Breaking the ADFGVX Cipher: From Theory to Practical Cryptanalysis

Paul Sigloch

Vancouver Island University

December 5, 2025

Abstract

The ADFGVX cipher, introduced by the German military in 1918, combined Polybius square substitution with columnar transposition to create one of the most sophisticated manual encryption systems of World War I. This work presents a modern computational approach to breaking ADFGVX using ciphertext-only attacks, building on the pseudo-algorithm proposed by Lasry et al. The implementation employs a multi-phase heuristic search strategy that combines Index of Coincidence scoring with hill-climbing optimization to recover the transposition key, followed by n-gram-based language modeling to solve the substitution alphabet. The resulting system successfully recovers keys up to length 20 from reasonable ciphertext lengths, demonstrating how historical cryptanalytic insights can be enhanced through modern algorithmic techniques.

1 Introduction

1.1 Historical Overview of the Cipher

In the last year of the First World War, the German Empire faced both military exhaustion and the collapse of its communication security. Radio messages had become essential to coordinate operations between the German Supreme Command and the front lines, but these transmissions were no longer safe. Allied intelligence services had learned to intercept and decipher many of them. Unlike the British or the French, Germany had no central institution responsible for managing cryptographic development. Each army group and ministry used its own systems, often simple substitution or transposition ciphers that could be broken within hours by the experienced analysts of the Allied intercept units [7, pp. 297–300]. This weakness became a serious operational threat at a time when military success increasingly depended on the ability to transmit orders securely.

In early 1918, Lieutenant Fritz Nebel of the German signal corps was asked to design a new field cipher that could resist such cryptanalysis while remaining easy to use by telegraph operators. His solution was introduced in March 1918 under the name ADFGX. The cipher combined two classical techniques: substitution and transposition. Each plaintext character was first replaced by a pair of symbols from a small set of five letters A, D, F, G, X arranged in a 5×5 Polybius square. These letters were deliberately chosen because their Morse code signals sounded distinct from one another, reducing the risk of transmission errors under the conditions of noisy front-line communication [7, p. 300], [12, pp. 101–102]. The resulting pairs were then rearranged according to a key word through columnar transposition. This combination of two cipher types made ADFGX one of the most sophisticated manual ciphers used in the war.

By June of the same year, Nebel extended his system to a 6×6 grid, adding the letter V to the code and therefore the possibility to encode also digits 0–9. The system was widely adopted across German army communication networks and used for both tactical and diplomatic radio traffic [12, pp. 101–103]. It represented the culmination of centuries of development in classical cryptography, from ancient fractionation ideas like the Polybius square to modern composition ciphers, designed to withstand the growing power of organized cryptanalysis.

However, the cipher's apparent strength did not last long. Within only a few weeks of its introduction, the French cryptanalyst Georges Painvin succeeded in breaking it. Painvin analyzed intercepted German radio messages during the spring of 1918 and reconstructed both the underlying structure and the daily keys [7, pp. 302–303]. His success came at a decisive moment: one of the decrypted messages, *Le Radiogramme de la Victoire*, revealed the plan for a major German offensive toward Paris. The information helped the Allies prepare their countermeasures and contributed directly to halting the advance [12, p. 102]. Despite the breach, the German army continued to rely on the cipher until the Armistice in November, unaware that their encrypted communications were being read on a daily basis.

The story of ADFGVX did not end with the war. During the final months of 1918, similar systems were used for communications on the Eastern Front, where the American officer James R. Childs worked on the decryption of German diplomatic traffic intercepted between Germany, Romania, and the Ottoman Empire. His recovered keys and transcripts later formed one of the earliest systematic studies of a fractionating transposition cipher [1, pp. 190–192]. After the war, both Painvin's and Childs's achievements remained classified. The Allies considered the details of their cryptanalytic methods a state secret, and the German side was never informed that the cipher had been broken. Only many decades later, when archives were declassified, did the full story become widely known. Thereby, Fritz Nebel, who continued to work on cipher machines after 1918, was informed that his wartime system had in fact been broken and was reportedly shocked by this revelation [1, p. 206]. Their research, together with the earlier documentation preserved by Childs, revealed how both the ADFGX and ADFGVX ciphers had been compromised long before the end of the war and how those early breakthroughs laid the groundwork for modern cryptanalytic practice [12, pp. 106–109].

From a modern perspective, ADFGVX is more than a historical curiosity. It represents a useful test case for computational cryptanalysis. The cipher's structure, substitution through a Polybius square, expansion by fractionation, and final transposition, creates a large but highly structured keyspace. It is simple enough to model mathematically, yet complex enough to make brute-force search inefficient without heuristic guidance. For this reason, it has often been revisited by modern researchers as a benchmark for algorithmic breaking techniques [12, pp. 111–113], [2], [19]. Reconstructing how Painvin and Childs succeeded in their manual analysis allows us to apply modern computational methods to an early twentieth-century cipher and to explore questions that remain central to contemporary codebreaking: how to evaluate candidate keys, how to combine probabilistic scoring with combinatorial search, and how to balance accuracy and runtime in cryptanalysis.

1.2 Motivation, Goals, and Structure of This Work

This work focuses on the algorithmic reconstruction and enhancement of the process used to break the ADFGVX cipher. It follows the approach proposed by Lasry et al. [12], who described a pseudo-algorithm for ciphertext-only attacks, but extends it by addressing practical programming issues and exploring possible improvements. The goal is not only to reproduce the results obtained by Lasry's team but also to translate their method into a concrete, efficient implementation. By doing so, this study connects historical manual cryptanalysis with modern computational techniques and examines how the interplay between mathematics and programming can shed new light on a cipher that once influenced the outcome of a world war.

In the following Section 2, some important technical terms, definitions, and concepts are explained that are further used in this work. Then, Section 3 details the cipher workings, provides some encryption examples, and shows the implementation of it using Python. Section 4 discusses the techniques used to break the cipher and outlines the implementation of a modern ciphertext-only attack inspired by the pseudo-algorithm of Lasry et al. [12]. Section 5 briefly explains how the references and external resources were used and integrated into this project. Finally, Section 6 concludes with a reflection on the project and its programming aspects, followed by an outlook on historical operational weaknesses and possible hardening strategies for ADFGVX.

2 Background

2.1 Technical Definitions

This section reviews formal definitions and properties of the basic mechanisms invoked by the ADFGVX cipher. These primitives, i.e., fractionation, substitution, and transposition, are conceptually described by Stinson and Paterson [18], Gaines [6], and Lasry [11], with the formal mathematical framework and notation being an adaptation and synthesis of their descriptions. Later, Section 3 treats ADFGVX as the composition of these primitives. Moreover, basics and mechanisms of breaking methods of ADFGVX are detailed likewise.

Fractionation

Fractionation is a mapping that expands symbols from one alphabet into tuples over another typically smaller alphabet. Formally, let $\mathcal{P}$ be a plaintext alphabet and $\mathcal{C}$ a ciphertext alphabet. A *fractionation map* is a function

$$F : \mathcal{P} \longrightarrow \mathcal{C}^{\,t},$$

for some integer $t \geq 1$. Extending F to length n plaintexts gives

$$F : \mathcal{P}^{\,n} \longrightarrow \mathcal{C}^{\,tn}, \qquad F(p_1 \cdots p_n) = F(p_1) \,||\, F(p_2) \, \cdots \,||\, F(p_n),$$

where $||$ denotes concatenation. Fractionation increases the symbol granularity: a single plaintext symbol becomes a t-tuple over $\mathcal{C}$. Key properties:

- **Expansion:** message length is multiplied by t, which changes frequency statistics of individual symbols but preserves block structure at size t.
- **Losslessness:** F is typically injective on $\mathcal{P}$ (so each plaintext symbol maps to a unique t-tuple) allowing an inverse F^{-1} on the image.
- **Security effect:** fractionation hides single-letter frequencies but creates structure in t-tuples that can be exploited if the adversary discovers correlations between tuple positions.

Substitution

A substitution is a symbol-wise permutation (or general mapping) of an alphabet. Let $\mathcal{A}$ be an alphabet. A (mono-)substitution is a bijection

$$S : \mathcal{A} \longrightarrow \mathcal{A}.$$

Applied to strings,

$$S(a_1 a_2 \dots a_n) = S(a_1)\, S(a_2) \dots S(a_n).$$

Properties and variants:

- **Monoalphabetic vs. polyalphabetic:** A monoalphabetic substitution uses a single fixed bijection S; a polyalphabetic substitution uses a sequence $(S_1, \dots, S_n)$ possibly keyed, so that position matters.
- **Permutation model:** substitutive ciphers are often described as elements of the symmetric group $S_{|\mathcal{A}|}$ acting on $\mathcal{A}$.
- **Statistical effect:** substitution preserves letter frequencies up to permutation, so frequency analysis remains viable unless combined with other transforms (e.g., fractionation).

Transposition

A transposition is a reordering of positions in a string and therefore can be modelled as a permutation acting on indices. For fixed length N, let $\sigma \in S_N$ be a permutation of $\{1, \dots, N\}$. The transposition operator T_σ acts on strings $x = (x_1, \dots, x_N)$ by

$$T_\sigma(x) = (x_{\sigma(1)}, x_{\sigma(2)}, \dots, x_{\sigma(N)}).$$

When transposition is applied to messages whose length is not a multiple of a block width, it is common to model it as operating on the padded message or on a rectangular table with column/row coordinates induced by a key. Key properties:

- **Length-preserving:** transposition does not change the alphabet or the message length, only the order.
- **Group property:** composition of transpositions corresponds to permutation multiplication: $T_\sigma \circ T_\tau = T_{\sigma\circ\tau}$.
- **Effect on statistics:** transposition destroys short-range adjacency statistics but preserves multiset of symbols, so single-symbol frequencies remain identical.

Ciphertext-Only Attack

A ciphertext-only attack is the most basic cryptanalytic model, where the adversary has access only to ciphertexts and no direct information about the corresponding plaintexts or encryption key. Formally, the attacker observes ciphertexts $c_1, c_2, \dots, c_n$ encrypted under an unknown key K, and attempts to recover the plaintexts or the key itself, [18, pp. 39–40]. The attacker does not have access to decryption oracles or chosen plaintexts, making this one of the most challenging attack models [14, p. 41].

Success in this model typically relies on exploiting statistical properties of the underlying plaintext language, such as letter frequencies, bigram/trigram distributions, or the Index of Coincidence. For example, if the ciphertexts $c_1, c_2, \ldots, c_n$ are encrypted under the same key K, the adversary can attempt to analyze the statistical patterns in these ciphertexts to recover K. In the case of classical ciphers like ADFGVX, both ciphertext-only and known-plaintext attacks were historically relevant. While adversaries rarely had access to decryption oracles or chosen-plaintext capabilities, they often possessed cribs, predictable message fragments or standard formats, which significantly aided cryptanalysis. Pure ciphertext-only attacks become more feasible with longer ciphertexts, as these reveal more statistical patterns [11, pp. 3–5].

2.2 Statistical and Heuristic Background

Classical attacks on fractionated substitution ciphers and transposition ciphers rely on the statistical regularities of natural language. When a ciphertext is transformed with an incorrect key, the resulting text typically resembles a random sequence of symbols. In contrast, partial or correct keys tend to reproduce characteristic linguistic patterns of the target language [11, pp. 4–5, 33]. The following concepts summarize the statistical and heuristic foundations relevant to the cryptanalysis of systems such as ADFGX and ADFGVX.

2.2.1 Index of Coincidence

The index of coincidence (IC) measures how unevenly symbols are distributed in a text. The concept was introduced by Friedman as a quantitative tool for distinguishing random from meaningful text [5, pp. 11–13, 39–40]. Let $x = x_1 \ldots x_N$ be a string over an alphabet A, and let n_a denote the number of occurrences of symbol $a \in A$. The IC is defined as

$$IC(x) = \frac{\sum_{a \in A} n_a(n_a - 1)}{N(N-1)}.$$

Natural languages typically exhibit substantially higher IC values than uniformly random sequences because of strong letter frequency biases. The same idea extends to bigrams, trigrams, and tetragrams by treating these units as atomic symbols and computing analogous coincidence measures [6].

2.2.2 N-gram Language Models

An n-gram language model estimates the likelihood of natural language by counting the frequencies of length-n sequences in large corpora [17, p. 1049]. Such models are central in both classical cryptanalysis and modern decipherment research. Quadgram statistics are especially effective for scoring candidate plaintexts, as they capture short-range structural dependencies without requiring very long texts. Formally, an n-gram score can be expressed as

$$score(x) = \sum_{g \in G} \text{count}_g(x) \; \log P(g),$$

where G is the set of n-grams, $\text{count}_g(x)$ counts occurrences in x, and $P(g)$ is the empirical probability estimated from a reference corpus [11, pp. 30–31], [17, p. 1049].

2.2.3 Heuristic Search Methods

The key spaces arising in classical ciphers are often too large for exhaustive enumeration. Heuristic optimization methods therefore play an important role in navigating these spaces.

Hill climbing. Hill climbing is a local search method widely used in automated attacks on substitution and transposition ciphers. Starting from an initial key (typically chosen randomly), the algorithm evaluates neighboring keys created by small permutations and moves to a neighbor if it yields a higher statistical score. Repeated iterations lead to a local optimum, which motivates multiple restarts from different initial configurations, whereby the use of random restarts makes the overall attack procedure stochastic [16, pp. 333–334], [11, pp. 5–6].

Simulated annealing. Simulated annealing introduces stochastic acceptance of inferior moves and has long been used as an effective tool for escaping local maxima. A temperature parameter controls the transition from exploratory to greedy search. This mechanism is well suited to the rugged search landscapes that arise in permutation-based cipher keys [8, pp. 671–673], [9, pp. 976–978].

Polishing and local refinements. Once a promising region has been reached, small deterministic refinements, such as systematic checks of single-symbol swaps or short permutations, can extract additional improvements. These inexpensive operations often serve as cleanup steps at the end of a heuristic cycle [16, p. 334].

Seed selection. Heuristic searches benefit from informed initial keys, or seeds. Such seeds may be derived from simple structural expectations, partial linguistic cues, or coarse statistical indicators. Good seed selection often improves both convergence speed and result quality [15, pp. 3, 7].

3 Formal Definition and Functioning of the Cipher

The ADFGVX cipher is a classical example of a fractionating substitution cipher, which combines the three previously defined cryptographic primitives: fractionation and substitution via a Polybius square, and transposition via a columnar rearrangement of symbols. The theory and definitions presented in this section are primarily derived from Lasry et al. [12], [11]. In this section, we focus on the exact composition of these primitives into the full cipher, restating the formal definitions of substitution, fractionation, and transposition in the concrete context of ADFGVX.

3.1 Polybius Square

The substitution step maps each plaintext character to a pair of letters from the ciphertext alphabet

$$\mathcal{C} = \{A, D, F, G, V, X\}$$

using a 6×6 Polybius square S. This square contains all 26 letters of the Latin alphabet and the 10 digits. For $p \in \mathcal{P}$, let (r, c) be its row and column in S, labeled by elements of $\mathcal{C}$. The mapping is

$$E_{\text{sub}}(p) = rc.$$

For a plaintext block $p_1 p_2 \ldots p_n$, the fractionated ciphertext is

$$C_{\text{frac}} = (c_1, c_2, \ldots, c_{2n}) \in \mathcal{C}^{2n}.$$

Example. If $p = \text{H}$ is located at row D, column G, then

$$E_{\text{sub}}(\text{H}) = DG.$$

3.2 Columnar Transposition

Next, the sequence C_{frac} is reordered by a columnar transposition defined by a secret key K of length m. The sequence is written row by row into a rectangular table with m columns. Sorting the key K alphabetically (or numerically, depending on convention) defines a permutation $\sigma \in S_m$, which is then applied to the columns of the table.

Formally, the transposition is a function

$$E_{\text{trans}} : \mathcal{C}^{2n} \longrightarrow \mathcal{C}^{2n}, \quad E_{\text{trans}}(C_{\text{frac}}) = (c_{\sigma(1)}, c_{\sigma(2)}, \ldots, c_{\sigma(2n)}).$$

This step changes only the *order* of symbols, not the alphabet or the length of the ciphertext.

3.3 Encryption Function

The complete encryption of a plaintext string $p_1 p_2 \ldots p_n$ is the composition of two steps:

$$E = E_{\text{trans}} \circ E_{\text{sub}}.$$

Step 1: Substitution. Each plaintext symbol $p_i \in \mathcal{P}$ is mapped to a pair $(r, c) \in \mathcal{C}^2$ by the Polybius square, producing the fractionated sequence

$$C_{\text{frac}} = E_{\text{sub}}(p_1 p_2 \ldots p_n) \in \mathcal{C}^{2n}.$$

Step 2: Padding. Let $m = |K|$ be the key length. If $2n \not\equiv 0 \pmod{m}$, append dummy symbols $x \in \mathcal{C}$ until the length of C_{frac} is divisible by m. Formally, we obtain the padded sequence

$$C_{\text{pad}} = (c_1, \ldots, c_{2n+r}) \quad \text{with } r < m \text{ and } (2n + r) \equiv 0 \pmod{m}.$$

Although padding is not strictly necessary in modern descriptions of the cipher, it was historically employed to ensure that the transposition stage always produced a complete rectangular table, which simplified manual encryption and decryption.

Step 3: Columnar Transposition. Write C_{pad} row by row into an array with m columns. Sorting the key K defines a permutation $\sigma \in S_m$, which determines the new column order. The transposition function is

$$E_{\text{trans}}(C_{\text{pad}}) = C_{\text{cipher}} \in \mathcal{C}^{2n+r}.$$

Overall Encryption Formula. Thus, encryption is

$$E(p_1p_2\dots p_n) = E_{\text{trans}}\big(E_{\text{sub}}(p_1p_2\dots p_n)\big),$$

where padding ensures compatibility with the transposition step. The ciphertext may therefore be slightly longer than $2n$ due to padding.

3.4 Decryption Function

Decryption is the inverse of encryption, applied in reverse order:

$$D = E_{\text{sub}}^{-1} \circ E_{\text{trans}}^{-1}.$$

Step 1: Inverse Transposition. Given $C_{\text{cipher}} \in \mathcal{C}^{2n+r}$, rearrange its columns according to the inverse permutation σ^{-1} induced by the key K. This yields the padded sequence:

$$C_{\text{pad}} = E_{\text{trans}}^{-1}(C_{\text{cipher}}).$$

Step 2: Removing Padding. If padding was applied during encryption, the last r symbols are dummy symbols. They must be removed to recover the original fractionated sequence:

$$C_{\text{frac}} = (c_1, c_2, \dots, c_{2n}).$$

Step 3: Inverse Substitution. Parse C_{frac} into pairs (c_{2i-1}, c_{2i}). Each pair (r, c) corresponds to a plaintext symbol from the Polybius square:

$$E_{\text{sub}}^{-1}(rc) = S_{r,c}.$$

This reconstructs the plaintext message $p_1p_2\dots p_n$.

Overall Decryption Formula. Thus, decryption is

$$D(C_{\text{cipher}}) = E_{\text{sub}}^{-1}\big(E_{\text{trans}}^{-1}(C_{\text{cipher}})\big),$$

with the additional step of removing padding.

3.5 Keyspace

The security of the ADFGVX cipher derives from two secret components: the arrangement of the 36 characters (letters A–Z and digits 0–9) in the Polybius square, and the columnar transposition key. Any permutation of the 36 characters is allowed for the square, giving a keyspace of

$$|\mathcal{K}_{\text{square}}| = 36! \approx 3.72 \cdot 10^{41} \approx 2^{138}.$$

The transposition key consists of m letters, where historical instructions required operators to choose a key of at least 16 characters, which is confirmed by analysis of some recovered keys, where the shortest found had 16 letters and the longest 23 [11, p. 106]. The key defines a columnar ordering of length m, giving $m!$ possibilities for that particular length. The total keyspace for the transposition can be written using summation notation as

$$|\mathcal{K}_{\text{transposition}}| = \sum_{n=16}^{23} n! \approx 2.7 \cdot 10^{22} \approx 2^{75}.$$

Combining both components, the overall keyspace of the ADFGVX cipher is

$$|\mathcal{K}_{\text{total}}| = 36! \times \sum_{n=16}^{23} n! \approx 10^{64} \approx 2^{213}.$$

This large keyspace is one reason why the cipher was considered secure for practical use during its period of deployment.

3.6 Manual Example Demonstration

To illustrate the complete process of encryption and decryption, we consider the plaintext message

$$P = \texttt{I LOVE CRYPTOGRAPHY},$$

together with the columnar key $K = \texttt{MATH}$.

Step 1: Polybius Square. We begin with the following 6×6 Polybius square S, which contains the 26 letters of the Latin alphabet followed by the 10 digits (see Figure 1):

	A	D	F	G	V	X
A	A	B	C	D	E	F
D	G	H	I	J	K	L
F	M	N	O	P	Q	R
G	S	T	U	V	W	X
V	Y	Z	0	1	2	3
X	4	5	6	7	8	9

Figure 1: Polybius square for ADFGVX cipher. The row and column headers are drawn from the ciphertext alphabet $C = \{A, D, F, G, V, X\}$.

> **Historical note.** In actual German military use, the content of the Polybius square was not fixed. Operators were instructed to change the internal alphabet order daily. The square shown above is therefore only an example for demonstration purposes.

Step 2: Fractionating Substitution. Each plaintext character is mapped to a pair (r, c) corresponding to its row and column in S. Spaces are omitted. Applying this to all characters of `ILOVECRYPTOGRAPHY` yields the fractionated sequence

$$C_{\text{frac}} = \texttt{DFDX FFGG AVAF FXFG VAGD FFDA FXAA FGDD VAXX}^{1} \in C^{36}.$$

[1]Spaces are added for clarity. The actual sequence is contiguous.

Step 3: Padding. Since the key length is $|K| = 4$, the fractionated sequence is padded with two additional symbols from $\mathcal{C}$ to make the total length divisible by 4. Historically, the padding symbols were often chosen as `X`. The padded sequence therefore has length 36.

Step 4: Columnar Transposition. The padded sequence is arranged row by row into a table with 4 columns labeled by the key letters (see Figures 2 and 3):

	M	*A*	*T*	*H*
	3	1	4	2
1	*D*	*F*	*D*	*X*
2	*F*	*F*	*G*	*G*
3	*A*	*V*	*A*	*F*
4	*F*	*X*	*F*	*G*
5	*V*	*A*	*G*	*D*
6	*F*	*F*	*D*	*A*
7	*F*	*X*	*A*	*A*
8	*F*	*G*	*D*	*D*
9	*V*	*A*	*X*	*X*

Figure 2: Original key order

	A	*H*	*M*	*T*
	1	2	3	4
1	*F*	*X*	*D*	*D*
2	*F*	*G*	*F*	*G*
3	*V*	*F*	*A*	*A*
4	*X*	*G*	*F*	*F*
5	*A*	*D*	*V*	*G*
6	*F*	*A*	*F*	*D*
7	*X*	*A*	*F*	*A*
8	*G*	*D*	*F*	*D*
9	*A*	*X*	*V*	*X*

Figure 3: Sorted key order

Sorting the key alphabetically ($\texttt{A} < \texttt{H} < \texttt{M} < \texttt{T}$) gives the column permutation

$$\sigma : \mathrm{M} \longrightarrow 3, \mathrm{A} \longrightarrow 1, \mathrm{T} \longrightarrow 4, \mathrm{H} \longrightarrow 2 \quad \text{or equivalently} \quad (1 \longrightarrow 3, 2 \longrightarrow 1, 3 \longrightarrow 4, 4 \longrightarrow 2),$$

where the first notation indicates that column M (position 1) moves to position 3 in sorted order, column A (position 2) moves to position 1, and so forth. Reading the columns in sorted key order (A, H, M, T) produces the final ciphertext

$$C_{\text{cipher}} = \texttt{FF VX AF XG AX GF GD AA DX DF AF VF FF VD GA FG DA DX}^{1} \in \mathcal{C}^{36},$$

which is read column-wise from top to bottom.

Step 5: Decryption. To decrypt, the receiver:

1. Reconstructs the padded rectangle using the column order σ,
2. Applies the inverse permutation σ^{-1} to restore the original column positions,
3. Removes the padding characters,
4. Parses the fractionated pairs (r, c) to retrieve the original plaintext letters from the Polybius square S.

Applying these steps recovers the original message

`ILOVECRYPTOGRAPHY.`

3.7 Implementation using Python

We implemented the ADFGVX cipher in Python to demonstrate both encryption and decryption. The main functions are `encrypt_message` and `decrypt_message`, which encapsulate the two main steps of the cipher: fractionation using a Polybius square and columnar transposition according to a key. The supporting functions handle the details of these operations and can be summarized as follows. The full implementation is available in the accompanying Git repository [2].

- `create_polybius_square`: builds the mappings between plaintext characters and label pairs for the Polybius square.
- `fractionate_text`: converts a plaintext string into a sequence of label pairs according to the Polybius square.
- `get_column_order`: calculates the permutation of columns based on the alphabetical order of the key.
- `apply_columnar_transposition`: writes the fractionated sequence into a rectangle and reads off the columns according to the key order, optionally adding padding symbols.
- `reverse_columnar_transposition`: reconstructs the fractionated text from the ciphertext, reversing the columnar transposition and removing padding if used.

As detailed in Listing 1, the encryption function first fractionates the plaintext, applies the columnar transposition, and then formats the resulting string into pairs of symbols separated by spaces for readability.

```python
def encrypt_message(plaintext, polybius_map, column_key):
    fractionated = fractionate_text(plaintext, polybius_map)
    ciphertext = apply_columnar_transposition(fractionated, column_key)
    return " ".join(ciphertext[i : i + 2] for i in range(0, len(ciphertext), 2))
```

Listing 1: ADFGVX Encryption Function

The decryption function shown in Listing 2 reverses these steps: it removes spaces, reverses the transposition, strips any padding symbols, and maps the label pairs back to plaintext characters.

```python
def decrypt_message(ciphertext, inverse_polybius_map, column_key):
    ciphertext = ciphertext.replace(" ", "")
    fractionated_text = reverse_columnar_transposition(ciphertext, column_key)
    fractionated_text = fractionated_text.rstrip("X")
    plaintext = ""
    for i in range(0, len(fractionated_text), 2):
        if i + 1 < len(fractionated_text):
            label_pair = fractionated_text[i : i + 2]
            if label_pair in inverse_polybius_map:
                plaintext += inverse_polybius_map[label_pair]
    return plaintext
```

Listing 2: ADFGVX Decryption Function

[2] `https://github.com/penthooose/ADFGVX_cipher_breaker`

This Python implementation closely follows the manual procedure described earlier, demonstrating the invertibility of the ADFGVX cipher and making it possible to verify examples or explore alternative keys and Polybius square arrangements. The supporting functions handle the construction of the Polybius square, the fractionation of plaintext into label pairs, the calculation of the column permutation from the key, and the application or reversal of the columnar transposition, including the optional use of padding symbols.

While online tools such as *CrypTool* [3] offer only the unpadded version of the cipher, our implementation also allows padding to reproduce the historical behavior of the ADFGVX cipher, where extra symbols (typically `X`) were added to complete the transposition rectangle.

4 Cryptanalysis and Breaking of the Cipher

This section explores methods for attacking the ADFGVX cipher, starting from introducing the potential weaknesses of the cipher and its applications, presenting historical approaches used during and after World War I, and culminating in a modern ciphertext-only attack by Lasry et al. [12]. Furthermore, an implementation of their proposed algorithm and practical insights for efficient realization of this attack follows.

4.1 Structural and Operational Weaknesses of ADFGVX

Before describing specific attacks, it is useful to summarize the principal weaknesses of the ADFGVX design and, perhaps more critically, the operational practices that rendered the cipher vulnerable in practice. These observations both motivate the historical techniques and frame the assumptions used by modern ciphertext-only algorithms. They are synthesized from Kahn [7], Lasry et al. [12], Bauer [1], and Childs [3].

1. Fractionation introduces structured correlations. The Polybius substitution expands each plaintext symbol into a fixed-length tuple (in ADFGVX, a pair) over the small label alphabet $C = \{A, D, F, G, V, X\}$. Formally the fractionation map $F : \mathcal{P} \to C^2$ is injective, so frequency information at the plaintext-symbol level is not destroyed but redistributed across ordered pairs. Consequently, while single-symbol frequencies in the final ciphertext no longer directly reflect plaintext-letter frequencies, correlations between positions inside each pair and between pairs remain. These correlations (for example, conditional distributions of the second label given the first) provide statistical structure that an attacker may exploit with an n-gram or digram scoring model.

2. Transposition preserves multisets and column continuity. The columnar transposition is an index permutation $T_\sigma : C^N \to C^N$ that preserves the multiset of symbols; in particular, single-symbol frequency counts are invariant under transposition. More importantly for historical cryptanalysis, transposition preserves the continuity of column fragments: a contiguous suffix, or prefix, of the fractionated message becomes a contiguous suffix, or prefix, of some column after the rectangle is written row-wise.

[3] `https://legacy.cryptool.org/en/cto/adfg-v-x`

3. Key reuse and limited key management. Security relies on the secrecy and freshness of two keys: the Polybius square ordering and the column key. In practice, keys were reused for whole days and squares were changed relatively infrequently. Reuse of the same square and column key across many messages dramatically increases the probability of finding exploitable redundancies, such as matching prefixes/suffixes, or repeated formulaic phrases, enabling multi-message attacks.

4. Predictable or low-entropy choices in practice. Although the theoretical keyspace is large, operational constraints reduced effective entropy. Operators often selected mnemonic or human-chosen column keys such as dictionary words, common phrases, or the names of known persons, and routine instructions sometimes led to predictable square arrangements. Padding conventions (commonly the symbol `X`) and the omission or normalization of punctuation and spaces also reduce entropy and create patterns that aid scoring functions.

5. Vulnerability to cribbing and chosen/plaintext models. If the adversary knows or can guess snippets of plaintext (cribs) or can cause encryptions of controlled plaintexts, then both substitution and transposition become far easier to invert. Historically, routine message formats (headers, signatures, repeated phrases) acted as effective cribs.

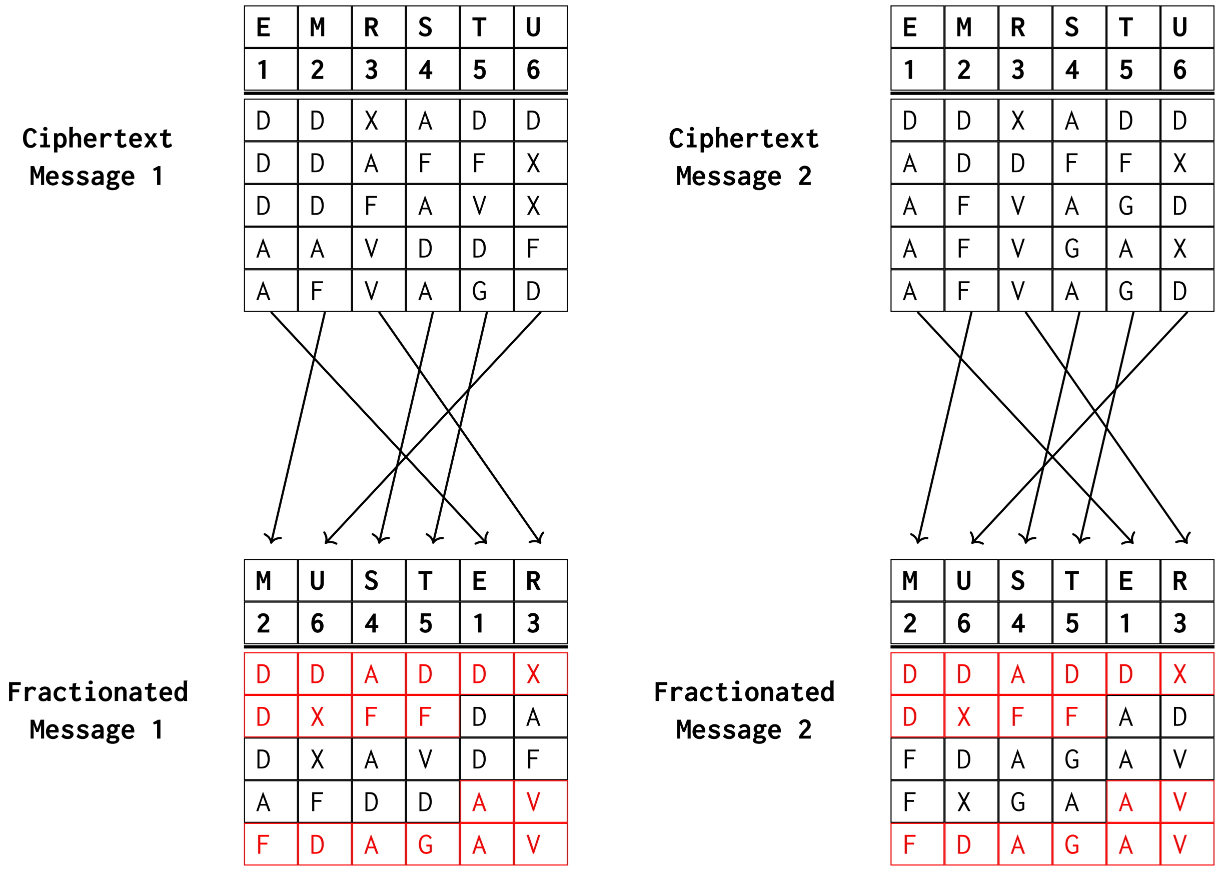
Taken together, these structural and operational weaknesses explain why the following historical techniques and later statistical/heuristic ciphertext-only attacks are effective in practice: fractionation supplies exploitable local structure, transposition preserves fragment continuity, and real-world key reuse and stereotyped message content amplify these effects. Modern attacks formalise this intuition by using probabilistic scoring (n-gram models) and heuristic search to exploit the same dependencies even when only a single ciphertext is available, given that it is reasonable long.

4.2 Historical methods (1918) — Painvin & Childs

Georges Painvin and James R. Childs exploited the same structural weakness of the ADFGVX system in 1918. Because many German messages reused a single daily key and followed stereotyped formats, repeated openings, closings, and routine phrases produced identical fragments after fractionation that partly survived the columnar transposition in recognizable form. By locating these coincidences across ciphertexts, both analysts were able to match corresponding columns and progressively reconstruct the transposition order. Painvin focused on long overlaps observed during the 1918 Spring Offensive, while Childs applied the same principle later that year using cribs from standard German message templates [7, pp. 300–303], [1, pp. 190–192]. Once enough columns were aligned, the remaining substitution on the Polybius digraphs could be solved through frequency analysis and predictable phrase structure.

Accumulating these column matches across several intercepted messages yielded the complete daily key. Painvin reported nine such successes, while Childs and British analysts achieved at least one additional recovery [12, p. 108]. The approach depended on favourable traffic conditions: many messages encrypted with the same key, or messages containing predictable patterns. When messages were short, unique, or used frequently changing keys, the method provided little leverage [12, pp. 107–108]. Its effectiveness thus hinged as much on German operational habits as on inherent weaknesses of the cipher.

To illustrate Painvin's method, Figure 4 shows two plaintext messages sharing matching prefix and suffix fragments. Each message is first fractionated using the Polybius square and written row-wise into a rectangle. Applying a columnar transposition with key MUSTER (permutation $\sigma^{-1} = (1 \rightarrow 5, 2 \rightarrow 1, 3 \rightarrow 6, 4 \rightarrow 3, 5 \rightarrow 4, 6 \rightarrow 2))$ produces the ciphertext, which is read column-wise. The matching fragments (shown in red) survive intact within certain columns, allowing an analyst to pair columns across messages and infer the hidden permutation.



E	M	R	S	T	U
1	2	3	4	5	6
D	D	X	A	D	D
D	D	A	F	F	X
D	D	F	A	V	X
A	A	V	D	D	F
A	F	V	A	G	D

E	M	R	S	T	U
1	2	3	4	5	6
D	D	X	A	D	D
A	D	D	F	F	X
A	F	V	A	G	D
A	F	V	G	A	X
A	F	V	A	G	D

M	U	S	T	E	R
2	6	4	5	1	3
D	D	A	D	D	X
D	X	F	F	D	A
D	X	A	V	D	F
A	F	D	D	A	V
F	D	A	G	A	V

M	U	S	T	E	R
2	6	4	5	1	3
D	D	A	D	D	X
D	X	F	F	A	D
F	D	A	G	A	V
F	X	G	A	A	V
F	D	A	G	A	V

Figure 4: Painvin's method: ciphertext messages (top) are written column-wise and transposed back to fractionated form (bottom) which is read row-wise. Bold letters show the key, with corresponding column numbers. Arrows indicate the columnar transposition mapping. Red cells mark overlapping fragments between messages.

4.3 Mid-century Theoretical Methods — Konheim

Alan G. Konheim [10] explored a purely combinatorial, theoretical approach to attacking the ADFGVX cipher, proposed 1985. His method treats the columnar transposition as a structured permutation problem and aims to reconstruct the column ordering by analyzing adjacency relations between column vectors in the fractionated ciphertext. Once the permutation is recovered, the underlying Polybius substitution can, in principle, be solved as a monoalphabetic substitution on digraphs.

Formally, the method relies on deriving linear constraints on the hidden permutation $\sigma \in S_m$ from observed column adjacency patterns. Sufficiently many such constraints determine σ uniquely, after which the substitution can be inverted.

Konheim's approach is largely impractical for historical ADFGVX traffic. It requires very long, uninterrupted fractionated sequences, low noise, and predictable padding, conditions rarely met in wartime messages [12, p. 110]. Consequently, the method is mainly of theoretical interest, illustrating what is algorithmically possible under idealised conditions.

4.4 Modern Ciphertext-Only Attack — George Lasry

Lasry's ciphertext-only attack [12, pp. 111–114] provides an effective and practical method for recovering ADFGVX encryptions by combining statistical scoring with heuristic search. The central idea is to view the recovery of the transposition key as an optimization task: if the columns are rearranged correctly, the resulting fractionated stream begins to exhibit the statistical patterns characteristic of valid ADFGVX digrams. These regularities can be measured without requiring plaintext knowledge, which makes the method suitable even when only a single or a few ciphertexts are available.

Combined IC scoring. To evaluate candidate transposition keys, the algorithm computes the IC over overlapping symbol groups in the fractionated stream. A *bigram IC* is calculated from pairs of ciphertext symbols; each such pair corresponds to one plaintext letter in the Polybius square. For even key lengths, the method also computes a *quadgram IC* from four ciphertext symbols, which correspond to two consecutive plaintext letters. These two IC values are combined into a single score, and the quadgram component is given relatively more weight for even key lengths where bigram IC alone is less discriminative. The resulting combined score serves as a language-independent proxy for how "regular" or "language-like" the decrypted fractionated stream appears.

Neighbourhood transformations. To explore the space of column permutations, the method uses a fixed set of local transformations:

1. swapping any two positions,
2. swapping two adjacent segments,
3. rotating a consecutive segment,
4. reversing the entire ordering,
5. or swapping adjacent pairs of elements.

Each transformation produces a nearby key candidate, defining the neighbourhood for the search.

Hill-climbing strategy. The transposition key is recovered using a simple hill-climbing loop:

1. initialise with a random column permutation;
2. decrypt the ciphertext according to this permutation and compute its combined IC score;
3. apply all neighbourhood transformations and keep any permutation that improves the score;
4. repeat until no improving neighbour exists (local maximum);
5. if the score is sufficiently high, attempt to recover the Polybius square using standard substitution-solving techniques;
6. otherwise restart from a new random key.

Short climbs from many random starting points are preferred over a single long climb, as this reduces the chance of becoming trapped in poor local maxima.

Overall technique. Lasry's approach follows the classical principle of recovering the transposition before the substitution. Once a promising column ordering is identified, the resulting digram stream is close enough to its correct alignment that the substitution stage can be handled by conventional monoalphabetic solving methods, typically guided by n-gram scoring. In practice, this combination of local search, combined IC scoring, and a carefully chosen set of transformations allows the method to recover keys from very limited ciphertext.

4.5 Implementation and Extension of Lasry et al.'s Algorithm

This section outlines how the breaking method proposed by Lasry et al. is turned into a practical ciphertext-only solver. The implementation follows this two-step strategy: first recover the column order, then optimise the substitution alphabet. Our program extends Lasry's pseudo-algorithm with multi-phase search, automatic parameter tuning, seed generation strategies, and a dedicated alphabet optimizer. The goal is a robust ciphertext-only attack that performs well even for long keys and scrambled alphabets. The full implementation of both the cipher and the breaker, including all algorithms, supporting utilities, and doc files, is available in the accompanying Git repository [4].

4.5.1 Overall Concept and Workflow

The breaking process is split into two stages that are executed sequentially:

1. **Transposition key recovery.** For each candidate key length, the solver searches the space of column permutations using a multi-phase strategy. Each candidate key is scored by a combined bigram and quadgram IC. Bigram IC evaluates pairs of fractionated symbols, corresponding to single plaintext letters, while quadgram IC inspects overlapping four-symbol windows, which become more informative for longer and even-length keys.
2. **Substitution solving.** Once promising column orders are identified, the corresponding fractionated streams are reconstructed. A dedicated alphabet optimizer then searches the 36-symbol substitution space using language-model scoring and several types of local and global refinements.

The program orchestrates these steps automatically. For each key length, it keeps the best-scoring candidates, re-ranks them with a lightweight language heuristic, and forwards only a small shortlist to the alphabet optimization stage. The best results are stored, allowing repeated inspection or manual verification. By design, the workflow reduces the extremely large keyspace into manageable segments and applies more computational effort only where intermediate results already show linguistic structure.

4.5.2 Transposition Key Recovery

The implementation uses a multi-phase search procedure that expands the core ideas of Lasry et al.:

Seed generation. For each key length, the solver generates many starting permutations. Two mechanisms are used:

[4] `https://github.com/penthooose/ADFGVX_cipher_breaker`

- *Fragment voting seeds.* For long keys, ciphertext is sliced into provisional column fragments. Adjacency relations are scored using overlapping bigrams or quadgrams, producing initial orderings that frequently approximate the correct column sequence.
- *Random seeds.* Additional shuffled candidates ensure coverage of the search space when fragment structure is weak.

Three-phase key search. The main search consists of three increasingly focused stages:

Phase 1: Many seeds are polished quickly with short hill-climb runs and no or only few restarts. Only a rough IC improvement is required.

Phase 2: The best Phase 1 candidates undergo deeper search by allowing more restarts and hill climbing iterations.

Phase 3: The best of Phase 2 are refined with long runs and optionally crosschecked with simulated annealing.

All phases operate on the same scoring principle: the decrypted fractionated stream is evaluated using a weighted sum of bigram and quadgram IC. The weight of quadgram IC increases with key length and receives an additional boost for even lengths, reflecting the more regular pair structure expected in those cases. A continuous modulation based on ADFGVX pair-regularity further improves discrimination between correct and incorrect permutations.

Neighbourhood and optimization. Each hill-climb iteration examines a rich set of Lasry-style transformations: single swaps, segment swaps, segment rotations, whole-key reversal, and pair-swaps. The solver always selects the best improving move. Multiple restarts ensure that it escapes poor local optima. For longer keys the search may run in parallel and dynamically allocate more restarts when strong candidates appear.

Shortlisting. For each key length, the program stores the best candidates and re-ranks near ties using a simple language-likeness score derived from the decoded text under a canonical alphabet. This yields a compact shortlist for the substitution stage.

4.5.3 Substitution Solving

After a strong column order is found, the remaining task is to identify the 6×6 Polybius square. The program reconstructs the fractionated streams for the shortlisted keys and applies a multi-step optimization process:

Search domain and seeds. The alphabet is treated as a 36-symbol permutation. Seeds include the canonical alphabet, simple rotations, and frequency-based candidates extracted from the provisional plaintexts. Pattern-based seeds are also constructed: long words in the decoded text yield repetition patterns that can match dictionary entries and impose partial letter-to-symbol constraints.

Local search. The optimizer performs hill climbing over the prefix of the alphabet (letters only, unless digits are allowed to vary). At each step, swaps or small permutations are tested, and those improving the n-gram score are accepted.

Language-model scoring. A combined n-gram model (mono-, bi-, tri-, and quadgram statistics) assesses the plausibility of the decoded texts. Additional penalties discourage digits appearing inside alphabetic words, and optional bonuses reward frequent words when enabled. Furthermore, dictionary-based pattern matching helps identify plausible plaintext words and increases the score when a high proportion of the decrypted text aligns with valid vocabulary.

Constraint solving and polishing. If enough structural constraints are discovered, the remaining letter assignments are completed by systematically extending the partial mapping and eliminating inconsistent choices through backtracking and forward checking. Deterministic polishing steps then exhaustively test pairwise swaps or focused permutations around suspected weak positions, yielding consistent improvements. When digits are free to move, they are assigned greedily at the end.

Result consolidation. The best alphabets are stored together with the corresponding plaintexts. If manual verification is enabled, the user can inspect and select the final output. Otherwise, the program returns the top candidates automatically.

Overall, this second stage completes the attack by aligning fractionated pairs with an internally consistent Polybius mapping and producing readable plaintexts from ciphertext-only input. If enabled, the best results are also written back into the configuration file, which stores all information associated with the currently selected attack, such as ciphertexts, plaintexts, and the key lengths under consideration.

5 Use and Integration of References

The references employed in this work span three interconnected domains: historical context, technical descriptions of the ADFGVX cipher, and methodological foundations for modern cryptanalytic techniques. The central reference is the paper by Lasry et al. [12], which provided both the conceptual basis for a ciphertext-only attack and a pseudo-algorithm adapted for practical implementation. Historical perspective was drawn from Bauer [1] and Kahn [7], which clarified the operational context in which the cipher was deployed and broken. Additional cryptanalytic insights were obtained from Li Wenjie [13], Diepenbroek [4], Chhatrapati [2], and Venkateswarlu & Kakarla [19], whose analyses of ADFGVX variants informed potential cipher hardening strategies briefly discussed in Section 6.

For algorithmic implementation, the work drew upon literature covering heuristic optimization techniques, including hill climbing, simulated annealing, and local-search strategies. Foundational concepts and terminology were informed by Stinson and Paterson [18], Gaines [6], and Lasry [11]. Statistical methods and search heuristics were guided by Friedman [5], Snyder et al. [17], and Selman [16]. These references informed the approach to problem-solving and optimization structure rather than providing direct algorithmic implementations.

Supplementary data resources supported scoring and dictionary-based components. Public English [5] and German [6] wordlists enabled dictionary lookups during substitution solving. Frequency tables and log-probability models for n-grams (English and German) were obtained from an openly available cryptanalysis dataset [7], providing empirical language models for scoring functions throughout transposition and substitution stages.

[5] https://github.com/dwyl/english-words
[6] https://gist.github.com/MarvinJWendt/2f4f4154b8ae218600eb091a5706b5f4
[7] https://github.com/torognes/enigma

6 Conclusion

The historical attacks on ADFGVX show that the cipher's weaknesses were not only mathematical but often operational. Many of the vulnerabilities that made cryptanalysis possible arose from operator behavior rather than from the design alone. Daily keys were reused across large batches of traffic, message formats followed predictable structures, and padding conventions were deterministic. In combination with limited variation in the Polybius squares, this created a fertile environment for pattern-based attacks and statistical leakage.

From today's perspective it is clear how the system could have been made considerably more resistant. Even without changing the fundamental concept, small operational improvements would have complicated both manual and automated cryptanalysis. Introducing per-message randomness, such as fresh randomized Polybius squares and transposition keys for each message, would have eliminated many of the structural regularities exploited by Painvin and later researchers.

The same holds for modern, program-based attacks. The approach developed in this project is highly effective against historically accurate traffic: deterministic padding, consistent alphabet ordering, and fixed structure make n-gram scoring and multi-phase key search viable. However, if the cipher had been used with more aggressive randomization, such as variable padding, fixed-size message blocks, rotating substitution alphabets, or multiple rounds of transposition, the search space would grow so rapidly that a ciphertext-only attack would become extremely hard. Polyalphabetic variants, in which several Polybius squares are cycled according to a long, non-periodic key, would further disrupt the statistical regularities exploited by IC-based scoring, and increasing the alphabet or extending the transposition key would add yet another layer of complexity. Taken together, these measures would inflate the search space to the point where a practical attack becomes essentially unattainable, even with modern computing resources.

Taken together, these insights show how closely theoretical design, practical implementation, and real-world usage are connected in classical cryptography. ADFGVX illustrates this relationship particularly clearly: a system that was technically advanced for its time ultimately became vulnerable because of how it was operated, not merely because of its mathematical structure. Working on this project made that interplay tangible, from studying the historical failures to implementing modern attacks and observing how small design or operational choices can dramatically change the difficulty of breaking the cipher. It is a reminder that the security of any cryptographic system depends as much on disciplined practice as on clever algorithms, and that even elegant designs can only remain secure when they are used with equal care.

References

[1] C.P. Bauer, *Secret history: The story of cryptology*, Discrete Mathematics and Its Applications, Taylor & Francis, 2013.

[2] Arush Chhatrapati, *On the construction and cryptanalysis of multi-ciphers*, Cryptology ePrint Archive, Paper 2021/1005, 2021.

[3] J. Rives Childs, *General solution of the adfgvx cipher system*, Aegean Park Press, Laguna Hills, California, 2000, Softcover, ISBN 10: 0894122843, ISBN 13: 9780894122842.

[4] Martine Diepenbroek, *From fire signals to adfgx: A case study in the adaptation of ancient methods of secret communication*, KLEOS - The Amsterdam Bulletin of Ancient Studies and Archaeology (2019), no. 2, 63–76 (English).

[5] William F. Friedman, *The index of coincidence and its applications in cryptanalysis*, Aegean Park Press, Laguna Hills, California, 1987, ISBN for soft cover: 0-89412-138-3 (library bound).

[6] H. F. Gaines, *Cryptanalysis a study of ciphers and their solutions*, Dover Publications, Inc., USA, 1989.

[7] David Kahn, *The codebreakers: The comprehensive history of secret communication from ancient times to the internet*, rev sub ed., Scribner, December 1996.

[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing*, Science **220** (1983), no. 4598, 671–680.

[9] Scott Kirkpatrick, *Optimization by simulated annealing: Quantitative studies*, Journal of Statistical Physics **34** (1984), no. 5, 975–986.

[10] Alan G Konheim, *Cryptanalysis of adfgvx encipherment systems*, Proceedings of CRYPTO 84 on Advances in Cryptology (Berlin, Heidelberg), Springer-Verlag, 1985, p. 339–341.

[11] George Lasry, *A methodology for the cryptanalysis of classical ciphers with search metaheuristics, kassel university press*, kassel university press, 2018 (english).

[12] George Lasry, Ingo Niebel, Nils Kopal, and Arno Wacker, *Deciphering adfgvx messages from the eastern front of world war i*, Cryptologia **41** (2017), no. 2, 101–136.

[13] Wenjie Li, *Experimental analysis of adfgvx cipher*, 2021 International Conference on Electronic Information Engineering and Computer Science (EIECS), 2021, pp. 77–80.

[14] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot, *Handbook of applied cryptography*, 1st ed., CRC Press, Inc., USA, 1996.

[15] Mauricio G. C. Resende and Celso C. Ribeiro, *Optimization by grasp: Greedy randomized adaptive search procedures*, 1st ed., Springer Publishing Company, Incorporated, 2016.

[16] Bart Selman and Carla P Gomes, *Hill-climbing search*, pp. 333–336, John Wiley & Sons, Ltd, 2006.

[17] Benjamin Snyder, Regina Barzilay, and Kevin Knight, *A statistical model for lost language decipherment*, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (Uppsala, Sweden) (Jan Hajič, Sandra Carberry, Stephen Clark, and Joakim Nivre, eds.), Association for Computational Linguistics, July 2010, pp. 1048–1057.

[18] D.R. Stinson and M.B. Paterson, *Cryptography: Theory and practice*, Chapman & Hall/CRC Cryptography and Network Security, CRC Press, Taylor & Francis Group, 2019.

[19] Isunuri Bala Venkateswarlu and Jagadeesh Kakarla, *Password security by encryption using an extended adfgvx cipher*, Int. J. Inf. Comput. Secur. **11** (2019), no. 4–5, 510–523.