it is the same length as the plaintext and, by extension, the resulting ciphertext. We retain the function ϕ from 2.1.1 that maps characters to integers. The keystream and plaintext are converted to integer sequences by ϕ. Then, encryption function $E(m, k_s$ is applied to obtain the ciphertext. E is defined

$$E(\phi(m_i), \phi(k_{s_i})) = (\phi(m_i) + \phi(k_{s_i}))(mod|A|) = c_i \tag{8}$$

Decryption is equally straightforward. $D = E^{-1}$ is defined

$$D(\phi(c_i), \phi(k_{s_i})) = (c_i - k_{s_i})(mod|A|) = m_i \tag{9}$$

The Kasiski Examination relies on the fact that if m is rearranged into groups corresponding to the equivalence classes of $\mathbb{Z}_t$ then the problem is simplified to solving t Ceasar Ciphers. First, the Kasiski Examination searches for repeated sequences in the ciphertext. Then, it computes the distances between these repeated sequences and counts the factors of these distances. Commonly repeated factors are likely key lengths. If the true key length is prime, then the most commonly repeated factor will be the true key length. However, if the true key length is composite, then all its factors will occur at least as many times as itself.

For each likely key length l, a the ciphertext is split into l equivalence class cuts. Each cut is decrypted by brute-forcing through every character in the alphabet and computing $D(\phi(c_{l_i}, \phi(\alpha_j))$, where c_{l_i} is the ith ciphertext cut for candidate length l and $\alpha_j \in A$. A frequency analysis is conducted on each decrypted cut to determine which α_j belongs fills which position in the key. A number of candidate keys can be generated in this way for each candidate key length. These keys can then be traversed to check which one produces the most plausible plaintext.

2.2 Permutation Ciphers

A permutation cipher, as its name suggests, is produced via a unique permutation – that is, a reordering function which describes how the character constituents of the plaintext should be rearranged. This is fundamentally different from a substitution cipher, in which characters previously not included in the plaintext may appear in the ciphertext. The canonical permutation cipher is the **Columnar Transposition Cipher**.

The Columnar Transposition Cipher permutes a message m using key k by organizing m in a grid whose width is determined by $|k|$. The columns of the grid are then concatenated into a ciphertext in alphabetical order of the key characters. That is, $m =$ "my name is god", encrypted with $k =$ "hope" is permuted as follows:

$$\begin{pmatrix} h & o & p & e \\ m & y & - & n \\ a & m & e & - \\ i & s & - & g \\ o & d & & \end{pmatrix} \rightarrow \text{n gmaioymsd e} \tag{10}$$

Cryptanalysis of the Columnar Transposition Cipher can be done using different approaches of varying efficiency. The main challenge is guess the correct key length. Once this is done, the cipher text can be organized into a grid of the correct dimension and brute force re-permuted to find the most plausible plaintext. This lends itself to a cryptanalysis paradigm known as **Hill Climbing**. Hill climbing works by initializing a random key guess, then gradually adopting small changes to the key in order to improve its "fitness". Fitness, here, can be thought of as how "English-like" a candidate decription is using a candidate key.

3 Solution for Test 1

The Test 1 Challenge is set up as a multiple choice question. We are given a ciphertext, generated using the modular arithmetic encryption scheme with opaque key scheduling, along with five plaintexts. We are then asked to choose which plaintext was used to generate the given ciphertext.

The main insight is that the key length determines the upper bound on the number of unique shifts that are possible. For example, if the key has a length of 10, there can be at most 10 unique shifts used to compute the ciphertext.

Our approach is as follows. Using each of the candidate plaintexts, we can compute the number of unique shifts that are required to go from that plaintext to the challenge ciphertext. If the candidate plaintext was not the plaintext used in the encryption, the generated key stream is random, and we would expect (with overwhelming probability) that all 27 unique shift values appear in that key stream. Only the plaintext that was actually used will generate a keystream with at most t unique shift values. Since t is at most 24, we determine that the plaintext with at most 24 unique shift values is the correct plaintext. If no plaintext fits this condition, we determine that none of test 1 candidates were used, and proceed to apply the test 2 solution.

Formally, we have the following algorithm. Note that UniqueShifts() returns the number of unique keyshifts required to for the transformation $candidate_i \mapsto c$ where c is the given ciphertext.

```
Data: candidates[1..$u$], ciphertext[1..$L$]
Result: imin, candidate with minimum unique key shifts
for $i = 1$ to $u$ do
    shiftAmounts[i] ← uniqueShifts(candidates[i], ctext);
end
min = uniqueShifts[0];
imin = 1;
for $i = 2$ to $u$ do
    if $shiftAmounts[i] < min$ then
        min = shiftAmounts[i];
        imin = i;
    end
end
```

Algorithm 1: Finding the Candidate Plaintext that Requires Least Unique Shifts

```
Function UniqueShifts(ciphertext, candidate):
    shifts[1...27] ← all zeros;
    for $i = 1$ to $L$ do
        shift = ciphertext[i]-candidate[i]  (mod 27);
        shifts[shift]++;
    end
    shifts = 0;
    for $i = 1$ to $L$ do
        if $shifts[i] > 0$ then
            shifts++;
        end
    end
    return shifts
end
```

Algorithm 2: Calculates the number of unique shifts required to go from candidate to ciphertext

4 Solution for Test 2

For this test, we use a depth first search to find appropriate candidates. We begin by assuming that the key length is of length t. This is necessary to rule out candidate plaintexts, which speeds up the search process and allows us to be more selective in the plaintext that we propose.

We begin by selecting all candidate plaintexts with one word. By definition there are v (which is set to 40), possible plaintexts of this length. We continue to extend each of these candidate plaintexts until the number of unique shifts exceeds the length of the expected key. This technique only works for sufficient small keys (i.e. 18 or less), otherwise the space of possible candidates is to large to search within a 3 minute time frame.

As the key length becomes larger, there are more candidates that can be suggested, and these are mostly the same aside from a different in a couple of words. If these were human readable messages, a human would most likely be able to determine which one makes the most sense. It would also be possible to determine fitness by similarity to English text, determined, for example, by the fraction of words that are English words and fraction of characters that are English characters. More sophisticated approaches might consider syntax and conceptual continuity. In our case, we only report the first candidate that fits the criteria, but in general we could print all possible messages.

```
Function DepthFirstSearch ($t$, $ciphertext$, $candidate$):
    for $i = 1$ to $v$ do
        append dictionary[i] to candidate;
        append space to candidate;
        u = UniqueShifts(ciphertext, candidate)
        if $u <= t$ then
            if $len(candidate) = len(ciphertext)$ then
                return $candidate$;
            else
                c = DepthFirstSearch($t$, ciphertext, candidate);
                if $c$ then
                    return $c$;
                end
            end
        end
    end
    /* no candidates found in this path                                   */
    return $false$;
```

Algorithm 3: DFS to Scan Plaintext Space for Viable Candidates

5 Conclusion

Over the course of this project, we tried many approaches of varying complexity. We attempted, for example, a hill climbing approach inspired by a general cryptanalyisis of homophonic ciphers. Unfortunately, though, these approaches did not turn out successfully. Also, classical approaches to finding the key length for a shift cipher, such as Kasiski Analysis and Index of Coincidence do not work since the key stream is not necessarily periodic.

We thus relied on simpler, but, in our view, more reliable cryptanalysis methods derived from an analysis of unique key shifts (Test 1) and clever plaintext search (Test 2). For keys lengths ≤ 15 our cryptanalysis meets the running time benchmarks set out in the project description.

Overall, we found the cipher to be quite robust. The secrecy of the scheduling algorithm, which is subject to change, adds additional entropy to the cipher. The scheduling algorithm produces a keystream as long as the the message. The only weakness of the encryption scheme is thus that the values in the key stream are limited to the t values in the key. Ultimately, this extra security is only achieved by obscurity and violates Kerckhoff's principle. In practice, the scheduling algorithm would be fixed and eventually leaked which would give the attacker even more statistical properties to exploit.

This was a fun and edifying project to work on!