A Novel Time-Memory Trade-Off Method for Password Recovery

By

Vrizlynn L. L. Thing and Hwei-Ming Ying

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2009 USA

Montreal, Canada (Aug 17th - 19th)

available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/diin

ELSEVIER

Digital Investigation

A novel time-memory trade-off method for password recovery

*Vrizlynn L.L. Thing**, *Hwei-Ming Ying*

Institute for Infocomm Research, Cryptography and Security Department, 1 Fusionopolis Way, # 21-01, Connexis (South Tower), Singapore 138632, Singapore

Keywords:
Password recovery
Time-memory trade-off
Cryptanalysis
Pre-computation
Rainbow Table

ABSTRACT

As users become increasingly aware of the need to adopt strong password, it hinders the digital forensics investigations due to the password protection of potential evidence data. In this paper, we analyse and discuss existing password recovery methods, and identify the need for a more efficient and effective method to aid the digital forensics investigation process. We show that our new time-memory trade-off method is able to achieve up to a 50% reduction in terms of the storage requirement in comparison to the well-known rainbow table method while maintaining the same success rate. Even when taking into consideration the effect of collisions, we are able to demonstrate a significant increase (e.g. 13.28% to 19.14%, or up to 100% based on considering total plaintext–hash pairs generation) in terms of the success rate of recovery if the storage requirement and the computational complexity are to remain the same.

1. Introduction

In Digital Forensics, the use of password protection presents a challenge for investigators while conducting examinations. While there exist methods to decode hashes to reveal passwords used to protect potential evidence, lengthier passwords with larger character sets have been encouraged to thwart password recovery. Awareness of the need to use stronger passwords and active adoption has rendered many existing password recovery tools inefficient or even ineffective.

The more common methods of password recovery techniques are guessing, dictionary, brute force and more recently, using rainbow tables. The guessing method is attempting to crack passwords by trying ‘‘easy-to-remember’’ and common passwords like ‘‘password’’, ‘‘1234’’, etc. It can also be a password based on a user's personal information or a fuzzy index of words on the user's storage media. A statistical analysis of 28,000 passwords recently stolen from a popular U.S. website and posted on the Internet revealed that 16% took a first name as a password and 14% relied on ‘‘easy-to-remember’’ keyboard combinations (Yahoo News, 2009). Therefore, the guessing method can be quite effective in some cases where users are willing to compromise security for the sake of convenience.

The dictionary attack method composes of loading a file of dictionary words into a password cracking tool to search for a match of their hash values with the stored one. Examples of some of these password cracking tools include Cain and Abel (Cain and Abel), John the Ripper (John The Ripper) and LCP (LCPSoft).

In the brute force cryptanalytic attack, every possible combination of the password characters is attempted to perform a match comparison. It is an extremely time consuming process but the password will be recovered eventually if a long enough time is given. Cain and Abel, John the Ripper as well as LCP are able to conduct brute force attacks.

In Hellman (1980), Hellman introduced a method which involves a trade-off between the computation time and storage space needed for performing recovery of a hash value to its plaintext form. It can be applied to retrieve Windows login passwords encrypted into LM or NTLM hashes (Todorov, 2007), as well as passwords in applications using the above-

* Corresponding author.
E-mail addresses: vriz@i2r.a-star.edu.sg (V.L.L. Thing), hmying@i2r.a-star.edu.sg (H.M. Ying).

1742-2876/$ – see front matter © 2009 Digital Forensic Research workshop. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2009.06.004

mentioned hash algorithms, such as the Adobe Acrobat. It can also be used to crack the encryption used in Microsoft Word and Excel documents. Passwords encrypted with hashing algorithms such as MD5 (Rivest, 1992), SHA-2 (National Institute of Standards and Technology (NIST), 2002) and RIPEMD-160 (Dobbertin et al., 1996) are also susceptible to this recovery method. In addition, this method is applicable to many searching tasks including the knapsack and discrete logarithm problems.

In Oechslin (2003), Oechslin proposed a faster cryptanalytical time-memory trade-off method, which is an improvement over Hellman's method. Since then, this method has been widely used and implemented in many popular password recovery tools. The pre-computed tables that are generated in this method are known as the rainbow tables. The details of this method will be covered further in Section 2.

There are currently numerous products of pre-computed rainbow tables in the market. The more prominent ones are RainbowCrack (Shuanglei) and Ophcrack (Objectif Sécurité). RainbowCrack is developed by S. Zhu. It is a direct implementation of Oechslin's method. Several patches have since been released to support more hash algorithms including MD5 and SHA-1 (National Institute of Standards and Technology (NIST), 1995), as well as to increase the character set size. Ophcrack is owned by Objectif *Sécurité*, a leading Swiss consulting company in the field of information systems. Ophcrack is also a direct implementation of Oechslin's method developed by Oechslin himself and C. Tissieres. However, Ophcrack can only process LM and NTLM hashes. AccessData (AccessData) is another company adopting rainbow tables in their password recovery tools.

In this paper, we present a new design of a time-memory trade-off pre-computed table structure. Comparison of our method is made against the rainbow table method (Oechslin, 2003), as it is the most efficient and effective password recovery method to-date. We demonstrate the improvement in effectiveness and performance, in terms of increased success rate in password recovery while maintaining the same amount of storage space and computational complexity.

Our new method is shown to significantly increase the success rate by up to 100% for total plaintext–hash pairs comparison and 13.28% to 19.14% in a few example scenarios for distinct pairs comparison. Up to a 50% reduction in terms of storage requirement is achieved. This translates to about 1.35 TB storage conservation compared to the AccessData rainbow tables, which has an average size of 2.7 TB each (AccessData), while maintaining the same success rate.

The rest of the paper is organised as follow. In Section 2, we present a detailed analysis and discussion on existing time-memory trade-off password recovery methods. We describe the rainbow table method in more details in Section 3. We present the design of our new method in Section 4. Analysis and performance evaluations are carried out in Section 5 to demonstrate the performance enhancement. Future work is described in Section 6. Conclusions follow in Section 7.

2. Analysis of existing work

The idea of a general time-memory trade-off was first proposed by Hellman in 1980 (Hellman, 1980). In the context of password recovery, we describe the Hellman algorithm as follows:

We let X be the plaintext password and Y be the corresponding stored hash value of X. Given Y, we need to find X which satisfies $h(X) = Y$, where h is a known hash function. However, finding $X = h^{-1}(Y)$ is feasibly impossible since hashes are computed using one-way functions, where the reversal function, h^{-1}, is unknown. Hellman suggested taking plaintext values and applying alternate hashing and reducing, to generate a pre-computed table. For example, for a 7-character possible password (composed from a character set of English alphabets), a corresponding 128-bit hash value is obtained by performing the password hashing function on the password. With a reduction function such as $H \bmod 26^7$, where H is the hash value converted to decimal digits, the resulting values are distributed in a best-effort uniform manner. For example, if we start with the initial plaintext value of "abcdefg" and upon hashing, we may get a binary output of 0000000…00001000000…01, which is 64 0's and a 1 followed by 62 0's and a 1. In this case, $H = 2^{63} + 1 = 922$ 3372036854775809. The reduction function will then convert this value to "3665127553" which will correspond to a plaintext representation "lwmkgij", computed from $(11(26^6) + 22(26^5) + 12(26^4) + 10(25^3) + 6(26^2) + 8(26^1) + 9(26^0))$. After a pre-defined number of rounds of hashing and reducing (making up a chain), only the initial and final plaintext values are stored. Therefore, only the "head" and "tail" of a chain are stored in the table. Using different initial plaintexts, the hashing and reducing operations are repeated, to generate a larger table (of increasing rows or chains). A larger table will theoretically contain more pre-computed values (i.e. disregarding collisions), thereby increasing the success rate of password recovery, while taking up more storage space. The pre-defined number of rounds of hashing and reducing, on the other hand, increases the success rate by increasing the length of the "virtual" chain, while bringing about a higher computation overhead. To recover a plaintext from a given hash, a reduction operation is performed on the hash and a search for a match of the computed plaintext with the final value in the table is conducted. If a match is not found, the hashing and reducing operations are performed on the computed plaintext to arrive at a new plaintext for another round of search to be made. The maximum number of rounds of hashing, reducing and searching operations is determined by the chain length, before giving up. If the hash value is contained in one of these chains in the first place, a match would be found in a particular chain. The values in the chain is then worked out by performing the hashing and reducing functions to arrive at the plaintext giving the specific hash value. Unfortunately, there is a likelihood that chains with different initial values may merge due to collisions. These merges will reduce the number of distinct hash values contained in the chains and therefore, diminish the rate of successful recovery. The success rate can be increased by using multiple tables with each table using a different reduction function. If we let $P(t)$ be the success rate of using t tables, then $P(t) = 1 - (1 - P(1))^t$, which is an increasing function of t since $P(1)$ is between 0 and 1. Hence, introducing more tables increase the

Table 1 – Definition of symbols.

Symbol	Definition
x_i	Initial value of a chain
y_i	Hashed value of a password
z_i	Reduced value of a hash
c_i	Final value of a chain
h	Hash function
r_i	Reduction function
n	Number of reduction functions
m	Number of rainbow chains

success rate but also cause an increase in both the computational complexity and storage space.

In Denning (1982), Rivest suggested a method of using distinguished points as end points for chains. Distinguished points are keys which satisfy a given criteria, e.g. the first or last q bits are all 0. In this method, chains are not generated with a fixed length but they terminate upon reaching a predefined distinguished point. This method decreases the number of memory lookups compared to Hellman's method and is capable of loop detection. If a distinguished point is not obtained after a large finite number of operations, the chain is suspected to contain a loop and is discarded. Therefore, the generated chains are free of loops. One limitation is that the chains will merge if there is a collision within the same table. The variable lengths of the chains will also result in an increase in the number of false alarms. Additional computations are also required to determine if a false alarm has occurred.

In 2003, Oechslin proposed a new table structure (Oechslin, 2003) to reduce the probability of merging occurrences. These rainbow chains use multiple reduction functions in such a way that merges will only occur if the collisions occur at the same positions in both the chains. For m chains of length n each, if there is a collision in the table, the number of ways the chains will merge is the number of pairs of positions where the two collisions appear in the same column, which is $nm(m-1)/2$. Total number of possible pairs not in the same chain $= n^2m(m-1)/2$. Hence, given that there is a collision, the probability there will be a merge in the table $= (nm(m-1)/2)/(n^2m(m-1)/2) = 1/n$. An experiment carried out and presented in Oechslin's paper also showed that given a set of parameters which is constant in both scenarios, the measured coverage in a single rainbow is 78.8% compared to the 75.8% from the classical tables of Hellman with distinguished points; not to mention that the number of calculations to search is reduced as well. More details of this method are presented in the following section.

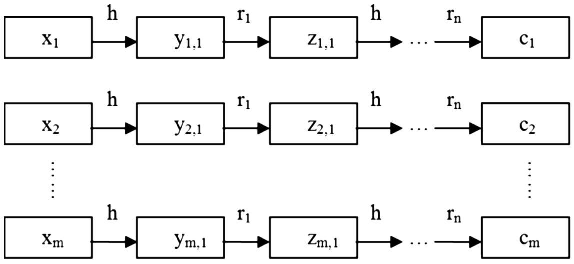

Fig. 1 – Rainbow table.

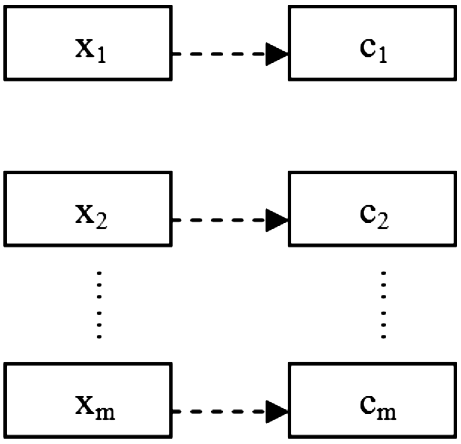

Fig. 2 – Stored values in rainbow table.

3. The rainbow table method

Rainbow tables provide an efficient way of recovering a password. Before proceeding, we state some definitions which will be used in the rest of the paper, in Table 1.

Before generating the tables, the reduction functions, $r_1,\ldots, r_n$, need to be specified. These reduction functions map hash values, y_i, to plaintexts, z_i. (An example of a reduction function is provided in Section 2). A large set of plaintexts, $x_1, x_2,\ldots, x_m$, are then chosen as the initial values of the table. As shown in Fig. 1, each of these plaintexts is then alternately hashed (by the password hashing algorithm) and reduced (by the reduction functions). Only the final values c_i are stored together with their corresponding initial values x_i in a rainbow table, as shown in Fig. 2.

To recover a password, we simply take its corresponding hash value and alternately apply reducing and hashing operations until we obtain a value that corresponds to one of the final values in the rainbow table.

For example, we would like to find the password corresponding to a password hash, v. We apply one round of reducing operation to the password hash, v, and obtain a plaintext, w. We perform a search and find a match of w to the final value of the 212th chain in the rainbow table shown in Fig. 3, i.e. $w = c_{212}$. The chain is then computed from its

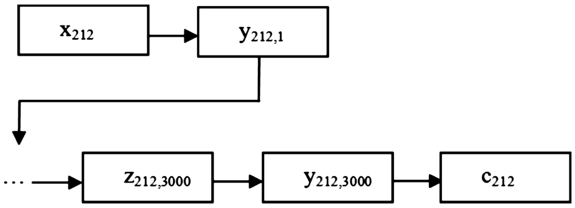

Fig. 3 – Password recovery example.

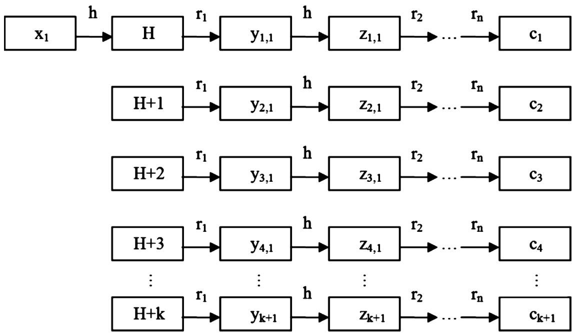

Fig. 4 – Design of new pre-computed table structure.

initial value, x_{212}, till the password hash, v, which is equal to $y_{212,\,3000}$, is reached. The password, which is the plaintext, $z_{212,\,3000}$, just before the password hash, $y_{212,\,3000}$, can then be obtained from the chain.

In the event that no matching value can be found on the rainbow table, it is deduced that the particular password does not exist within the table and cannot be recovered. To increase the success rate of recovery, the number of reduction functions (i.e. n) and chain count (i.e. m) have to be increased. However, this will at the same time, increase the computational complexity and storage requirement significantly.

4. A new design

In this section, we propose a new design of a time-memory trade-off pre-computed table structure, which is more efficient than the existing rainbow tables. In the subsequent analysis, by assuming that a greater number of hashes generated will result in a higher success rate, we will show that the new design gives a better success rate compared to rainbow tables while keeping the other parameters such as computational complexity, constant.

In this design, the same reduction functions as in the rainbow table method are used. The novelty lies in the chains generation technique. Instead of taking a large set of plaintexts as our initial values, we systematically choose a much smaller unique set. As shown in Fig. 4, we choose a plaintext x_1 and compute its corresponding hash value by applying the password hash algorithm. We let the resulting hash value written in decimal digits be H. Following that, we compute $(H+1)\,mod\,2^j$, $(H+2)\,mod\,2^j$,… ,$(H+k)\,mod\,2^j$ for a variable k, where j is the number of bits of the hash output value. For example, in MD5 hash, $j=128$. These hash values are then noted as the branches of the initial plaintext x_1. We proceed by applying alternate hashing and reducing operations to all these branches. We call this resulting extended chain, a block. The final values of plaintexts are then stored with this 1 initial plaintext value (i.e. only x_1, c_1, c_2,…, c_{k+1} are stored). We perform the same operations for the other plaintexts (i.e. x_2, x_3, etc.). These sets of initial and final values make up the new pre-computed table.

To recover a password given a hash v, we apply reducing and hashing operations alternatively until we obtain a plaintext, w, that corresponds to one of the stored final values, as in the rainbow table method. After which, we generate the corresponding branch (e.g. if $k=99$ and *chain id* $=212$, the initial value is x_3 and branch id is 12), till the value of the password hash, v, is reached.

4.1. *Differences and similarities in the designs*

We identify and list the differences and similarities between the design of our new method and the rainbow table method as follow:

- Both use n reduction functions.
- Instead of storing the initial and final values as a pair as in the rainbow table, the initial value is stored with multiple output plaintexts after a series of hashing and reducing operations. This results in a large amount of storage conservation in the new method.
- The hashes H, $(H+1)\,mod\,2^j$, $(H+2)\,mod\,2^j$,…, $(H+k)\,mod\ 2^j$ are calculated in order to generate subsequent hashes, resulting in the uniqueness of the values in the 1st column of hashes in the new method. The uniqueness of the hash values is guaranteed unless the total number of hashes is greater than 2^j. This situation is not likely to happen as it assumes an extremely large table which fully stores all the possible pre-computed values.
- In this new design, the recovery of some passwords in the 1st column is not possible as they are not stored in the first place. However, we show in our analysis and evaluation that the effect is negligible when compared to the enhancement brought about by the new design.

Next, we analyse and demonstrate the performance enhancement of our new method.

5. Analysis and performance evaluation

In this section, we perform an analysis and performance evaluation of our new method in comparison with the rainbow table method.

5.1. *Success rate analysis without collision consideration*

The success rate of recovering a password depends on the number of distinct plaintext–hash pairs generated in the chains, which in turn is dependent on the total possible number of plaintext–hash pairs generated. We will firstly perform an evaluation with the assumption that a higher total number of such pairs generated will result in a higher success recovery, disregarding collision of keys. In the subsequent subsection, we proceed with an analysis to present results based on the number of distinct pairs, as well as demonstrate the effect on the success rate.

Assuming that there are m rainbow chains, each consisting n reduction functions and requiring storage of 2 plaintexts (1 for the initial value and 1 for the final value in the chain), a rainbow table would need to store a total of $2m$ plaintexts. In this case, a rainbow table would have mn such plaintext–hash pairs.

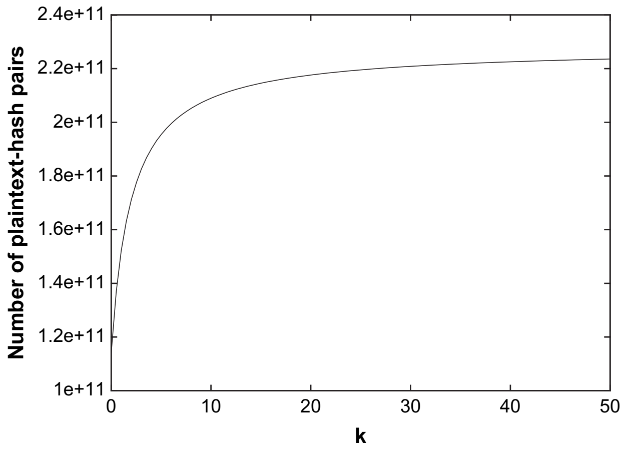

Fig. 5 – Plaintext–hash pairs against k.

In our method, each plaintext input will generate $k+1$ plaintext outputs. Therefore, each block will require $k+2$ plaintexts of storage space. If we use the same storage space as in the rainbow table, we can form $2m/(k+2)$ blocks.

Number of plaintext–hash pairs in 1 block:

$$n + kn - k = n + k(n-1) \tag{1}$$

Total number of pairs generated:

$$\begin{aligned}\frac{2m(n+kn-k)}{k+2} &= \frac{2m(nk+2n-k-2-n+2)}{k+2}\\ &= \frac{2m(n-1)\,(k+2)-2m(n-2)}{k+2}\\ &= 2m(n-1) - \frac{2m(n-2)}{k+2}\end{aligned} \tag{2}$$

This value is always greater than mn for all $k > 0$, which results in more plaintext–hash pairs and thus a higher success rate of recovery, while maintaining the same storage requirement as the rainbow table method. We prove this statement is true as follow.

We observe that the number of plaintext–hash pairs increases as k increases since $2m(n-1) - (2m(n-2))/(k+2)$ is an increasing function of k. This suggests that a large k is desirable. It is also important to note that when k is lowest at $k=0$, the chains in our method will transform to those of the rainbow table method, which is the worst case. Since we require at least 1 block to generate the output plaintexts, the upper bound for k is $(2m-2)$. Let $f(k) = 2m(n-1) - (2m(n-2))/(k+2)$. We can rewrite this equation (i.e. Equation (2)) into $f(k) = A - (B/(k+2))$, where A and B are positive constants. Taking the 1st derivative, $f^{(1)}(k) = B/(k+2)^2$. The 2nd derivative yields $f^{(2)}(k) = -2B/(k+2)^3 < 0$ for all $k > 0$. Hence, $f(k)$ is a concave function for all $k > 0$ which means that the rate of increase of f decreases as k increases (i.e. the increase in the number of plaintext–hash pairs generated is less significant as k increases). Fig. 5 illustrates this point by showing the number of plaintext–hash pairs generated against k when m is 20 million and n is 5700, which are reasonable values for the chain count and chain length in practical usage scenarios.

As shown in Fig. 5, at $k=0$ (i.e. same as the rainbow table method), the number of plaintext–hash pairs generated is $1.14e^{+11}$ compared to $1.95e^{+11}$, $2.09e^{+11}$ and $2.24e^{+11}$ when $k=5$, $k=10$ and $k=50$, respectively. The increase in the number of plaintext–hash pairs is 71.05%, 83.33% and 96.49% for values of $k=5$, $k=10$ and $k=50$, respectively. The optimal table structure in the new method is having one single block with $k=2m-2$. At this optimal configuration, which is at $k=39999998$, the number of plaintext–hash pairs generated would be $2.28e^{+11}$, resulting in an improvement factor of 100%, as shown in this example.

Next, we analyse the success rate taking into consideration the collision of plaintext–hash pairs generated.

5.2. Success rate analysis with distinct pairs

In this part of the analysis, we assume that we have a storage space of $2m$ plaintexts, and analyse the success rate of the rainbow table in comparison with the new method (with configuration of $k=2m-2$), taking into consideration the distinct plaintext–hash pairs. The number of reduction functions (i.e. n) for both methods is fixed to be the same.

We let the keyspace, which in this case, is composed of all the possible plaintext passwords, be N. We let m_i be the number of distinct keys in the ith column of a rainbow table. m_i and m_{i+1} satisfy the following recurrence relation: $m_{i+1} = N(1-(1-1/N)^{m_i})$ where $m_1 = m$. The probability of success of recovery in a rainbow table is $P(R) = 1-(1-(m_1/N))\,(1-m_2/N)\ldots(1-m_n/N)$.

Next, we let r_i be the number of distinct keys in the ith column of the new method. r_i and r_{i+1} satisfy the following recurrence relation: $r_{i+1} = N(1-(1-1/N)^{r_i})$ for all $i > 1$, $r_1 = 1$ and $r_2 = 2m-1$. The probability of success of recovery in the new design is then $P(S) = 1-(1-r_1/N)\,(1-r_2/N)\ldots(1-r_n/N)$.

In this case, $P(S) > P(R)$, which we prove as follow.

Proof:

Firstly, we note that $r_i > m_i$ for all $i > 2$.

$$\frac{m-1}{N-m} > \frac{m-1}{N}$$

$$1-\frac{m-1}{N-m} < 1-\frac{m-1}{N}$$

$$1-\frac{m-1}{N-m} < \left(1-\frac{1}{N}\right)^{m-1},$$

by Bernoulli's inequality.

$$\begin{aligned}&N-2m+1 < (N-m)\left(1-\frac{1}{N}\right)^{m-1}\\ &\left(1-\frac{2m-1}{N}\right) < \left(1-\frac{m}{n}\right)\left(1-\frac{1}{N}\right)^{m-1}\\ &\left(1-\frac{1}{N}\right)\left(1-\frac{2m-1}{N}\right) < \left(1-\frac{m}{n}\right)\left(1-\frac{1}{N}\right)^{m}\\ &\left(1-\frac{1}{N}\right)\left(1-\frac{2m-1}{N}\right)\left(1-\frac{r_3}{N}\right)\ldots\left(1-\frac{r_n}{N}\right) <\\ &\left(1-\frac{m}{n}\right)\left(1-\frac{1}{N}\right)^{m}\left(1-\frac{m_3}{N}\right)\ldots 1\left(-\frac{m_n}{N}\right)\end{aligned} \tag{3}$$

Therefore, $1-P(S) < 1-P(R)$ and hence, $P(S) > P(R)$.

Based on the above equations, we compute the success rates of recovery in the rainbow tables and the new design (with configuration of $k=2m-2$) for different example values of the number of plaintexts to be stored (i.e. storage space).

Table 2 – Success rate based on distinct plaintext–hash pairs.

Plaintext stored ($\times 10^7$)	Rainbow table	New design
3.0	57.31%	76.45%
4.0	65.69%	82.85%
5.0	71.83%	86.95%
6.0	76.46%	89.74%

The results are presented in Table 2. The common parameters used for both methods are:

- 5700 reduction functions (i.e. n).
- The character set is alpha-numeric.
- The plaintexts/passwords are 1–7 characters long.
- The hash algorithm is MD5.
- The storage space for each comparison round is the same.

As shown in Table 2, the improvement over the rainbow table method for the number of stored plaintext of 3e^7, 4e^7, 5e^7 and 6e^7 are 19.14%, 17.16%, 15.12% and 13.28%, respectively. The results show that when the number of plaintext being stored is limited, subjected to a constraint on resources, the improvement factor brought about by the new method is very significant. While the storage space increases (i.e. due to a lesser constraints on resources), the improvement factor brought about by the new method decreases. However, the success rate of recovery is still much higher than the rainbow table method. Therefore, even taking into consideration the distinct plaintext–hash pairs, the new method still shows very promising results.

5.3. *Storage space usage analysis*

In this subsection, we analyse the effectiveness of the new method by investigating the storage space usage, while keeping the number of plaintext–hash pairs as well as the computational complexity constant for both methods. Using the same rainbow chains parameters as above and by equating the number of plaintext–hash pairs for both, we obtain the following equation if only one block is generated:

$$n + k(n-1) = mn \tag{4}$$

Solving for k, we get:

$$k = \frac{mn - n}{n-1} \tag{5}$$

Hence, storage space required for the new design is obtained as follow:

$$k + 2 = \frac{m(n+1) - 2}{n-1} \tag{6}$$

Next, we prove that this value is less than $2m$ (the storage space required for the rainbow chains).

Proof:

From $m(3-n) - 2 < 0$ for chain length $n > 2$, as the new method's chain generation stops at the 2nd last column, we obtain:

$$3m - mn - 2 < 0$$
$$2m + m - 2mn + mn - 2 < 0$$
$$m + mn - 2 < -2m + 2mn$$
$$m(n+1) - 2 < 2m(n-1)$$

Hence,

$$\frac{m(n+1) - 2}{n-1} < 2m \tag{7}$$

Using the same parameters of m and n in the examples in the previous subsections, we compute the improvement factor for the new method over the rainbow table method, in terms of storage space conservation. While maintaining the recovery rate and computational complexity, this improvement factor is a significant 50% reduction in storage requirement. For example, the average size of an AccessData (AccessData) rainbow table is 2.7 TB. With the new method, the storage size can be reduced to 1.35 TB. Therefore, with the new method, only half of the storage is required as compared to the rainbow table method.

6. Future work

We are currently exploring the enhancement of our password recovery method as well as developing our method to conduct experiments to obtain results in practical scenarios. Planned work includes designing a hybrid system for password recovery by taking into consideration dictionary words in the selection of initial plaintext values. In addition, the current time-memory trade-off methods do not support recovering passwords with salt added in before hashing. We intend to explore further in this area to research on the possibility of password recovery for salted passwords.

7. Conclusions

In this paper, we firstly identified the need for a better password recovery method due to the increasingly stronger passwords users are using. We proceeded to introduce the common but weaker techniques used, such as brute force and dictionary attacks. More efficient time-memory trade-off methods were then analysed and discussed. We concluded that the rainbow table method is to-date the most popular and efficient method to perform password recovery. However, with the improvement in the adoption of stronger passwords, the rainbow table method is facing increasing challenges as seen in the huge increase in the storage requirement for newer tables. This prompted us to perform an in-depth evaluation of the rainbow table method and propose a better method to solve the challenge.

The novelty of the new method lies in the new chain generation process as well as the significant storage space conservation or successful recovery rate improvement brought about by this process. A better recovery rate is achieved if the storage space is maintained the same, or vice versa.

Comparison of our method is made against the rainbow table method due to its well-known efficiency and effectiveness in password recovery. The new method is shown to significantly increase the success rate by up to 100% for total plaintext–hash pairs comparison and 13.28% to 19.14% in a few example scenarios for distinct pairs comparison. Up to a 50% reduction in terms of storage requirement is achieved if the success rate is maintained to be the same as the rainbow table method. Therefore, the newly proposed method shows very promising results due to its significant improvement over the existing methods, to aid in the process of digital forensics investigations, to uncover password-protected potential evidence.

REFERENCES

AccessData. Decryption tools, http://www.accessdata.com.

Cain and Abel. Password recovery tool, http://www.oxid.it.

Denning DER. Cryptography and data security. Addison-Wesley Publication; Jun 1982.

Dobbertin H, Bosselaers A, Preneel B. Ripemd-160: a strengthened version of RIPEMD. In: Third international workshop on fast software encryption, lecture notes in computer science, vol. 1039; Apr 1996, pp. 71–82.

Hellman ME. A cryptanalytic time-memory trade-off. In: IEEE transaction on information theory, vol. IT-26(4); Jul. 1980, pp. 401–406.

John The Ripper. Password cracker, http://www.openwall.com.

LCPSoft. Lcpsoft programs, http://www.lcpsoft.com.

Yahoo News. Favorite passwords: ‘‘1234’’ and ‘‘password’’, http://news.yahoo.com; Feb 2009.

National Institute of Standards and Technology (NIST). Secure Hash Standard. In: Federal Information Processing Standards Publication 180-2; Aug 2002.

National Institute of Standards and Technology (NIST). Secure Hash Standard. In: Federal Information Processing Standards Publication 180-1; Aug 1995.

Objectif Sécurité. Ophcrack, http://ophcrack.sourceforge.net.

Oechslin P. Making a faster cryptanalytic time-memory trade-off. In: 23rd Annual international cryptology conference (CRYPTO 2003) – advances in cryptography, lecture notes in computer science, vol. 2729. Springer-Verlag; Oct 2003. p. 617–30.

Rivest R. The MD5 message-digest algorithm. In: IETF RFC 1321; Apr 1992.

Shuanglei Z. Rainbowcrack – the time-memory tradeoff hash cracker, http://project-rainbowcrack.com.

Todorov D. Mechanics of user identification and authentication: fundamentals of identity management. Auerbach Publications, Taylor and Francis Group; Jun 2007.

Vrizlynn L. L. Thing received the Ph.D. degree in Computing from the Imperial College London, U.K., in 2008, and the B.Eng. and M.Eng. degrees in Electrical and Electronics Engineering from the Nanyang Technological University, Singapore, in 2000 and 2003, respectively. She joined the Institute for Infocomm Research, Singapore, formerly known as Kent Ridge Digital Labs, in 2000, and worked on research and development in IP Mobility and Internet Security. At the same time, she also conducted Optical Fiber Communications research for her part-time Master's degree studies. Her Ph.D. work was on detecting and mitigating Distributed Denial-of-Service attacks. After obtaining her Ph.D. degree, she returned to the Institute for Infocomm Research, where she currently leads the Digital Forensics research group. Her research interests include Digital Forensics, Network Security, Internet Protocols, and Optical Fiber Communications.

Hwei-Ming Ying received the B.Sc. degree in Mathematics from the Imperial College London, U.K., in 2006. He joined the Institute for Infocomm Research, Singapore, in the same year. Currently, he is in the Digital Forensics research group. Prior to this, he has been working in areas of cryptography. His interests in Mathematics include Number Theory and Algebra.