Cracking Matrix Modes of Operation with Goodness-of-Fit Statistics

George Teşeleanu
Advanced Technologies Institute
10 Dinu Vintilă, Bucharest, Romania
tgeorge@dcti.ro
and Simion Stoilow Institute of Mathematics of the Romanian Academy
21 Calea Grivitei, Bucharest, Romania

Abstract

The Hill cipher is a classical polyalphabetical cipher based on matrices. Although known plaintext attacks for the Hill cipher have been known for almost a century, feasible ciphertext only attacks have been developed only about ten years ago and for small matrix dimensions. In this paper, we extend the ciphertext only attacks against the Hill cipher in two ways. First, we describe an attack against the affine version of the Hill cipher. Secondly, we show how to extend the (affine) Hill attack to several modes of operations. We also provide the reader with several experimental results and show how the message's language can influence the presented attacks.

1 Introduction

Two classical ciphers based on linear algebra are the Hill cipher (Hill, 1929) and its affine version (Hill, 1931). Both use invertible matrices over integers modulo a to encipher messages, where a is the size of the language alphabet $\mathscr{A}$. The first step of the encryption process is the encoding of each plaintext letter into a numerical equivalent. The simplest encoding is "a" $= 0$, "b" $= 1$ and so on. After encoding, the plaintext is divided into blocks of size λ and, then, each block is multiplied with an invertible matrix of size λ. In the affine case, a second matrix is added to the result. After each block is transformed, the result is converted back into letters. To decipher messages, one must perform the above steps in reverse.

Although both ciphers are vulnerable to known plaintext attacks[1], efficient ciphertext only attacks have been developed only a decade ago (Bauer and Millward, 2007) and only for the Hill cipher[2] with small λs. Note that as λ increases simple brute force attacks fail. For example, in the case of the Hill cipher with $a = 26$, we have around 2^{17} keys for $\lambda = 2$, 2^{40} keys for $\lambda = 3$ and 2^{73} keys for $\lambda = 4$ (Bauer and Millward, 2007). According to (Overbey et al., 2005; Bauer, 2002), given a and λ the exact number of invertible matrices can be computed. Note that in the case of the affine Hill cipher the computational effort made to brute force the Hill cipher is multiplied with a^{λ}.

In 2007, Bauer and Millward (Bauer and Millward, 2007) introduced a ciphertext only attack for the Hill cipher[3], that was later improved in (Yum and Lee, 2009; Leap et al., 2016; McDevitt et al., 2018). The attack was independently published by Khazaei and Ahmadi (Khazaei and Ahmadi, 2017). The main idea of these attacks is to do a brute force attack on the key rows, instead of the whole matrix, and then recover the decryption matrix.

In (Kiele, 1990), Kiele suggests the usage of block-chaining procedures to complicate the algebraic cryptanalytic techniques developed for the Hill cipher. We will show in this paper how to adapt the attacks described in (Bauer and Millward, 2007; Yum and Lee, 2009; Khazaei and Ahmadi, 2017) to different modes of operation (not only the block-chaining one) for both the Hill cipher and its affine version. Note that some modes do not require the key to be invertible, thus the attack presented in (Leap et al., 2016) does not work for all Hill based modes. For uniformity, we will only extend Yum and Lee's attack and leave as future work the extension of (Leap et al., 2016) to modes requiring invertible matrices. We stress that

[1] *i.e.* after a number of known messages are encrypted, one can easily recover the encryption key(s) if he has access to the corresponding ciphertexts.

[2] To the authors' knowledge no attack against the affine Hill cipher has been published.

[3] Bauer and Millward's attack for $\lambda = 3$ was previously and independently described online by Wutka (Wutka,).

out of the three attacks (Bauer and Millward, 2007; Yum and Lee, 2009; Khazaei and Ahmadi, 2017) Yum and Lee's attack has the best performance to message recovery ratio.

Another paper that motivated this study is (Bauer et al., 2016). The authors of (Bauer et al., 2016) conjecture that the fourth cryptogram of the Kryptos sculpture (kry, 2020) is either encrypted using the affine Hill cipher or some other sort of cipher mode of operation. We provide the reader with a preliminary study of these conjectures. To prove or disprove these conjectures, one has to find a way to adapt all the presented ciphertext attacks to the secret encoding versions of the (affine) Hill cipher and their corresponding modes of operation. Various partial answers for the secret encoding Hill cipher are provided in (Yum and Lee, 2009).

Structure of the paper. Notations and definitions are presented in Section 2. The core of the paper consists of two parts, Sections 3 and 4, that contain several key ranking functions and ciphertext only attacks. Experimental results are provided in Section 5. We conclude in Section 6. The letter frequencies use in our attacks are given in Appendix A.

2 Preliminaries

Notations. Throughout the paper, λ will denote a security parameter. We use the notation $x \xleftarrow{\$} X$ when selecting a random element x from a sample space X. We also denote by $x \leftarrow y$ the assignment of value y to variable x. The subset $\{0,\ldots,q-1\} \in \mathbb{N}$ will be referred to as $[0,q]$. The set of matrices with α rows, β columns and entries from $\mathbb{G}$ is denoted by $M(\alpha,\beta,\mathbb{G})$, the set of invertible matrices by $GL(\alpha,\mathbb{G})$ and the transpose of matrix A by A^T. The number of letters in a string m is represented by $|m|$ and the set of all strings by $\mathscr{A}^{\times}$.

In this paper we use some C++ language operators (*i*.e. $==$ for equality testing, $+=$, $*=$ as compound assignment operators, $++$ for incrementing a variable and & as reference to a variable) as well as some native function (*i*.e. $size()$ for returning the size of the object, $substring(pos,npos)$ for returning a substring starting from pos and containing $npos$ characters, $push\_back(val)$ to add val at the end of a vector and $sort$ to sort a vector in descending order). For initializing all the entries of a vector vec with a value val we use the notation $vec \leftarrow \{val\}$. When presenting algorithms, we consider only lower case messages represented by ASCII codes (*i*.e. "c" $-$ "a" $= 99-97 = 2$).

Conventions. To minimize repetitions, we employ the following system. When reading the attacks against the Hill based modes of operation we invite the reader to ignore red colored text, while in the case of the affine Hill based modes, the blue text. Also, when describing algorithms, we prefer using verbose names for variables, while, for mathematical descriptions, we prefer notations. The last convention used is to store constants in lookup tables when their size is small (*e.g.* letter frequencies) and in maps, otherwise (*e.g.* quadgraph frequencies).

2.1 Ciphers

A cipher consists of three probabilistic polynomial-time algorithms: *Setup*, *Encrypt* and *Decrypt*. The first one takes as input a security parameter and outputs the secret key. The secret key together with the *Encrypt* algorithm are used to encrypt a message m. The last algorithm decrypts any message encrypted using the known secret key.

Hill cipher. The Hill cipher is a polyalphabetical cipher based on linear algebra introduced by Lester S. Hill in (Hill, 1929). We briefly provide the algorithms for the Hill cipher.

Setup(λ): Choose $K_1 \xleftarrow{\$} GL(\lambda,\mathbb{Z}_a)$. Also, choose a public one-to-one function $convert : \mathscr{A}^{\times} \to \mathbb{Z}_a^{\times}$ and compute its inverse $unconvert : \mathbb{Z}_a^{\times} \to \mathscr{A}^{\times}$. Output the secret key is $sk = K_1$. Publish the *convert* and *unconvert* functions.

Encrypt(sk,m): Pad message m until $|m| \equiv 0 \bmod \lambda$[4]. Convert and divide m into blocks $convert(m) = m_1\|\ldots\|m_\ell$, where $|m_i| = \lambda$. Compute $c_i^T \leftarrow K_1 \cdot m_i^T$. Output the ciphertext $c = uncovert(c_1\|\ldots\|c_\ell)$.

Decrypt(sk,c): Divide $convert(c)$ into ℓ blocks and compute $m_i^T \leftarrow K_1^{-1} \cdot c_i^T$. Recover m by applying *uncovert* and removing the padding.

Affine Hill cipher. An affine variation of the Hill cipher was introduced in (Hill, 1931). We shortly provide the algorithms for the affine Hill cipher.

[4]Usually an uncommon letter, such as "x", is appended to m until we get the desired length.

Setup(λ): Choose $K_1 \xleftarrow{\$} GL(\lambda, \mathbb{Z}_a)$ and $K_2 \xleftarrow{\$} M(\lambda, 1, \mathbb{Z}_a)$. Also, choose a public one-to-one function $convert : \mathscr{A}^\times \to \mathbb{Z}_a^\times$ and compute its inverse $unconvert : \mathbb{Z}_a^\times \to \mathscr{A}^\times$. Output the secret key is $sk = (K_1, K_2)$. Publish the *convert* and *unconvert* functions.

Encrypt(sk, m): Pad message m until $|m| \equiv 0 \bmod \lambda$. Convert and divide m into blocks $convert(m) = m_1 \| \dots \| m_\ell$, where $|m_i| = \lambda$. Compute $c_i^T \leftarrow K_1 \cdot m_i^T + K_2$. Output the ciphertext $c = uncovert(c_1 \| \dots \| c_\ell)$.

Decrypt(sk, c): Divide $convert(c)$ into ℓ blocks and compute $m_i^T \leftarrow K_1^{-1} \cdot (c_i^T - K_2)$. Recover m by applying *uncovert* and removing the padding.

Affine variations. In Table 1 we present all the possible affine variations of the Hill cipher. Note that $K_3 \xleftarrow{\$} M(\lambda, 1, \mathbb{Z}_a)$. After performing some computations, we can see that all variations can be decrypted using the function $f(c_i) = K_1' \cdot c_i^T + K_2'$. Since we are interested only in recovering the encrypted messages and not the initial secret keys, all the presented attacks try to recover K_1' and K_2'. Thus, for the affine Hill cipher we consider f as the decryption function.

Encrypt	*Decrypt*	K_1'	K_2'
$c_i^T \leftarrow K_1 \cdot m_i^T + K_2$	$m_i^T \leftarrow K_1^{-1} \cdot (c_i^T - K_2)$	K_1^{-1}	$-K_1^{-1} K_2$
$c_i^T \leftarrow K_1 \cdot (m_i^T + K_2)$	$m_i^T \leftarrow K_1^{-1} \cdot c_i^T - K_2$	K_1^{-1}	$-K_2$
$c_i^T \leftarrow K_1 \cdot (m_i^T + K_2) + K_3$	$m_i^T \leftarrow K_1^{-1} \cdot (c_i^T - K_3) - K_2$	K_1^{-1}	$-K_1^{-1} K_3 - K_2$

Table 1: Affine variations of the Hill cipher.

2.2 Cipher Modes of Operation

When we encrypt messages block by block[5], equal blocks are mapped into equal ciphertexts. Thus, block patterns are preserved. In some cases, this leakage can lead to security concerns. To address this issue several cipher modes of operation where introduced (Dworkin, 2001): CBC, CTR, CFB and OFB.

In (Alagic and Russell, 2017), the authors introduce a generalization of the CBC-MAC construction[6]. Based on Alagic *e*t al.'s generalization, we present a possible adaptation of the CBC, CTR and CFB modes of operation to the (affine) Hill cipher. Note that the CFB and CTR modes do not require K_1 to be invertible.

Let $E_k, D_k : M(\lambda, \lambda, \mathbb{Z}_a) \to M(\lambda, \lambda, \mathbb{Z}_a)$ be the matrix transformations of the (affine) Hill cipher's encryption and decryption. We further describe the encryption and decryption algorithms for CBC and CFB.

Encrypt(sk, m): Choose $iv \xleftarrow{\$} M(1, \lambda, \mathbb{Z}_a)$ and pad message m until $|m| \equiv 0 \bmod \lambda$. Convert and divide m into blocks $convert(m) = m_1 \| \dots \| m_\ell$, where $|m_i| = \lambda$. Let $m_0 \leftarrow IV$. For CBC compute $c_i \leftarrow E_k(c_{i-1} + m_i)$, while for CFB compute $c_i \leftarrow E_k(c_{i-1}) + m_i$. Let $c = unconvert(c_1 \| \dots \| c_\ell)$. The output is ciphertext (iv, c).

Decrypt(sk, iv, c): Convert and divide c into ℓ blocks. For CBC compute $m_i \leftarrow D_k(c_i) - c_{i-1}$ and for CFB compute $m_i \leftarrow c_i - E_k(c_{i-1})$. Recover m by applying *unconvert* and removing the padding.

In the case of CTR, the sender and the receiver each keep a state $ctr \xleftarrow{\$} M(1, \lambda, \mathbb{Z}_a)$ that is updated before each encryption.

Update(ctr): Let $ctr^T = (\alpha_0, \dots, \alpha_{\lambda-1})$ and $i \leftarrow \lambda - 1$. Compute the following

1. $\alpha_i \leftarrow (\alpha_i + 1) \bmod a$,
2. If $\alpha_i == 0$, then $i \leftarrow (i-1) \bmod \lambda$ and go to step 1.

Encrypt(sk, m): Pad message m until $|m| \equiv 0 \bmod \lambda$. Convert and divide m into blocks $convert(m) = m_1 \| \dots \| m_\ell$, where $|m_i| = \lambda$. Compute $ctr \leftarrow Update(ctr)$ and $c_i \leftarrow E_k(ctr) + m_i$. The output is ciphertext $c = unconvert(c_1 \| \dots \| c_\ell)$.

Decrypt(sk, iv, c): Convert and divide c into ℓ blocks. Compute $ctr \leftarrow Update(ctr)$ and $m_i \leftarrow c_i - E_k(ctr)$. Recover m by applying *unconvert* and removing the padding.

A generalization of the OFB mode can also be derived. Unfortunately, our attacks do not apply to it. Thus, we omit OFB's description.

[5]ECB mode of operation

[6]the XOR operation is replaced with a generic group operation

2.3 Statistical Models

In order to rank[7] all possible rows for the decryption key, Yum and Lee (Yum and Lee, 2009) introduce a goodness-of-fit score function. Compared to the score functions presented in (Bauer and Millward, 2007; Khazaei and Ahmadi, 2017), Yum and Lee's function describes the exact probability of the recovered plaintext. We briefly describe the goodness-of-fit score function in Algorithm 1.

Let E_K and D_K be the encryption and, respectively, decryption function of a cipher. Also, let $c \leftarrow E_K(m)$ be the given cryptogram and K' the key we want to rank. The goodness-of-fit function takes as input the letter frequency table $letter\_freq$ associated with the language m is written in (see Appendix A for some examples) and the letter frequency table occ observed in $D_{K'}(c)$.

Algorithm 1. The goodness-of-fit score function.

Input: A vector of letter occurrences occ.
Output: occ's goodness-of-fit score scr.

Function $gof(letter\_freq, occ)$**:**
  $scr \leftarrow 1$;
  for $i \in [0, alph\_sz]$ **do**
    $scr *= letter\_freq[i]^{occ[i]} / occ[i]!$
  return scr;

To automatically separate meaningful messages from random texts, we use an approach similar with the ones described in (Hasinoff, ; Lyons, 2012). When testing a list of strings for meaning, we first score each of them using Algorithm 2 and, then, output the highest scoring message.

The first and second inputs of the score function are a string in and the block frequency map (in our case either a digraph di_freq or a quadgraph $quad\_freq$ frequency map) associated with the language we are interested in. The fourth variable $nb\_letters$ controls if we are observing digraphs (*i.e.* $nb\_letters = 2$) or quadgraph (*i.e.* $nb\_letters = 4$). When computing block frequency maps, some blocks may be missing entirely from the training corpus. To avoid assigning a likelihood of zero to these blocks, we use the *ad hoc* method found in (Lyons, 2012)[8].

To ease description, all frequency tables/maps will be implicit when presenting algorithms, unless otherwise specified.

Algorithm 2. The score function.

Input: A string in, the bound nb_rows.
Output: The string's score scr.

Function $scr\_fct(in, block\_freq, block\_def, nb\_letters)$**:**
  $scr \leftarrow 0$;
  for $i \in [0, in.size() - nb\_letters]$ **do**
    $temp \leftarrow in.substr(i, nb\_letters)$;
    if $temp \in block\_freq$ **then**
      $scr += block\_freq[temp]$;
    else
      $scr += block\_def$;
  return scr;

3 Ranking Functions

The first step in attacking the (affine) Hill cipher and the associated modes of operation is to rank all possible rows according to their relevance to a given cryptogram. In this section we describe the ranking functions latter used in the attacks presented in Section 4.

3.1 (Affine) ECB

In (Yum and Lee, 2009), the authors describe a ranking algorithm for the Hill cipher. We chose to present it in this section (Algorithm 3, red text) because it is tightly linked with the affine version we introduce (Algorithm 3, blue text).

Let $mat\_sz = \lambda = 2$ and let $enc = c$ be a Hill cipher cryptogram. We illustrate the influence of a given row on the decrypted plaintext p in Figure 1. We observe that if the first and second rows are equal we obtain the same letter p^i after decryption. Thus, is enough to decrypt the ciphertext using only the first row ($hill\_line\_dec$). Since we do not have duplicates, the resulting text msg is λ times shorter than c. After decryption we compute the letter frequency observed in msg and use the gof function to obtain the row's score. After all the rows have been ranked, we sort them in descending order according to their score. In the case of the affine Hill cipher the ranking algorithm is similar. The main difference is that instead of having to brute force k_0 and k_1, we also have to do an exhaustive search on k_2 (Figure 2). The algorithm for the generic case is given in Algorithm 3.

In some cases storing a vector of size a^{λ}[9] might be troublesome. Thus, we further consider that $fit.size() = B$, where B is dependent on the available memory. Note that in this case fit must be

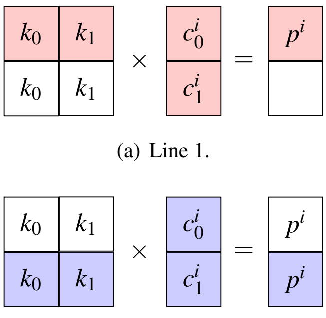
[7] according to their relevance to a given cryptogram

[8] *i.e.* $block\_def \leftarrow \log_{10}(0.01/nb\_blocks)$, where the total number of blocks found in the training corpus is denoted by nb_blocks

[9] $a^{\lambda+1}$ for the affine version

sorted and when an element is inserted we first check if its score is higher than the lowest score from fit and if it is, the element replaces the lowest scoring element from fit.

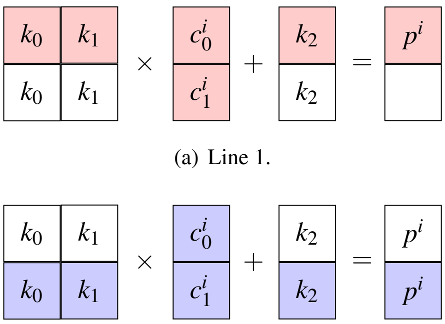
We usually work with small values of $alph\_sz$ and $msg.size()$ and thus we consider the complexity of the gof and of multiplication as $O(1)$. Hence, the Hill version of Algorithm 5 performs $O(a^{\lambda})$ $hill_line_dec$s and sorts a vector of size B. So, it has a complexity of $O(\lambda a^{\lambda} + B \log B)$. In the case of the affine Hill cipher, the only change is that we perform $O(a^{\lambda+1})$ $aff_hill_line_dec$s. So, the complexity becomes $O(\lambda a^{\lambda+1} + B \log B)$.



(a) Line 1.

(b) Line 2.

Figure 1: Line propagation in ECB.



(a) Line 1.

(b) Line 2.

Figure 2: Line propagation in affine ECB.

3.2 (Affine) CBC, CTR, CFB

Again, let $mat\_sz = 2$ and let enc be a Hill cipher cryptogram. The effect of a given row on the decrypted plaintext is shown in Figure 3 for CBC, in Figure 4 for CTR and in Figure 5 for CFB. Compared to ECB, we can easily see that if the first and second row are identical the resulting letters are different. Thus, we need the full decryption of the Hill cipher to rank rows. After decryption, we break the resulting msg in two parts msg_0 and

Algorithm 3. The algorithm for ranking all possible rows for (affine) ECB.

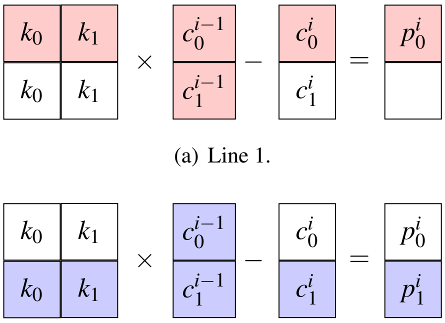
```
Input: The ciphertext enc.
Output: A vector fit containing all
        possible rows sorted by the
        goodness-of-fit score.
Function aff_hill_line_dec(conv, k1, k2):
    msg_int[enc.size()/mat_sz] ← {0};
    for i ∈ [0, conv.size()/mat_sz] do
        for j ∈ [0, mat_sz] do
            idx ← i · mat_sz + j;
            msg_int[i] ← (msg_int[i] +
              k1[j] · conv[idx]) mod alph_sz;
        msg_int[i] ← (msg_int[i] +
          k2[i mod mat_sz]) mod alph_sz;
    return msg_int;
Function aff_ecb_rank(enc):
    for
      k1[0], ..., k1[mat_sz − 1] ∈ [0, alph_sz]
      do
        for k2 ∈ [0, alph_sz] do
            occ[alph_sz] ← {0};
            conv ← convert(enc);
            msg_int ←
              hill_line_dec(enc, k1);
            msg_int ←
              aff_hill_line_dec(enc, k1, k2);
            msg ← unconvert(msg_int)
            for i ∈ [0, msg.size()] do
                occ[msg[i] − "a"]++;
            scr ← gof(letter_freq, occ);
            fit.push_back((k1, scr));
            fit.push_back((k1, k2, scr));
    fit.sort();
    return fit;
```

msg_1. The first part contains the letters in even positions and the second one the letters in odd positions. After we score each part, we store them in $fit[0]$ and, respectively, $fit[1]$. The last step is to sort the two vectors in descending order by score. The case of the affine Hill cipher is similar.

For the Hill modes attack, we perform $O(a^{\lambda})$ decryptions, while for the affine version the number of decryptions is $O(a^{\lambda+1})$. Both algorithms sort λ vectors of size B. Thus, the complexities are $O(\lambda^2 a^{\lambda} + \lambda B \log B)$ and $O(\lambda^2 a^{\lambda+1} + \lambda B \log B)$ for the Hill attack and, respectively, for the affine attack.

4 Message Recovering Attacks

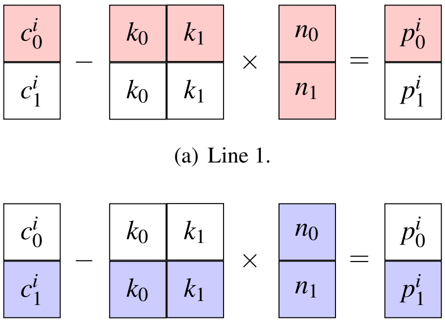
After the ranking step is over, we can proceed to the recovering step. When searching for the original message a lot of random text is produced. To filter random messages from ones with meaning we use the scr_fct to score each message and we always output the highest scoring one.



(a) Line 1.

(b) Line 2.

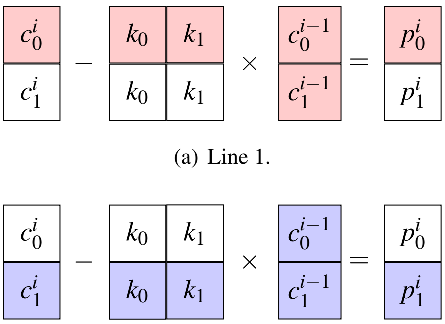
Figure 3: Line propagation in CBC.



(a) Line 1.

(b) Line 2.

Figure 4: Line propagation in CTR.



(a) Line 1.

(b) Line 2.

Figure 5: Line propagation in CFB.

4.1 (Affine) ECB

The authors of (Bauer and Millward, 2007; Yum and Lee, 2009) describe the message recovering algorithm for the Hill cipher, but they do not provide an automatic detection method for the original message. On the other hand, the authors of (Khazaei and Ahmadi, 2017) trade-off success probability for an unique output. The gap is filled in (Leap et al., 2016). We present the algorithm in this section (Algorithm 5, red text), instead of Section 2, because of its link to the affine version we introduce (Algorithm 5, blue text). Due to better results in practice, in Algorithm 5 we use a different scoring function[10] than the one from (Leap et al., 2016)[11]. Also, compared to (Leap et al., 2016), we only output the highest scoring message without lowering the success probability.

Algorithm 4. The algorithm for ranking all possible rows for (affine) CBC, CTR, CFB.

```
Input: The ciphertext enc and the
       initialization vector iv.
Output: A family of vectors fit containing
        all possible rows sorted by the
        goodness-of-fit score.
Function aff_mode_rank(enc, iv):
    for a[0], ..., a[mat_sz − 1] ∈ [0, alph_sz]
     do
        for b ∈ [0, alph_sz] do
            occ[mat_sz][alph_sz] ← {0};
            for i ∈ [0, mat_sz] do
                for j ∈ [0, mat_sz] do
                    k1[i][j] ← a[j];
                k2[i] ← b;
            conv ← convert(enc);
            msg_int ← mode_dec(enc,
             iv, k1);
            msg_int ← aff_mode_dec(enc,
             iv, k1, k2);
            msg ← unconvert(msg_int)
            for i ∈ [0, msg.size()/mat_sz]
             do
                for j ∈ [0, mat_sz] do
                    occ[j][msg[i ·
                      mat_sz + j] − "a"]++;
            for i ∈ [0, mat_sz] do
                scr ←
                 gof(letter_freq, occ[i]);
                fit[i].push_back((a, scr));
                fit[i].push_back((a, b, scr));
    for i ∈ [0, mat_sz] do
        fit[i].sort();
    return fit;
```

After ranking all possible rows, we need to find the decryption key's rows (*ck_vars*) and their order (*ck_var*). Thus, Algorithm 5 checks all possible row combinations with index less than $nb\_rows = B$. Note that the success probability is dependent on *nb_rows*[12]. After selecting λ rows from *fit*, we test all possible row permutations[13], decrypt *enc* and rank the result. If one of the decrypted texts has a higher score than the stored message *glb_msg*, we overwrite *glb_msg* and up-

[10] based on quadgraphs

[11] based on the index of coincidence

[12] see Section 5 for the experimental results

[13] σ_i denotes the *i*th permutation of length *mat_size*

date *glb_scr*. The main differences between the Hill cipher attack and the affine Hill cipher attack are: the call to the affine ranking algorithm, the creation of k_2 and the call to the affine decryption algorithm.

Algorithm 5. The algorithm for breaking (affine) ECB.

```
Input: The ciphertext enc, the bound nb_rows.
Output: The best possible message glb_msg and its associated score glb_scr.
Function ck_var(enc, rows, &glb_scr, &glb_msg):
    best_scr ← −∞;
    for i ∈ [0, mat_sz!] do
        for s ∈ [0, mat_sz] do
            for t ∈ [0, mat_sz] do
                k1[s][t] ← rows[σi[s]].k1[t];
            k2[s] ← rows[σi[s]].k2;
        try_msg ← hill_dec(enc, k1);
        try_msg ← aff_hill_dec(enc, k1, k2);
        try_scr ← scr_fct(try_msg, quad_freq, quad_freq, 4);
        if try_scr > best_scr then
            best_scr ← try_scr;
            best_msg ← try_msg;
    if best_scr > glb_scr then
        glb_scr ← best_scr;
        glb_msg ← best_msg;
Function ck_vars(enc, fit, nb_rows):
    glb_scr ← −∞;
    glb_msg ← "";
    for i0 ∈ [0, nb_rows] do
        for i1 ∈ [i0 + 1, nb_rows] do
            ...
            for i_{mat_sz−1} ∈ [i_{mat_sz−2} + 1, nb_rows] do
                try_rows ← ∅;
                for j ∈ [0, mat_sz] do
                    try_rows.push_back(fit[i_j]);
                ck_var(enc, try_rows, glb_scr, glb_msg);
    return (glb_scr, glb_msg);
Function aff_ecb_attack(enc, nb_rows):
    fit ← aff_ecb_rank(enc);
    return ck_var(enc, fit, nb_rows);
```

For the same reasons as in Section 3.1, we further consider the complexity of the *scr_fct* as $O(1)$. After the row ranking step, both message recovering algorithms perform $O(B!/(B-\lambda)!)$ decryptions. Thus, the complexities for the Hill attack and for the affine attack are $O(\lambda a^{\lambda} + B\log B + \lambda^2 B!/(B-\lambda)!)$ and, respectively, $O(\lambda a^{\lambda+1} + B\log B + \lambda^2 B!/(B-\lambda)!)$.

4.2 (Affine) CBC, CTR, CFB

The main difference between ECB and the other modes is that after the ranking step is over, in the former case we know the exact position of the key rows. Thus, in Algorithm 6 we iterate over all rows (*ck_vars_mode*), decrypt the cryptogram and then score the result (*ck_var_mode*).

The *ck_vars_mode* function performs $O(B^{\lambda})$ decryptions. Thus, Algorithm 6's complexity for the Hill based modes attack and for the affine versions is $O(\lambda^2 a^{\lambda} + \lambda B\log B + \lambda^2 B^{\lambda})$ and, respectively, $O(\lambda^2 a^{\lambda+1} + \lambda B\log B + \lambda^2 B^{\lambda})$.

Algorithm 6. The algorithm for breaking (affine) CBC, CTR, CFB.

```
Input: The ciphertext enc, the initialization vector iv, the bound nb_rows.
Output: The best possible message glb_msg and its associated score glb_scr.
Function ck_var_mode(enc, iv, rows, &glb_scr, &glb_msg):
    for s ∈ [0, mat_sz] do
        for t ∈ [0, mat_sz] do
            k1[s][t] ← rows[s].a[t];
        k2[s] ← rows[s].b;
    try_msg ← mode_dec(enc, iv, k1);
    try_msg ← aff_mode_dec(enc, iv, k1, k2);
    try_scr ← scr_fct(try_msg, quad_freq, quad_freqault, 4);
    if try_scr > glb_scr then
        glb_scr ← try_scr;
        glb_msg ← try_msg;
Function ck_vars_mode(enc, fit, nb_rows):
    glb_scr ← −∞;
    glb_msg ← "";
    for i0 ∈ [0, nb_rows] do
        for i1 ∈ [0, nb_rows] do
            ...
            for i_{mat_sz−1} ∈ [0, nb_rows] do
                try_rows ← ∅;
                for j ∈ [0, mat_sz] do
                    try_rows.push_back(fit[j][i_j]);
                ck_var_mode(enc, iv, try_rows, glb_scr, glb_msg);
    return (glb_scr, glb_msg);
Function aff_mode_attack(enc, nb_rows):
    fit ← aff_mode_rank(enc, iv);
    return ck_vars_mode(enc, iv, fit, nb_rows);
```

5 Experimental Results

We implemented Algorithms 5 and 6 in order to see the relation between B and the algorithms' success probability. The results are presented in Tables 3 to 8. To see the influence of the message's native language on the attack algorithms' recovery rate, we tested this type of relation for eight languages: Danish (DN), English (EN), Finnish (FN), French (FR), German (GE), Polish (PL), Spanish (SP) and Swedish (SW). We also computed the running time of Algorithms 5 and 6 for English and $\lambda = 2$ (Section 5.2).

In our implementations, frequency tables have $a = 26$ values and are derived from the frequencies provided in (Lyons, 2012). For completeness, we describe the tables in Appendix A. The quadgrams for the English language are downloaded from (Lyons, 2012), while the digraph[14] frequencies are computed from the quadgraph map.

For computing the success probability we used 100 texts with 100 letters (without diacritical marks) for each language. Each text was encrypted with a different key(s)/initialization vector/counter. The texts are taken from news items found in the Leipzig Corpora Collection (Goldhahn et al., 2012). The keys, initialization vectors and counters are generated using the default generator found in the GMP library (gmp,). When invertible keys were needed, we computed the inverse using the Armadillo library (Sanderson and Curtin, 2016) and tested if the determinant is coprime with 26.

5.1 Unicity Distance of a Cipher

When analyzing the experimental results, the reader will observe different message recovery rates for different languages. These differences arise from distinct unicity distances[15] for different languages. The exact formula for the unicity distance when $a = 26$ is $\log_2 26^\lambda / (\log_2 26 - H)$, where H is the language's entropy. Note that in our case the unicity distance is computed for one key row and we estimated the entropy from the frequency tables provided in Appendix A. The results for the unicity distance are provided in Table 2. We can see that in the case of the Polish language we need more letters per row than for the Finnish language. This gap will be more pronounced when determining the message recovery rates.

[14] If *abcd* is a quadgraph, we consider *ac* as a digraph.

[15] The minimum ciphertext length required to determine the secret key almost uniquely.

Language	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$
Danish	15.4323	23.1485	30.8647
English	18.2180	27.3270	36.4359
Finnish	12.0307	18.0460	24.0614
French	13.3713	20.0569	26.7425
German	15.6257	23.4386	31.2515
Polish	22.3918	33.5878	44.7837
Spanish	13.7891	20.6836	27.5781
Swedish	16.4837	24.7256	32.9674

Table 2: Unicity distance.

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	2	94	93	100	96	95	84	96	95
	4	99	100	100	98	100	91	100	100
CBC	1	95	95	100	99	97	84	99	99
	2	99	99	100	100	100	90	100	100
CTR	1	96	93	100	96	98	87	100	98
	2	99	98	100	99	100	90	100	100
CFB	1	97	92	99	96	95	87	98	98
	2	100	99	100	100	99	91	100	100

Table 3: Number of recovered messages for the Hill modes of operation when $\lambda = 2$.

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	8	88	59	97	90	71	22	87	80
	16	95	77	100	95	86	45	96	94
	32	97	87	100	98	94	68	99	99
CBC	4	86	57	99	92	71	18	91	78
	8	93	68	99	96	80	34	96	86
	16	96	80	100	96	89	55	97	96
CTR	4	64	40	84	65	46	11	68	45
	8	80	59	94	87	67	19	83	66
	16	91	75	97	93	80	48	92	77
CFB	4	85	53	99	90	73	12	89	78
	8	93	66	99	94	81	36	94	87
	16	96	79	100	97	91	52	96	96

Table 4: Number of recovered messages for the Hill modes of operation when $\lambda = 3$.

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	512	78	48	97	89	72	10	85	74
	1024	88	65	98	91	89	19	94	86
	2048	95	80	99	95	94	39	95	93
CBC	32	78	50	97	89	69	13	88	72
	64	87	67	99	91	86	21	93	84
	128	93	78	99	95	94	45	95	93
CTR	32	71	37	91	77	55	6	80	64
	64	87	58	97	90	79	21	90	83
	128	93	75	100	95	94	40	99	88
CFB	32	78	48	97	88	69	14	86	73
	64	87	65	98	91	85	18	92	85
	128	93	75	99	95	95	45	94	95

Table 5: Number of recovered messages for the Hill modes of operation when $\lambda = 4$.

5.2 Running time

In this section we provide some benchmarks for Algorithms 5 and 6. The algorithms were run

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	2	89	80	100	90	88	54	93	92
	4	97	94	100	98	99	79	98	99
	8	99	99	100	99	99	87	99	100
CBC	1	93	85	100	99	85	57	96	93
	2	97	88	100	99	93	68	98	100
	4	99	95	100	99	99	78	100	100
CTR	1	92	72	100	93	90	48	96	95
	2	97	88	100	96	98	68	99	99
	4	98	97	100	99	99	78	100	100
CFB	1	89	80	100	95	91	54	98	93
	2	97	92	100	98	97	69	100	99
	4	99	97	100	99	99	83	100	100

Table 6: Number of recovered messages for the affine Hill modes of operation when $\lambda = 2$.

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	32	70	43	97	86	49	3	85	63
	64	84	50	99	91	62	11	87	75
	128	93	65	99	93	79	21	94	88
CBC	32	71	40	98	86	47	5	83	61
	64	82	50	99	93	65	11	90	74
	128	90	65	99	93	78	25	95	97
CTR	32	35	13	56	40	19	3	37	18
	64	58	28	85	63	36	6	60	45
	128	81	49	98	82	59	13	83	77
CFB	32	70	38	97	87	50	3	83	74
	64	84	49	99	93	64	8	89	86
	128	91	63	99	93	77	23	94	96

Table 7: Number of recovered messages for the affine Hill modes of operation when $\lambda = 3$.

	B	DN	EN	FN	FR	GE	PL	SP	SW
ECB	16384	82	53	98	90	79	14	89	79
	32768	92	69	99	93	93	26	94	88
	65536	96	83	100	95	95	54	96	94
CBC	16384	80	53	98	89	76	14	88	78
	32768	89	69	99	93	92	27	94	87
	65536	96	80	100	95	95	61	96	93
CTR	16384	77	46	95	86	63	11	86	74
	32768	87	66	98	92	89	26	92	85
	65536	95	79	100	97	95	53	96	92
CFB	16384	81	53	98	89	76	15	88	77
	32768	90	68	99	93	92	27	94	87
	65536	96	81	100	95	95	59	96	93

Table 8: Number of recovered messages for the affine Hill modes of operation when $\lambda = 4$.

on a CPU Intel i7-4790 4.00 GHz and compiled with GCC with the O3 flag activated and the *omp_get_wtime()* function (omp,) was used to compute the running times. Due to resource constraints, we stopped the experiments at $\lambda = 3$ for the Hill attacks and at $\lambda = 2$ for the affine attacks. To obtain a fair comparison, when computing the running times, we used higher B values than the one presented in Tables 3 to 8. We present the exact margins in Table 9.

Mode	Hill ($\lambda = 2$)	Afine Hill ($\lambda = 2$)	Hill ($\lambda = 3$)
ECB	4 (100%)	8 (99%)	128 (97%)
CBC	2 (99%)	4 (95%)	128 (95%)
CTR	2 (98%)	4 (97%)	128 (96%)
CFB	2 (99%)	4 (97%)	128 (96%)

Table 9: The threshold B and the corresponding success probability for the English language.

In Table 10, the second and third columns contain the total time necessary to recover 100 independent texts, while the fourth column contains the total time necessary to recover 8 texts.

Mode	Hill ($\lambda = 2$)	Afine Hill ($\lambda = 2$)	Hill ($\lambda = 3$)
ECB	0.94057	23.1658	1415.60
CBC	1.75324	45.4769	1502.20
CTR	1.75827	45.9883	1423.39
CFB	1.75271	48.5864	1509.62

Table 10: Running times of Algorithms 5 and 6.

Let $\lambda = 2$. To see if the chosen bounds have the same success rate for other texts, we encrypted 1000 independent texts[16] and then we ran Algorithms 5 and 6. The number of plaintexts recovered is presented in Table 11. We can see that for the Hill based modes the success probabilities are almost the same, while for the affine versions the probabilities are a little lower than the initial estimates.

Cipher	ECB	CBC	CTR	CFB
Hill	995	987	982	982
Affine Hill	970	956	945	953

Table 11: Success rates for Algorithms 5 and 6 when $\lambda = 2$.

6 Conclusions

In this paper we adapted Yum and Lee's attack to the affine Hill cipher. Also, we introduced new ranking and message recovery algorithms for the CBC, CTR and CFB modes of operation. We also conducted a series of experiments to determine and test the success rates of these algorithms.

Future Work. The row ranking algorithms perform the same instructions for disjoint rows. Thus,

[16]different from the 100 texts used for computing the bounds

an interesting implementation direction is to parallelize Algorithms 3 and 4. The recovering algorithms also perform the same instructions, but for independent keys. Hence, Algorithms 5 and 6 can also be parallelized.

Another possible speed-up is to parallelize the algorithm presented (Leap et al., 2016) for the Hill cipher. Note that this speed-up can also be applied to the Hill CBC mode. From a theoretical point of view, it would be interesting to see if the Leap *et.al.*'s algorithm can be tweaked to work for the affine Hill cipher. If it can be tweaked we might obtain faster decryption times for the affine Hill and the corresponding CBC mode.

A time-memory trade-off attack for the Hill cipher is presented in (McDevitt et al., 2018). Thus, it might be interesting to see if this attack can be adapted to the affine version and to the (affine) modes of operation versions. From an implementation point of view, it might worth seeing if McDevitt *et.al.*'s attack can be parallelized.

In (Yum and Lee, 2009), the authors provide a ranking algorithm when the *convert* and the *unconvert* functions are unknown, but they do not describe a message recovery algorithm. This cipher can be seen as a composition of a substitution cipher, a Hill cipher and a second substitution cipher. Note that the two substitution ciphers do not necessarily have the same key. A generic version of the secret coding cipher can be obtained by combining a generic Vigenère cipher[17], a Hill cipher and a second generic Vigenère cipher. Note that in this case Yum and Lee's ranking algorithm still works. Hence, another possible research direction is to find message recovery algorithms[18] for this generic cipher.

In (Hill, 1931), Hill introduces a variation of the affine Hill cipher in which the elements of the key matrix are matrices. Thus, an interesting problem is to study the impact of the message recovering algorithms on the version presented in (Hill, 1931).

References

[Alagic and Russell2017] Gorjan Alagic and Alexander Russell. 2017. Quantum-Secure Symmetric-Key Cryptography Based on Hidden Shifts. In *EUROCRYPT 2018*, volume 10212 of *Lecture Notes in Computer Science*, pages 65–93. Springer.

[17]By a generic Vigenère cipher we understand a Vigenère cipher with random alphabets.

[18]that might use Yum and Lee's ranking algorithm

[Bauer and Millward2007] Craig Bauer and Katherine Millward. 2007. Cracking Matrix Encryption Row by Row. *Cryptologia*, 31(1):76–83.

[Bauer et al.2016] Craig Bauer, Gregory Link, and Dante Molle. 2016. James Sanborns Kryptos and the Matrix Encryption Conjecture. *Cryptologia*, 40(6):541–552.

[Bauer2002] Friedrich Ludwig Bauer. 2002. *Decrypted Secrets: Methods and Maxims of Cryptology*. Springer.

[Dworkin2001] Morris Dworkin. 2001. Recommendation for Block Cipher Modes of Operation. Methods and Techniques. Technical report, NIST.

[gmp] The GNU Multiple Precision Arithmetic Library. `https://gmplib.org/`.

[Goldhahn et al.2012] Dirk Goldhahn, Thomas Eckart, and Uwe Quasthoff. 2012. Building Large Monolingual Dictionaries at the Leipzig Corpora Collection: From 100 to 200 Languages. In *LREC 2012*, volume 29, pages 31–43. European Language Resources Association (ELRA).

[Hasinoff] Sam Hasinoff. Solving Substitution Ciphers. `https://people.csail.mit.edu/hasinoff/pubs/hasinoff-quipster-2003.pdf`.

[Hill1929] Lester S Hill. 1929. Cryptography in an Algebraic Alphabet. *The American Mathematical Monthly*, 36(6):306–312.

[Hill1931] Lester S Hill. 1931. Concerning Certain Linear Transformation Apparatus of Cryptography. *The American Mathematical Monthly*, 38(3):135–154.

[Khazaei and Ahmadi2017] Shahram Khazaei and Siavash Ahmadi. 2017. Ciphertext-Only Attack on $d \times d$ Hill in $O(d13^d)$. *Information Processing Letters*, 118:25–29.

[Kiele1990] William A Kiele. 1990. A Tensor-Theoretic Enhancement to the Hill Cipher System. *Cryptologia*, 14(3):225–233.

[kry2020] 2020. Kryptos. `https://en.wikipedia.org/wiki/Kryptos`.

[Leap et al.2016] Tom Leap, Tim McDevitt, Kayla Novak, and Nicolette Siermine. 2016. Further Improvements to the Bauer-Millward Attack on the Hill Cipher. *Cryptologia*, 40(5):452–468.

[Lyons2012] James Lyons. 2012. Practical Cryptography, `http://practicalcryptography.com/`.

[McDevitt et al.2018] Tim McDevitt, Jessica Lehr, and Ting Gu. 2018. A Parallel Time-memory Tradeoff Attack on the Hill Cipher. *Cryptologia*, 42(5):1–19.

[omp] OpenMP. `https://www.openmp.org/`.

[Overbey et al.2005] Jeffrey Overbey, William Traves, and Jerzy Wojdylo. 2005. On the Keyspace of the Hill Cipher. *Cryptologia*, 29(1):59–72.

[Sanderson and Curtin2016] Conrad Sanderson and Ryan Curtin. 2016. Armadillo: A Template-Based C++ Library for Linear Algebra. *Journal of Open Source Software*, 1(2):26.

[Wutka] Mark Wutka. The Crypto Forum, http://s13.zetaboards.com/Crypto/topic/123721/1/.

[Yum and Lee2009] Dae Hyun Yum and Pil Joong Lee. 2009. Cracking Hill Ciphers with Goodness-of-Fit Statistics. *Cryptologia*, 33(4):335–342.

Appendix A Letter Frequencies

To have uniform letter frequency tables, we added the probability of letters with diacritical marks to the probability of their base letter. For example, in Danish, the letter O has a 0.0464 occurrence probability and the letter Ø one of 0.0094. We added the two and we recorded O's probability as 0.0558. Note that the frequency tables we used for computing our tables are from (Lyons, 2012).

A, Å, Æ	0.0809	J	0.0073	S	0.0581
B	0.0200	K	0.0339	T	0.0686
C	0.0056	L	0.0523	U	0.0198
D	0.0586	M	0.0324	V	0.0233
E	0.1545	N	0.0724	W	0.0007
F	0.0241	O, Ø	0.0558	X	0.0003
G	0.0408	P	0.0176	Y	0.0070
H	0.0162	Q	0.0001	Z	0.0003
I	0.0600	R	0.0896		

Table 12: Relative frequencies of Danish letters.

A	0.0855	J	0.0022	S	0.0673
B	0.0160	K	0.0081	T	0.0894
C	0.0316	L	0.0421	U	0.0268
D	0.0387	M	0.0253	V	0.0106
E	0.1210	N	0.0717	W	0.0183
F	0.0218	O	0.0747	X	0.0019
G	0.0209	P	0.0207	Y	0.0172
H	0.0496	Q	0.0010	Z	0.0011
I	0.0733	R	0.0633		

Table 13: Relative frequencies of English letters.

A, Ä	0.1580	J	0.0204	S	0.0786
B	0.0028	K	0.0497	T	0.0875
C	0.0028	L	0.0576	U	0.0501
D	0.0104	M	0.0320	V	0.0225
E	0.0797	N	0.0883	W	0.0009
F	0.0019	O, Ö	0.0605	X	0.0003
G	0.0039	P	0.0184	Y	0.0174
H	0.0185	Q	0.0001	Z	0.0005
I	0.1082	R	0.0287		

Table 14: Relative frequencies of Finnish letters.

A, À, Â	0.0808	J	0.0030	S	0.0798
B	0.0096	K	0.0016	T	0.0711
C, Ç	0.0344	L	0.0586	U, Ù	0.0559
D	0.0408	M	0.0278	Û, Ü	
E, È, É, Ê	0.1745	N	0.0732	V	0.0129
F	0.0112	O, Ô, Œ	0.0546	W	0.0008
G	0.0118	P	0.0298	X	0.0043
H	0.0093	Q	0.0085	Y	0.0034
I, Î, Ï	0.0726	R	0.0686	Z	0.0010

Table 15: Relative frequencies of French letters.

A, Ä	0.0688	J	0.0027	S, ß	0.0656
B	0.0221	K	0.0150	T	0.0643
C	0.0271	L	0.0372	U, Ü	0.0376
D	0.0492	M	0.0275	V	0.0094
E	0.1599	N	0.0959	W	0.0140
F	0.0180	O, Ö	0.0299	X	0.0007
G	0.0302	P	0.0106	Y	0.0013
H	0.0411	Q	0.0004	Z	0.0122
I	0.0760	R	0.0771		

Table 16: Relative frequencies of German letters.

A, Ą	0.0997	J	0.0226	S, Ś	0.0504
B	0.0139	K	0.0354	T	0.0394
C, Ć	0.0422	L, Ł	0.0418	U	0.0259
D	0.0323	M	0.0273	V	0.0000
E, Ę	0.0849	N, Ń	0.0602	W	0.0478
F	0.0041	O, Ó	0.0879	X	0.0000
G	0.0154	P	0.0292	Y	0.0370
H	0.0125	Q	0.0000	Z, Ź, Ż	0.0590
I	0.0809	R	0.0506		

Table 17: Relative frequencies of Polish letters.

A	0.1250	J	0.0045	S	0.0744
B	0.0127	K	0.0008	T	0.0442
C	0.0443	L	0.0584	U	0.0400
D	0.0514	M	0.0261	V	0.0098
E	0.1324	N, Ñ	0.0731	W	0.0003
F	0.0079	O	0.0898	X	0.0019
G	0.0117	P	0.0275	Y	0.0079
H	0.0081	Q	0.0083	Z	0.0042
I	0.0691	R	0.0662		

Table 18: Relative frequencies of Spanish letters.

A, Ä, Å	0.1252	J	0.0061	S	0.0659
B	0.0154	K	0.0314	T	0.0769
C	0.0149	L	0.0528	U	0.0192
D	0.0470	M	0.0347	V	0.0242
E	0.1015	N	0.0854	W	0.0014
F	0.0203	O, Ö	0.0579	X	0.0016
G	0.0286	P	0.0184	Y	0.0071
H	0.0209	Q	0.0002	Z	0.0007
I	0.0582	R	0.0843		

Table 19: Relative frequencies of Swedish letters.