Encryption Algorithms for Secure Communication over the Internet

BICS student: Desislava MARINOVA*, PhD Student Asya MITSEVA†, Prof. Dr. Thomas ENGEL ‡
Faculty of Science, Technology and Communication, University of Luxembourg,
Luxembourg

*Email: *desislava.marinova.002@student.uni.lu, †asya.mitseva@uni.lu, ‡thomas.engel@uni.lu*

***Abstract*—Cryptography is a broad method for secure data disguise and transmission. The primary purpose of this paper is to explore the state-of-the-art of cryptographic primitives, both in theory and in practice. The project mainly focuses on symmetric-key cryptography and reviews the types of ciphers it encompasses: transposition and substitution along with block and stream. The project carries on with an implementation of a chat application, in Python programming language, that encrypts messages before transmitting them over the Internet. The encryption algorithms used to develop the client/server infrastructure are: Caesar cipher, ROT13, DES and AES. The latter two have been implemented through the means of PyCryptodome library. Being able to choose the type of encryption, once connected to the server, the client can easily compare the efficiency of the implemented algorithms. As a result, the basic features, advantages, drawbacks and applications of various symmetric-key cryptography algorithms have been mentioned in this paper.**

1. Introduction ($\pm$ 5% total words)

Communication technologies have turned the past decades into an information age, dense with electronic connectivity, eavesdropping and fraud. We witness an influx of sensitive information breaches. The reason being is the lack of strong access control measures. Various methods exist for hacking and circumventing sensitive data, e.g social engineering attacks, distributed brute forcing. It is worth mentioning, the use of the same password to access different accounts that grant easy account access to determined scammers. We should not fall short in addressing todays threat landscape. For all that, when we send private data around the world instantaneously, we do not want it to be intercepted, redirected or stolen. Thus, information is an asset that has precious value like any other. Modern society relies on an evolving panoply of information technologies and technology-enabled services. Future needs of data security is of utmost importance given the increasing global nature of commerce, business, communication, eHealth and eGovernment. We all use the Internet, data is becoming congested. Therefore, how do we maintain privacy, integrity, online-identity? Network based attacks spring up every day and distance for hackers is not a problem anymore since data has been digitalized. There are various factors contributing to the vulnerability of Internet security. For this reason, the disciplines of Cryptography and Network Security have matured. Thanks to them financial flows and medical records are being protected. Cryptography and Network Security combined defend state secrets and the corporate sort, and make the e-commerce industry possible. Without cryptography, credit-card details, bank transfers would zip around the Internet unprotected, for anyone to see or steal. From obscure science, Cryptography has become one of the major pillars of modern cyber reality. To develop a security mechanism (algorithm or protocol) one must always consider potential attacks, take countermeasures, role play as an intruder and exploit eventual weaknesses.

In order to understand the principles of cryptography, we dive into the terminology that governs our report. The word *cryptography* comes from two Greek words meaning *secret writing* [1]. Cryptography is the art and science of concealing meaning, of keeping the enciphered information secret and designing and analyzing encryption schemes. Mathematics is the foundation on which modern encryption rests. Cryptanalysis is the breaking of encoded data and uses statistical and mathematical approaches. The areas of cryptography and cryptanalysis together are called cryptology [1]. The basic component of cryptography is a cryptosystem which is based on substitution and permutation. The former changes characters in the plaintext to produce the ciphertext.

The latter is an ordered sequence of elements in a finite set with each character appearing only once. In order to make information secret, we use a cipher a cryptographic algorithm that converts plaintext (i.e. data in readable form) into ciphertext (i.e. a message in unreadable but protected form). We ought to be in possession of a key to be able to convert back the plaintext, otherwise it would not be possible to discover the original content. The process of making text secret is called encryption, and the reverse process is called decryption.

There are three types of cryptography: symmetric key, asymmetric key and hash functions. For this project, we shall focus mainly on the symmetric, also called *conventional*. It comprises a single key for both encryption and decryption and it is the preferred choice when we need to encrypt large amounts of data. The reason being is because it is faster and less power consuming, mainly on the decryption side. Some of the encryption algorithms that use symmetric key encryption are: DES (Data Encryption Standard),Triple DES, AES (Advanced Encryption Standard), Blowfish, Twofish. One of the shortcomings of symmetric key cryptography is related to transferring secure files because the same key has to be used for both encryption and decryption. The sender must find a secure way to provide the recipient with the key so that the latter is able to decrypt the files. Otherwise, they risk placing the key in the wrong hands, leading to deciphering of the encrypted files. Moreover, when the file transfer environment involves multiple users dispersed around the world, it is highly impossible to distribute the key using this crypto method.

On the other hand, asymmetric cryptography, also called *public key* or *two-key cryptography*, makes use of two keys: public and private. The public key is used to encrypt the message, whereas the private key is used to decrypt it. Most widely employed asymmetric key algorithms are the Rivest-ShamirAdleman (RSA) and the Digital Signature Algorithm (DSA). However, the asymmetric method is computationally costly and cannot deal with large quantities of data [2]. The reasons for its downside are the runtime of employed supporting processes, the more storage space required and the more time needed to move over a link. However, asymmetric key cryptography is more secure when we want to transfer over an insecure channel. Furthermore, it does not have the issue with multiple user key distribution, as long as the private key is kept secret.

A cryptographic hash function takes an arbitrary block of data and returns a fixed-size bit string (a hash value, also called (message) digest) such that any change to the data will change the hash value. Prominent examples among hash functions are: Message Digest 5 (MD5), Secure Hash Algorithm 256 (SHA256), Password-Based Key Derivation Function 2 (PBKDF2).

The choice to be made on which system to use for encryption and decryption depends on the purposes, requirements, ease of distribution and on the functions of each algorithm. Modern applications employ a hybrid versions of both symmetric and asymmetric key cryptography as they have their own advantages. For example, Secure Sockets Layer (SSL) and Secure Shell (SSH) are classified as hybrid cryptosystems.

If someone wants to break the ciphertext, knowing the algorithm but not the specific cryptographic key, there are three main types of attacks. A good cryptosystem should protect against them [3]:

- Ciphertext only attack: the ciphertext is known and the goal is to find the corresponding plaintext. If possible, the key may be found as well.
- Known plaintext attack: both ciphertext and plaintext are known and the goal is to find the key.
- Chosen plaintext attack: specific plaintexts are enciphered in order to find out the corresponding ciphertexts and the goal is to find the key.

2. Project description ($\pm$ 10% total words)

2.1. Domain

The report draws on a variety of disciplines. However, it is impossible to appreciate the vastness and significance of the topic in a limited number of pages. Nevertheless, we attempt to make the report self-contained by providing the reader with an intuitive understanding of our survey and application results. The domains that are associated with our project are:

- Internet security: a branch to computer security, related to the Internet, comprised of measures to

protect transactions done over a collection of interconnected networks [4].

- Computer Security: the process of preventing and detecting unauthorized use of a computer [5].
- Cryptographic primitives: well-established, low-level cryptographic algorithms frequently used to build cryptographic protocols for computer security systems [6].
- Symmetric-key cryptography: an algorithm for cryptography that uses the same key for both encryption and decryption [7].
- Cryptographic algorithms: ensure alteration of data in order to preserve its authenticity, confidentiality, integrity through various security mechanisms such as symmetric encryption, asymmetric encryption and cryptographic hash functions [1].

2.2. Objectives

The objectives of this project are three-fold.

1) As a preliminary task, we address a highly needed background material. We give a basic introduction to the concepts and primitives of cryptography, i.e. transposition/permutation, encryption/decryption, symmetry/asymmetry, block ciphers/stream ciphers and how they preserve their security in constantly emerging attacks.
2) Our main focus falls on symmetric encryption, taking into account both classical and modern algorithms. Therefore, we review existing state-of-the-art encryption algorithms: Caesar cipher, ROT13, One-time pad, DES, 3DES, AES.
3) Last but not least, we implement a chat system that encrypts user's messages before they are transmitted over the Internet.

2.3. Constraints

Our project takes into account substitution/transposition; block/stream ciphers but we do not consider:

- Asymmetric ciphers (public-key algorithms, including RSA and elliptic curve);
- Data integrity algorithms (cryptographic hash functions, message authentication codes (MACs); digital signatures;
- Authentication techniques: key management and key distribution (protocols for key exchange);

3. Background ($\pm$ 15% total words)

3.1. Scientific

In this section we present substitution and transposition ciphers along with stream and block ciphers. On the one hand, substitution ciphers they take the i-th element of a string and change it for another value to produce a ciphertext. Substitution ciphers can be monoalphabetic or polyalphabetic. The former, also called simple substitution cipher, maps a plaintext letter to a ciphertext letter based on a single alphabet key. The latter uses multiple substitution alphabets. In transposition ciphers the values of plain letters are not changed for another but rearranged. Modern ciphers use a combination of both substitution and transposition ciphers. Ciphers can also be divided into stream and block ciphers. The former convert one symbol (1 bit or byte) of plaintext directly into a symbol of ciphertext. The latter encrypt a group of plaintext symbols as one block and outputs a block of ciphertext. We add that substitution ciphers can operate on either blocks or streams.

In the rest of the section, we describe different examples of ciphers for each of the groups presented above.

Substitution ciphers: The most popular example of substitution cipher is the One - Time Pad (OTP), designed in 1917. What makes it very secure is that a pad (secret key) is never reused. Instead, OTP utilizes a random key which has an equal length to the message. It shifts each plain letter depending on the corresponding keyword letter. The random key is also used in encryption and decryption of the message, and afterwards is discarded [8]. If we want to generate a new message, a new key is required. Thus, this scheme leads to an output unrelated to the plaintext. If we try to decipher a ciphertext using this method, and let us suppose

we have succeeded in finding the key, our attempts would lead us to fairly different decrypted outputs. For this reason, it would be difficult to decide upon the correct decryption. However, there are two fundamental complexities related to the one-time pad: (1) making large quantities of random keys; (2) key distribution and protection (since a key of equal length is needed by both sender and receiver) [8]. Hence, if we ought to employ the one-time pad, it would be best for low-bandwidth channels requiring very high security [8]. Nevertheless, this cryptosystem is regarded as the sole exhibiting perfect secrecy [8]. Some cipher machines attempted mechanically then electronically, in both cases nevertheless unsuccessfully, to create approximations to one-time pads (OTPs). Many snake oil algorithms claim to be unbreakable, posing as OTPs. Hence, these algorithms give rise to pseudo-OTPs providing pseudo-security [9].

How does a substitution cipher work? In order to produce the ciphertext, the cipher modifies characters in the plaintext. Can it be broken? We can break the monoalphabetic cipher by using letter frequency analysis since letter frequencies are preserved (e.g. 'e is the most common letter in English, followed by 't,i,o,a,n,s,r '). The polyalphabetic cipher can be broken by decomposing into individual alphabets and as a consequence to treat it as a simple substitution cipher. A historical example of breaking the substitution cipher is the execution of Mary, Queen of Scots, in 1587 for plotting to kill Queen Elizabeth [10]. The "Vigenere Cipher" is the most popular polyalphabetic cipher. The Vigenere cipher is an improvement of the Caesar cipher but not as secure as the unbreakable One Time Pad. Caesar cipher encodes each plain letter by a constant shift, whereas the One Time Pad shifts each plain letter depending on the corresponding keyword letter. The Vigenere cipher uses a keyword of given length repeatedly to determine the encoding shift of each plain letter. For instance, we are given a word, we convert it to numbers, according to the letter position in the alphabet. Next, we repeat the sequence of numbers along the message. Then, we encrypt each letter in the message by shifting, according to the number below it, and send the encrypted message openly to the recipient. Hence, we employ multiple shifts. To decrypt the message, we should subtract the shifts according to the secret word or key we have a copy of. Figure 6 shows a Vigenere cipher.

Cipher machines: Cryptography was mechanized by the 1900s in the form of encryption machines. The basic component of a cipher machine is the wired rotor. When we get prior to the Second World War, the Germans realized that, thanks to radios, messages can be sent across the battlefield in an instant but that always meant the other side could also tap into those radio channels. Therefore high-tech encryption was highly needed. The Germans invented Enigma,whose complexity guaranteed their privacy. The Enigma was a substitution cipher, but a more sophisticated one because it used three rotors in a row, each feeding into the next. The Germans typed their messages on a keyboard that came out as gibberish on a lampboard. Then they sent it over the radio to the other side, which also has the same Enigma machine to help them decrypt the unintelligible message. Furthermore, to decrypt the message, it is assumed that the other side should know the algorithm and must have configured the machine the same way as the Enigma machine encrypting the message. The Enigma relies on a random letter generator. Hence, its encryption does not seem to follow any kind of pattern. There are 26 wires coming out of the keyboard, running through three rotors with 6 permutations, going into the lamps representing the output letter. Once the first of the three rotors hits a full evolution, it kicks the next rotor to start moving, yet again, when it completes its cycle, it transmits the process to the last rotor. The inside of the rotors resembles scrambled wiring. A rotor's side has 26 junctions or contacts, accepting the incoming wires and outputting them to the other side of the rotor. The rotor moves each time a letter is typed in. Even if the same letter is passed sequentially, the rotor would move and a lamp representing a different output letter would light up. Thus, the dynamism of the rotors accounts for the complicated encryption. Finally, a plugboard at the front of the machine allows letters to be optionally swapped so that the machine it is interacting with is configured the same way. Alan Turing and his colleagues at Bletchley Park were able to break the Enigma codes and automate the process by developing a new machine called *the Bombe* [11]. The rotor machines were cracked because the same key had been used over an extended period of time and due to the use of old compromised keys while encrypting. In addition, the circuit's configuration showed that it was impossible for a letter to be encrypted as itself, which turned out to be a cryptographic flaw.

Transposition ciphers: Another kind of mapping is achieved by performing permutation on the plaintext letters. This technique is referred to as a transposition cipher. For example, a simple cipher of this sort is the rail fence technique. The plaintext is written as a sequence of diagonals and read off as a sequence of rows. The key is the number

of rows used to encode. Figure 5 shows a Rail Fence Cipher.

With today's computer power transposition ciphers can be broken quickly. Trying to compute the frequencies of the cipher letters is one method. Another way is testing possible rearrangements. For instance, one may try to read the cipher text backwards. If that does not yield the plain text then the rail fence technique could be tested. If that does not yield the plain text, one could check if two consecutive letters were switched.

Stream ciphers: A stream cipher generates a stream of bytes, one for each byte of the text we want to encrypt. An example is Rivest Cipher 4 (RC4), designed in 1987 by Ron Rivest. RC4 uses either 64-bit or 128-bit key sizes. Its most popular implementation is in WEP for 802.11 wireless networks and in SSL. RC4 consists of a key-scheduling algorithm, which initializes a permutation of all 256 possible bytes (since the permutation array has a length of 256 bytes) [3]. The permutation itself is started with a length key between 40 and 2048 bits. RC4 is also composed of pseudo-random generation algorithm (PRGA) which generates the stream of bits. PRGA has two 8-bit index pointers (i and j) on which, during the 256 iterations, it executes operations such as XOR-ing, swapping, modulo. Stream ciphers are vulnerable to attacks, mostly to bit-flipping [12] . Bit-flipping attack deals with changing a bit in a ciphertext so that it results in a predictable plaintext. It is not targeting the cipher itself but a message or a series of messages on a channel. For this reason, it could turn into a Denial of Service attack. We should never reuse a key with a stream cipher. The main reason being that we can recover the plaintext, using the keystream and the ciphertext. The keystream could be recovered as well using the plaintext and the ciphertext. Furthermore, if we use two ciphertexts from the same keystream, we can recover the XOR-encryption of the plaintexts.

Block ciphers: Block ciphers date back to late 1960s when IBM attempted to develop banking security systems [13]. The result was Lucifer, an encryption method, with 128-bit key and block size, aimed at protecting data for cash-dispensing system in the UK. However, it was not secure in any of its version implementations. When we use block ciphers, each block is encrypted independently producing a ciphertext block of equal size. The block size can be 64-bits, 128-bits or 256-bits. The ciphertext is generated from the plaintext and the key by iterating a *round function* (as we go through it several times). Input to the round functions consists of key and the output of previous round. Block ciphers are also known as *product* ciphers and are built with a *Feistel structure*, as first described by Horst Feistel of IBM in 1973. The plaintext is split into left and right halves. The Feistel design consists of a number of identical rounds of processing. During each round, substitution is performed on one half of the processed data, followed by a permutation that interchanges the two halves. What is more, the original key is expanded so that a different key is used for each round. Symmetric block encryption algorithms are based on this structure. In general, it is accepted that block ciphers are applicable to a broader range of applications in software as opposed to stream ciphers.

When we want to encrypt longer than a 16-byte plaintext (email, file, etc.) with a symmetric key block cipher algorithm, there exist several modes of operation:

- ECB (Electronic Code Book mode)
- CBC (Cipher Block Chaining mode)
- CFB (Cipher Feedback mode)
- OFB (Output Feedback mode)
- CTR (Counter mode)

These modes aim at providing confidentiality and protection for sensitive data. The latter three modes use the block cipher as a building block for a stream cipher [14]. To encrypt data with DES and AES our project utilizes ECB and CBC modes, respectively. Figure 7 shows a comparison between ECB and CBC. ECB requires that the length of the plaintext is a multiple of the block size of the cipher used. If the plaintext does not conform to the required length, it must be padded. Padding is achieved by appending as many zero bits as possible in order to reach a multiple of the block length (i.e. 8, 16, 32, etc.). In case the plaintext conforms to the length, an extra block is appended consisting of only padding bits. Furthermore, in ECB mode each block is encrypted separately. One of the advantages of ECB is that if a transmission problem occurs, the received encrypted block could still be decrypted [14]. Moreover, parallelization is allowed in ECB mode, i.e. one encryption unit encrypts block 1, the next one block 2, etc., which is useful in high-speed implementations [14]. Nevertheless, there are some

weaknesses associated with the ECB mode. During encryption identical plaintext blocks result in identical ciphertext blocks as long as the key does not change [14]. Hence, an attacker could easily deduce information. The ECB mode is susceptible to substitution attacks because once block mapping from plaintext to ciphertext is known, a sequence of ciphertexts can be manipulated [14]. With regard to CBC, all encrypted blocks are chained together such that a certain ciphertext depends not only on a certain block but on all previous plaintext blocks as well [15]. Each plaintext block also gets XORed with the previous ciphertext block prior to encryption [15]. In addition, the encryption is randomized by using an initialization vector (IV). In our case, the IV is new every time we encrypt and is always incremented when a new session starts. If we encrypt a string once with a first IV and a second time with a different IV, the two resulting ciphertext sequences look completely unrelated to each other. To strengthen the encryption method, a mode of operation is used, for AES we have chosen CBC mode.

DES: Cryptography gradually moved from hardware to software with the advent of computers. Most famous example of the block cipher design and the classic Feistel structure is the Data Encryption Standard (DES), designed by IBM under the advisement of NASA in 1977 and standardized 2 years later. It was meant to encipher sensitive but non classified data. DES complexity is comprised of a simple repetition of the primitives of transposition, substitution, split, concatenation and bit-wise operation. DES uses a 56-bit key. The cipher is thoroughly examined in section 4 of this report for it has been used during our hands-on approach.

TRIPLE DES (3DES): has replaced DES since the simpler version is susceptible to brute force attacks. 3DES is a more secure method of symmetric- key encryption, as it encrypts data three times in contrast to DES, i.e. the one 56-bit key becomes three individual keys rendering a 168-bit key. However, initiating three instances of DES, implies that 3DES is much slower than other methods of encryption. The text is encrypted firstly with key 1, then decrypted with key 2 and lastly encrypted once more by key 3. Triple DES offers a security level of 2^{112} instead of 2^{168},i.e., with only two keys of encryption, since the 168-bit key can be cumbersome to implement. This method is known as Encrypt-Decrypt-Encrypt (EDE): key 1 encrypts the message; then the message is decrypted using key 2, afterwards the text is encrypted again with key 2. We should mention that there exists double DES as well, which is composed of two successive instances of DES. 2DES offers a security level of 2^{57} instead of 2^{112} due to the cryptographic attack, called *meet-in-the-middle* [16]. If we take some plaintexts and encode them and at the same time take some encrypted values and start decrypting them, we only have to look for where they meet in the middle with the same value. Those intersections then reveal the key. 3DES avoids this as we would need to perform a third operation to tell if they met in the middle. Thus, it is not enough to look for where the first and last operation produce the same value.

AES: The Advanced Encryption Standard (AES) replaced DES in 2000 as the US Government encryption technique to protect classified information. The symmetric block cipher was developed by two Belgian cryptographers Joan Daemen and Vincent Rijmen. AES was designed to be efficient in both hardware and software. Again, it is thoroughly described in section 4 of the report.

3.2. Technical

One of the deliverables of this project is a chat system implemented in Python. The application employs ciphers reviewed in this report to encrypt messages. Python is a programming language created by Guido van Rossum in late 1980s in the Netherlands. Python is a free software and the latest version can be downloaded from www.python.org. Python is a simple and very powerful general purpose computer programming language. This allows for the language to remain fresh and current with the newest trends. Python has libraries for just about everything. It can be used for web development, web scraping, writing scripts, browser automation, GUI development, data analysis, machine learning, computer vision, and game development, etc. In addition, Python is an object-oriented programming language (OOP). There are four pillars of OOP: encapsulation, abstraction, inheritance, and polymorphism. Before turning to OOP, there is procedural programming, a simple and straightforward programming, that divides a program into a set of functions. However, as our program grows we end up with *spaghetti code*, i.e. many functions all over the place that are interdependent. Thus, OOP offers a solution to this issue. We can bundle a group of related variables (referred to as *properties*) and functions that operate on them (referred to as *methods*) into an object. This grouping is called *encapsulation*. Using this technique, we can reduce complexity and increase reusability. In Python, objects are

data and have a certain type (integer, float, list, etc.). Once we have created our objects we can manipulate and interact with them (append, sort, delete, concatenate, etc.). We can hide the details and complexity (e.g. some methods and properties) from the outside and show only the essentials through the process of *abstraction*. This is beneficial as we produce simpler interface and reduce the impact of change (i.e. no inner changes leak to the outside of the contained object). *Inheritance* is a mechanism that allows us to eliminate redundant code. *Polymorphism* is a technique that allows us to refactor long if/ else or switch/case statements.

Our platform is macOS and we can open Python on a Terminal. Moreover, we can use any Text Editor or IDLE of our choice. For this project, we have used the source code editor Visual Studio Code (VSC) developed by Microsoft. VSC is a lightweight yet powerful editor. It offers support for debugging, embedded Git control, syntax highlighting, snippets, extensions, smart code completion and code refactoring. A GitHub repository was set up to share and build our project. The link is provided in the annex of the report. GitHub is a web-based hosting service of open source projects for version control using Git.

To build the chat program for our project we need a client and a server. To establish a connection between both and be able to transmit information back and forth over the Internet, we need a socket link. Sockets aid the communication between these two entities. The client requests information from the server and the server carries out data to the client. Similarly, the client is a program as well. However, altogether, they are referred to as *client/server architecture*. For instance, when we visit a website, we are using a socket and accessing a port of the web server. In this case, the server has generally port 80 open, used to transfer HTTP data. Other websites have ports 21 and 20 open for FTP access, which is not very secure, some have port 22 intended for SSH. The lower number ports are specific ports, whereas the higher number ports signify general purpose rights. More often than not, questions arise regarding security when using higher number ports.

Our client should send an initial request to the server's port number, *the listening port*. To keep the communication over TCP (valid for every other protocols as well), we have the client's IP address and local port number as well as the server's IP address along with its port number. Each client that connects to the server gets unique set of values of two IP addresses and two port numbers in a tuple, a collection of items.

4. BSPro - A Third Bachelor Semester Project in BiCS-land

4.1. Requirements ($\pm$ 15% total words)

In this section we describe the algorithms which we implement in our chat application. Later on, we carry on a speed comparison between them.

Ciphers have been used long before apparition of computers. During Roman times, Julius Caesar invented an encryption for his private correspondence. Today, it is known as the Caesar cipher. Given the English alphabet, the cipher shifts a letter from the alphabet three places to the right, i.e. A is encrypted as D, B as E, etc. It is a monoalphabetic cipher. To decrypt the message, the other party needs to know both the algorithm and the shifting number. The secret key shared by the sender and the recipient of the message is k=3. Breaking the Caesar Cipher can be done by testing all possible shifts. Since an alphabet of length 26 is used, we have to test 26 shifts.

ROT13 is a simple monoalphabetic substitution cipher, a special class of Caesar cipher, that encodes a certain letter with another letter that is 13 positions after it. Only those letters which occur in the English alphabet are affected. Numbers, symbols, whitespace, and all other characters are left unchanged. It is an example of a cipher providing weak encryption, since both operations encryption and decryption are identical. Hence, this cipher is its own inverse. Because there are 26 letters in the English alphabet, if we wish to apply twice 13 it would give us one shift of 26. Thus, it leads us back to the original text. Moreover, the direction of the shift is of no importance, since it will always give the same output [17]. ROT13 is used in online forums as a means of hiding spoilers, punchlines, puzzle solutions, and offensive materials from the casual glance. Furthermore, it has inspired a variety of letter and word games online [18].

Another cipher our project deals with is the block cipher Data Encryption Standard (DES). In DES, we put a 64 bit block of plaintext. Its key, doing the processing, is 64 bit which is 8 bytes. For each byte there is one parity bit, therefore, the value in the key is only 56 bits. Thus,

there are 2 to the power of 56 different keys. The output ciphertext is a 64-bit block. Figure 8 shows a DES Round Encryption. From the 56-bit key, 16 bits are generated (one for each round). See Figure 9 for a DES Top View. Each DES round works consecutively, has the same operations and uses a different key. Each round uses a combination of proper substitution, where we take some bits, substituted with another combination of bits. Each DES round takes as an input the ciphertext produced by the previous round and outputs the ciphertext for the next round. The input is divided into a left half and a right half. The output left half is just the right half of the input. The right output is the result of XOR-ing the left half of the input and the output of the Mangler Function. This function takes as an input the 32-bit right half, expands it to 48-bit (bit-wise operation), then XORs it to 48-bit key, and finally uses the S-Boxes to substitute the 48-bit value into a 32-bit value. The algorithm process of decryption in DES is the same as the encryption process. It uses the ciphertext as an input to DES but the keys are run in reversed order, i.e. k = 16 is used as the first round of decryption, k = 15 is used as a second round of decryption and so on, so forth. Diffusion is one of the principles in encryption. It is achieved through permutation (initial transposition). Permutation works by changing the position of the bits in DES. The mixed bits are taken to a sub-box which receives the 56-bit key and the 64-bit plaintext. Once it completes processing, it outputs the 64 bits into another transposition subsection, which in turn produces a 64-bit ciphertext. The larger the block size, the key size and number of rounds means greater security. The two most significant attacks applicable to symmetric-key block ciphers are differential cryptanalysis and linear cryptanalysis. The former is a chosen plaintext attack, in which the attacker is able to select inputs and examine outputs in order to derive the key. The latter is a known plaintext attack: that is, it is premised on the attacker having information on a set of plaintexts and the corresponding ciphertexts. However, according to Stallings, DES has proven to be resistant to them [19]. By 1999, a computer could try every possible key in a couple of days rendering the cipher insecure.

AES supports a block length of 128 bits and key lengths of 128, 192, and 256 bits. Figure 10 shows a simplified concept behind AES algorithm. Thus, brute-force attacks are much harder to be launched against it. AES chops data up into 16-byte blocks, transferred to a *state array*, and then applies a series of substitutions (e.g. *substitute bytes*, *MixColumns*) and permutations (e.g. *ShiftRows*), based on the key value. We note that the state array undergoes various modifications throughout both encryption and decryption. Moreover, that way, it obscures the message, by adding diffusion, confusion and non-linearity and repeating the processes ten (for a 16-byte key) or more times (12 rounds for 24-byte key; 14 rounds for a 32-byte key) for each block. In order to *substitute bytes*, an S-box is used to perform a byte-by-byte substitution of the block. The other substitution, called *MixColumns*, utilizes arithmetic over (2^8) [20]. In fact, all encryption algorithms necessitate arithmetic operations. The key is expanded into an array of key schedule four-byte words. Employing a bitwise XOR of the current block with part of the expanded key is a stage called *AddRoundKey*, solely used on the key. Therefore, the cipher is locked around this stage adding to the security of the AES encryption. At the end, the state of the plaintext is copied to an output matrix where the bytes are ordered in a column. Similarly, the bytes of the expanded key, which form a word, are placed in the column of the matrix. Today, AES is used everywhere, from encrypting files and sensitive data, transmitting data over WiFi with WPA2, to accessing websites using HTTPS.

4.2. Design ($\pm$ 20% total words)

The technical part of our project consists of creating a chat application. Our source code editor is Visual Studio Code and we program in Python. We create a TCP server as well as a TCP client that connects to the server. To implement the client, we use a socket module, imported from Python's standard library. It serves as a communication endpoint with two parameters:

```
s= socket.socket
    (socket.AF_INET, socket.SOCK_STREAM)
```

The first parameter *AF_INET* stands for IPv4, i.e., the connection type we want to use, and the second parameter, *SOCK_STREAM*, means we are going to use TCP connection. We specify a variable for port number (on which the server is running) and IP, which holds the IP of the host machine it is running on. We use *gethostbyname* to look for the hosts IP since the host machine, used to conduct the technical simulation, does not have a fixed IP address and often when it reconnects to an Internet connection is assigned a new IP address.

```
ip = str(socket.gethostbyname
           (socket.gethostname()))
port = 1234
```

Once the port and IP address are known, we can connect to the server by passing the function to a data structure:

```
s.connect(((ip, port))
```

To implement the server, we use a socket module that serves as a communication endpoint with two parameters:

```
s= socket.socket
    (socket.AF_INET, socket.SOCK_STREAM)
```

For the client to be able to connect to the server, it should make use of the same port number. When the port and host have been bounded, we can listen to incoming connections with socket variable s and calling listen on it:

```
s.listen(10)
```

We pass the variable 10 which means that up to 10 different connections can be queued for the server to handle the requests. To accept requests from outside, we use connection variable and address variable:

```
conn, addr = s.accept()
```

which stores the IP import of the client, trying to connect and we call accept on the socket we have created. Upon establishing connection, the client can send information to the server that will cause it to echo back the data. The server can receive data up to size 4096 from the client:

```
incoming_message = conn.recv(4096)
```

Afterwards we can close the connection between client and server

```
conn.close()
```

on the server side, as well as close the socket itself as it allows connections to exist, on the client side,

```
s.close()
```

For our project, we aim at implementing several encryption and decryption applications in Python. The client stores the encryption algorithms, whereas the server holds the decryption methods. We make use of Caeser cipher, ROT13, DES, and AES. DES and AES have been implemented through the means of the crypto library PyCryptodome [21]. It is a self-contained Python package of low-level cryptographic primitives [21]. It supports PyPy, Python2 and Python3. PyCryptodome can be used as a replacement for the old PyCrypto library. We install all modules under Crypto package with pip3 install pycryptodome. However, having both PyCrypto and PyCryptodome installed at the same time is not a good idea as they will interfere with each other. If, however, one insists on having them both, it would be best to deploy them in a virtual environment. To have an independent library of the old PyCrypto, it suffices to type the following command in Terminal shell:

```
pip3 install pycryptodomex
```

and all modules are installed under Cryptodome package. PyCrypto and PyCryptodome can still coexist.

4.3. Production ($\pm$ 20% total words)

We implement all encryption algorithms with Python programming language. Table 1 shows the settings of the algorithms we are implementing.

Algorithm	Key size	Type of cipher
Caesar cipher	3	Letter substitution
ROT13	13	Letter substitution
DES	64 bits	Block cipher of size 64
AES	256 bits	Block cipher of size 256

TABLE 1: Algorithm settings

Caeser cipher is a substitution cipher and a type of shift cipher. Shift ciphers work by using the modulo operator to encrypt and decrypt messages. In the following lines we demonstrate the algorithm. We take an alphabetic message (a to z). We set the key, an integer from 0 to 25, since there are 26 letters in English alphabet, to be equal to 3.To encrypt, we do a right-shift letter by letter by the value of key mod 26. To decrypt, a left-shift of the message letter by letter is performed, subtracting the value of key and taking the modulus 26. The characters are calculated in the input string as this will be used for the number of iterations that need

to be done to encrypt and decrypt everything (one character at a time). Our code can be presented as follows: for every letter in the message, we find the letter that matches its position in the alphabet starting from 0. We calculate

```
newPosition = (position + key) mod 26
```

then convert *newPosition* into a letter that matches its order in the alphabet starting from 0. During the iteration it uses the alphabets' string as its base for shifting the characters, skipping those not part of the base string (such as numbers and symbols) to keep the structure of the input intact. Finally it outputs the ciphertext (*newMessage*) for us to use. The decryption process is similar; however, instead of adding the key, we subtract it during the modular operation.

ROT13 is a special case of Caeser cipher, therefore, both codes are similar. However, ROT13 is implemented by setting the value of the key to 13. The key is used by both encryption and decryption algorithms to shift the plaintext by 13 characters. Firstly, we initialize the letters. An ASCII table contains upper letters *A-Z* from positions 65 to 90, whereas, lower case letters *a-z* are within the range of 97 to 122. The *chr()* function is the opposite of *ord()*. The *ord()* function returns an integer representing the unicode code point of a character string. For example, *ord('z')* returns the integer 122. On the other hand, *chr()* returns the string character whose Unicode code point is an integer. For instance, *chr(122)* returns string z. What makes this technique fairly simple is that if we implement rot13() on the ciphertext, then we get back the plaintext since each letter in the text is rotated by the same shift. Thus, the same *encrypt_rot13()* function is defined for both encryption and decryption operations to rotate the letters. Depending on the provided letter (lower or upper case), the rot13 function encrypts according to the ASCII table and returns the ciphertext.

To implement DES, we import DES functional dependency from Cryptodome library module. Then we create and specify a fixed *key = mysecret* of length 8 bytes, a password that shall be used to encrypt the text. We have created a pad function, *despad*, taking as input *plaintext*. Within this function, we run a while loop checking the length of text mod 8, i.e. the modulus divides it and shows the remainder. If remainder is not equal to zero, we add a blank space at the end of text such that it becomes multiple of 8 (e.g., 8, 16, 24, 32, etc.). Otherwise the encryption is not going to work as DES encryption algorithm, takes input in 8 bytes. For instance, if we type abcd, the function adds four empty spaces at the end of abcd to make sure it is a total of 8 characters/bytes. Then, we create an object, called *des*. We use function *new()* of module DES that takes two parameters: key and *DES.MODE_ECB* which is the encryption mode. We have created a variable taking an input from the client that shall be encrypted. The variable *padded_text* uses the despad() function on the client's input to make sure it is a multiple of 8 and if necessary adds empty spaces to conform to the multiple condition. Next, we have created the variable *encrypted_text* that uses the function *encrypt()* on the *padded_text* and turns it into an encrypted cipher. To represent ciphertexts, it is more readable to use hex strings rather than bytes. Moreover, hex() makes it easier to parse the decryption input, implemented in the server. To be sure that data remains intact without modification during transfer from client to server, we need to encode it. To reverse the process, the server needs to decode the message. In order to decrypt, we run *des* submodule from the library and pass function *decrypt()* on it. Since we have used padding before encryption, we need to unpad the recovered cleartext by using rstrip().

AES has been implemented using the *PyCryptodome* library. We import the module AES along with the hash package and hash object *SHA-256*. In function *make_key()*, SHA-256 produces a 256-bit digest of a password. *Digest()* returns a binary digest of the message that has been hashed. Since *digest()* produces a random output it is impossible to derive the original input data. On the client side, the *aesencrypt()* function has a pad variable which appends to the back as many bytes we need to reach the next 16-byte boundary, before encrypting the bytes message. On the server side, an unpad function is used to reverse the padding executed by the client. The *block_size* is the size of the message block in bytes. The IV (initialization vector) is a unique and random piece of data used to initialize the processing. This setting does not allow an attacker to get relations between encrypted messages segments. The IV does not need to be private so that we can send it with the ciphertext without risk. Its length is equal to 16 bytes. The key and IV have been loaded on separate files and are used for both encryption and decryption. The IV is written and overwritten in a file each time it is used to encrypt a message. And it is being read from the file during decryption by the server. When we couple the key with the mode of operation CBC (Ciphertext Block Chaining), where the IV

is randomly generated for each new message. Thus, we can encrypt a lot of data under the same key. CBC requires the length of plaintext and ciphertext to be always a multiple of 16 bytes. The function *new()* instantiates a new object for the AES algorithm. It has 3 parameters: key (in bytes), mode *(MODE_CBC)* and IV (in bytes as a read-only attribute). The function *new()* returns a CBC cipher object. To encode and decode the message, base64 has been used. It allows to transfer over a medium relying on textual data by converting from binary data to text strings and vice versa. The method *encrypt()* (and likewise *decrypt()*) of a CBC cipher object expects data to have length multiple of the block size, i.e., 16 bytes for AES. The message is being translated from a Unicode string into a sequence of bytes through UTF-8 (Unicode Transformation Format) encoding. *Encode(utf-8)* and *Decode(utf-8)* return byte representations of the unicode string, encoded in UTF-8.

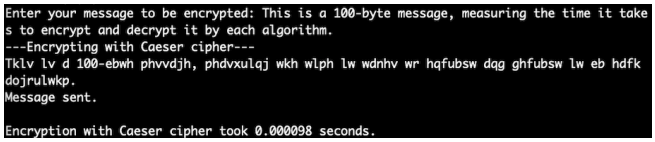
4.4. Assessment ($\pm$ 15% total words)

We have used a TCP transport-layer protocol with the same IP address and port number for both client and server. We have shown a secure communication over the Internet between a client and a server, having the choice to encrypt and decrypt with four different algorithms. We highlight that both client and server should support the same set of encryption algorithms, modes of operation, as well as keys, set for encryption and decryption processes. In this section, we assess the encryption and decryption operations utilized by the algorithms. We also take into account the modes of operations and supplementary features such as IV and key generation. Furthermore, we conduct a performance analysis choosing as a factor the speed necessary to encipher and decipher information.

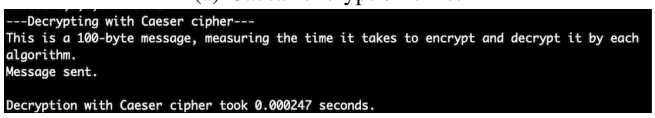
In general, the mode of operation plays an important role in the security of the cipher. We have used two block ciphers, DES and AES. ECB (Electronic Code Book), used along DES in our project, splits up the plaintext into blocks and encrypts them independently of each other in a parallel operation. This is an advantage of ECB. ECB does not use an initialization vector to launch the encryption. Thus, it executes the same algorithm to encrypt data. However, when multiple messages are encrypted, ECB cannot be considered secure, because identical plaintext blocks result in identical ciphertext, also known as deterministic encryption. Hence, repetition makes it easier for an attacker to guess the original message. CBC cannot operate in parallel since it waits on finishing encryption on each block. CBC, used along AES in our project, introduces an initialization vector (IV), which alters the plaintext before the start of the encryption process. The vector ensures that the initial ciphertext blocks are different even with an identical plaintext. To accomplish this, the IV is unpredictable, unique and cryptographically random for each separate key to avoid deterministic encryption. As both the encryption and decryption need to be performed using the same unique IV. If we use a different IV in decryption, we would get a garbled message indicating an erroneous use of IV. Furthermore, the IV must be of the same length as the block size (e.g., given 64-bit block size, 64-bit IV, etc.) As mentioned earlier, when a predictable IV is used, CBC is considered vulnerable as plaintexts could be deduced. Hence, the IV cannot be transmitted alongside the cipher. The IV is the result of the previous encrypted block, XORed with the plaintext block before encryption. Hence, with CBC, every block depends on the output of the previous block. Nevertheless, the disadvantage of this property is that if there is an error or failure in a given block, it will result in the following block as well. The key in AES is hashed to provide a 256-bit output, then the hash is encrypted, and a 256-bits encrypted output is produced. The IV is then used to encrypt the next 256-bits. This results in a chaining algorithm. Considering the comparison between the two modes of operation, it is not a good idea to use ECB for message confidentiality as it is a rather simple and raw mode for a block cipher. Instead, we can turn to CBC and its random use of IV.

We conducted a performance analysis of the implemented algorithms. The simulation was administered on a 3,1 GHz Intel Core i7 processor with 16 GB 1867 MHz DDR3 memory. The experiments have been performed couple of times to assure that the results are consistent and are valid to compare the different algorithms. The time needed for each method to encrypt and decrypt a message of 100 bytes was measured in seconds. The message is the following: *This is a 100-byte message, measuring the time it takes to encrypt and decrypt it by each algorithm*. It is imperative to note that we prepared the sentence in advance and copied-pasted it to the console. If we take the time to write a message on the spot, the time measurements will differ.

Figures 1 to 4 represent the time it took for each algorithm to encrypt and decrypt the 100-byte message. Table 2 summarizes the results of the experiment. The

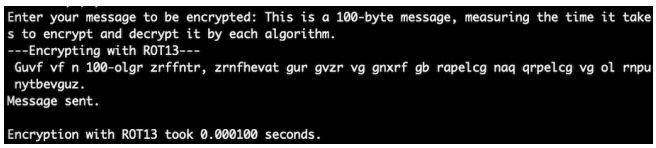


(a) Caesar encryption time.

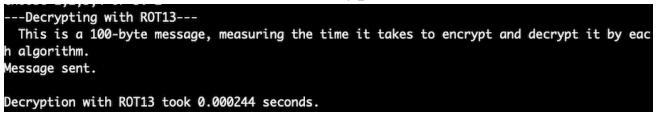


(b) Caesar decryption time.

Figure 1: Caesar cipher Encryption/Decryption time of a 100-byte message.



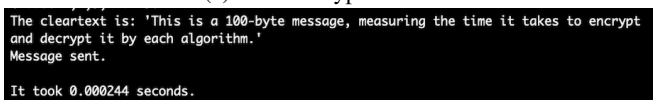
(a) ROT13 encryption time.



(b) ROT13 decryption time.

Figure 2: ROT13 Encryption/Decryption time of a 100-byte message.

(a) DES encryption time.



(b) DES decryption time.

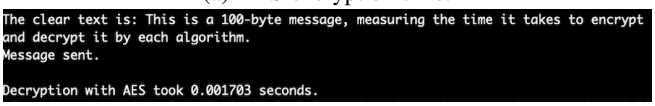
Figure 3: DES Encryption/Decryption time of a 100-byte message.

	Time for encryption	Time for decryption
Caesar cipher	0.000098	0.000247
ROT13	0.000100	0.000244
DES	0.001571	0.000244
AES	0.002869	0.001703

TABLE 2: Comparative execution time (in seconds) of implemented algorithms

security features of each algorithm as their strength against cryptographic attacks is already known and discussed. The

(a) DES encryption time.



(b) DES decryption time.

Figure 4: AES Encryption/Decryption time of a 100-byte message.

factor we have chosen to determine the performance is the speed to encrypt and decrypt the data of size 100 bytes. It is easy to observe that Caesar cipher and ROT13 performed similarly in time, as expected, since they are identical cases. Moreover, DES outputs faster than AES because it is not relying on initialization vector (iv) and because the ECB mode is simpler than CBC, but these make it a weak encryption mechanism. AES performed poorly in terms of speed since CBC mode adds extra processing time due to its key-chaining nature. Nevertheless, CBC mode of operation is better than ECB and for that AES is considered the most secure of all four algorithms. We note that AES needs almost twice the time of DES to encrypt the 100-byte message and roughly ten times to produce the plaintext. In addition, AES consumes more resources and requires more processing time since it utilizes an iv and CBC mode. Overall, the slow performance of AES can be considered as negligible for applications that require secure transmission of sensitive data over the Internet.

In addition, to prevent access to the message from the outside while transferring information between clients and server, we should not store any data as plaintext. The reason being is that any stored piece of information can be accessed during an attack on the server, such as a social engineering or man in the middle attack. Another remark to be made is to never create our own cryptography security schemes and to never implement an existing standard on our own. We will most probably do major security mistakes leading to insecure applications. However, for the purpose of academic education, we could not obey this rule of law. As a future work, we could simulate a couple of attack scenarios such as a frequency letter analysis (on Caesar cipher and ROT13), man-in-the-middle attack or brute force (on both AES and

DES) and test their success.

Acknowledgment

The authors would like to thank the BiCS management and education team for the amazing work done.

5. Conclusion

Online communication has created a convenient and simple method for us to connect and transmit data over communication channels. Thus, transferring information over the Internet requires higher level of security when it comes to sensitive data. This could be achieved through the means of cryptosystems and cryptography. Primitives of symmetric-key cryptography lay the foundation of our report. Surveying its mechanisms, we have implemented a simple chat application in Python that encrypts messages before transferring them over the Internet. Substitution and transposition ciphers, alongside block and stream ciphers have been introduced in our project. The palette of reviewed ciphers stretches from simple to more complex ones: Caesar cipher, ROT13, One-time pad, Vigenere cipher, Enigma machine, Rivest Cipher 4, Data Encryption Standard (DES), 3DES, Advanced Encryption Standard (AES). Existing modes of operations (ECB, CBC, CFB, OFB, CTR) that can be applied to block and stream ciphers were also presented as they allow for larger data to be encrypted in order to guarantee its confidentiality. A client-server architecture was built thanks to a TCP transport layer protocol with same IP address and port number. Once a connection was established, communication flow of messages was tested. The client script contains the encryption functions of the chosen four algorithms (Caesar cipher, ROT13, DES, AES) in order to encrypt messages before sending them to the server over the Internet. The recipient, aka the server, on the other side, is responsible for decrypting the messages, hence it holds all decryption operations, in order to echo back to the client the deciphered messages. These encryption and decryption techniques depend upon the type of data and the channel through which the data is being communicated. Throughout the report, we have drawn a comparison analysis of the proposed mechanisms based on their basic features, advantages, drawbacks and applications. The performance analysis, at the end, takes into account the speed factor, i.e., the time it takes for each algorithm to encrypt and decrypt messages. The results showed equal performance by Caesar cipher and ROT13. DES outperformed AES. Even though AES took more processing time, it proved to be the most secure encryption algorithm due to the initialization vector and the better mode of operation, CBC. DES has no IV and is built alongside a weak and simple mode of operation ECB. Moreover, nowadays, DES is considered as a weak encryption standard. In 2012, a system was designed to crack any DES key in 26 hours [22]. Nevertheless, it can still be used for applications where sensitive data is not required. AES replaced DES as it is more powerful and offers a great number of possible keys, 2^{256}, which has no comparison. Even the number of atoms in the universe is smaller than AES keys (with a range from between 10^{78} to 10^{82} [23]. We suppose, even if we had many supercomputers to each test the 2^{256} keys per second, the time to crack AES would still be much longer than the age of the universe (estimated at 13.82 billion years [24]). The report concludes that encryption algorithms attempt to thwart risks posed by untrusted networks, such as the Internet. In response to these risks, we deploy methods to encrypt messages before sending them over the Internet, so that outside access is prevented and content is preserved. In fine, sensitive data encryption should be on the radar of enterprises, governments, banks, retailers, institutions of higher learning and health organizations nowadays.

6. Appendix

A link to the GitHub repository of the project: https://github.com/dmarinova1/BSP-S3-Encryption-algorithms

WECRLTEERDSOEEFEAOCAIVDEN

W . . . E . . . . C . . . .R . . . .L . . . .T . . . . E .
.E . R . D . S . O . E . E . F . E . A . O . C .
. A . . . . I . . . V . . . .D . . . .E . . . .N . .

Figure 5: Rail Fence Cipher. Ref: Taken from Security I lecture notes

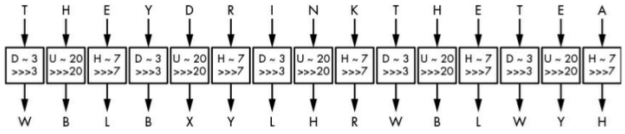


Figure 6: Vigenere Cipher. Ref: Taken from Security I lecture notes

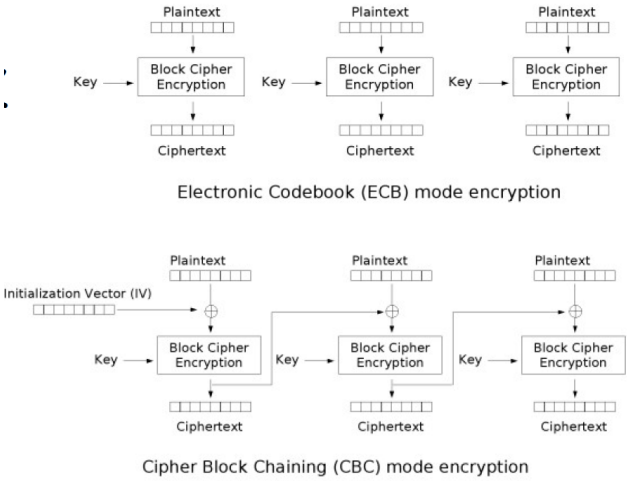


Figure 7: ECB and CBC modes of operation. Ref: Taken from Security I lecture notes

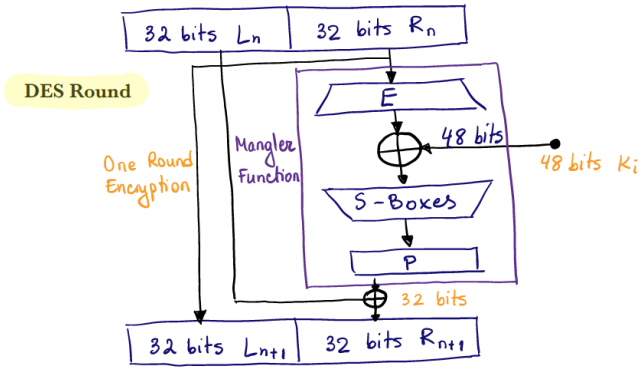


Figure 8: DES Round Encryption. Ref: As presented in Security I course

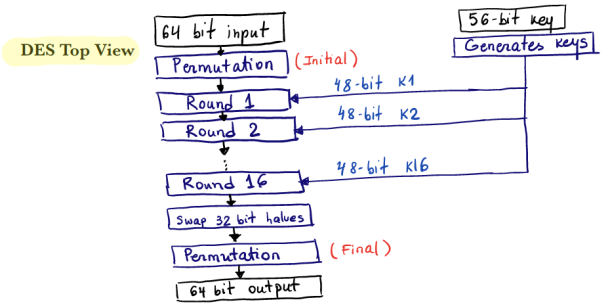


Figure 9: DES Top View. Ref: As presented in Security I course

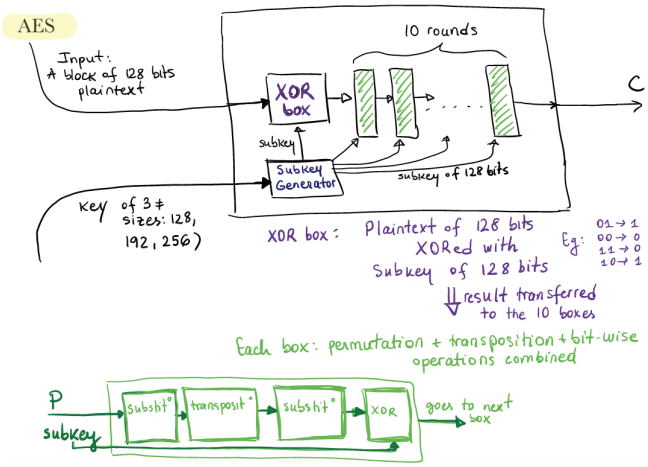


Figure 10: AES. Ref: General idea from *Cryptography and Network Security* by Stallings. Ed.7

References

[1] Wikipedia contributors, "Cryptography — Wikipedia, the free encyclopedia," 2018, [Online; accessed 26-October-2018]. [Online]. Available: https://en.wikipedia.org/wiki/Cryptography

[2] ——, "Public-key cryptography — Wikipedia, the free encyclopedia," 2018, [Online; accessed 26-October-2018].

[3] W. Stallings, *Cryptography and Network Security*, 7th ed. Pearson, 2017.

[4] "Internet security," Nov 2018. [Online]. Available: https://en.wikipedia.org/wiki/Internet_security

[5] “Computer security,” Dec 2018. [Online]. Available: https://en.wikipedia.org/wiki/Computer_security

[6] “Cryptographic primitive,” Nov 2018. [Online]. Available: https://en.wikipedia.org/wiki/Cryptographic_primitive

[7] “Symmetric-key algorithm,” Nov 2018. [Online]. Available: https://en.wikipedia.org/wiki/Symmetric-key_algorithm

[8] W. Stallings, *Cryptography and Network Security*, 5th ed. Pearson, 2011.

[9] M. Curtin, *Snake Oil Warning Signs: Encryption Software to Avoid*, 1998.

[10] “The black chamber: Mary queen of scots,” https://www.simonsingh.net/The_Black_Chamber/maryqueenofscots.html, accessed: 2018-10-26.

[11] “Alan turing: Creator of modern computing,” https://www.bbc.com/timelines/z8bgr82, accessed: 2018-10-26.

[12] “Rc4,” https://en.wikipedia.org/wiki/RC4, accessed: 2018-10-26.

[13] “Ibm: Cryptography for a connected world,” http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/cryptography/, accessed: 2018-10-26.

[14] C. Paar and J. Pelzl, *Understanding cryptography a textbook for students and practitioners*. Springer, 2010.

[15] Dworkin and Morris, “Recommendation for block cipher modes of operation: Methods and techniques,” Dec 2001. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-38a/final

[16] Wikipedia contributors, “Meet-in-the-middle attack — Wikipedia, the free encyclopedia,” https://en.wikipedia.org/w/index.php?title=Meet-in-the-middle_attack&oldid=856926199, 2018, [Online; accessed 26-October-2018].

[17] C. Swenson, *Modern cryptanalysis: techniques for advanced code breaking*. John Wiley & Sons, 2008.

[18] Wikipedia contributors, “Rot13 — Wikipedia, the free encyclopedia,” https://en.wikipedia.org/w/index.php?title=ROT13&oldid=865606310, 2018, [Online; accessed 26-October-2018].

[19] W. Stallings, *Cryptography and Network Security*, 5th ed. Pearson, 2011.

[20] ——, *Cryptography and Network Security*, 7th ed. Pearson, 2017.

[21] “Pycryptodome.” [Online]. Available: https://pycryptodome.readthedocs.io/en/latest/src/introduction.html

[22] [Online]. Available: https://crack.sh/

[23] A. M. Helmenstine and Helmenstine, “How many atoms are there in the universe?” [Online]. Available: https://www.thoughtco.com/number-of-atoms-in-the-universe-603795

[24] N. T. Redd, “How old is the universe?” Nov 2017. [Online]. Available: https://www.space.com/24054-how-old-is-the-universe.html