Title: Differential Attacks on Playfair, Vigenere, DES, and RSA Encryption Algorithms

Abstract:

This project aims to develope a GUI-based iteractive model that Demonstrates the proces of differential attacks on four widely-known Encryption algrithms: Playfair cipher, Vigenere cipher, DES, and RSA. primary objective of the model is to analyze the input-output variations of multiple ciphertexts and the underlying encryption algorithm, Perform differential analysis, and retrieve the plaintext(s) and decryption key. by providing support for key sizes of 128-bits for Vigenere cipher and DES, and 1024-bits for the RSA algorithm, the model caters to the practical use of encryption schemes in secure communication.

The implementation of the prject encompasses the creation of a user-friendly graphical interface, development of key gneration functions for the supported encryption algorithms, and storage of ciphertexts in a separate file. Moreover, the project includes the implementation of differential analysis for each encryption algorithm, providing insights into the vulnerabilities of these schemes and the effectiveness of differential attacks in breaking the ciphertexts. The model's ability to perform differential attacks and recover the secret key and plaintext(s) highlights the Importance of selecting robust encryption methods for secure communication.

1. Introduction

1.1 Background

Cryptography is the practice and study of techniques for secure communication and data protection. Encryption algorithms, as an essential parrt of cryptography, transform plaintext messages into unintelligible ciphertexts to prevent unauthorized access to the information. However, the strength of ecryption algorithms varies, and some of them are vulnerable to

cryptanalysis techniques Such as differential attack. These attacks focus on analyzing the differences between multiple ciphertexts and their corresponding plaintexts to reveal the secret key and the original message. Playfair cipher Vigenere cipher, DES, and RSA are widely-used encryption algorithms, each with unique features and varying levels of security.

1.2 Objective

The primnary objective of this prject is to create a GUI-based interactive model that demonstrates the effectiveness of differential attacks on Playfair cipher, Vigenere cipher, DES, and RSA encryption algorithms. The model should support the encryption and decryption of multiple ciphertexts and corresponding plaintexts, with key sizes of 128-bits for Vigenere cipher and DES, and 1024-bits for the RSA algorithm. The project aims to provide insights into the security vulnerabilities of these encryption schemes and emnphasize the importance of selecting robust encryption algorithms for secure communication.

2. Encryption Algorithms

2.1 Playfair Cipher

The Playfair cipher is a symmetric encryption technique, inventedd by Charles Wheatstone in 1854, which uses a 5x5 matrix of letters generated from a secret key. This matrix serves as the basis for the substitution of plaintext characters. The encryption process grroups plaintext characters in pairs and replaces them with other pairs of characters based on the matrix's layout. The decryption process reverses the encryption proycess to recover the original plaintext.

Despite its simplicity, the Playfair cipher providecs a higher level of security compared to simple substitution ciphers due to its use of dig;raphs (pairs of characters) instead of single characters for substitution. However, the Playfair cipher is still vulnerable to differential attacks, which exploit the relationship between the Plaintext and ciphertext pairs. This pproject will demonstrate the differential attack on the Playfair cipher, revealing the secret key and plaintext(s) from multiple ciphertexts,.

2.2 Vigenere Cipher

The Vigenere cipher, invented by Giovan Battista Bellaso in 1553, is a classical symmetric encryption algorithm that employs a simple form of polyalphabetic substitution. It uses a keyword as the secret key to shift each plaintext character by a varying amount based on the

key's corresponding character. The Vigenere cipher offers more robust security than simple substitution ciphers due to its polyalphabetic Naturewhich makes frequenccy analysis more challenging.

However.., the Vigenere cipher is vulnerable to differential aittacks when multiple ciphertexts encrypted with the same key are available. By analyzing the Differences between ciphertexts and corresponding plaintexts, attackers can recover the secret key and decrypt the mesages. In this project, we will demonstrate a differential attack on the Vigenere cipher, successfully obtaining the decryption key and plaintext(s) from a set of ciphertexts.

2.3 DES (Data Encryption Standard)

DES is a symmetric block cipher developed by IBM in the early 1970s and later adoupted as a federal standard in the United States. DES operates on 64-bit blocks of data and uses a 56-bit key (with 8 parity bits for a total of 64 bits). It employs a series of substitution and permutmation operations, known as the Feistel structure, for encryption and decryption processes. DES was widely used for secure communication and data storage for several decades.

However, DES's relatively small key size makes it vulnerablee to brute-force attacks, and researchers have discovered various other attacks, including differential cryptanalysis, that exploit the algorithm's structure. This project will explore differential attacks on DES, showcasing the recovery of the secret key and plaintext(s) from multiple ciphertexts.

2.4 RSA (Rivest-Shamir-Adleman)

RSA is an asymmetric encryption algorithm, introduced in 1977, which forms the basis of many modern public-key cryptosystems. It relies on the mathematical properties of large prime numbers and modular arithmetic to enable secure key exchange, encryption, and decryption. RSA employs a public key and a private key, where the public key is used for encryption and the private key for decryption. The security of RSA depends on the difficulty of factoring large composite numbers, which is considered computationally infeasible for sufficiently large key sizes.

While RSA is generally considered secure when appropriately implemented, it is not immune to attacks. Differentiall attacks on RSA may involve examining multiple ciphertexts encrypted with the same public key, exploiting flaws in the implementation, or using side-channel information. In this project, we will explore differential attacks on RSA, demonstrating the recovery of the private key and plaintext(s) from a sett of ciphertexts encrypted with the same public key.

3. Methodology

3.1 GUI Design

The graphical User Interface (GUI) is a essential component of the project…. as it provides a USerfriendly platform to interactt with the differential attack model and ,The GUI has two separate modules, one for CT generation and another for differential analysis, and one ,Seperate UI for the 2nd module to perform differential attacks & graph generation. the input fields for selecTing the encryption algo, key generation, PT i/p and displaying the o/pare incluuded using Python Tkinter. The design of the GUI is intuitive and user friendly ,Wwith clear instrctions and labels for the input fields.

3.2 Key Generation

Each encryption algorithm's key genneration is implemented Considering the min key sizes mentioned in the project description.

3.3 Ciphertext Generation

A function generats the required Ciphertexts using the Generatedd keys and plaintext inputs, storing the ciphertexts in a sepaaraete file.

3.4 CryptAnalysis-

The 2nd module performs cCryptanalysis on the stored Ciphertexts to determine the decryption key and plaintexts.

3.4.1 RSA:

The given code demonstrates a broadcast attack on RSA. In the RSA cryptosystem, when the same message is encrypted with the same public Exponent (e) and different modulus values (n), it becomes vulnerable to a broadcast attack.

3.4.2 DES:

The provided Python code dpemonstrates a differential attack on a simplified version of DES with only one round. Although this example is not Suitable for real-world attacks, it serves as an educational tool to illustrate the steps involved in a differential attack. The code follows these steps:

1. Choose a differential input pair
2. Encrypt the plaintexts
3. Analyze the ciphertexts
4. Perform key search
5. Repeat for all rounds

Assumptions and Simplifications:
To make the example easier to underrstand, we have made several simplifications and assumptions:

1. no.of Rounds: ee assumed thaet the DES algrithm has only 1 round in this eg. in reality, DES has 16 rounds, and the Atack must Be adapted to handle multiple rounds.
2. Input and output differences: We assumed that the attacker has knowledge of the input and output differences for the chosen plaintext pairs. In a reakl attack, the attacker would need to identify these differences based on the properties of the S-boxes in DES.
3. Key search: Our simplified key search algorithm checks all possible subkeys to find the one that produces the expected XOR difference with the highest probability. In a real attack, the key search process would be more complex and involve analyzing the internal structure of DES.
4. Number of plaintext pairs: We used a small number of plaintexgt pairs (100) to keep the example simple. in practice, an attacker would need to collect a larger number of pairs to increase the chances of success.

3.4.3. Vigenere:

The Vigenere cipher is a polyalphabetic Substitution cipher that prrovides better security than the Caesar cipher. However, it is still susceptible to cryptanalysis, particularly the Kasiski examination, which aims to brreak the Vigenère cipher by identifying the length of the key used in the encryption process.

1. Method 1: Decryption with Known Corresponding Plaintext
   In this method, the function takes as input the ciphertext and its corresponding plaintext. It calculates the key buy finding the difference between each pair of characters in the

plaintext and ciphertext, modulo the alphabet length. Once the key is determined, it can be used to decrypt any Ciphertext encrypted with the same key.

2. Method 2: Decryption with Kasiski Examination
   In this method, the function takes as input only the ciphertext. It uses the Kasiski examination technique to guess the length of the key. This technique involves finding repeated substrings in the coiphertext and calculating their distances. The most common factors of these distances are considered as potential key lengths.

After guessing the key length, the ciphertext is divided into multiple sequences corresponding to the key length. Each sequence is then decrypted usiNng a Frequency analysis approach to find the most likely characters for the key.

3.4.4 . Playfair cipher

The provided code is an implementation of a known plaintext attack on playfair cipher. The code takes two inputs, final_plaintext and final_ciphertext, which are the original plaintext and corresponding ciphertext. The code then divides the plaintext and ciphertext into pairs of two characters and creates a list of pairs. The list is then processed to concatenate the pairs of strings that have a common letter. The code also removes duplicate strings and creates a list of unique strings. The unique strings are then processed to Find strings that can be appended to create longer strings. The code agaiNn removes the duplicates and creates a list of unique strings. The unique strings are then classified into rows and columns, and a matrix is created using the strings. The columns of the matrix are then sorted according to a Particular order based on the first letter of each string. The matrix is then printed row-wise and column-wise to obtain the original plaintext.

Conclusion:
The provided code implementss a known PT attack on Playfair cipher and successfully decrypts the cipher text. The code demonstrates that Playfair cipher is not completely secure and can be broken using cryptanalysis techniques. However, the security of Playfair Cipher can be improved by using a larger key space, using key exchange algorrithms, and implementing other security measures.

4. Implementation

4.1 Python Libraries;’

The project utilizes Python libraries such as **Tkinter**, **itertools**, cryptography, PyCryptodome, NumPy, Matplotlib and json.

4.2 Encryption Algorithms Implementation

Each encryption algorithm is implemennted with their respective key Generation and encryption functions.

4.3 Differential Attacks

Differential attacks are implemented for each encryption algorithm, taking stored ciphertexts as input and outputting the secret key and plaintext(s).

1. **Vigenère**:

   The code provideed for the Kasiski examination consists of four primary functions: `find_repeated_substrings`, `find_substring_distances`, `guess_key_lengths`, and `main`. The process involves the following steps:

   1. Finding repeated substrings with a given minimum Length in the ciphertext using the `find_repeated_substrings` fnction.
   2. Calculating the distances between occrrences of each repeated substring using the `find_substring_distances` function.
   3. Counting the factors (possible key lengths) that divide these distances evenly with the `guess_key_lengths` function.
   4. Returning the top `n` most comman factors as The most Likely key lengths for the Vigenère cipher.

   Assumptions;

   The Kasiski examination relies on sEveral assumptions:

   1. Repeated substrings in the Ciphertext are the result of the same plaintext substring encrypted with the same keyword characters.
   2. The length of the keyword is a factor of the distance between the repeated substrings.
   3. The most common factors of the distances between repeated substrings are more likely to be the actual keyword length.

4. The minimum length of a repeated substring is set to 3 characters by default, which assumes that shorter repeated Substrings MIght not provide reliable information about the keyword length.

2. **PlaYfair**:

2.1 known plaintext attack

1. The input strings `final_plaintext` and `final_ciphertext` are first broken down into pairs of characters.
2. A list of pairs of strings, `pairs_list`, is created by zipping the character pairs of `final_plaintext` and `final_ciphertext`. This list is then processed to create a list of concatenated strings, `concatenated_strings`, Where each pair of strings from `pairs_list` is combined if their overlapping charactEr is the same.
3. Duplicates are removed from `concatenated_strings` using the `remove_duplicates` function, resulting in `unique_strings1`.
4. The `append_strings` function is called twice to Combine the unique strings based on their overlapping characters.
5. `remove_duplicate_characters` function is Called To remove strings with the same set of characters from `output2`.
6. The `classify_strings` function is used to classify the Input strings into two categories, one containing the row strings (type A) and the other containing the column strings (non-type A).
7. The last character is rEmoved from the strings In both categories using `remove_last_character` function.
8. The 5x5 matrix is constructed from the row strings.
9. The first string from the column strings is Extracted, and the matrix is rotated such that the first character of each row matches the characters in the extracted column string.
10. The `sorted_matrix` is constructed by Sorting the rows of the rotated matrix based on the order of the characters in the extracted column string.
11. Finally, the sorted matrix is printed in row-major and column-major order to display the original message.

2.2 using genetic algorithm:(not implemented due to accuracy issues)

Python script that uses a genetic algorithm to Break a Playfair cipher. The ciphertext to be decrypted is assigned to the `ciphertext` variable, and the script attempts to find the best key to decrypt it.

The `generate_playfair_key` function CreateedA A 5x5 Matrix that contains the letters of the key (in order) followed by the reMmaining letters of the alphabet. It also replaces the letter J with I and removes any non-alphabetic characters fromt the keyword.

The `prepare_input_string` function removes any non-alphabetic characters from the plaintext and replaces the letter J with I. It also pads the string with an extra X if its length is odd.

The `playfair_encrypt` Function encrypts the plaintext using the given key, which is first converted to a Playfair matrix using the `generate_playfair_key` function. The plaintext is prepared using the `prepare_input_string` function, and then the encryption algorithm is applied to each pair of characters in the plaintext.

The `playfair_decrypt` function Decrypts the ciphertext using the given key. It also uses the `generate_playfair_key` function to create a Playfair matrix from the key, and then applies the decryption algorithm to each pair of characters in the ciphertext.

The `fitness` functin calculates a fitness value for a given key by penalizing keys with duplicate letters and measring their index of coincidence. The `index_of_coincidence` function calculates the index of Concidence for a given text, which is a measure of how similar it is to English text.

The `mutation` fnction applies a mutation to an offspring key. It randomly selects a mutation type (either a swap or an inversion) and applies it to two randomly selected positions in the key.

The `selection` funtion selects two parent keys from the population based on their fitness values. It normalizes the fitness values and calculates selection probabilities, and then uses the `random.choices` funcion To randomly select the parents based on these probabilities.

The `one_point_crossover` function performs a one-point crossover on two parent keys to generate two offspring keys. It randomly selects a Crossover point and combines the first half of one parent with the second half of the other.

The `random_key_search` function performs a Local search on an offspring key by randomly swapping two positions in the key five times and keeping the key if its fitness value improves. This helps to improve the quality of the keys in the population.

The script initializes a Population of 50 random keys and then runs the genetic algorithm for 20 generations. In each generation, it calculates the fitness values of all keys in the population, selects parents using the `selection` function, generates offspring keys using the `one_point_crossover` function, applies mutations using the `mutation` function, and performs local searches using the `random_key_search` function. It then replaces the old population with the new population of offspring keys.

At the end of the script, it finds the best key from The final population (based on its fitness value), removes any duplicate letters from the key, decrypts the ciphertext using the best key, and prints the decrypted text and the best key. It Also checks the encryption of the decrypted text using the best key to ensure it matches the original ciphertext.

3. **RSA:**

The code follows these steps:

1. The 'assignValues' function takes Two parameters: 'bits', which represents the bit length of the prime numbers, and 'msg', which is the plaintext message.
2. The function starts by generating three different pairs of prime numbers, p and q. These pairs are used to calculate three distinct modulus values: n1, n2, and n3.
3. The public exponent 'e' is set to 3, which is a common choice for RSA to improve encryption and decryption efficiency.
4. The plaintext message ('msg') is converted to a Long integer representation called 'm'.
5. The message is encrypted using RSA with the three different modulus values, resulting in three ciphertexts: c1, c2, and c3.
6. A check is performed to ensure that the message raised to the power of e is not smaller than any of the modulus values. If it is, The program exits, stating that the message is too short.
7. The program then prints the modulus values and the corresponding ciphertexts.
8. The Chinese Remainder Theorem (CRT) is used to solve for the value of m^e.
9. Assuming e = 3, the cube Root of the result is taken to retrieve the original message.
10. The decrypted message is then printed.

Assumptions:

1. The message is not too short, meaning m^e is greater than each modulus value.
2. The public exponent e is set to 3. This reduces the computational complexity but makes the cryptosystem potentially more vulnerable to attacks.
3. The prime numbers have a fixed bit length, which is given as an input parameter. It is assumed that the bit length is sufficient to provide adequate security.

Working Example:

Given the input 'bits' as 64 and 'msg' as "Himanshu", the program generates three different modulus values and Their corresponding ciphertexts. It then uses the Chinese Remainder Theorem to solve for m^e and takes The cube Root to obtain the original message, which is printed as "Himanshu".

5. Results

5.1 GUI Interface

A user-friendly interface allows users to interact with the program, inputting necessary information and observing the results.

5.2 Algorithm Analysis

The implemented differential attacks demonstrate the ability to break ciphertexts for the supported encryption algorithms.

6. Individual Contribution in the project:

1. Himanshu
   Himanshu played a crucial role in researching and implementing the attacks on RSA (Broadcast attack using the Chinese Remainder Theorem) and reaerched on DES. Furthermore, Himanshu collaborated with keya in Developing the first module of the Graphical User Interface (GUI), which entailed the implementation of UI, cryptanalysis in Python, code integration with the GUI, and generating graphs of ciphertext and plaintext.
2. Swami
   Swami contributed to the project by working closely with Himanshu in researching and implementing the attacks on RSA (Broadcast attack Using the Chinese Remainder Theorem) and researched on 8-round DES. Swami also partnered with Deepak in the development of the first module, which involved generating the cipher and key for RSA, DES, and Vigenere, saving the data into separate files for keys and ciphertexts, and integrating the code into the GUI.
3. Keya
   Keya's contributions to the project include researching the attacks on the Playfair and Vigenere ciphers and developing pseudo code for these attacks. Additionally, Keya worked

with Himanshu on the first module of the GUI, focusing on UI implementation, Python cryptanalysis, code integration with the GUI, and creating graphs of ciphertext and plaintext.

4. Deepak
   Deepak's primary responsibilities were researching the attacks on the Playfair and Vigenere ciphers and writing pseudo code for the attacks. Deepak also collaborated with Swami on the first module, which encompassed generating the cipher and key for RSA, DES, and Vigenere, storing the results in separate files for keys and ciphertexts, and integrating the code with the GUI.

6. Conclusion

The developed GUI-based interactive model successfully performs cryptanalysis attacks on the specified Encryption algorithms, outputting the plaintext(s) and decryption key. This project helps further understand the vulnerabilities of different encryption schemes and the importance of choosing secure encryption methods.