**ByMySelf(BMS) Encryption Algorithm**

**Research paper**

**Authors:**

**Marusic Diana**

**Savva Emilia**

**TABLE OF CONTENTS**

**Introduction. Research questions. Engineering goals.**

1. **Materials. Methods. Procedure**
2. **Algorithm description**

**2.1. Algorithm(V 1.0 ) description**

**2.2. Algorithm(V 2.0 ) description**

**2.3. Algorithm(V 3.0 ) description**

**2.4 Algorithm(V 4.0 ) description**

**2.5 Algorithm(V 4.2 ) description. Mathematical model**

**III. ALGORITHM COMPLEXITY**

**IV. DISCUSSIONS. APPLICATIONS**

**V. ALGORITHM IMPLEMENTATION**

**4.1. Implementation using C++**

**4.2. Implementation using Flask and SQLAlchemy**

**Conclusions**

**Bibliography**

**Annex A. Terminology**

**INTRODUCTION**

Cyber Security and Information Security are and will be matters that influence both big companies and individuals. Current services and software products do provide the user security to a certain extent with the help of encryption, but with emerging modern fields, newer mathematical discoveries and increasing computational power algorithms considered secure today will be vulnerable to cyber attacks in the future, therefore the need of newer, secured and efficient algorithms always persists.

**RESEARCH QUESTIONS**

* Will the algorithm be secured?
* Will it be feasible?
* What will be the average encryption and decryption time?
* Will the algorithm be more secured than than existing algorithms?
* Faster than existing?
* What will be the advantages of the system?
* What will be the complexity of a brute force attack?
* How will it be affected by increasing computational power?
* How will it be protected from possible hacker attacks?

**ENGINEERING GOALS**

* Simple but efficient architecture
* Modern technologies
* Applicable to existing systems
* Feasible
* Secure
* Implemented efficiently in a programming language
* Fast encryption & decryption
* User-friendly system
* Easy to implement
* Transparent(Open source)

1. **MATERIALS. METHODS. PROCEDURE**

**1.1 MATERIALS**

- C, C++

- Python

- Flask framework

- SQLAlchemy

- Linux

- Sublime Text

- Digital Ocean server

- CrossFTP software

- HTML, CSS, Bootrstrap

- JavaScript, Chart.js

**1.2 METHODS**

- Sciencific resarch

- Experiments

- Cryptanalysis

- Time complexity deduction

- Real time testing

- Observations

**1.3 PROCEDURE**

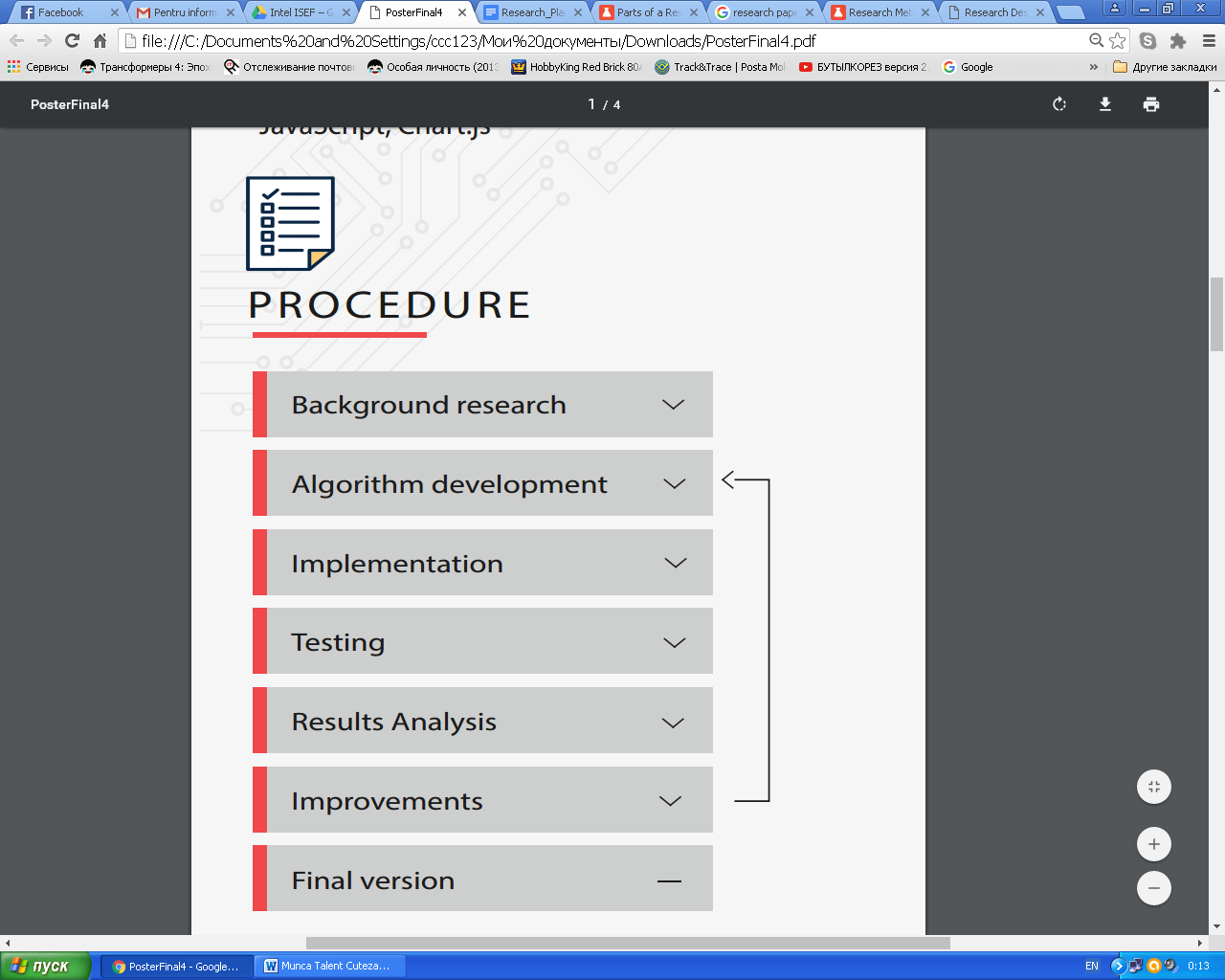


Fig 1. The process

**II. Algorithm description**

**2.1. Algorithm(V 1.0 ) description**

First step consists in creating a key that consists only of digits. If the key consists of letters, it should be transformed to digits.

For example, the key is **1243.**

The algorithm will be analyzed for the following case:

***key: 1243***

***plaintext: AAAAAAA***

*Note: For the purpose of simplicity the algorithm will be explained using letters, but to be more secured it should be used at the level of bits - conventionally each letter represents one bit. Also for simplicity, it is assumed that “A” letters are bits of plaintext and “R” letters - random bits*

The ciphertext will be obtained in the following way:

1. The first bit of the ciphertext will be the first bit of the plaintext: **A**
2. The first digit of the key is 1 - this means that after the first bit of plaintext(**A** respectively) a random bit will be addeed. The message becomes **AR**
3. The second bit of the plaintext is added. **ARA**  is obtained
4. The second bit of the key is 2 - this means that after the second bit of the plaintext(**A** respectively) 2 random bits will be added. The message becomes  **ARARR**
5. The third bit of the plaintext is added. **ARARRA**  is obtained
6. The third bit of the key is 4 - this means that after the third bit of the plaintext(**A** respectively) 4 random bits will be added. The message becomes  **ARARRARRRR**
7. This procedure is repeated until all bits of the plaintext are used. In the case the key is too short, it will be repeated as many times as necessary

The ciphertext for this example will be: **ARARRARRRRARRRARARRA**

*Note: For the purpose of masking the last bit, the key should be applied one more time after the last bit. So the ciphertext becomes* **ARARRARRRRARRRARARRARRRR***. The same procedure can be used for the first bit.*

**2.2. Algorithm(V 2.0 ) description**

As in the previous version(1.0), a digit key should be used.

Assuming that the key is **1243,** the algorithm will be analyzed for the following case:

***key: 1243***

***plaintext 1: AAAAAAA***

***plaintext 2: BBBB***

*Note: For the purpose of simplicity the algorithm will be explained using letters, but to be more secured it should be used at the level of bits - conventionally each letter represents one bit. Also for simplicity, it is assumed that “A” letters are bits of plaintext and “R” letters - random bits*

The ciphertext will be obtained in the following way:

1. The first plaintext will be encrypted exactly as in the version 1.0(Simple encryption). The obtained ciphertext, conventionally named *first ciphertex*t, is **ARARRARRRRARRRARARRARRRR**
2. The second plaintext will be encrypted exactly as in the version 1.0(Simple encryption). The obtained ciphertext, conventionally named *second ciphertex*t, is **BRBRRBRRRRBRRR**
3. “Shift” the second ciphertext with one position to right, The second ciphertext becomes **RBRBRRBRRRRBRRR**
4. Unite the first ciphertext with the second ciphertext in the following way:

Where there is a plaintext bit in on of the ciphertexts, this bit remains. Where there are 2 random bits remains a random letter.

**A R A R R A R R R R A R R R A R A R R A R R R R**

**R B R B R R B R R R R B R R R**

**A B A B R A B R R R A B R R A R A R R A R R R R**

*The final ciphertext ist:*

**A B A B R A B R R R A B R R A R A R R A R R R R**

*Note: This algorithm can be used with multiple messages simultaneously. In this case, there is one more unknown variable for the potential attacker - the number of encrypted messages.*

**2.3. Algorithm(V 3.0 ) description**

message: **ABCDEFGHIJKL**

initial key: **1241234414**

For the purpose of masking the key length, a new parameter is introduced. This parameter(conventionally named del\_pos - delete position ), specifies the number of following bits that are nonsense and will be deleted.

Example:

del\_pos = 2

|  |  |
| --- | --- |
| position | **0123456789** |
| initial key(masked) value | **1241234414** |

Explanation: del\_pos is 2, this means that the digit on the second position - 4 - specifies that the following 4 digits from the key(including the digit itself) should be deleted.

So the new key, that will be used, is: **12414**

The new length of the key is **6.**

To increase the level of security, the message is split in blocks of certain length, that is specified in the key. For example, the position 9 will specify the length of the blocks. Conventionally named block\_pos, this parameter will also be deleted from the key.

The final key becomes **1241**

**ABCD|EFGH|IJKL**

Each block will be encrypted using the same algorithm(Simple variant 1.0), but a different key. Each key is obtained in the following way:

In the first block the real-length key(Nr. 2) will be used :

For ABCD key = 1241 (ARBRRCRRRRDR)

**1 2 4 1**

For the second block, the key will be obtained by taking the first digit of the kay and putting it at the end:

For EFGH key = 2411 (ERRFRRRRGRHR)

**2 4 1 1**

This procedure will be obtained for the following blocks:

For IJKL key = 4112 (IRRRRJRKRLRR)

**4 1 1 2**

Final message: **ARBRRCRRRRDRERRFRRRRGRHRIRRRRJRKRLRR**

**1 2 4 1 2 4 1 1 4 1 1 2**

**2.4. Algorithm(V 4.0 ) description**

message: **ABCDEFGHIJKL**

initial key: **1241234414**

The final key becomes **1241**

**ABCD|EFGH|IJKL**

Each block will be encrypted using the same algorithm(V1.0), but a different key. Each key is obtained in the following way:

1. Generate a random key(conventionally named rand\_key1)

*rand\_key1 = 3521*

1. **ABCD + rand\_key1**

***ABCD****3521*

1. Encrypt the first block using final\_key

In the first block the real-length key(Nr. 2) will be used :

*For ABCD key = 1241 (RARRBRRRRCR3R5RR2RRRR1R)*

**1 2 4 1**

1. Generate a random key(conventionally named rand\_key2)

*rand\_key2 = 5341*

1. **EFGH + rand\_key2**

***EFGH****5341*

1. Encrypt the second block using rand\_key1

For the second block, the key will be rand\_key1::

*For EFGH key = rand\_key1 = 3521 (RRR****E****RRRRR****F****RR****G****R****H****RRR****5****RRRRR****3****RR****4****R****1****RRR)*

This procedure will be used for all the following blocks.

**2.5. Algortihm(V 4.2) description**

V 4.2 solves a weakness found in V 4.0 - the vulnerability of the first block. The principles are the same as 4.1, excepting the encryption of the first block. A “zero block” containing only a random key, encrypted with the initial real key(not masked) is added before the first block, which is then encrypted with this random key.

ENCRYPTION:

1. key = FirstKeyToKey()

2. Generate Random Key rand\_key

3. Encrypt rand\_key with the key - this will be the first block

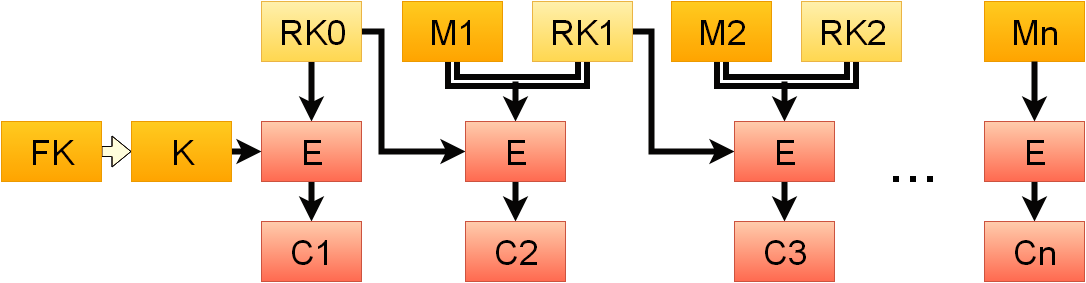
4. While ExistLettersInMessage (loop):

Generate new\_rand\_key

Add new\_rand\_key at the end of the plaintext block

Encrypt obtained block with the previous rand\_key

end\_loop()



**Notations:**

FK – First Key

K – Real Key(after transformation)

RK – Random Key

E – Encryption

Mi – ith block of the message(plaintext)

Ci - ith block of the cyphertext

**Final version (4.2) Mathematical Model**

P - a finite set of plaintexts

C - a finite set of ciphertext

K - a finite set of initial keys

E - a finite set of encryption functions

D - a finite set of decryption functions.

FK – First key(masked)

RK – Random key

FirstKeyToKey – The function used to transform FK to K and save block\_size parameter from FK

1. FK = {[1..9], [1..9], [1..9], [1..9], …, [1..9]}

first\_key\_len

1. FirstKeyToKey()
2. K = {[1..9], [1..9], [1..9], [1..9], …, [1..9]}

key\_len

1. For the first block P0 a random key will be generated:

RK0 =

key\_len

1. The Cipherblock C0 will be obatined by encrypting RK0 with K
2. For each of the following blocks Pi a random key will be generated:

RKi =

key\_len

1. Each cipherblock Ci will be obatined by encrypting Pi+RKi with RKi-1

Output: C0||C1||C2||C3||C4||C5||...||Cn

**III. Algorithm Complexity**

keylib.cpp

|  |  |  |
| --- | --- | --- |
| **Function** | **O(n) complexity** | **Extended form** |
| readKey | n | 2n |
| readArray | n | 1+2n |
| printKey | n | n |
| printArray | n | n |
| keyTransform | n | 2+2n |
| generateRandomKey | n | n |
| separateText | n | 2n |
| separateKey | n | 2n |
| **total:** | n | 3+13n |

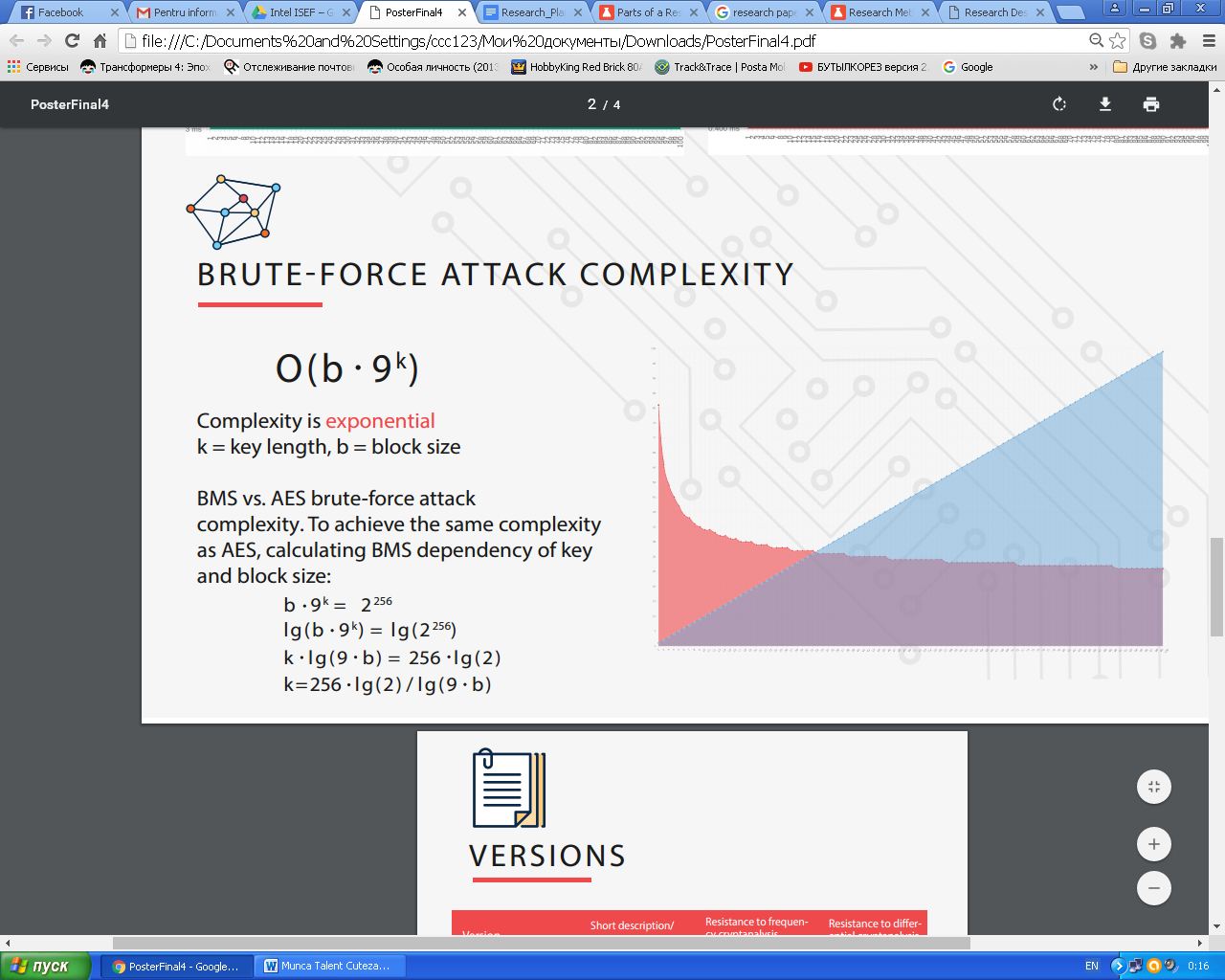
encryption.cpp

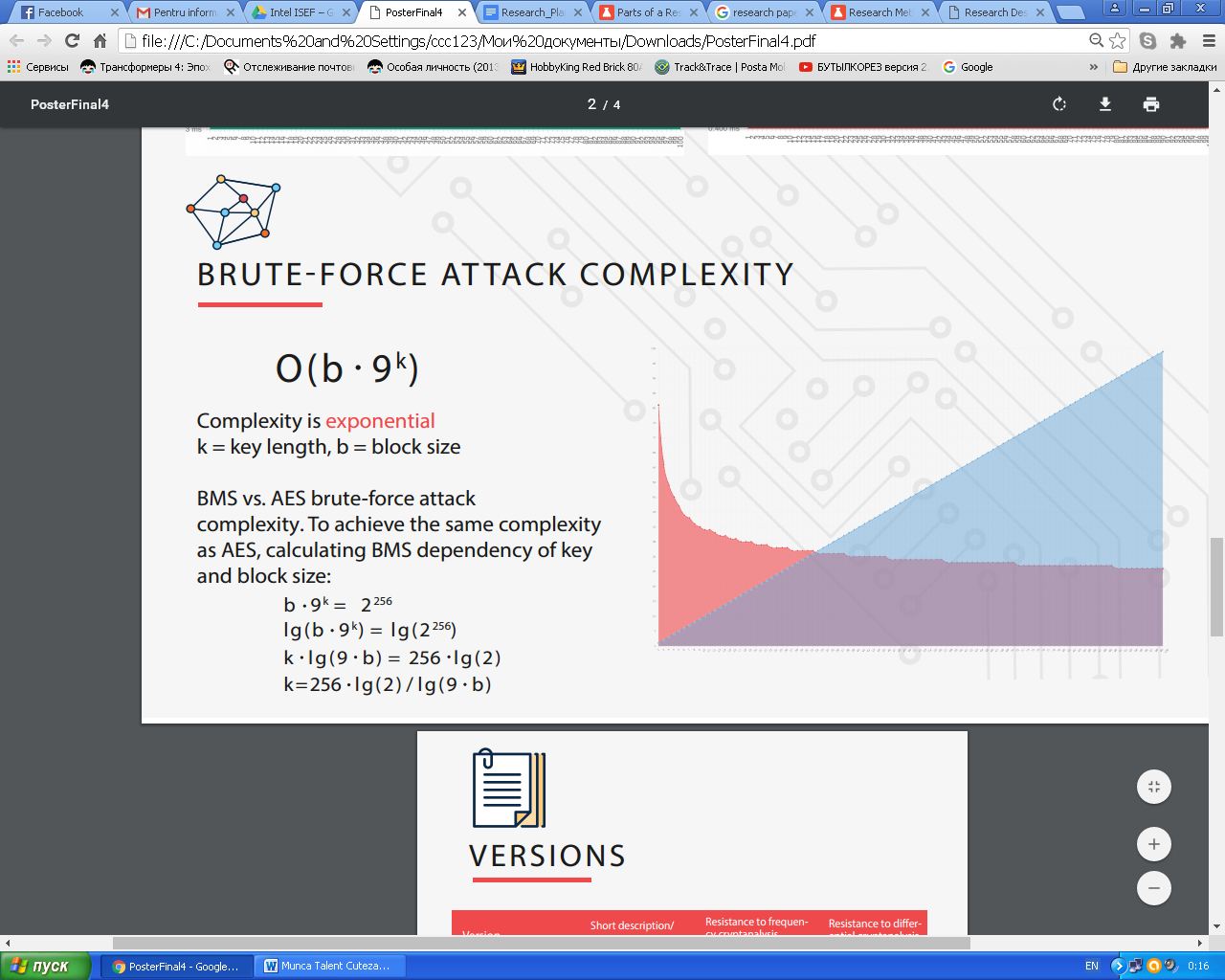
|  |  |  |
| --- | --- | --- |
| **Function** | **O(n) complexity** | **Extended form** |
| readKey | n | 5+5n |
| simpleencryptMessage | n^2 | 4+n^2+6n |
| blockEncryption | n | 4+6n |
| intArrTochar | n | 3+n |
| NumberToString | 1 | 1 |
| random\_ch | 1 | 1 |
| pointerToArray | n | n |
| blockEncryptionV4 | n^2 | 4+10n+n^2 |
| **total:** | n^2 | 19+27n+2n^2 |

decryption

|  |  |  |
| --- | --- | --- |
| **Function** | **O(n) complexity** | **Extended form** |
| simpleDecrypt | n | 4+7n |
| calcCipherblockLen | n | 2+7n |
| calcCipherblockLen | n | 2+4n |
| calcCipherblockLenV4 | n | 6+8n |
| blockDecrypt | n | 5+7n |
| blockDecryptV4 | n | 4+5n |
| **total:** | n | 21+38n |

**Brute-Force Attack Complexity**





**IV. DISCUSSIONS**

In the first version(1.0) of the algorithm a key containing the number of random bits to be added was used. In this way, one parameters was stored in the key. Later it was observed that in the case the attacker has access to the key length, ciphertext and a part of plaintext, he ban obtain the key, therefore version 1.0 doesn’t comply with the Kerckhoff’s principle. Operating with the information at the level of bits, not characters can improve algorithm’s security.

In Version 2.0 this weakness was also found, but encrypting multiple messages at the same time increases the probability of unknown information for the potential attacker. When tested for differential cryptanalysis, requiring even less iterations and less time to be broken than by using brute-force methods.

Version 4.0 uses an initial key of length masked by additional random bits, therefore the real length of the key is unknown. It is also secured to differential cryptanalysis by block encryption and using a random key for each block. However, the first block remains vulnerable, because it is encrypted with the same V 1.0 vulnerable algorithm.

Version 4.2 eliminates this weakness by adding a “zero” block - a block containing a random key in the form of the message. For obtaining even a more secured algorithm, the random key digits should be transformed to chars and the entire encryption should be done at the level of bits, not chars.

**APPLICATIONS**

1. data storage services
2. personal data protection
3. car security
4. copyrighted products
5. financial and banking security
6. smart home systems
7. electronic keys

By further improving the algorithm it could be extended to other cybersecurity or data protection areas.

**V. ALGORITHM IMPLEMENTATION**

**4.1. Implementation using C++**

The algorithm was implemented in C++ by creating 3 libraries - encryption.h, decryption.h and keylib.h. The reason of creating 3 libraries instead of writing a long main.cpp file was the need of clarity and structuring the code for an easier access by all other developers.

The first library, encryption.h, contains functions needed for encryption. Because each of the iterations of the algorithm depend on earlier iterations, this library has all versions implemented.

It contains the following functions:

* void readKey(int int\_key, arr &K, int &key\_len);
* char random\_ch();
* string simpleencryptMessage(string message, arr K, int key\_len);
* string blockEncryption (int block\_size , string plaintext , arr K, int key\_len);
* string intArrTochar(arr J,int key\_len);
* string blockEncryptionV4 (int block\_size , string plaintext , arr K, int key\_len);

The second library, decryption.h, serves for the purpose of ciphertext decryption. It also has all of the versions implemented.

It contains the following functions:

* string simpleDecrypt(string ciphertext, arr K, int key\_len);
* int calcCipherblockLen(int block\_size, arr K, int key\_len);
* string blockDecrypt(string ciphertext, arr K, int key\_len, int block\_size);
* string blockDecryptV4(string ciphertext, arr K, int key\_len, int block\_size);

The last library, keylib.h, contains functions needed for operations with key and operations between encryption and decryption libraries. It was made as a separate file, because it was intended to make the structure of the main file as simple as possible.

Functions of keylib.h:

* int\* readKey(int FK\_int, int first\_key\_len);
* void readArray(int FK\_int, int key\_len);
* void printKey(arr FK, int first\_key\_len);
* void printArray(arr A, int len);
* int \* keyTransform(arr first\_key, int first\_key\_len);
* int \* generateRandomKey(int key\_len);
* string separateText(string plaintext, int key\_len);
* string separateKey(string plaintext, int key\_len);

These libraries allow programmers to implement the algorithm in their projects, being an efficient implementation of BMS.

**4.2. Implementation using Flask and SQLAlchemy**

The algorithm was implemented in C++ and Python. For the purpose of illustrating real A web-based system was created using Flask framework and SQLAlchemy database. The same procedure as with the C++ implementation was done - there were created 3 libraries - encryption.py, decryption.py and keylib.py, containing similar functions. Because Python is an interpreted programming language, the tests showed a slower encryption and decryption, but for a message 10000 bytes the difference is of the order of milliseconds.

The users can create their profiles, by introducing regular information for any system that uses a database and an additional digit key for registration. The registration page is presented in the Fig. 2.

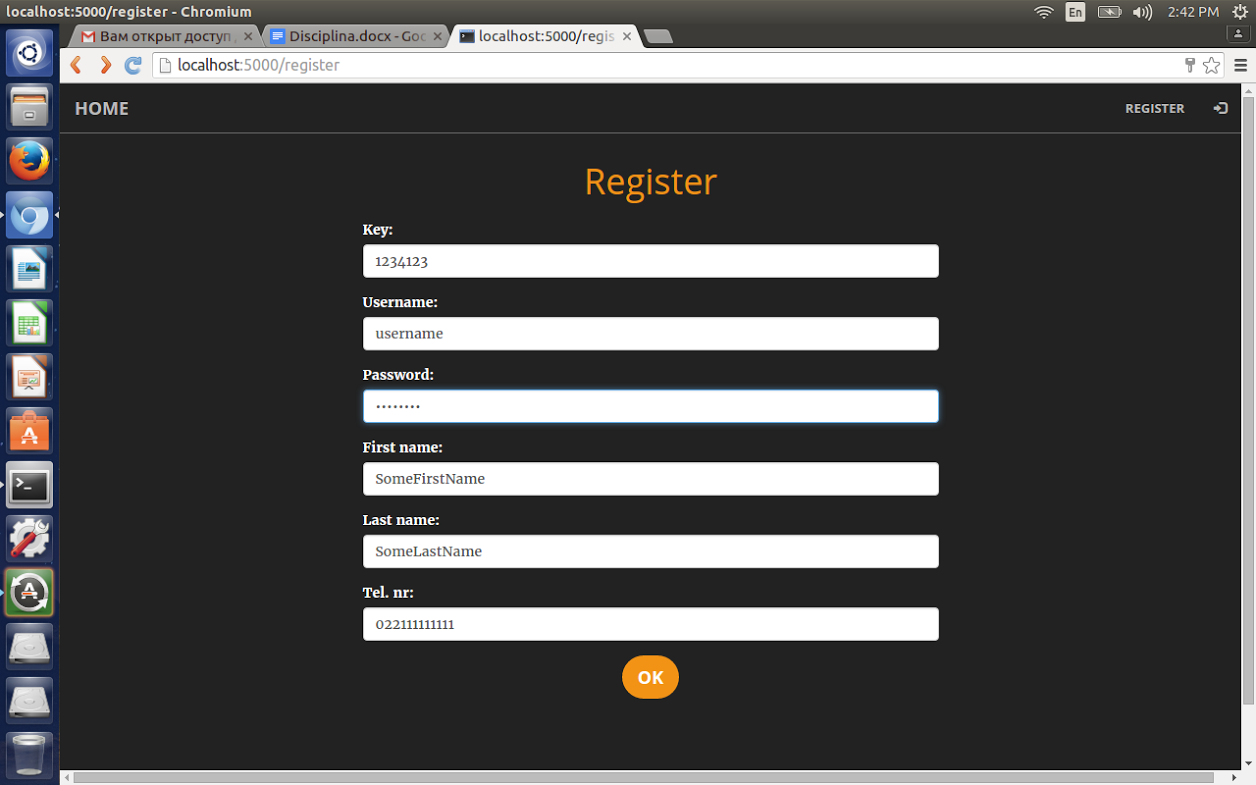


Fig 2. Registration page

After a user creates a profile, he can access the system. Log in page is presented below.:

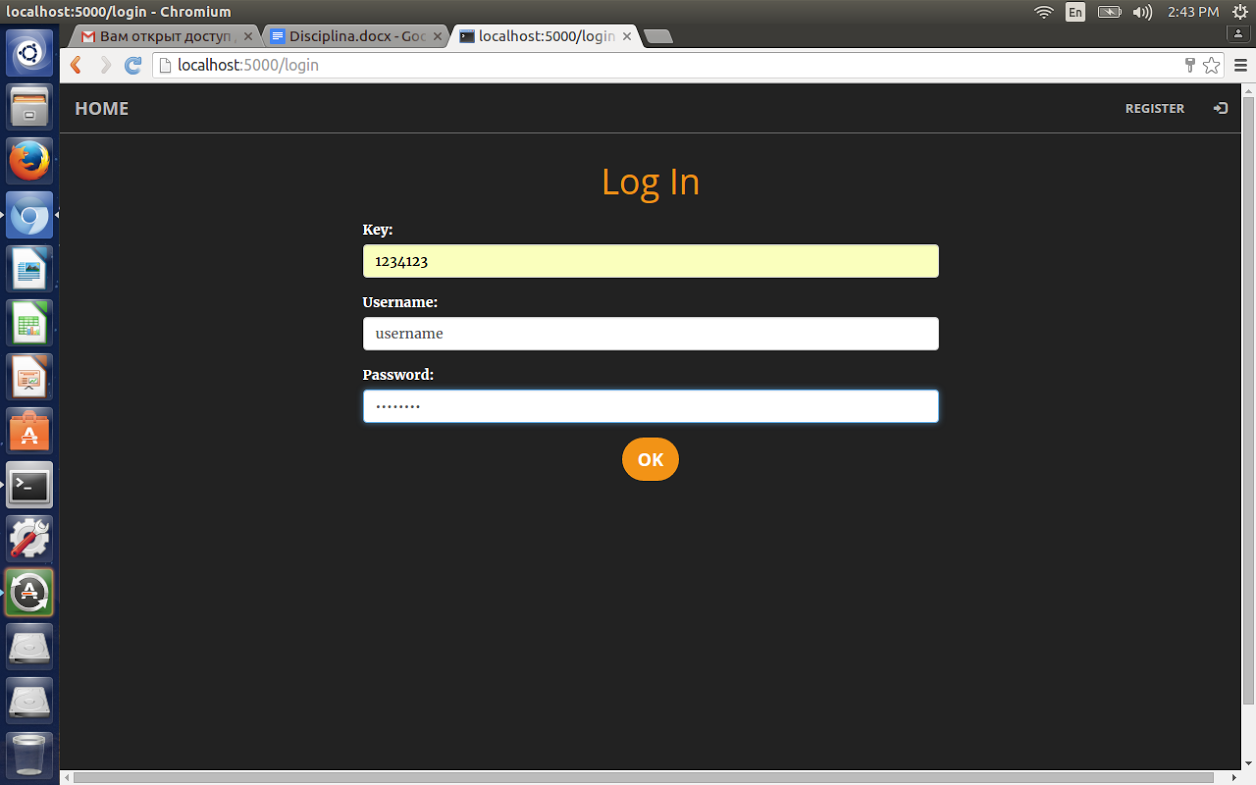


Fig 3. Log in page

After a successful login, the users can save data into the database and access their data. Data can by any text or number, for example their passport ID, their favourite color or their bank account number. The system keeps their data secure.

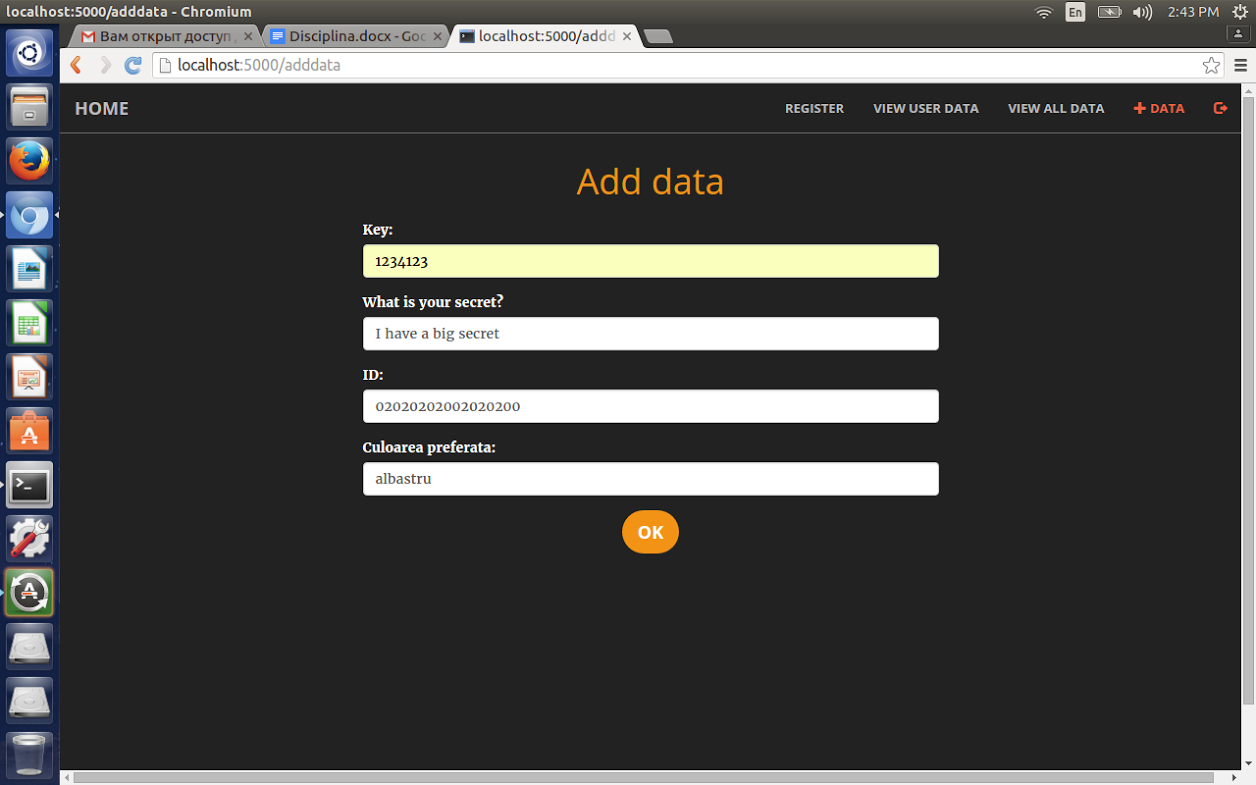


Fig 4. Page that allows users to add data

For illustrative purposes, the system was build in such a way, that it can present both decrypted and encrypted data.

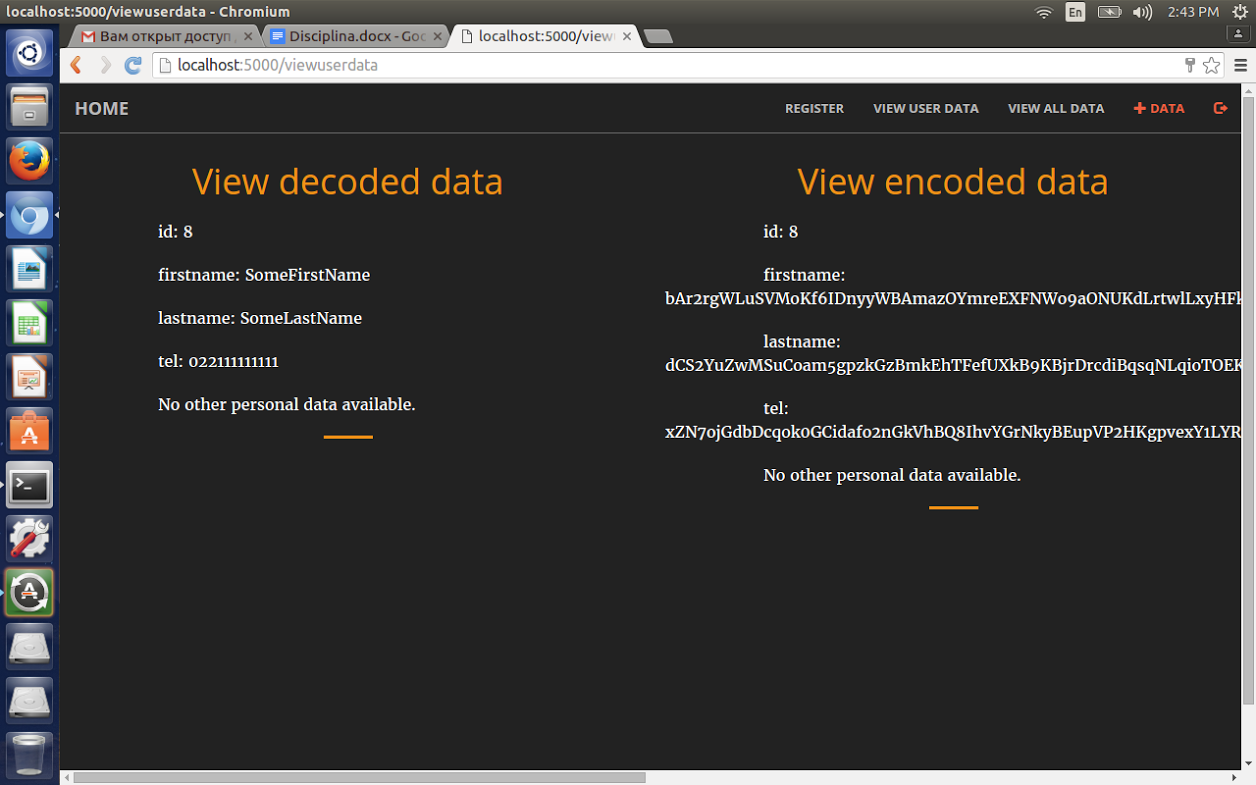


Fig 5. The possibility to view encrypted and decrypted data

This system can be extended to store other data and have other applications as well.

The system uses the database named *security,* created with SQLAlchemy. The *security* database contains 2 tables - *confidential* and *personal\_data.* The table named *confidential* contains user credentials(without storing the digit key) and the table named *personal\_data* stores data added the users.



Fig. 6. Data through the eyes of the administrator. Security database. Confidential table

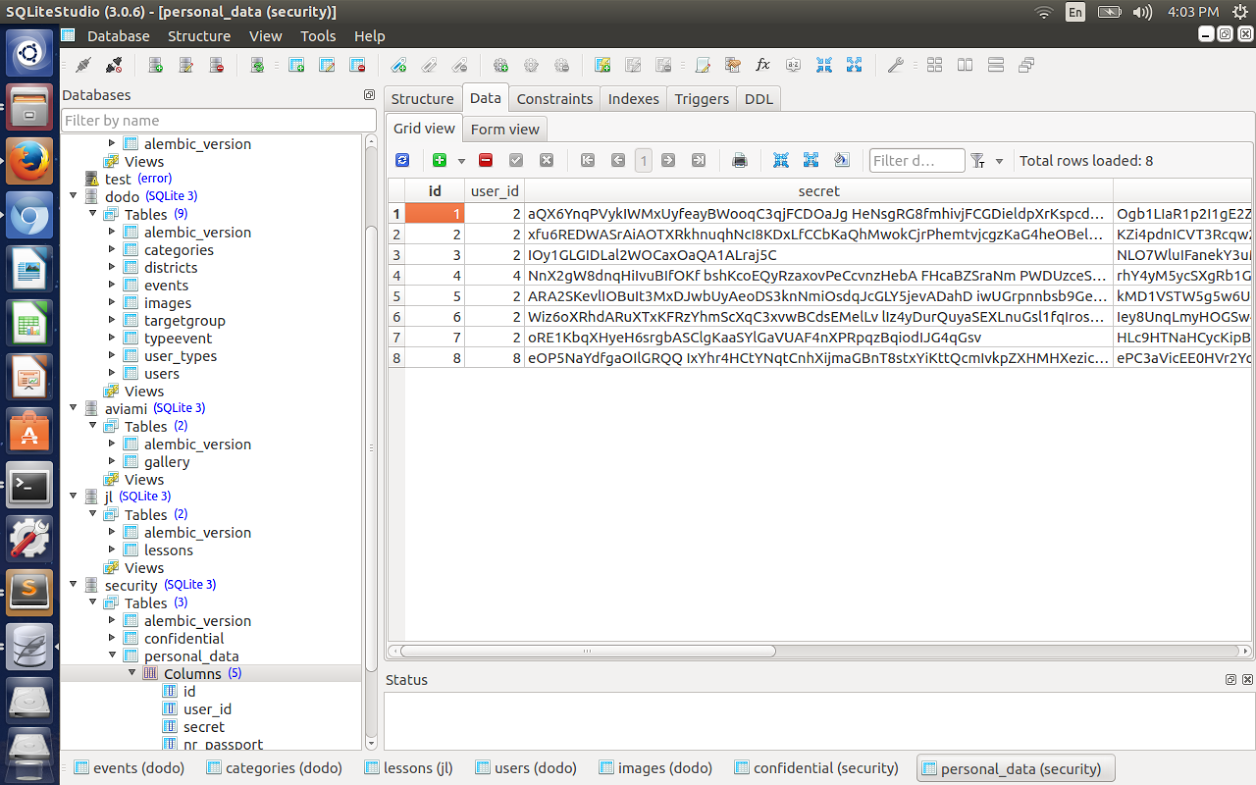


Fig. 7. Data through the eyes of the administrator. Security database. Personal data table

**CONCLUSIONS**

This paper presented ByMySelf (BMS) encryption algorithm.

The Background Research was the point of start; analysis of encryption algorithms, protocols, instruments: AES, SSL, RSA, PGP. Next step was developing several concepts of the algorithm, and through the process of evolution that included the following phases: Algorithm Development, Implementation, Testing, Results and Analysis, Improvements; resulted the current version. In each iteration the algorithms were tested and compared by using: cyber-attack simulations, cryptanalysis and complexity calculations.

The resulted algorithm has a shorter key and block size than AES but possesses the same brute force complexity. It has a fast encryption and decryption and offers the possibility to set manually the key length and block size (thus a potential hacker needs to brute force not only the information itself but the key and block size as well).

The hypothesis has been proven partially true, because besides storing encryption algorithm parameters in the key, a secure system depends very much on the design itself of the algorithm.

The algorithm was implemented in C++ and Python. Using Flask Framework and SQLAlchemy, an example of a real application was shown. This system can be extended to store other data and have other applications as well.

The obtained algorithm can be used for the following purposes: data storage services, personal data protection, car security, copyrighted products, financial and banking security, smart home systems. By further improving the algorithm it could be extended to other cybersecurity or data protection areas.

**BIBLIOGRAPHY**

1. <http://mathworld.wolfram.com/RSAEncryption.html> - Wolfram MathWorld, RSA encryption
2. <http://searchsecurity.techtarget.com/definition/RSA> - Search Security website, RSA algorithm (Rivest-Shamir-Adleman)
3. <http://searchsecurity.techtarget.com/definition/cryptanalysis> - Search Security website, cryptanalysis
4. <https://www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf> - [M](http://www.mun.ca/memorial/)emorial University website, A Tutorial on Linear and Differential Cryptanalysis by Howard M. Heys
5. <http://www.storagecraft.com/blog/5-common-encryption-algorithms/> - 5 Common Encryption Algorithms and the Unbreakables of the Future
6. <http://www.cryptomuseum.com/crypto/vernam.htm> - Crypto Museum, The Vernam Cipher
7. <https://www.cs.bham.ac.uk/~mdr/teaching/modules/security/lectures/symmetric-key.html> - Computer Security lecture notes, The University of Birmingham, Symmetric-key cryptography
8. <http://guan.cse.nsysu.edu.tw/note/provable.pdf> - Introduction to Security Proof of Cryptosystems D. J. Guan
9. <https://www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf> - A Tutorial on Linear and Differential Cryptanalysis by Hovar M. Heys
10. <http://cs.ucsb.edu/~koc/ccs130h/notes/dc1.pdf> - Differential Cryptanalysis Cetin Kaya Koc Oregon State University
11. <https://www.python.org/> - Python Official Website
12. <https://pypi.python.org/pypi/pycrypto> - Py cryptography library
13. <https://github.com/ricmoo/pyaes/blob/master/README.md> - AES implementation in Python
14. <https://www.coursera.org/learn/cryptography/> - Coursera, Cryptography- Course by Jonathan Katz, University of Maryland, College Park

**ANNEX A**

**Terminology**

**Cryptography or cryptology -** the practice and study of techniques for [secure communicatio](https://en.wikipedia.org/wiki/Secure_communication)n in the presence of third parties called [adversarie](https://en.wikipedia.org/wiki/Adversary_(cryptography))s

**Plaintext** - information a sender wishes to transmit to a receiver

**Ciphertext** - the result of [encryptio](https://en.wikipedia.org/wiki/Encryption)n performed on [plaintex](https://en.wikipedia.org/wiki/Plaintext)t using an algorithm, called a [cipher](https://en.wikipedia.org/wiki/Cipher)

**Encryption** - the process of encoding messages or information in such a way that only authorized parties can read it

**Decryption -** the reverse of encryption; moving from the unintelligible ciphertext back to plaintext

**Cipher *-*** a pair of [algorithm](https://en.wikipedia.org/wiki/Algorithm)s that create the encryption and the reversing decryption

**Cryptanalysis -** the study of analyzing information systems in order to study the hidden aspects of the systems.Used to breach cryptographic security systems and gain access to the contents of encrypted messages, even if the cryptographic key is unknown

**Differential cryptanalysis** - a general form of cryptanalysis that studies how differences in information input can affect the resultant difference at the output

**Brute-force search** - a problem-solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement.

## Kerckhoffs' law - "The security of a cryptographic system is based solely on the key." It states that as long as the key is kept secret, the cryptographic algorithm can be made public without compromising the system.