CRACKING HASHES WITH RAINBOW TABLES

by

Ryan J. Letts

Computer Engineering Technology

a submission to

Simon Walker, Computer & Networking Programs

and

Alicia Christianson

Prepared for partial credit in

ASE414

13 Duncan Court
St. Albert, AB T8N 4Z1

October 31, 2012

Simon Walker
Instructor & Associate Chair – Computer & Networking Programs
School of Information Communications and Engineering Technologies (SICET)

NAIT
10504 Princess Elizabeth Avenue
Edmonton, AB T5G 3K4

Dear Mr. Walker:

I have enclosed a copy of the technical report entitled "Cracking Hashes With Rainbow Tables" for submission as a requirement of completion to the ASE414 course at NAIT.

This paper delineates the theory rainbow tables are based on and describes the thought that went into the design and implementation of these tables in software. Cryptography is a vast and deep subject and this report was an excellent learning experience. The topic was difficult, yet fulfilling.

Please grade the attached document; any feedback would be appreciated. If there are any inquiries about this report, contact me as I will be glad to address any concerns.

Sincerely,

[illegible]

Ryan Letts
CNT Student

28 pgs Encl.

Table of Contents

LIST OF FIGURES

ABSTRACT

Learning to use rainbow tables to crack hashes is a great way for any crytpography researcher or enthusiast to gain a deeper grasp on how hashing functions can be broken programmatically using a more efficient method than brute force. The practical use of Rainbow tables can be seen as destructive towards the internet community, however, they push the boundaries of what was once thought possible in encryption breaking software. They really require an advanced perception of what is going on behind the scenes in order to be properly implemented in the real world. Space-time trade-offs are also an important concept in computer science. Rainbow tables are a good example of how this property of software can be taken advantage of to inordinately decrease the time in which a program will take to crack a given hash. The reason this is so dangerous is because as computers become faster, a password chosen by a user's window of safety becomes quite short and can lead to large losses of data.

Understanding how these technologies work helps engineers and mathematicians adapt current architectures, and form new standards for bleeding edge encryption algorithms to conform to.

1 INTRODUCTION

Data encryption is one of the most important aspects of communication in modern society. As technology has evolved over the last few decades, the need for data security has grown larger and more prominent. Along with this advancement of technology encryption, standards have been developed and broken continuously. It is crucial, as a computer engineering technologist, to understand and remain on the bleeding edge of data encryption to maintain quality levels of security to the public.

1.1 Purpose

Hashes are exhaustive of CPU resources and time consuming to crack. Rainbow tables are a tool that has been developed to drastically reduce the amount of time it can take to break various types of modern cryptography. The theory behind the functionality of rainbow tables is unigenous with dictionary-style lookup tables with major overhauls. These overhauls help to save memory space while maintaining the majority of the speed associated with the older model of storing all possible hashes in a single table. The purpose of investigating rainbow tables is to breakdown the methods and ideas implemented, in order to speed up the hash breaking process. By the end of this report one can expect to have a full understanding of what occurs under the hood of a rainbow table. Also included will be a comprehensive breakdown of what is happening to a cipher text or hash as it gets passed through the table, right up until the original plan text is found. This is beneficial to anyone who is handling cryptography at any point in their career as one will be more capable of implementing such techniques in a workplace, including forensics, local security, or developing and testing encryption standards. Lastly, because of the inherent way that one must learn about rainbow table generation in order to have a full working knowledge of implementation, one will also have the perspicacity to make tables for different encryption standards. Encryption has been the foundation on which electronic communication has been based for many years, and it will continue to grow increasingly important as technology progresses. It is important to be aware of threats to security in order to protect private and valuable information.

1.2 Scope

Although alternate techniques exist to expedite the cracking of hashes, I will be limiting my focus to only rainbow tables in this report because of time and space limitations. I have chosen rainbow tables because their origins, approximately in the year 2003, make them a relatively newer tool in the encryption world. In this report I will cover what a hashing function is, as well as its basic principles, helping one understand why some of the issues that arise with rainbow tables occur. This will allow greater capacity for adapting this concept in the future to increase speed or efficiency in implementations or systems that utilize this type of encryption cracking method, without breaking the way this algorithm works. Additionally, the Space-Time trade-off theory will be covered. This theory is the concept that allows rainbow tables to store nearly every possible hash of a large set of characters, without taking up a massive amount of space in memory. It is important to understand this trade-off in computer science because it allows many of humanity's current technologies to work fluidly with specific needs. The third main topic is on implementing this tradeoff in a rainbow table. This will require covering a few more sub topics such as reduction functions, as well as table generation in this section. I will be concluding by demonstrating how to implement rainbow tables to crack hashes, as well as going over how these types of attack will affect data security in the future. The first, I placed near the end because one needs to have all of the information provided, prior to knowing how to use a rainbow table without having large gaps in logic. One will then be able to fully implement rainbow tables to accelerate the cracking of different types of hashes. This report is provided as a learning resource and is for informational purposes only. I take no responsibility for how you use the information contained within.

1.3 Background

Cryptography has been prominent in society for a long time and rainbow tables are but one of many many techniques that have been developed to help stress test security. The direct precursor to rainbow tables were the more primitive Hellman tables based off of a paper written by Martin Edward Hellman. Hellman proposed that one could use memory space on a computer to store precomputed hashes in order to expedite cracking of hashes. His method turned out to be highly efficient and led to breakthroughs in brute force style attacks against many hashing algorithms. His basic theory was that if one puts a single hash through a reduction function and hashed it again through multiple iterations, one could make a large string of hashes with plain text keys called chains. When you have formed a long chain one simply takes the starting plain text and the last has to make a single entry in the table. After you have constructed many entries, statistical calculations can be made to determine how likely your table contains every possible hash, not necessarily in an entry, but as long as it exists somewhere within a chain, the hash is crackable with your table. Brute forcing is a technique that has been around since the early days of plain letter substitution ciphers. Hellman tables have brought massive improvements upon this old method that often still works in the world of internet security. Adding speed or a greater probability of finding the hash to these methods is a highly sought after commodity. Rainbow tables are a simple way to improve upon the Hellman table and cut down on collisions that can be detrimental to attempting to crack a hash using one of these types of tables. For example, what would happen if two chains in the Hellman table at some point, reduced into the exact same hash? The answer is discombobulating. In cryptography, collisions are a vast hindrance that one must understand. Countless CPU cycles would be wasted producing duplicate data because the two chains values would remain the same throughout the rest of the iterations after a distinct point. One would then have to merge the two chains into one. This would also force the table to become more complicated, having to deal with entries that have multiple starting plain texts and one ending hash. To deal with it, one could store both of the separate chains, and lose even more CPU cycles and time each instance one would utilize the table to decrypt a hash because of the inefficiency. The bigger problem is created having to compute two chains, and having to keep track of where the merge happened in the case the desired hash is above the merge in either of the two chains. Rainbow tables intuitive design take care of much of this problem for us.

2 HASHING FUNCTIONS

The word “hashing” comes to computer science from the English definition meaning “to chop”. In computer science a perfect hashing function chops large chunks of data down into unique, yet much smaller and simpler pieces of data.

2.1 Purpose of Hashing Functions

Hashing functions are useful for maintaining a level of data integrity when downloading large files. The property that hashing functions attempt to perfect, making sure that each hash is unique for each piece of data, also comes of use in security. This is specifically true when it comes to storing passwords. Since an average password has a much shorter length than a hash would, one can somewhat ignore the data loss issue.

2.1.1 Data security in the 21st century

The world relies on cryptography. Maintaining a high level of data security in the present is of utmost importance. As society expands online usage, the concern for data security causes more alarm. People use credit cards to place millions of online transactions every day. It is clearly required to protect our credit card numbers, addresses, and identities secret online. This is where encryption plays an important role.

Using mathematics, over the course of history man has developed one way algorithms, hashing functions, that have already been brushed on, to put sensitive data through before storing it on a hard drive. This means that if an attacker gained access to the location that the data was stored, it would be very difficult for him to recover any of the original information. A problem arises when hashing smaller size strings of text such as a password, as opposed to large blocks of data. If an attacker wanted to recover a hashed password it would prove difficult, yet not impossible.

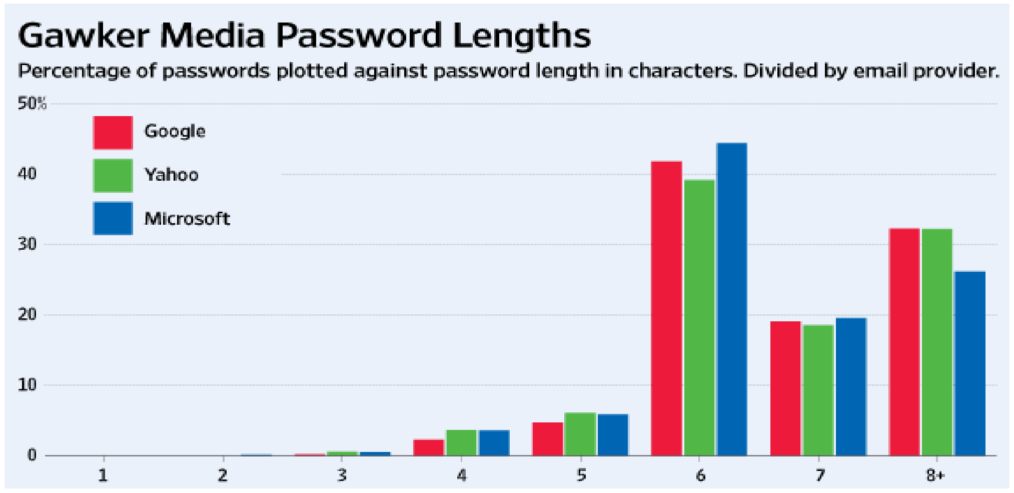

Figure 1. Gawker Media Password Lengths (The Wall Street Journal, December 13, 2010,*The Top 50 Gawker Media Passwords.*).

As illustrated in Figure 1, most people use very short passwords which reduces the number of possible passwords to a finite number depending on what characters are allowed to be used and the minimum password length. This shows that nearly 50% of people use a password that is only 6 characters long. This translates into an attacker simply constructing a list of all possible passwords, and hash each one individually and compare that hash with his table to see if he can find the plain text. Taking a look at a severe case, if only lowercase alphabetic characters are allowed in the password, that means that there are only (26^6 + 26^5 + 26^4 + 26^3 + 26^2 + 26^1 + 1) or 321272407 possible passwords that exist. Having just 321272407 passwords might seem like a lot for a human to try and figure out, but with a computer, a lookup table of this size can be generated and crack your password on a desktop computer in less than a tenth of a second.

2.1.2 Data Integrity

As mentioned, hashing functions also serve the purpose of maintaining data integrity when transferring data anywhere around a network, or even performing local hard drive checks to see if any data has been corrupted over time. The way this would be implemented is to perform a hashing function such as md5 on the contents of a file one wishes to send and store the resulting hash in a text file that can be downloaded with the file. When a user downloads the file he can perform the same hashing function on the fresh copy of the file. If the resulting hash does not match the hash contained in the text file it signifies something was corrupted along the way. This method of error checking is called a checksum. Checksums use is widespread across all of computing. Checksums appear in the smallest of program architectures, to the deployment of publicly released video games. Before network protocols like TCP were largely adopted, programmers had to rely on this technology heavily. Now the network protocol takes care of any data loss and checksumming automatically, however, this is still a relevant topic because error checking of this kind is far from disappearing. Now that some of the basic principles of hashing have been covered it is time to take a look at what goes on under the hood of a real hashing function.

2.2 Principles of Hashing Functions

Collisions are one of the properties that a working hashing function must avoid. A collision is when two plain text messages or files, when hashed, produce an indentical hash. This is an extremely rare occurrence in all modern hashing functions. Even if one character is changed in a plain text, the resulting hash is completely changed as in the aforementioned example using the md5 hashing function. md5(31337) is 6ae8caaf43a5e4a71e32d94c51d4e918 but md5(31338) is 7f3222ed7a4907370d1b41d6144ca3d7. So one can see how easy it becomes to detect even the smallest change in a plain text using a method such as this. This is an important property for hashing functions to have. If two messages, when put through a hashing function, produce the same hash, many things go wrong. Looking at it from a security standpoint, it means that a user has vastly increased the probability of his password being able to be cracked. If multiple passwords will produce the same hash as a target's password, the attacker might not know the target's personal plain text password, but he will still be able to get access to the target account, or other encrypted data because when the attacker types in the secondary password it will still allow him through. This occurring in md5 is very improbable. But when using one's own hashing function, there becomes a much greater chance of this type of collision happening. When looking at collisions from the data integrity perspective, it also remains a problem but in a different way. If a hashing function produces many collisions and one hashes their file and posts the hash for other users to download, what would happen if the function has a high chance of collision when the user checks to see if they have the correct data? If the user got corrupted data, but a correct hash, there would be no explainable error in the file that they possess. If this structure was applied to a program that does thousands of network transmissions per second, this could mean catastrophic failure. The second thing that hashes must abide to is that they come out of the function with a fixed length. If the hash comes out with a fixed length one can guarantee that there is nothing that can be learned about the plain text. This includes the length of the original message. This aspect is of great importance because knowing anything about the plain text whatsoever can give the attacker a huge advantage, and save him exponential amount of time when he is trying to crack a hash. Going back to the example of the maximum length of 321272407, if the attacker knows that the passwords length is 6 characters and no shorter, he can deduce that there are only 308915776 possible passwords that it could be. He just cut back 3.846% of the calculations he must do. Now say the user has a much weaker password, and only used 5 characters instead. He knows now that there are only 11881376 passwords and cuts back a whopping 96.301% of the calculations. This is incredibly perturbing if the hash was supposed to take 100 years to crack and it suddenly becomes feasible to crack the hash in just 4 years.

3 PROVING THE SPACE-TIME TRADEOFF THEOREM

The Space-Time trade-off is an important notion in computer science. This theorem describes a way in which memory or hard drive storage can be used in order to speed up any process on a computer that does not already use this technique.

3.1 Towers of Hanoi

The Towers of Hanoi is a simple puzzle game that stems from the year 1883. It was conceived by a French mathematician named Edouard Lucas. The idea of a physical world representation of a Space-Time trade-off came from John E. Savage's description, comparing such a trade-off to a similar but more complicated pebble game. This game demonstrates how the use of “space” or a place to hold information can be used in order to speed up an operation. The Towers of Hanoi, depicted below in Figure 2, is a game that uses three pegs along with discs that must be moved from one peg, to another.

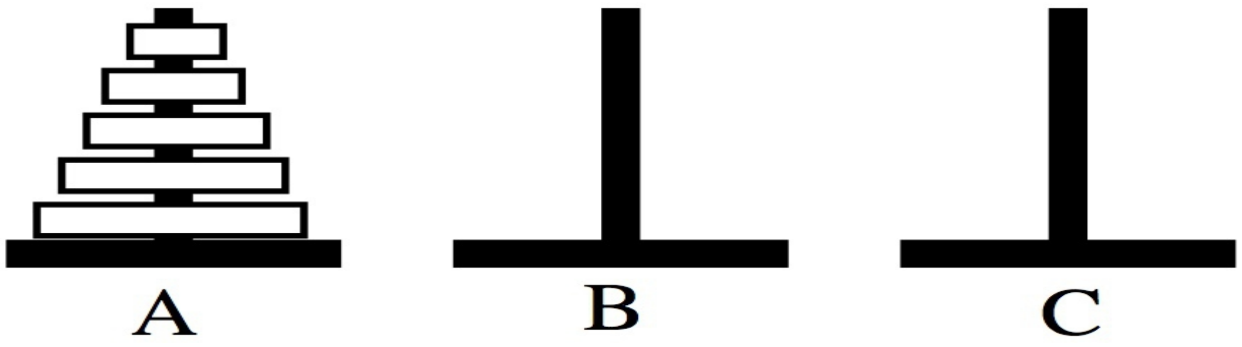

Figure 2 Towers of Hanoi
http://www.cs.brandeis.edu/~storer/JimPuzzles/MANIP/TowersOfHanoi.

As stated above, the object of the game is to move all of the discs from peg A on to C. The two rules dictate that only one disc may move per turn, and that no larger disc may sit on top of a smaller disc. This problem is fairly trivial to solve. It is determined that for this 3 peg version of the game, the minimum number of moves required to complete the puzzle with 4 discs is 15. If a fifth disc is added, (Figure 2), this same puzzle requires 31 moves to complete. Now what if a fourth peg named D were added to the puzzle simulation and an extra amount of storage space becomes available that can be utilized? The 4 disc version would actually cut down to only 9 moves required at minimum to solve the puzzle and the 5 disc version is reduced to 13. One can see the time spent moving discs goes down enormous amounts when extra space is accessible. The algorithm for the highest efficiency might become more complicated, but the overall number of minimum moves required gets cut.

3.2 Use In Computing

This theorem is veritably applied in many aspects of computing, and is even commonplace among things like file compression as well as the storing of images. If one wanted to generate a gradient that was very large, it would be much more efficient computing wise to generate a bitmap of the gradient, and store that on the hard drive. If access to the gradient were required, the bitmap could be loaded into various applications. This is, however, only one side of the coin. If storage space on the hard drive was an issue, for example if the gradient was of an extremely high resolution and required several Megabytes to store, one would likely opt for storing the algorithm, or code used to generate the gradient fresh each time costing physical CPU time, but saving all of that space. If access were required to the gradient in this case, the application would simply have to execute the algorithm in some way. A rainbow table is another way that this algorithm can be taken advantage of. This theory can be applied to hash cracking when we look at ways of storing lots of already computed hashes. If a hash is computed and stored for every possible plain text that exists then one would possess a comprehensive dictionary table that could be used to help brute force passwords. Rainbow tables are designed to improve upon this and compromise between speed, and storage space.

Implementation of a space time trade-off in a rainbow table is not as simple as one might suspect. There are a multitude of problems and issues that have to be accounted for and solved in order for the table to function. The major subtopics are comprised of reduction functions, and how they work, per-computing a chain for the table, and then generating the entire table. They have been placed in this order to try and give a solid foundation before delving directing into how these tables work.

3.3 Reduction Functions

Reduction functions are a tool that rainbow tables use in both the generation and the cracking processes. Many different types of these functions exist that all use their own algorithm. The one thing that every reduction function has in common is that it takes a given hash, and puts it through a series of steps to get a new plain text value out on the other end. This new plain text must have a high degree of uniqueness because having two hashes that produce the same plain text causes a propagation of problems in the table when it comes time to use it. The goal in feeding a certain plain text through many iterations of the same hashing and reduction function is to achieve as many unique plain texts as possible, and avoid looping through the same ones continuously. A simplistic example would be to take the last 5 values that conform to radix 10 of a hash while ignoring any hex letter values. This example is not cryptographically strong, because it will produce many duplicates, but it can demonstrate the purpose of what a reduction function does. If one took md5(12345) you get 827ccb0eea8a706c4c34a168**91**f**84**e**7**b. The reduction function known as “r1” from here on, would show that r1(md5(12345)) is 91847.

4 IMPLEMENTING SPACE-TIME TRADE-OFF IN A RAINBOW TABLE

4.1 Pre-computation of Hash Chains

The first thing that must be done when generating a rainbow table is that one must produce is a plain text, hash pair for the table called a chain. In a normal look up table one would simply hash each plain text and store both the hash and plain text values as an entry and move on to the next plain text seed to repeat this. In a rainbow table chain, the process is quite different. First, the generator must choose a variety of unique reduction functions. Take r1 from topic 4.1 and r2, which will be taking the first 5 values in the hash that conform to Radix 10 while ignoring any hex letter values. The order that these are to be used in must be stored for reference. So the generator starts a new chain by taking a new plain text, this can be random, or seeded and hashing it.

Md5(12345) = 827ccb0eea8a706c4c34a16891f84e7b. Every chain must start with a unique seed plain text. The process deviates from a lookup table here. Rather than storing the hash on the other side of the table one actually runs it through the first reduction function. Now the new plain text that we will refer to as P2 will give you a new unique seed. At this point P2 would be equal to 91847 as per the example. P2 is taken and run through the hash function again to obtain a second hash H2. H2 = md5(P2) = 1cf53d52cdd15282abceff69025b8fa4. The hash H2 must be taken and follow the process that has been described using the next reduction function and so on for as long as many reduction functions as determined. In this case P3 = r2(**1**cf**53**d**52**cdd15282abceff69025b8fa4) = 15352. H3 = md5(P3) = d71abbcc8fbb80bcfa7e373dd48b4e2a. At the end, one will be left with a long chain of plain texts and hashes. In order to make a single entry in the rainbow table, one simply takes the starting seed text and pair it with the final hash that was generated. This pair is called a chain. The chain that was generated in the example is 12345, d71abbcc8fbb80bcfa7e373dd48b4e2a.

4.2 Generating A Table

To generate a rainbow table, one continues to populate a table with these special chains. Depending on the number of entries produced, one can obtain an extremely high probability of containing every possible password inside the table. These probabilities have been known to approach 1.0 if the table generated is large enough. By altering the number of chains contained in the table and the length of each chain respectively the specifications of the table can be manipulated substantially. If the desired table needs to be decreased in size, by making the chains longer, and cutting the number of chains generated, the same probability can be maintained achieving the desired effect. If speed is the desired effect, then shorter chains and more entries are what would be required.

Now that the process of rainbow table generation is understood it is time to cover some of the problems that can occur in generation of similar types of tables that the rainbow table method helps to overcome. Hellman tables have collisions similar to those found in hashing functions. These collisions cause problems that must be overcome. This is the reason why good, unique reduction and hashing functions must be selected. If a collision occurs in a table such as this, then the table will contain duplicate data. Two chains will have the same values inside of them somewhere, but there is no way of knowing this from the outside. This situation is referred to as a split. Two chains, if looked at graphically, would appear to have started separately but merged at some point down the line. Rainbow tables account for this innately by using a different reduction function for each round of the chain generation. This means that in order for duplicate data, or a collision to occur, that two chains have to generate the same plain text on the exact same round, or row in the chain generation. The probability of such an occurrence is so low that it becomes negligible. If the same reduction function was used throughout, such as in a Hellman table, collisions would become commonplace and need to be dealt with.

Using a rainbow table to crack hashing is a complex process. The implementation of a rainbow table requires knowledge of how many reduction rounds the table went through during generation, as well as what order they were used in. This becomes important for both of the steps that are required to crack a hash.

5 USING A RAINBOW TABLE TO CRACK HASHES

5.1 Hashing and Lookup Rounds

The first thing that must be discovered is the row that contains the desired hash or, if the table even contains the hash at all. This searching phase is the part that is indeterminate. There is no way to know beforehand, if the hash is going to be contained in the table, unless specially crafted testing values are used. The process begins by taking the desired hash and comparing it against all of the values on the hash side of the rainbow table. If the hash is discovered, the chain that contains the hash has a chance of being the chain that contains the solution. The odds of finding a hash on the first search through the table is very low, but when the chain of interest has been found, the next thing to do is take note of the original plain text that is a paired with the hash, and move on to part 2 of the rainbow table implementation process, recomputing the chain. If the hash was not found in the list of hashes, it must go through a series of manipulations similar to what we do to a plain text when we are generating a chain for a rainbow table. The hash must be fed through the last reduction function used in the table, resulting in a plain text that should be hashed immediately to produce a secondary hash. This hash should be taken to the start of this process again recursively, starting with searching for it in the table. This recursion collapses back up when the number of hashes deep is equal to the number of rounds that each chain has contained in it. This prevents the table from going into an infinite recursion, should it not contain the hash. When the deepest hash is reached and there is still no match in the table, the current hash will be backed up by one round. So if the hash 827ccb0eea8a706c4c34a16891f84e7b was reduced and hashed into 1cf53d52cdd15282abceff69025b8fa4, and it was not found in the table, the current hash would be set back to 827ccb0eea8a706c4c34a16891f84e7b. From here, the second last reduction function in this case, or the next most penultimate function, depending on the round, would be used on the hash. The loop would start all the way back at the beginning again from this new hash as if one had just begun to search the table again, starting with the last used reduction function. The entire series would continue this way until it has been determined that the entire rainbow table does not contain the entry that is being searched for, or it is found.

5.2 Re-computing The Chain

When the chain that contains the hash has been discovered, all that is left to do is recompute the chain and find the hash that was initially being searched for. The steps from the creation of chains can be followed here with only a small tweak. The starting plain text will be the plain text that matches the chain that was discovered to contain the hash that is being cracked. Each time this plain text is hashed, it will simply be compared with the original hash. If the two match, then the plain text that was just previously hashed is the hashes original value and cracking has succeeded. If the two do not match, the reduction functions used during generation of the table must be applied to the hash that is being cracked. This must be done in the same order that the chain was generated in, comparing each iteration to try and determine the solution. This will occur because somewhere along the line, these reductions' same reduction functions were applied to the original hash in the reverse order and the table was confirmed to contain the answer when one of the reduced and re-hashed values matched a chain in the table. Philippe Oechslin mentions what would happen in the case that a chain was discovered that did not contain the solution in his paper *Making a Faster Cryptanalytic Time-Memory Trade-Off*. "**False alarms**[**:**]When searching for a key in a table, finding a matching endpoint does not imply that the key is in the table. Indeed, the key may be part of a chain which has the same endpoint but is not in the table. In that case generating the chain from the saved starting point does not yield the key, which is referred to as a false alarm. False alarms also occur when a key is in a chain that is part of the table but which merges with other chains of the table. In that case several starting points correspond to the same endpoint and several chains may have to be generated until the key is finally found." In this case, the next step would be to revert back to the last hash that was being used in step one and continue as if no match had occurred. This case was much more complicated to deal with in the generation of the older Hellman tables. Various tricks were done such as making chains of variable length to try and reduce collisions but these tweaks took a toll on the overall performance of the tables. Also from Oechslin's paper, it was proven that 500 Hellman tables averaged a 68.9s to success rate against a single rainbow table that did the same task in 9.37s with a false alarm rate favoring rainbow tables as well at 1492 for rainbow against 4157 for Hellman.

6 CHANGING MODERN COMPUTING

6.1 Impact On Security

Rainbow tables have had a significant impact on modern internet security. More complex passwords are often a requirement on websites these days to increase the size of a table needed to be able to crack such a password. Information security has been on the rise for many years and rainbow tables are contributing to this. Now that WEP encryption is a broken implementation of Wi-Fi security WPA-PSK and WPA2 have begun to take over as the standard. There exist rainbow tables emerging that provide an effective way of cracking these new types of security technology. This is beginning to introduce risks even in the newer routers that are going to have to be monitored. Andy O'Donnell (O'Donnell, 2012) puts it into these terms, “Hackers can crack weak Pre-Shared Keys by using brute-force cracking tools and/or Rainbow Tables to crack weak keys in a very short amount of time. All they have to do is capture the SSID (wireless network name), capture the handshake between an authorized wireless client and the wireless router or access point, and then take that information back to their secret lair so they can ‘commence to cracking’ as we say in the south.”(pg 22). Rainbow tables are bringing a new wave of wireless threats into focus. As information security continues to evolve more opportunities for rainbow tables are going to present themselves in daily life. With algorithms that allow us to crack LANMan passwords in seconds, access to information is growing. Computer speed is also improving every year which opens the doorway to more exhaustive attacks and only strengthens the chances of cracking a password with a rainbow table or other encryption cracking methods.

6.2 Improving Security

One method that has been suggested to thwart the effectiveness of rainbow tables is to salt the plain texts before hashing them and adding them to the servers list of stored hashes. This is done in an attempt to prevent the efforts of a rainbow table user. A salt is usually a randomly generated string that is attached to the plain text right before it enters a hashing function. This means that the hash will be harder to crack. A salt can be dealt with in a rainbow table but it’s functionality is out of the scope of this paper, however, if a salt is discovered through other means, it can be used to recover all of the passwords as normal after new tables have been generated for the leaked salt. Although the obstacles can be overcome, this effort is often enough to deter an attacker to another, easier target since the majority do not salt passwords in this fashion. New forms of security are developing and when they are being stress tested, rainbow tables are a factor that should be considered as a threat. Testing against this form of attack will allow security standards to have a longer lifespan and decrease costs of implementing new technologies frequently. As a software designer who deals with encryption, it is beneficial to understand how stored hashes can be exploited and defend against it.

7 CONCLUSION

Rainbow tables are one of a variety of techniques used to accelerate cracking of hashed passwords that exists today. This technique, daunting at first glance, is quite elegant when implemented in code. This solution to cracking hashes paves the way for better implementations of such tables to be developed along with even more new and advanced levels of encryption technology. Although rainbow tables still have a few inherent problems such as the possibility of a collision occurring on the same link in multiple chains, they remain one of the most popular methods of password cracking in use today. Rainbow tables will be looked upon as a great stepping stone in the future that was used to aid in the progress of information security. This stepping stone in the context of building a knowledge base is part of the reason why it is important to understand how rainbow tables function. Familiarity with this architecture will be invaluable when learning and evolving more advanced forms of encryption along with being versed in the current anti-security that is out there today. If cryptanalytic time-memory trade offs are of interest, "Making a Faster Cryptanalytic Time-Memory Trade-Off" written by Phillippe Oechslin is a great resource to gain a deeper understanding of rainbow tables.

8 SOURCES

1. Oechslin, P. (2003). *Making a Faster Cryptanalytic Time-Memory Trade-Off*. Lecture notes in computer science, 2729/2003. DOI: 10.1007/978-3-45146-4_36

2. Marechal, S. (2008). *Advances in password cracking.* Journal in computer virology, 4. DOI: 10.1007/s1146-007-0064-y

3. Nobis, J. (2011). *Rainbow Tables: Past, Present, and Future.* DFW Security professionals. Retrieved September 18, 2012, from http://insomnia.quelrod.net/docs/dfwitsecpro_2011-03_frt.pdf

4. RSA Laboratories. Retrieved September 18th, 2012, from http://www.rsa.com/.

5. Savage, J. E. (1998). *Models of computation* (Vol. 136). Reading, MA: Addison-Wesley.

6. Seward, Z. M., and Sun, A. The Top 50 Gawker Media Passwords. The Wall Street Journal, December 13, 2010.

7. Savage, J., & Swamy, S. (1979). Space-time tradeoffs for oblivious integer multiplication. *Automata, Languages and Programming*, 498-504.

8. Allouche, J. P., Astoorian, D., Randall, J., & Shallit, J. (1994). Morphisms, squarefree strings, and the tower of Hanoi puzzle. *The American Mathematical Monthly, 101*(7), 651-658.

9. O'Donnell, A. (2012, June 27). *Think your wpa2-encrypted wireless network is secure? think again.*. Retrieved from http://netsecurity.about.com/od/secureyourwifinetwork/a/WPA2-Crack.htm

10. Paul Faulstich (2009). *Rainbow Tables*. [ONLINE] Available at: http://stichintime.wordpress.com/2009/04/09/rainbow-tables-part-5-chains-and- rainbow-tables/. [Last Accessed November 19 2012].

11. Hinz, A. M. (1989). An iterative algorithm for the tower of Hanoi with four pegs. *Computing*, *42*(2), 133-140.