Indonesian Journal of Electrical Engineering and Informatics (IJEEI)
Vol. 10, No. 4, December 2022, pp. 930~944
ISSN: 2089-3272, DOI: 10.52549/ijeei.v10i4.4247

Cryptanalysis the SHA-256 Hash Function Using Rainbow Tables

Olga Manankova[1], Mubarak Yakubova[2], Alimjan Baikenov[3]
[1,3]Department of Telecommunications and Space Engineering, Faculty of Telecommunications and Innovation Technologies, Almaty University of Power Engineering and Telecommunications name after Gumarbek Daukeev, Almaty, Kazakhstan
[2]Department of Information Technology, Faculty of Information Systems and Cybersecurity, Almaty University of Power Engineering and Telecommunications name after Gumarbek Daukeev, Almaty, Kazakhstan

Article Info

Article history:

Received Oct 21, 2022
Revised Dec 3, 2022
Accepted Dec 20, 2022

Keywords:

Hash function
Attack
Rainbow tables
SHA-256
Java

ABSTRACT

The research of the strength of a hashed message is of great importance in modern authentication systems. The hashing process is inextricably linked with the password system, since passwords are usually stored in the system not in clear text, but as hashes. The SHA-256 hash function was chosen to model the attack with rainbow tables. An algorithm for constructing a rainbow table for the SHA-256 hash function in the Java language 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. As research shows, rainbow tables can reveal a three-character password in 3 seconds. As the password bit increases, the decryption time increases in direct proportion.

Corresponding Author:

Olga Manankova
Department of Telecommunications and Space Engineering, Faculty of Telecommunications and Innovation Technologies, Almaty University of Power Engineering and Telecommunications name after Gumarbek Daukeev, Almaty, Kazakhstan
Email: olga.manank@gmail.com

1. INTRODUCTION

The value of data increases every day. Data is a key factor both in scientific research and in the field of 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 and big data processing technologies. This growing demand entails renewed interest in data privacy methods and processes. When a user needs to log in securely, he enters a password and his password is compared with the password stored in the system database. To be more precise, it is not the passwords themselves that are compared, since passwords are not stored in plain text, but their encrypted form is compared. This encrypted form will be the password hash. If the hashes match, then we can assume that the user has authenticated and can log in.

Hash functions are one of the first ways to ensure the protection of personal data through the user authentication process [1], [2]. However, the fact that most computer systems use a username and password for protection, often the same for different systems, as well as short passwords using only numbers and letters, often only numbers in the form of the user's date of birth, the problem of vulnerabilities of cryptographic systems remains open. [3], [4]. In addition, Hash functions themselves also have vulnerabilities that are exploited by an attacker. Among the common hash attacks, there are attacks by brute force or brute force and by dictionary. In contrast to such attacks, hash disclosure using rainbow tables significantly speeds up the hacking process [5], [6].

Rainbow tables are tables containing precomputed values of known hashes for a particular cryptosystem. If the cryptographic security of the system is violated by an unauthorized subject using a rainbow table, the attacker receives comprehensive information about the encryption scheme used [7]-[9]. That is, if for some reason access to the password hash tables is obtained, then with the help of rainbow tables, you can easily restore all the encrypted passwords that it contains. This can happen in case of password leaks, low password database security, the use of outdated hashing methods such as MD5 and SHA1 [10], or access to the password database by phishing those users who have access to the password database. For example, frequent traffic leaks in the IP telephony network occur due to the use of outdated hash functions [11], which do not provide strong protection and as a result lead to loss or theft of traffic, in addition to attacks on personal data [12]-[16]. The use of rainbow tables in practice is not limited to attacks, they can also be used as a mechanism for cryptanalysis of telecommunication systems or recovery of a forgotten password.

Cryptanalysis allows using an attack on the hashing mechanism to assess the degree of its resistance to disclosure. When an attacker steals a password hash, he can quickly determine whether the passwords have been over-salted or they have been hashed N times. Next, using the rainbow table, the attacker searches for the 100 most popular passwords. When coincidences are detected, the work goes in the opposite direction. After that, the hacker receives the decrypted password. But the time spent on this process will determine the stability of the system. Since the precomputed rainbow table contains hashes of all open characters in the password, it can take up large amounts of memory on the hard disk. Therefore, to implement cryptanalysis using rainbow tables, it is necessary to take into account the fact that the amount of allocated memory will depend on the time of password disclosure and the durability of the system [17]-[19].

To reveal the forgotten password, the administrator, having access to the password hash table and the rainbow table, can restore the encrypted password and provide the user with the plaintext of his password. In this case, the presence of a hash vulnerability makes it possible not to lose user data [20]-[22].

The security sector is developing very quickly and modern methods and procedures of attacks are used, but rainbow attacks remain a threat to organizations to this day. This is especially true for those organizations that do not use adequate password protection or save on security. It is recommended to use good knowledge of your cryptographic system as measures to increase the level of system security. In addition, use a modern Salting technique, which is based on the principle of adding an additional random value to each hashed password, which allows you to create new password hash values that will participate in authentication. To date, many modern password authentication systems include salt, which reduces the risk of successful attacks on rainbow tables [23].

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 [24]-[26].

The article proposes to investigate the resistance of the modern sha-256 hash function to the vulnerability of rainbow tables. From the analysis of publications, it is clear that previously outdated hashes were more often subjected to cryptanalysis, or modern hashes were often investigated by brute force and dictionary attacks less using rainbow tables. The relevance of the study lies in the fact that with the use of rainbow tables, the process of searching and comparing hashes becomes easier, since all the values in the rainbow table should already be calculated in advance. At the same time, it is not necessary to know the exact password, if the hashes match, then the user will authenticate. The exception is salted hashes, since for their disclosure it is necessary to know the salting algorithm and how many times they are hashed. But most often these practical settings are neglected by admins, which increases the risk of password hacking using rainbow tables.

The implementation of the rainbow attack on a modern hash function is written in Java and consists of two separate programs. Rainbow tables are files stored on a hard disk. Therefore, one program will generate rainbow tables based on user-defined parameters, and the second will process rainbow tables to provide a quick hash search.

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

In simple terms, a certain table is created in advance with matching chains in which the hash and password alternate. Moreover, in this table, all possible variants of passwords of a given length range and a given alphabet are sorted out (for example, passwords consisting of Latin capital letters from 1 to 5 characters long). An example of such a chain is shown below [23]:

$$\mathbf{aaaaaa} \underset{H}{\longrightarrow} \mathrm{281DAF40} \underset{R}{\longrightarrow} \mathrm{sgfnyd} \underset{H}{\longrightarrow} \mathrm{920ECF10} \underset{R}{\longrightarrow} \mathbf{kiebgt}, \quad (1)$$

here H is the hash function (eg SHA-1 or SHA-2) and R is the reduction function. In fact, this is just the generation of another key from the set of all possible ones in order to continue the chain. A rainbow table consists of many such chains, with different reduction functions applied at each iteration (i.e., R1, R2, R3…Rn where n is the length of the chain). Each chain starts with a random possible password, then is subjected to a hash function and a reduction function. A simplified rainbow table is shown in Figure 1.

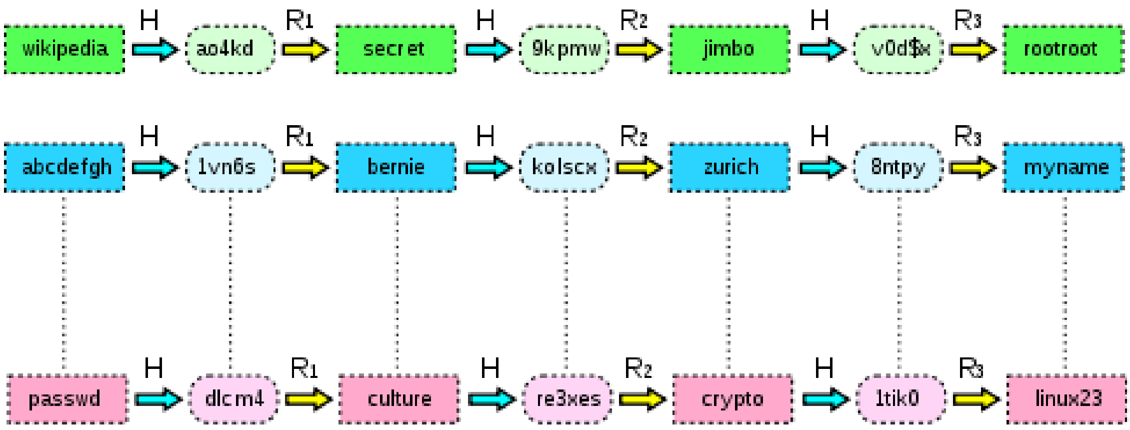


Figure 1. Diagram of a rainbow table with a chain length of three [27]

The main idea of the rainbow table is that intermediate passwords in the chain are discarded and only the first and last elements of the chains are written to the table. Creating tables takes time and memory, but they allow you to recover the original password very quickly (compared to conventional methods).

To recover the password, this hash value is reduced and looked up in the table. If no match is found, then the hash function and reduction function are applied again. This operation continues until a match is found. After finding a match, the chain containing it is restored to find the discarded value, which will be the desired password.

The construction of the rainbow table occurs in stages:

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 [28].

Next, we generate a certain number of chains, which are conveniently represented as a table with two columns (two-dimensional array), the first of which contains the initial key of the chain, and the second - the final one. After the chains are generated, you can already search for a key in them.

3. RESULTS AND DISCUSSION

To demonstrate the use of rainbow tables, a simple example can be given: suppose a password is generated consisting of two decimal digits in the range from 1 to 4. The SHA-256 hash algorithm is used to hash the password.

Password hash: 73475cb40a568e8da8a045ced110137e159f890ac4da883b6b17dc651b3a8049.

Suppose we know exactly the key length (2), the alphabet (1, 2, 3, 4) and the hashing algorithm. This means that the rainbow table will be small, and it will not be difficult to guess the password.

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. And this example was simplified as much as possible in order to simplify "manual" calculations.

The number of all possible passwords under these conditions is 16. 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 chains with a length of 3 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.

To calculate the SHA256 function, the Internet resource [29] was used.

Examples of some hash functions:
22 - 785f 3ec7 eb32 f30b 90cd 0fcf 3657 d388 b5ff 4297 f2f9 716f f66e 9b69 c05d dd09;
12 - 6b51 d431 df5d 7f14 1cbe cecc f79e df3d d861 c3b4 069f 0b11 661a 3eef acbb a918
Chain iterations involving functions R and H:
1) 32 --- e29c9c180c6279b0b02abd6a1801c7c04082cf486ec027aa13515e4f3884bb6b --- 34 --- 86e50149658661312a9e0b35558d84f6c6d3da797f552a9657fe0558ca40cdef --- 41 --- 3d914f9348c9cc0ff8a79716700b9fcd4d2f3e711608004eb8f138bcba7f14d9 --- 12;
2) 14 --- 8527a891e224136950ff32ca212b45bc93f69fbb801c3b1ebedac52775f99e61 --- 42 --- 73475cb40a568e8da8a045ced110137e159f890ac4da883b6b17dc651b3a8049 --- 22 --- 785f3ec7eb32f30b90cd0fcf3657d388b5ff4297f2f9716ff66e9b69c05ddd09 --- 21;
3) 24 --- c2356069e9d1e79ca924378153cfbbfb4d4416b1f99d41a2940bfdb66c5319db --- 23 --- 535fa30d7e25dd8a49f1536779734ec8286108d115da5045d77f3b4185d8f790 --- 31 --- eb1e33e8a81b697b75855af6bfcdbcbf7cbbde9f94962ceaec1ed8af21f5a50f --- 33;
4) 11 --- 4fc82b26aecb47d2868c4efbe3581732a3e7cbcc6c2efb32062c08170a05eeb8 --- 13 --- 3fdba35f04dc8c462986c992bcf875546257113072a909c162f7e470e581e278 --- 43 --- 44cb730c420480a0477b505ae68af508fb90f96cf0ec54c6ad16949dd427f13a --- 44;

Chains have been created. Further, we will assume that their first and last elements have been written to memory. That is:
32 --- 12;
14 --- 21;
24 --- 35;
11 --- 44.

After that, the attack begins, 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 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.

We start checking the last column of hashes:
3d914f9348c9cc0ff8a79716700b9fcd4d2f3e711608004eb8f138bcba7f14d9;
785f3ec7eb32f30b90cd0fcf3657d388b5ff4297f2f9716ff66e9b69c05ddd09;
eb1e33e8a81b697b75855af6bfcdbcbf7cbbde9f94962ceaec1ed8af21f5a50f;
44cb730c420480a0477b505ae68af508fb90f96cf0ec54c6ad16949dd427f13a.
None of them match our hash:
73475cb40a568e8da8a045ced110137e159f890ac4da883b6b17dc651b3a8049.
Therefore, we are looking in the penultimate column:
86e50149658661312a9e0b35558d84f6c6d3da797f552a9657fe0558ca40cdef
73475cb40a568e8da8a045ced110137e159f890ac4da883b6b17dc651b3a8049
535fa30d7e25dd8a49f1536779734ec8286108d115da5045d77f3b4185d8f790
3fdba35f04dc8c462986c992bcf875546257113072a909c162f7e470e581e278
The desired hash is found, Therefore, we restore the desired chain:
14 --- 8527a891e224136950ff32ca212b45bc93f69fbb801c3b1ebedac52775f99e61 --- 42 --- 73475cb40a568e8da8a045ced110137e159f890ac4da883b6b17dc651b3a8049 --- 22 --- 785f3ec7eb32f30b90cd0fcf3657d388b5ff4297f2f9716ff66e9b69c05ddd09 --- 21;

Answer: the required password is 42.

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 Java attack algorithm

The rainbow table generation algorithm (one program) and the hash search algorithm for this table (the second program) will be implemented in Java. The table will consist of all possible 3-character combinations of the 36-character input alphabet, consisting of lowercase Latin letters and numbers.

If you count how many possible combinations you get, then you get a total of 363 = 46656. The table will consist of 16 columns: 8 columns with passwords and 8 with their corresponding hashes.

The general algorithm is:

Generation of all possible (moreover, non-repeating) combinations of 3 characters of the input alphabet and their entry into a dynamic array;

Writing the first 5832 values from the array to the first column in Excel (these will be the initial elements of the chains) and parallel writing the corresponding hashes to the adjacent column.

Recording all other passwords in accordance with the reduction function (each column with passwords has its own reduction function).

Search for the required hash in the generated table.

The basis is the code for the SHA-256 hash function is shown in Figure 2 (hereinafter, parts of the code are shown without mentioning the connected libraries). The code is implemented using a ready-made library of methods in Java - Apache Common Codec [30].

```
class SHA256 extends Main{
    public static String shaApache(String st) {
        String sha256Hex = DigestUtils.sha256Hex(st);
        return sha256Hex;
    }
}
public class Main {
        public static void main(String[] args) throws IOException {
            String a = SHA256.shaApache("DFGBKJDBF");
            String b = SHA256.shaApache ("ab1");
            System.out.println(a);
            System.out.println(b);
        }
    }
```

Figure 2. Java description of the code for the SHA-256 hash function

The result of the generate hash function is shown in Figure 3.

```
in
"C:\Program Files\Java\jdk1.8.0_181\bin\java" ...
d3b874a5a0aee096c5623dd61fbad2009d50aada914719fb7963aa4ce043c6e1
ca5ba87c93d42f8a45c1e0f569bba8bac92c80f4ce6c864bd44d136572411b7e

Process finished with exit code 0
```

Figure 3. The result of the hash function

The next step is to generate all possible passwords. Please note that the generator will work first, and only then the received passwords will be distributed over the table.

Generator of a random non-repeating sequence of a string of 3 characters, and this sequence includes all possible combinations of three characters with an input alphabet of 36 characters (lowercase Latin letters and numbers) is shown in Figure 4:

```java
public class RandomGen extends Main {
    String AB2 = "0123456789abcdefghijklmnopqrstuvwxyz";
    SecureRandom rnd = new SecureRandom();
    public String randomString() {
        StringBuilder sb = new StringBuilder(3);
        for (int i = 0; i < 3; i++)
            sb.append(AB.charAt(rnd.nextInt(AB.length())));
        return sb.toString();
    }
     public static ArrayList<String> random_first(ArrayList<String> random_pass) {
         RandomGen rand = new RandomGen();
         String randd = rand.randomString();
         random_pass.add(randd);
         int count1 = 0, count2 = 0;
         for (int i = 0; i < 600000; i++) {
             String rand2 = rand.randomString();
             for (int j = random_pass.size(); j > 0; j--) {
                 if (!rand2.equals(random_pass.get(j - 1))) {
                     continue;
                 }
                 count1++;
                 if (count1 == 1) continue;
             }
             if (count1 == 0) {
                 random_pass.add(rand2);
                 count2++;
             }
            count1 = 0;
            if (count2 == 46656) {
                break;
            }
            // System.out.println((i+1)+"."+random_pass.get(i));
        }
        for (int i = 0; i < random_pass.size(); i++) {
            System.out.println((i + 1) + "." + random_pass.get(i));
        }
        return random_pass;
     }
```

Figure 4. Java description of the code for the SHA-256 hash function

Call the functions of this class in Main as shown in the Figure 5. The result of the generator is ahown in the Figure 6.

```java
public class Main {
    public static void main(String[] args) throws IOException {
        ArrayList<String> random_pass = new ArrayList<String>();
        RandomGen.random_first(random_pass);
    }
}
```

Figure 5. Java description of the code for the class main

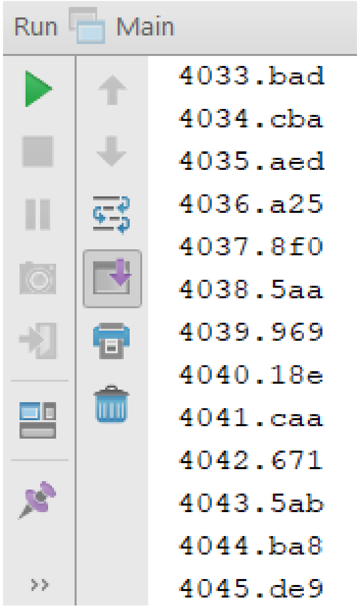


Figure 6. The result of the generator

A sequence of 46656 words with all checks for "non-repeatability" was generated in about 3 minutes. It is worth saying that in real conditions, rainbow tables contain not only millions, but billions of possible passwords. I assume that, for example, 4-character passwords with the same 36-character input alphabet (which is 364 = 1679616) will be under the same conditions for about an hour.

Next comes the function to write the first elements of the chains with their hashes in adjacent columns (Figure 7):

```
public class Excel extends Main {
        public static ArrayList<String> writeIntoExcel(String file, ArrayList<String> random_pass)
throws FileNotFoundException, IOException {
Workbook book = new XSSFWorkbook();
Sheet sheet = book.createSheet("Passwords");
Row row1 = sheet.createRow(0);
Cell cell1; Cell cell2;
for (int i = 0; i < 5832; i++) {
    row1 = sheet.createRow(i);
    cell1 = row1.createCell(0);
    cell2 = row1.createCell(1);
    cell1.setCellValue(random_pass.get(i));
    cell2.setCellValueSHA256.shaApache (random_pass.get(i)));
}
    book.write(new FileOutputStream(file));
        book.close();
        return random_pass;
    }
}
```

Figure 7. Java description of the code of function for writing the first elements of the chains with their hashes in adjacent columns

Call to Main.java for write the first elements of the chains with their hashes in adjacent columns in Excel file is shown in Figure 8.

```java
public class Main {
    public static void main(String[] args) throws IOException {
        ArrayList<String> random_pass = new ArrayList<String>();
        RandomGen.random_first(random_pass);

        Excel excel = new Excel();
        Excel.writeIntoExcel("wWWw.xlsx", random_pass);
    }
}
```

Figure 8. Java description of the code to generate Excel file

Result of generating Excel file is shown in Figure 9.

	A	B
1	0fx	a0c79cbda8f97f37be153752cc81915465b62aeb81b70a5d348f6d34bf82a279
2	vqs	d851758243fddc536fec42bbddcbefbdee9747d59a44efc636bdfd8cae38f208
3	6u2	fe5c48e671a6453721c612430c820cbc5c3a9a7488ca2db94ef0411f223897a9
4	ni3	8329666d4b3b005bdaa8986ddab89f4872f98123b6b8701705e49dfee4ad9360
5	y5g	d7d48a53c7e8264493aad509eb4580f8a4499de0bbd02bf0ed97140bbf25fb81
6	kpq	346502a5820109051cbd39ea1a66f2bb8f797df9f12a41cc72932848891f8218
7	sar	bdb8ec313eb09be46a2527fcae4af190c85cbf925d86353ccf517413f1c0a645
8	xqb	8ab4ae86c5e61444b0d59c22d6c838b753f972bfb2111e13e5a63c892157c002
9	21r	92cc28f7c904deaf0d6c6cea1a6fd9da19037e0dc24b56d2906f7f8ec6915461
10	h1n	f270e9c2d15ce51225a677d46205d31548b7490098bb44b4d5ead30d6f99f2a2
11	35v	578bcdfdd0a792b0e4085424d67e3f82646107ca27fb841e74cffae6cb81fc07
12	6em	e353bd48113b89c5ad46ca9d20e8e8b975df97b32d60b9f476da1e858838036d
13	apr	02b3dce98c61897231fdcf1a0594d6c2e159ff8a812386001c769706727efb3e
14	qpl	6c8b7e528dcef7b5fcedc28a4fa4bafe33c73d2b4fc84b2560ccdb402f818c0d
15	dyy	4fd4e8c6cc0ab10836674138e62833802befa1b2c0ce1819309fa9b472545ac8
16	fkd	829d19a083ac91f336272deaf9f9176439385c2b71328311557665beff66369f
17	b2c	5527f7356a87cec5d061767332baf6dd52a8d98ad469704be27e6eb403ab92e7
18	lj7	d37dc39322780b7458aa1648931462a17eeaf4e1daa6c51377262e8cfa578b16
19	hg2	ff023264bc3726ba2bc5e3351d861ff763f2fb1cd95525c51b170d2e0eb3372f
20	vwe	d1c1f90211549d3c07414234450edb0631ee934f4a485fce3eda87b8e06f01fd
21	i67	a645347a70687fec1f582b9174e4db1ea224fe9c84909ef6240e9e6fcf0ef6cc
22	ykq	1e9960d25935faa856398b72a50693d5cf30a3f5dcb0efaa6adad536ee9d0d1c
23	df0	d9ff4e514f14b91fb55e834d7f69d664b71f8df58ab5a0016793d3b13180a56a
24	6oe	722fee06a93b6cd53c05c95449613d90ac1996b0071b62d005cb9ec9af4cce2d
25	1eh	8b917d849cafdb2838fe7379bbb9fec483fff7074438c9ac5319f7d2a4b9cef9
26	9rq	7f1f160df53ae21f3158224f10f9e1a24df5ccedcecd0ea6e7e9dc9b8f5f7419
27	x56	868ba58d0081c758a322bec68c567037abe2f8222f2e95a7acd082f97ebd91ad
28	gah	ac1ab4a66c7f113bb66c250f581c5dd41f84c4fe4f0399e6bce609f6fcbd97a5
29	cay	6e5ed49aebca719385144d1bb100f804151022dafc12dac0e5feceb9f5d5acfe
30	t82	fd2bedd91720936165efd3c6815f05bc59cdfb7b296aa7be07af6f8cd2b670d1

Passwords

Figure 9. Recording result

For each subsequent column of passwords it is necessary to apply the reduction function. Let each of the reduction functions take an even bit of the hash as input, which increases each time (for example, the first reduction function takes the 0th bit of the password hash from the previous column as input, the second reduction function takes the 2nd, etc.). The code for these functions is also generate in the Excel.java file (Figure 10).

```java
    int column_pass=9, column_hash=10, byte_in_hash=0, get_cell=1;
    String random_pass2 = "";
    for (int k=0; k<6; k++){
    for (int i = 0; i < 5832; i++) {
        Cell cell5 = sheet.getRow(i).createCell(column_pass);
        Cell cell6 = sheet.getRow(i).createCell(column_hash);
        for (int j = 5832; j < random_pass.size(); j++) {
if ((SHA256.shaApache(random_pass.get(j)).toCharArray())[byte_in_hash] ==(sheet.getRow(i).getCell(get_cel
l).getStringCellValue().toCharArray())[byte_in_hash]) {
                cell5.setCellValue(random_pass.get(j));
                cell6.setCellValue(SHA256.shaApache (random_pass.get(j)));
                random_pass.remove(j);
                break;
            }
        }
    }
    column_pass = column_pass+9;
    column_hash = column_hash+9;
    byte_in_hash = byte_in_hash+2;
    get_cell = get_cell+9;
}

    for (int i = 5832; i < 5832*2; i++) {
       cell1 = sheet.getRow(i-5832).createCell(63);
       cell2 = sheet.getRow(i-5832).createCell(64);
        cell1.setCellValue(random_pass.get(i));
        cell2.setCellValue(SHA256.shaApache (random_pass.get(i)));
    }
```

Figure 10. Java description of the code of the reduction function

The result of calculating the reduction function for each password is shown in the Figure 11.

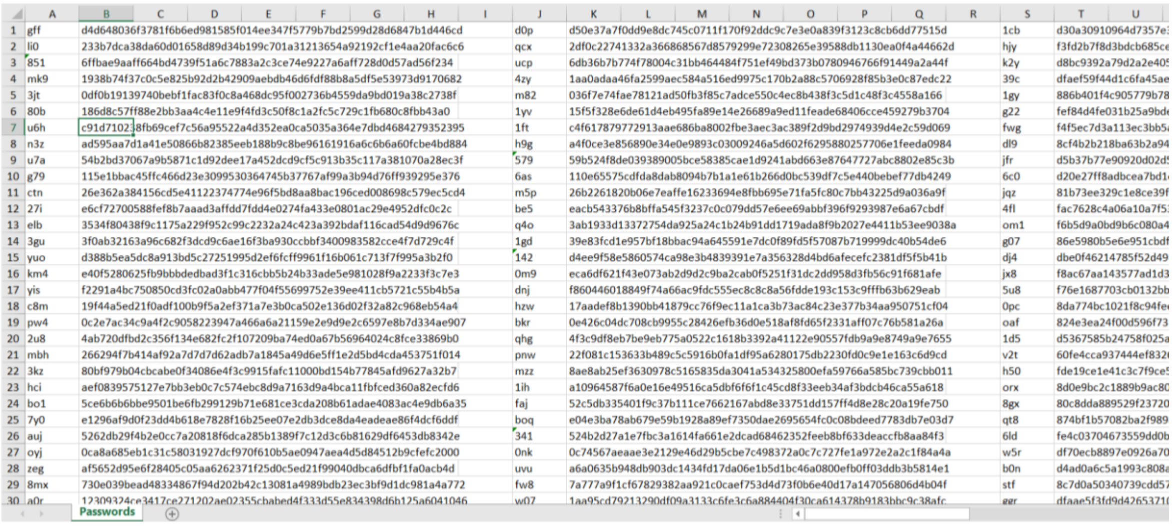


Figure 11. Recording result

The whole process - generating all possible combinations and writing these combinations with hashes to a table - took about 5 minutes. Again, the table only contains 46,000 passwords (and weighs in at around 2.3

MB). Real tables (again, for 8-character passwords with a 72-character input alphabet) can weigh about 20GB, and take hours to generate (depending on the speed of the algorithm and the power of the equipment). Moreover, most often a complete “database” is not one such table, but several. As a result, the code for generating the rainbow table is shown in Figures 12-15.

```
import java.io.IOException;
import java.util.ArrayList;

public class Main {
    public static void main(String[] args) throws IOException {
        ArrayList<String> random_pass = new ArrayList<String>();
        RandomGen.random_first(random_pass);

        Excel excel = new Excel();
        Excel.writeIntoExcel(random_pass);
    }
}
```

Figure 12. Java description of the code Main.java

```
import org.apache.commons.codec.digest.DigestUtils;
class SHA256 extends Main{
    public static String shaApache(String st) {
        String sha256Hex = DigestUtils.sha256Hex(st);
        return sha256Hex;
    }
}
```

Figure 13. Java description of the code SHA256.java

```
import java.security.SecureRandom;
import java.util.ArrayList;

public class RandomGen extends Main {
    String AB = "0123456789abcdefghijklmnopqrstuvwxyz";
    SecureRandom rnd = new SecureRandom();

    public String randomString() {
        StringBuilder sb = new StringBuilder(3);
        for (int i = 0; i < 3; i++)
            sb.append(AB.charAt(rnd.nextInt(AB.length())));
        return sb.toString();
    }
```

Figure 14. Java description of the code RandomGen.java

```
public static ArrayList<String> random_first(ArrayList<String> random_pass) {
    RandomGen rand = new RandomGen();
    String randd = rand.randomString();
    random_pass.add(randd);
    int count1 = 0, count2 = 0;
    for (int i = 0; i < 600000; i++) {
        String rand2 = rand.randomString();
        for (int j = random_pass.size(); j > 0; j--) {
            if (!rand2.equals(random_pass.get(j - 1))) {
                continue;
            }
            count1++;
            if (count1 == 1) continue;
        }
        if (count1 == 0) {
            random_pass.add(rand2);
            count2++;
        }
        count1 = 0;
        if (count2 == 46656) {
            break;
        }
    }
    for (int i = 0; i < random_pass.size(); i++) {
        System.out.println((i + 1) + "." + random_pass.get(i));
    }
    return random_pass;
}
```

Figure 15. Java description of the continuous of the code RandomGen.java

Java description of the generating Excel file is shown in Figure 16.

The password and hash table is ready. It remains only to search for it. Let's just scan each of the hash columns, starting with the last one, to see if the hash we entered is among these hashes. And the password corresponding to it is in the cell on the left.

To implement the search, we create a new project and copy the resulting Excel file with the generated rainbow table into it. The code for matching the hash we entered and the hashes from this table is shown in Figure 17. Result of the finding password is shown in Figure 18.

```
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.apache.poi.ss.usermodel.*;
public class Excel extends Main {
    public static ArrayList<String> writeIntoExcel(ArrayList<String> random_pass) throws FileNotFoundExcept
ion, IOException {
        Workbook book = new XSSFWorkbook();
        Sheet sheet = book.createSheet("Passwords");
        Row row1 = sheet.createRow(0);
        Cell cell1; Cell cell2;
        for (int i = 0; i < 5832; i++) {
            row1 = sheet.createRow(i);
            cell1 = row1.createCell(0);
            cell2 = row1.createCell(1);
            cell1.setCellValue(random_pass.get(i));
            cell2.setCellValue(SHA256.shaApache(random_pass.get(i)));
            //sheet.autoSizeColumn(1);
        }
        int column_pass=9, column_hash=10, byte_in_hash=0, get_cell=1;
        String random_pass2 = "";
        for (int k=0; k<6; k++){
            for (int i = 0; i < 5832; i++) {
                Cell cell5 = sheet.getRow(i).createCell(column_pass);
                Cell cell6 = sheet.getRow(i).createCell(column_hash);
                for (int j = 5832; j < random_pass.size(); j++) {
if ((SHA256.shaApache(random_pass.get(j)).toCharArray())[byte_in_hash] ==(sheet.getRow(i).getCell(get_cel
l).getStringCellValue().toCharArray())[byte_in_hash]) {
                        cell5.setCellValue(random_pass.get(j));
                        cell6.setCellValue(SHA256.shaApache(random_pass.get(j)));
                        //sheet.autoSizeColumn(column_hash);
                        random_pass.remove(j);
                        break;
                    }
                }
            }
            column_pass = column_pass+9;
            column_hash = column_hash+9;
            byte_in_hash = byte_in_hash+2;
            get_cell = get_cell+9;
        }
        for (int i = 5832; i < 5832*2; i++) {
            cell1 = sheet.getRow(i-5832).createCell(63);
            cell2 = sheet.getRow(i-5832).createCell(64);
            cell1.setCellValue(random_pass.get(i));
            cell2.setCellValue(SHA256.shaApache(random_pass.get(i)));
        }
        book.write(new FileOutputStream("wWWw.xlsx"));
        book.close();
        return random_pass;
    }
}
```

Figure 16. Java description of the generating Excel file

```java
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Scanner;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.apache.poi.ss.usermodel.*;

public class Main {

    public static void main(String args[]) throws IOException {
        System.out.print("Enter Hash: ");
        Scanner in = new Scanner(System.in);
        String hash1 = in.nextLine();

        System.out.print("Probably, you password : ");
        String pass = readFromExcel(hash1);

    }
    public static String readFromExcel(String hash) throws FileNotFoundException, IOException
    {
        XSSFWorkbook book = new XSSFWorkbook(new FileInputStream("wwww.xlsx"));
        XSSFSheet sheet = book.getSheet("Passwords");

        int column=64, count1=0;
        for (int i=0; i<8; i++)
        {
            for (int j=0; j<5832; j++)
            {
                Cell cell1 = sheet.getRow(j).getCell(column);
                Cell cell2 = sheet.getRow(j).getCell(column-1);
                if ((cell1.getStringCellValue().equals(hash)))
                {
                   System.out.println(cell2.getStringCellValue());
                   count1++;
                   break;
                }
            }
            if (count1 > 0){break;}
            column = column-9;
        }
       book.close();
        return hash;
    }
}
```

Figure 17. Java description of the code of finding password

```
"C:\Program Files\Java\jdk1.8.0_181\bin\java" ...
 Enter hash : 39e83fcd1e957bf18bbac94a645591e7dc0f89fd5f57087b719999dc40b54de6
 Password  : 1gd

Process finished with exit code 0
```

Figure 18. Search result

The search took only 3 seconds.

4. CONCLUSION

Considering all that was said earlier about the speed of generating rainbow tables and their sizes, as well as the fact that rainbow tables are effective against ordinary hashes and useless against salty ones, many consider them a bad and outdated hacking tool. However, we think that this is not entirely true and rainbow tables can still be useful. The paper proposes an implementation in the Java language, which allows them to be implemented in different network applications. If you use a powerful enough computer and a large amount of free hard disk space, you can pre-calculate hashes for passwords that contain more than 8 characters, which will speed up the process of decrypting passwords.

When conducting a preliminary analysis of the attacked system and the presence of vulnerabilities in hashing mechanisms, using rainbow tables will be more effective against dictionary search or brute force.

To attack the system, sometimes it is necessary to learn only one password, which also allows you to speed up the process of decrypting the password. This is especially important when using rainbow tables as a password recovery tool. In this case, security will be ensured by the work of the system administrator.

A cryptanalysis of the modern SHA-256 hash function showed that an attack using a rainbow table allows you to recover a password that has 3 characters in 3 seconds. This proves the fact that modern hashes also have vulnerabilities and need protection. In the future, you can investigate how the hash size affects the volume of the rainbow tab, as well as how the use of a character-letter password combination affects the speed of disclosure.

ACKNOWLEDGEMENTS

This research has been/was/is funded by the Science Committee of the Ministry of Education and Science of the Republic of Kazakhstan AP14871745 «Development of a method for improving the security of a telecommunications network based on IP-PBX Asterisk».

REFERENCES

[1] A. Habeeb, "Introduction to Secure Hash Algorithms". [Online]. Available: https://www.researchgate.net/publication/325581582_Introduction_to_Secure_Hash_Algorithms.doi:10.13140/RG.2.2.11090.25288.
[2] T. Bhorkar, "A Survey of Password Attacks and Safe Hashing Algorithms," *Int. Res. J.of Eng. and Tech.,* vol. 4, no. 12, pp.1554-1556, 2017.
[3] D. Smith-Tone1, R. Perlner, "Rainbow Band Separation is better than we thought*," Cryptology ePr. Arch.*, 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," *in 17th IEEE Symp. on Field Programmable Custom Comp. Machines*, 2009, pp.287-290, doi:10.1109/fccm.2009.48
[5] P. Patel, P. Goswami, A. Mishra, S. Khan, A. Choudhary, "Brute force, dictionary and rainbow table attack on hashed passwords," *Int.J. of Creative Res. Thoughts*, vol. 9, no. 4, Apr. 2021, pp.1899-1905. [Online]. Available: https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf.
[6] D. M. A. Cortez, A. M. Sison, R. P. Medina, "Cryptanalysis of the Modified SHA256", *Proceedings of the 2020 4th High Performance Computing and Cluster Technologies Conference & 2020 3rd International Conference on Big Data and Artificial Intelligence*. doi:10.1145/3409501.3409513.
[7] 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.
[8] J. Blakstad, R. Walso Nergard, M. G. Jaatun, "All in a day's work: Password cracking for the rest of us," *in Norwegian Symp. on Inf. Sec.,* pp.69-83, 2009.
[9] L.Zhang, C. Tan, F. Yu, "An Improved Rainbow Table Attack for Long Passwords," *Procedia Comp. Sci.,* vol. 107, Apr. 2017. pp. 47-52, doi:10.1016/j.procs.2017.03.054.
[10] X. Wang, Y. L. Yin, H. Yu, "Finding Collisions in the Full SHA-1," *in 25th Annual Int. Cryptology Conf.,* Santa Barbara, California, USA, Aug. 14-18, pp. 17-36, 2005.
[11] S. Verma, R. Choubey, R. Soni, "An Efficient Developed New Symmetric Key Cryptography Algorithm for Information Security," *Int. J. of Emerging Tech.and Adv. Eng.,* vol. 2, no. 7, pp.18-21, Jul. 2012.
[12] G. Avoine, A. Bourgeois, X. Carpent. (2015). Analysis of Rainbow Tables with Fingerprints. [Online]. Available: doi:10.1007/978-3-319-19962-7_21.
[13] 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.
[14] 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," *in Int. Conf. on Inf. Sci. and Commun. Tech.,* Oct. 2020, Art. no. 20486746, doi: 10.1109/ICISCT50599.2020.9351405.
[15] 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," *J.of Theoretical and App.Inf. Tech.,* vol. 99, no.14, pp. 3617-3630, 2021.

[16] 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,” *in 11th Int.Forum on Strategic Tech.*, 2017, pp. 510-515, Art. no. 7884168.

[17] A. Joshi, M. Wazid, R.H. Goudar, “An Efficient Cryptographic Scheme for Text Message Protection Against Brute Force and Cryptanalytic Attacks,” *Procedia Comp. Sci.*, vol. 48, pp. 360-366, 2015.

[18] S. Tayal, N. Gupta, P. Gupta, D. Goyal, M. Goyal, “A Review paper on Network Security and Cryptography,” *Adv. in Computat. Sci.and Tech.*, vol.10, no. 5, pp. 763-770, 2017.

[19] A. K. Kendhe, A. Himani, "A Survey Report on Various Cryptanalysis Techniques," International Journal of Soft Computing and Engineering (IJSCE), vol. 3, no. 2, 2013, pp. 287-293.

[20] 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.

[21] A. M. Qadir, N. Varol, “A Review Paper on Cryptography,” *in 7th Int. Symp. on Digit. Forensics and Sec.*, Barcelos, Portugal, 2019, pp. 1-6, doi: 10.1109/ISDFS.2019.8757514.

[22] Th. Velmurugan, S. Karthiga, “Security based Approach of SHA 384 and SHA 512 Algorithms in Cloud Environment,” *J. of Comp. Sci.*, vol. 16, no. 10, 2020, pp. 1439-1450, doi: 10.3844/jcssp.2020.1439.1450.

[23] S. Park, K. Kim, "An Approach to Defense Dictionary Attack with Message Digest Using Image Salt," Proceedings of the 13th International Conference on Ubiquitous Information Management and Communication (IMCOM), 2019, pp. 769-777.

[24] P. Semwal, M. K. Sharma, “Comparative study of different cryptographic algorithms for data security in cloud computing,” *in IEEE 3rd Int. Conf. on Adv. in Comp., Commun. & Automat.*, pp.1-7, Sept. 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8344738/ (ascessed 12.09.2022).

[25] M. Ansar, I. A. Shokat, M. Fatima, K. Nazir, “Security of Information in Cloud Computing: A Systematic Review,” *American Scientific Res. J. for Eng., Tech., and Sci.*, vol. 48, no.1, pp. 90-103, 2018. [Online]. Available: http://www.asrjetsjou (ascessed 12.09.2022).

[26] C. Yang, Q. Huang, Z. Li, K. Liu, F. Hu, “Big Data and cloud computing: innovation opportunities and challenges,” *Int.J. of Digit. Earth*, vol. 10, no. 1, pp.13-53, 2017.

[27] Rainbow table. [Online]. Available: https://ru.wikipedia.org/wiki/Rainbow table (ascessed 12.09.2022).

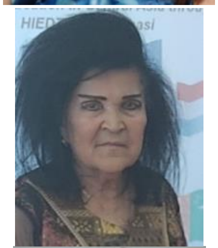
[28] Security online community. Antichat: FAQ - Rainbow Tables. [Online]. Available: https://forum.antichat.ru/threads/37964 /(ascessed 12.09.2022).

[29] SHA-256 hash calculator. [Online]. Available: https://www.xorbin.com/tools/sha256-hash-calculator (ascessed 12.09.2022).

[30] SHA-256 usage example in Java. [Online]. Available: https://devcolibri.com/sha256-пример-использования-в-java / (ascessed 12.09.2022).

BIOGRAPHIES OF AUTHORS

Olga Alexandrovna Manankova is a PhD student at the Almaty University of Power Engineering and Telecommunications after Gumarbek Daukeyev (AUPET). After receiving a master's degree in 2010, she began working at AUPET as a teacher. During this time, under her supervision, more than 25 bachelors graduated, students became winners of the Republican competitions of research and development in the field of IT. Currently works at AUPET as a senior lecturer and is engaged in research in the field of radio engineering, electronics and telecommunications in accordance with the topic of the Ph.D. thesis "Research and creation of information security transmitted over an open communication channel using PBX Asterisk". She can be contacted at email: olga.manank@gmail.com.



Mubarak Zakhidovna Yakubova Doctor of Technical Sciences, Academician, Professor of the Department of Information Technology. She was the first specialist in the USSR who worked in the field of automation of information systems and libraries. During her career, she published more than 250 scientific papers, 50 articles, 3 monographs, 15 textbooks in Kazakh, Russian and English. In 2001, for selfless work and contribution to the education and upbringing of students. Now he is actively working with doctoral students in the field of optimizing multiservice networks. She can be contacted at email: m.yakubova@aues.kz.

Alimjan Sergeevich Baikenov is a Candidate of Engineering Sciences, Professor at the Department of Telecommunications and Innovation Technologies. He had overall experience of 41 years. He is familiar with software’s like MATLAB, Multisim, Packet Tracer, EVE NG, Wireshark. 2019 received the Title "Kurmetti baylanysshy", awarded in 2018 Medal "Bilim beru salasyn uzdigi". Now he is actively working with doctoral students in the field of Multi-channel communication, Teletraffic theory, system modeling, wireless networks and systems (5G), fiber optic systems, Internet of things (IoT), artificial intelligence (AI), Information security in 5G and IoT. Expert IAAR NU "Independent Agency for Accreditation and Rating". He can be contacted at email: a.baikenov@aues.kz