Improved Cryptanalytic of Time-memory Trade-off Based on Rainbow Table

Yulong Tian[1], Dawu Gu[2], Haihua Gu[3]

Computer Science and Engineering Department, Shanghai Jiao Tong University, 200240 Shanghai,P.R. China[1,2]

Shanghai Huahong Integrated Circuit Co.,Ltd, 201203 Shanghai,P.R. China[3]

mathewes@sjtu.edu.cn, dwgu@sjtu.edu.cn, guhaihua@shhic.com

Cryptanalytic time-memory trade-off (TMTO) has been studied for thirty years since the original paper of Hellman. It has benefited from several important variants. Rainbow Table is one of the best known technologies among all variants of TMTO. Much of the previous work suffered from minor flaws in the description of the benchmark process used. Now we will present a set of criteria for benchmark. Based on the new criteria, we contribute several improvements for Rainbow Table. To be special, our analysis yields more than 28% improvements in time or 15% improvements in memory compared with previous work. In the second part of this paper, we will introduce an improved table structure and discuss the benefits of our proposal. Our analysis yields 25% improvements in time with the cost of about 5% success rate. As an example, we have implemented an attack platform for cracking of MS-Windows password hashes and all experiments are performed on the attack platform.

Key Words: Time-Memory Trade-Off, Rainbow Table, Improved Table Structure, NTLM

1. INTRODUCTION

Brute-force and Look-up table [11] could be used to solve most of hash-based [16], which means the one-way function, cryptanalytic problems. However, in practice, these methods both have obvious limitation, as brute-force will spend too much time each time and look-up table will require unacceptable space requirement. Thus, time-memory trade-off (TMTO), which was proposed by Hellman in 1980s [2], is a worthy replacement. The fundamental idea of TMTO is to carry out an exhaustive search once and all following instances of the original problem become easier to solve. We should notice that, in practice, TMTO can't be used to solve what can't be solved. It is used to speed up the attack for solvable problems.

1.1 Previous works

In 1980 [2], Hellman presented the TMTO and gave out a choice for parameters which satisfied $mt^2 = N$. Cycle and Merge [14] which are shown in **Figure 1**, are two flaws of classic tables and with the increase of table, the probability of cycle and merge will increase significantly. Cycle happens when two point are same to each other and merge happens when two points in the same columns are same to each other. Therefore, one of leakages in Hellman's paper is that the size of a single table should be controlled into an appropriate range. To avoid leakages of classic tables, Rivest and Oechslin introduced different variants. In 1982 [3] [15], Rivest used distinguish points to detect collision when a function is iterated and each chain would end with the distinguish point, which has special property, such as ten digitals are zero. In 2003 [4], Oechslin introduced the trade-off based on rainbow table and used it to demonstrate the efficiency of the technique by recovering the passwords of Windows XP. In rainbow table, the reduction functions in different columns are different from the others. Therefore, two different chains can merge if and only if at least two values in the same columns are same and it won't generate a cycle. Thus, it can be used to generate large tables. In 2006 [1], Barkan described a general model of TMTO and give out the rigorous bound of different variants and they stated that almost all variants are same to each other. In their works, they also propose several variants of TMTO tables and give out theory proof for time-memory-data trade-off problems and our work will use some of the basic idea of their new proposal.

1.2 Our Contribution

This paper formalizes and extends some results of rainbow table. Especially, we will focus on the analysis of parameter selection in rainbow table. We will also discuss an improved table structure and discuss some benefits for it. In the rest of this paper, we will discuss some in-depth optimization technology based on the previous result. Our work can be divided into has two parts below:

---Parameter Selection: We will introduce a set of criteria for benchmark in that part. All of following results are based on the new benchmark. In previous papers, most researchers don't take multiple rainbow table [5] into consideration. However, to get high, such as 99%, success rate, multiple tables is preferred than large single table. We will discuss it both in theory and experiments in this section. We will also discuss the situation in depth comparison of efficiency.

---Improved Table Structure: We will propose an improved table structure and this structure will benefit both from classic table and rainbow table. In this paper, we will discuss the comparison of performance between the new structure and the origin structure of rainbow table for time and success rate. We will also discuss some other benefits of improved table structure in this section.

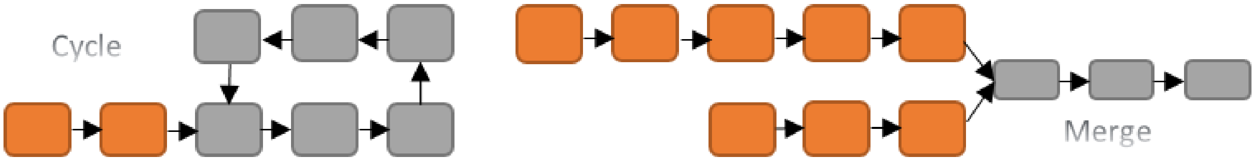


Fig. 1. Cycle and Merge. Cycle happens when same point in the chain is detected. Merge happens when two points in same column are same to each other. Thus, the table size can't be controlled into a suitable limitation.

2. PRELIMINARIES

2.1 Rainbow Table

Let $E : y = H(x)$ be a hash-based [16] function. In it, x is the key and y is the cipherkey.

Giving y, the target of the attacker is to recover y. The hash-based function is designed so it is hard to reverse it and get any answer at all, even a different password with the same hash.

Rainbow Table Method can be divided into two phases and they are shown in **Figure 2** and **Figure 3**.

Pre-computation: Pre-computation takes a one way function and creates a chain of a one-way function with t times. There are m chains in a single table, which means that there are m-number of starting points. At the end of pre-computation, starting points and ending points will be stored into files. Pre-computation is a computationally heavy solution where computing large chains through a large number of iterations. Therefore, it is identical with brute force.

In each round, it starts from x_{i-1} and ends with $x_i = R_i(H(x_{i-1}) = f_{i-1}(x_{i-1})$. R_i is the reduction function and each column in the table use different reduction function, which means that $R_i(y) \neq R_j(y)\ if\ i \neq j$.

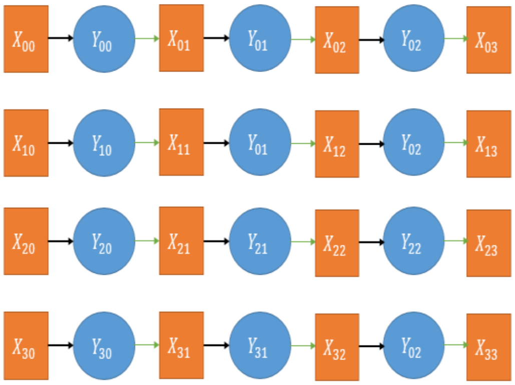


Fig. 2. Offline Phase (Pre-computation Phase).

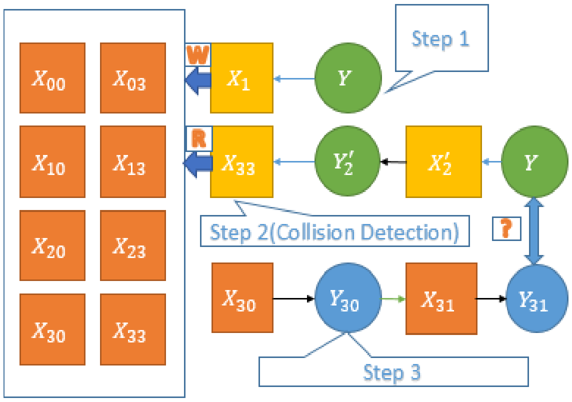


Fig. 3. Online-Analysis Phase

Online-Analysis: It can be divvied into three steps.

Step 1: Generating the pseudo-endpoints of different lengths.

Step 2: Detecting collision with the help of tables obtained in Pre-computation.

Step 3: Finally, when the collision is detected, regenerating the rest part of the chain and detecting the collision again.

In other words, it can be divided into regenerating and collision detection phases.

2.2 NTLM

In Windows, NT LAN Manager (NTLM) [6] is a suite of Microsoft security protocols that provides authentication, integrity, and confidentiality to users and NTLM is based on NT-Hash function. NT-Hashed function is better than LM-Hashed function, which is the default hash-based function [7] used by Windows XP, as it doesn't limit of the length of the password and it can full use the strength of password.

NT-Hash uses MD4 and there are several steps in it,

1. Change the source password into Unicode representation using little-endian.

2. Using MD4 to generate the target hashed-password. The length of hashed-password is 128 bits.

In our experiments, we choose 6 words and each words can choose from $a-z, A-Z,\ 0-9$ and ten common special characters. The total space of keys is $72^6 \approx\ 2^{37}$. We need to notice that all of our improvements can be applied to any hash-based function, such as DES and FEAL-32 [13].

3. PARAMETER SELECTION

To explain our result well, we consider a basic constrain, which is that success rate would be between 99.0% and 99.1%.

There are three parameters that can be adjusted in Rainbow table: the length of the chains, the number of chains per table and the number of tables produced.

We assume $\mathrm{N},\ \mathrm{m},\ \mathrm{t},\ l$ respectively for size of key space, rows of a single table, columns of each table and numbers of tables. And $\mathrm{t}_0, \mathrm{m}_0$ are unit for time and memory. To simplify the formual, we suggest $\mathrm{t}_0 = 1$ and $\mathrm{m}_0 = 1$.

With the increase of m and t, more identical points would be found. However, it is easy to find that when they exceed a certain limitation, the increase trend will be degraded. Also, with the increase of $m\ and\ t$, which means the number of different end points of a table, the probability of collision and merge will increase and more false alarms will be detected on the online-analysis phase. Finally, with the increase of l, the total run time and the total memory will be in linearly growth [12]. For the same k, with the increase of l, the total memory will linearly decrease and the total time will be in exponent growth.

Therefore, it is hard to measure the pros and cons of different parameters and we give out a criteria in following paragraphs.

3.1 Measure Criteria

Definition 1: relative ratio $k,\ k = \frac{mt}{N}$. Relative ratio means the computation capability of a single table compared with N. Also, we define the total relative ratio be that $\mathrm{K} = l\mathrm{k} = l\frac{\mathrm{mt}}{\mathrm{N}}$.

For rainbow table,

$$\mathrm{T} = \frac{\mathrm{t}^2}{2}\ l, \mathrm{M} = 2\ \mathrm{m}\ \ l, \mathrm{k}\,\mathrm{N} = \mathrm{m}\,\mathrm{t}$$

Therefore,

$$\mathrm{TM}^2\ =\ 2\ l^3\ \mathrm{k}^2\ \mathrm{N}^2\ \ (1)$$

The criteria is based on **(1)** and we combine the time and memory together in this formula. The efficiency of different parameters is determined by l^3k^2 and we need to compare l^3k^2 for different parameters to find the best parameters.

We have known that [4] for a single rainbow table, the success rate is

$$\mathrm{P}_{\mathrm{sig}} = 1 - \prod_{\mathrm{i}=1}^{\mathrm{t}} (1 - \frac{\mathrm{m}_{\mathrm{i}}}{\mathrm{N}})$$

Where $\mathrm{m}_1\ =\ \mathrm{m},\ \mathrm{m}_{\mathrm{n}+1}\ =\ \mathrm{N}(1 - \mathrm{e}^{-\frac{\mathrm{m}_\mathrm{n}}{\mathrm{N}}})$.

Therefore, for multiple rainbow tables:$\mathrm{P}_{\mathrm{succ}} = 1 - \left(1 - \mathrm{P}_{\mathrm{sig}}\right)^{\mathrm{l}}$. We shows the tables required when we need to get a success rate which isn't less than 99% in **Figure 4**. It is easy to get $1 - \left(1 - \frac{\mathrm{m}_1}{\mathrm{N}}\right)^{\mathrm{t}}\ \le\ \mathrm{P}_{\mathrm{sig}} \le 1 - \ \left(1 - \frac{\mathrm{m}_\mathrm{t}}{\mathrm{N}}\right)^{\mathrm{t}}$. Therefore, we can know $\mathrm{e}^{-\frac{\mathrm{m}_\mathrm{t}\mathrm{t}}{\mathrm{N}}} \le P_{sig} \le e^{-\frac{m_1 t}{N}}$.

3.2 Best Parameters

In origin paper, Hellman [2] proposed we should choose parameters which satisfy $mt^2 = N$ and $l = t$, which means $K = 1$. Oechslin [8] suggested that it should satisfy $m = N^{\frac{2}{3}},\ t = N^{\frac{1}{3}}$ and $l = 1$. However, success rate of all these methods are both 55% in theory and practice. Therefore, large table and multiple tables should be taken into consideration if we need to get higher success rate.

Hypothesis 1: The success rate of single table is almost determined by $m \times t$. It means that the success rate of single table is almost same (difference would be small than 2%) as soon as $m_1 t_1 = m_2 t_2$. We express it like that $P_{sig} \approx g(k)$

We conducted some experiments of different parameters and show them in **Figure 5** to explain the hypothesis.

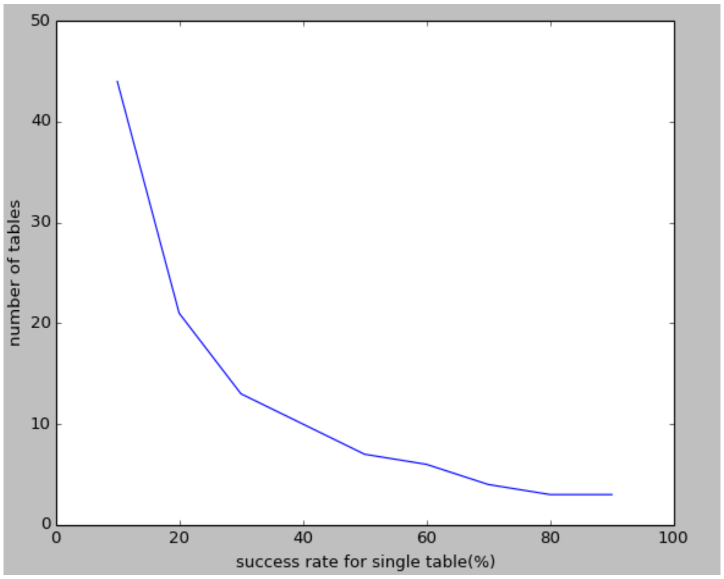
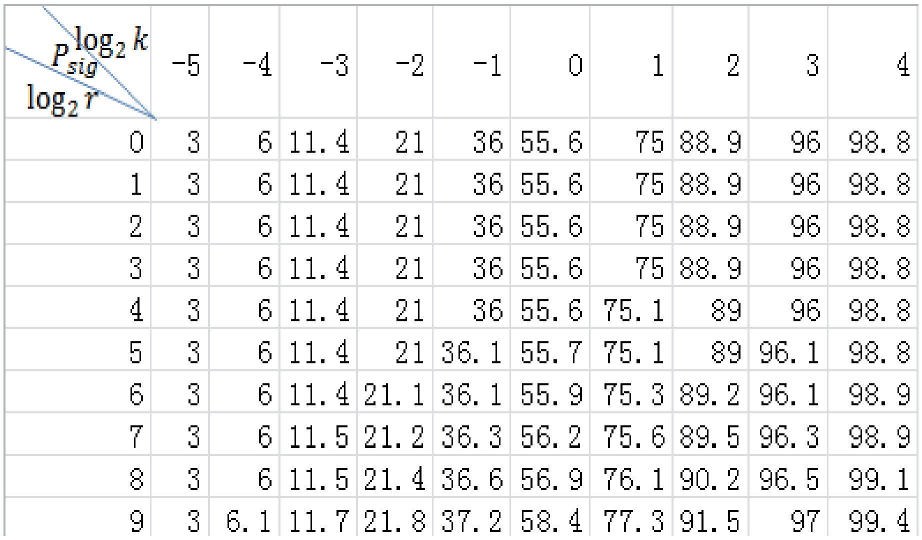


Fig. 4. Tables required as the success rate is bigger than 99%



$\log_2 k$ P_{sig} $\log_2 r$	-5	-4	-3	-2	-1	0	1	2	3	4
0	3	6	11.4	21	36	55.6	75	88.9	96	98.8
1	3	6	11.4	21	36	55.6	75	88.9	96	98.8
2	3	6	11.4	21	36	55.6	75	88.9	96	98.8
3	3	6	11.4	21	36	55.6	75	88.9	96	98.8
4	3	6	11.4	21	36	55.6	75.1	89	96	98.8
5	3	6	11.4	21	36.1	55.7	75.1	89	96.1	98.8
6	3	6	11.4	21.1	36.1	55.9	75.3	89.2	96.1	98.9
7	3	6	11.5	21.2	36.3	56.2	75.6	89.5	96.3	98.9
8	3	6	11.5	21.4	36.6	56.9	76.1	90.2	96.5	99.1
9	3	6.1	11.7	21.8	37.2	58.4	77.3	91.5	97	99.4

Fig. 5. Fixed $m \times t$. Almost same success rate for different parameters. In this Figure $m \times t = k\,N$ and $r^2 = \frac{m}{t}$. From this figure, we can conclusion that the difference of success rate between different parameters would be controlled into 2%.

We have known

$$P_{succ} = 1 - \left(1 - P_{sig}\right)^l, P_{sig} = g(k)$$

Therefore,

$$l = \lceil log_{g(k)} (1 - P_{succ}) \rceil$$

$$TM^2 = 2\left(\lceil log_{g(k)}\,(1-P_{succ})\rceil\right)^3 k^2\,N^2 = R(g(k),P_{succ})N^2$$

Figure 6 gives out the relationship between k and P_{sig}.

Figure 7 and **Figure 8** shows the final experiments using different parameters. We can conclude that $k = 1.56$ and $l = 4$ is the best choice for 99% success rate. And the parameters we choose can achieve 28% improvement in *time* or 15% improvement in *memory* compared with $k = 1$ and $l = 6$ which was chosen in previous reports.

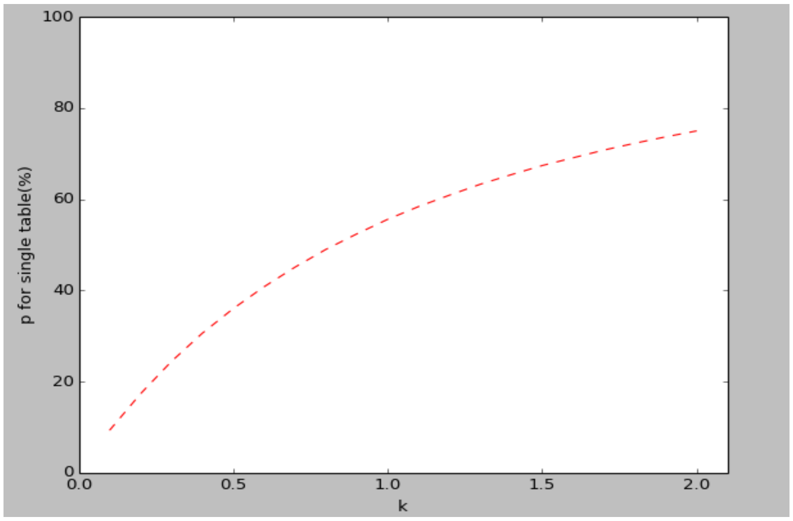
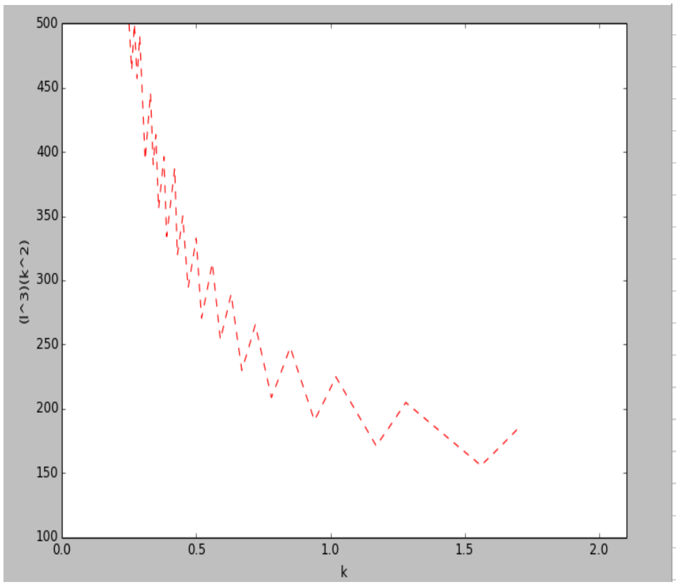
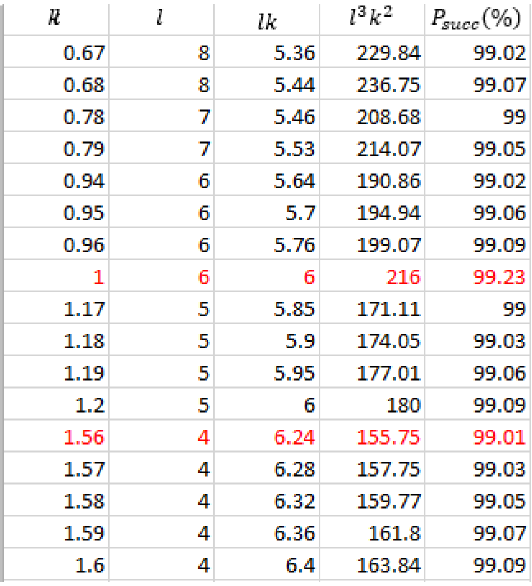


Fig. 6. Relationship between k and P_{sig}. It can be used to get any parameters which satisfy special success rate.





k	l	lk	l^3k^2	$P_{succ}(\%)$
0.67	8	5.36	229.84	99.02
0.68	8	5.44	236.75	99.07
0.78	7	5.46	208.68	99
0.79	7	5.53	214.07	99.05
0.94	6	5.64	190.86	99.02
0.95	6	5.7	194.94	99.06
0.96	6	5.76	199.07	99.09
1	6	6	216	99.23
1.17	5	5.85	171.11	99
1.18	5	5.9	174.05	99.03
1.19	5	5.95	177.01	99.06
1.2	5	6	180	99.09
1.56	4	6.24	155.75	99.01
1.57	4	6.28	157.75	99.03
1.58	4	6.32	159.77	99.05
1.59	4	6.36	161.8	99.07
1.6	4	6.4	163.84	99.09

Fig. 7. Relationship between k and l^3k^2. From it, we can know that $k = 1.56$ is the best choice.
Fig. 8. Relationship between k and $l^3\,k^2$. From it, we can conclude $k = 1.56$ can save about $\frac{216-155}{216} = 28.2\%$.

We give out **Hypothesis 1** without any theory proof and we need to state that no matter whether it is right or wrong for different N, it won't affect our improvements compared with previous choices. **Hypothesis 1** just gives us a method to choose best parameters with given success rate of a single table.

We present the general steps to find the best parameters in **Algorithm 1**.

ALGORITHM 1: General Method to get the best k

```
$arr = []$
$for\ k\ in\ [0.1 \to 2.0]$:
    Getting $g(k)$
    Computing $l = (log_{g(k)}(1 - P_{succ}))$.
    Compute $l^3k^2$ and append it to $arr$
Find the smallest element in $arr$ and find the corresponding parameter $k$.
Return $k$
```

3.3 Discussion In-Depth

l Must be an integer and it can produce a big difference of success rate and we hope to generate the exactly the same success rate.

In previous discussion, we treated that each table is exactly the same. However, it is easy to notice that isn't necessary to be same to each other. We hope to generate tables with 99.0% success rate.

$$P(k=1) = 55.5\%, 1 - \left(1 - P(k=1)\right)^6 = 0.9916$$

And,

$$1 - (1 - 0.5)\left(1 - P(k=1)\right)^5 = 0.9907$$

Therefore, if we need the lower bound of success rate is 99.0%, we can compute five tables as same as origin method. However, in the last of table, we only need to generate a table which the success rate for this table is 50% and it can save about 2.5% time for pre-computation. We can observe that the method can be used to solve the problem we mentioned above.

4. IMPROVED TABLE STRUCTURE

The main limitation of the original rainbow scheme is the fact that the time of regenerating will be in exponent growth with t [4]. At the same time, classic table will be in linearly growth [2] with the length of chain. Therefore, we want to combine the classic table and rainbow table together.

In our structure, it uses the same reduction function in the previous half and uses different reduction functions in the rest of the chain. To be specific, **Figure 9** and **Figure 10** show the difference between rainbow table and improved table structure.

$$\begin{bmatrix} a_{1,1} \xrightarrow{f_1} a_{1,2} \xrightarrow{f_2} a_{1,3} \cdots \xrightarrow{f_{\frac{t}{2}-1}} a_{1,\frac{t}{2}} \xrightarrow{f_{\frac{t}{2}}} a_{1,(\frac{t}{2}+1)} \cdots \xrightarrow{f_{t-1}} a_{1,t} \\ a_{2,1} \xrightarrow{f_1} a_{2,2} \xrightarrow{f_2} a_{2,3} \cdots \xrightarrow{f_{\frac{t}{2}-1}} a_{2,\frac{t}{2}} \xrightarrow{f_{\frac{t}{2}}} a_{2,(\frac{t}{2}+1)} \cdots \xrightarrow{f_{t-1}} a_{2,t} \\ \cdots \\ a_{m,1} \xrightarrow{f_1} a_{m,2} \xrightarrow{f_2} a_{m,3} \cdots \xrightarrow{f_{\frac{t}{2}-1}} a_{m,\frac{t}{2}} \xrightarrow{f_{\frac{t}{2}}} a_{m,(\frac{t}{2}+1)} \cdots \xrightarrow{f_{t-1}} a_{m,t} \end{bmatrix} \begin{bmatrix} a_{1,1} \xrightarrow{f_0} a_{1,2} \xrightarrow{f_0} a_{1,3} \cdots \xrightarrow{f_0} a_{1,\frac{t}{2}} \xrightarrow{f_{\frac{t}{2}}} a_{1,(\frac{t}{2}+1)} \cdots \xrightarrow{f_{t-1}} a_{1,t} \\ a_{2,1} \xrightarrow{f_0} a_{2,2} \xrightarrow{f_0} a_{2,3} \cdots \xrightarrow{f_0} a_{2,\frac{t}{2}} \xrightarrow{f_{\frac{t}{2}}} a_{2,(\frac{t}{2}+1)} \cdots \xrightarrow{f_{t-1}} a_{2,t} \\ \cdots \\ a_{m,1} \xrightarrow{f_0} a_{m,2} \xrightarrow{f_0} a_{m,3} \cdots \xrightarrow{f_0} a_{m,\frac{t}{2}} \xrightarrow{f_{\frac{t}{2}}} a_{m,(\frac{t}{2}+1)} \cdots \xrightarrow{f_{t-1}} a_{m,t} \end{bmatrix}$$

Fig. 9. Rainbow Table. Using different reduction functions in each column of the matrix.
Fig. 10. Improved Structure. Using both same reduction function and different reduction functions.

In theory, the regeneration will cost

$$\frac{t}{2}+\frac{t}{2}*\frac{t}{2}+\left(1\ +\ 2+\ldots+\frac{t}{2}\right)\approx\frac{3}{8}\,t^2$$

Therefore, for new table, we can get

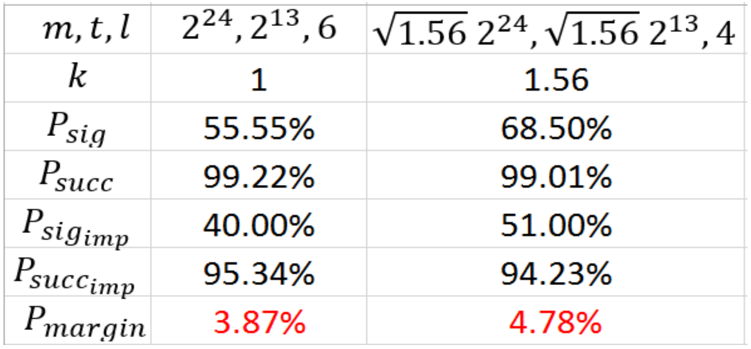
$$T=\left(\frac{t}{2}+\frac{(3t^2)}{8}\right)l\ =\frac{3}{8}\,lt^2, M=2ml, kN\ =\ mt$$

So we can know

$$TM^2\ =\frac{3}{2}\,(l^3k^2N^2)N^2$$

Therefore, the runtime of regeneration phase can be decreased by 25% in theory with same memory requirement.

Based on previous result, we choose several parameters for rainbow table and verify the efficiency of improved table structure. The results are given in **Figure 11**:



m,t,l	$2^{24},2^{13},6$	$\sqrt{1.56}\,2^{24},\sqrt{1.56}\,2^{13},4$
k	1	1.56
P_{sig}	55.55%	68.50%
P_{succ}	99.22%	99.01%
$P_{sig_{imp}}$	40.00%	51.00%
$P_{succ_{imp}}$	95.34%	94.23%
P_{margin}	3.87%	4.78%

Fig. 11. Performance of Improved Structure. $P_{sig_{imp}}$ means the success rate of improved single table and $P_{succ_{imp}}$ means the success rate of improved table. $P_{margin}=P_{succ}-P_{succ_{imp}}$.

It clearly shows that improved structure can achieve 25% improvment in time and the success rate would be only decreased by about 5% in practice. And with the increase of k, the margin of success rate become large.

There is another variant of improved structure, which was shown in **Figure 12**. It will costs

$$T\ =\left(\left(\left(\frac{t}{2}\right)+\left(\frac{t}{2}+1\right)+\ldots+t\right)+\frac{t}{2}\right)l\approx\frac{t^2}{2}l$$

So it can't give any optimization in time.

$$\begin{bmatrix} a_{\ 1,1}\xrightarrow{f_1}a_{1,2}\xrightarrow{f_2}a_{1,3}\cdots\xrightarrow{f_{\frac{t}{2}-1}}a_{1,\frac{t}{2}}\xrightarrow{f_Q}a_{1,(\frac{t}{2}+1)}\cdots\xrightarrow{f_Q}a_{1,t} \\ a_{2,1}\xrightarrow{f_1}a_{2,2}\xrightarrow{f_2}a_{2,3}\cdots\xrightarrow{f_{\frac{t}{2}-1}}a_{2,\frac{t}{2}}\xrightarrow{f_Q}a_{2,(\frac{t}{2}+1)}\cdots\xrightarrow{f_Q}a_{2,t} \\ \cdots \\ a_{m,1}\xrightarrow{f_1}a_{m,2}\xrightarrow{f_2}a_{m,3}\cdots\xrightarrow{f_{\frac{t}{2}-1}}a_{m,\frac{t}{2}}\xrightarrow{f_Q}a_{m,(\frac{t}{2}+1)}\cdots\xrightarrow{f_Q}a_{m,t} \end{bmatrix}$$

Fig. 12. Another Improved Structure. With different order of reduction functions.

4.1 Discussion In-Depth

There are two benefits of Hellman's work. The first one has been introduced in previous section and the second benefit is that it can save large sum of memory compared with Rainbow Table. We will discuss it in detail in this subsection and our improved table structure can benefit from it at the same time.

There are many technologies for memory optimization.

First, for starting points, we can choose consecutive points [9] instead of random points. In that way, each starting point only requires $\log_2 \mathrm{m}$ bits instead of $\log_2 \mathrm{N}$.

Second, during the online phase, the calculated X' are compared with the sorted ending points in the table. The best method is to divide the tables into several parts and used the hash-based bloom filter.

When we have used the two methods mentioned above, $\mathrm{m}_0' = 2\mathrm{mlog}_2\mathrm{m}$ instead of $\mathrm{m}_0 = 2\mathrm{mlog}_2\mathrm{N}$.

The next idea for staring points is that we can store $\log_2\mathrm{m}$ instead of $\mathrm{mlog}_2\mathrm{m}$ for starting points [10]. The negative side effect of this solution is a table which can no longer be sorted by the ending points which increase the search times during the online phase. In collision detection phase, the sorted table will cost $\mathrm{tlog}_2\mathrm{m}$ for sorted ending points. Therefore, if the endings points aren't sorted in advanced, it will cost $\mathrm{mlog}_2\mathrm{t}$.

Thus, if we take $\mathrm{m} = \mathrm{t}$, the saving of starting points won't give any burden compared with previous optimization. However, in that way, it will cost $\frac{\mathrm{t}^2}{2}$ during online analysis and can't be better than brute force in rainbow table. We should notice that $m = t = N^{\frac{1}{3}}$ in classic table and it can benefit from the optimization. In improved table structure, we have calculated that regeneration will cost $\frac{3}{8}\mathrm{t}^2$ instead of $\frac{\mathrm{t}^2}{2}$. Therefore, if $\mathrm{m} = \mathrm{t}$, it can benefit from classic table and won't be difficult than the brute force.

5. CONCLUSION

As there are many parameters in Rainbow Table and they can affect the efficiency in different degree, it is hard to present a criteria of benchmark. In this paper, we discuss common problems encountered when trying to compare the benchmarking results of different implementations. From this discussion, we presented a set of criteria for benchmark for Rainbow Table and gave out a pseudo-code for general N. Based on new criteria, we can find better choice of parameters than previous suggestion. In short, new parameters can achieve 28% improvments in time or 15% improvements in memory. We also discussed an improved table structure, which is a new idea of combination between classic table and rainbow table and discuss the benefits inherited from classic table and rainbow table.

Our experiments are done for the NT-based function, but the method and implementation can easily be adjusted for any Hash-based function.

We hope that our paper can assist you to find the best parameters in different atmosphere and can open your mind to find other table structure.

REFERENCE

1. Barkan E, Biham E, Shamir A. Rigorous bounds on cryptanalytic time/memory tradeoffs[M]//Advances in Cryptology-CRYPTO 2006. Springer Berlin Heidelberg, 2006: 1-21.

2. Hellman M E. A cryptanalytic time-memory trade-off[J]. Information Theory, IEEE Transactions on, 1980, 26(4): 401-406.
3. Rivest R L, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems[J]. Communications of the ACM, 1978, 21(2): 120-126.
4. Oechslin P. Making a faster cryptanalytic time-memory trade-off[M]//Advances in Cryptology-CRYPTO 2003. Springer Berlin Heidelberg, 2003: 617-630.
5. Hong J, Moon S. A comparison of cryptanalytic tradeoff algorithms[J]. Journal of cryptology, 2013, 26(4): 559-637.
6. Malhotra M, Dua B. A Review of NTLM Rainbow Table Generation Techniques[J]. GJCST-E: Network, Web & Security, 2013, 13(7).
7. Zorn G, Cobb S. Microsoft ppp chap extensions[J]. 1998.
8. Avoine G, Junod P, Oechslin P. Characterization and improvement of time-memory trade-off based on perfect tables[J]. ACM Transactions on Information and System Security (TISSEC), 2008, 11(4): 17.
9. Avoine G, Junod P, Oechslin P. Time-memory trade-offs: False alarm detection using checkpoints[M]//Progress in Cryptology-INDOCRYPT 2005. Springer Berlin Heidelberg, 2005: 183-196.
10. Spitz S, Güneysu D I T, Rupp D I A, et al. Time Memory Tradeoff Implementation on Copacobana[D]. Master’s thesis, Ruhr-Universität Bochum, 2007.
11. Borst J, Preneel B, Vandewalle J. On the time-memory tradeoff between exhaustive key search and table precomputation[C]//Symposium on Information Theory in the Benelux. TECHNISCHE UNIVERSITEIT DELFT, 1998: 111-118.
12. Il-Jun K I M, Matsumoto T. Achieving higher success probability in time-memory trade-off cryptanalysis without increasing memory size[J]. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 1999, 82(1): 123-129.
13. Kusuda K, Matsumoto T. Optimization of Time-Memory Trade-Off Cryptanalysis and Its Application to DES, FEAL-32, and Skipjuck[J]. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 1996, 79(1): 35-48.
14. Quisquater J J, Delescaille J P. How easy is collision search? Application to DES[C]//Advances in Cryptology—EUROCRYPT’89. Springer Berlin Heidelberg, 1990: 429-434.
15. Standaert F X, Rouvroy G, Quisquater J J, et al. A Time-Memory Tradeo. Using Distinguished Points: New Analysis & FPGA Results[M]//Cryptographic Hardware and Embedded Systems-CHES 2002. Springer Berlin Heidelberg, 2003: 593-609.
16. Rogaway P, Shrimpton T. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance[C]//Fast Software Encryption. Springer Berlin Heidelberg, 2004: 371-388.