

**CYBER SENTINEL SUITE:**

# INTRODUCTION :

**Cyber Sentinel Suite** is a comprehensive cybersecurity toolkit designed to protect systems and networks from a wide range of cyber threats. It incorporates advanced encryption techniques, including AES encryption and decryption, brute-force password cracking, and secure user authentication. The suite offers functionalities such as encryption and decryption of messages using AES, password brute-forcing, and file-based password authentication, along with user notifications for suspicious activities. The goal of Cyber Sentinel Suite is to provide robust security measures for data integrity and confidentiality, as well as tools for vulnerability testing and password security.

# BACKGROUND (RESEARCH & PROJECT SELECTION)

**Background Research**

**In today’s increasingly digital world, cyber threats have become one of the most significant risks to individuals, businesses, and governments. With the rise of sophisticated attacks, from ransomware to data breaches, the need for robust cybersecurity measures has never been more crucial. Cybersecurity tools play a vital role in detecting, preventing, and mitigating these threats, and they come in various forms, from firewalls and intrusion detection systems to encryption and password protection tools.**

**One of the key components of cybersecurity is data encryption. AES (Advanced Encryption Standard) has emerged as the global standard for securing sensitive data, widely adopted across industries due to its strength, efficiency, and versatility. However, despite its robust security measures, cryptographic systems can be vulnerable to attacks, especially if passwords or encryption keys are weak or poorly managed. As a result, there is a need for reliable tools to enhance security practices, particularly those that focus on securing passwords and encrypted data.**

* **PROJECT IDEA AND REASON FOR SELECTION**

**The Cyber Sentinel Suite was selected as a project idea due to the growing need for effective, user-friendly cybersecurity tools in both personal and organizational contexts. This project aims to create a versatile cybersecurity toolkit that incorporates essential features like AES encryption and decryption, brute-force password cracking, and secure user authentication. The rationale behind this selection stems from the following factors:**

1. **Growing Demand for Cybersecurity Tools: As cyber-attacks continue to rise globally, there is a significant demand for more accessible and effective cybersecurity solutions, particularly for individuals and small businesses that may not have access to enterprise-level security infrastructure.**
2. **Focus on Encryption and Authentication: AES encryption is widely considered one of the most secure methods for protecting sensitive data. However, its effectiveness is often compromised by weak password practices. By including tools for both encryption and password cracking, this project provides a holistic approach to data security.**
3. **Relevance to Modern Cybersecurity Challenges: Password-based authentication remains a major vector for cyber-attacks. Tools like brute-force password cracking simulations are important for users and administrators to test their systems’ vulnerability and strengthen password security. Furthermore, by focusing on encryption and user authentication, this project directly addresses some of the most pressing concerns in modern cybersecurity.**
4. **Idea from Kali Linux: The idea for your tools, including encryption, decryption, and brute-force attacks, is inspired by the tools available in Kali Linux. Kali Linux is a widely recognized platform for cybersecurity and ethical hacking, offering a range of penetration testing and security analysis tools. By incorporating these concepts into your project, you're leveraging industry-standard techniques for securing and testing systems, which ensures your work is aligned with current best practices in cybersecurity.**

# PROJECT SPECIFICATION

The **Cyber Sentinel Suite** is a terminal-based application developed in C, incorporating the following specifications:

* **AES Encryption/Decryption: The AES implementation seems correct, but the Mix Columns, Shift Rows, and AddRoundKey functions are not being used consistently in the encryption/decryption process. Ensure that you're using the correct number of rounds for AES, and the Mix Columns operation should be applied only in the intermediate rounds, not the last one.**
* **Brute Force Password Cracking (BFP): The brute force loop uses incr\_pass to increment the password, which will generate combinations starting from "a". Ensure that it works properly when trying larger sets of characters like uppercase letters, digits, or special characters if needed.**
* **Password Authentication: In this you might want to add logic to limit the number of authentication attempts, and potentially lock the user out after a certain number of failed attempts.**

# PROBLEM ANALYSIS

* **Fragmented Security Tools**

Users often rely on multiple, specialized security tools for tasks like password management and file encryption. This fragmentation leads to inefficiencies, inconsistent security practices, and a steep learning curve, especially for those with limited technical expertise.

* **Usability Barriers**

Many existing security applications prioritize functionality over user-friendliness, resulting in interfaces that are not intuitive. This complexity discourages regular use and hinders effective implementation of essential security measures.

* **Educational Gaps**

There is a lack of integrated solutions that not only provide security functionalities but also educate users on best practices. Users may not fully understand the importance of strong passwords or the mechanics of encryption, leading to vulnerabilities.

* **Security Vulnerabilities**

Inadequate password management and insufficient file encryption expose users to significant security risks, including unauthorized access and data breaches. Without comprehensive tools, users are more likely to adopt weak security practices.

* **Project-Specific Challenges**

Developing an integrated Cyber Sentinel Toolkit involves balancing comprehensive functionality with simplicity, ensuring robust security measures, and optimizing performance. Additionally, creating a user-friendly interface within a command-line environment presents its own set of design and implementation hurdles.

# SOLUTION DESIGN (PROJECT DETAIL, FUNCTIONALITY, AND FEATURES)

The solution is architected as a modular command-line application, leveraging C programming for performance and portability. The primary components and their functionalities are as follows:

**Authentication and Access Control**:

* **Authentication (authenticate user):** Users must be authenticated before accessing sensitive data or tools. Implement authentication mechanisms (passwords, two-factor authentication, etc.).
* **Role-based access**: Define roles (admin, user) to restrict access to specific features of the program.

**Incident Logging**:

* **Incident Logging (log\_incident\_to\_file)**: Create a file-based logging system to record digital forensics incidents, errors, and tool usage.
* **Log format**: Use a structured log format (plain text) with relevant fields like timestamps, user actions, and error messages.

**Progress Visualization**:

* **Progress Bar**: Use ASCII or GUI-based progress bars to indicate ongoing operations (e.g., encryption, decryption, tool execution).

**Sound and Visual Feedback**:

* **Sound (play\_sound)**: Provide audio cues to alert users about significant events like completion of tasks or errors.
* **Animation (show\_welcome\_animation)**: Implement a simple animation, likely text-based or graphical, to enhance user experience during program startup.

1. **Brute Force Attack:**
   * **Functionality:** Simulates brute-force attacks by incrementally guessing passwords based on a defined character set.
   * **Features:** Tracks the number of attempts, calculates the time taken, and provides progress updates.
2. **AES Encryption/Decryption:**
3. **User Feedback Collection:**

**Functionality:** Enables users to provide feedback, which is timestamped and stored in a text file for future reference.

* **Startup**:
* Show welcome animation and load the main menu.
* **Authentication:**
* Prompt the user for credentials. If successful, proceed to the main menu; otherwise, terminate or re-prompt.
* **Main Menu:**
* The user can choose between encryption, decryption, brute force attack, help section, or exiting by giving feedback.
* **Tool Execution:**
* Based on the user's choice, invoke appropriate functions such as encryption, decryption, brute force attack.
* Show progress bar during long operations and provide sound/animation feedback.
* **Exit/Help:**
* If the user selects "Exit", clean up resources and close the program.
* If the user selects "Help", display the help menu with instructions

# IMPLEMENTATION & TESTING

**Implementation**

The application was implemented using the C programming language, chosen for its efficiency and control over system resources. Key implementation aspects include:

* **User Interface Enhancements:** Utilizes ANSI color codes to differentiate messages, enhancing readability and user engagement.
* **Input Validation:** Ensures robust handling of user inputs, mitigating potential errors and enhancing security.
* **File Handling:** Implements secure file operations for encryption and decryption processes.

**Testing**

Comprehensive testing was conducted to ensure the reliability and security of the suite:

* **Unit Testing:** Each tool was individually tested for functionality, ensuring accurate performance of encryption, decryption, password generation, and strength checking.
* **Integration Testing:** Verified seamless interaction between the login system and the various tools.
* **Security Testing:** Assessed the robustness of the login mechanism, ensuring lockout protocols functioned correctly after multiple failed attempts.
* **User Acceptance Testing:** Collected feedback from test users to refine the user interface and improve overall usability.

**Project Breakdown Structure (Workload Distribution with Timeline)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Tasks** | **Duration** | **Responsible** |
| **Planning** | Project Planning | 2 Week | Muhammad Sami |
| **Development** | (Login, Tools, UI) | 2 Weeks | Muhammad Sami + Talal Ali  +Mawavia Safi. |
| **Testing** | Final Review | 3 days | Muhammad Sami + Talal Ali  +Mawavia Safi. |
| **Documentation** | Writing Project Report | 1 days | Muhammad Sami + Talal Ali  +Mawavia Safi |
| **Design** | UI Design | 4 days | Muhammad Sami + Talal Ali  +Mawavia Safi |

**Task Assignments**

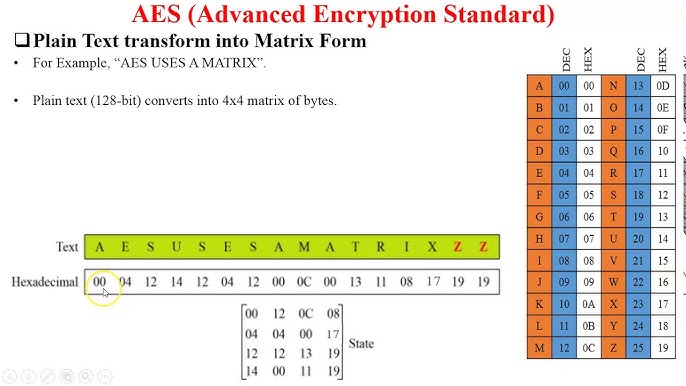
**(Project Distribution with Timeline)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Duration** |  | **Responsible** |
| **Password authentication** | 4 days |  | Mawavia Safi |
| **AES Encryption & Decryption** | 5 days |  | Talal Ali |
| **Brute force attack** | 3 days |  | Muhammad Sami |
| **Help section / feedback** | 4 days |  | Muhammad Sami |
| **GUI** | 3 days |  | Talal Ali + Mawavia Safi |

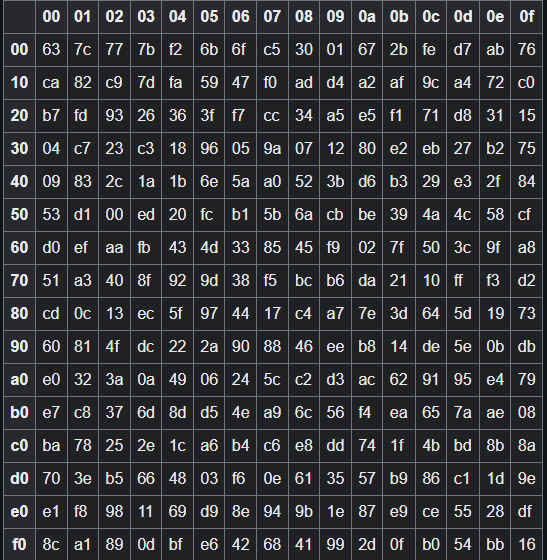
# BRIEFLY EXPLANATION OF ENCRYPTION DECRYPTION:

**Overview:**

* The project follows the principle of AES (advanced encryption system). This system turns a data of 16 bytes (in this case a string of 16 characters) into a 4x4 matrix.



* Like all encryption system keys this also utilizes round keys; a predefined key gets edited/changed with each round of the encryption.
* An encryption can be 128, 192 and 256 bits. 128 bits takes 10 rounds, 192 takes 12 and 256 takes 14.
* This matrix will have its value substituted by a predefined sub box



* The rows are of the matrix (now will be referred to as “state”) will shift according to the index of the row i.e. if index of the row is 0 then it will shift zero times to the left; however, if the index is 1 then the values will shift by one to the left and the value that will move out of the matrix will return to the last place of the row.
* The column will then be mixed by multiplying with a predefined matrix, whose inverse does exist

A number of numbers on a white background

Description automatically generated

* The column mixing is done by using Galois field, the basic concept of the field is in this case is to not leave the range of 1 byte (0-255) and all operations like addition are done with XOR. To oversimplify it, it works like analogue clocks, when the hand reaches 12 you then go to 1 or if the hour hand is at 11 and you have to add 2 hours you would reach 1.

[A diagram of a computer process

Description automatically generated](#_top)

<- This diagram encapsulates the operations of the code.

* The decryption is a mirror version of the above-mentioned process.
* First the round keys will add.
* The values for the reverse column mixing are the following:

A number of letters and numbers

Description automatically generated

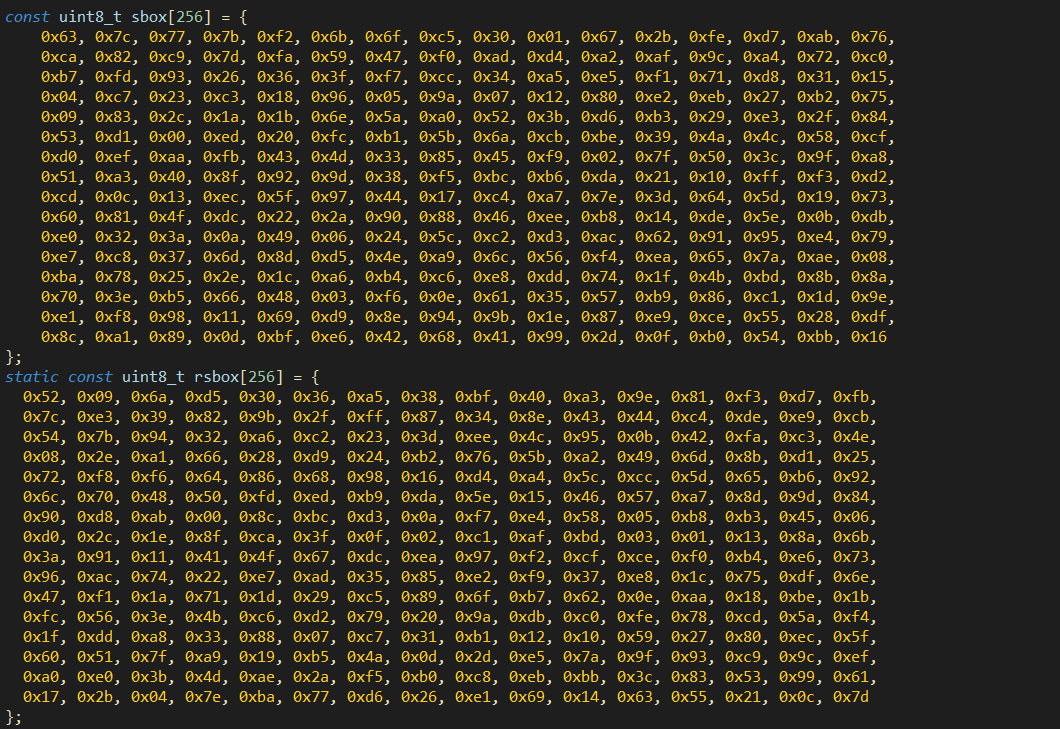
* Rows will be shifted exactly as mentioned before.



* This is the table that represents the reverse of the sub box.
* This process will be repeated as many times as encryption.

**The Code:**

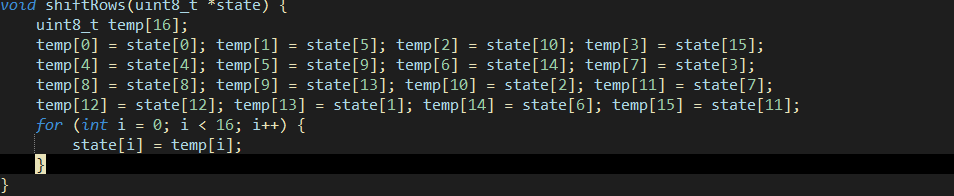
For this code we will be using uint8\_t from stdint library, to ensure that our data will be dealt in 8 bits.

****

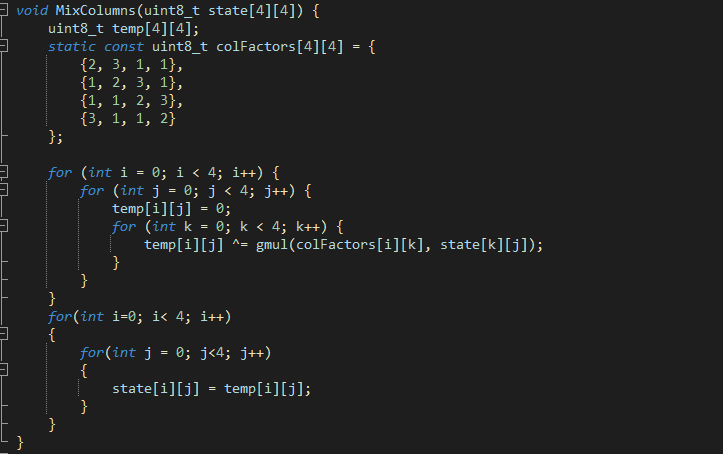
* These arrays represent the contents of the substitution boxes
* A black screen with white text

  Description automatically generatedThis function will take the current state and of the data and substitute it appropriately.

i.e. if the “A” must be substituted it will be substitute by sbox[41] since “A” is 0x41 in hexa



* This is a simple code to shift rows appropriately.

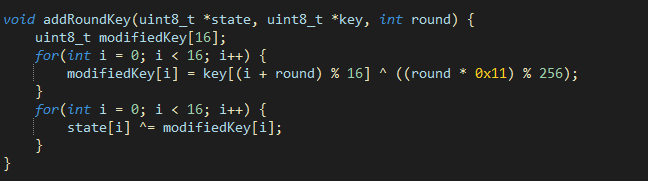


* This function handles the column mixing by incorporating the gmul function which utilizes the above-mentioned Galois Field.
* Here is the gmul function

A computer screen with text and numbers

Description automatically generated

The formula is very simple. A local variable is first initialized by zero, and a loop will iterate till the second number of the multiplication is a non-zero. In the loop, if the least significant bit is 1 then a simple XOR operation will be commenced on the variable with the first value. The second if statement deals with the overflow of the most significant bit, if it occurs then the bits will be shifted to the left and a XOR operation will be done with 0x1b. We chose 0x1b because it is an irreducible number of the Galois Field. If there is no overflow then a simple bit shifting is done. Throughout the loop, one thing remains constant: shifting b towards the right side. This whole operation outputs the result of p.

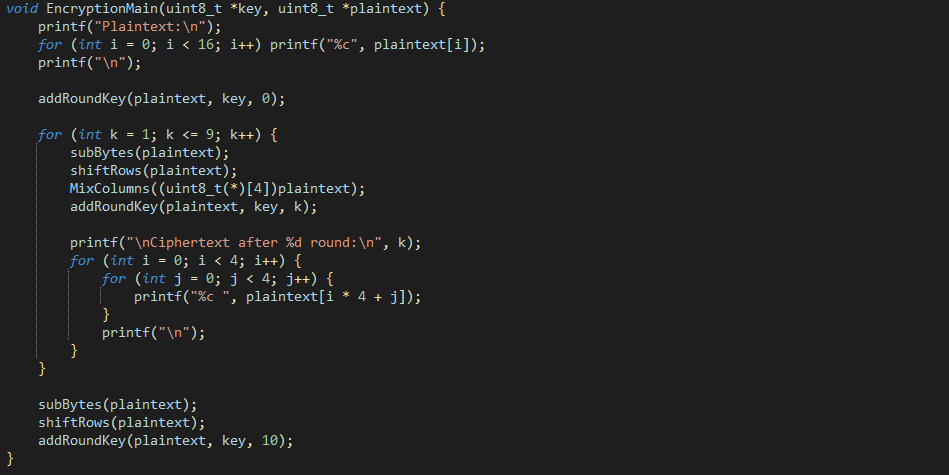


* This function handles the addition of the round keys. A simple formula was designed to handle this function and the XOR operation done in the first operation is there to check for any overflow values.

A screen shot of a computer code

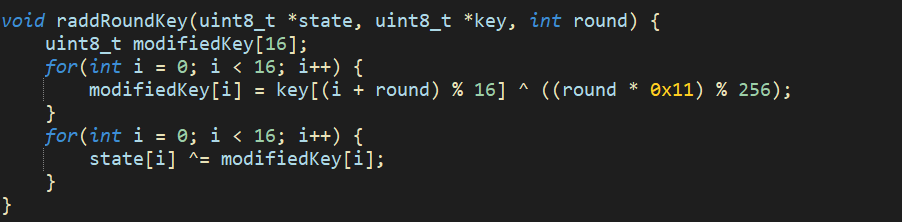
Description automatically generated

* All of the functions above are concised in this one function to be called in a loop. But this was later scrapped because initially only the round key is first added so every function has to be called independently and hence removed from the main functions.

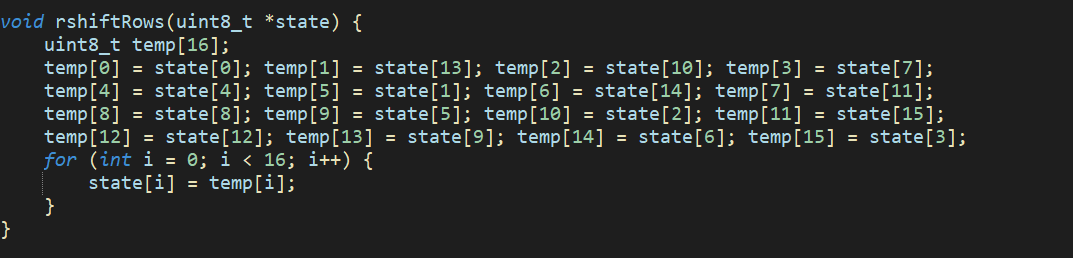


* Encryption main function follows Figure 1.

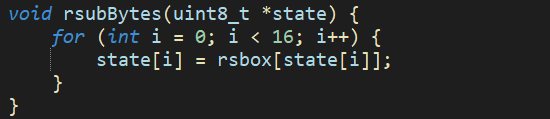
The decryption process is no different but rather mirrored version of it. Understanding encryption by itself was enough to grasp the concept of decryption and it’s approach



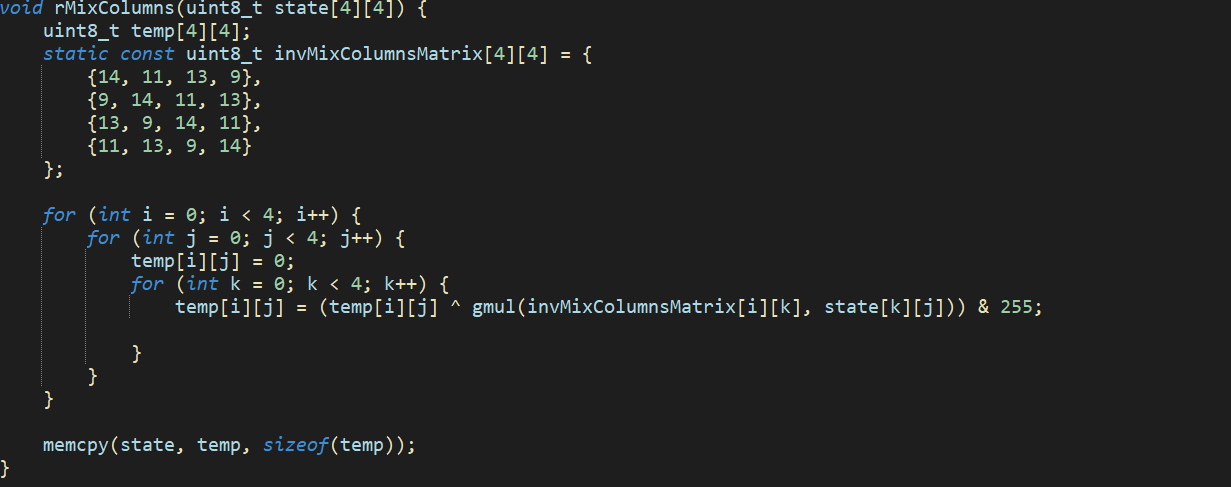
* Reverse of the addition of the key.



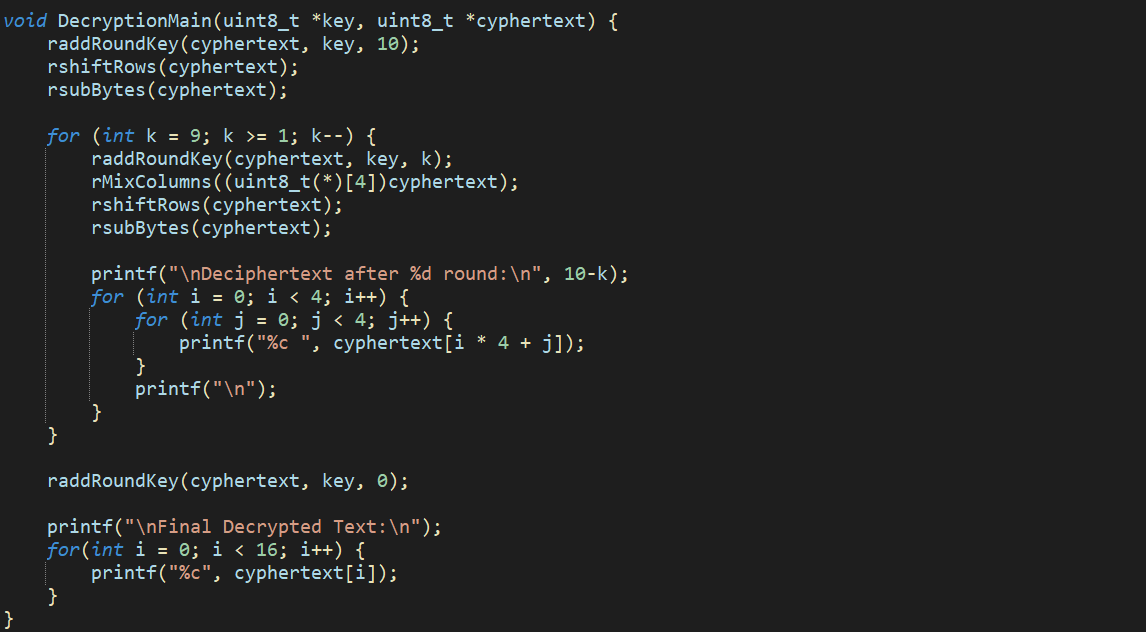
* Then comes the reversal of the row shifting



* These values are then substituted back with the help of the reverse sub box

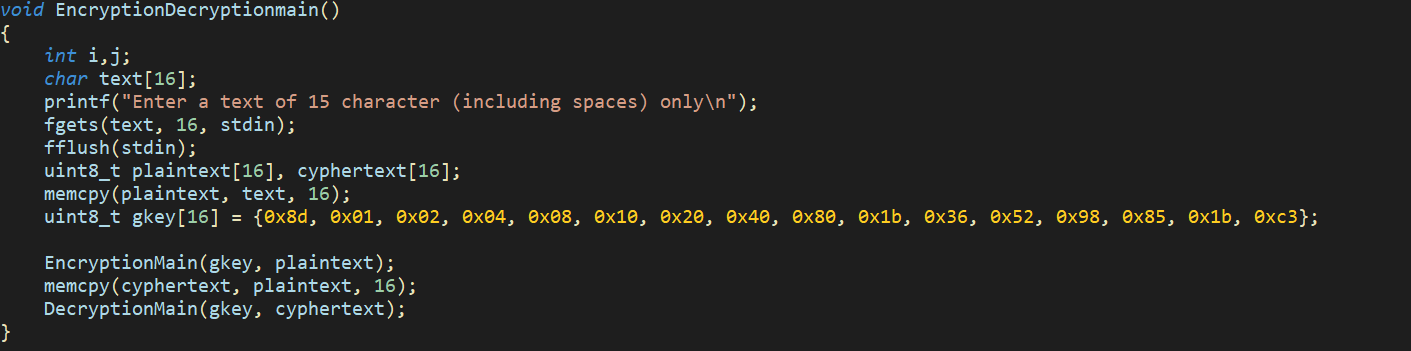


* After that comes reverse mixing of the column matrix



* Finally all of it is then called in the Decryption main function

The following function is called in the main, this handles everything up until now. It takes input from the user and copies the contents of the memory in a variable which is then passed through the functions.



Issues Faced:

* At first a function was used to generate random keys, but due to time constraints and the complexity of the project it was then removed.
* Another function was first made to make a column shuffling generator but that was also scraped since not all values had inverses.
* Initially, before the creation of the galois field, the module was not correctly decrypting the code.

**Strengths:**

* If the key is safe and the round key aswell, it is very difficult to crack.
* The galois field is a powerful tool, since it is efficient and provides an acceptable encryption.
* Since everything is in a sub box (also known as a lookup table), everything works smoothly and efficiently.
* Since the key changes each iteration it is almost impossible to get to the original key without the algorithm to add round keys.

**Limitation:**

* A text must be up to 15 characters (1 character for null character).
* The algorithm is very basic since this is a demonstration, the key is much shorter and only does 10 rounds.

Resources Used:  
<https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard>

<https://github.com/kokke/tiny-AES-c/tree/master>

<https://en.wikipedia.org/wiki/Rijndael_S-box>

<https://crypto.stackexchange.com/questions/2418/how-to-use-rcon-in-key-expansion-of-128-bit-advanced-encryption-standard>

<https://www.youtube.com/watch?v=O4xNJsjtN6E&ab_channel=Computerphile>

# BRIEFLY EXPLANATION OF BRUTE FORCE ATTACK.

**Report: Brute Force Password Cracker**

**1. Introduction**

The purpose of this project is to implement a brute-force password cracking tool that systematically tries all possible combinations of lowercase letters ('a' to 'z') for a password, up to a specified maximum length. The program will measure the time taken to crack the password and show the progress of the brute force attack. This project serves as a demonstration of how a brute force attack works and emphasizes the importance of using strong, complex passwords.

**2. Problem Statement**

Brute force is a method of cryptanalysis where an attacker attempts all possible combinations of characters until the correct one is found. The task was to implement a brute force algorithm to crack a password that is comprised only of lowercase English letters ('a' to 'z'). The user specifies the target password and the maximum length of the password to brute-force.

**3. Design & Functionality**

**Key Features:**

1. **Brute Force Approach**:
   * The algorithm generates all possible combinations of characters ('a' to 'z') for password lengths ranging from 1 to the specified maximum length.
   * It attempts each generated combination and checks if it matches the target password.
2. **Progress Feedback**:
   * A rotating progress indicator (|, /, -, \) is displayed to show the current status of the brute-force attempt, helping the user visualize the progress in real time.
3. **Time Measurement**:
   * The program measures the time taken to crack the password and displays this information after successfully finding the password.
4. **Customizable Length**:
   * The user can specify the maximum length of the password (up to 50 characters), which determines how many different combinations will be generated.

**Workflow:**

1. **User Input**:
   * The user is prompted to enter the target password and the maximum password length they want to test.
2. **Brute Force Execution**:
   * The program generates password combinations of increasing length, starting from 1 character and going up to the user-specified maximum length.
3. **Match Check**:
   * For each generated password attempt, the program checks if it matches the target password.
   * If a match is found, the program stops and prints the cracked password along with the time taken.
4. **Progress Display**:
   * A progress bar with rotating characters shows the status of the brute-force attack as the program progresses through all combinations.
5. **End of Process**:
   * If the target password is found, it is displayed along with the time taken.
   * If the target password is not found after all combinations have been tested, the program informs the user that the password was not found.

**4. Code Explanation**

**incr\_pass Function:**

This function handles the incrementation of password attempts. It starts with the last character of the current password attempt, and if the character is less than 'z', it increments it. If the character is 'z', it resets it to 'a' and moves to the previous character, continuing this process until a valid password is generated.

* **Inputs**:
  + atmpt: The current password attempt.
  + len: The length of the current password.
* **Outputs**:
  + Returns 1 if the password has been incremented successfully; 0 if all possible combinations for the current length are exhausted.

**BFP Function:**

The core brute-force logic is implemented here. The function tries all combinations of passwords starting from length 1 up to mlen (maximum length). For each length, it generates all possible combinations of lowercase letters, and checks if the current attempt matches the target password.

* **Inputs**:
  + target: The target password to be cracked.
  + mlen: The maximum length for password attempts.
* **Outputs**:
  + Returns 1 if the target password is cracked, and prints the time taken.
  + Returns 0 if no match is found after trying all combinations.

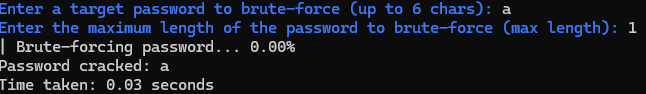
**main Function:**

This function serves as the entry point for the program. It prompts the user to input the target password and the maximum password length and then calls the BFP function to initiate the brute-force cracking process.

* **Inputs**:
  + target\_pass: The target password to be cracked (up to 50 characters).
  + mlen: The maximum length of the password to brute-force.
* **Outputs**:
  + Calls the brute-force cracking process and displays results.

**5. Testing**

**Test Case 1:**

****

**Test Case 2:**

A screen shot of a computer screen

Description automatically generated

**Test Case 3:**

**A screen shot of a computer screen

Description automatically generated**

**Edge Case:**

**A screen shot of a computer screen

Description automatically generated**

**6. Performance Considerations**

* **Time Complexity**:
  + The time complexity is exponential with respect to the password length. The number of attempts increases exponentially as the length of the password grows. For a password length of n, there are 26n26^n possible combinations.
  + For example, a 5-character password will have 265=11,881,37626^5 = 11,881,376 possible combinations.
* **Memory Efficiency**:
  + The program uses only a small amount of memory for storing each password attempt, as it works with a temporary atmpt array for each combination.
* **Speed**:
  + The speed of the brute force attack depends on the length of the password and the computational power of the machine. For small passwords, the attack is fast, but for longer passwords, the attack can take significant time.

**7. Conclusion**

This program demonstrates a simple brute-force password cracker that uses the brute-force approach to generate all possible combinations of lowercase letters. While this method is highly inefficient for longer passwords, it highlights the importance of strong, complex passwords. The tool also provides real-time progress feedback and measures the time taken to crack the password.

# RESULTS (OUTPUT SCREENSHOTS)

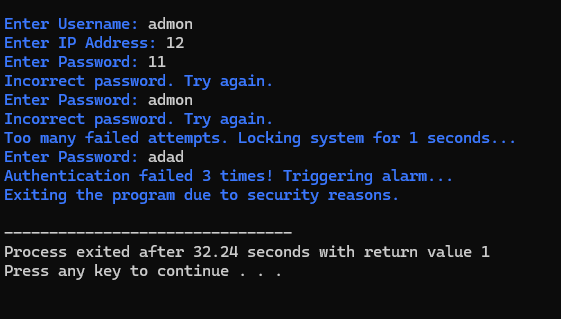
**Welcome Screen**

A screenshot of a computer program

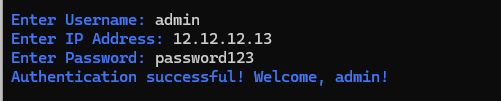
Description automatically generated

**Login Screen**

Displayed a coloured header with welcome messages and prompts for password entry.



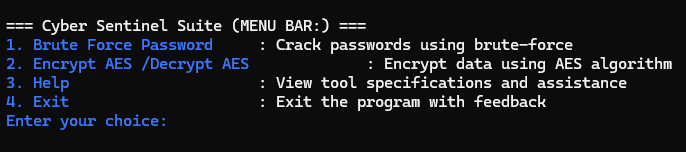
Showed lockout messages upon exceeding maximum login attempts.





**Tools Menu**

Presented a color-coded menu listing all available tools with descriptions which Allows users to select tools by entering corresponding numbers.



**Brute Force Attack Simulator**

Displayed progress percentages and success messages upon cracking the target password.

**A screen shot of a computer screen

Description automatically generated**

**AES Encryption/Decryption:**

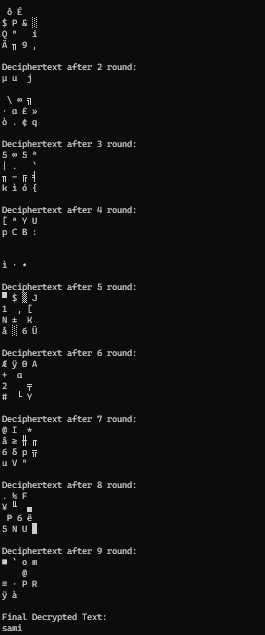
Showed prompts for file names and keys And Confirmed successful encryption/decryption with coloured success messages.

Encryption:

A screenshot of a computer

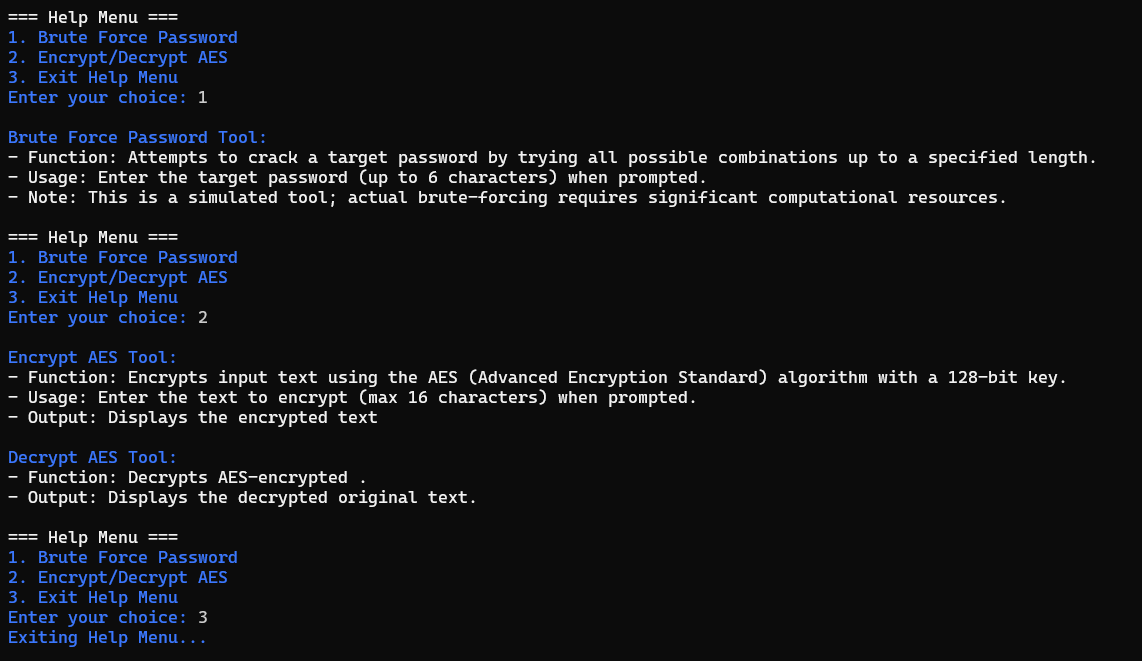
Description automatically generated

Decryption:



**Help Section**

Presented clear instructions and guidance on using each tool within the suite.



**Exit Screen (Feedback):**

A screenshot of a computer program

Description automatically generated





# CONCLUSION (SUMMARY & DISCUSSION)

**Summary:**

The two codes provided demonstrate essential concepts in cybersecurity: **password cracking** via brute force and **data encryption/decryption** using the AES algorithm, coupled with forensic tools like incident logging and user authentication. The first code implements a brute-force password cracking tool that systematically tries all possible combinations of lowercase letters to guess a target password. The second code outlines functions for encryption, decryption, user authentication, logging, and system interaction, typically seen in digital forensic tools.

**Brute-Force Password Cracker:**

The first code demonstrates a **brute-force password cracker**, which attempts every possible combination of lowercase letters ('a' to 'z') for passwords up to a specified maximum length. The program offers real-time feedback with a rotating progress bar and tracks the time taken to crack the password.

**Key Features of the Brute-Force Cracker**:

1. **Password Cracking**: The program generates combinations of lowercase letters for passwords up to the user-defined length.
2. **Progress Feedback**: Displays a rotating progress indicator (|/-\\) during the brute-force process.
3. **Time Measurement**: Records and displays the time taken to crack the password once found.
4. **User Input**: The user specifies the target password and maximum length for the brute-force process.

**Encryption/Decryption and Forensic Tool Functions:**

The second set of functions belongs to a **digital forensic toolset** that includes encryption and decryption functionality (using AES), user authentication, and system interaction. These functions are designed for securing and auditing data, and include:

1. **AES Encryption and Decryption**: Implements symmetric encryption (AES) for securely encrypting and decrypting data.
2. **User Authentication**: Verifies the identity of users to control access to sensitive operations.
3. **System Interaction**: Includes menu options, progress bar animations, sound notifications, and user interaction functions for a better user experience.

# ACKNOWLEDGEMENTS

We extend our gratitude to our instructor, Sir Ghulam Qadir , for providing invaluable guidance and support throughout the development of this project. Additionally, we acknowledge the contributions of our group members, Muhammad Sami ,Talal Ali and Mawavia Safi for their dedication and collaborative efforts.

# CONTACT INFORMATION

**Muhammad Sami**

Email: k240515@nu.edu.pk

**Talal Ali**

Email: k240863@nu.edu.pk

**Mawavia Safi**

Email: k240507@nu.edu.pk