Informative Overview of Rainbow Hashes

Víctor Hornedo Martínez
Computer Science
Jeffrey Duffany, Ph.D.
Electrical and Computer Engineering and Computer Science
Polytechnic University of Puerto Rico

***Abstract* —** This article gives an overview for rainbow tables and the results of testing rainbow tables according to the length of the chosen chain. The article presents a password cracking process that contains its own algorithms for reduction functions, changes the length of the chain and generates tables accordingly. These are measured to see the effectivity of the password search in detail. Within the executed tests it was noticed that there is a dependence of rainbow tables size in relation to the password length, the affection of the hash search by the size of the chosen chain and their links to collisions. After completing the testing with different passwords and tables the cause of this arises from the principle of using the reduction function. These results objectively describe the pros and cons of using rainbow tables and finally the article ends talking about what are some effective use cases for this password cracking method.

INTRODUCTION

Theoretically all passwords are "crackable" Breaking any encryption system can be done with unlimited time and unlimited computing power, both of which do not exist. Anything less than that unlimited power and time will require chance and good investigative skills. Several methods to break encryption include dictionary attacks, brute-force attacks, and rainbow tables. Knowing that recovering the password requires time, computing power and most of all luck for a dictionary or brute-force attack to find a valid password. "Strong passwords increase the likelihood, if not guarantee that it would be harder for attackers to break the encryption of it ". [1]

Several styles to break encryption include wordbook attacks, brute-force attacks, and rainbow tables. A dictionary attack tries variations of words in the wordbooks. The speed at which depends upon the computing power of the system being used. Millions of words can be tried each second using a suitable computer system for password breaking. However, dictionary attacks should not be overlooked because of not knowing the password. Although using the highest-grade encryption is easy, quick and effective, a flaw remains with the user in choosing a strong password. The password can make a seemingly impossible to crack file easily done in minutes.

On the other hand, brute-force attacks are similar to dictionary attacks in that guessing is the key method. Brute-force attacks try variations of characters of various lengths that could be the password. The amount of time and computing power required depends on the complexity and length of the password. Short passwords can be recovered quickly, but longer passwords increase the time exponentially according to password length and complexity. "Dictionary attacks are generally the chosen method over brute-force attacks ". [1] None of these methods are guaranteed to work all the time or even some of the time depending on the encrypted file properties.

The other method being rainbow tables that are the focal point of this article. These are tables of reversed hashes used to crack password hashes. Computer systems requiring passwords typically store the passwords as a hash value of the user's password. When a computer user enters a password, the system hashes the password and compares it to the stored hash. If the hashes match, the user is given access. Rainbow tables use precomputed hashes to recover the pre-hashed password.

Rainbow tables rely on a clever time/memory tradeoff. This technique was researched by Martin Hellman and improved upon by Philippe Oechslin. This technique contains a long chain of password hashes known as plaintext/ciphertext pairs these are connected. In which thousands or millions of pairs may be connected into one chain called a rainbow chain and many chains may be formed, connected via a reduction function, which takes a hash and converts it into another possible variation for a password. At the end, everything in the chain may be removed, except the first and last entry. These chains may be rebuilt as needed, reconstituting all intermediate entries. “This saves a large amount of storage, in exchange for some time and CPU cycles”. [2] This in fact is why rainbow hashes are known as the most efficient method when dealing with brute forcing hashes. All this comes at a cost, which is the sizes of the hash tables that are used.

THE RAINBOW HASH OPERATION

Passwords are hashed using encryption rather than being saved as plain text in a computer system. It is impossible to decrypt a hash function because it is a one-way function. Once the user inputs a password, it is transformed to a hash value and contrasted to the hash value that has previously been recorded.

The rainbow table is a dataset that can be used to crack the password hash and get authorization. It's a pre-computed database of unencrypted passwords and associated hash values which could be used to figure out just what unencrypted password generates a specific hash. Because multiple texts can yield the same hash, it isn't necessary to know what the initial password was as long as the hash is still the same.

Password Decryption

The rainbow table operates by fast and accurately performing cryptanalysis. Unlike with brute force, that calculates the hash function of each string they have, calculates their hash value, and then compares it to that in the system at each step. The hashed text is first verified to see if it consists in the database. If that's the case, go back to the beginning of the chain and hash until you find a match. The process ends when the match is found, and the validation is broken. The stages are illustrated in Figure 1 below:

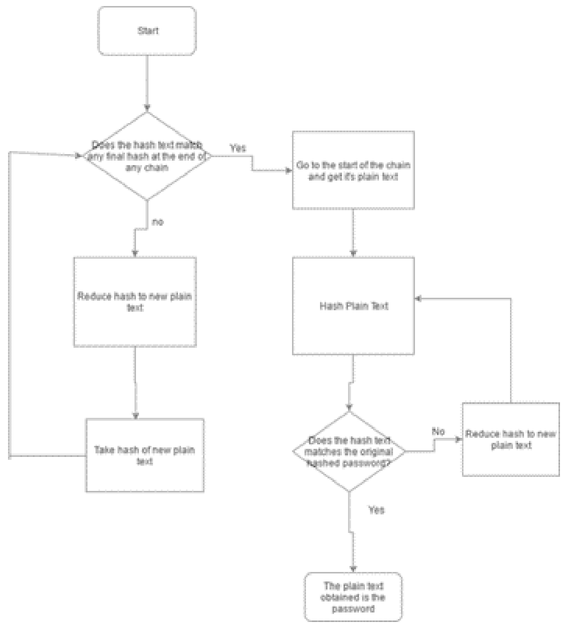

Start
Does the hash text match any final hash at the end of any chain
Yes
no
Go to the start of the chain and get it's plain text
Reduce hash to new plain text
Hash Plain Text
Take hash of new plain text
Does the hash text matches the original hashed password?
No
Reduce hash to new plain text
Yes
The plain text obtained is the password

Figure 1
Rainbow Hash Flowchart

By generating hashes of the large collection of accessible strings, a rainbow table attack removes the requirement for this. In this process, there are two basic steps:

- **Putting Together a Table:** A string's hash is extracted and then decreased to form a new string, which will then be decreased again and again. Let's make a table with the most used password, by using the MD5 hash function, see Table 1 below to understand reasoning, on the first eight characters. We begin by running the string through the md5 hash function. Only the first eight letters are used to lower the hash. This process is continued until the output chain has enough hashes. When there are enough chains collected, they can be placed on a table. Some hash algorithms used in Rainbow Tables are given below.
- **Reduction function:** Rainbow Table files are very large, though. So that the required storage space doesn’t get out of hand, rainbow tables use a reduction function below (Figure 2) that changes the hash value into plaintext. Important: “The reduction function doesn’t

reverse the hash value; it outputs a completely new one"[2].

Table 1
List of Hash Algorithms

Hash Algorithm	Hash Length	Plaintext Length
MD5	16	0-7
SHA1	20	0-20
SHA256	32	0-20
LM	8	0-7
NTLM		015

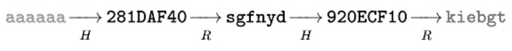


Figure 2
Reduction Function

A new hash value is generated from this text. In a rainbow table, this takes place not only one time, but many times, resulting in a chain. In the final table, however, only the first password and the last hash value of a chain appear.

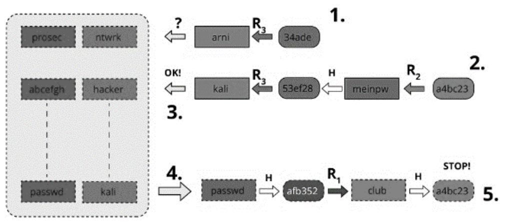


Figure 3
Reduction process

A brute force hash cracker generates all possible plaintexts and computes the corresponding hashes on the fly, then compares the hashes with the hash to be cracked. Once a match is found, the plaintext is found, see Figure 3. If all possible plaintexts are tested and no match is found, the plaintext is not found. With this type of hash cracking, all intermediate computation results are discarded.

METHODOLOGY

The storage techniques to be tested are: MD5 Hashing and SHA-256 Hashing. Although not proven to be secure, a commonly used superset of password hashing is hash chaining. Following the National Institute of Standards and Technology (NIST) guidelines on password strength, "both a weak and robust password will be passed through the stated techniques" [3]. Then, reversal of each of the resulting strings will be attempted using online and offline rainbow tables. The data recorded will be the time taken to reverse the hash or whether the attack was successful.

Creating a Hash to Crack

Creating a hash is very simple to do. You can create a hash of a file or of a string of text. In the below example we will create two hashes using the words P@55w0rD and thisismypassword (Table 2 & Table 3). Next, we will create a file containing our hashes so we can put them into a cracking program. Now that we have our hashes let's try to crack them! Methods of password cracking use for this experiment are Ophcrack for the offline cracking using rainbow tables and Crackstation for the online cracking component of this comparison.

Table 2
MD5 Hashes

P@55w0rD	B884DBCC2FEDE312066A9B7609A2E3C9
thisismypassword	31435008693CE6976F45DEDC5532E2C1

Table 3
SHA256 Hashes

P@55w0rD	7F0897E8D62D3E3641EAFC270D311CBF777E67B9DF608571C93056D5AACF3189
thisismypassword	1DA9133AB9DBD11D2937EC8D312E1E2569857059E73CC72DF92E670928983AB5

In the above example, while the password "thisismypassword" is not unique it serves the purpose of this experiment which is testing a raw simple and complex password using two different hashing algorithms and compare the results. As a plus in future testing we can try salting it such as "thisismypassword" + "s41Ty" to make it more unique. It is highly unlikely to be a password that would show up in a rainbow table already. This means that hackers would need to do all the costly the computation themselves. Adding a salt to the hashing process is a great way to force the hash to be more unique, complex, and increase its security without giving extra requirements to a user. The salt is usually stored with the password string in the database. Adding salt can help to mitigate password attacks, like rainbow tables, because it will be encrypting the user's password with a random string that wouldn't be naturally included in any rainbow table.

You can also add pepper to extra secure your data from this sort of attack. The difference between salt and pepper is that "pepper is a site-wide static value that is kept a secret and not stored in the database"[3]. It is oftentimes hard coded into the application's actual source code. Its main added use is that since it isn't store in the database, if there was a large-scale compromise of the database, the application's password table containing the hashes wouldn't be able to be brute forced as the pepper was still a secret.

Cracking Passwords with Crackstation

When you have a hash, it is always best to first try to use a rainbow table to crack it. This is because it can take an astronomical amount of time to brute force crack the hash. To save time we will use a well-known website called Crackstation.net. This website has billions of hashes which could save us centuries of time. So, let's enter our hashes into the box and see what we get. As you can see, it successfully found passwords for both the hashes! Now that we have this our work is done!

The results for both SHA256 and MD5 hashes using the online method of cracking we can start to see its drawbacks. While not needing the user to download tables to run, the process comes with the drawback of slower times. The reason for this is "because it searches for all available tables it can find while proceeding with the validation of the hash existence" [4]. If it does not exist, it will make a new entry of the hash in its database. After this step it keeps going threw the process until it can find the result. The result of the two passwords is displayed in the table below.

Table 4
MD5 Password Crack Time Using Crackstation

P@55w0rD	Not found
thisismypassword	0.7 seconds

Table 5
SHA256 Password Crack Time Crackstation

P@55w0rD	Not found
thisismypassword	0.8 seconds

As we can see the hash for "thisismypassword" both in the MD5 and SHA256 variants were the easy passwords variable used for the experiment. Knowing this we can also see both variants being cracked fast because the hashes were found in public tables, see Table 4 & Table 5. "As for the more complex passwords both variants could not be found in any public hash table" [4]. This can be because how this password was constructed. Meaning it contains uppercase, lowercase, numbers and symbols. This ensures the password is more unique and harder to crack.

Cracking Passwords with Ophcrack

Ophcrcack is based on rainbow tables and a popular Windows password cracker freeware. It can crack password within minutes but can take time also depending on the password strength, for example "1234567" will take less time than "wuntsg256". The free version of Ophcrack comes with a table which can break password not more than 14 characters using only alpha numeric characters. Ophcrack uses brute force method to crack password. Ophcrack is an extremely fast password cracker because it uses rainbow tables. Brute-force cracking tools typically try thousands of combinations of letters, numbers and special characters each second, but cracking a password by attempting every conceivable combination can take hours or days.

Rainbow tables pre-computes the hashes used by passwords, allowing for a speedy password lookup by comparing the hashes it has, instead of computing them from scratch. Thinking of it another way, someone else has already generated the password hashes for millions of potential passwords using the same algorithm as Windows XP and Vista. Ophcrack simply loads the megabytes of hashes it already has and compares the password hash in Windows against its giant database. When it finds a match, Ophcrack reveals the password in plain text.

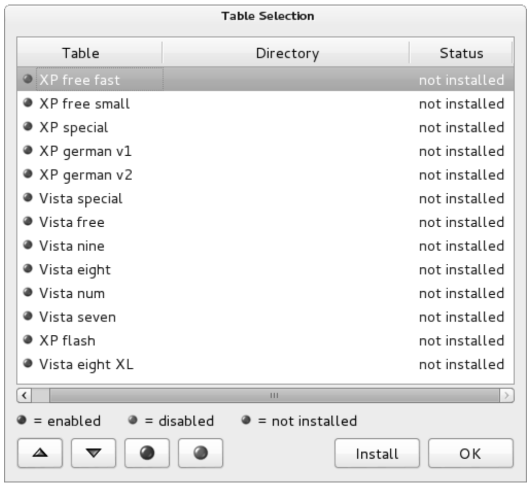


Table	Directory	Status
● XP free fast		not installed
● XP free small		not installed
● XP special		not installed
● XP german v1		not installed
● XP german v2		not installed
● Vista special		not installed
● Vista free		not installed
● Vista nine		not installed
● Vista eight		not installed
● Vista num		not installed
● Vista seven		not installed
● XP flash		not installed
● Vista eight XL		not installed

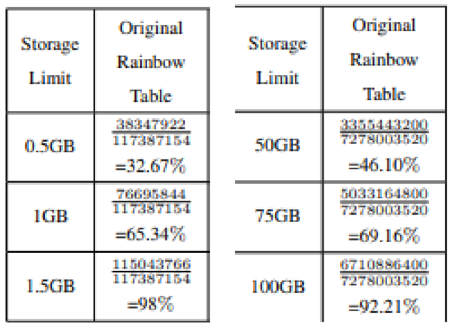
● = enabled ● = disabled ● = not installed

Figure 4
Ophcrack Rainbow Table List

Rainbow tables are great but there are plenty of times where you will be unable to find the password for a hash. When this occurs, you will need to brute force a password using either pure brute or using hash tables. Brute force is rarely feasible for passwords of 10 or more characters. So, let's see if we can crack the hashes, we created above using a table offered in Figure 4.

Note that when thinking of the success rate for the tables it was calculated using the cases where the passwords have a determined characters in length. To do so, the storage requirement had a fixed value and compute the achievable success rate. The equation used to find the success rate was dividing the result of all possible combinations inside the intended character length and the number of hashes stored inside each table. The examples of Figure 5 shows the success rate when the storage is capped at a certain value for different password lengths. While looking through all the success rates we can notice that the bigger the table the more plausible it is to find the correct hash and its accompanying hash value.

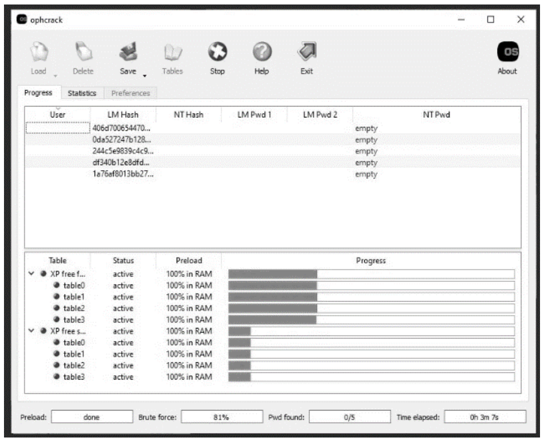
The password recovery success rate computed ignores collisions. Because the collision probability increases with the size of a rainbow table, ignoring collisions is reasonable only for very small rainbow tables. We compute a more realistic password recovery success rate based on collisions with the distinct plaintext-hash pairs that are generated.



Storage Limit	Original Rainbow Table
0.5GB	$\frac{38347922}{117387154}$ =32.67%
1GB	$\frac{76695844}{117387154}$ =65.34%
1.5GB	$\frac{115043766}{117387154}$ =98%

Storage Limit	Original Rainbow Table
50GB	$\frac{3355443200}{7278003520}$ =46.10%
75GB	$\frac{5033164800}{7278003520}$ =69.16%
100GB	$\frac{6710886400}{7278003520}$ =92.21%

Figure 5
Examples of Success Rate for Character Alpha-Numeric Passwords



User	LM Hash	NT Hash	LM Pwd 1	LM Pwd 2	NT Pwd
	406d700654470...				empty
	0da527247b128...				empty
	244c5e9839c4c9...				empty
	df340b12e8dfd...				empty
	1a76af8013bb27...				empty

Table	Status	Preload	Progress
XP free f...	active	100% in RAM	
table0	active	100% in RAM	
table1	active	100% in RAM	
table2	active	100% in RAM	
table3	active	100% in RAM	
XP free s...	active	100% in RAM	
table0	active	100% in RAM	
table1	active	100% in RAM	
table2	active	100% in RAM	
table3	active	100% in RAM	

Preload: done Brute force: 81% Pwd found: 0/5 Time elapsed: 0h 3m 7s

Figure 6
Rainbow Table Hash Lookup

Table 6
MD5 Password Crack Time Using Ophcrack

P@55w0rD	414 seconds
thisismypassword	34 seconds

Table 7
SHA256 Password Crack Time Using Ophcrack

P@55w0rD	419 seconds
thisismypassword	39 seconds

As we can see from the Table 6 & Table 7, it takes considerably longer to crack these hashes. It takes a bit more time than the online method because we don’t have the luxury of just comparing the testing hash to public hashes available. Also, in this process we have larger pre-computed tables to expedite the process, see Figure 6. It takes a long time to generate these massive rainbow tables, but once they're out there, every attacking computer can leverage those tables to make their attacks on hashed passwords that much more potent. The smallest rainbow table available is the basic alphanumeric one, and even it is 388 megabytes. That's the default table you get with the Ophcrack bootable ISO. Even that small-ish table is remarkably effective.

It wasn’t expected that this rainbow table would not work on the passwords with non-alphanumeric characters (%&^$#@!*) because the table doesn't contain those characters. So, to the next one we go. The table that found the result for both variants had size of 207 gigabiytes in total. The size of the table is tied to how long the plain text pasword is and what combinations can exist. For example, if the table is only alphabetic characters then the size would be smaller than one that holds full alphanumeric or even non alphanumeric charcter like the one we used. It also accounts for all the possible plain text values that may exist.

Time Memory Trade-Off

A time-memory tradeoff is basically when you accept a longer runtime in favor of fewer memory requirements or the other way around. A table, on the other hand, in which billions of passwords are presented together with their hash values, takes up an enormous amount of storage space, but can very quickly run decryptions. Rainbow tables represent a compromise for both. In principle, they also perform real-time calculations, but to a lesser extent, and so save a lot of storage space compared to complete tables. Rainbow tables will fit in between. When the table is built, you choose a parameter t called the "average chain length". The table size will be proportional to N/t: it is reduced by a factor of t, compared to the precomputed table. On the other hand, each attack will imply a computational effort proportional to t^2 hash computations, and t lookups. Depending on the operational conditions.

Lastly the experiment has demonstrated that the time-memory trade-off allows anybody owning a modern personal computer to break cryptographic systems which were believed to be secure when implemented years ago and which are still in use today. This goes to demonstrate the importance of phasing out old cryptographic systems when better systems exist to replace them. Since memory has the same importance as processing speed for this type of attack, typical workstations benefit doubly from the progress of technology.

Advantages and Disadvantages of Rainbow Table

For a short period of time, rainbow tables were an effective way of cracking passwords. With a big enough table, the chances of finding some matches were quite likely. Mainly because of the hash algorithm used and because of common hashes that circulated the web. However, as the popularity of less secure hashing algorithms fell, and as password salting became a more common practice, rainbow tables have fallen out of common use.

Advantages

Rainbow Tables have the advantage of most data being pre-computed, resulting finding faster the target hash. Therefore, the whole process is just a simple search and compare operation on the table unlike the Brute Force Attacks. Another crucial advantage using of Rainbow Tables is the ability of authentication without serious obstacles. This occurs because the exact password string does not have to be known or estimated. If a hash match occurs, then it is enough for the attack to be performed.

Disadvantages

One main and most common problem when dealing with Rainbow Table is the fact that these tables need to be stored in a huge memory partition (Hard Disks). Sometimes terabytes are needed, resulting in an increased maintenance cost. Another disadvantage of rainbow tables is the fact that if the target hash that is looking to be broken is not in the table used, then he will be unable to find the resultant password in a short period of time. This gives the limitation of the table in use.

Lastly rainbow table attacks, can be thwarted using salt a technique that forces the hash dictionary to be recomputed for each password, making precomputation infeasible, provided that the number of possible salt values is large enough. Salts are a random token usually used only once that is combined with the password before hashing. It artificially increases the length of a password in the rainbow table, so to crack a 4-character password with a 4-character salt, you'd need to generate an 8-character rainbow table.

Protective Measures Against Rainbow Table

Don't use MD5 or SHA1 in your password hashing function. MD5 and SHA1 are outdated password hashing algorithms. Consider using more modern hashing methods and a cryptographic "Salt" in your password hashing routine. Countermeasures for attacks with rainbow tables are the use of modern key derivation functions. These are special hash functions that should be used for hashing passwords.

Primarily for protection against rainbow tables is the use of a so-called salt. A salt is a random string that is combined with the entered password the first time it is hashed and then saved together with the password hash and the username. If the password is re-entered, the salt is added to the input each time and the correct hash for authentication can only be generated through this combination. The salt doesn't have to be kept secret. To be able to crack such a password successfully with a rainbow table, the tables would have to be precalculated for each individual possible salt value. If the salt is sufficiently complex, this is not possible because the computing effort and memory requirements are too great to realistically calculate these tables.

"Salts completely thwart precomputed tables, including rainbow tables". [5] Building a precomputed table for *N* passwords has cost *N*, building a rainbow table for the same *N* passwords has even higher cost. This is worth the effort only if the table can be used at least twice, to attack two distinct hash values; a one-shot table is not competitive with exhaustive search. But the point of salts is that there is not *one* function; in fact, there is a big family of functions, and the salt value tells you which one is actually used. A table built for a specific salt has absolutely no value in breaking a hash value for any other salt.

Another measure that every user has in their own hands to counteract a rainbow table attack is the choice of a sufficiently complex password. By lengthening the passwords used and using as many different characters as possible, the complexity of the password increases so quickly that it is no longer possible to calculate rainbow tables or crack using brute force. A recommendation for this is a password length of at least 12 characters and the use of upper- and lower-case letters, numbers, and special characters. The password should also be randomly combined from these character sets. Since such secure passwords are difficult to remember, a password manager should always be used.

Are Rainbow Table Attacks Still A Threat?

Some security experts argue that rainbow tables have been rendered obsolete by modern password cracking methodologies. Instead, most attackers now use the more advanced Graphics Processor Unit (GPU) based password cracking methods.

A moderately-sized GPU farm can easily recreate a rainbow table within a few seconds. This means that encoding those passwords into a rainbow table would not make that much sense. Moreover, "most passwords are salted anyway, meaning we would need rainbow tables for each salt value, and for larger salts, this is entirely impractical".[5] Bitcoin and other cryptocurrency miners have been tapping GPU technology to calculate hashes for bitcoin farming. There are existing tools that can leverage GPU technology to decrypt password hashes potentially. Nonetheless, rainbow tables may not be the biggest threat to organizations today. Still, they are certainly a threat and should be considered and accounted for as part of an overall security strategy.

CONCLUSION

That was a lot of technical info to take in. But I hope this article has provided a better understanding of what rainbow tables are, how they work, and what you can do to secure an organization's stored password hashes against rainbow table attacks. Just remember that password security is a continually changing practice. There's no perfect way to prevent or thwart every type of password cracking attempt. Unlike other techniques, huge storage is needed for Rainbow Table Attacks and sadly the decreasing price per Mbyte for storage solutions nowadays doesn't help our safety. To avoid being a victim of Rainbow Table Attack, it is strongly advised to perform frequent password changes. Password security it's all about just following recommended best practices and trying to at least keep up with cybercriminals (if not staying one step ahead).

For a short period of time, rainbow tables were an effective way of cracking passwords. With a big enough table, the chances of finding some matches were mainly because of the hash algorithm used and because of common hashes that circulated the web. However, as the popularity of less secure hashing algorithms fell, and as password salting became a more common practice, rainbow tables have fallen out of common use.

REFERENCES

[1] Shavers, B., & Bair, J. "Cryptography and Encryption. In Hiding Behind the Keyboard", (2016) (pp. 133–151). Elsevier.

[2] Information Security Stack Exchange. [Online]. https://security.stackexchange.com/questions/92865.

[3] Rainbow table attacks and cryptanalytic defenses. (2022, February 26). [Online]. https://www.esecurityplanet.com/threats/rainbow-table-attack/

[4] CrackStation. (2019, June 5). Secure Salted Password Hashing - How to do it Properly. [Online]. https://crackstation.net/hashing-security.htm

[5] International Journal on Advances in Software, vol. 4 no 3 & 4, year 2011. IARIA Conferences. [Online]. http://www.iariajournals.org/software/