OmniaScience

Journal of Technology and Science Education
JOTSE, 2019 – 9(3): 257-268 – Online ISSN: 2013-6374 – Print ISSN: 2014-5349
https://doi.org/10.3926/jotse.491

A NEW CHARACTER-LEVEL ENCRYPTION ALGORITHM: HOW TO IMPLEMENT CRYPTOGRAPHY IN AN ICT CLASSROOM

David Arboledas-Brihuega

Departamento de Tecnología, IES Profesor Domínguez Ortiz (Spain)

darboledas@edu.jccm.es

Received June 2018
Accepted February 2019

Abstract

It is evident that the society in which we live will demand more and more qualified and specialized positions in the different branches of engineering. Now we are in a highly digitized world in which information is continuously transmitted through data communication networks with the expectation of security and confidentiality. Students who are in their last year at high school face the problem of deciding their professional future. Therefore, there is a wide field of research for them in cryptographic techniques. In this work we have developed, together with a group of high school students, a cryptographic algorithm with substitution and transposition techniques as described by Feistel (1973) in order to create a more comprehensive knowledge about what they have been studying in their ICT subjects.
In most cases the teaching methods are based on the teachers' own vision, as well as on the absence of knowledge of alternative methods and/or the impossibility of applying them physically in the classroom.
With the active and cooperative methodology put forward in this work, objectives such as hard work, autonomous and collaborative learning and the exchange of knowledge have been absolutely fulfilled. Achieving them requires replacing traditional methods so that the student can adapt to new work challenges.
A group of students, only three of whom finished, were voluntarily provided with the mentoring service in which the algorithm was designed. As a result, we were able to program in Python as a final project a symmetrical character-level encryption algorithm we've referred to as Azrael.
Our findings indicate a demand for future endeavours to take into account the need for more project-based work in ICT classrooms. The exchange of ideas between teacher and students has been the driven force behind the success of this activity.

Keywords **–** Cryptography, Azrael, Symmetrical character-level encryption algorithm, ICT, Substitution-permutation network, Student-centred methodologies.

To cite this article:

Arboledas-Brihuega, D. (2019). A new character-level encryption algorithm: How to implement cryptography in an ICT classroom. *Journal of Technology and Science Education,* 9(3), 257-268. https://doi.org/10.3926/jotse.491

---

1. Introduction

The protection of information has garnered much attention in recent years despite being a concern of companies and governments since time immemorial. Cryptography is an art and science. It is a playing major role in every information and security division. The main aim of the cryptography is protecting the data from unauthorized users (Zaidan et al., 2010). Encryption techniques occur or used by using shifting -positions held by units of plaintext are shifted according to a regular system- and substitution -units of plaintext are replaced with ciphertext, according to a fixed system- techniques, as well as mathematical operations (Hannan & Asif, 2017).

The modern design of symmetric encryption algorithms is block ciphers –encrypt a group of plaintext symbols as one block– and is based on the concept of iterative product ciphering (Bard, 2009). Shannon (1949) analysed product encryption and suggested that improving security involved replacement and transposition operations. Iterative product ciphers perform the encryption process in multiple stages, known as rounds, each of which uses a different subkey derived from the original key (Even & Goldreich, 1985). Iterative product encryption algorithms are based on the concepts of confusion (trying to hide the relationship between clear text, ciphertext and key by substitution) and diffusion (trying to spread the influence of each symbol of the original message among the ciphered message by permutations), which are combined to give rise to so-called product ciphers (Biham & Dunkelman, 2012).

These techniques basically consist of splitting the message into blocks of a certain size and applying the encryption function to each of them. Product encryptions using only substitutions and permutations are called substitution-permutation networks (Feistel, 1973). A substitution-permutation (SP) network takes a block of plain text and a key and applies several rounds of substitution transformations followed by transposition operations (Shannon, 1949).

The basic components of these symmetric key algorithms are the S-boxes (substitution boxes) and the P-boxes (permutation boxes). An S-box replaces a small block of text with another in a one-to-one transformation. A secure S-box should ensure that changing a single input symbol changes at least half the output symbols (Hoffstein, Pipher & Silverman, 2008). A P-box performs a permutation or transposition of all symbols and feeds the S-boxes of the next round. A secure P-box will have the property that the output symbols of any S-box are distributed in the next round among the largest number of S-boxes (Hoffstein et al., 2008).

The transformation process in each round is controlled with a subkey derived from the original key (Figure 1).

This paper presents how an algorithm using a SP network has been implemented in Python. Finally, in order to determine the degree of student satisfaction, they were given a final survey. From their answers it can be deduced that the students have recognised the extra effort it has meant for the teacher and themselves to achieve the objective proposed.

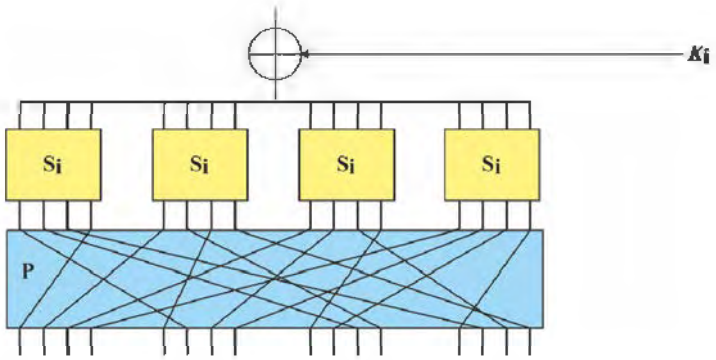

Figure 1. General scheme of a SP network

2. AIMS

Taking as our starting point the knowledge acquired in the last two years of computer science (on classical substitution and transposition techniques), we have successfully designed a symmetric key block cipher unit that we named Azrael that operates in groups of symbols of variable length, called blocks, applying an invariant transformation to them. When encrypting, Azrael takes a block of plain text as input and yields a block of the same size of encrypted text. The exact transformation is controlled by a second input, the secret key.

3. Design

The designed algorithm is a symmetrical encryption cipher implemented to use dynamic substitution and permutation boxes. This means that the blocks into which the plain text or the cryptogram is divided are variable in length.

The algorithm takes as input a text of length *n* and divides it into blocks of equal size as the key used. The default key is a random string of ten symbols. Interestingly, each time the program is run, a new password is generated, so you are guaranteed not to repeat the same key again. That key is then fragmented into two subkeys (*K*1 and *K*2), so that they are used to divide each block into two parts (*Lo*, left and *Ro*, right) of the same length as the subkeys.

Each subkey will operate in one transposition and one substitution over *Lo* and *Ro* in each input block. Then, *L* and *R* fragments will rotate twice to ensure that the *K*1 and *K*2 subkeys act on both parts.

3.1. First Round

Round 1 consists on taking *Lo* to be subjected to a permutation and *Ro* to a Vigenère substitution (Hannan & Asif, 2017; Arboledas-Brihuega, 2017), both controlled by subkey *K*1 (Figure 2, Stage 1).

3.2. Second Round

Round 2 consists on a substitution over *L*1 and a permutation over *R*1, both controlled by the subkey *K*2. Now, both fragments exchange positions before entering the following boxes (Figure 2, Stage 2).

3.3. Third Round

In the third round, the fragment *L*2 is subjected to a transposition and the fragment *R*2 to a Vigenère substitution, both ruled by the subkey *K*1 (Figure 2, Stage 3).

3.4. Fourth Round

The fourth round involves a replacement of fragment *L*3 and a permutation of fragment *R*3, managed by subkey *K*2, just before undergoing a new exchange of positions and regenerating the block again, which will already have the same length as the original password used (Figure 2, Stage 4).

The whole regenerated block is then subjected to a new substitution, managed by the original key, to introduce further confusion (Figure 2, last substitution).

Finally, to avoid that two identical blocks of plain text can be encrypted in the same way, a transposition, controlled by the entire original key, is carried out on the complete cryptogram, thus achieving the greatest possible diffusion in the encrypted message (Figure 2, last transposition).

The result is a cryptogram with enough confusion and diffusion so that obtaining plain text without the original password is not trivial. The diagram of the algorithm implemented in Python to encrypt a plain text is as follows (Figure 2).

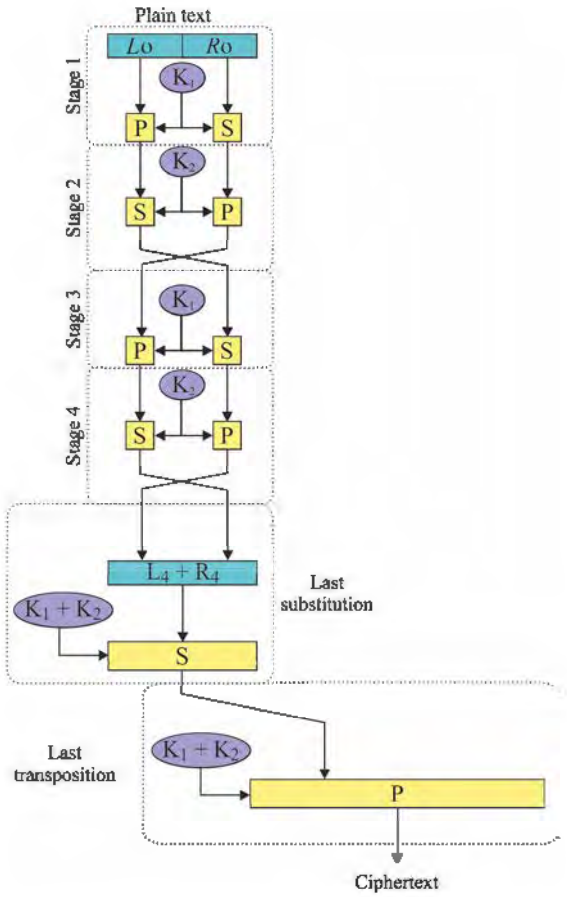

Figure 2. Azrael algorithm encryption scheme

4. Source Code to Encrypt

The source code of the program *azrael.py*, which works according to the scheme in Figure 2, is as follows:

```
# Implementation of a Dynamic Boxes Encryption System (DBES) 'Azrael'
# David Arboledas-Brihuega
# Feel free to use it and break it!

from math import ceil
import S_Box
import P_Box
import random
import pyperclip
ALPHABET='abcdefghijklmnopqrstuvwxyz0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
PASS_LEN = 10

def main():
    num = 0
    key = ''
    message = input('Plaintext: ')

    while num <= PASS_LEN:
        symbol = random.choice(ALPHABET)
        if symbol not in key:
            key = key + symbol
        num = len (key) +1
    password = key
    key1 = password[0:PASS_LEN // 2]  # subkey K1
    key2 = password[PASS_LEN // 2:PASS_LEN]  # subkey K2
```

```
    Boxes = generate_boxes(message)
    message = S_andP_Boxes(Boxes, len(Boxes), key1, key2)  # Rounds
    message = substitution(message, password)  # Last S-Box
    message = permutation(message, password)  # Last P-Box

    print('Ciphertext: ', '\n', message)
    print('Key: ', '\n', key)

    pyperclip.copy(message)


def fill_box(last_box):  # Fill the last box with Ç if necessary
    gaps = PASS_LEN - len(last_box)
    last_box += 'Ç' * gaps
    return last_box


def generate_boxes(message):
    Boxes = []
    number_of_boxes = ceil(len(message) / PASS_LEN)

    for i in range(number_of_boxes):
        inf = PASS_LEN * i
        sup = PASS_LEN * (i + 1)
        Boxes.append(message[inf:sup])

    Boxes[-1] = fill_box(Boxes[-1])

    return Boxes


def S_andP_Boxes(Boxes, number_of_boxes,key1,key2):
    # Round1
    message_Round1 = []

    for j in range(number_of_boxes):
        first = Boxes[j][0:PASS_LEN // 2]  # Box first half
        second = Boxes[j][PASS_LEN // 2:PASS_LEN]  # Box second half
        message_Round1.append(permutation(first, key1))
        message_Round1.append(substitution(second, key1))

    # Round2
    message_Round2 = []

    for j in range(0, number_of_boxes):
        first = message_Round1[2 * j]
        second = message_Round1[2 * j + 1]
        message_Round2.append(substitution(first, key2))
        message_Round2.append(permutation(second, key2))

    message = swapp(message_Round2, Boxes)

    # Round3
    message_Round3 = []

    for i in range(number_of_boxes):
        message_Round3.append(permutation(message[2 * i], key1))
        message_Round3.append(substitution(message[2 * i + 1], key1))

    # Round4
    message_Round4 = []

```

```
88      for j in range(0, number_of_boxes):
89          message_Round4.append(substitution(message_Round3[2 * j], key2))
90          message_Round4.append(permutation(message_Round3[2 * j + 1], key2))
91
92      message = swapp(message_Round4, Boxes)
93
94      return ''.join(message)
95
96
97  def substitution(message, password):
98      return S_Box.method(password, message, 'encrypt')
99
100
101 def swapp(message, Boxes):
102     for i in range(len(Boxes)):
103         message[2 * i], message[2 * i + 1] = message[2 * i + 1], message[2 * i]
104     return message
105
106
107 def permutation(message, password):
108     return P_Box.encrypt(message, password)
109
110
111 if __name__ == '__main__':
112     main()
```

4.1. The Module S_Box.py

```
1   # This module encrypts or decrypts substitution boxes using Vigenère algorithm
2   # Usage: S_Box(key, message, ['encrypt'/'decrypt'])
3
4   LETTERS = r""" 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzÇ"""
5
6
7   def method(key, message, use):
8       translated = []  # stores the S_Box string
9
10      key_index = 0
11
12      for symbol in message:  # loop through each symbol in message
13          num = LETTERS.find(symbol)
14
15          if num != -1:  # symbol found in LETTERS
16              if use == 'encrypt':
17                  num += LETTERS.find(key[key_index])
18              else:  # decrypt
19                  num -= LETTERS.find(key[key_index])
20              num %= len(LETTERS)  # handle the potential wrap-around
21
22              # add the encrypted symbol to the end of translated
23              translated.append(LETTERS[num])
24              key_index += 1  # move to the next letter in the key
25
26              if key_index == len(key):
27                  key_index = 0
28
29      return ''.join(translated)
```

4.2. The Module P_Box.py

```
# This module jumbles up the symbols with a columnar transposition.
# The permutation is defined by the alphabetical order of the symbols in the keyword.
# Usage: P_Box.[encrypt/decrypt](message, key)

from math import ceil


def encrypt(message, key):
    limit = ceil(len(message) / len(key))
    # Each string in ciphertext represents a column
    ciphertext = [''] * len(key)
    for col in range(len(key)):  # Loop through each column in ciphertext
        pointer = col
        while pointer < len(
                message):  # Keep looping until pointer goes past the length of the message
            # Place the character at pointer in message at the end of the
            # current column in the ciphertext list.
            ciphertext[col] += message[pointer]
            pointer += len(key)  # Move pointer over
    return order(ciphertext, key, 'encrypt')


def order(ciphertext, key, use):  # Order columns in grid by ASCII
    translated = []  # stores the P_Box string
    order_key = sorted(key)
    for i in range(len(key)):
        symbol = order_key[i]
        if use == 'encrypt':
            translated.append(ciphertext[key.index(symbol)])
        else:
            symbol = key[i]
            translated.append(ciphertext[order_key.index(symbol)])
    # Convert the translated list into a single string value and return it.
    return ''.join(translated)


def decrypt(ciphertext, key):
    return permutation_decipher(decipher(ciphertext, key), len(key))


def permutation_decipher(ciphertext, key):
    # Number of columns in the matrix
    col_number = ceil(len(ciphertext) / key)
    row_number = key  # Number of rows
    empty_cell = (col_number * row_number) - \
        len(ciphertext)  # Number of empty cells
    plaintext = [''] * col_number  # Each text string is a column
    col = 0
    row = 0
    for symbol in ciphertext:
        plaintext[col] += symbol
        col += 1  # Next column
        # If there are no more columns or it is an empty cell, we go back to
        # the first column of the next row
        if (col == col_number) or (col == col_number -
                                   1 and row >= row_number - empty_cell):
            col = 0
            row += 1
    return ''.join(plaintext)


```

```
62  def decipher(ciphertext, key):
63      col_number = ceil(len(ciphertext) / len(key))
64      text = []
65      block = len(key)
66      items = col_number
67      for i in range(0, len(ciphertext) + 1, items):
68          text.append(ciphertext[i:i + items])
69      return order(text, key, 'decrypt')
```

5. Encrypting with Azrael

Once the program and its two modules just described above are written (downloadable from http://bit.ly/azraelcode), it is only needed to run the module *azrael.py* and enter the plaintext. The key will be randomly chosen and the ciphertext is then obtained:

Plaintext: In a village of La Mancha the name of which I have no desire

Ciphertext:

aPWqPXCaUhcvLbEWVR8oj8RmWkdaxxggHRMTFLL8LÇseHzwwBqM8ALcTcPRc

Key: NMgBDCZUX4

6. Source Code to Decrypt

We have designed Azrael as a symmetric encryption system; therefore, it is enough to invert the process in order to decode a ciphertext obtained with this algorithm. The source code of the program a*zrael_de.py*, which works according to the scheme in Figure 2 in reverse order, is as follows:

```
1   # Decipher implementation of 'Azrael'
2   # David Arboledas-Brihuega, 2017
3   # Feel free to use it and break it!
4
5   from math import ceil
6   import S_Box
7   import P_Box
8   import pyperclip
9
10
11  def main():
12      message = input('Ciphertext: ')
13      password = input('Key: ')
14      pass_len = len(password)
15      key1 = password[0:pass_len // 2]  # subkey K1
16      key2 = password[pass_len // 2:pass_len]  # subkey K2
17      message = permutation(message, password)  # First P-Box
18      message = substitution(message, password)  # Second S-Box
19      boxes = generate_boxes(message, pass_len)
20      message = S_andP_Boxes(boxes, len(boxes), pass_len, key1, key2)  # Rounds
21
22      print('Plaintext: ', '\n', message.rstrip('Ç'))
23
24
25  def fill_box(last_box, pass_len):  # Fill the last box with Ç char if necessary
26      gaps = pass_len - len(last_box)
27      last_box += 'Ç' * gaps
28      return last_box
29
30
31  def generate_boxes(message, pass_len):
32      boxes = []
33      number_of_boxes = ceil(len(message) / pass_len)
```

```
34
35      for i in range(number_of_boxes):
36          inf = pass_len * i
37          sup = pass_len * (i + 1)
38          boxes.append(message[inf:sup])
39
40      boxes[-1] = fill_box(boxes[-1], pass_len)
41      return boxes
42
43
44  def S_andP_Boxes(boxes, number_of_boxes, pass_len, key1, key2):
45      messageRound1 = []
46
47      for j in range(number_of_boxes):
48          first = boxes[j][0:pass_len // 2]  # boxes' first half
49          second = boxes[j][pass_len // 2:pass_len]  # boxes' second half
50          messageRound1.append(permutation(first, key2))
51          messageRound1.append(substitution(second, key2))
52      message = swapp(messageRound1, boxes)
53      messageRound2 = []
54
55      for i in range(number_of_boxes):
56          messageRound2.append(permutation(messageRound1[2 * i], key1))
57          messageRound2.append(substitution(messageRound1[2 * i + 1], key1))
58      messageRound3 = []
59
60      for j in range(0, number_of_boxes):
61          first = messageRound2[2 * j]
62          second = messageRound2[2 * j + 1]
63          messageRound3.append(permutation(first, key2))
64          messageRound3.append(substitution(second, key2))
65
66      message = swapp(messageRound3, boxes)
67      messageRound4 = []
68
69      for j in range(number_of_boxes):
70          first = message[2 * j]
71          second = message[2 * j + 1]
72
73          messageRound4.append(permutation(first, key1))
74          messageRound4.append(substitution(second, key1))
75
76      return ''.join(messageRound4)
77
78
79  def substitution(message, key):
80      return S_Box.method(key, message, 'decrypt')
81
82
83  def swapp(message, boxes):
84      for i in range(len(boxes)):
85          message[2 * i], message[2 * i + 1] = message[2 * i + 1], message[2 * i]
86      return message
87
88
89  def permutation(message, password):
90      return P_Box.decrypt(message, password)
91
92
93  if __name__ == '__main__':
94      main()
```

7. Decrypting with Azrael

When *azrael_de.py* is executed the program will ask for the encrypted text and password that was used to encrypt the original message. Then, the plaintext will be shown:

Ciphertext: aPWqPXCaUhcvLbEWVR8oj8RmWkdaxxggHRMTFLL8LÇseHzwwBqM8ALcTcPRc

Key: NMgBDCZUX4

Plaintext: In a village of La Mancha the name of which I have no desire

8. Project Assessment

At the end of the project, all the students, including those who left the project, completed a final survey to assess the adequacy of the effort required. From the data gathered in this survey, it can be deduced that most of them considered they had to work harder (Figure 3).

They were also asked if they thought that this methodology of cooperative learning on a project basis was the most appropriate for this purpose. The answered questions about this teaching-learning method can be seen on Figure 4.

They were finally asked about their preferences regarding the method they preferred to be assessed in the subject and why among the following: continuous evaluation, work-group exams and co-evaluation (Table 1).

From the data gathered in this survey, it can be seen that the most popular method of evaluation by tutorized students was continuous evaluation and then work-group tests. Regarding this last method, students think they established a sense of responsibility among their members, since the score depended on all of them.

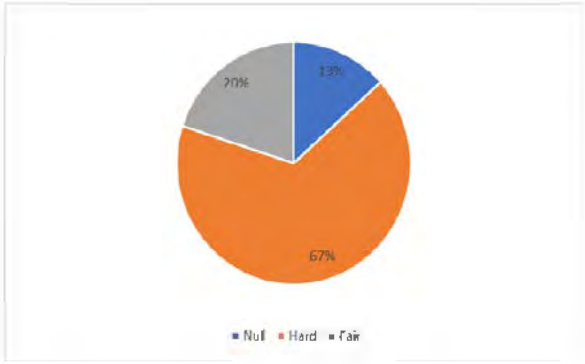

Figure 3. Students' opinion about the degree of effort with the activity

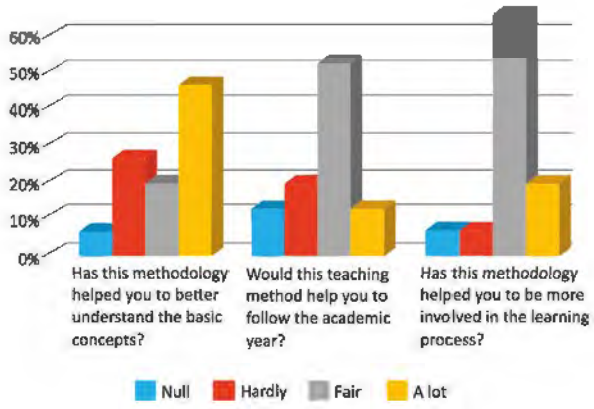

Figure 4. Degree of satisfaction with the methodology used

Assessment method	First option	Second option	Third option
Continuous evaluation	67.9 %	19.9 %	12.2 %
Work-group exams	19.9 %	67.9 %	12.2 %
Coevaluation	12.2%	12.2 %	75.6%

Table 1. Preferred Methods of Assessment

9. Conclusions

As a result of the new educational evaluation focused on students and their development, we have seen the need to define a tutorial project, of a voluntary nature, to evaluate teamwork through a formative evaluation, so that students can be convinced of their level of competence with feedback techniques (Pachler, Daly, Mor & Mellar, 2010).

The project used as a framework was the coding in Python of an encryption program that introduces enough confusion and diffusion into the cryptogram so that obtaining the clear text without the original password is not trivial.

For its design we have only used transpositions and substitutions operations, which have been learned by the students in their classes. Although the two operations may be simple to break separately, their combination as product encryption creates enough complexity to find a solution easily.

The coded program shows students how to achieve a practical application from the different theoretical knowledge they acquire during their training. Even though the algorithm is simple, it uses the same methodology as other strong encryption algorithms, such as AES.

It is very important to note that Azrael does not use secure S-boxes, which will be the next implementation we will be made in a new project in the ICT classroom, so it should only be used as an educational tool, not to encrypt sensitive information.

In this paper we have come to the conclusion that while project work introduces a very interesting methodology, it is also true that the current school system is not prepared for it. The same curricula continue to be used, with increasingly shorter times, and this means extra work that not all students can do. However, despite the fact that only three students finished the project, the general feeling was one of success in having achieved an objective that seemed impossible for them.

In conclusion, we can assure that the development of the mentoring project has been a success. Students have been left with a great feeling of achievement working in teams because they have felt themselves an essential part of their own training.

Acknowledgements

The author would like to thank all the students in the final year of Information and Communication Technologies at the secondary school Professor Domínguez Ortiz. Their opinions will undoubtedly help us to improve the transmission of the necessary skills.

Declaration of Conflicting Interests

The author declares no potential conflicts of interest with respect to the research, authorship or publication of this article.

Funding

Unfortunately, the author did not receive financial support for the research, authorship and/or publication of this article, only the personal satisfaction of seeing how students put into practice the different skills worked in class.

References

Arboledas-Brihuega, D. (2017). *Criptografía sin secretos con Python* (1st ed.) (283-306). Paracuellos de Jarama, Madrid: Ra-Ma.

Bard, G. (2009). *Algebraic cryptanalysis* (29-52). Springer.

Biham, E., & Dunkelman, O. (2012). *Techniques for Cryptanalysis of Block Ciphers.* New York, NY: Springer.

Even, S., & Goldreich, O. (1985). On the power of cascade ciphers. *ACM Transactions on Computer Systems,* 3, 108-116. https://doi.org/10.1145/214438.214442

Feistel, H. (1973). Cryptography and computer privacy. *Scientific American,* 228(5), 15-23. https://doi.org/10.1038/scientificamerican0573-15

Hannan, S., & Asif, A. (2017). Analysis of Polyalphabetic Transposition Cipher Techniques used for Encryption and Decryption. *International Journal Of Computer Science And Software Engineering,* 6(2), 41-46.

Hoffstein, J., Pipher, J., & Silverman, J. (2008). *An Introduction to Mathematical Cryptography.* New York, NY: Springer.

Pachler, N., Daly, C., Mor, Y., & Mellar, H. (2010). Formative e-assessment: Practitioner cases. *Computers & Education,* 54(3), 715-721. https://doi.org/10.1016/j.compedu.2009.09.032

Shannon, C.E. (1949). Communication theory of secrecy systems. *The Bell System Technical Journal,* 28, 656-715. https://doi.org/10.1002/j.1538-7305.1949.tb00928.x

Zaidan, B., Zaidan, A., Al-Frajat, A., & Jalab, H. (2010). On the Differences between Hiding Information and Cryptography Techniques: An Overview. *Journal of Applied Sciences*, 10(15), 1650-1655. https://doi.org/10.3923/jas.2010.1650.1655

Published by OmniaScience (www.omniascience.com)

Journal of Technology and Science Education, 2019 (www.jotse.org)