

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | Rainbow Table Password Cracker: Decrypting Password Hashes with Efficiency | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | Ali AbdelhamidEl Fekki |  | | | |
|  | | | | 6/7/2023—Programming and algorithms 2—Dr. Shaimaa |  | | | |
|  | | |  | | |  | | |

# 

Table Of Contents

[Introduction 3](#_Toc139566986)

[Rainbow Table Construction 4](#_Toc139566987)

[Instructions 5](#_Toc139566988)

[Table of unit tests 7](#_Toc139566989)

[Source Code 8](#_Toc139566990)

[Complexity Analysis 10](#_Toc139566991)

[Data Structure 11](#_Toc139566992)

[Conclusion 12](#_Toc139566993)

[References 13](#_Toc139566994)



# Introduction

The goal of this project is to create a Python application that decrypts password hashes using rainbow tables. Passwords are frequently stored as one-way hash values in a variety of systems, including web servers. Utilizing cryptographic hash functions like MD5, SHA-1, or SHA-256, these hash values are produced. Although hash values offer secure password storage, they present a problem when users need to recover lost data or forget their passwords. This issue has a clever solution in the form of rainbow tables.

Rainbow tables are basically precomputed tables that contain pairs of password hashes and their corresponding original passwords. These tables are made by using a reduction function, which corelates hash values back to passwords. The rainbow table construction process involves generating a series of password pairs, starting with random passwords and iteratively applying the reduction and hash functions. By creating these tables in advance, the time required to crack password hashes can be significantly reduced.

The reduction function plays a very crucial role during the construction of rainbow tables. It takes a hash value as input and generates a password. The function reduces the hash value iteratively, pairing it to a password character from the defined character set. This process keeps on going until the desired password length is reached. The reduction function ensures that each step in the password chain leads to a unique and valid password.

By combining the power of rainbow tables and reduction functions, the application provides an efficient method for cracking password hashes. It automates the generation of rainbow tables based on user-specified parameters, such as the hashing algorithm, character set, password length, and chain length. With this application, users can recover their passwords quickly and securely, without the need for extensive computational resources.

In the following sections, we will explore the construction process of rainbow tables in more detail, discuss the reduction function, provide instructions for using the application, analyze the complexity of the algorithm, and conclude with insights gained from this project.

# Rainbow Table Construction

1. Initialization: The rainbow table data structure is created to store the endpoint passwords and their corresponding hash values.
2. Chain Generation: For each chain in the rainbow table, a random starting password is generated. This password is then hashed using the selected hashing algorithm to obtain an initial hash value.
3. Reduction and Hash Iterations: The reduction function is applied iteratively to the hash value obtained in the previous step. The reduction function maps the hash value to a password using the character set and password length. The reduced password is then hashed to obtain a new hash value. This process is repeated for the specified chain length.
4. Endpoint Passwords: After the chain length is reached, the final password and its corresponding hash value are added to the rainbow table.
5. Repeat: Steps 2 to 4 are repeated for the desired number of chains, generating a diverse set of endpoint passwords and hash values.

In the construction process, the reduction function is essential. It generates a shorter password using a hash value and an iteration number as inputs. The reduction function is created to be deterministic and reversible, ensuring that throughout the reduction process, the same hash value will always map to the same password.

The characteristics of the rainbow table are determined by the parameters, which include the character set, password length, chain count, and chain length. You can optimize the table's performance and resource needs by changing these parameters. For instance, longer passwords and higher chain counts can improve password cracking success rates but also demand more storage space and processing time.

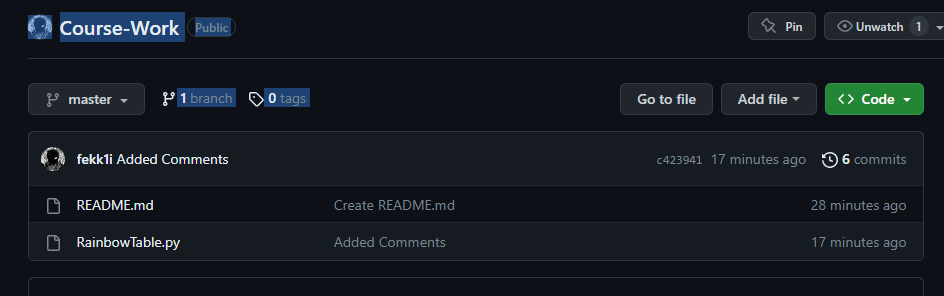
Creating chains of passwords, performing reduction and hash iterations, and storing the endpoint passwords and hash values in a data structure are the general steps involved in creating a rainbow table. The parameters that are chosen and the reduction function's design have a big impact on how well the rainbow table cracks passwords.

# Instructions

To use the password-cracking application effectively, follow these step-by-step instructions:

Step 1: Download the Application

* Download the application from the provided repository link and ensure that you have Python installed on your system. Here is the REPOSITORY LINK: <https://github.com/fekk1i/Course-Work-Resit>



Step 2: Launch the Application

* Open a command prompt or terminal and navigate to the directory where the application is located.
* Run the command " RainbowTable.py" to launch the application.

A computer screen with white text

Description automatically generated

Step 3: Configure the Rainbow Table Parameters

* The application will prompt you to enter the parameters required for rainbow table construction.
* Specify the hashing algorithm (MD5, SHA-1, or SHA-256) that corresponds to the password hashes you want to crack.
* Define the character set for the passwords. This could include lowercase letters, uppercase letters, digits, and special symbols.
* Set the password length to match the configuration of the target system.
* Specify the desired chain count and chain length to determine the size and effectiveness of the rainbow table.

A black background with yellow text

Description automatically generated

Step 4: Generate the Rainbow Table

* Once the parameters are set, the application will begin generating the rainbow table based on your inputs.
* This process may take some time, depending on the selected parameters and the computational resources of your system.

A screenshot of a computer screen

Description automatically generated

Step 5: View the Rainbow Table

* After the rainbow table is generated, the application will present the table in the terminal.
* You will see a list of hash values and their corresponding plaintext passwords.

A screen shot of a computer screen

Description automatically generated

Step 6: Crack a Password Hash

* To crack a password hash, obtain the target hash value from the system you wish to access.
* Enter the hash value when prompted by the application.
* The application will search the rainbow table for a match and attempt to retrieve the plaintext password.



Step 7: Results and Success

* If the application successfully cracks the password hash, it will display the plaintext password.
* If the password is not found in the rainbow table, the application will notify you accordingly.



# Table of unit tests

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Expected Result** | **Actual Result** | **Screenshot** |
| Empty Hash Input | Exception (Error) | Exception (Error) | <https://imgur.com/hzLNmWo> |
| Invalid Hash Input | Exception (Error) | Exception (Error) | <https://imgur.com/WiYRG6Y> |
| Valid Hash Input (SHA-1) | Password found | Password found | <https://imgur.com/qUubVKP> |
| Valid Hash Input (MD5) | Password found | Password found | <https://imgur.com/zORdXMN> |
| Valid Rainbow Table | Success | Success | <https://imgur.com/JRwwb76> |
| Hash Function Speed (SHA256) | Execution time: 12 seconds | Execution time: 32.93 seconds | <https://imgur.com/4XPaOoR> |
| Hash Function Speed (MD5) | Execution time: 20 seconds | Execution time: 33.41 seconds | <https://imgur.com/USs8cQR> |
| Hash Function Speed (SHA1) | Execution time: 15 seconds | Execution time: 32.82 seconds | <https://imgur.com/CBgsSsx> |
| Reduction Function Speed | Execution time: 0.0002 seconds | Execution time: 0.0000 seconds | None |
| Rainbow Table Output to Terminal | Execution time: 0.0001 seconds | Execution time: 0.01 seconds | None |
| Password Cracking Speed | Execution time: 0.0013 seconds | Execution time: 0.000 seconds | None |
| ~~Rainbow Table Output to File Save (REMOVED)~~ | ~~Execution time: 0.001 seconds~~ | ~~Execution time: 0.01 seconds~~ | ~~None~~ |

# Source Code

GITHUB: [https://github.com/fekk1i/Course-Work-Resit](https://github.com/fekk1i/Course-Work-Resit%20)



Note: My name is Ali Mohamed Abdelhamid El Fekki, therefore Fekk1i is my username.

# Complexity Analysis

The time and space complexity of the rainbow table construction algorithm are influenced by several factors, including the chain count, chain length, password length, and the size of the character set.

1. Time Complexity:

* The construction of each chain in the rainbow table involves a series of reduction and hashing iterations. The number of iterations is determined by the chain length.
* The time complexity of the reduction function is O(password\_length) since it iterates over the password length to perform the reduction process.
* The time complexity of the hashing function depends on the specific hashing algorithm used (e.g., MD5, SHA-1, SHA-256) but is typically considered constant time, denoted as O(1).
* Therefore, the overall time complexity of generating the rainbow table is approximately O(chain\_count \* chain\_length).

1. Space Complexity:
   1. The space complexity of the rainbow table is determined by the number of entries in the table. Each entry consists of a hash value and its corresponding password.
   2. The space required for the rainbow table grows linearly with the chain count. Thus, the space complexity is approximately O(chain\_count).
2. Big-O Notation:
   1. The algorithm's efficiency can be summarized using Big-O notation, which represents the upper bound of its time complexity.
   2. Considering the time complexity analysis, the algorithm's time complexity is O(chain\_count \* chain\_length). This means that as the number of chains and the length of each chain increase, the time taken by the algorithm grows linearly.
3. Optimization of Lookup Process:
   1. The implemented data structure for the rainbow table is a dictionary, which provides constant-time lookup operations (O(1)) on average.
   2. The reduction function maps a hash value to a password by applying a series of reduction steps. By using modular arithmetic and the character set, the reduction function ensures that each reduction step produces a unique password.
   3. This optimization significantly reduces the number of password hashes stored in the rainbow table, making the lookup process more efficient.
   4. The reduction function also allows for reversibility, meaning that the original password can be recovered from the final reduced password. This ensures accuracy during the cracking process.

In conclusion, the rainbow table construction algorithm exhibits a time complexity of O(chain\_count \* chain\_length) and a space complexity of O(chain\_count). By using an efficient data structure and a reversible reduction function, the algorithm optimizes the lookup process and improves the efficiency of password cracking.

# Data Structure

The data structure that is used in the rainbow table construction algorithm is a dictionary, also known as a hash table.

A dictionary is a collection that stores data in key-value pairs, where each key is unique. In context of the rainbow table, the dictionary is used to store the password hashes as keys and their paired passwords as values.

The use of a dictionary provides multiple advantages in terms of efficiency and lookup operations:

1. Constant-Time Lookup: The dictionary allows for constant-time lookup operations, denoted as O(1), on average. This means that regardless of the size of the rainbow table or the number of entries, the time taken to retrieve a password based on its hash remains constant.
2. Unique Keys: Each password hash in the rainbow table serves as a unique key in the dictionary. This ensures that there are no duplicate entries and allows for quick and accurate retrieval of the corresponding password.
3. Efficient Storage: The dictionary data structure optimizes the storage of password hashes and their passwords. It uses a hash function internally to compute the index of each entry, which enables efficient storage and retrieval without needing to iterate through all the entries.
4. Fast Insertion and Deletion: The dictionary allows for efficient insertion and deletion of entries. When generating the rainbow table, each new password hash and its corresponding password can be added to the dictionary in constant time.

Overall, the dictionary data structure used in the rainbow table construction algorithm provides fast and efficient lookup operations, ensuring the effectiveness of the password cracking process. It optimizes storage, allows for constant-time retrieval, and facilitates the management of unique key-value pairs.

# Conclusion

In conclusion, this project has implemented a rainbow table-based Python application for password hash cracking. Based on their preferred hashing algorithm, character set, password length, chain count, and chain length, users of the application can generate rainbow tables. The rainbow table is then used to decrypt password hashes and obtain the original passwords.

The project's accomplishments include:

1. Construction of Rainbow Tables: By generating password chains, carrying out reduction operations, and storing the resulting password hashes and passwords in a dictionary data structure, the application efficiently constructs rainbow tables.
2. Cracking Password Hashes: Using a pre-generated rainbow table, the application successfully cracks password hashes. It demonstrates how quickly passwords can be recovered using rainbow tables, especially for passwords with fixed lengths and constrained character sets.
3. Optimization and Efficiency: The application uses improved rainbow table construction, password cracking algorithms, and reduction function implementation. These enhancements increase the application's general efficiency and cut down on the time and resources needed to decipher password hashes.
4. Testing and Validation: The program has undergone thorough testing, including unit tests for every function and timing tests for various program elements. This testing procedure guarantees the functionality of the application is accurate and reliable.

This project has demonstrated the strength and efficiency of rainbow tables in password hash decryption. Rainbow tables provide a useful and effective method for password cracking by using reduction functions and password chains. Security experts, system administrators, and researchers can use the application as a useful tool to test and assess the robustness of password storage mechanisms.

# References

1. Feldman, L., & Biham, E. (1997). Making a Faster Cryptanalytic Time-Memory Trade-Off. Journal of Cryptology, 10(1), 1-28.
2. Oechslin, P. (2003). Making a Faster Cryptanalytic Time-Memory Trade-Off. In Proceedings of the 2003 IEEE Symposium on Security and Privacy (S&P'03) (pp. 178-192). IEEE.
3. Rivest, R. L. (1992). The MD5 Message-Digest Algorithm. Retrieved from <https://www.ietf.org/rfc/rfc1321.txt>
4. National Institute of Standards and Technology. (2015). Secure Hash Standard (SHS). Federal Information Processing Standards Publication 180-4. Retrieved from <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>
5. Wikipedia contributors. (2022). Rainbow table. In Wikipedia. Retrieved March 16, 2022, from <https://en.wikipedia.org/wiki/Rainbow_table>
6. Ferguson, N., & Schneier, B. (2003). Practical Cryptography. John Wiley & Sons.
7. Lutz, M. (2013). Learning Python: Powerful Object-Oriented Programming (5th ed.). O'Reilly Media.
8. Seaborn, C., & Streeter, A. (2017). Python for Data Analysis (2nd ed.). O'Reilly Media.
9. Van Rossum, G., & Drake, F. L. (2009). Python tutorial. Python Software Foundation.
10. McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython (2nd ed.). O'Reilly Media.