A Not-Very-Secure System

Implementation and Analysis of Affine Hill Cipher Using MatLab

https://github.com/ziad-eliwa/Linear-Algebra-Project.git

Hazem Nasr, Mostafa Bahaa, and Ziad Eliwa
MACT 2132 Final Project: Summer 2025

Abstract. This paper presents an implementation and analysis of the Affine Hill Cipher, a classical cryptographic algorithm rooted in linear algebra and modular arithmetic. The primary aim is educational: to illustrate the mathematical foundations and practical steps involved in constructing and analyzing such ciphers, rather than to propose a secure encryption method for real-world use. Through detailed discussion of modular arithmetic, matrix operations, and key generation, the paper demonstrates how the Hill Cipher operates and explores its vulnerabilities. MATLAB code examples are provided to reinforce the concepts and facilitate hands-on learning. Section 1 introduces the preliminary knowledge to set the stage for section 2 which includes a detailed analysis and implementation of the cipher.

Acknowledgments. The style of the formatting of this document is accredited to Evan Chen and in particular, his Infinitely Large Napkin.

§1 Preliminaries

Almost all our future discussion is based on the basic notions of modular arithmetic. We start by describing these notions for integers and then extend the theory to matrices.

§1.1 Modular Arithmetic

Definition 1.1. Given an integer n and a positive integer m, we know by the division algorithm (refer to [2]) that there exist unique integers q and r such that $n = qm + r$. We call r the **remainder** of dividing n by m and denote it by $r = (n \bmod m)$.

The mod operator has some two properties described in the following theorem.

Theorem 1.2

Let a, b be integers and m be a positive integer. Then,

(i) $(a + b) \bmod m = (a \bmod m) + (b \bmod m)$.

(ii) $(ab) \bmod m = (a \bmod m)(a \bmod m)$.

Proof. Let $a = q_1 m + r_1$ and $b = q_2 m + r_2$ where q_1, q_2, r_1, r_2 are the quotients and remainders given by applying the division algorithm on a, m and b, m. Then,

(i) $a + b = q_1 m + r_1 + q_2 m + r_2 = (q_1 + q_2)m + (r_1 + r_2)$. But clearly, $0 \leq r_1 < a, 0 \leq r_2 < b \Rightarrow 0 \leq (r_1 + r_2) < a + b$ and none of r_1, r_2 divides $a + b$. Then, $(a + b) \bmod m = (r_1 + r_2) = (a \bmod m) + (b \bmod m)$.

(ii) $ab = (q_1 m + r_1)(q_2 m + r_2) = q_1 q_2 m^2 + q_1 r_2 m + q_2 r_1 m + r_1 r_2 = (q_1 q_2 m + q_1 r_2 + q_2 r_1)m + r_1 r_2$. But similar to the previous argument, $0 \leq r_1 r_2 < ab$ and does not divide ab. Therefore, $(ab) \bmod m = r_1 r_2 = (a \bmod m)(a \bmod m)$.

Putting the two arguments together completes the proof. □

We will also need the notion of congruence modulo m.

Definition 1.3. Let $a, b \in \mathbb{Z}$ and $m \in \mathbb{Z}^+$. We say that a is **congruent** to b modulo m and write $a \equiv b \pmod m$ if and only if $a - b = mc$ for some $c \in \mathbb{Z}$.

Theorem 1.4

Let a, b be integers and m be a positive integer. Then $a \equiv b \pmod m$ if and only if $a \bmod m = b \bmod m$.

Proof. ($\Longrightarrow$) Let $a \equiv b \pmod m$. By definition, $a - b = mc$ for some integer c. Reducing the equation modulo m we get: $(a - b) \bmod m = mc \bmod m \Rightarrow a \bmod m - b \bmod m = 0 \Rightarrow a \bmod m = b \bmod m$.
($\Longleftarrow$) We write $a = mq_a + r_a$ and $b = mq_b + r_b$ where $0 \leq r_a, r_b < m$. By definition, $a \bmod m = r_a$ and $b \bmod m = r_b$. Then, $a - b = (mq_a + r_a) - (mq_b + r_b) = (q_a - q_b)m + (r_a - r_b) = (q_a - q_b)m$. Therefore, $a - b$ is a multiple of m and $a \equiv b \pmod m$ by definition.
Putting the two directions together completes the proof. □

Now that we know how to add and multiply modulo m, we need to find the multiplicative inverse modulo m (if exists). For this, we will need the following algorithm.

Algorithm 1.5 (Extended Euclidean Algorithm)

Note: This is the recursive form of the algorithm.

1. **Input**: Two integers a, b.
2. **Expected Output**: A triple (d, x, y) such that $d = \gcd(a, b)$ and $ax + by = d$.
3. **Base Case**: When $b = 0$, $\gcd(a, 0) = |a|$. Return $(a, 1, 0)$.
4. **Recursive Step**: Reduce the problem from (a, b) to $(b, a \bmod b)$.

A detailed discussion of this algorithm (with a proof of validity) is provided in [4]. We just need the statement of the algorithm for our paper.

Lemma 1.6 (Bézout's identity)

Let a and b be integers with $\gcd(a, b) = d$. Then, there exist integers x and y such that $ax + by = d$.

Refer to [2] for the proof of this lemma and a more detailed discussion.

Theorem 1.7

Let m be a positive integer and let a be an integer. Then a is invertible modulo m (i.e., there exist some b with $ab \equiv 1 \pmod{m}$) if and only if $\gcd(a, m) = 1$. Moreover, if a is invertible, then a^{-1} can be calculated by running algorithm 1.5 on inputs a and m.

Proof. ($\Longrightarrow$) Assume a has a multiplicative inverse modulo m. That is, there exists some b such that $ab \equiv 1 \pmod{m}$. By definition of congruence, this means $ab - 1 = km$ for some $k \in \mathbb{Z}$. Rearrange: $ab - km = 1$. Notice that any common divisor of a and m must then also divide the left-hand side, hence divide 1. But the only positive integer that divides 1 is 1 itself, so $\gcd(a, m) = 1$.
($\Longleftarrow$) Assume $\gcd(a, m) = 1$. By lemma 1.6, there exist two integers x, y such that $ax + my = 1$. Rearrange: $ax - 1 = -my$. Then by definition of congruence again, $ax \equiv 1 \pmod{m}$. Thus, x is a multiplicative inverse of a modulo m. Equivalently, a is invertible modulo m.
Putting the two directions together, we have shown that

$$a \text{ is invertible mod } m \Longleftrightarrow \gcd(a, m) = 1.$$

□

§1.2 Basic Linear Algebra

In this section, we discuss three main ideas from basic linear algebra: matrix multiplication, the determinant function, and the adjoint matrix.

Matrix Multiplication

First, we define the product of two matrices as follows:

Definition 1.8 (Matrix Multiplication)**.** Let $A = [a_{ij}]$ be an $m \times k$ matrix and $B = [b_{ij}]$ be a $k \times n$ matrix. Their **product** AB is the $m \times n$ matrix whose (i, j)-entry is equal to the sum of the products of the corresponding entries from the i^{th} row of A and j^{th} column of B. In other words,

$$c_{ij} = \sum_{l=1}^{n} a_{il} b_{lj}.$$

Determinants

There are many equivalent ways to define the determinant function. We consider the one discussed in the course.

Abuse of Notation 1.9. In the following discussion, we use the subscript ij to represent two different things, the (i, j)-entry and the (i, j)-minor. We believe the distinction is clear from the context.

Definition 1.10 (Determinant of a Matrix)**.** Let A be an $n \times n$ square matrix. The **determinant** is a function from the set of square matrices to the set of real number denoted by $\det(A)$ or $|A|$. It is defined recursively as follows:

1. If $n = 1$ and $A = [a]$, define $|A| = a$.
2. For $n > 1$,
$$|A| = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} |A_{1j}|,$$

where A_{ij} is the $(n-1) \times (n-1)$ matrix obtained by deleting the i^{th} row and the j^{th} column of A.

Theorem 1.11

The determinant function is multiplicative. In other words,

$$\det(AB) = \det(A) \det(B)$$

for all matrices A, B of suitable sizes.

Refer to [3] for the proof (eliminated for space reasons).

Adjoint Matrix

Along with the determinant, we need the idea of the adjoint matrix to calculate matrix inverses.

Definition 1.12 (Adjoint of a Matrix)**.** Let A be a square matrix. The **adjoint** of A denoted by $\text{adj}(A)$ is the matrix whose (i, j)-entry is

$$(\text{adj}(A))_{ij} = (-1)^{i+j} \det(A_{ji})$$

where A_{ji} is the $(n-1) \times (n-1)$ matrix obtained from A by deleting the j-th row and the i-th column.

Theorem 1.13

Let A be a square matrix of size $n \times n$. Then,

$$A \text{ adj}(A) = \text{adj}(A)A = \det(A)I_n.$$

Proof. Let A be a square matrix of size $n \times n$. We prove that $A \text{ adj}(A) = \det(A)I_n$. We start by finding the (i, j)-entry of the left-hand side using the definition of matrix multiplication.

$$(A \text{ adj}(A))_{ij} = \sum_{k=1}^{n} a_{ik}[\text{adj}(A)]_{kj} = \sum_{k=1}^{n} a_{ik}C_{jk}.$$

But by the cofactor expansion of $\det(A)$ along row j,

$$\det(A) = \sum_{k=1}^{n} a_{jk}C_{jk}.$$

There are now two cases:

1. When $i = j$:
$$(A) \text{ adj}(A)_{jj} = \sum_{k=1}^{n} a_{jk}C_{jk} = \det(A).$$
2. When $i \neq j$: We are summing $a_{ik}C_{jk}$ for fixed $j \neq i$. But this is precisely the cofactor expansion of the determinant of the matrix obtained from A by replacing row j with row i. Since two rows of that matrix are equal, its determinant is zero. Hence,
$$(A \text{ adj}(A))_{ij} = 0.$$

Putting these together, we find that $A \text{ adj}(A)$ is a square matrix that has $\det(A)$ as the diagonal entries and 0 elsewhere. Thus, $A \text{ adj}(A) = \det(A)I_n$ as desired. A similar argument shows that $\text{adj}(A)A = \det(A)I_n$. □

§1.3 Matrices over $\mathbb{Z}_m$

Similar to integers, we develop notions of modular arithmetic but now on integer matrices (i.e., matrices with integral entries).

Definition 1.14. Let $A = [a_{ij}]$ be an integer matrix and m be a positive integer. We define A modulo m entry-wise. In other words, $A \bmod m = [a_{ij} \bmod m]$.

Definition 1.15 (Congruence of Matrices)**.** Let A and B be matrices of the same size and m be a positive integer. We say A is **congruent** to B modulo m and write $A \equiv B \pmod{m}$ if and only if $A - B = mC$ for some integer matrix C, or equivalently, $a_{ij} \equiv b_{ij} \pmod{m}$ for all i, j.

Definition 1.16 (Invertible Mod m)**.** Let A be a matrix of size $n \times b$ and m be a positive integer. We say that A is **invertible** modulo m if and only if there exists a matrix B of size $n \times n$ such that $AB \equiv I_n \pmod{m}$ and $BA \equiv I_n \pmod{m}$. We call B the inverse of A and denote it by $B = A^{-1}$.

We will soon show that if A^{-1} exists, then it is unique.

Lemma 1.17

Let A be a matrix, c be a real number, and m be a positive integer. Then,

$$(cA) \bmod m = (c \bmod m)(A \bmod m).$$

Proof. Let A be any integer matrix, let $c \in \mathbb{R}$, and let $m \in \mathbb{Z}^+$ be a positive integer. By the definition of scalar multiplication and modular arithmetic applied element-wise, we have:

$$cA = [ca_{ij} \bmod m], \quad \text{where } ca_{ij} \in \mathbb{R}.$$

Now, by the modular arithmetic identity:

$$ca_{ij} \bmod m = ((c \bmod m)(a_{ij} \bmod m)) \bmod m,$$

we get:

$$cA = [ca_{ij} \bmod m] = [(c \bmod m)(a_{ij} \bmod m) \bmod m].$$

This is equivalent to scalar multiplication of the matrix $A \bmod m$ by the scalar $c \bmod m$, so:

$$cA \bmod m = (c \bmod m)(A \bmod m).$$

Thus, the identity holds.

□

Lemma 1.18

Let A, B be matrices and m be a positive integer. Then,

$$(AB) \bmod m = (A \bmod m)(B \bmod m).$$

Proof. Let $A_{m\times k}, B_{k\times n}$ be any integer matrices, and let $m \in \mathbb{Z}^+$ be a positive integer.
L.H.S:

$$(AB) \bmod m = [c_{ij}], \quad \text{where } c_{ij} \in \mathbb{Z}.$$

Such that:

$$c_{ij} = \left(\sum_{l=1}^{k} a_{il}b_{lj}\right) \bmod m$$

Expand:

$$c_{ij} = (a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{ik}b_{kj}) \bmod m$$

Now, by theorem 1.2, we have:

$$c_{ij} = (a_{i1}b_{1j} \bmod m) + (a_{i2}b_{2j} \bmod m) + \cdots + (a_{ik}b_{kj} \bmod m)$$

R.H.S: We examine $(A \bmod m)(B \bmod m)$
We have:

$$A \bmod m = [a_{ij} \bmod m], \quad B \bmod m = [b_{ij} \bmod m]$$

Also:

$$(A \bmod m)(B \bmod m) = [d_{ij}], \quad \text{where } d_{ij} \in \mathbb{Z}.$$

Such that:

$$d_{ij} = \sum_{l=1}^{k} (a_{il} \bmod m)(b_{lj} \bmod m)$$

Expand:

$d_{ij} = (a_{i1} \bmod m)(b_{1j} \bmod m) + (a_{i2} \bmod m)(b_{2j} \bmod m) + \cdots + (a_{ik} \bmod m)(b_{kj} \bmod m)$

By theorem 1.2, again:

$$d_{ij} = (a_{i1}b_{1j} \bmod m) + (a_{i2}b_{2j} \bmod m) + \cdots + (a_{ik}b_{kj} \bmod m) = \textbf{L.H.S}$$

Thus, the identity holds.

□

Theorem 1.19

The determinant function from matrices over $\mathbb{Z}_m$ to the real numbers is multiplicative. In other words,

$$\det(AB) \equiv \det(A)\det(B) \pmod{m}$$

for any A and B over $\mathbb{Z}_m$ of suitable sizes.

Proof. Let $A_{m\times k}, B_{k\times n}$ be any matrices over $\mathbb{Z}$, and $m \in \mathbb{Z}^+$ be a positive integer. We know that the determinant of two matrices are multiplicative,

$$\det(AB) = \det(A)\det(B)$$

We apply the mod function to both sides:

$$(\det(AB)) \bmod m = (\det(A)\det(B)) \bmod m$$

Since the determinant is an integer, by theorem 1.4,

$$\det(AB) \equiv \det(A)\det(B) \pmod{m}$$

□

Theorem 1.20

Let A be a square matrix and m be a positive integer. Then, A is invertible modulo m if and only if $\det(A)$ is invertible modulo m. Moreover, if A is invertible, then

$$A^{-1} = ((\det(A))^{-1} \operatorname{adj}(A)) \bmod m.$$

Proof. ($\Longrightarrow$) Let $A_{n\times n}$ be invertible modulo m. By definition, there exists an integer matrix B such that $AB \equiv I_n \pmod{m}$. Then,

$$AB = I_n + mX$$

for some integer matrix X. Taking determinants of both sides and using theorem 1.19,

$$\det(A)\det(B) = \det(I_n + mX).$$

Now consider the right-hand side. By definition 1.10,

$$\det(I_n + mX) = \sum_{j=1}^{n} (\delta_{1j} + mx_{1j})C_{1j}$$

where $C_{1j} = (-1)^{1+j} \det\left((I_n + mX)_{1j}\right)$ and δ_{1j} is the $(1, j)$-entry of I_n. But:

- When $j = 1$, $\delta_{11} = 1$ and the minor $(I_n + mX)_{11}$ is itself of the form $I_{n-1} + mX'$ for some integer matrix X'. By induction, $C_{11} \equiv 1 \pmod{m}$.
- When $j > 0$, $\delta_{1j} = 0$. so each such term is $mx_{1j}C_{1j}$, which is divisible by m.

Hence,

$$\det(I_n + mX) = 1 \cdot (\text{an integer } \equiv 1 \pmod{m}) + m \cdot (\text{some integer}) \equiv 1 \pmod{m}.$$

Since $\det(A)\det(B) = \det(I_n + mX)$, $\det(A)\det(B) \equiv 1 \pmod{m}$, i.e., $\det(A)$ is invertible modulo m.

($\Longleftarrow$) Let $\det(A)$ be invertible modulo m. By theorem 1.13,

$$A \operatorname{adj}(A) = \det(A) I_n,$$

where $\operatorname{adj}(A)$ is the adjoint matrix of A. Reducing the equation modulo m by lemma 1.17 and lemma 1.18,

$$(A \bmod m)(\operatorname{adj}(A) \bmod m) = (\det(A) \bmod m) I_n.$$

By hypothesis, $\det(A)$ is invertible modulo m. Then, we can multiply both sides by $(\det(A))^{-1} \bmod m$ and get

$$A(\det(A))^{-1} \operatorname{adj}(A)) \equiv I_n \pmod{m}.$$

Therefore, A is left-invertible modulo m. A similar argument shows that A is also right-invertible and, consequently, invertible. Putting the two directions together completes the proof. □

§2 The Cipher

The **Hill Cipher** is a classical polygraphic substitution cipher developed by the American mathematician **Lester S. Hill** in **1929**. It was the first cipher to utilize concepts from **linear algebra** and **matrix operations** for encryption, marking a significant advancement in classical cryptographic techniques.
Unlike monoalphabetic ciphers that encrypt a single letter at a time, the Hill Cipher encrypts blocks of letters simultaneously by treating them as vectors and multiplying them by an **invertible matrix** modulo m, where m is typically 29 (corresponding to the number of letters in the English alphabet).
Hill's approach introduced the innovative use of invertible matrices as encryption keys, making the cipher more resistant to frequency analysis than earlier methods such as the Caesar or Vigenère ciphers. Although the Hill Cipher is no longer considered secure by modern standards, it remains an important tool for teaching the interplay between **cryptography** and **linear algebra**.

§2.1 The Bi-Operational Alphabet

The Hill cipher is based on the idea of mapping the English alphabet to the set

$$\mathbb{Z}_{29} = \{0, 1, 2, 3, \ldots, 28\},$$

where each letter corresponds to a unique integer modulo 29. Blocks of plaintext letters are then represented as vectors with entries in $\mathbb{Z}_{29}$. A key is chosen in the form of an invertible square matrix over $\mathbb{Z}_{29}$ of suitable dimension. This key matrix is used to transform the plaintext vectors into ciphertext vectors via matrix multiplication, effectively encrypting the message by converting each vector into another set of letters in $\mathbb{Z}_{29}$.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

q	r	s	t	u	v	w	x	y	z	$,$	$.$	*space*
16	17	18	19	20	21	22	23	24	25	26	27	28

Definition 2.1. Let $a_0, a_1, a_2, \ldots, a_{28}$ be letters in the English alphabet mapped to an integer i in $\mathbb{Z}_{29}$.

1. (Modular addition of letters modulo 29) Let $a_i + a_j = a_r$, such such that r is the remainder of dividing the integer $i + j$ by 29.
2. (Modular Multiplication of letters modulo 29) Let $a_i a_j = a_t$ such that t is the remainder of dividing the integer ij by 29.

Here is the mapping process we used implemented using MATLAB:

```
function asciistr = toascii(~, string)
    letters = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', ...
               'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', ...
               'u', 'v', 'w', 'x', 'y', 'z', '', ',', '.'};
    nums = 0:28;
    d = dictionary(letters, nums);
    string = lower(string);
    chars = cellstr(string(:));
    asciistr = d(chars);
end

function str = tochar(~, asciistr)
    nums = 0:28;
    letters = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', ...
               'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', ...
               'u', 'v', 'w', 'x', 'y', 'z', ' ', ',', '.'};
    d = dictionary(nums, letters);
    chars = d(asciistr);
    str = [chars{:}];
end
```

§2.2 Key Generation

Algorithm 2.2

Key generation proceeds as follows:

1. Determine the size of the alphabet m.
2. Fix block size n.
3. Pick a square matrix K of size $n \times n$.
4. If K is invertible modulo m, return K.
   Otherwise, set $K := K + I_n$ and repeat **Step 4**.
5. (Optional) Generate a matrix B of size $n \times 1$ for an additional shift (no constraints).

Note: The original Hill Cipher didn't include the optional shift described in **Step 5**. However, for a more secure cipher, an arbitrary shift (often referred to as an affine shift) was added. So, we call this general form **Affine Hill Cipher**.

Theorem 2.3

Step 4 of algorithm 2.2 terminates after at most $n + 1$ repetitions.

Proof. Let $f : \mathbb{R} \to \mathbb{R}$ such that $f(\lambda) = \det(A - \lambda I_n)$. From definition 1.10, we observe that f is a polynomial of degree n (left as exercise for the reader). Hence, f has at most n real roots. The roots represent the values of λ for which the matrix is singular. These values are exactly the values for which the matrix is not invertible modulo m. To see this, consider these two cases:

- $f(\lambda) = \det(A+\lambda I_n) = 0 \Rightarrow \gcd(\det(A+\lambda I_n), m) = 0 \not\equiv 1 \pmod{m}$. Then, $A+\lambda I_n$ is not invertible modulo m.
- Otherwise, $\gcd(\det(A + \lambda I_n), m) = 1 \equiv 1 \pmod{m}$ because m is prime.

Therefore, in the worst case scenario, the algorithm will terminate after $n + 1$ steps (n steps of generating invalid keys and an additional one for the valid key). □

§2.3 Encryption

Algorithm 2.4 1. **Input**: A string P of length s.

2. **Process plaintext**
   a) If s not a multiple of n, append padding symbols until it becomes divisible by n.
   b) Convert each letter of P to its numerical value as discussed in section 2.1.
3. **Partition** the resulting sequence of numbers into t consecutive column matrices, $p_1, p_2, \ldots, p_t$ where $p_i \in \mathbb{Z}_m^{n\times 1}$ and $\left(t = \frac{s}{n}\right)$.
4. **Encrypt each block**: For $i = 1, 2, \ldots, t$, generate $c_i = (Kp_i + B) \bmod m$ where $c_i \in \mathbb{Z}_m^{n\times 1}$.

5. **Process ciphertext**
   a) Convert each entry of $c_1, c_2, \ldots, c_t$ back to its corresponding letters.
   b) Concatenate these letters to form the ciphertext C.
6. **Return** C.

Here is an implementation of this algorithm using MATLAB:

```
% Button pushed function: EncryptButton
function EncryptButtonPushed(app, event)
    str=char(app.EnterStringTextArea.Value);
    strsize = strlength(str);
    asciistr = toascii(app, str);
    keysize = randi([1, min(strsize, 10)]);
    key = mod(randi(29, keysize),29);
    while gcd(int64(det(key)),29) ~= 1
        key = mod(key + eye(keysize), 29);
    end
    if mod(strsize, keysize) ~= 0
        zerosize = keysize-mod(strsize,keysize);
        asciistr = [asciistr; zeros(zerosize,1) + 26];
    end
    shift = randi(29, length(asciistr), 1);
    start = 1;
    enctxt = char(length(asciistr));
    while start < length(asciistr)
        encmatrix = mod(key*asciistr(start:start+keysize-1), 29);
        encmatrix = mod(encmatrix + shift(start:start+keysize-1),29);
        enctxt(start:start+keysize-1) = tochar(app,encmatrix.');
        start = start + keysize;
    end
    app.EncryptedTextTextArea_Encrypt.Value = enctxt;
    app.KeyTextArea_Encrypt.Value = strtrim(cellstr(num2str(key)));
    app.ShiftTextArea_Encrypt.Value = strtrim(cellstr(num2str(shift)))
end
```

§2.4 Decryption

Algorithm 2.5 1. **Input**: A ciphertext C of size s over an alphabet of length m, and the matrices $K_{n\times n}$ and $B_{n\times 1}$ used in encryption.

2. **Compute** K^{-1} using theorem 1.20.
3. **Process ciphertext**
   a) Convert each letter of C to its numerical value as discussed in *section* 2.1.
   b) Partition the resulting sequence into t column matrices $c_1, c_2, \ldots, c_t$ where $t = \frac{s}{n}$.
4. **Recover plaintext blocks** For $i = 1, 2, \ldots, t$, $p_i = K^{-1}(c_i - B) \bmod m$.

5. **Process plaintext**
   a) Convert each entry of p_i back to letters.
   b) Concatenate to form a padded plaintext P.
   c) Remove any padding symbols.
6. **Return** P.

Here is an implementation of this algorithm using MATLAB:

```
% Button pushed function: DecryptButton
function DecryptButtonPushed(app, event)
    str=char(app.EnterEncryptedStringTextArea.Value);
    strsize = strlength(str);
    asciistr = toascii(app, str);
    key = app.KeyTextArea_Decrypt.Value;
    key = cell2mat(cellfun(@str2num, key, 'UniformOutput', false));
    shift = app.ShiftTextArea_Decrypt.Value;
    shift = cell2mat(cellfun(@str2num, shift, 'UniformOutput', false));
    [~, detinv] = gcd(int64(det(key)), 29);
    detinv = mod(detinv, 29);
    keyinv = mod(detinv*adjoint(sym(key)), 29);
    keysize = length(key);
    asciistr = asciistr - shift;
    start = 1;
    enctxt = char(length(asciistr));
    while start < length(asciistr)
        encmatrix = mod(keyinv*asciistr(start:start+keysize-1), 29);
        enctxt(start:start+keysize-1) = tochar(app,encmatrix.');
            start = start + keysize;
    end
    app.DecryptedTextTextArea.Value = enctxt(1:strsize);
end
```

§2.5 Cryptanalysis

Cryptanalysis is the study and practice of analyzing and breaking cryptographic systems, which generally uses discovering the plaintext from ciphertext. Now, we discuss two methods to attack Hill Cipher.

Linear System Recovery via Known–Plaintext

This is the most effective way to break hill cipher when the attacker have access to pairs of plaintext and cipher text.

Algorithm 2.6 (How it works)
For every pair (P_i, C_i) such that P_i is the vector of plain text and C_i the vector of cipher text. Both of size $n \times 1$ There exists a key K of size $n \times n$. We have that for each pair,

$$C_i = KP_i \bmod 29$$

If an attacker has n linearly independent plaintext vectors and cipher text vectors, we can construct the two matrices.

- $P = [P_1, P_2, \ldots, P_n]$.
- $C = [C_1, C_2, \ldots, C_n]$.

Then the key matrix can be recovered as:

$$K = CP^{-1} \bmod 29.$$

Brute–Force Exploration of Key Space

This method attempts to try all possible keys until the correct one is found. It does not require any known plaintext, only a ciphertext and a way to recognize correct decryption (e.g., checking if the output resembles English using frequency analysis or dictionary matching).
For a Hill cipher with an $n \times n$ key matrix over $\mathbb{Z}_{29}$, the number of possible keys is the number of invertible matrices over $\mathbb{Z}_{29}$.

- For a 2×2 matrix, there are approximately **7,000** valid keys.
- For a 3×3 matrix, there are about **11 million** valid keys.

Thus:

- Brute-force attacks are feasible for 2×2 matrices, even manually or using simple programs.
- They become infeasible for larger matrix sizes without significant computational power.

Algorithm 2.7 (How it works)
It proceeds as follows:

1. Generate all possible $n \times n$ matrices over $\mathbb{Z}_{29}$.
2. For each matrix K, check if it is invertible, i.e., $\gcd(\det(K), 29) = 1$.
3. For each invertible key matrix:
   - Decrypt the ciphertext using K^{-1}.
   - Analyze the result to see if it resembles valid plaintext (e.g., using frequency analysis, dictionary lookup, or structure patterns).

Bibliography

[1] Hill, Lester S. “Cryptography in an Algebraic Alphabet.” The American M athematical Monthly 36, no. 6 (1929): 306–12. https://doi.org/10.2307/2298294.

[2] Siniora, Daoud. “Number Theory”. In Discrete Mathematics Lecture Notes, 139–61, 29 January, 2025. https://sites.google.com/view/daoudsiniora/lecture-notes?authuser=0.

[3] Siniora, Daoud. “Matrices and Linear Equations, Invertible Matrices.” Chapters. In Linear Algebra Lecture Notes, 1–76, n.d. https://sites.google.com/view/daoudsiniora/lecture-notes?authuser=0.

[4] “Extended Euclidean Algorithm.” Wikipedia, June 9, 2025. https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm.

[5] OpenAI, ChatGPT (July 19 version), https://chat.openai.com/.

[6] “Adjugate Matrix.” Wikipedia, May 10, 2025. https://en.wikipedia.org/wiki/Adjugate_matrix#References.