CS4040 Report - Evaluating the resiliency of hashing algorithms to realistic TMTO implementations

Brennan Ormes

CS4040 Report

Department of Computing Science **November 2017**
University of Aberdeen
King's College
Aberdeen AB24 3UE

CS4040 Report - Evaluating the resiliency of hashing algorithms to realistic TMTO implementations

Brennan Ormes

Department of Computing Science
University of Aberdeen

November 30, 2017

The Time-memory trade-off implementation originally proposed by Hellman in 1980 and later improved in 2003 by Oechslin with his implementation called rainbow crack. The purpose of the experiments covered in this paper is to find out if the implementation is still worth using given a memory constraint and if hashes are more resilient than others when used with rainbow crack. It was discovered that there was very little difference between the hashing algorithms SHA-1 and SHA-256 when comparing the number of hashes broken. There was a huge time difference, however, there was only a 3.4% difference in the total amounts of hashes found given the dataset provided.

1 Introduction

I am going to be investigating which hashes rainbow crack can break effectively. I will be using SHA-1 and SHA-256 hash algorithms to see which one is more resistant to time-memory trade-off given a memory constraint. Because Rainbow crack is used as a benchmark for securing passwords, I will be trying to investigate what the difference between these hashes are when being cracked by rainbow crack given a memory constraint of how many tables the user/attacker can hold. The report is organised as follows: Section 2 describes the brief background of the rainbow tables and how they are used to find matching plaintexts to hashes. Related work in this area will also be discussed, Section 3 will state the research question and what the approach will be in addressing this question. In section 4, the design of the experiment will be laid out and further discussed in section 5, Section 6 will discuss the results and section 7 will then conclude this report. Explanations of terms will be included in the appendix.

2 Background and Related Work

Rainbow tables are pre-computed tables that are used to break hashes. The tables are made by using character lists that are pre-defined and can be created by the user, each table is made up of several million chains. A very important feature of rainbow tables is the reduction function, it is noteworthy that it does not reverse hash functions, but creates new hashes. Basically, if you take the hash of a plaintext that you are encrypting, and then take the reduction of that hash, it will give us another plaintext. An example of a reduction is that say we had a character set and the password length is six, and the hashing algorithm MD5, we can get the following hash for the number 300466 which is bfae48de3cd88f762e6dffa44a6be3a2. So the reduction function for this, for example, could be taking the first six numbers of this hash which will be 483887 and from there, we hash that text and then apply the reduction function to it again. The main goal of a chain of a rainbow table is that it decreases the amount of space required as each chain is represented by the final value hash. Basically, if a match is found, then we take the starting password of that chain and follow it to the point where it reaches the hash, so the plaintext that was reduced for that hash will be the password. Rainbow tables can also contain a salt for hashes which is extremely useful as a salt can be incredibly hard to crack and will take more time for a normal password cracker. If the salt is solved already solved ahead of time, there is a better chance of recovering a hash password. [4] Despite the advantages that pre-computed tables can offer, there is a large problem with the size of them, tables with high success rates can go up to hundreds of gigabytes large [1].

A Cryptanalytic Time-Memory Trade-Off by Martin E. Hellman[2] covers the algorithm by discussing that creating all possible cipher-text blocks with matching plain-text blocks given a reduction function that can be simple as dropping the last eight bits of a ciphertext will give us all possible plaintext passwords matched with the corresponding hash. Ciphertexts (C) and keys (k) are created with the following function:

$$k_i \xrightarrow{S_{k_i}(P_0)} C_i \xrightarrow{R(C_i)} k_i + 1$$

Which then goes onto:

$$k_i \xrightarrow{f} k_i + 1 \xrightarrow{f} k_i + 2 \rightarrow \dots$$

If we have a fixed plaintext P_0 and the corresponding ciphertext C_0 then we can produce all possible plaintexts by applying the reduction function f on them

In this paper, he discusses that in order to reduce memory requirements, tables are only searched by their endpoints. Hellman concludes that the time-memory trade-off technique is effective against block cipher and can even be used on synchronous stream ciphers, however, he also covers that there are limitations to the algorithm. This implies the minimum safe key length would be 128 bits. [2]

In Phillipe Oeschslins paper, he also covers the algorithm again but offers a new table structure, rainbow tables. [3] His proposed improved tables offered reduced lookups and merges and showed that rainbow chains have no loops since the reduction function can only appear once per chain. Rainbow chains also have a constant length instead of the variable length. In his paper he tests Microsoft Windows passwords as at the release of the paper, they had a real-world significance. Tests from his paper found that the rainbow table method was about 7 times faster than the original method by Hellman.

3 Research Question

As seen in the previous section, there are works that use these precomputed tables to test how quickly hashes can be broken and the cost of them, but never have spoken about what realistic storage space will be needed for high success rates. Others have improved on the time-memory trade-off technique by Hellman, however, the main issue persists with these precomputed these tables is that they are very large; rainbow tables can be over hundreds of gigabytes large, even nearing a terabyte [1]. The goal of this report will contribute to identifying suitable memory constraints that can allow penetration testers use to effectively test the limits of their own password hashes. I want to investigate how penetration testing is affected by time-memory trade-off, as much as rainbow crack can be used as a benchmark to see if we have secure passwords or not. This research does not take into account that people walk about with large amounts of storage space on them.
The research questions are as follows:

- Can a memory constraint of 8 gigabytes hold enough rainbow tables to crack common passwords?
- Is there a difference in success rates between the hash types given the constraint of physical memory?

In order to answer these questions, an experiment will be conducted where a variety of hash algorithms will be cracked using a list of common passwords made up of five hundred, eight character passwords which were taken from the famous rockyou.txt password list. [5] To investigate these constraints, I will be using the previously mentioned software Rainbow Crack. Rainbow Crack is a password penetration tool that uses Rainbow tables (That were described in section 2) to break hashes and find plaintexts. I will also be using "rtgen" to generate the tables and "rtsort" to maximise performance. I will be generating the tables with the following algorithms; SHA-1 and SHA-256. The tables I will be generating allow me to easily set the minimum and maximum plaintext length of the hashes and allow me to control the character sets that these hashes will be using. I will be creating nine tables for each hashing algorithm, the success rate of each crack will be recorded during the experiment, the success of the tables and their hash breaking power will be given by the amounts of hashes broken over the total amount of hashes. The reason why I am only going to use nine tables, is because this will make up 4.5 gigabytes. Because this experiment is aimed towards realistic penetration testing, the idea behind this is that the tables will be stored on a portable Linux system on an 8GB USB stick, so there will need to be at least 3Gb left free on the storage medium.

4 Experimental Design

The hypothesis was:

- With a memory constraint, the hypothesis is that the cracking software will successfully break these passwords for each of the hash types.
- For each of the hashes, the success rate will not differ much.

The target population for this experiment is SHA-1 and SHA-256 hashes, as they are more relevant than md5 in terms of password security. Testing all combinations of passwords was not feasible for this experiment or possible. Real world examples of passwords were used from the rockyou.txt [5] file that contains millions of passwords. I had taken five hundred of the passwords that were eight characters long and consisted of lowercase letters and numbers to test the software on. The reason why I have picked eight character passwords is because it's a realistic outlook of companies with security policies that would require their users and/or employees to create passwords that must have at least eight characters within their password, however, they would usually require users to also include numbers within their passwords as well. I have not included passwords with symbols in my testing because the creation of these rainbow tables will require me to have large amounts of hard disk space which I do not have or can get access to and go by the contents of the rockyou.txt word list, most of the common passwords within do not include upper-case or symbols, so I decided to not include it in my tests. To ensure a consistent rate of the password cracking I will run the cracking software under controlled conditions using the full potential of the system I have access to currently. System hardware will be listed in the appendix, as it's important to know what the software was run on. I will also run the ”rtsort” part of the rainbow crack software on each of the tables to ensure maximum performance of the tables when used for breaking the hashes. I have hashed all the passwords with each of the hashing methods mentioned in the previous section. To test the hypothesis, each of the algorithms had the password cracker used on them with the limited amount of rainbow tables given.

5 Results

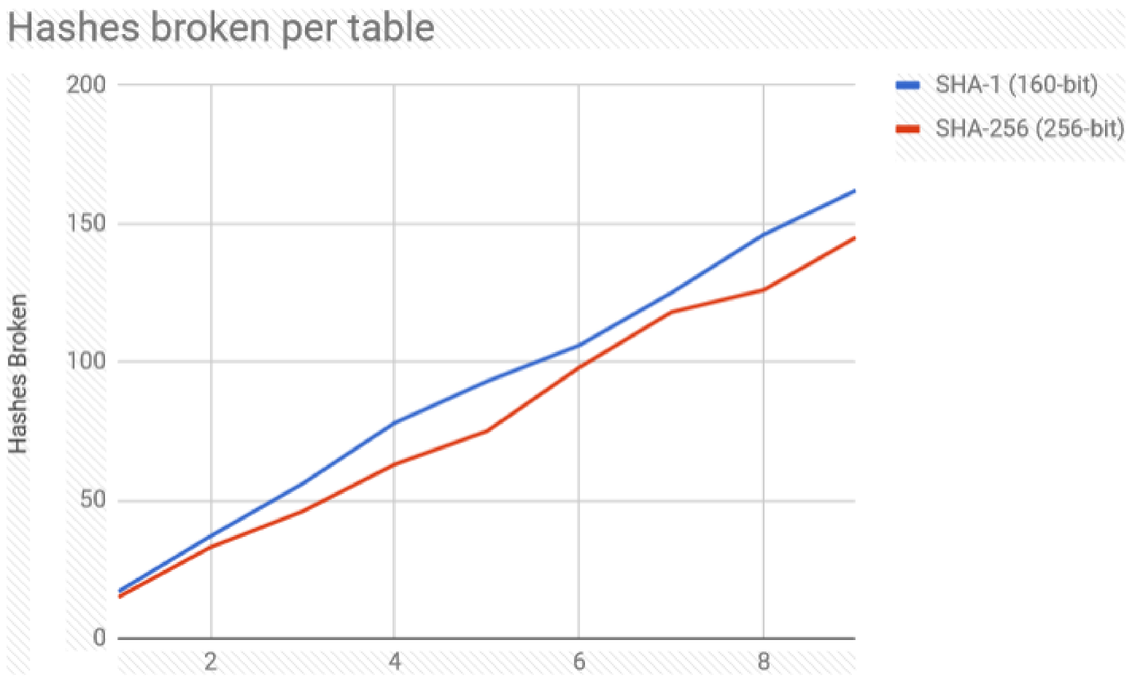
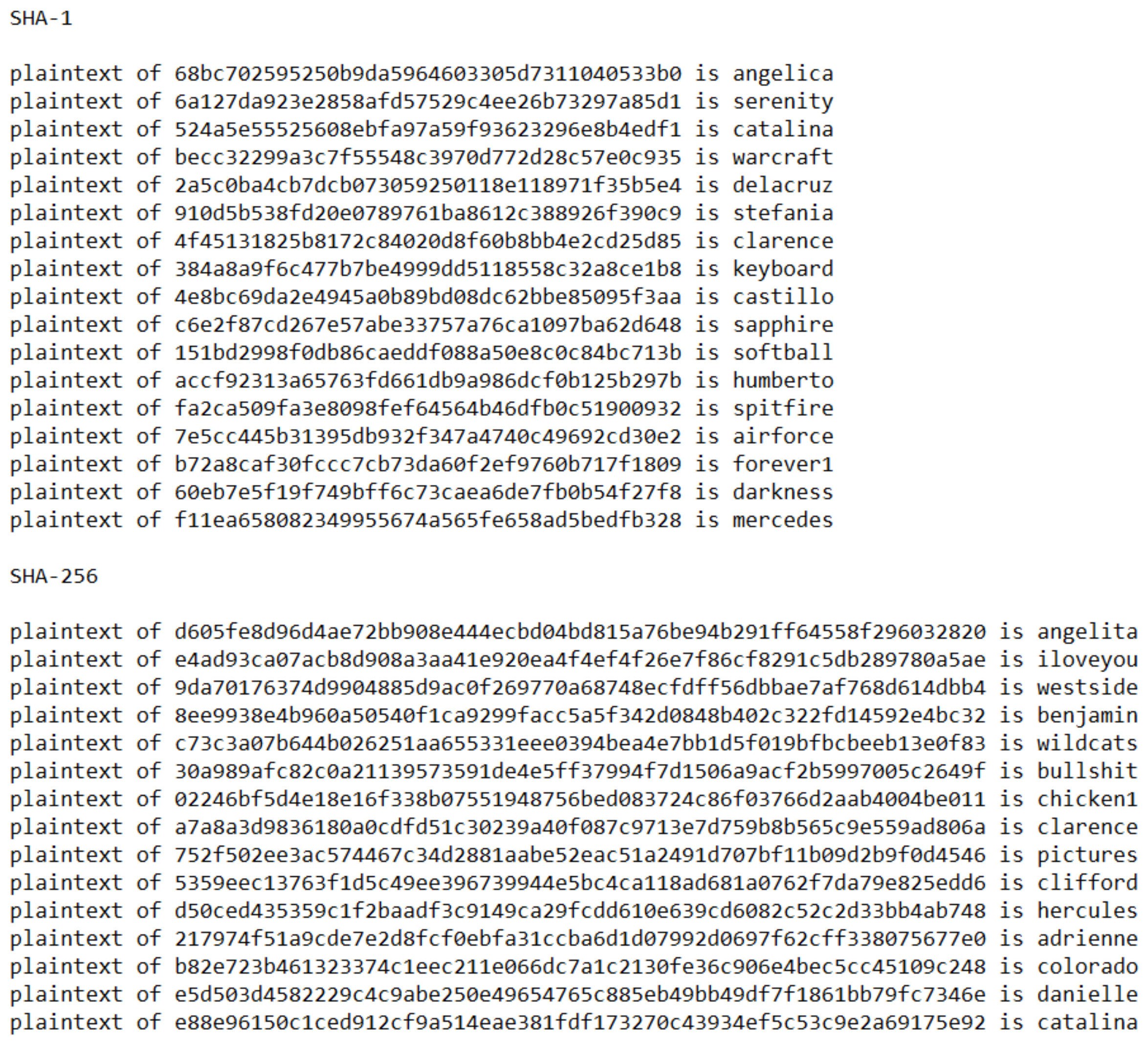


Figure 1: Hashes broken per amount of tables

Figure 1 displays the performance of SHA-1 and SHA-256 cracking. The amount of rainbow tables used is increased through each iteration of testing. The results suggest that hashing performance is gained with more tables being used with the software. SHA-256 is only slightly more resistant than SHA-1 hashes, rainbow crack managed to break 162 SHA-1 hashes compared to SHA-256, where only 145 got broken. With each extra table used in the cracking process, the amount of hashes cracked grows by roughly 5-30 extra hashes per table. The total success for the number of tables is 32.4% for SHA-1 and 29% for SHA-256.

An interesting comparison to make is the time it takes for the program to search through and find the plaintexts, SHA-256 took consideredly more time to find all the possible plaintexts compared to the 160-bit SHA-1 hashes. However, still quicker than regular brute-force password crackers. The time results can be seen in figure 3

An observation to make as well is the first set of hashes found with one table. As seen in figure 2 we can see that we have different results, yet both tables use the same reduction function. That said, we know that reduction function chops off parts of the hashed text to be used as plaintext to create another hash. It is seen that we have two overlaps, which is the plaintext ”clarence” and ”catalina”, other than that, the hashes we found were widely different from each other, I believe that this overlap is just a coincidence and can be passed off as circumstantial.



```
SHA-1

plaintext of 68bc702595250b9da5964603305d7311040533b0 is angelica
plaintext of 6a127da923e2858afd57529c4ee26b73297a85d1 is serenity
plaintext of 524a5e55525608ebfa97a59f93623296e8b4edf1 is catalina
plaintext of becc32299a3c7f55548c3970d772d28c57e0c935 is warcraft
plaintext of 2a5c0ba4cb7dcb073059250118e118971f35b5e4 is delacruz
plaintext of 910d5b538fd20e0789761ba8612c388926f390c9 is stefania
plaintext of 4f45131825b8172c84020d8f60b8bb4e2cd25d85 is clarence
plaintext of 384a8a9f6c477b7be4999dd5118558c32a8ce1b8 is keyboard
plaintext of 4e8bc69da2e4945a0b89bd08dc62bbe85095f3aa is castillo
plaintext of c6e2f87cd267e57abe33757a76ca1097ba62d648 is sapphire
plaintext of 151bd2998f0db86caeddf088a50e8c0c84bc713b is softball
plaintext of accf92313a65763fd661db9a986dcf0b125b297b is humberto
plaintext of fa2ca509fa3e8098fef64564b46dfb0c51900932 is spitfire
plaintext of 7e5cc445b31395db932f347a4740c49692cd30e2 is airforce
plaintext of b72a8caf30fccc7cb73da60f2ef9760b717f1809 is forever1
plaintext of 60eb7e5f19f749bff6c73caea6de7fb0b54f27f8 is darkness
plaintext of f11ea658082349955674a565fe658ad5bedfb328 is mercedes

SHA-256

plaintext of d605fe8d96d4ae72bb908e444ecbd04bd815a76be94b291ff64558f296032820 is angelita
plaintext of e4ad93ca07acb8d908a3aa41e920ea4f4ef4f26e7f86cf8291c5db289780a5ae is iloveyou
plaintext of 9da70176374d9904885d9ac0f269770a68748ecfdff56dbbae7af768d614dbb4 is westside
plaintext of 8ee9938e4b960a50540f1ca9299facc5a5f342d0848b402c322fd14592e4bc32 is benjamin
plaintext of c73c3a07b644b026251aa655331eee0394bea4e7bb1d5f019bfbcbeeb13e0f83 is wildcats
plaintext of 30a989afc82c0a21139573591de4e5ff37994f7d1506a9acf2b5997005c2649f is bullshit
plaintext of 02246bf5d4e18e16f338b07551948756bed083724c86f03766d2aab4004be011 is chicken1
plaintext of a7a8a3d9836180a0cdfd51c30239a40f087c9713e7d759b8b565c9e559ad806a is clarence
plaintext of 752f502ee3ac574467c34d2881aabe52eac51a2491d707bf11b09d2b9f0d4546 is pictures
plaintext of 5359eec13763f1d5c49ee396739944e5bc4ca118ad681a0762f7da79e825edd6 is clifford
plaintext of d50ced435359c1f2baadf3c9149ca29fcdd610e639cd6082c52c2d33bb4ab748 is hercules
plaintext of 217974f51a9cde7e2d8fcf0ebfa31ccba6d1d07992d0697f62cff338075677e0 is adrienne
plaintext of b82e723b461323374c1eec211e066dc7a1c2130fe36c906e4bec5cc45109c248 is colorado
plaintext of e5d503d4582229c4c9abe250e49654765c885eb49bb49df7f1861bb79fc7346e is danielle
plaintext of e88e96150c1ced912cf9a514eae381fdf173270c43934ef5c53c9e2a69175e92 is catalina
```

Figure 2: Sample hashes found between SHA-1 and SHA-256

Overall my original hypothesis is that there would not be much difference in the hashing algorithms, this is partly true but the time difference does shed a different light on what we would classify as good performance. If there was a time constraint, then I would say SHA-256 would be better to use but since time is not in the question here but storage space is, then I would say there is little to no difference in the hashing algorithms. My other hypothesis, suggesting that rainbow crack will crack all the common passwords was wrong (I will admit, this is partially a little overconfident), but to my surprise, it still performed relatively good in my opinion given that it's only four and a half gigabytes worth of rainbow tables, when compared to the creators of rainbow crack stating that for a 99.9 percent success rate you would need over 100 gigabytes worth of tables.[1]

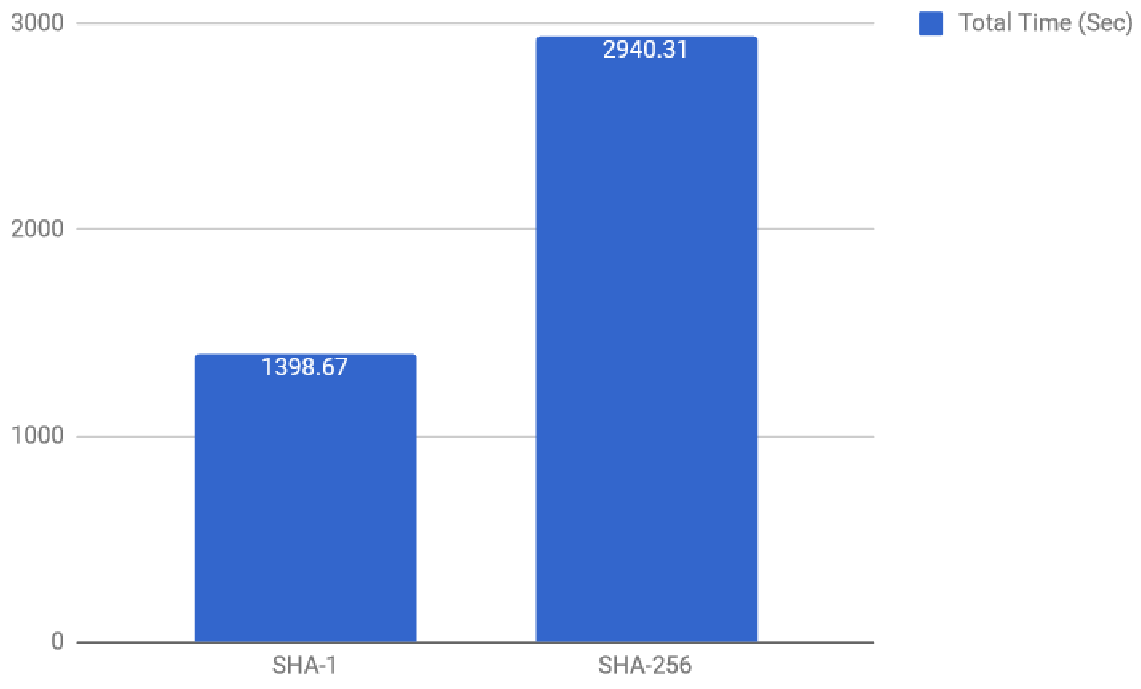


Figure 3: Total time of password breaking.

6 Discussion

The results from the experiment suggest that the hashing algorithms are only slightly more resistant to one another, this sheds light on how rainbow tables construct their pre-computed hashes, however, it should be observed how the chaining and reduction process explicitly works. My assumption was correct with the hashing algorithms not have that much of a difference, but to my surprise, it only managed to crack three-tenths of the number of passwords from both of the hash types. I hoped for better results, but on the other hand, if you were an attacker or a pen tester, you would find a considerable amount of flaws in the system. This can show that perhaps the policy will need to be improved because for an attacker, they would only need one account to do damage to a system. If the hashing system was SHA-256, it would take the attacker almost double the amount of time to get these passwords, but in the bigger picture, roughly 49 minutes to get over a hundred user accounts is considered a huge success.

Given more time, I would like to create further experiments in order to see if there is a drop-off point where we wouldn't need any more tables or even have issues to do with chain traversing (More chains can lead to more time, due to an increase in potential false alarms.)

As seen with the results, it should be observed how the reduction function picks what bits to be chopped off from hashes to create other plaintexts. I would also like to expand my dataset more, to include password less than eight lengths and perhaps longer passwords as well. More complicated passwords would also offer better test data as well. I would also like to try more varied hashing algorithms, but given my own personal memory constraints, I can only commit to two hash algorithms. Rainbow crack also has limited support for what types of hashes it can crack. If I were to perform the experiment again, I would definitely use more up to date implementation of the time-memory trade-off techniques originally proposed by Hellman.

7 Conclusion

I have conducted an experiment that evaluates rainbow cracks implementation of the time-memory trade-off algorithm given a memory constraint. This was done to identify whether it is feasible to still use rainbow crack given these memory constraints and whether are not certain hashes are more resilient to rainbow crack than others. The results show that there is only a slight difference between higher bit hashing algorithms and that performance of rainbow crack would only increase the performance by 2-5% for each extra table created. It is noted that we do not lose any performance at all giving rainbow crack more tables to use, but the time of cracking the hashes will take longer depending on the hash type we are trying to crack.

8 Appendix

Hardware Used to test rainbow crack:

- CPU: i7-7700k (3.8Ghz, 4 cores, 8 threads)
- GPU: GTX 1060 (6GB DDR5 VRAM)
- RAM: DDR4 RAM (16GB, 2400Mhz)

Chains:
Chains store possible plaintext which uses the reduction function to get new plaintexts for hashes
False Alarms:
If an ending plaintext within a chain is flagged, but the password is not found within the chain, this is considered a false alarm.
Reduction Function:
A method for extracting new plaintexts from ciphertext which is then used to create new ciphertext
Ciphertext:
Plaintext that has been encrypted using a hash function
Plaintext:
Text used for hashing
Character Set:
Used to determine what characters rtgen will use to create the rainbow tables.

References

[1] Project rainbowcrack. www.project-rainbowcrack.org, November 2017.

[2] Martin E. Hellman. A cryptanalytic time-memory trade-off, 1980.

[3] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off, 2003.

[4] Brennan Ormes. Cracking hashes with rainbow tables by using rainbowcrack, 2017.

[5] Wikipedia. rockyou.txt password list. https://en.wikipedia.org/wiki/RockYou, November 2017.