A Faster Cryptanalytic Time-Memory Tradeoff*

Jung Woo Kim
Seoul National University
jkim@theory.snu.ac.kr

Jungjoo Seo
Seoul National University
jjseo@theory.snu.ac.kr

Jin Hong
Seoul National University
jinhong@snu.ac.kr

Kunsoo Park
Seoul National University
kpark@theory.snu.ac.kr

Abstract

There has been extensive research on a cryptanalytic time-memory tradeoff for recent 30 years. Since Hellman's work in 1980, some improved variants and techniques have been proposed, and the rainbow method is known as the best time-memory tradeoff. As for the memory size, however, the required number of bits per start point and end point was not explicitly considered in these works. With this in mind, we propose a new time-memory tradeoff and analyze the expected cryptanalysis time.

1: Introduction

One-way functions are fundamental tools for cryptography, and it is a hard problem to invert them. There are three generic approaches to invert a one-way function such as $C(x) = E_x(P_0)$, where E is any encryption scheme, x is the key of size N, and P_0 is the known fixed plaintext. The simplest approach is an exhaustive search. An attacker tries all possible keys until the pre-image of $C(x)$ is found; however, it needs a lot of time. Another simple approach is a table lookup, in which an attacker precomputes all $(x, C(x))$ pairs and stores in a table in sorted order with respect to the value $C(x)$. The attack can be carried out quickly, but a large amount of memory is needed. Cryptanalytic time-memory trade-offs are compromise solutions between time and memory. It can invert a one-way function in time shorter than an exhaustive search approach using memory smaller than a table lookup approach. The idea of a tradeoff is to use chains of keys. It is achieved by using a reduction function $R(\cdot)$ which creates a key from a ciphertext.

Cryptanalytic time-memory trade-offs consist of two phases: precomputation and online phases. In a precomputation phase, an attacker precomputes sufficiently many $(x, f(x), f(f(x)), \ldots, f^{(t)}(x))$ chains, stores the pairs of start and end points, $(x, f^{(t)}(x))$, out of them in a table, and sorts the list with respect to the end points $f^{(t)}(x)$, where $f(x) := R(E_x(P_0))$. This concise table is used to find the key x in time shorter than an exhaustive search in an online phase.

Cryptanalytic time-memory tradeoff [2, 3, 5, 7, 16, 19, 14, 17, 20, 12, 15] was firstly introduced by Hellman in 1980 [9]. Hellman's method consists of several tables, and for each different table, different function f's are used to make chains by taking different reduction

* A preliminary version of this paper appeared in [15].

functions R. Using this method, any N key cryptosystem can be cryptanalyzed in $N^{2/3}$ operations with $N^{2/3}$ words of memory after a precomputation requiring N operations. Since then, there has been an extensive research on time-memory tradeoff to speed up cryptanalysis time. Rivest proposed to apply *distinguished points* technique [6] to Hellman's method which reduces the number of table lookup operations. Avoine et al. introduced a technique detecting false alarms, called *checkpoints* [1]. Using the technique, the cost of false alarms is reduced with a minute amount of memory. Hong et al. proposed *variable distinguished points* method, a variant of the distinguished points technique [11]. In 2003, a new method, which is referred to as *rainbow method*, was suggested by Oechslin [18]. The idea is to use different reduction functions for each column of a table. Thus, the probability that two different chains merge in a table becomes smaller, and one big table covers a large amount of keys. The rainbow method saves a factor of two in the worst case time complexity compared to Hellman's method. Up until now, the rainbow method is the most efficient one.

Many prior works assume that the numbers of bits that are needed to represent start points and end points are equal, so they consider only the number of start points and end points. However, different time-memory tradeoffs consist of different numbers of tables of different sizes, and the number of bits for each point is actually different. Roughly speaking, a rainbow table of $m \cdot t$ chains of length t has the same success probability as t Hellman's tables of m chains of length t. Thus, $\lceil \log_2 m \rceil$ bits per start point are required in a Hellman's table, but $\lceil \log_2 m + \log_2 t \rceil$ bits in a rainbow table. In this paper, we propose a new time-memory tradeoff that improves further the performance of the rainbow method. With simple modification of the table structure, we can reduce the number of bits required to store start and end points of chains, and store more chains under the same amount of memory. Furthermore, our approach introduces more functions in a table, which results in the increase of success probability. We also theoretically analyze the success probability and the expected cryptanalysis time of the tradeoff, and show its performance improvement compared to rainbow method with the intensive experimental results. Our method speeds up the original rainbow method without indexing technique by about 30% and the rainbow method with indexing technique by about 6% for the success probability of 99%.

The rest of the paper is organized as follows. The paper first gives a brief overview of the preliminary work, the rainbow method, in Section 2. Section 3 proposes our time-memory tradeoff. Section 4 analyzes our time-memory tradeoff, and compare the performance of ours with that of the rainbow method. Finally, we conclude in Section 5.

2: Rainbow Method

In this section, we briefly review the rainbow method. Let $C(\cdot)$ be a one-way function such that $C(x) = E_x(P_0)$, where E is any encryption scheme, x is the key, and P_0 is the fixed plaintext. The method aims at finding the key x, given the fixed plaintext P_0 and the corresponding ciphertext $C(x)$.

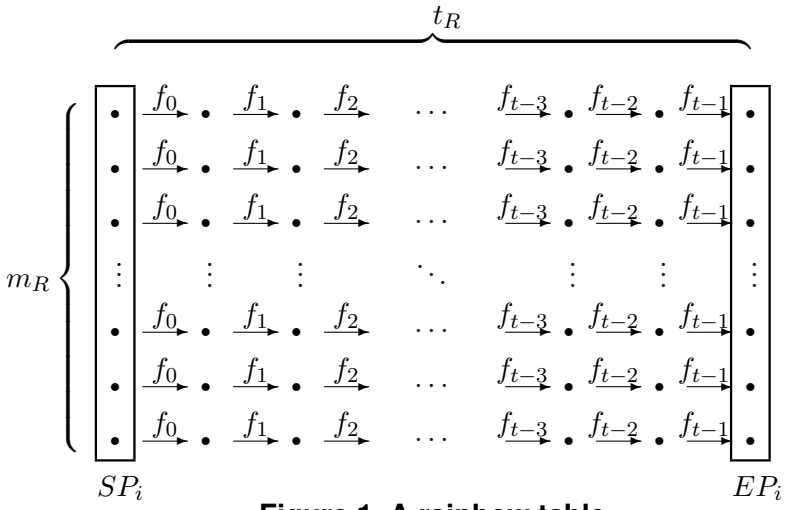

Figure 1. A rainbow table

2.1: Precomputation Phase

In a precomputation phase, we randomly choose m_R keys, or start points, labeled SP_0, SP_1, ... , SP_{m_R-1}.[1] For each $0 \leq i < m_R$, we set

$$x_{i,0} = SP_i,$$

and compute

$$x_{i,j} = f_{j-1}(x_{i,j-1}), 1 \leq j \leq t$$

recursively, where $f_j = R_j \circ C$ and $R_j(x)$ is a reduction function that maps a ciphertext to an element in the key space by simply dropping and permuting some bits of x. In other words, m_R chains of length t are produced starting from SP_i $(0 \leq i < m_R)$ as shown in Figure 1. The last element $x_{i,t}$ for each i-th chain is called an end point (EP_i). The pairs of the start and end points, (SP_i, EP_i), are stored in a table, and are sorted with respect to the end points. Note that all intermediate points are discarded to reduce memory requirements.

2.2: Online Phase

In an online phase, given an image $y_0 = C(x_0)$, we try to invert the one-way function $C(\cdot)$ to find the key x_0, the preimage of $C(\cdot)$, by generating online chains that start from y_0. First, we generate the online chain of length one by computing $y_1 = R_{t-1}(y_0) = f_{t-1}(x_0)$ and check whether it is an end point on the table. If so and $y_1 = EP_i$, that means x_0 is next to EP_i in Figure 1 or EP_i has more than one inverse image. The latter case is referred to as a *false alarm*. Therefore, we compute $x_{i,t-1}$ starting from SP_i, and check whether it is a false alarm or not by computing $C(x_{i,t-1}) = y_0$. If $y_1 \neq EP_i$ or a false alarm occurred, then we compute $y_2 = f_{t-1}(R_{t-2}(y_0))$, the online chain of length two, and check whether it is an end point. The above process is repeated until the key x_0 is found

[1]To reduce the memory size of the table, we fix the start points to 0 through $m_R - 1$ instead of using randomly chosen keys. That is, $SP_0 = 0, SP_1 = 1, \cdots, SP_{m_R-1} = m_R - 1$. Then, the memory size for each start point is $\lceil \log_2 m_R \rceil$, not $\lceil \log_2 N \rceil$.

or all t online chains fail to invert the given image y_0. For example, at the i-th iteration, $y_i = f_{t-1}(f_{t-2}(\cdots(f_{t-i}(x_0))\cdots))$ is computed and checked.

To achieve high success probability, multiple tables can be used. The online phase with the multiple tables is done by searching for the key in the last column of each table and then searching sequentially through previous columns of all tables. Assume that ℓ_R is the number of rainbow tables, and then ℓ_R chains of length i are generated at the i-th iteration, which requires $(i-1)\cdot\ell_R$ encryptions.

2.3: Analysis

We summarize the analysis results [13] of the *non-perfect* rainbow table. We present not only the success probability and the expected time of the table but also the selection of optimal parameters. These results will be used to estimate the performance of the rainbow method when comparing the performance between the rainbow method and ours in Section 4.2.

Theorem 1. *The success probability of the rainbow method is*

$$1-\left(\frac{2}{2+c}\right)^{2\cdot\ell_R}$$

where $m_R\cdot t_R = c\cdot N$.

Theorem 2. *The expected time of the rainbow method with false alarms is*

$$T_R = t_R^2\ell_R\frac{\left(\begin{array}{l}\big\{(2\ell_R-1)+(2\ell_R+1)c\big\}(2+c)^2\\ -4\big\{(2\ell_R-1)+\ell_R(2\ell_R+3)c\big\}\big(\frac{2}{2+c}\big)^{2\ell_R}\end{array}\right)}{(2\ell_R+1)(2\ell_R+2)(2\ell_R+3)c^2}$$

where $m_R\cdot t_R = c\cdot N$.

Theorem 3. *Let* $0<P_R<1$ *be any given fixed value.*

$$c = 2\big\{(1-P_R)^{-\frac{1}{2\ell_R}}-1\big\}.$$

Then the rainbow method that uses any parameters m *and* t *satisfying the relation*

$$m_R\cdot t_R = c\cdot N$$

attains the given value P_R *as its probability of success. Then the expected time is*

$$T_R = \frac{t_R^2\ell_R}{(2\ell_R+1)(2\ell_R+2)(2\ell_R+3)c^2}\left(\begin{array}{l}\big\{(2\ell_R-1)+(2\ell_R+1)c\big\}(2+c)^2\\ -4\big\{(2\ell_R-1)+\ell_R(2\ell_R+3)c\big\}(1-P_R)\end{array}\right).$$

Given a success probability, the optimal number of tables, ℓ_R, is shown in [13]. The value ℓ_R only depends on the success probability. For the success probability of 70%, 90% and 99%, the optimal numbers of tables are 1, 3 and 6, respectively.

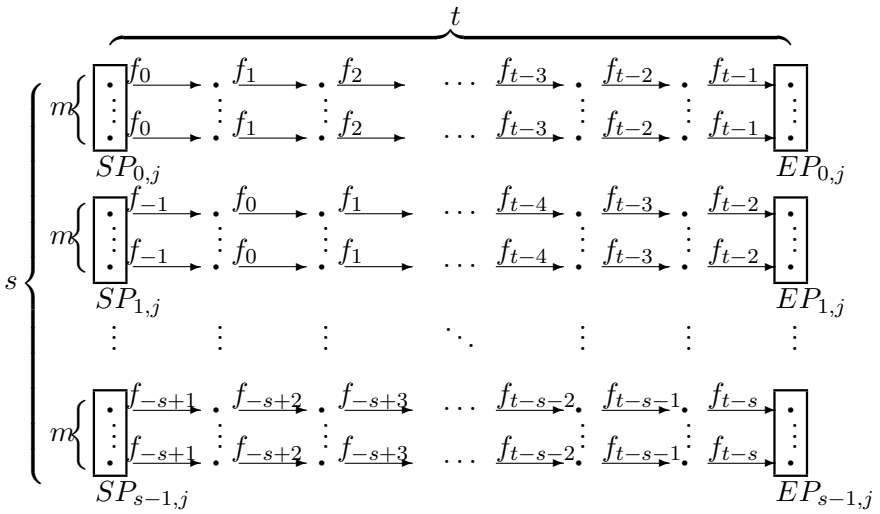

Figure 2. Our table

3: A Faster Time-Memory Tradeoff

In this section, we propose our time-memory tradeoff, which introduces more functions and spends less memory for each start point compared to the rainbow method by modifying the structure of the table in Figure 1. We divide the rainbow table into s sub-tables, each of which has m chains. The order of the computation of function $f(\cdot)$ is shifted to the right by one position in each sub-table. For example, the chain of the first sub-table is computed starting with $f_0(\cdot)$ and ending with $f_{t-1}(\cdot)$. The second sub-table is computed starting with $f_{-1}(\cdot)$ and ending with $f_{t-2}(\cdot)$. Our time-memory tradeoff table is shown in Figure 2.

In a precomputation phase, we first fix $m \cdot s$ start points, labeled $SP_{i,j}(0 \leq i < s, 0 \leq j < m)$, where $SP_{i,j} = j$. $SP_{i,j}$ represents the start point at the j-th chain in the i-th sub-table. For each $0 \leq i < s$ and $0 \leq j < m$, we set

$$x_{i,j,0} = SP_{i,j},$$

and compute

$$x_{i,j,k} = f_{k-i}(x_{i,j,k-1}) \text{ for } 1 \leq k \leq t,$$

recursively. In each sub-table, the pairs of start and end points are stored in sorted order with respect to the end points. In other words, s different sub-tables with m chains are generated starting from $SP_{i,j}(0 \leq i < t, 0 \leq j < m)$, and the pairs of the start and end points, $(SP_{i,j}, EP_{i,j})$, are stored in sorted order within a sub-table.

In an online phase, given an image $y_0 = C(x_0)$, we want to find x_0, that is, invert the one-way function $C(\cdot)$. For the j-th iteration, let $y_{i,j}$ be the end point of the online chain corresponding to the i-th sub-table. First, we compute $y_{1,1} = R_{t-1}(y_0) = f_{t-1}(x_0)$ and check whether it is an end point in the first sub-table. If so, we check whether it is a false alarm. If it is not an end point or the false alarm occurs, then we compute $y_{1,j} = R_{t-j}(y_0) = f_{t-j}(x_0)$ and check the false alarms for $j = 2, 3, \cdots, s$, sequentially. Next, to find the key in the previous column, i.e., at the second iteration, we compute

$$y_{2,j} = \begin{cases} f_{t-j}(y_{1,j+1}) & \text{for } j = 1, 2, \cdots, s-1 \\ f_{t-s}(R_{t-s-1}(y_0)) & \text{for } j = s \end{cases}$$

and check the false alarms. Note that the value, $y_{1,j}$, which is computed in the first iteration, is reused to generate the online chain in the second iteration for $j = 1, 2, \cdots, s-1$. In the i-th iteration, we compute

$$y_{i,j} = \begin{cases} f_{t-j}(y_{i-1,j+1}) & \text{for } j = 1, 2, \cdots, s-1 \\ f_{t-s}(f_{t-s-1}(\cdots(f_{t-s-i+2}(R_{t-s-i+1}(y_0)))\cdots)) & \text{for } j = s \end{cases}$$

and check the false alarms. To achieve high success probability, multiple tables can also be used in our method. Let ℓ be the number of our tables. Then the worst case time to create the online chain is $\ell \cdot \sum_{k=1}^{t-1}(k + (s-1)) \approx \ell \cdot (\frac{1}{2}t^2 + (s - \frac{3}{2})t)$.[2]

4: Performance Analyses and Comparisons

In this section, we analyze our method explained in Section 3, and compare the performance of ours with the rainbow method. The analysis includes the success probability, the expected cryptanalysis time with false alarm costs, and the memory size of the table.

4.1: Analyses

Let us denote the j-th column in the i-th sub-table by the *(i, j)-th sub-column*. The table has $s \times t$ sub-columns. We write $m_s(i, j)$ for the number of new points added by the (i, j)-th sub-column in the sub-columns to which the same f function is applied, where $0 \le i < s$ and $0 \le j < t$. We first count the number of distinct points in each sub-column, and then analyze the success probability at each iteration. Based on these, the expected time will be calculated.

Lemma 4. *$m_s(i, j)$ satisfies the following recurrence relation.*

$m_s(i, j) =$

$$\begin{cases} m & \text{if } i = s-1 \text{ and } j = 0 \\ g(m_s(i, j-1)) & \text{if } i = s-1 \text{ and } j \neq 0 \\ m \cdot \left(1 - \dfrac{\sum_{x=1}^{s-1-i} m_s(i+x, x)}{N}\right) & \text{if } i \neq s-1 \text{ and } j = 0 \\ g(m_s(i, j-1)) \cdot \left(1 - \dfrac{\sum_{x=1}^{\substack{min(s-1-i,\\ t-1-j)}} m_s(i+x, j+x)}{N}\right) & \text{otherwise} \end{cases}$$

where $g(x) = N \cdot (1 - e^{-\frac{x}{N}})$.

Proof. We count $m_s(i, j)$ in the order from bottom to top and from left to right. First, we set $m_s(s-1, 0)$ to m. The second equation represents the number of points in the bottom sub-table, i.e., $m_s(s-1, j)$ for $j = 1, \cdots, t-1$. It can be easily computed by the

[2]The online phase with multiple tables searches for the key from the last column to the first column of each table in column-major order, as in the rainbow method.

recurrence relation $m_s(s-1,j) = N(1-(1-\frac{1}{N})^{m_s(s-1,j-1)})$, which can be approximated by $m_s(s-1,j) = N(1-e^{-\frac{m_s(s-1,j-1)}{N}})$ [8].

For the rest part, we have to consider the number of distinct points to which the same f function is applied. That is, for $i \neq s-1$, the points in the (i,j)-th sub-column should be distinct to the points in all $(i+x, j+x)$-th sub-columns for $x > 0$, which leads to the third and fourth equation. □

Once we figure out the number of distinct points, the success probabilities of a single table and of ℓ tables can be easily calculated. We first prove the probability to fail until the k-th iteration.

Lemma 5. *The probability to fail until the k-th iteration is*

$$P_{funtil}(k) = \prod_{i=t-k}^{t-1}\left(1-\frac{D(0,i)}{N}\right)\cdot\prod_{i=1}^{s-1}\left(1-\frac{D(i,t-k)}{N}\right)$$

where $D(r,c) = \sum_{i=0}^{min(t-r-1,s-r-1)} m_{s-r-1}(r+i,c+i)$ *and* $\leq k \leq t$.

Proof. Considering m_s as a two-dimensional table, each element $m_s(i,j)$ is the number of distinct points in the sub-column (i,j) with respect to the diagonal where the element belongs to, i.e. sub-columns $(i+x, j+x)$ where $0 \leq x \leq min(s-i-1, t-j-1)$. The function $D(r,c)$ calculates the total number of distinct points in the diagonal that begins from (r,c). The failure until the k-th iteration means that the preimage does not exist in the k rightmost columns. The first product in the equation is the probability of failure of the diagonals from $(0,t-k)$ to $(0,t-1)$, and the second product is of the diagonals from $(1,t-k)$ to $(s,t-k)$. Thus, the total product yields the probability to fail until k-th iteration. □

Theorem 6. *The success probability* P_1 *of a single table of* $s \times t$ *sub-columns is*

$$P_1 \quad = 1 - P_{funtil}(t) = 1 - \prod_{i=0}^{t-1}\left(1-\frac{D(0,i)}{N}\right)\cdot\prod_{i=1}^{s-1}\left(1-\frac{D(i,0)}{N}\right)$$

Proof. The proof is straightforward. □

Collorary 7. *The success probability* P *of* ℓ *tables is*

$$P = 1-(1-P_1)^{\ell}$$

Proof. The proof is straightforward. □

Theorem 8. *Let* z *be the number of truncated bits in the end point.*[3] *The expected time of our tradeoff with false alarms is*

$$T = \ell\cdot\left((t-1)\left(\frac{ms}{2^r}+\frac{ms}{N}\right)+\sum_{k=2}^{t}\left(k+s-2+(t-k)\left(\frac{ms}{2^r}+\frac{msk}{N}\right)\right)\cdot P_{funtil}(k-1)\right),$$

where $r = \lceil \log_2 N \rceil - z$.

[3]End point truncation technique is the method that simply truncates a part of the end point before storage. [13]

Proof. The k-th iteration is processed with probability $P_{funtil}(k-1)$. The probability of alarm associated with a single chain in a row of a table at the k-th iteration is $(k+1)/N$, and the probability of the false match due to the end point truncation is $1/2^r$ where r is the number of the bits in the end point after the truncation [10]. Hence, considering ms chains in a single table, the cost of false alarms and truncation at the k-th iteration is $(t-k)\cdot(ms\,(k+1)\,/N - ms/N + ms/2^r)$. The first term $(t-1)\cdot(ms/N + ms/2^r)$ is the number of encryptions at the first iteration. Multiplying the number of tables to the summation of the expected number of encryptions at each iteration yields the formula. □

Theorem 9. *The memory size of the table is*

$$M = ms\ell\left(\lceil\log_2 m\rceil + \lceil\log_2 N\rceil - \left\lceil\log_2 \frac{m}{\lceil\log_2 m\rceil}\right\rceil - z\right) + s\ell\cdot 2^{\left\lceil\log_2 \frac{m}{\lceil\log_2 m\rceil}\right\rceil}\lceil\log_2 m\rceil.$$

Also, the memory size of the table without indexing technique is

$$M = ms\ell\left(\lceil\log_2 m\rceil + \lceil\log_2 N\rceil - z\right).$$

Proof. Start points can be chosen as consecutive integers which can be represented in $\lceil\log_2 m\rceil$ bits. Hence, we only need to store the $\lceil\log_2 m\rceil$ lower bits per start point. To store an end point, $\lceil\log_2 N\rceil$ bits are required, as suggested in [4]. Also, index file technique can be applied for reducing storage. After sorting the table on the end points, one can store the most significant x bits of the end points in an index file, where the number of the entries of the index file is 2^x. The i $(0 \le i \le 2^x - 1)$ -th entry of the index file points to the starting position of the end points, the first x bits of which is i. Using this technique, $\lceil\log_2 \frac{m}{\lceil\log_2 m\rceil}\rceil$ bits can be removed. By truncating z more bits, $\lceil\log_2 m\rceil + \lceil\log_2 N\rceil - \lceil\log_2 \frac{m}{\lceil\log_2 m\rceil}\rceil - z$ bits are required to store a pair of start and end points. The first term is obtained with $m\cdot\ell\cdot s$ rows taken into account. The second term is the required space for the index file. Given the number of index bits x and entry size e, the space for the index file is $2^x e$. s and l are multiplied since there are s index files in each tables. □

4.2: Comparisons

In this section, we compare the performance of ours with that of the rainbow method. The success probability P and the expected time T of ours are calculated from the equations in Section 4.1, and those of the rainbow method (i.e., P_R and T_R, respectively) are presented in Section 2.3.

The success probability and the expected time varies with the number of sub-table, s. As shown in Figure 3, the success probability increases as s grows because new different functions are introduced in the table.[4] Also, the most interesting result to emerge from the figure is that some increasing steps happen. These steps happen when the required number of bits per start point, $\lceil\log_2 m\rceil$, decreases by one, so that more numbers of points can be stored under the same memory amount. On the other hand, we can see that the expected time linearly increases. In the expected time graph some small decreasing steps occur at the same s in which the increasing steps happen in the success probability graph. This is because the $\prod P_{failat}(x)$ factor in Theorem 8 is rapidly decreased at that point.

[4]Our table with $s = 1$ has the same structure to the rainbow table, the success probability of which is 90% in Figure 3.

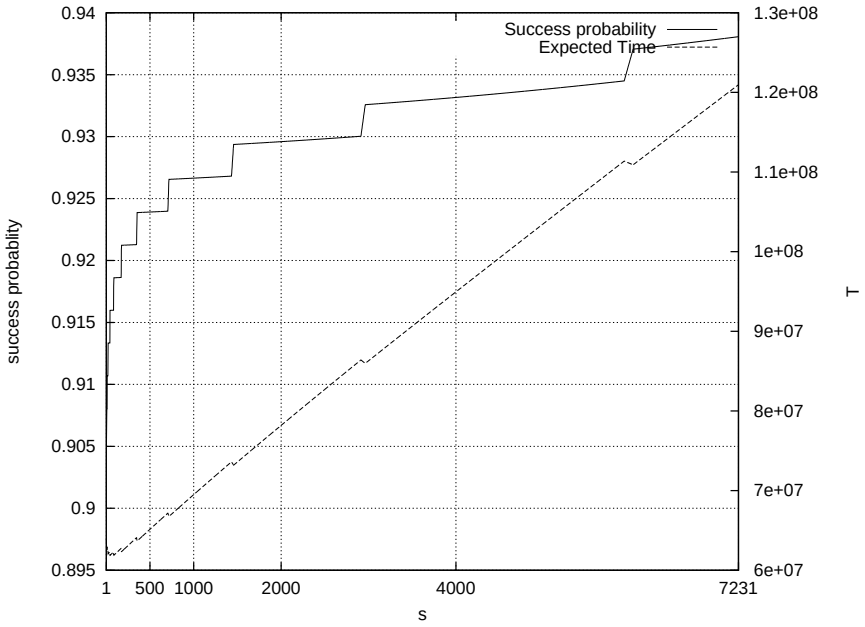

Figure 3. Success probability and expected time w.r.t s **(**$\ell = 3$**,** $t = 10,379$**,** $M =$ 8 GB**,** $N = 3.58 \times 10^{12}$**)**

To compare the performances of tradeoffs, the expected time under a same success probability and memory size should be calculated. To make the success probability of our method equal to that of the rainbow method for $s > 1$, we can adjust the chain length t.[5]

Figures 4 and 5 show the chain length t and the expected time T w.r.t s under the same success probability of 90%. Our table with $s = 1$ has the same structure to the rainbow table, i.e., $t = t_R$ and $T = T_R$ with $s = 1$. As can be seen from Figures 4 and 5, some decreasing steps of t can be seen exactly when the number of bits per pair of start and end point is decreased by 1. The expected time also decreases together. For small s, the number of bits per pair of start and end point frequently decreases, and the performance of our method gets better than that of the rainbow method. Our method speeds up the original rainbow method without indexing technique by 24% at $s = 429$ under the success probability of 90% and memory size of 4 GB. Also, our method speeds up the rainbow method with indexing technique [1, 13] by 5.8% at $s = 156$ under the same success probability of 90% and memory size of 1 GB.

Now we present our experimental results for $N = 3.58 \times 10^{12}$ with various success probabilities and memory sizes. Table 1 and 2 show the performance of the rainbow and our method without and with indexing technique, respectively. We can see that the new method accelerates the cryptanalysis time by about 30% without indexing technique and 6% with that for the success probability of 99%.

Interestingly, our method gets better performance for high success probabilities and small memory sizes. As the chain length t is larger, the ratio of $(s - \frac{3}{2})t$ to $\frac{1}{2}t^2 + (s - \frac{3}{2})t$ gets smaller. Hence, the larger the chain length t is, the slower the time increases w.r.t s. That is, the slopes of the time graphs in Figures 4 and 5 become smaller so that our method gets better performance.

[5]The optimal number of truncated bits for ours is equal to that for rainbow method, i.e., $z = z_R$.

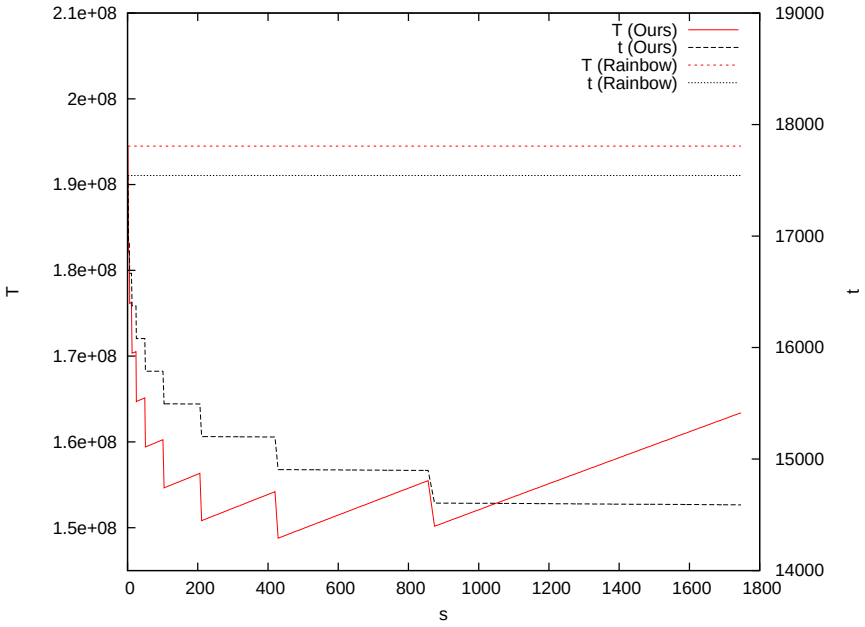

Figure 4. Expected time and chain length w.r.t s without indexing technique under the same success probability of 90% and the memory size of 4 GB ($N = 3.58 \times 10^{12}$)

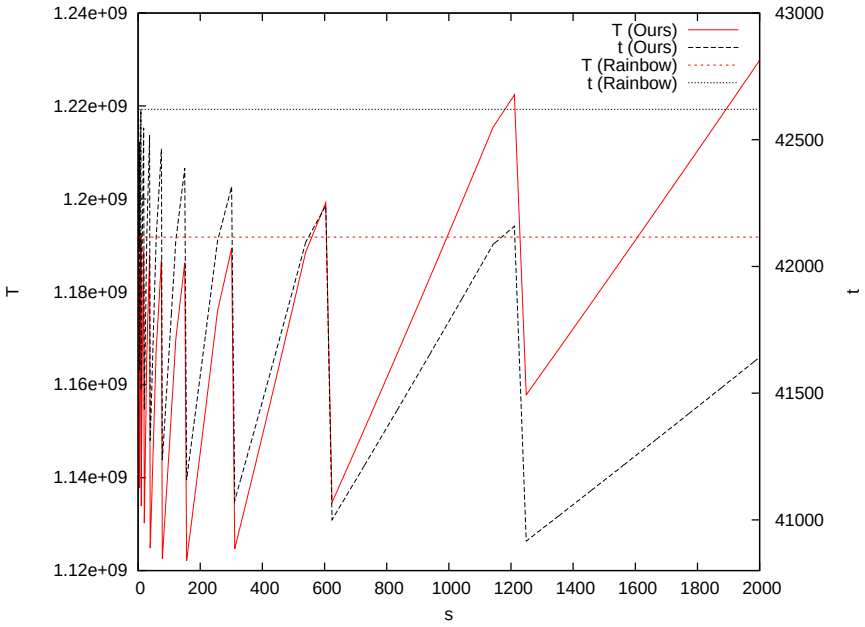

Figure 5. Expected time and chain length w.r.t s with indexing technique under the same success probability of 90% and the memory size of 1 GB ($N = 3.58 \times 10^{12}$)

5: Conclusion

In this paper, we proposed a new time-memory tradeoff faster than the rainbow method. We also theoretically analyzed the success probability and the expected cryptanalysis time

Table 1. Optimal parameters and expected time without indexing technique ($N = 3.58 \times 10^{12}$)

Rainbow						
P_R (%)	M (GB)	ℓ_R	m_R	t_R	z_R	T_R
70	1	1	145592111	40601	11	728793122
	2	1	281637199	20988	10	194401390
	4	1	545392672	10838	9	51749926
90	1	3	51130563	65495	12	2722096848
	2	3	98734880	33917	11	728410980
	4	3	190887435	17543	10	194481028
99	1	6	26030104	128652	12	8588411884
	2	6	50233535	66665	11	2301841797
	4	6	97061407	34502	10	615490109

Ours								
P (%)	M (GB)	ℓ	s	m	t	z	T	Gain
70	1	1	1366	131008	32977	11	510507132	30%
	2	1	643	523888	17522	10	142819054	27%
	4	1	619	1047329	9092	9	39709810	23%
90	1	3	1942	32764	52601	12	1893408147	30%
	2	3	911	130960	28056	11	532429675	27%
	4	3	429	523481	14905	10	148775053	24%
99	1	6	4065	8190	100520	12	5792355136	33%
	2	6	1900	32761	53771	11	1634110946	29%
	4	6	892	131020	28641	10	458046888	26%

of the tradeoff, and showed its performance improvement compared to rainbow method with the intensive experimental results. Our method speeds up the original rainbow method without indexing technique by about 30% and the rainbow method with indexing technique by about 6% for the success probability of 99%.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 20120006492 and 20110005764).

References

[1] Gildas Avoine, Pascal Junod, and Philippe Oechslin. Time-memory trade-offs: False alarm detection using checkpoints. In *INDOCRYPT*, pages 183–196, 2005.

[2] Elad Barkan, Eli Biham, and Adi Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *CRYPTO*, pages 1–21, 2006.

[3] Alex Biryukov, Sourav Mukhopadhyay, and Palash Sarkar. Improved time-memory trade-offs with multiple data. In *Selected Areas in Cryptography*, pages 110–127, 2005.

[4] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of a5/1 on a pc. In *FSE*, pages 1–18, 2000.

Table 2. Optimal parameters and expected time with indexing technique ($N = 3.58 \times 10^{12}$)

Rainbow						
P_R (%)	M (GB)	ℓ_R	m_R	t_R	z_R	T_R
70	1	1	232084821	25470	11	296375323
	2	1	451171078	13101	10	78234429
	4	1	877713300	6734	9	20624011
90	1	3	78573294	42620	12	1191777040
	2	3	152548389	21952	11	315307951
	4	3	296397482	11298	10	83304399
99	1	6	26030104	128652	12	8588411884
	2	6	76390703	43838	11	1021700547
	4	6	148425460	22562	10	270056536

Ours								
P (%)	M (GB)	ℓ	s	m	t	z	T	Gain
70	1	1	113	2093274	24990	11	287244708	3.1%
	2	1	55	8341739	12881	10	76088679	2.7%
	4	1	27	32977137	6644	9	20185296	2.1%
90	1	3	156	521528	41160	12	1122116597	5.8%
	2	3	76	2074222	21242	11	297859812	5.5%
	4	3	37	8263571	10958	10	78989254	5.2%
99	1	6	156	261246	82169	12	3622527989	6.1%
	2	6	151	523992	42322	11	962256822	5.8%
	4	6	37	4139067	21866	10	255169155	5.5%

[5] Johan Borst, Bart Preneel, and Joos Vandewalle. On the time-memory tradeoff between exhaustive key search and table precomputation. In *Proc. of the 19th Symposium in Information Theory in the Benelux, WIC*, pages 111–118, 1998.

[6] Dorothy E. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982. p.100.

[7] Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 1999.

[8] Philippe Flajolet and Andrew M. Odlyzko. Random mapping statistics. In *IN ADVANCES IN CRYPTOLOGY*, pages 329–354. Springer Verlag, 1990.

[9] M. Hellman. A cryptanalytic time-memory trade-off. *Information Theory, IEEE Transactions on*, 26(4):401 – 406, July 1980.

[10] Jin Hong. The cost of false alarms in Hellman and rainbow tradeoffs. *Des. Codes Cryptography*, 57(3):293–327, 2010.

[11] Jin Hong, Kyung Chul Jeong, Eun Young Kwon, In-Sok Lee, and Daegun Ma. Variants of the distinguished point method for cryptanalytic time memory trade-offs. In *ISPEC*, pages 131–145, 2008.

[12] Jin Hong, Ga Won Lee, and Daegun Ma. Analysis of the parallel distinguished point tradeoff. In *INDOCRYPT*, pages 161–180, 2011.

[13] Jin Hong and Sunghwan Moon. A comparison of cryptanalytic tradeoff algorithms. *Journal of Cryptology*. To appear.

[14] Jin Hong and Palash Sarkar. New applications of time memory data tradeoffs. In *ASIACRYPT*, pages 353–372, 2005.

[15] Jung Woo Kim, Jungjoo Seo, Jin Hong, and Kunsoo Park. A faster cryptographic time-memory tradeoff. In *ISA*, pages 168–171, 2012.

[16] Koji Kusuda and Tsutomu Matsumoto. Optimization of time-memory trade-off cryptanalysis and its application to DES, FEAL-32 and Skipjack. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E79-A(1):35–48, 1996.

[17] Sourav Mukhopadhyay and Palash Sarkar. Application of LFSRs in time/memory trade-off cryptanalysis. In *WISA*, pages 25–37, 2006.

[18] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *CRYPTO*, pages 617–630, 2003.

[19] François-Xavier Standaert, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. A time-memory tradeoff using distinguished points: New analysis & FPGA results. In *CHES*, pages 593–609, 2002.

[20] Wenhao Wang, Dongdai Lin, Zhenqi Li, and Tianze Wang. Improvement and analysis of VDP method in time/memory tradeoff applications. In *ICICS*, pages 282–296, 2011.