Shape

Description automatically generated

DEPARTMENT OF COMPUTER SCIENCE

Data Privacy and Security - CS 528

Project Report:

Cryptography Ciphers: Vulnerabilities and Assurances



**By: Alexis Daniel Ortega (A20435250)**

Contents

Page

1. Team Description………………………………………………………… 3
2. Motivation & Application……………………………………………… 3
3. Goal & Design……………………………………………………………….. 3
4. Caesar’s Cipher…………………………………………………………….. 4-5
   * 1. Vulnerabilities & Assurances
5. AES………………………………………………………….…………………… 4-11
   * 1. Vulnerabilities & Assurances
6. RHA…………………………………………………………………………….. 11-15
   * 1. Vulnerabilities & Assurances
7. References………………………………………………………….……… 16
8. **Team Description**

The project design, implementation, testing, and documentation is brought forth by a solo Illinois Institute of Technology 3rd Semester Master of Computer Science student, Alexis Ortega. Alexis completed a Bachelor of Arts degree in Political Science at Northern Illinois University on track to attend Law School, but decided to instead make a career change toward Computer Science. He began his academic career at IIT in 2020, and although he has little background in Data Science, he has a curiosity and enthusiasm for Information Security. Alexis is projected to graduate from IIT this summer with specializations in Information Security and Assurance and Software Engineering.

1. **Motivation & Application**

There exists a growing need for data protection and security as the world rapidly and inevitably moves toward being totally digital. Data and encryption and protection is used in most aspects of the digital world. It can be used to secure military communications where messages, images, and other files can be safely sent back and forth between allies with little worry that an adversary can acquire the sensitive information. More locally and perhaps more commonly it is also used to protect user information in databases such as account information, medical history, browser history, etc.

Cryptographic Encryption can be used to Encrypt various forms of files from images, files, text, and video. To ensure the privacy and safety of such files there exists various ciphers that can be applied as well as some different protocols. A few will be displayed in this project

1. **Goal & Design**

This project shows the encryption, decryption, and potential attacks through various ciphers (Caesar's, AES, RSA, MD5, symmetric, and asymmetric cryptosystems) and file types (text input, text file, images, and large data files)

1. **Caesar’s Cipher**

The Caesar’s Cipher is the simplest cryptosystem among those that will be shown in this report, and serves strictly as a historical context for modern day cryptography. This basic cipher is based originally on simple substitution. Meaning that one letter in the alphabet will be switched with another character. This switch can be done a number of ways, and *how* it is done is the key in this cipher. For example in Figure 4.1 a three letter shift was implemented, so the encrypted message will have each letter be substituted for the one that is exactly 3 letters to it’s right.



*Figure 4.1*

To decrypt this message we simply go 3 letters in the opposite direction for each letter. As shown in Figure 4.2. The original message *‘attackatdawn’* is encrypted to produce the ciphertext *‘dwwdfndwgzq’*. Once decrypted (if one knows that the key is the letters are shifted three to the right, the decryption can be done easily) the plaintext is revealed.

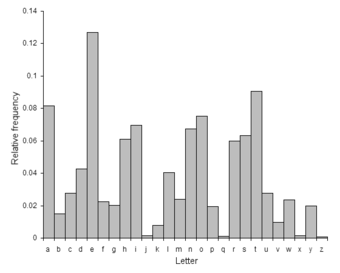
Text

Description automatically generated

*Figure 4.2*

* + 1. Vulnerabilities & Assurances

Obviously this cipher does not offer a great deal of assurance with means of protection. Although a plaintext can be encrypted and the plaintext’s original meaning can be a bit difficult to decipher, a *Frequency Analysis*  can be implemented to decipher the key and find the plaintext translation of the cipher text easily. A *Frequency Analysis* becomes much more simple with any background knowledge. For example, if the adversary knows that the language the message is in - English - then they can simply substitute the most frequently used letters in the cipher for those that are most frequently used in the English language such as Figure 4.3 below.

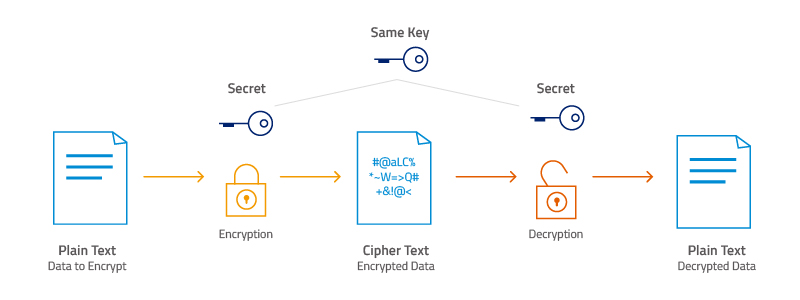


*Figure 4.3*

Using the frequency of these letters we clearly see that ‘e’ is the most common letter, so we find the letter that is most used in the cipher text and replace it with ‘e’, and we do the same for the next most commonly used letter. There are also letters that appear in pairs more frequently than others like ‘tt’. Also, there are letters that often appear right after eachother such as ‘th’, ‘wh’, etc. Using this knowledge cracking the key is very easy. If the user notices a pattern in the displacement of the substitution like the 3 letter shift in this example the key will be totally compromised. This can be experimented with in the *Caesar.py* program in the files attached.

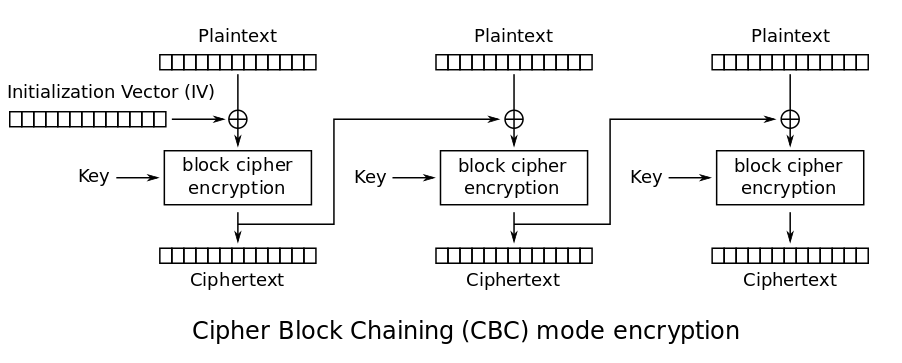
1. **AES – Symmetric Encryption**

The AES block cipher is a *symmetric encryption cryptosystem* that uses a secret shared key among Alice and Bob as well as an Initial Vector XOR with a *plaintext* to encrypt and decrypt a text string, text file, image, or video. The protocol is depicted in *Figure 5.1* and *Figure 5.2* below.



*Figure 5.1*

Alice picks the file she would like to encrypt, applies the IV and secret share key to it through the encryption algorithm, and produces the *ciphertext.* Bob receives the ciphertext, applies his shared secret key and the IV to it, and produces the plaintext that Alice sent over.

****

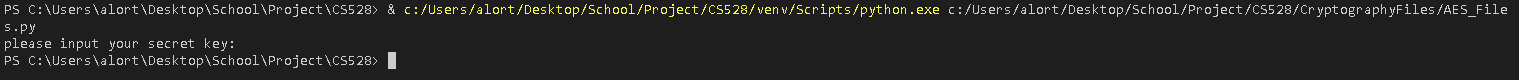
*Figure 5.2*

AES is block cipher, however, this means that the encryption happens in specific byte sizes. To ensure that a message is always within these byte sizes a layer of padding may be necessary, so for example if we choose a block size of 256 bytes, but our message is only 200 bytes, then we must add a layer of 56 bytes worth of padding. When decrypting the message one must *unpad* it as well.

This protocol in the python programs *AES.py, AESDecrypt.py, AES\_File.py,* and *AES\_File\_Decrypt.py* can be experimented with. The *AES.py* and *AESDecrypt.py* does a simple string encryption and decryption between an Alice and Bob scenario. This protocol expanded can be seen in the *AES\_File.py* and *AES\_File\_Decrypt.py*. Below in *Figures 5.3*

Text

Description automatically generated



*Figure 5.3*

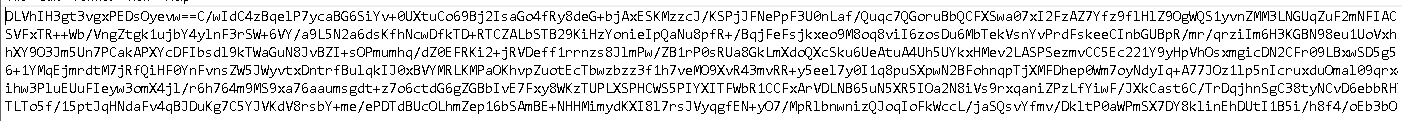
The program runs, then asks the user for their secret key of choice, this key is shared with Bob to Decrypt the message, so don’t forget or lose it. One can change the file they wish to encrypt by changing the path of the file in *Line 25* of the code block above.

Text

Description automatically generated with medium confidence

*Figure 5.4*

The text file provided (*Figure 5.4*) is encrypted to create the file cipher text below (*Figure 5.5*)



*Figure 5.5*

This is then sent to Bob, who decrypts the ciphertext from the file provided and creates a new file (*Figure 5.6*)

A picture containing text

Description automatically generated

*Figure 5.6*

The code to decrypt this text is shown below in *Figure 5.7.*

Text

Description automatically generated

*Figure 5.7*

The same can be done for an image, video, or really any file type including a .csv containing database information. Example of image below in *Figure 5.8* and *Figure 5.9*

A dog wearing a vest

Description automatically generated with medium confidence

*Figure 5.8*

Text

Description automatically generated

*Figure 5.9*

*Figure 5.10* portrays the way AES encryption works. Basically, the algorithm works in rounds with multiple layers of Byte Substitution, mixing, shifting of rows, and splitting the key to be added into the bytes. This creates *confusion, permutation,* and *diffusion*.

Diagram

Description automatically generated

*Figure 5.10*

* + 1. Vulnerabilities & Assurances

The AES CBC cipher offers quick and reliable encryption that is very difficult to crack. An adversary would only be able to attack this cipher with *Bruteforce* calculation of the key and IV, or by exploiting the implementation of the algorithm, which is more of a human error than a shortcoming of the algorithm. These exploits come from instances such as a secret key leak, reusing the same IV for multiple encryptions and decryptions where an adversary can intercept the ciphertext, etc. An adversary attack based on background information gained from information gained is called a *Side-channel attack.* These vulnerabilities are for the most part negligible and the cipher can be used with near certainty that the information is secure when implemented correctly. There is, however, a key distribution issue in AES because each pair of personnel in the group needs to have a key. So if there are 4 people in a network they will each need a key for each other such that

n x (n-1) / 2

4 \* 3 / 2 = 6 different pairs of keys among the network.

This can prove to get out of control in larger networks.

1. **RSA – Asymmetric Encryption**

RSA is an *asymmetric encryption cryptosystem* meaning that the users each have different keys. This covers the issue from the symmetric encryption network such that each user has 2 keys. A public key they will use to send encrypted messages, and a private key that they will use to decrypt messages. This makes the key distribution O(n); a much better outcome. To distribute these keys, a secure connection must be made. These keys are also rather large, so computation is slower than symmetric. To generate the keys the algorithm shown in *Figure 6.1* is used.

Graphical user interface, text, application

Description automatically generated

*Figure 6.1*

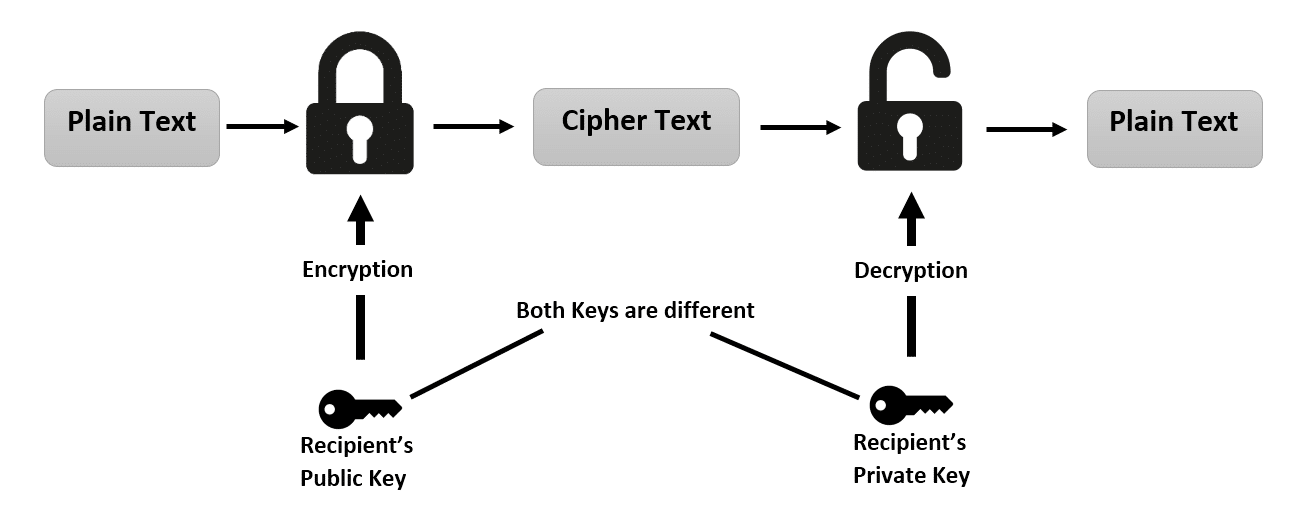
These keys are mathematically connected so that the private key can be used to decipher a ciphertext encrypted with a public key. Asymmetric encryption also applies a *signature* so that Alice and Bob can ensure that they are communicating with one another as shown in *Figure 6.2* below from the attached program *RSA.py* below. A signature must be verified to guarantee that the communication is secure.

Text

Description automatically generated



*Figure 6.2*



*Figure 6.3*

*Figure 6.3* shows the protocol of RSA Encryption and Decryption which is also carried out in the program *RSA.py* depicted in *Figure 6.4* below

Text

Description automatically generated

The program creates a public key (*Figure 6.5*) and a private key (*Figure 6.6*)

Text

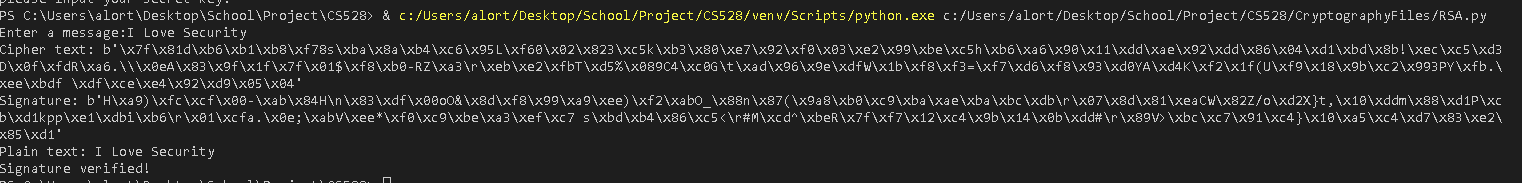
Description automatically generated

*Figure 6.5*

Text

Description automatically generated

*Figure 6.6*

*Figure 6.7*

*Figure 6.7* depicts the cipher text and the decrypted message that comes from the input message. The functions to generate keys , encrypt, and decrypt with a signature from *RSA.py* are shown below in *Figure 6.8, Figure 6.9, Figure 6.10* and *Figure 6.11* respectively.

Text

Description automatically generated

*Figure 6.8*

Text

Description automatically generated

*Figure 6.9*

Graphical user interface, application

Description automatically generated

*Figure 6.10*

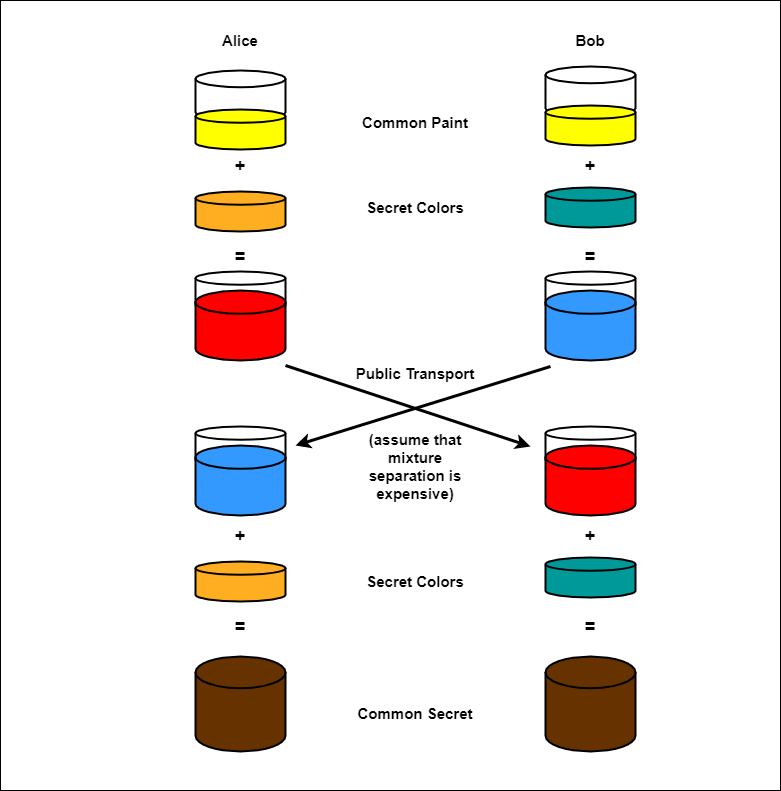
A screenshot of a computer

Description automatically generated with medium confidence

*Figure 6.11*

* + 1. Vulnerabilities & Assurances

Much like AES, RSA does not have much vulnerability in its algorithm, but exploitation of its implementation can prove to reduce its assurance of security. The RSA algorithm is susceptible to time attacks, but can be countered by adding delays (*wait()*) or by applying randomization to a program.There also exists the problem of exchanging Keys to which the Diffie-Hellman Key Exchange can help depicted in *Figure 6.12*.



*Figure 6.12*

References

* References
  + Inspired by SEED Laboratory
  + *Computer Security: Principles and Practice*
* *Special thanks to Alfred for his support in the making of this project*

A dog wearing a vest

Description automatically generated with medium confidence