Journal of Theoretical and Applied Information Technology
28th February 2023. Vol.101. No 4
© 2023 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SIMULATION OF THE RAINBOW ATTACK ON THE SHA-256 HASH FUNCTION

MANANKOVA O.A.[1], YAKUBOVA M.Z.[2], RAKHMATULLAEV M.A.[3], BAIKENOV A.S.[4]

[1,2,4] Almaty University of Power Engineering and Telecommunications named after Gumarbek Daukeyev, Almaty, Kazakhstan

[3] Tashkent University of Information Technologies named after Muhammad al-Khwarizmi Tashkent, Uzbekistan

E-mail: [1]olga_manank@gmail.com, [2]m.yakubova@aues.kz, [3]marat56@mail.ru, [4]a.bailenov@aues.kz

ABSTRACT

The value of data is growing every day. Data is a key factor in both scientific research and public administration. The development of IT technologies has led to the generation of a large amount of personal data, which has become the basis for the development of machine learning technologies and big data processing. This growing demand is bringing renewed interest in data privacy practices and processes. The study of the strength of a hashed message is of great importance in modern authentication systems. The hashing process is inextricably linked to the password system, since passwords are usually stored in the system not in clear text, but in the form of hashes. The SHA-256 hash function was chosen to model therainbow tables attack. An algorithm for constructing a rainbow table for the SHA-256 hash function in the Cryptool 2 environment is proposed. The conditions under which the use of rainbow tables will be effective are determined. This article aims to practically show the process of generating a password and rainbow tables to organize an attack on the SHA-256 hash function. Studies show that rainbow tables can reveal a password without salt faster than with salt. As the password bit increases, the decryption time increases in direct proportion.

Keywords: *Simulation, Salt, Hash Function, Attack, Rainbow Tables, SHA-256, Cryptanalysis.*

1. INTRODUCTION

A hash function is a mathematical function that takes in an input (called the "message") and produces a fixed-size output (called the "hash value" or "message digest"). Hash functions have a number of important properties:

They are deterministic, meaning that the same input will always produce the same output.

They are one-way, meaning that it is computationally infeasible to reconstruct the original input from the hash value.

They are collision-resistant, meaning that it is computationally infeasible to find two different inputs that produce the same hash value.

Hash functions are widely used in computer science and cryptography for a variety of purposes, including data integrity, password storage, and digital signatures. There are many different types of hash functions, each with its own set of characteristics and trade-offs. Some examples of popular hash functions include MD5, SHA-1, SHA-2, and SHA-256.

It's important to choose a strong and secure hash function for your specific use case, as weaker hash functions may be vulnerable to attacks that can compromise their security properties. It's also important to consider the speed and efficiency of the hash function, as some functions may be slower or more resource-intensive to compute than others [1]-[4].

SHA-256 (Secure Hash Algorithm 256-bit) is a widely-used cryptographic hash function that produces a fixed-size output (called a "message digest") from an input (called a "message"). It is part of the SHA-2 family of hash functions, which also includes SHA-224, SHA-384, and SHA-512.

SHA-256 was designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) in 2001. It is widely used in a variety of applications, including data integrity, password storage, and digital signatures.

One of the main advantages of SHA-256 is its security. It is considered a strong and secure hash function, with a very low probability of collision (i.e., two different inputs producing the same hash value). It is also resistant to a number of known attacks, including preimage attacks and second-preimage attacks. SHA-256 is also relatively fast and efficient, making it well suited for use in a variety of applications. It produces a 256-bit hash value, which is longer than some other hash functions (such as MD5 and SHA-1), providing an additional layer of security.

Overall, SHA-256 is a widely used and trusted hash function that is suitable for a variety of applications requiring strong security and efficiency.

The rainbow attack is a cryptanalytic attack on the Rainbow cryptographic hash function. The attack involves constructing a "rainbow table" of precomputed hash values for a given set of inputs, which can then be used to efficiently determine the input associated with a given hash value.

To simulate the rainbow attack, you would need to create a rainbow table for a given set of inputs and a target hash value. This could involve generating a large number of inputs, hashing them with the Rainbow hash function, and storing the resulting hash values in the rainbow table. Then, you could use the rainbow table to search for a given target hash value, starting with the hash values at the end of the table and working backwards towards the beginning. If the target hash value is found in the table, you can determine the corresponding input that produced it.

There are various tools and software packages available that can help you generate and use rainbow tables for simulating the rainbow attack [5]-[7].

Rainbow attacks can be a threat to organizations because they can be used to crack weak or easily guessable passwords. It is important for organizations to use strong, unique passwords for all accounts and to regularly update them to reduce the risk of successful rainbow attacks. In addition, organizations can use techniques such as salting and hashing to make it more difficult for attackers to crack passwords using rainbow tables.

It is also important for organizations to implement other security measures, such as enabling encryption and using firewalls, to protect against other types of attacks. Keeping systems and software up to date with the latest security patches and updates can also help reduce the risk of successful attacks [8]-[10].

Rainbow tables are used to crack password hashes by precompiling a table of hashes for a large number of possible passwords. The attacker can then use the table to quickly look up the hash of a given password and compare it to the hash of a password that they are trying to crack. If the hashes match, the attacker has successfully cracked the password.

Rainbow tables can be used to attack systems that use simple or weak passwords, as well as systems that use unsalted hashes. However, rainbow tables are less effective against systems that use strong, unique passwords and that employ techniques such as salting and hashing to make it more difficult to crack passwords.

To protect against rainbow attacks and other types of cryptanalytic attacks, it is important to use strong, unique passwords and to implement security measures such as encryption and firewalls. Regularly applying security updates and patches can also help reduce the risk of successful attacks [11].

It is important to use strong and secure methods for protecting data and traffic in an IP telephony network. Weak or poorly implemented hash functions can make it easier for attackers to conduct cryptanalytic attacks, such as rainbow attacks, and potentially compromise the security of the network.

Rainbow attacks rely on precomputing a table of hashes for a large number of possible passwords and then using the table to quickly look up the hash of a given password. If the hashes match, the attacker has successfully cracked the password. Rainbow attacks can be used to crack weak or easily guessable passwords and may be able to compromise the security of systems that use simple or unsalted hashes.

To protect against rainbow attacks and other types of cryptanalytic attacks, it is important to use strong, unique passwords and to implement security measures such as encryption and firewalls. Regularly applying security updates and patches can also help reduce the risk of successful attacks. In addition, using secure hash functions and employing techniques such as salting and hashing can make it more difficult for attackers to crack passwords and compromise the security of the network [13]-[15].

Rainbow tables can indeed be used for purposes other than attacking and cracking passwords. In some cases, rainbow tables may be used to help recover or reset a forgotten password.

For example, if a user has forgotten the password for their account, the system administrator may be able to use a rainbow table to look up the hash of the user's password and compare it to the stored hash in the system. If the hashes match, the administrator can then reset the user's password to a new, secure value.

It is important to note, however, that using rainbow tables to recover or reset passwords should only be done with the proper authorization and in accordance with relevant policies and procedures. Additionally, rainbow tables should only be used as a last resort, as other methods, such as password reset questions or security questions, may be more secure and less risky.

It is also important to remember that rainbow tables can be used by attackers to conduct unauthorized password cracking and to compromise the security of systems. Therefore, it is important to use strong, unique passwords and to implement other security measures to protect against rainbow attacks and other types of cryptanalytic attacks [16]-[18].

Also, one of the modern directions of development of the IT sector is the introduction of cloud services. In this area, ensuring the security of the end user comes to the fore, since the system needs to guarantee not only uninterrupted access to cloud services, but also the confidentiality of the transmitted data. At the same time, it is proposed to use a secure hashing algorithm [19]-[22].

The main advantage of using rainbow tables is that they can significantly reduce the time and resources required to crack a password.

Instead of trying to crack a password by guessing and hashing each possible combination, an attacker can use a rainbow table to look up the hash of a given password in a matter of seconds. This can make it much easier for an attacker to crack simple or weak passwords and compromise the security of a system.

The novelty of rainbow tables lies in their ability to significantly reduce the time and resources required for password cracking. Prior to the development of rainbow tables, attackers had to rely on more time-consuming and resource-intensive methods, such as brute-force attacks, to crack passwords. Rainbow tables provided a more efficient way to conduct password cracking and made it easier for attackers to compromise the security of systems.

However, it is important to note that rainbow tables are less effective against systems that use strong, unique passwords and that employ techniques such as salting and hashing to make it more difficult to crack passwords. In addition, rainbow tables can be used for purposes other than attacking and cracking passwords, such as to help recover or reset a forgotten password.

The article proposes to simulate a rainbow table attack on one of the modern sha-256 hash functions. When using a rainbow table, the article will show how vulnerable this hash function is.

2. METOD

Rainbow tables are a type of attack tool that can be used to crack password hashes by precomputing a table of hashes for a large number of possible passwords. Here is a general overview of how rainbow tables work:

The attacker generates a large number of possible passwords and hashes each one using the same hashing algorithm that is used by the system they are trying to crack.

The attacker creates a table that contains the hashes of the possible passwords and stores it in a convenient format, such as a database or file. This table is known as a rainbow table.

When the attacker wants to crack a password, they hash the password they are trying to crack and look up the resulting hash in the rainbow table.

If the hash of the password they are trying to crack is found in the rainbow table, the attacker can determine the original password by looking up the corresponding entry in the table.

If the hash is not found in the rainbow table, the attacker must either generate a new rainbow table or use a different cracking method.

A rainbow table is a special variant of lookup tables for inverting cryptographic hash functions, using a reasonable compromise mechanism between table lookup time and memory footprint [23]-[24].

Rainbow tables are primarily used to crack passwords that have been converted using a hard-to-reverse hash function.

Here is an example of a simple rainbow table that could be used to crack the SHA-256 hash of a six-character lowercase password:

$$\mathbf{aaaaaa} \xrightarrow[H]{} 281DAF40 \xrightarrow[R]{} sgfnyd \xrightarrow[H]{} 920ECF10 \xrightarrow[R]{} \mathbf{kiebgt}, \quad (1)$$

here H is the hash function (eg SHA-1 or SHA-2) and R is the reduction function.

A function of reduction is used in rainbow table attacks to reduce the resulting hash of a plaintext password to a fixed-length string. This is done to reduce the size of the rainbow table and make it more computationally efficient. The reduction function is typically a mathematical function that maps the hash value to a smaller value that is used as the starting point for a chain of hashes.

To use the rainbow table to crack a password hash, you would start by looking up the hash in the last column of the table. If the hash is found, you can trace the chain of hashes back to the start value to find the original password. If the hash is not found, you would move on to the next rainbow table.

To create a chain of hashes for a rainbow table, you can use the following steps:

1. The working alphabet is fixed, that is, the set Q of all possible keys is given.

2. An element q from the set Q is fixed and the value h of the hash function on it is calculated.

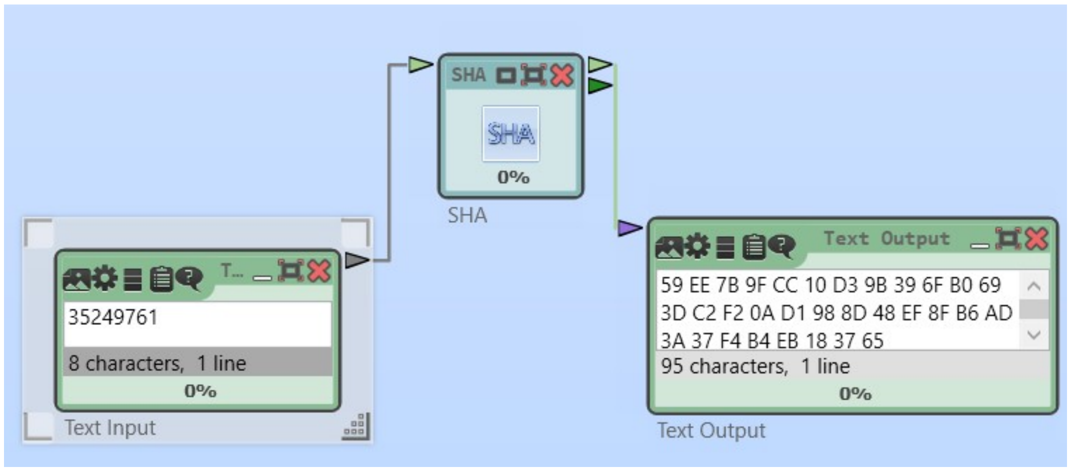
3. Using some function R, a key belonging to the set Q is generated from the hash: q=R(h). If the number of elements in the chain is less than the specified one, go to step 2.

These operations will be repeated until a chain of length t keys is obtained. This sequence is not placed entirely in memory, only the first and last elements of it are written. This is the time-memory tradeoff - let's say we generate a chain of 2000 keys, and only the first and last elements are recorded, we get huge savings, but on the other hand, the cryptanalysis time increases.

You can also use software tools to automate the process of creating rainbow tables. the article proposes a program that will generate rainbow tables. In [25] program is written in Java and we performed a stability check of SHA-256, taking into account the use of 3 characteristics. In this work, we perform a simulation on a Cryptool 2 with an increase in the number of characteristics and visualisation process.

3. SIMULATION AND RESULTS

To demonstrate the use of rainbow tables, a simple example can be given: suppose a password is generated consisting of 8 decimal digits in the range from 0 to 9. To simulate and calculate the SHA-256 function using Cryptool 2 environment (Figure 1).



```
35249761
8 characters, 1 line
0%
```

```
59 EE 7B 9F CC 10 D3 9B 39 6F B0 69
3D C2 F2 0A D1 98 8D 48 EF 8F B6 AD
3A 37 F4 B4 EB 18 37 65
95 characters, 1 line
0%
```

Figure 1: The SHA-256 hash function

Password hash: 59 EE 7B 9F CC 10 D3 9B 39 6F B0 69 3D C2 F2 0A D1 98 8D 48 EF 8F B6 AD 3A 37 F4 B4 EB 18 37 65.

The number of passwords that can be generated in the rainbow table alphabet for 8 characters from the decimal digits 0 to 9 is calculated by the formula:

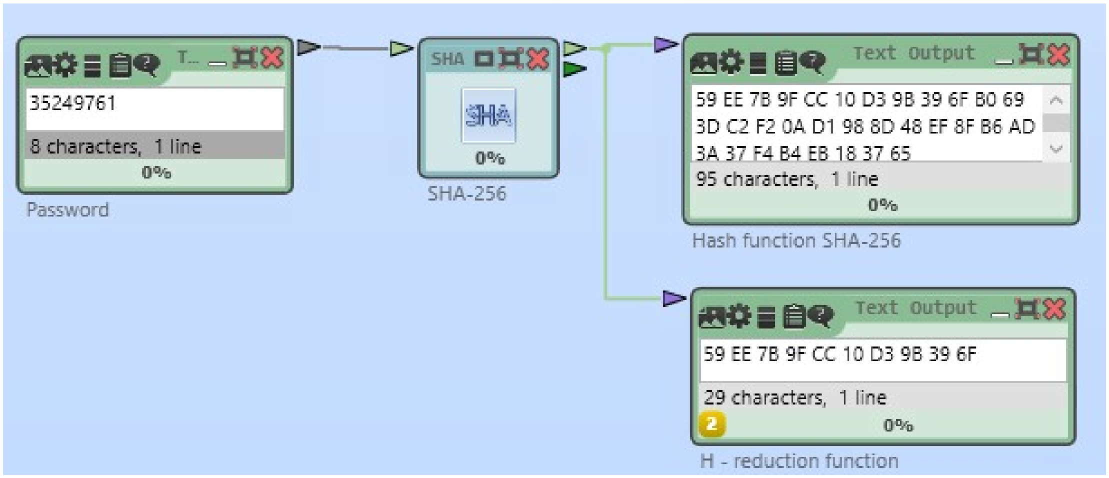
Number of passwords = 10^8. (1)

This is because there are 10 decimal digits (0 to 9) and 8 characters, so each character can take any of the 10 possible values.

It is worth saying that a real rainbow table that stores all possible passwords up to 6 characters long (and this includes all printable characters) stores about 2 million values and consists of chains of about 1000 iterations in length. It can take up to 10 hours to look up a password against such a table if it is performed on a medium power machine, for example, based on a Core i3 processor. To demonstrate the principle of operation of rainbow tables, we will generate chains of only 4 hashes to save time and computer performance.

So, the number of all possible passwords under these conditions is 100 000 000. It is worth noting that ideally this table should consist of a much larger number of chains and iterations in them (even with such a small number of possible passwords), however, for simplicity, our table will consist of 4-x chains with a length of 8 iterations (thus, there will be 4 passwords in each of the chains). Although in this scenario it is obvious that the last elements of the chains will not receive their hashes. Each of them will have its own reduction function - R1, R2, R3, respectively. Here we need to remember that the only requirement for the reduction function is to return values from the same alphabet as the passwords.

An example of a Cryptool 2 that generate one chain of 4 password rainbow table for an password of 8 digits alphabet from 0 to 9 shows in Figure 2. Figure 2 shows adding function of reduction R and create of chain of password.


Password: 35249761
8 characters, 1 line
0%

SHA-256
0%

Hash function SHA-256 (Text Output):
59 EE 7B 9F CC 10 D3 9B 39 6F B0 69
3D C2 F2 0A D1 98 8D 48 EF 8F B6 AD
3A 37 F4 B4 EB 18 37 65
95 characters, 1 line
0%

H - reduction function (Text Output):
59 EE 7B 9F CC 10 D3 9B 39 6F
29 characters, 1 line
2
0%

Figure 2: The SHA-256 hash function with R

Let 4 passwords be chained like this: 35249761 — 31526487 — 35716429 — 82457163. A hash of passwords is generated for each chain in the Cryptool 2 how shows in Figure 2.

The one of the chain consist of 4 passwords, using Cryptool 2 generating shows in Figure 3.

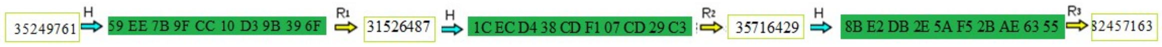

35249761
H
59 EE 7B 9F CC 10 D3 9B 39 6F
R1
31526487
H
1C EC D4 38 CD F1 07 CD 29 C3
R2
35716429
H
8B E2 DB 2E 5A F5 2B AE 63 55
R3
82457163

Figure 3: Chain of passwords

Thus we generate chains of sequences. Ideally, for the selected number of characteristics 8 with an alphabet from 0 to 10, there are quite a lot of such hashes. for the experiment we take only 4 chains:

35249761 — 31526487 — 35716429 — 82457163;
56487216 — 26814329 — 96315247 — 17426853;
12345678 — 85213467 — 98741236 — 52416387;
78461325 — 41963527 — 64921857 —24356891.

This sequence is not placed entirely in memory, only the first and last elements of it are written.

The chain has been created. Further, we will assume that its first and last elements have been written to the memory.

That is:

35249761— 82457163;
56487216 — 17426853;
12345678 — 52416387;
78461325 — 24356891.

After that, we begin to simulate an attack, which in essence is a search for a hash from this table and the password corresponding to it.

The search is carried out as follows: first, the last column with hashes is checked for a match with the required hash. If no match is found, the penultimate column is checked, and so on. When

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the desired hash has been found in a certain column of a certain chain, the entire chain will be restored, and thus, the password preceding this hash in the chain is the one being sought. So, we start checking the column of hashes:

59 EE 7B 9F CC 10 D3 9B 39 6F
1C EC D4 38 CD F1 07 CD 29 C3
8B E2 DB 2E 5A F5 2B AE 63 55

We check for example hash:

59 EE 7B 9F CC 10 D3 9B 39 6F B0 69 3D C2 F2 0A D1 98 8D 48 EF 8F B6 AD 3A 37 F4 B4 EB 18 37 65.

The desired hash is found. Therefore, we restore the desired chain:
35249761 — 31526487 — 35716429 — 82457163.
Answer: the desired password is 35249761.

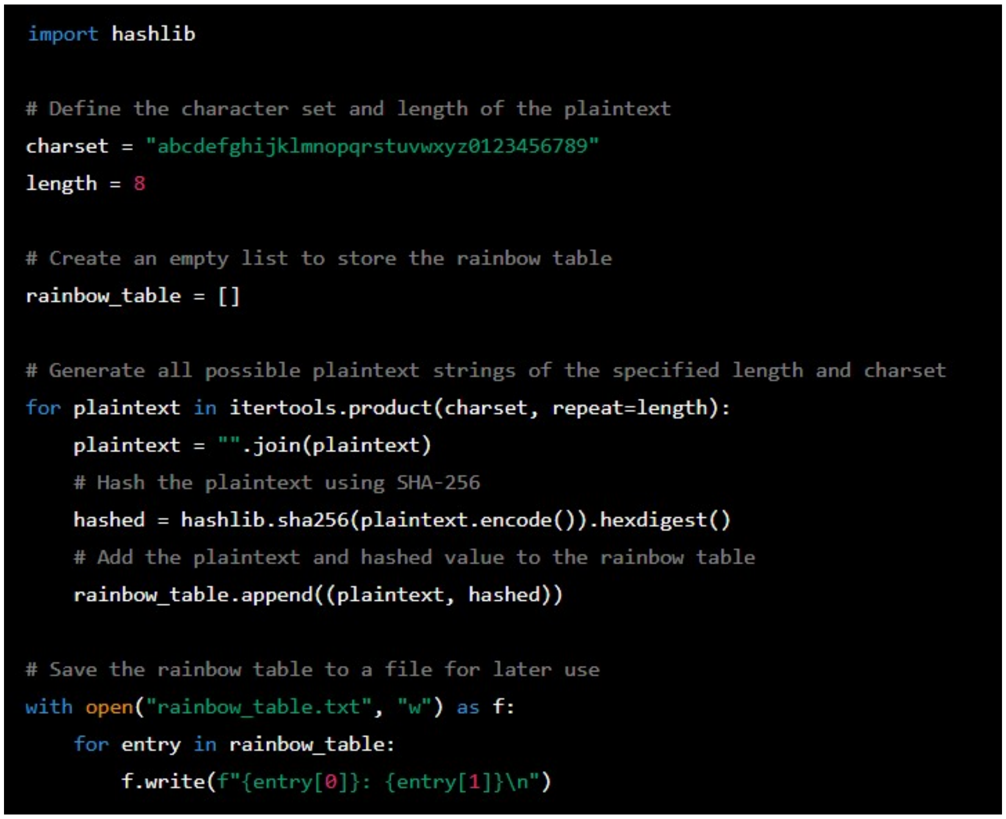
It may seem that all these manipulations with searching for a hash by columns and restoring the chain in which the hash was found are superfluous, because there are only 4 chains and it is so perfectly clear in them which hashes correspond to which passwords. However, do not forget that this is the most simplified example, and the computer will have to deal with tens of thousands, millions or even billions of passwords.

3.1 Rainbow tables without "salt" in Python

Here is an example of how you might simulate a rainbow table in Python. Generate a list of plaintext passwords and hash shows in Figure 4.

This code uses the itertools library to generate all possible plaintext strings of a specified length and character set, which are then hashed using the SHA-256 algorithm. The resulting plaintext and hashed values are stored in a list, which is then saved to a file for later use.

It is important to note that this is a simplified example, and the specific requirements and constraints of the system will determine the size and complexity of the rainbow table. Building a rainbow table for a larger set of characters or longer plaintexts will require more processing power and storage. Some values are presented in Table 1.



```python
import hashlib

# Define the character set and length of the plaintext
charset = "abcdefghijklmnopqrstuvwxyz0123456789"
length = 8

# Create an empty list to store the rainbow table
rainbow_table = []

# Generate all possible plaintext strings of the specified length and charset
for plaintext in itertools.product(charset, repeat=length):
    plaintext = "".join(plaintext)
    # Hash the plaintext using SHA-256
    hashed = hashlib.sha256(plaintext.encode()).hexdigest()
    # Add the plaintext and hashed value to the rainbow table
    rainbow_table.append((plaintext, hashed))

# Save the rainbow table to a file for later use
with open("rainbow_table.txt", "w") as f:
    for entry in rainbow_table:
        f.write(f"{entry[0]}: {entry[1]}\n")
```

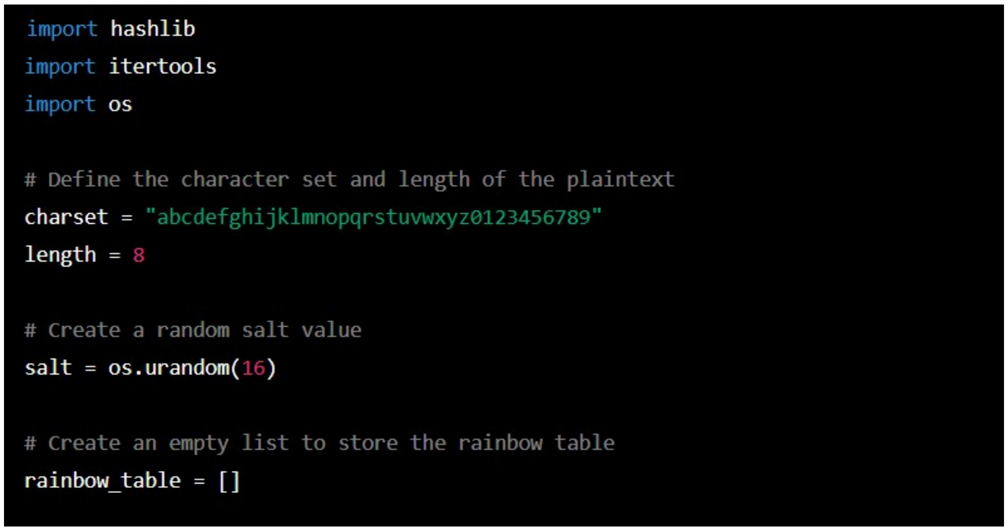
Figure 4: Python description of the code for the generate a list passwords and SHA-256 hash function

Table 1: Hash without "salt"

Password	Hash
35249761	59 EE 7B 9F CC 10 D3 9B 39 6F B0 69 3D C2 F2 0A D1 98 8D 48 EF 8F B6 AD 3A 37 F4 B4 EB 18 37 65
31526487	1C EC D4 38 CD F1 07 CD 29 C3 16 25 C4 D7 BB 3C 56 F1 90 0A 6A 11 0E 5B EE C8 46 6B 88 FC A3 8A
35716429	8B E2 DB 2E 5A F5 2B AE 63 55 08 E2 B2 33 2F C8 B4 8C 0C E3 B6 01 A0 34 01 E9 43 3D 25 2D CD 22
82457163	E2 58 2A 02 87 35 4F 05 71 D8 35 40 55 7A E8 87 05 48 A9 28 CB 1E 03 11 5B 87 51 98 19 ED 93 50

3.2 Rainbow tables with "salt" in Python

A rainbow table with salt is a precomputed table of hash values that can be used to quickly reverse the process of a one-way hash function, such as SHA-256. The salt value is added to the plaintext before hashing, to make it more difficult to crack the hash. Here is an example of how you could create a rainbow table with salt for SHA-256 in Python (Figure 5, 6):



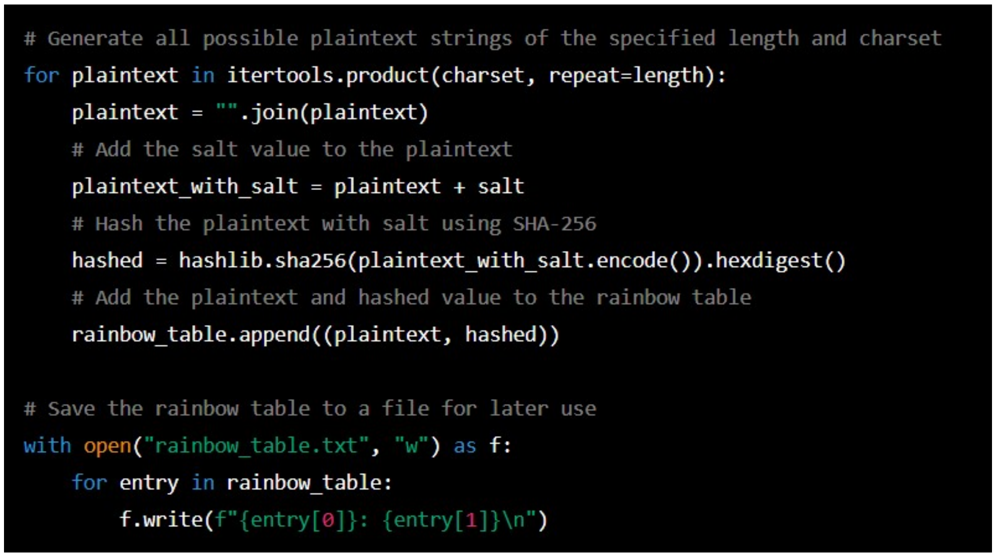
```
import hashlib
import itertools
import os

# Define the character set and length of the plaintext
charset = "abcdefghijklmnopqrstuvwxyz0123456789"
length = 8

# Create a random salt value
salt = os.urandom(16)

# Create an empty list to store the rainbow table
rainbow_table = []
```

Figure 5: Python description of the code for the generate "salt"



```
# Generate all possible plaintext strings of the specified length and charset
for plaintext in itertools.product(charset, repeat=length):
    plaintext = "".join(plaintext)
    # Add the salt value to the plaintext
    plaintext_with_salt = plaintext + salt
    # Hash the plaintext with salt using SHA-256
    hashed = hashlib.sha256(plaintext_with_salt.encode()).hexdigest()
    # Add the plaintext and hashed value to the rainbow table
    rainbow_table.append((plaintext, hashed))

# Save the rainbow table to a file for later use
with open("rainbow_table.txt", "w") as f:
    for entry in rainbow_table:
        f.write(f"{entry[0]}: {entry[1]}\n")
```

Figure 6: Python description of the code for generate a list passwords and SHA-256 hash function with "salt"

This code uses the itertools library to generate all possible plaintext strings of a specified length and character set. It also uses the os.urandom(16) function to create a random 16 bytes salt value. Next, it concatenates the plaintext with the salt value, then it hashes the plaintext with salt using the SHA-256 algorithm.

The resulting plaintext and hashed values are stored in a list, which is then saved to a file for later use. Some values are presented in Table 2.

Table 2: Hash with "salt"

Password	Salt	Hash
35249761	123	CA 21 8B 75 CE D2 61 02 87 C3 32 FC A7 7F B4 81 FF C1 49 8E 7D 1B 34 22 6B 17 AE 67 3C B5 CE D2
31526487	456	16 22 3F A8 F7 02 F3 E4 0F 1B 51 8E 87 C3 1E 01 2E B6 A8 D5 4B B7 11 EC 14 48 2C 28 3F 98 A8 F7
35716429	789	35 A2 81 31 D6 F2 AB 6B 21 FC 75 E0 AD 5A 23 D4 B3 44 D1 77 4D C3 FD B9 06 A0 76 CF BE 7B AE B1
82457163	015	AA A9 71 D7 AA A6 2F 2A C6 B8 A5 7C 8F B1 1D 1C 6F 1A B1 CB 1F A6 86 D6 11 B2 3D B7 64 7E 2E FE

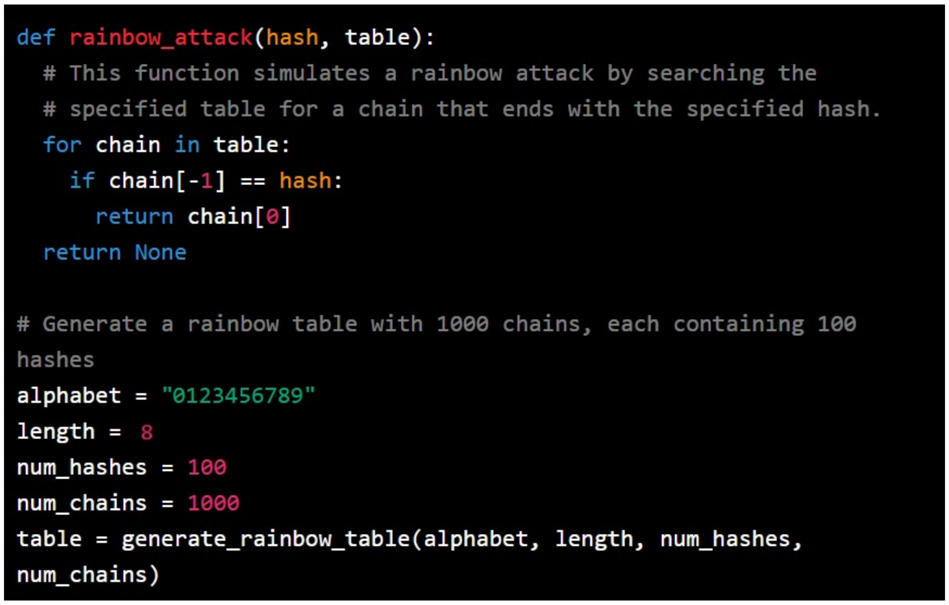
3.2 Rainbow table attack

A rainbow table attack is a type of precomputation attack used to crack hashed passwords. The idea behind a rainbow table is to precompute a large table of possible plaintext password and their corresponding hashed values, so that when an attacker wants to crack a specific password, they can simply look it up in the table, rather than having to perform a brute-force or dictionary attack.

Rainbow tables are typically used to crack simple, short, and unsalted passwords, as the size of the table required to crack such a password can be made small enough to fit on a single computer.

This code will generate a rainbow table with 1000 chains, each containing 100 hashes, and then use the table to try to recover the password for the hash hash in Python. If the password is found in the table, it will be printed to the console. If the password is not found, the message "Failed to recover password." will be printed (Figure 7,8).

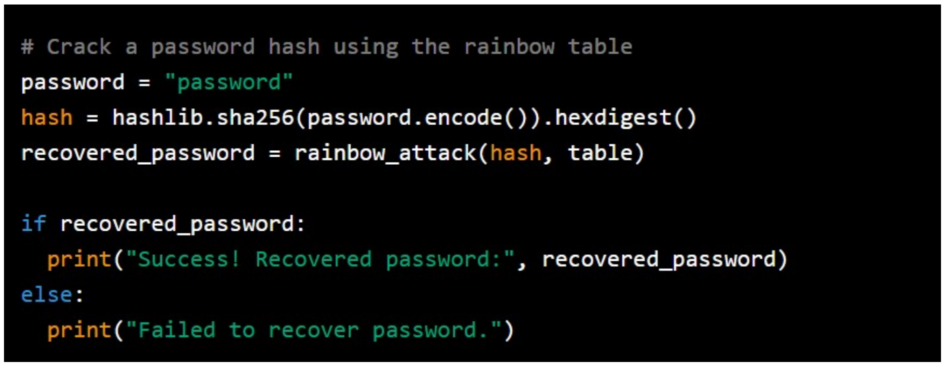
Keep in mind that this is just a simple example, and in a real rainbow attack, you would need to use a much larger table and a more sophisticated reduction function to have a good chance of success. For 8 characteristics with alphabet from 0 to 9 success result was 1 min.



```
def rainbow_attack(hash, table):
  # This function simulates a rainbow attack by searching the
  # specified table for a chain that ends with the specified hash.
  for chain in table:
    if chain[-1] == hash:
      return chain[0]
  return None

# Generate a rainbow table with 1000 chains, each containing 100
hashes
alphabet = "0123456789"
length =  8
num_hashes = 100
num_chains = 1000
table = generate_rainbow_table(alphabet, length, num_hashes,
num_chains)
```

Figure 7: Python description of rainbow tables attak



```
# Crack a password hash using the rainbow table
password = "password"
hash = hashlib.sha256(password.encode()).hexdigest()
recovered_password = rainbow_attack(hash, table)

if recovered_password:
  print("Success! Recovered password:", recovered_password)
else:
  print("Failed to recover password.")
```

Figure 8: Python description of the result

CONCLUSION

Data privacy and security is becoming increasingly important as the amount of personal data generated continues to grow. The study of the strength of a hashed message is critical in modern authentication systems, and the use of secure hashing algorithms such as SHA-256 is essential in protecting sensitive information. Rainbow table attacks are a precomputation method used to crack hashed passwords, and while they are effective against simple and unsalted passwords, they become less practical as the length and complexity of a password increases. The proposed algorithm for constructing a rainbow table for the SHA-256 hash function in Cryptool 2 and Python shows that the use of rainbow tables can reveal a password without salt faster than with salt, and the time it takes to crack a password increases in direct proportion to the length of the password. It's important to note that cracking password without permission is illegal in many jurisdictions, it should be used only for educational or testing purposes, and not for malicious intent. It's highly recommended to use long, unique and complex passwords, combined with a secure password storage method and multi-factor authentication, in order to make the use of a rainbow table attack infeasible.

REFERENCES

[1] X. Wang, Y. L. Yin, H. Yu, “Finding Collisions in the Full SHA-1,” *25th Annual International Cryptology Conference*, Santa Barbara, California, USA, Aug. 14-18, pp. 17-36, 2005.
[2] T. Bhorkar, “A Survey of Password Attacks and Safe Hashing Algorithms,” *International Research Journal of Engineering and Technology,* vol. 4, no. 12, pp.1554-1556, 2017.
[3] D. Smith-Tone1, R. Perlner, “Rainbow Band Separation is better than we thought,” *Cryptology ePrint Archive*, 2020, Art. no. 2020/702. [Online]. Available: https://eprint.iacr.org/2020/702.
[4] K. Theoharoulis, C. Manifavas, I.Papaefstathiou, “HighEnd Reconfigurable Systems for Fast Windows’ Password Cracking,” *17th EEE Symposium on Field Programmable Custom Computing Machines*, 2009, pp. 287-290, doi:10.1109/fccm.2009.48
[5] B. Shavers, J. Bair, “Cryptography and Encryption. Hiding behind the Keyboard,” *Syngress,* ch.6, Mart. 2016, pp. 133-151, doi:10.1016/b978-0-12-803340-1.00006-9.
[6] J. Blakstad, R. Walso Nergard, M. G. Jaatun, “All in a day's work: Password cracking for the rest of us,” *Norwegian Symposium on Information Security,* pp.69-83, 2009.
[7] L.Zhang, C. Tan, F. Yu, “An Improved Rainbow Table Attack for Long Passwords,” *Procedia Computer Science,* vol. 107, Apr. 2017. pp. 47-52, doi:10.1016/j.procs.2017.03.054.
[8] P. Patel, P. Goswami, A. Mishra, S. Khan, A. Choudhary, “Brute force, dictionary and rainbow table attack on hashed passwords,” *2021 IJCRT*, vol. 9, no. 4, Apr. 2021, pp.1899-1905. [Online]. Available: https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf.
[9] A. Joshi, M. Wazid, R.H. Goudar, “An Efficient Cryptographic Scheme for Text Message Protection Against Brute Force and Cryptanalytic Attacks,” *Procedia Computer Science,* vol. 48, pp. 360-366, 2015.
[10] S. Verma, R. Choubey, R. Soni, “An Efficient Developed New Symmetric Key

Cryptography Algorithm for Information Security," *International Journal of Emerging Technology and Advanced Engineering,* vol. 2, no. 7, pp.18-21, Jul. 2012.

[11] G. Avoine, A. Bourgeois, X. Carpent. (2015). Analysis of Rainbow Tables with Fingerprints. [Online]. Available: doi:10.1007/978-3-319-19962-7_21.

[12] K.C.Redmon, "C0D3 CR4CK3D: Means and Methods to Compromise Common Hash Algorithms," Jul. 2006. [Online]. Available: http://uninfo.mecon.gov.ar/htmls/boletinSI/images/Hash_Algorithms_KRedmon.pdf.

[13] S.V. Konshin, M.Z.Yakubova, T.N. Nishanbayev, O.A. Manankova, "Research and development of an IP network model based on PBX asterisk on the opnet modeler simulation package," *International Conference on Information Science and Communications Technologies (ICISCT 2020),* Oct. 2020, Art. no. 20486746, doi: 10.1109/ICISCT50599.2020.9351405.

[14] O.A. Manankova, B.M. Yakubov, T.G. Serikov, M.Z. Yakubova, A.K. Mukasheva, "Analysis and research of the security of a wireless telecommunications network based on the IP PBX Asterisk in an Opnet environment," *Journal of Theoretical and Applied Information Technology*, vol. 99, no.14, pp. 3617-3630, 2021.

[15] T.G. Serikov, M.Z. Yakubova, A.D Mekhtiev, A.V.Yurchenko, A.D. Alkina, "The analysis and modeling of efficiency of the developed telecommunication networks on the basis of IP PBX asterisk now," *Proceedings - 2016 11th International Forum on Strategic Technology*, 2017, pp. 510-515, Art. no. 7884168.

[16] S. Tayal, N. Gupta, P. Gupta, D. Goyal, M. Goyal, "A Review paper on Network Security and Cryptography," *Advances in Computational Sciences and Technology,* vol.10, no. 5, pp. 763-770, 2017.

[17] T. Saravanan, S.V. Kumar, "A Review Paper on Cryptography-Science of Secure Communication," *Int. J. of Comp. Sci. Trends and Tech.*, vol. 6, no 4, 2018.

[18] A. M. Qadir, N. Varol, "A Review Paper on Cryptography," *7th International Symposium on Digital Forensics and Security (ISDFS),* Barcelos, Portugal, 2019, pp. 1-6, doi: 10.1109/ISDFS.2019.8757514.

[19] Th. Velmurugan, S. Karthiga, "Security based Approach of SHA 384 and SHA 512 Algorithms in Cloud Environment," *Journal of Computer Science,* vol. 16, no. 10, 2020, pp. 1439-1450, doi: 10.3844/jcssp.2020.1439.1450.

[20] P. Semwal, M. K. Sharma, "Comparative study of different cryptographic algorithms for data security in cloud computing," *3rd International Conference on Advances in Computing, Communication & Automation (ICACCA),* pp.1-7, Sept. 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8344738/.

[21] M. Ansar, I. A. Shokat, M. Fatima, K. Nazir, "Security of Information in Cloud Computing: A Systematic Review," *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*, vol. 48, no.1, pp. 90-103, 2018. [Online]. Available: http://www.asrjetsjou.

[22] C. Yang, Q. Huang, Z. Li, K. Liu, F. Hu, "Big Data and cloud computing: innovation opportunities and challenges," *International Journal of Digital Earth,* vol. 10, no. 1, pp.13-53, 2017.

[23] Rainbow table. [Online]. Available: https://ru.wikipedia.org/wiki/Rainbow table.

[24] Security online community. Antichat: FAQ - Rainbow Tables. [Online]. Available: https://forum.antichat.ru/threads/37964 /.

[25] O. Manankova, M.Yakubova, A. Baikenov. Cryptanalysis the SHA-256 Hash Function using Rainbow Tables. Indonesian Journal of Electrical Engineering and Informatics (IJEEI), vol. 10, no. 4, 2022. doi: 10.52549/ijeei.v10i4.4247