Analysis and Improvement of Thing's Time-Memory Trade-Off Attack

Zhenqi Li, Dongdai Lin and Wenhao Wang

State Key Laboratory Of Information Security
Institute of Software Chinese Academy of Sciences,Beijing

Abstract. Cryptanalytic time memory trade-off is a probabilistic algorithm for inverting a generic one-way function. Since its first introduction by Hellman, many variants and their analysis results have appeared. Thing gave a new time-memory trade-off method in 2009. His new method has higher success probability and lower memory requirements, compared with Oechslin's rainbow table, but its drawback is obvious and make it to be an impractical method. In this paper, we analyze and improve Thing's method and propose a new method. We compare the success probability, memory requirements and analysis time between our method and Thing's method. Results show that our new method is better than Thing's method and fix its innate drawback.
Keywords. time memory tradeoff, rainbow attack.

1 Introduction

A basic problem in symmetric-key cryptology is the computation of preimages or inversion of one-way functions. A brute-force attack on a block cipher in a known plaintext attack considers the mapping of the key to the ciphertext, which should be a one-way function. If no shortcut method is known, and the function has an n-bit result, there are two straightforward methods: first one can perform an exhaustive search over an average of 2^{n-1} values until the target is reached. A second solution is to precompute and store 2^n input and output pairs in a table. If one then needs to invert a particular value, one just looks up the preimage in the table, so inverting requires only a single table lookup. Both methods will be impractical if n becomes larger.

The first TMTO algorithm was given by Hellman[1], and many of its extensions [2] and variants [3] [8] have appeared. All cryptanalytic TMTO algorithms consist of two phases. For a fixed one-way function to invert, certain tables are created in the pre-computation phase. Then, the inversion of a given target point is done in the online phase, utilizing the pre-computed tables. TMTO is a probabilistic process whose success probability depends on the created tables.

In 2009, Thing gave a new time-memory trade-off method[4] based on rainbow table. This new method has higher success probability and lower memory requirements compared to the traditional rainbow attack. But its drawback is obvious and makes it to be an impractical method.

Our contributions of this paper is to improve the Thing's method and present a new time-memory trade-off method. Through proof, our method has higher success probability, lower memory requirements and lower analysis time than Thing's method and resolves the innate defect of Thing's method. In section 2, we briefly introduce the basic time-memory trade-off methodology and describe Thing's method and analyze its drawback. In section 3, we propose our new method in details. In section 4, we analyze the time and memory complexity of our new method and compare our method with Thing's method in success probability, time complexity

and memory requirements. The results show that our method is better than Thing's method. We finally give the conclusion in section 5.

2 Time-memory Trade-Off Methodology and Analysis of Thing's Method

First,we give some definitions of parameters here:

- N: Search space
- M: Memory complexity
- T: Analysis time complexity
- m: Number of chains in a single table
- t: Average chain length
- r: Number of tables

2.1 Time-memory Trade-Off Methodology

Suppose that the one-way function is $f(x)$ and we got the value of $y(y \in Y)$, and we want to solve $x(x \in X)$ which meets $f(x) = y$. Hellman's time-memory trade-off method contains two stages. The first stage is precomputation(also known as *offline* stage). In this stage, we random choose m start points: $SP_0, SP_1, ..., SP_{m-1}$ while $SP_i \in X(0 \leq i \leq m-1)$. For each start point SP_i, we do iterative computations t times by using function: $F = R \circ f(f$ is the one-way function and R is called *reduction* function or *mask* function which map Y to $X)$ as follows:

$$\begin{aligned} SP_0 &= x_{0,0} \xrightarrow{F} x_{0,1} \xrightarrow{F} x_{0,2} \cdots \xrightarrow{F} x_{0,t} = EP_0 \\ SP_1 &= x_{1,0} \xrightarrow{F} x_{0,1} \xrightarrow{F} x_{1,2} \cdots \xrightarrow{F} x_{1,t} = EP_1 \\ &\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ SP_{m-1} &= x_{m-1,0} \xrightarrow{F} x_{m-1,1} \cdots \xrightarrow{F} x_{m-1,t} = EP_{m-1} \end{aligned}$$

we only save the start points and end points: $(SP_0, EP_0), (SP_1, EP_1), ..., (SP_m - 1, EP_m - 1)$ in a table and ignore the intermediate points. Use end points as the *key* to sort.

The second stage is called *online* stage. Calculating the known $y(y \in Y)$ by using *reduction* function R, we can get $x_0 = R(y)$. We search x_0 in the end points array, if we find a match: $\exists i(0 \leq i \leq m-1)$ *st.* $x_0 = x_{i,t}$. Then we calculate $x_{i,t-1}$ from the start point: $x_{i,0}$, and $x_{i,t-1}$ is likely to be the preimage of y, because:

$$x_{i,t-1} \xrightarrow{f} y \xrightarrow{R} x_{i,t} = x_0.$$

If no match is found, then keep calculate y iteratively by F as follows:

$$y \xrightarrow{R} x_0 \xrightarrow{F} x_1,$$

we search x_1 in the end points array, if we find a match: $\exists i(0 \leq i \leq m-1)$ *st.* $x_1 = x_{i,t}$. Then we calculate $x_{i,t-2}$ from the start point: $x_{i,0}$, and $x_{i,t-2}$ is likely to be the preimage of y, because:

$$x_{i,t-2} \xrightarrow{f} y \xrightarrow{R} x_{i,t-1} \xrightarrow{F} x_{i,t} = x_0.$$

We calculate y iteratively at most t times, if no match is found, then the algorithm fails. So this is a probabilistic algorithm. The success probability(also known as rate of coverage) depends on the size of the different points in the precomputaion table. More different points, higher success probability.

According to *matrix stopping rule*[1][5], searching space is $N = mt^2$. Memory requirements is $M = rm$ and analysis time is $T = rt$. Hellman set $r = t$, so the time-memory trade-off curve is $TM^2 = N^2$. Hellman recommand $T = M = N^{2/3}$. If f is a random permutation, then the curve is $TM = N$. The recommanded parameter will be $M = T = N^{1/2}$. More details can be found in [1].

In *online* stage, we may get the preimage which is not the real preimage, as follows:

$$\left.\begin{array}{l} x_i \xrightarrow{f} y_i \xrightarrow{R} x_{i+1} \\ x \xrightarrow{f} y \xrightarrow{R} x_{i+1} \end{array}\right\} \xrightarrow{F} x_{i+2} \cdots \xrightarrow{F} x_t$$

where $x_i \neq x, y_i \neq y$ $(0 \leq i \leq t)$, we get x_i after the attack, but the real preimage is x. This kind of phenomenon is called *false alarm*. The reason for *false alarm* is that two different chain may *merge* into one chain when doing the precomputation, so the real preiamge may not exist in all the calculated tables or may exist in other chains of the table. *Merge* will definitely decrease the rate of coverage of the calculated tables. So it is often to choose r different *reducation* function: $R_1, R_2, ..., R_r$ to build r different tables. In this way, if two chains of different tables collides, *merge* only occur when they collides in the same column. Collision in different column will not lead to *merge*.

In 1982, Rivest improved Hellman's method by introducing *distinguished points* which is short for DP. He noticed that in Hellman's method, we need to check the table eveytime we do the iterative calculation of y. This process is time comsuming because of the I/O operation, so he suggested that we should stop to calculate the current chain when meeting a predefined DP(Like the most significant k-bit is of the point is 0) when doing the precomputaion, and set the current DP as the end point of the current chain. So in *online* stage, we only need to check the table when we meet a predefined DP while doing the iterative calculation of y. This greatly reduce the access time of *online* stage and make the time-memory trade-off more practical.

At Crypto 2003, Oechslin proposed a new time-memory trade-off method called rainbow attack which combines the advantages of Hellman's method and DP method. He implemented different *reducation* functions to each column, and set the iteration function $F_i = R_i \circ f$ $(1 \leq i \leq t)$. The precomputation is as follows:

$$\begin{array}{rcccccl} SP_0 & = & x_{0,0} & \xrightarrow{F_1} & x_{0,1} \xrightarrow{F_2} x_{0,2} \cdots \xrightarrow{F_t} & x_{0,t} & = & EP_0 \\ SP_1 & = & x_{1,0} & \xrightarrow{F_1} & x_{1,1} \xrightarrow{F_2} x_{1,2} \cdots \xrightarrow{F_t} & x_{1,t} & = & EP_1 \\ & \vdots & & & & & \vdots & \\ SP_{mt-1} & = & x_{mt-1,0} & \xrightarrow{F_1} & x_{mt-1,1} \cdots \xrightarrow{F_t} & x_{mt-1,t} & = & EP_{mt-1} \end{array}$$

the contributions of this implementation is that firstly it can greatly reduce the probability of *merge*, because *merge* only occur when collision occureed in the same column and collision of different column will not lead to *merge*. Secondly, this method reduce the *online* analysis time by a factor of 2, from $O(t^2)$ of Hellman's method to $O(t^2/2)$. More details can be found in [3].

Tables without *merge* are called perfect tables. Creating perfect rainbow and perfect DP tables are easy since merging chains can be detected by their identical endpoints. Since endpoints need to be sorted to facilitate the look-ups, identifying the *merges* comes for free. Perfect Hellman table do not have this advantage. More details on perfect table can be found in [6].

2.2 Analysis of Thing's Method

In 2009, Thing proposed a new time-memory trade-off attack based on rainbow attack. It can save about 50% storage compared with the rainbow table with the same success probability. With the same storage, he proved that his method has higher success probability than rainbow table. The basic idea is same as rainbow table, but different in its table structure. In order to save the memory, all the start points are not chosen randomly from the search space. Here we define $f = h(h$ is a Hash function), and $F_i = R_i \circ h$ $(1 \leq i \leq t)$. The precomputation of i-th table is as follows:

$$\begin{array}{rcl} x^i \xrightarrow{h} H & \xrightarrow{R_1} x^i_{1,1} \xrightarrow{F_2} x^i_{1,2} \cdots \xrightarrow{F_t} x^i_{1,t} \\ H+1 & \xrightarrow{R_1} x^i_{2,1} \xrightarrow{F_2} x^i_{2,2} \cdots \xrightarrow{F_t} x^i_{2,t} \\ H+2 & \xrightarrow{R_1} x^i_{3,1} \xrightarrow{F_2} x^i_{3,2} \cdots \xrightarrow{F_t} x^i_{3,t} \\ \vdots & \vdots \\ H+k & \xrightarrow{R_1} x^i_{k+1,1} \xrightarrow{F_2} x^i_{k+1,2} \cdots \xrightarrow{F_t} x^i_{k+1,t} \end{array}$$

we only need to save one start point of x^i and $k+1$ end points for i-th table . k is a constant value and the *online* analysis is same as rainbow attack. When a match is found in j-th chain , it is easy to calculate the table index i and the corresponding start point $H + q(0 \leq q \leq k)$ by using the equation $j = (k+1) * (i-1) + q + 1$, then to calculate iteratively to get the preimage. For example, if $k = 99$ and the index of matching chain is $j = 212$, then we can know the table index is $i = 3$ and the branch index $q = 11$.

The optimal value of k is $2m - 2(m$ is number of chains in rainbow table). In this way, the memory usage can be maximum and only one table is calculated in precomputation stage. More details can be found in [4].

Thing's attack exits 3 main problems:

1) It is hard to recalculate a chain when *merge* is found during the precomputation stage, because the hash value of the second column increased in order and we must keep it in order for the sake of *online* phase. So we have to ignore the merged chain and save an end point of NULL value. This will greatly decrease the success probability of the precomputed tables and increase the occurrence of *false alarm*.

2) Compared with a single table of rainbow attack, provided that memory requirements of both attack are $2m$, the precomputation time of Thing's method is $P = (k+1) * t = (2m-1) * t$(we set k to be the optimal value),which is about 2 times larger than rainbow attack of $P = mt$. So, it use more precomputaion time to get higher rate of coverage which Thing didn't mentation in his paper.

3) Analysis time of *online* phase is horrific and is no better than brute force attack. In traditional time-memory trade-off, we often sort the start and end point pairs and searching time in a sorted array is often ignored during the analysis on *online* phase. But in Thing's attack, we can not sort the end points, otherwise we will not correctly find the corresponding chain. Because the hash value of the second column increased in order and sorting will break this order, thus disturb the analysis on *online* phase. So we can not ignore the searching time in a unsorted array during the analysis, this will greatly increase the time complexity on *online* phase.

Based on these problems, we give a new time-memory trade-off method that has better performance than Thing's attack in next section.

3 A New Design

The basic idea of our new design is based on Hellman+VDP scheme proposed by Hong[8], but we implement it to rainbow table in a different way and optimize the values of parameters through proof. New design has higher success probability, lower memory requirement and lower table look-up time complexity than Thing's attack. Besides, it does not need sorting during the precomputation phase.

In order to simplify the discussion, we shall restrict the typical parameters $m = t = r = N^{1/3}, d = \frac{1}{3}\log_2 N$. When calculating i-th chain of j-th table, we set the start point $SP_i^j = (0 \parallel i \parallel j)$ $(0 \le i \le m-1, (0 \le j \le r-1))$, where the length of each of the three fraction is d-bit. When doing iterative calculation from the start point, we stop calculation if the value of the most significant d-bit of the point equals i. We set this DP to be the end point of the current chain, so different chain may have different DP because of the different values of i. In this way, we do not need to store the start point and the most significant d-bit of the end point.

Generally speaking, different chain may have different length, but if we want the chain length to be around 2^β, we often set the most sigificant β-bit to be a fixed pattern(for example all β-bits are 0). So the average chain length of our design is 2^d. In practice, some chain may enter a loop and will never reach a DP, so we often set a range $[t_{min}, t_{max}]$ to control the chain length. If the chain length is not in the range $[t_{min}, t_{max}]$, this chain will be discard. In order to reduce the *online* time complexity, we also need to save the length of each chain. The value of t_{max} and t_{min} will be discussed in details in section 4.

3.1 *Offline* stage

Like the precomputation phase in rainbow attack, we choose t_{max} different evaluation function: $F_1, F_2, ..., F_{t_{max}}$ and $F_k = R_k \circ f, 1 \le k \le t_{max}$ where f is the one-way function and R_k is *reduction* function. The iterative calculation of i-th chain is $X_{i,k}^j = F_k(X_{i,k-1}^j), 1 \le k \le t_{max}$ and the calculation of j-th table is as follows:

$$\begin{aligned}
SP_0^j = (0 \parallel 0 \parallel j) &\xrightarrow{F_1^j} \circ \xrightarrow{F_2^j} \cdots \xrightarrow{F^j_{l_0^j}} (0 \parallel R_0^j) = EP_0^j \\
SP_1^j = (0 \parallel 1 \parallel j) &\xrightarrow{F_1^j} \circ \xrightarrow{F_2^j} \cdots \xrightarrow{F^j_{l_1^j}} (0 \parallel R_1^j) = EP_1^j \\
&\vdots \\
SP_{m-1}^j = (0 \parallel m-1 \parallel j) &\xrightarrow{F_1^j} \circ \xrightarrow{F_2^j} \cdots \xrightarrow{F^j_{l^j_{m-1}}} (0 \parallel R_{m-1}^j) = EP_{m-1}^j
\end{aligned}$$

for each chain, we only store:

$$\begin{aligned}
RT[0, j] = \{R_0^j, l_0^j\} \quad & t_{min} \le l_0^j \le t_{max} \\
RT[1, j] = \{R_1^j, l_1^j\} \quad & t_{min} \le l_1^j \le t_{max} \\
&\vdots \\
RT[m-1, j] = \{R_{m-1}^j, l_{m-1}^j\} \quad & t_{min} \le l_{m-1}^j \le t_{max}
\end{aligned}$$

where R_i^j $(0 \le i \le m-1, 0 \le j \le r-1)$ is the rest $(n-d)$-bits of the end point in i-th chain of j-th table, and l_i^j $(0 \le i \le m-1, 0 \le j \le r-1)$ is the length of the i-th chain in j-th table.

3.2 *Online* stage

Provided that we have got the cipher-text X, and we want to calculate its preimage. Firstly we calculate $Y_0 = R_{t_{max}}(X)$, then we search each table(provided that the current table index is j and $0 \leq j \leq r-1$). We assign chain index i to be the most significant d bits of Y_0, means $i = (Significant\ d\ bits\ of\ Y_0)$. By using i and j, we can easily get $RT[i,j] = \{R_i^j, l_i^j\}$, then we compare as follows:

$$\begin{cases} R_i^j \stackrel{?}{=} (rest\ (n-d)\ bits\ of\ Y_0) \\ l_i^j \stackrel{?}{=} t_{max} \end{cases} \tag{1}$$

if both equations succeed, then we find a match and we can calculate $X_{i,t_{max}-1}^j$ from the start point $(0 \parallel i \parallel j)$ of i-th chain in j-th table. Then we check whether $X_{i,t_{max}-1}^j$ is the real preimage or a *false alarm*.

If either equation of (1) fails, we keep searching next table. If we still can not find the real preimage of Y after searching all the r tables, then we keep calculating Y iteratively as follows:

$$X \stackrel{R_{t_{max}-1}}{\longrightarrow} Y_1 \stackrel{F_{t_{max}}}{\longrightarrow} Y_0,$$

then to check Y_0 and Y_1 separately in all the tables. Provided that we have calculated X iteratively for k times and $1 \leq k \leq t_{max}$ as follows:

$$X \stackrel{R_{t_{max}-(k-1)}}{\longrightarrow} Y_{k-1} \stackrel{F_{t_{max}-(k-2)}}{\longrightarrow} Y_{k-2} \cdots \stackrel{F_{t_{max}}}{\longrightarrow} Y_0. \tag{2}$$

Because the maximum chain length is t_{max}, the index $t_{max}-(k-1) \leq t_{max}$ in (2). Provided that the current searching table index is j, then analysis can be divided in two different situations:
(1) when $t_{min} \leq t_{max} - (k-1) \leq t_{max}$, equals to $1 \leq k \leq t_{max} - t_{min} + 1$:

For each $Y_q (0 \leq q \leq k-1)$, We assign chain index i to be the most significant d bits of Y_q, means $i = (Significant\ d\ bits\ of\ Y_q)$. Then we can get $RT[i,j] = \{R_i^j, l_i^j\}$ and compare as follows:

$$\begin{cases} R_i^j \stackrel{?}{=} (rest\ (n-d)\ bits\ of\ Y_0) \\ l_i^j \stackrel{?}{=} t_{max} - q \end{cases}$$

if both equations succeed, then we find a match and $X_{i,t_{max}-q}^j$ can be calculated from the start point of i-th chain in j-th table by iteratively doing the computation from F_1 to $F_{t_{max}-q-1}$. Then we check whether $X_{i,t_{max}-q}^j$ is a *false alarm* or the real preimage. if no match is found or *false alarm* occurred, we keep finding a match in next table. If we didn't find the real preimage after searching all the tables, then we set $k \leftarrow k+1$ and iteratively calculate the $Y_q (0 \leq q \leq k-1)$ chain and repeat the process above untile $k > t_{max} - t_{min} + 1$.
(2) when $1 \leq t_{max} - (k-1) < t_{min}$, equals to $t_{max} - t_{min} + 1 < k \leq t_{max}$:

All the chain length lies in the rage $[t_{max}, t_{min}]$, so according to the inequation $t_{max} - t_{min} + 1 < k \leq t_{max}$, we can divide $q (0 \leq q \leq k-1)$ into two disjoint ranges as follows:

$$\begin{cases} 0 \leq q < t_{max} - t_{min} \\ t_{max} - t_{min} \leq q \leq t_{max} - 1 \end{cases}$$

for all $Y_q, q \in [t_{max} - t_{min}, t_{max} - 1]$, we do not need to find the corresponding match, because $t_{max} - q < t_{min}$, and $t_{min} \leq l_i^j \leq t_{max}$, the equation $l_i^j \stackrel{?}{=} t_{max} - q$ never succeed. While for all $Y_q, q \in [0, t_{max} - t_{min})$, we find the corresponding match as situation (1).

4 Analysis and Performance Evaluation

4.1 The Optimal Value of t_{max} and t_{min}

In our design, average chain length is 2^d and precomputation time of a single table is mt. Provided that there are $r = t$ tables, then precomputation time on *online* phase is $P = mt^2$ which meet the *matrix stopping role*. If a chain length does not lie in the range $[t_{max}, t_{min}]$, we will ignore this chain and save $RT = \{-, 0\}$. This will reduce the rate of coverage of tables, but if we control the range $[t_{max}, t_{min}]$ appropriately, the influence can be ignored.

Proposition 1. *The success probability of a single table will be greater than or equal to a constant value C, if the proportion $\bar{t}/t$ is greater or equal to $-\ln(1-C)$, where $\bar{t} = t_{max} - t_{min} + 1$.*

Proof. Provided that there is no *merge* or intersection during the computation of a single table, we only consider the influence of the canceled chain to the success probability. Here we give a definition of event F:{The calculation of i-th chain in a single table failed}(also means that the length of i-th chain did not lie in the range:$[t_{max}, t_{min}]$). The probability of F is :

$$P(F) = (1 - \frac{1}{2^d})^{t_{max} - t_{min} + 1}. \tag{3}$$

The number of chains in a single table is $m = N^{1/3}$, so the average number of chains canceled during the precomputation of a single table is:

$$n_F = m * P(F) = m(1 - \frac{1}{2^d})^{t_{max} - t_{min} + 1}. \tag{4}$$

The average length of the chain canceled is t, so the rate of coverage is reduced to:

$$P_S = \frac{mt - n_F t}{mt} = 1 - (1 - \frac{1}{2^d})^{t_{max} - t_{min} + 1} \approx 1 - e^{-(\frac{t_{max} - t_{min} + 1}{2^d})}. \tag{5}$$

We assign $\bar{t} = t_{max} - t_{min} + 1$. If we want $P_S \geq C$, where C is constant value, then:

$$\begin{aligned} & P_S \geq C \\ \Rightarrow \ & 1 - e^{-(\frac{\bar{t}}{2^d})} \geq C \\ \Rightarrow \ & \bar{t} \geq -2^d \ln(1-C) \end{aligned}$$

then the proportion of $\bar{t}/t$ is:

$$\bar{t}/t \geq -2^d \ln(1-C)/t = -\ln(1-C). \tag{6}$$

Through experiment, we get:

C	$-\ln(1-C)$
99%	4.60517
95%	2.99573
90%	2.30259
80%	1.60944
70%	1.20387
60%	0.91629
50%	0.68315

Table 1. Lower bound of $\bar{t}/t$

From the table, we can see that if we assign $\bar{t} = t_{max} - t_{min} + 1 \approx 5t$, then the influence of canceled chains to success probability can be reduced to less than 1%. Less the $\bar{t}$ is, greater the influence will be. In real application, if we want the influence of canceled chains to success probability to be lower than 5%, then we can assign $t_{max} = 3.5t, t_{min} = 0.5t$, the value of $\bar{t}$ is:

$$\bar{t} = t_{max} - t_{min} + 1 \approx 3 > -\ln(1 - 95\%) = 2.99573. \tag{7}$$

In real scenario, there are *merges* and intersections in the precomputed tables. So when calculating the influence probability of the canceled chains to the success probability of a single table, we can introduce a decay factor θ which indicates the proportion of the losing points to the total calculated points. Then $1 - \theta$ indicates the proportion of the remaining points to the total calculated points. Introducing this factor to equation (5) will have no influence to the result. So the theoretical analysis of the results can be applied to the real scenario.

4.2 Time & Memory Complexity Analysis

Now, let's focus on the time complexity on *online* stage. We assign $t_{max} = c_1 t, t_{max} - t_{min} = c_2 t$ ($c1, c2$ are constant value). If we want the influence probability of the canceled chains to be equal to or less than 5%, then we only need to make sure the constant value $c_2 \geq 3$(we ignore the constant value 1 of equation (5) in the previous section), while $t_{min} \leq t$, then $c_1 \geq 4$. The time complexity of the times of the function calculation of F_i on *online* phase is:

$$O(\frac{1}{2}t_{max}^2) = O(\frac{c_1^2}{2}t^2). \tag{8}$$

While to the times of table look-up, according to the analysis of (2) in section 3.2, if $0 \leq q < t_{max} - t_{min}$, times of table look-up is $O(\frac{1}{2}(t_{max} - t_{min} + 1))$. If $t_{max} - t_{min} \leq q \leq t_{max} - 1$, times of table look-up is $O(t_{min})$. So the time complexity of total times of table look-up is:

$$O(\frac{1}{2}(t_{max} - t_{min} + 1) + t_{min}) = O(\frac{c_2^2}{2}t^2). \tag{9}$$

The time of each table look-up is $O(1)$, because we use the DP pattern to find the corresponding chain index. The time complexity above is based on a single table. Actually, we can apply the *online* phase analysis of different $r = t$ tables in parallel. This will not only reduce the analysis time of both function calculation and table look-up, but also can reduce the occurrence of *false alarm*[7]. Hong give a brief introduction of Rainbow+VDP design in [8], his design can not be implemented in parallel on the *online* phase. Besides, when finding a corresponding table by DP pattern, it will be time consuming to find the corresponding chain in the table, because he did not do sorting in precomputaion stage by chain length. If sort, the *offline* time complexity will rise. So our design is better than his design.

For memory requirements, we can see from section 3.1 that the total entry is mt and for each entry, we need $|R_i^j| + |l_i^j| (0 \leq i \leq m-1, 0 \leq i \leq r-1, r = t)$ bits. While $|R_i^j| = 2d$ bits and $t_{min} \leq |l_i^j| \leq t_{max}$, we need $\log_2(t_{max} - t_{min} + 1)$ bits to represent l_i^j. While $t_{max} - t_{min} + 1 = c_2 t, t = N^{1/3}$, so $|l_i^j|$ is:

$$|l_i^j| = \log_2(t_{max} - t_{min} + 1) = \log_2(c_2 t) = \log_2 c_2 + \log_2 t = log_2 c_2 + d \approx d. \tag{10}$$

So $|R_i^j| + |l_i^j| \approx 3d$, this result indicates that our design can save about 50% storage compared to rainbow attack.

4.3 Compare with Thing's Method

In order to facilitate the comparison, we use I to indicate our new design and II to indicate Thing' design.

Success Probability Comparison: For a single table, provided that I and II have the same storage of m entries and each entry contains $|N| = \log_2 N$ bits. The average chain length of I is $t = N^{1/3}$ which is same to each chain length of II. Now we compare the success probability of a single table of I and II taking into the consideration of collision.

Proposition 2. *The success probability of a single table of I is larger than the success probability of a single table of II, namely $P(I) > P(II)$.*

Proof. Let m_i^I denotes the number of different points in i-th column of I and $1 \leq i \leq t$. According to the memory analysis of section 4.2, each chain in I only need $3d$-bit storage, so the number of points in the first column is:$m_1^I = m$, the relationship between m_i^I and m_{i+1}^I for $1 \leq i \leq t$ is:

$$m_{i+1}^I = N(1 - (1 - 1/N)^{m_i^I}), \tag{11}$$

then the success probability for a single table of I is:

$$P(I) = 1 - (1 - (\frac{m_1^I}{N}))(1 - (\frac{m_2^I}{N})) \cdots (1 - (\frac{m_t^I}{N})). \tag{12}$$

Likewise, let m_i^{II} denotes the number of different points in i-th column of II and $1 \leq i \leq t$. According to the description of [4], we can get $m_1^{II} = 1, m_2^{II} = m - 1$ and the relationship between m_i^I and m_{i+1}^I for $i \geq 2$ is:

$$m_{i+1}^{II} = N(1 - (1 - 1/N)^{m_i^{II}}), \tag{13}$$

then the success probability for a single table of II is:

$$P(II) = 1 - (1 - (\frac{m_1^{II}}{N}))(1 - (\frac{m_2^{II}}{N})) \cdots (1 - (\frac{m_t^{II}}{N})). \tag{14}$$

It is obvious that:

$$\begin{gathered} m/N > 1/N \\ \Rightarrow 1 - (m/N) < 1 - (1/N) \end{gathered}$$

means:

$$1 - (m_1^I/N) < 1 - (m_1^{II}/N). \tag{15}$$

While

$$\begin{aligned} m_2^I &= N(1 - (1 - 1/N)^{m_1^I}) \\ &= N(1 - (1 - 1/N)^m) \end{aligned}$$

so

$$1 - (m_2^I/N) = (1 - 1/N)^m. \tag{16}$$

While

$$1 - (m_2^{II}/N) = 1 - (m - 1)/N = 1 - \frac{m}{N} + \frac{1}{N}. \tag{17}$$

The *taylor* expansion of equation (16) is:

$$1 - \frac{m}{N} + \frac{m(m-1)}{2}\frac{1}{N^2} - \frac{m(m-1)(m-2)}{6}\frac{1}{N^3} + O(\frac{1}{N^4}), \tag{18}$$

where $O(\frac{1}{N^4})$ can be ignored from equation (18), then the result of (17)-(16) is:

$$\begin{aligned}
&\tfrac{1}{N} - \tfrac{m(m-1)}{2}\tfrac{1}{N^2} + \tfrac{m(m-1)(m-2)}{6}\tfrac{1}{N^3} \\
&= \tfrac{1}{N} - \tfrac{1}{N^2}(\tfrac{m^2-m}{2})(1 - \tfrac{m-2}{3N}) \\
&= \tfrac{1}{N} - \tfrac{1}{N}(\tfrac{m^2}{2N} - \tfrac{m}{2N})(1 - \tfrac{m-2}{3N}) \\
&> \tfrac{1}{N} - \tfrac{1}{N}(\tfrac{m^2}{2N} - \tfrac{m}{2N}) \\
&> \tfrac{1}{N} - \tfrac{1}{N} \\
&= 0
\end{aligned}$$

so, we know:

$$1 - (m_2^I/N) < 1 - (m_2^{II}/N), \tag{19}$$

then we can get:

$$m_2^I > m_2^{II}. \tag{20}$$

When $i \geq 2$, from equation (11),(13) and (20), we can get:

$$m_i^I > m_i^{II}, \tag{21}$$

then, we know

$$1 - (m_i^I/N) < 1 - (m_i^{II}/N), \tag{22}$$

succeed for all $i > 2$, so from equation (15),(19) and (22), we can know that:

$$\begin{aligned}
&(1 - (m_1^I/N))(1 - (m_2^I/N)) \cdots (1 - (m_i^I/N)) \\
&< (1 - (m_1^{II}/N))(1 - (m_2^{II}/N)) \cdots (1 - (m_t^{II}/N))
\end{aligned}$$

this means:

$$1 - P(I) < 1 - P(II), \tag{23}$$

then the conclusion is:

$$P(I) > P(II).$$

From the conclusion, taking into the consideration of collision, our design is better than Thing's design in success probability. While Thing's design has higher success probability than rainbow attack, so our design also has higher success probability than rainbow attack.

Storage Usage Analysis: For a single table, provided that both I and II have the same precomputation complexity of mt. Denote the memory usage of I and II is M_I and M_{II} separately, then:

Proposition 3. *The memory usage of a single table of I is lower than the memory usage of a single table of II, namely $M_I < M_{II}$.*

Proof. For design I, the average chain length is t, so the it only need m entries and each entry's size is $|N|$, namely $M_I = m$.

While for design II, there are altogether k chains except for the first chain, so we can get:

$$t + k(t-1) = mt, \tag{24}$$

then, the value of k is:

$$k = \frac{mt - t}{t - 1}. \tag{25}$$

So the memory requirements for design II is:

$$M_{II} = k + 2 = \frac{mt + t - 2}{t - 1}. \tag{26}$$

There are $k + 2$ entries and each entry's size is $|N|$. Then we prove $M_I < M_{II}$. From $m > 1$ and $t > 1$, we can easily get:

$$\begin{aligned} & t + m - 2 > 0 \\ \Rightarrow\ & mt + t + m - 2 > mt \\ \Rightarrow\ & mt + t - 2 > mt - m \\ \Rightarrow\ & m < \tfrac{mt+t-2}{t-1} \\ \Rightarrow\ & M_I < M_{II} \end{aligned}$$

From the conclusion,our design is better than Thing's design in memory usage. While Thing's design has lower storage usage than rainbow attack, so our design also has lower storage usage than rainbow attack.

***Online* Time Analysis:** This section will compare the *online* time complexity of I and II in terms of operation times of function evaluation and table look-up. Time complexity analysis of I has been discussed in section 4.2, while Thing did not give the *online* time complexity analysis of II in [4]. So, here we give the time complexity analysis of design II.

From analysis of section 4.2, we know that the memory requirement of design I is mt entries and each entry contains $|N| = 3d$ bits where $d = \frac{1}{3}\log_2 N$ and $m = t = N^{1/3}$. By using this size of storage, according to [4], the optimal value of k is $mt - 2$ in design II. Then the number of chains in II is $k + 1 = mt - 1$. In order to simplify the comparison, provided that the chain length of II is t, same as the average chain length of I.

The times of function evaluation and table look-up in II is $O(\frac{t^2}{2})$ and $O(t)$ separately, same as the rainbow table, but the time for each table look-up can not be ignored because there is no sorting to the saved end point on precompuation stage of II. So in order to find the corresponding index of end point, we need to compare each item one by one and the time for each table look up is thus $O(k + 1) = O(mt - 1) = O(t^2)$.

Here we give the comparison of *online* time complexity of I,II and rainbow attack as follows:

Desgin	I	II	Rainbow attack
Function evaluation	$O(\frac{C_1^2}{2}t^2)$	$O(\frac{1}{2}t^2)$	$O(\frac{1}{2}t^2)$
Table look-up	$O(\frac{C_2^2}{2}t^2)$	$O(t)$	$O(t)$
Time of each table look-up	$O(1)$	$O(t^2)$	$O(2\log_2 t)$

Table 2. *Online* time complexity of design I,II and rainbow attack

From the table, we can see that time complexity of function evaluation of design I is almost the same to design II and rainbow attack. Times of table look-up of design I is larger than that of design II, but the time for each table look-up of design I is $O(1)$, while design II is $O(t^2)$. So design I has lower time complexity of table look-up than that of design II. Besides, time complexity of table look-up of design II is: $O(t^2 * t) = O(t^3) = O(N)$ ($t = 2^d = 2^{1/3 \log_2 N}$). By adding the time complexity of function evaluation, time complexity of design II is no better than that of brute force attack. Even through design I has higher time complexity of table look-up than that of rainbow attack, it save a lot of storage space. So our design is better than Thing's method, and has better performance than rainbow attack in memory limited application.

Our design I can also avoid *merge* by checking end point naturally, because each chain has different pattern of DP. While for Thing's design II, if *merge* occurred, it can not recompuate the merged chain and have to cancel it and save a NULL value because the hash value of the second column increased in order and we must keep it in order for the sake of *online* phase. This will greatly decrease the success probability of the precomputed tables. What's more, no sorting is needed in our design. So our design also resolves the innate problems of Thing's method. Further discussion of our design can be found in Appendix A.

5 Conclusion

In this paper, we carefully analyze the drawback of Thing's time-memory trade-off method and give a new design to improve his method. Through proof, our design has higher success probability, lower storage space and lower *online* analysis time than Thing's methods and fix its innate problems. Furthermore, our design also has higher success probability and lower storage space than rainbow attack, even though the *online* time complexity is larger than rainbow attack, yet it can save a lot of storage. So our design is better than Thing's method, and has better performance than rainbow attack in memory limited application.

References

1. Hellman, M. (1980). "A Cryptanalytic Time-Memory Trade-Off." IEEE TRANSACTIONS ON INFORMATION THEORY 26(4): 401.
2. D.Denning. (1982). "Cryptography and Data Security." p.100. Addison-Wesley.
3. Oechslin, P. (2003). "Making a faster cryptanalytic time-memory trade-off." Advances in Cryptology-CRYPTO 2003: 617-630.
4. Thing, V. L. L. and H. M. Ying (2009). "A novel time-memory trade-off method for password recovery." Digital Investigation 6: S114-S120.
5. Biryukov, A., Shamir, A. (2001). " Cryptanalytic time/memory/data tradeoffs for stream ciphers." In Proceedings of Advances in Cryptology - 7th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'01). vol.2248:1-13.
6. Avoine, G., Junod, P., and Oechslin, P. (2008). "Characterization and improvement of time-memory trade-off based on perfect tables." ACM Transactions on Information and Systems Security, Vol.11, No.4, Article 17.
7. Hoch, J., Shamir, A. (2009). "Parallel Hellman." Invited talk in Tsinghua university for one hundred years anniversary.
8. Hong, J., K. C. Jeong, et al. (2008). "Variants of the distinguished point method for cryptanalytic time memory trade-offs." Springer-Verlag.
9. Avoine, G., Junod, P., Oechslin, P. (2005). "Time-memory trade-offs: False alarm detection using checkpoints." In Proceedings of Progress in Cryptology - 6th International Conference on Cryptology (INDOCRYPT'05). vol.3797:183-196.

A Further Discussion of New Method

From the description of section 4, if a chain length does not lie in the range $[t_{max}, t_{min}]$, we will ignore this chain and save $RT = \{-, 0\}$. This will reduce the rate of coverage and increase *false alarm* occurrences. We can improve this process by making use of the first d-bit of the start point. This idea is same as Hellman+VDP design in [8], but we will further discuss the selection of parameters which [8] did not mentioned.

First, we will briefly introduce the basic idea of the improvement. If the calculation of j-th chain in i-th table failed, we will increment the value of the first d-bit and recompute the current chain. So the start point of j-th chain in i-th table is thus $SP_i^j = (\lambda_i^j | i | j)$ and the storage is changed to $RT_i^j = \{\lambda_i^j | R_i^j | l_i^j\}$, where $0 \leq \lambda_i^j \leq 2^g$ and 2^g is the upper bound of times of recomputation. That means, if all 2^g times of recalculation failed, we have to save $RT = \{-, 0\}$. This improvement will not influence *online* analysis, the only change is that when we find a match, we need to combine λ_i^j, i and j instead of d-bit 0, i and j to be the start point.

A.1 The Selection of g

In this section, we will discuss the optimal value of g.

Proposition 4. *In order to gain a successful recalculation, the lower bound of g is* $\log_2 (1 - (\frac{1}{e})^{c_2})^{-1}$, *namely,* $g \geq \log_2 (1 - (\frac{1}{e})^{c_2})^{-1}$ *where* $c_2 = \frac{t_{max} - t_{min}}{t}$.

Proof. Provided that function F_k $(0 \leq k \leq t_{max} - 1)$ is a random function on precompuation stage. The probability that the output is not a DP after one calculation of F_k is:

$$P = 1 - \frac{1}{2^d} \tag{27}$$

we denote an experiment A :{Finding a corresponding DP in $\bar{t} = t_{max} - t_{min} + 1$ times of calculation}, then the probability that experiment A failed is:(this is also the probability that the calculation of current chain failed)

$$P(A) = P^{\bar{t}} = (1 - \frac{1}{2^d})^{\bar{t}} \approx e^{-(\frac{\bar{t}}{2^d})} \tag{28}$$

then the probability that experiment A succeeded is thus $1 - P(A) = 1 - P^{\bar{t}}$. We can do this experiment for at most 2^g times. If we meet a successful experiment, then the calculation of the current chain succeed. The probability that k-th experiment succeed while all previous $k - 1$ experiments fail is:(Apparently it accords with geometric distribution)

$$P(k) = (P^{\bar{t}})^{k-1}(1 - P^{\bar{t}}) = e^{-(\frac{\bar{t}}{2^d})(k-1)}(1 - e^{-(\frac{\bar{t}}{2^d})}) \tag{29}$$

then in order to gain a successful experiment, the expected times of experiment is:

$$E = \frac{1}{1 - P^{\bar{t}}} = \frac{1}{1 - e^{-(\frac{\bar{t}}{2^d})}} \tag{30}$$

then we can expect a successful recalculation of the current chain if $2^g \geq E$, namely:

$$2^g \geq \frac{1}{1 - e^{-(\frac{\bar{t}}{2^d})}} \tag{31}$$

while $\bar{t} = t_{max} - t_{min} + 1 \approx t_{max} - t_{min} = c_2 t$, so we get:

$$g \geq \log_2 (1 - (\frac{1}{e})^{c_2})^{-1} \tag{32}$$

Through experiment, we get:

c_2	$\log_2 (1 - (\frac{1}{e})^{c_2})^{-1}$
1.0	0.661728
1.5	0.364255
2.0	0.209787
2.5	0.123568
3.0	0.073677
3.5	0.044237
4.0	0.026669

Table 3. Lower bound of g

From the table, when c_2 increases(means $t_{max} - t_{min} = c_2 t$ becomes larger), the lower bound of g becomes lower, this is what we expected. Because when $t_{max} - t_{min} = c_2 t$ becomes larger, means the calculated chain becomes longer, then success probability of one experiment becomes higher.

From the discussion above, $g \in [0, d]$ and we often set $c_2 \in [3, 5]$. So if $g = 1$, equation (31) succeed. This means we only need to save 1 bit for each chain to do the improvement and we can get an expected successful calculation in 2 attempts.

A.2 Others

Our new design is different from *distinguished point* scheme proposed by Rivest[2] and rainbow table, because different chains in a single table have different patterns of DP of end points and different lengths. Accordingly, to get a perfect table, *merge* can be detected freely in chains of same length by comparing the end points, while for chains with different lengths, we need to compare the intermediate points of the current chain with the previous saved end points which have the same length to the current calculated point. Additional comparison is thus needed during precomputaion. However, we choose different *reduction* function for each column and this will greatly reduce the probability of occurrence of *merge* like rainbow attack. *Merge* only occur when collision occurred in the same column and collision of different column will not lead to *merge* and the probability of *merge* is only $\frac{1}{t}$ when collision occurred.

In order to reduce the *false alarm*, we can refer to the checkpoint method[9] proposed by Avoine or the parallel operation on *online* phase proposed by shamir[7].