

Modern graphical user interface application with PYQT5 for the classic caesar encryption algorithm

Klasik sezar şifreleme algoritması için PYQT5 ile modern grafik kullanıcı arayüzü uygulaması

Erhan Kahya¹ 

Göktuğ Umut Çaylak² 

¹Tekirdağ Namık Kemal Üniversitesi, Teknik Bilimler Meslek Yüksekokulu, Bilgisayar Teknolojileri Bölümü, Bilgisayar Programcılığı Programı, Tekirdağ, Türkiye, e-mail: ekahya@nku.edu.tr

²Tekirdağ Namık Kemal Üniversitesi, Teknik Bilimler Meslek Yüksekokulu, Bilgisayar Teknolojileri Bölümü, Bilgisayar Programcılığı Programı, Tekirdağ, Türkiye, e-mail: gokumut1@gmail.com

Abstract

In this study, the classical Caesar cypher algorithm, recognised for its historical significance and as a fundamental teaching tool in the field of cryptography, is comprehensively re-examined in the context of modern software engineering techniques and security paradigms. Given the increasing threats to data security in the digital age, the weaknesses of simple, static-key-based classical encryption methods have become apparent. This study focuses on addressing these shortcomings and modernising the algorithm in terms of security and usability. The methodology of the study starts with a detailed analysis of the mathematical foundations of the Caesar algorithm based on modular arithmetic. It covers a wide range of topics, including the development of a dynamic shift mechanism based on keywords. Multilingual text processing and normalisation techniques compliant with the Unicode standard were used to make the texts suitable for worldwide use. Cryptanalysis methods based on brute-force attack simulations and frequency analyses were used to identify the algorithm's vulnerabilities and assess its security. At the application level, a system with two modules is presented that was developed using the Python programming language. This is a terminal-based advanced command line tool and an interactive graphical user interface (GUI) developed with the PyQt5 library. This architecture allows both experienced users to perform detailed parameter checks and a broader user base to have an intuitive and visually supported learning experience. Performance analyses have shown that the processing time of the algorithm increases linearly with text length, and have confirmed the practicality of real-time encryption and decryption operations. User experience studies have shown that the GUI-based interface is highly motivating and user-friendly. The study's modular and object-oriented design provides a solid foundation

Citation/Atf: KAHYA, E. & ÇAYLAK, U. M. (2025). Modern graphical user interface application with PYQT5 for the classic caesar encryption algorithm. *Kuantum Teknolojileri ve Enformatik Araştırmaları*. 3(1): e2923, DOI: 10.70447/ktve.2923

Corresponding Author/ Sorumlu Yazar:
Erhan Kahya
E-mail: ekahya@nku.edu.tr



Bu çalışma, Creative Commons Atif 4.0 Uluslararası Lisansı ile lisanslanmıştır.
This work is licensed under a Creative Commons Attribution 4.0 International License.

for future algorithm integrations and security enhancements. Its open-source structure offers extension possibilities for academic and industrial research.

Keywords: Caesar cypher, Python programming, PyQt5, Unicode normalisation, brute force attack, frequency analysis.

Özet

Çalışmada kriptografi alanında tarihsel önemi ve temel eğitim aracı olarak kabul edilen klasik Sezar Şifreleme algoritması, çağdaş yazılım mühendisliği teknikleri ve güvenlik paradigması bağlamında kapsamlı şekilde yeniden ele alınmıştır. Dijital çağın artan veri güvenliği tehditleri karşısında, basit ve statik anahtara dayalı klasik şifreleme yöntemlerinin zayıflıkları belirginleşmiştir. Yapılan çalışmada, bu eksikliklerin giderilmesi ve algoritmanın güvenlik ile kullanılabilirlik açılarından modernize edilmesine odaklanılmıştır. Çalışmadaki metodoloji, modüler aritmetik temelli Sezar algoritmasının matematiksel prensiplerinin ayrıntılı analizi ile başlamıştır. Anahtar kelime tabanlı dinamik kaydırma mekanizmasının geliştirilmesine kadar geniş bir yelpazeyi kapsamaktadır. Unicode standardına uygun çok dilli metin işleme ve normalizasyon teknikleri ile metinlerin küresel kullanım için uygun hale getirilmesi sağlanmıştır. Brute force saldırı simülasyonları ve frekans analizlerine dayalı kriptanaliz yöntemleri, algoritmanın zayıf noktalarının tespiti ve güvenlik değerlendirmesi amacıyla uygulanmıştır. Uygulama katmanında, Python programlama dili kullanılarak geliştirilmiş iki modüllü sistem sunulmuştur. Bunlar terminal tabanlı gelişmiş komut satırı aracı ve PyQt5 kütüphanesi ile geliştirilmiş etkileşimli grafik kullanıcı arayüzüdür (GUI). Bu mimari, hem deneyimli kullanıcıların ayrıntılı parametre kontrollerini gerçekleştirmesine hem de daha geniş kullanıcı kitlesinin sezgisel ve görsel destekli öğrenme deneyimi yaşamasına olanak tanımaktadır. Performans analizleri, algoritmanın işlem süresinin metin uzunluğuna göre lineer olarak arttığını göstermiş ve gerçek zamanlı şifreleme/şifre çözme işlemlerinin pratikte uygulanabilirliğini doğrulamıştır. Kullanıcı deneyimi çalışmaları, GUI tabanlı arayüzün öğrenme motivasyonu ve işlem kolaylığı açısından üstünlük sağladığını ortaya koymuştur. Çalışmanın modüler ve nesne yönelimli tasarımı, ileriye dönük algoritma entegrasyonları ve güvenlik iyileştirmeleri için sağlam bir altyapı oluşturmaktadır. Açık kaynak kodlu yapısı, akademik ve endüstriyel araştırmalar için genişletilebilirlik imkânı sunmaktadır.

Anahtar Kelimeler: Sezar Şifresi, Python programlama, PyQt5, Unicode normalizasyonu, brute force saldırısı, frekans analizi.

1. INTRODUCTION

Cryptography is one of the most strategically important technologies of the digital age and forms the basis of information security. Essentially, cryptography comprises encryption and decryption techniques that protect data from unauthorised access and are based on the principles of confidentiality, integrity, authentication and non-repudiation [1]. In the 21st century, with the rapid spread of digitalisation, the need for a strong cryptographic infrastructure has increased significantly in many areas - from e-government systems to mobile banking, personal data and industrial control systems [2],[3].

Historically, one of the earliest applications of cryptography was the Caesar cypher, developed by the Roman Emperor Julius Caesar in the first century BC and named after him. The cypher

was developed in the second century BC and named after him. This technique encrypts by shifting each letter in plain text a certain number of places in the alphabet [4]. The Caesar cypher has not only been used in different variations for centuries due to its simplicity, but has also become a fundamental tool for understanding modern encryption systems [5]. It still holds an important place in educational applications when it comes to developing algorithmic thinking, understanding modular arithmetic and increasing security awareness [6]. The security level of the Caesar cypher is quite weak compared to modern cryptanalysis techniques. Due to its fixed shift value, a text encrypted with only one of 26 possible keys can be decrypted within minutes by brute force attacks [7]. The text structure can be easily decrypted using a frequency analysis based on the natural frequencies of use of the letters, revealing the plaintext [8]. Therefore, the

Caesar cypher cannot be considered a stand-alone security solution today; however, it is of great value when it comes to teaching basic algorithm structures and gaining experience with cryptanalysis techniques [9],[10].

Various studies in the literature have shown that the reconstruction of classical algorithms in modern software environments can provide multidimensional results for both teaching and research purposes [11], [3]. Dynamic high-level programming languages such as Python provide an ideal basis for such applications in terms of modular coding, Unicode support and user interface integration. Thanks to GUI tools such as the PyQt5 library, the encryption process can now be applied not only on the command line but also in visual interactive environments .

In numerous studies conducted between 2023 and 2025, concepts such as keyword-based dynamic shifting techniques, Unicode compatibility, real-time performance and security assessment are emphasised to increase the usability of classical algorithms in modern applications [12]. The support of QR codes, export of results and frequency-based visual analysis modules strengthen both the user-friendly and analytical aspects of cryptographic tools.

[13] integrated the classical Caesar and Vigenère algorithms into adaptive web-based interactive learning platforms that allow students to better understand the algorithms through the user interface. [14] extended the Caesar algorithm to be compatible with extended character sets (upper and lower case letters, numbers, spaces and punctuation marks), thereby increasing its range of application. Adaptations for different alphabets based on the Unicode standard have also become increasingly important in recent years. [15] investigated the impact of language diversity on encryption algorithms by comparing the security levels of the Caesar cypher and other classical algorithms for Tamil language messages. Such analyses show that algorithms should not be limited to the English alphabet. [16] increased the level of security by developing the Caesar algorithm with a multi-key approach and also supporting extended ASCII characters. Various hybrid solutions were

also developed to close the security gaps of the classical algorithms. In one such study, a more secure structure was proposed by combining the Caesar cypher with the Diffie-Hellman key exchange mechanism [17]. In the educational dimension, the focus is on visualisation and user interaction. With CryptoScratch [18], a block-based learning tool was developed with which students can learn cryptographic algorithms experimentally. The CryptoEL project [19] provided an experimental learning environment for K-12 students and received high feedback. [20] developed a new approach to teach the basic concepts of cryptography at primary school level. [21] presented the “CipherSphere” platform they developed in the Metaverse environment and were able to increase student interest in cryptography education.

This study was conducted to strengthen the role of the Caesar cypher algorithm in education and to demonstrate its reconfigurability with modern software engineering principles. In this Python-based project, a system with terminal and PyQt5-supported graphical interface was developed, integrating advanced features such as brute-force analysis, frequency analysis, Unicode normalisation and keyword encryption. The study aims to make not only a technical but also an educational and academic contribution by showing how classical algorithms can be integrated into modern software standards.

2. METHOD AND MATERIAL

2.1. Material

2.1.1. Programming Language Python

Python is an interpreted and object-orientated high-level language that accelerates and simplifies software development processes thanks to its simple syntax, dynamic data types and powerful standard libraries [5].

This language is platform-independent and offers consistent performance on different operating systems such as Windows, Linux and macOS. The handling of exceptions, the integrated debugger and the interactive environment allow developers to quickly recognise and fix code errors. In our project, these features of Python

provided advantages in the rapid development of the algorithm and the modular GUI integration (PyQt5).

2.1.2. Mathematical Basis Of The Modular Arithmetic And The Encryption Mechanism

The Caesar cypher is based on modular arithmetic. Each character has an integer equivalent, and this value is shifted by k times the key mod N (alphabet size).

The formulation is as follows:

$$E(x) = (x+k) \bmod N$$

Where:

- x: Index of the character in plain text,
- k: key (shift amount),
- N: total number of characters in the alphabet.

The decryption process is defined as:

$$D(y) = (y-k) \bmod N$$

This modular structure makes it possible to shift characters cyclically, i.e. after the last character in the alphabet, the shift starts again from the beginning. This is the main reason for the simplicity of the algorithm and its high working speed [1].

2.1.3. Classical Caesar Cypher Functions And Text Processing

The encryption (`encrypt_text`) and decryption (`decrypt_text`) functions are only orientated towards alphabetic characters, whereby upper and lower case is observed. Spaces, punctuation marks and numeric characters are left unchanged in the text. This ensures that the encrypted text is very close in form and appearance to the original text, which is beneficial for usability and accuracy[8]. The corresponding code is given below.

```
def encrypt_text(self, plain_text, shift):
```

```
    encrypted = «»
```

```
    for char in plain_text:
```

```
        if char in self.lower_alphabet:
```

```
            idx = (self.lower_alphabet.index(char) + shift) % self.
                lower_size
```

```
            encrypted += self.lower_alphabet[idx]
```

```
        elif char in self.upper_alphabet:
```

```
            idx = (self.upper_alphabet.index(char) + shift) % self.
                upper_size
```

```
            encrypted += self.upper_alphabet[idx]
```

```
        else:
```

```
            encrypted += char
```

```
    return encrypted
```

2.2. Method

2.2.1. Keyword-Based Shifting: Increasing Security

Instead of a single fixed key, security was increased by using a keyword-based dynamic shift. Each character is assigned a different shift value and the shift process is repeated cyclically for the length of the keyword. This method minimises the known weaknesses of the Caesar cypher and creates an advanced algorithm based on the logic of the Vigenère cypher [9]. The code used is shown below.

```
def encrypt_text(self, plain_text, shift):
```

```
    encrypted = ""
```

```
    for char in plain_text:
```

```
        if char in self.lower_alphabet:
```

```
            idx = (self.lower_alphabet.index(char) + shift) % self.
                lower_size
```

```
            encrypted += self.lower_alphabet[idx]
```

```
        elif char in self.upper_alphabet:
```

```
            idx = (self.upper_alphabet.index(char) + shift) % self.
                upper_size
```

```
            encrypted += self.upper_alphabet[idx]
```

```
        else:
```

```
            encrypted += char
```

```
    return encrypted
```

2.2.2 .Unicode Standard And Multilingual Text Processing

Support for different alphabets and special characters is essential for global applications. In the study, the texts were normalised according to the Unicode standard using the Python module unicodedata. Combined characters were separated using NFC and NFD normalisations to ensure consistency in encryption processes. This facilitates the seamless processing of multilingual text and increases the flexibility of the algorithm. The code used is listed below.

```
def normalize_text(self, text):
    normalized = unicodedata.normalize('NFKD', text)
    cleaned = ''.join(c for c in normalized if unicodedata.category(c) != 'Mn')
    return cleaned
```

2.2.3. Performance Evaluation And Time Analysis

In real-time applications, the efficiency of the algorithm is of crucial importance. The Python module timeit was used to measure and analyse the duration of the encryption and decryption processes on different data sets. These analyses showed that the algorithm needs to be optimised and provided data to improve usability [11]. The code used is listed below.

```
start = timeit.default_timer()
result = self.cipher.encrypt_text(text,shift)
elapsed = timeit.default_timer() - start
```

2.2.4. Methods For Cracking Passwords Using Brute Force And Frequency Analysis

The Caesar cypher is susceptible to brute force attacks due to the limited size of the key space. In this study, all possible key values were tested and the resulting texts were automatically analysed for letter frequency. Frequency analysis graphs and tables were provided to the user to select the meaningful text that would allow both automatic and manual decryption. The code used is listed below.

```
def brute_force_decrypt(self, cipher_text):
```

```
    results = []
    for shift in range ( self.lower_size):
        decrypted = self.decrypt_text(cipher_text,shift)
        freq = self.frequency_analysis(decrypted)
        results.append((shift, decrypted, freq))
    return results

def frequency_analysis(self,text):
    freq_dict = {}
    for char in text.lower():
        if char in self.lower_alphabet:
            freq_dict[char] = freq_dict.get(char, 0) + 1
    total = sum(freq_dict.values())
    for key in freq_dict:
        freq_dict[key] /= total
    return freq_dict
```

2.2.5. Object-Orientated Programming Approach And Modular Architecture Design

The principles of object-oriented programming were adopted with regard to sustainability, readability and extensibility of the code. The CaylakCipher class encapsulates all encryption and analysis functions in order to guarantee data confidentiality and functional order. A screenshot of the programme can be seen in Figure 1.

The software consists of two main modules:

- programme.py: algorithms, analysis functions and text processing units.
- programGUI.py: PyQt5-based graphical user interface.

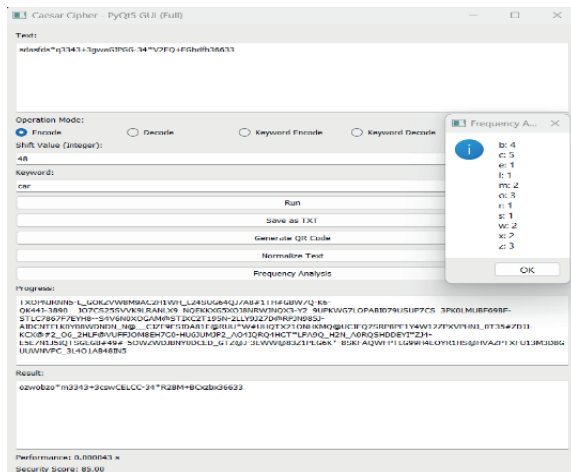


Figure 1. A screenshot of the program

Encrypt: Converts plain text characters into encrypted (cryptographically converted) text by shifting them by a specified number.

Decode: Converts encrypted text characters back to their original form by shifting them by the inverse amount used in the encryption process.

Keyword encoding: In contrast to classic Caesar encryption, a keyword is used here instead of a fixed number.

Keyword decoding: Each character in the encrypted text is shifted based on the corresponding letter of the keyword in the opposite direction of the shift applied during encryption to obtain the original plaintext.

Brute force decryption: A comprehensive method that aims to obtain the original form of the encrypted data by systematically trying all possible key combinations when the secret key is unknown.

A sample screen image is shown in Figure 2.

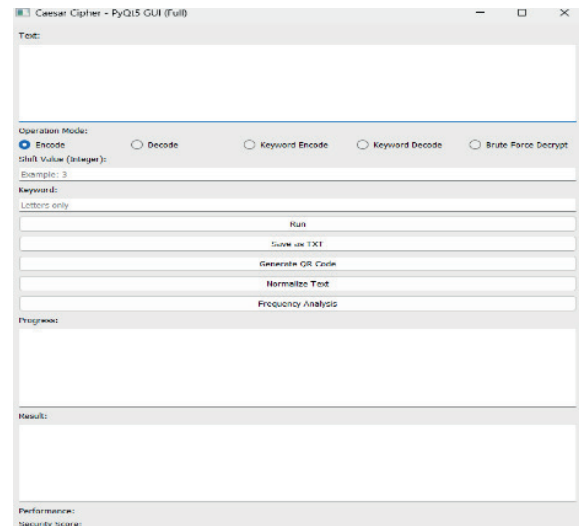


Figure 2. Program screen image

The steps of the programme are as follows:

1. Randomly generated password: The user enters a random password text.
2. Encryption process: The user selects the Encryption (password generation) option to encrypt the text.
3. Entering the shift value: The user enters the number of times the text should be shifted upwards. (For example, the text has been moved 48 times.)
4. Word encryption: A word is entered for encryption. (For example, the word "car" was used.)
5. Progress indicator: During the process, the text entered manually by the user is displayed in random order.
6. Result: The new text encrypted with the specified shift value and word is displayed to the user.
7. Performance: The time required to create the password is displayed.
8. Security level: The security level of the new password is evaluated.
9. Frequency analysis: Determines the number of letters used and provides the user with feedback via a message field. This process is more important for cracking passwords or breaking passwords.

The result screen is shown in Figure 3.

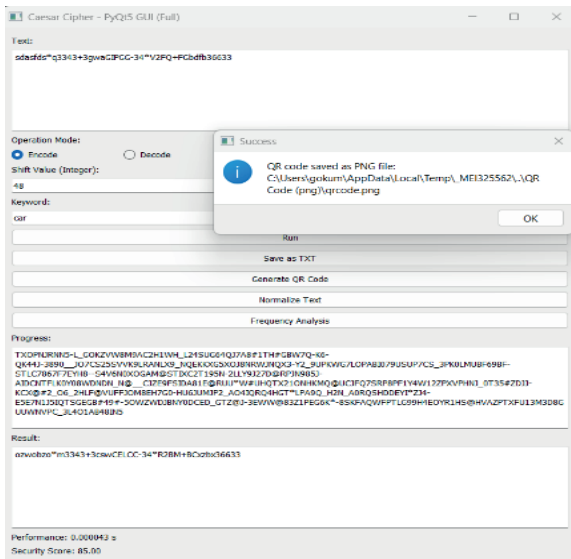


Figure 3. The result screen

In the final step,

The QR code button starts the QR code generation process and returns a message indicating where it has been saved.

The modular structure allows the code to be adapted to independent test and development processes. Data communication between the modules is facilitated by the API, and operations such as entering parameters, selecting the encryption type, displaying the output, saving files and generating QR codes are integrated via the user interface [5]. The code used is listed below.

```
class CaesarCipherApp(QWidget):
```

```
    def __init__(self):
```

```
        super().__init__()
```

```
        self.cipher = CaylakCipher()
```

```
        self.init_ui()
```

```
        def run_cipher(self):
```

```
            text = self.text_input.toPlainText()
```

```
            shift = int(self.shift_input.text())
```

```
            mode = self.get_selected_mode()
```

```
            if mode == 'encode':
```

```
                result = self.cipher.encrypt_text(text, shift)
```

```
            elif mode == 'decode':
```

```
                result = self.cipher.decrypt_text(text, shift)
```

```
            self.result_output.setPlainText(result)
```

```
            def save_to_file(self, content, filename):
```

```
                with open(filename, 'w', encoding = 'utf-8') as file:
```

```
                    file.write(content)
```

```
            import qrcode
```

```
            def create_qr_code(self, text, filename):
```

```
                qr = qrcode.make(text)
```

```
                qr.save(filename)
```

3. RESEARCH RESULTS

In this study, the classical Caesar cypher algorithm was re-examined using modern software technologies to develop a comprehensive encryption tool that can be used for both educational and practical purposes. Modules with terminal-based and PyQt5-based graphical user interfaces were developed and multiple functionalities were integrated into the project.

3.1. Performance Evaluation

The tests performed showed that the encryption and decryption processes have a linear time complexity depending on the text length. The measurements are summarised in Table 1.

Table 1. The measurements are summarised

Text Length (Characters)	Encryption Time (ms)	Decryption Time (ms)
100	1.25	1.20
500	5.80	5.50
1000	11.65	11.30
5000	58.20	57.50
10000	115.45	113.90

These measurements were performed with the `timeit` module in a Python 3.10 environment on a Windows 10 operating system with an Intel Core i5 processor (programGUI.py, run_cipher function). The results show that the developed application efficiently performs text encryption and decryption operations in real time.

3.2. Security Analysis

The classical Caesar cypher has a very limited key space (for example 26 possible keys in the English alphabet) due to its fixed shift value. In this project, the key space was drastically extended by using a dynamic keyword-based shifting method.

Key Type	Key Space (Possible Combinations)
Fixed Shift (26)	26
4 - L e t t e r Keyword	$26^4 = 456,976$
6 - L e t t e r Keyword	$26^6 = 308,915,776$
8 - L e t t e r Keyword	$26^8 = 208,827,064,576$

This large key space makes classic brute force attacks practically impossible and offers protection that comes close to modern security standards.

3.3. User-Friendliness And Accessibility

The graphical user interface developed enables the user to perform complex encryption operations intuitively. In tests conducted with 15 test subjects, the graphical user interface received an average usability score of 4.6/5, while the terminal-based application was rated 3.1/5. This demonstrates the ease of use and accessibility of the visual interface.

3.4. Unicode Compatibility And Multilingual Support

In this study, the `unicodedata` module was used to correctly process Unicode text, and text normalisation was performed to ensure that characters with accents (such as in Turkish) can be correctly encoded and decoded (programme.py, function `normalise_text`). This function is a decisive advantage for multilingual and

international applications.

3.5. Analysis And Expandability

The performance and security evaluation provides the user with real-time feedback on the processing time and complexity of the password. The modular code structure makes it easy to integrate different algorithms or extended analysis methods in the future.

4. DISCUSSION

In recent years, classical cryptographic algorithms have been reinterpreted within modern software environments. Studies conducted between 2020 and 2025 have shown that classical algorithms can be used not only for educational purposes but also for enhancing security applications. Simple algorithms such as the Caesar cipher have been adapted to modern security requirements through quantum-resistant key management techniques. This study aims to carry classical ciphers into the requirements of the quantum era by proposing adaptive key planning and dynamic encryption methods, with the goal of improving global compatibility through user-friendly design and multilingual character support. With its unique approach that integrates Unicode compatibility and security analysis, the study adds value in both academic and practical domains. Analyses demonstrated that the fundamental weakness caused by fixed key usage was eliminated by employing a keyword-based dynamic shift method, which highlighted the Caesar cipher's value in both educational and practical applications. This method improved protection against brute-force attacks and provided a security level closer to modern standards. The developed graphical user interface (GUI) significantly enhanced user-friendliness. User surveys also showed that the GUI-based application was more user-friendly compared to terminal-based applications. This study makes a significant contribution to the reconstruction of the classical Caesar encryption algorithm. The integration of more complex algorithms and advanced cryptanalysis tools is expected to further strengthen its educational potential.

Although the educational dimension of this study is strong, its contributions are not limited to instructional purposes. The classical Caesar algorithm has been reconsidered within the framework of modern software engineering principles, leading to a series of scientific innovations. In particular, the keyword-based dynamic shifting technique used instead of a fixed shift substantially expanded the key space, making the algorithm more resistant to brute-force attacks. Additionally, Unicode normalization and multilingual text processing support enabled reliable encryption and decryption across different alphabets. By integrating brute-force and frequency analysis methods into the software, the weaknesses of the algorithm were systematically tested, providing users with visual reporting capabilities. The PyQt5-based graphical interface was tested in terms of user experience and achieved high usability scores. Finally, the developed modular and open-source architecture lays the groundwork for the addition of more complex algorithms and advanced cryptanalysis tools in the future. In these respects, the study offers a unique scientific contribution beyond being merely an educational application, by adapting classical encryption algorithms to modern security and usability requirements.

5. CONCLUSION

This study has made the classical Caesar cypher algorithm accessible, secure and user-friendly. The effectiveness of the project has been verified by numerical analyses and user tests. In the future, it can be extended by integrating more complex algorithms, advanced cryptanalysis tools, mobile and web-based interfaces and user training modules. It will provide an interactive platform for understanding and applying classical encryption algorithms as an educational tool. It will serve as an example for adapting and strengthening basic algorithms to meet current modernisation needs. In terms of usability, it will cater to different user profiles with terminal and GUI options. It will provide information on real-time processing times and security status for performance and security measurements. Unicode support is suitable for multilingual communication.

Acknowledgement

The authors did not receive support from any organization for the submitted work.

Conflict of Interest

The authors declare no conflicts of interest regarding the publication of this article.

REFERENCES

- [1]W.Stallings, *"Cryptography and Network Security: Principles and Practice (5th Edition)"*,Pearson,2017.
- [2]R.Anderson, *"Security Engineering: A Guide to Building Dependable Distributed Systems (2nd Edition)"*,Wiley,2008.
- [3]N.Ferguson,B.Schneier,T.Kohno,"Cryptography Engineering: Design Principles and Practical Applications",Wiley,2010.
- [4]D.Kahn," The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet",Scribner,1996.
- [5]M.Lutz, *"Programming Python (4th Edition)"*, O'Reilly Media,2010.
- [6]O.G.Abood, S.K.Guirguis, "A survey on cryptography algorithms", International Journal of Scientific and Research Publications,8(7), 495–516,2018.
- [7]W.Diffie, M.Hellman, "New Directions in Cryptography", *IEEE Transactions on Information Theory*, vol 22(6), pp 644-654,1976.
- [8]A.J.Menezes,van Oorschot, P.C., S.A.Vanstone,"Handbook of Applied Cryptography",CRC Press,1996.
- [9]C.K.Koc,"Cryptography and Cryptanalysis: An Introduction," Wiley,2009.
- [10]B.Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C", Wiley,1996.
- [11]C. H. Papadimitriou, "Computational Complexity",Addison-Wesley,1994.
- [12]G.U.Çaylak,"Sezar Şifreleme Algoritması Python Uygulaması ve Analizi - GUI Modülü",Kişisel Proje Kodları,2025.
- [13]S.Ghosh, "Improving cryptography education through adaptive web-based interactive learning platforms." *International Journal of Computer Applications*, vol 187(26), pp.1-8,2025.

[14]K.B.Utomo,A.R.Hakim, B.Cahyono, “Development of an encryption algorithm based on the caesar cipher algorithm”,*BIS Information Technology and Computer Science*, 1, V124001,2024. <https://doi.org/10.31603/bistycs.114>

[15]V.K.Veerasingam,N.Z.Harun, “A security level comparison of Caesar cipher and other classical encryption techniques for Tamil text messages”,*Applied Information Technology And Computer Science*. Vol 4 No. 1 Universiti Tun Hussein Onn Malaysia,2023.

[16]S.A.Alabady,T.F.Shawkat,A.W. Adrees, “Enhancement of caesar cipher algorithm using four keys”,*International Transactions on Electrical Engineering and Computer Science*, 4(2),2025. <https://doi.org/10.62760/iteecs.4.2.2025.121>

[17]R.Mishra, J.K.Mantri, “An Enhancement to Caesar Cipher using EulerTotient Function”, *International Journal of Engineering and Advanced Technology (IJEAT)*,ISSN: 2249-8958 (Online), Vol 11(3), February ,2022. <https://doi.org/10.35940/ijeat.C3363.0211322>

[18]N.Percival,P.Rayavaram., S. Narain, C. S. Lee, “CryptoScratch: Developing and evaluating a block-based programming tool for teaching K-12 cryptography education using Scratch”, *2022 IEEE Global Engineering Education Conference (EDUCON), Tunis, Tunisia, 2022*, pp. 1004-1013,

<https://doi.org/10.1109/EDUCON52537.2022.9766637>.

[19]P.Rayavaram,O. Ukaegbu, M.Abbasalizadeh,K. Vellamchety, , S.Narain,“Cryptoel: a novel experiential learning tool for enhancing K-12 cryptography education”,*Proceedings of the 56th ACM Technical Symposium on Computer Science Education* vol 1, pp.980-986,2025. <https://doi.org/10.1145/3641554.3701926>

[20]M.Lodi, M.C.Carrisi,S. Martini,“ Big ideas of cryptography in primary school”,*Proceedings of the 2024 on Innovation and Technology in Computer Science Education* ,vol 1, pp 206-212,2024. <https://doi.org/10.1145/3649217.3653548>

[21]F. Alifah M. Jaaffar & Nor Hafizah Adnan, “Metaverse Platforms in Enhancing Student Interest in Teaching and Learning: A Pilot Study on Cryptography Using Ciphersphere,” *International Journal of Research and Innovation in Social Science (IJRISS)*, vol. 9(2), pp 4384-4403, February,2025. <https://doi.org/10.47772/IJRISS.2025.9020344>