Analysis of the Parallel Distinguished Point Tradeoff*

Jin Hong[1], Ga Won Lee[1], and Daegun Ma[2]

[1] Seoul National University, Seoul, Korea
{jinhong,gwlee87}@snu.ac.kr
[2] Konkuk University, Seoul, Korea
ma.daegun@gmail.com

Abstract. Cryptanalytic time memory tradeoff algorithms are tools for quickly inverting one-way functions and many consider the rainbow table method to be the most efficient tradeoff algorithm. However, it was recently announced, mostly based on experiments, that the parallelization of the perfect distinguished point tradeoff algorithm brings about an algorithm that is 50% more efficient than the perfect rainbow table method. Motivated by this claim, we provide an accurate theoretic analysis of the parallel version of the non-perfect distinguished point tradeoff algorithm.
Performance differences between different tradeoff algorithms are usually not very large, but even these small differences can be crucial in practice. So we take care not to ignore the side effects of false alarms while analyzing the online time complexity of the parallel distinguished point tradeoff algorithm. Our complexity results are used to compare the parallel non-perfect distinguished point tradeoff against the non-perfect rainbow table method. The two algorithms are compared under identical success rate requirements and the pre-computation efforts are taken into account. Contrary to our anticipation, we find that the rainbow table method is superior in typical situations, even though the parallelization did have a positive effect on the efficiency of the distinguished point tradeoff algorithm.

Keywords: time memory tradeoff, parallel distinguished point, distinguished point, rainbow table

1 Introduction

Cryptanalytic time memory tradeoff algorithms are tools for quickly inverting one-way functions with the help of pre-computed tables. By changing the algorithm parameters, one is able to balance the size of the stored pre-computed table against the average time required for each inversion. The tradeoff techniques are typically used to recover passwords from the information of password hashes and there are commercial implementations which allow one to defeat the

* This paper is to be (was) presented at INDOCRYPT 2011.

access control mechanisms that are embedded in widely used document file formats. The tradeoff technique is not only used among hackers, but is also an important tool for the law enforcement agencies.

It has long been known that the tradeoff attacks can be prevented by designing the security system to incorporate what are referred to as random *salts*, but there still are many systems in use today that do not incorporate this countermeasure, and the only obstacle in applying the tradeoff attacks to these systems is the large pre-computation requirement. However, as was shown in [12], even this barrier is lowered when the security ultimately relies on human generated passwords. Hence, many current security systems are susceptible to the tradeoff attacks, and finding the exact capabilities and limitations of the tradeoff algorithms remains an interesting subject of study.

The first explicit time memory tradeoff algorithm was invented by Hellman [8]. Shortly thereafter, the distinguished point (DP) technique was introduced. This idea, attributed to Rivest in [7], greatly reduces the table lookup requirements of Hellman's original algorithm. The tradeoff algorithm most widely known to the public today is the rainbow table method [13]. When the search space size is N, a typical tradeoff algorithm allows storage requirement M and inversion time T to be balanced, subject to the equation $TM^2 \approx N^2$, through the choice of associated parameters. Rough analyses of the tradeoff algorithms show that the *tradeoff coefficient* $\mathtt{X}_{tc} = TM^2/N^2$ is close to 1 for reasonable choices of parameter sets. However, it is different for each algorithm and does change with the choice of the parameter set.

The tradeoff coefficient $\mathtt{X}_{tc}$ is a measure of how efficiently an algorithm balances storage against online time, with a smaller number corresponding to better tradeoff efficiency. In a recent work [11], that builds on the works [1, 10], the value of $\mathtt{X}_{tc}$ was accurately computed for the Hellman, DP, and rainbow tradeoffs, and then the tradeoff coefficients of the three algorithms were compared against each other. Unlike previous attempts, the comparison of [11] took the inversion success rate, pre-computation cost, and the number of bits required to store each table entry fully into account. Their conclusion, in oversimplified terms, was that the classical Hellman and the DP tradeoffs perform comparable to each other and that the rainbow table method outperforms these two. Even though this was in agreement with what many had believed for some time, the performances of the algorithms were shown to be quite close to each other for moderate success rates, justifying the need for such careful analyses before any comparison.

In this work, we give a treatment analogous to [11] of the DP tradeoff variant that processes its multiple pre-computation tables in parallel. We present an accurate theoretic online time complexity analysis of the parallel DP tradeoff and compare its performance with that of the rainbow tradeoff. Clearly, the wall-clock running time of the parallel DP tradeoff will be much shorter than the original serial DP tradeoff, but our focus will be on the combined running time of all processors, which is the measure of algorithm execution complexity that is widely used in cryptology.

The conclusions are that parallelization has a positive effect, not only on the wall-clock running time, but also on the combined processor time of the DP tradeoff. However, for typical range of parameters, the gain in performance obtained is shown to be insufficient to overcome the superiority of the rainbow table method over the DP tradeoff. If the wall-clock running time is more important than the total processor time, depending on the degree of parallelism available, there can be situations where the parallel DP tradeoff is desirable over the rainbow tradeoff. However, if the combined processor time is more important, one should use the rainbow method rather than the parallel DP method.

Our analysis may be seen as theoretic or practically limited in two respects. First, the above mentioned combined processing time may not directly correspond to the real-world cost of running an algorithm, but this issue is clearly outside the scope of this paper. Second, we do not discuss whether implementing the parallel DP tradeoff is practical on various specific platforms. In fact, the demand for online memory by the parallel DP tradeoff is higher than other tradeoff algorithms and this could be critical in certain environments. However, aside from issues that are specific to implementation environments, we are as practical as possible in treating the two compared algorithms, in that the success probabilities, pre-computation costs, and bits required per table entry are taken into account during algorithm comparison.

We acknowledge that parallel processing of tradeoff tables is not a new idea. Distributed key search with a central pre-computation table repository is mentioned in [2, 3] as an application for DP tradeoffs. It was noted in [13] that processing multiple rainbow tables in parallel will reduce the total combined processor time. In fact, the rainbow tradeoff is usually taken to process its tables in parallel, even though the number of its tables is much smaller than that of the DP tradeoff and allows lower degree of straightforward parallelization. In [10] one can find a very rough heuristic argument as to why the classical Hellman tradeoff will not benefit from parallelization. Finally, the work [9, 14] claims that the parallel version of the *perfect*[3] DP tradeoff is twice as efficient as the perfect rainbow table method, mainly based on experiment results.

The current work was motivated by the above mentioned [14], which announced the perfect table parallel DP tradeoff as the "New World Champion" of tradeoff algorithms. However, all the algorithms considered in this paper are the non-perfect table versions. Let us explain our choice to analyze the non-perfect table version of the parallel DP tradeoff rather than the perfect table version.

The perfect DP tradeoff was first studied in [2, 3]. They left some unresolved problems and many view [15] as completing the analysis. However, the analysis of [15] does not provide figures that are accurate enough for use in this paper as data representing the performance of a competing algorithm. For example, an entry in Table 2 of the paper gives 21.6425 and 21.1357 as experimental and corresponding theoretic figures concerning a certain storage count. On the

[3] The use of perfect tables is not stated explicitly in [9, 14], but was clarified to us during private communication. We also learned that short chain length bounds were used in their experiments.

surface, the two values seem to be in good agreement, but one must note that they are presented in logarithmic scale. We cannot explain the details here, but when the two storage figures representing experiment and theory are translated to tradeoff efficiency figures, they become the ratio $\left(\frac{2^{21.6425}}{2^{21.1357}}\right)^2 \approx 2.02$. Theoretic result of such inaccuracy may be acceptable for many applications, but is not appropriate for the purpose of comparing different algorithms, since algorithm performances are likely to differ by a comparable ratio.

A perfect DP table is obtained by removing redundancies present in a non-perfect DP table and this implies that each perfect table requires more pre-computation to construct than the non-perfect version, if the two are to bring about the same success rate. Even though the perfect table DP tradeoff is expected to be more efficient than the non-perfect table version, with the current state of knowledge explained above, it is hard to judge whether the degree of enhancement in efficiency justifies the additional pre-computation involved.

Since the state of current knowledge is far from ready for the treatment of perfect table parallel DP tradeoffs, and since it is unclear if the efficiency advantage of the perfect version will be worth the higher pre-computation cost, the non-perfect table parallel DP tradeoff is treated in this paper. The perfect table parallel DP tradeoff is certainly of interest and is left as a subject of future study, which can be approached only after a more accurate analysis of the perfect table (serial) DP tradeoff has been developed. The initial view of the effects of parallelism we obtain and the method of approaching we develop in dealing with the non-perfect case will be of guidance in studying the perfect case.

The rest of this paper is organized as follows. We fix the basic terminology and recall some previous results in Section 2. The parallel DP tradeoff algorithm is made explicit in Section 3 and its theoretic analysis is given in Section 4. After verifying our theoretic developments with experiments in Section 5, the efficiency figures obtained through our analysis are compared against those of the original DP and rainbow table methods in Section 6. The final section contains a summary of our results. Most of the technical proofs are deferred to the appendix.

2 Preliminaries

Throughout this paper $F : \mathcal{N} \to \mathcal{N}$ will be a function acting on a set $\mathcal{N}$ of size N. As is done by any theoretic analysis of tradeoff algorithms, we take F to be the random function during our theoretic arguments.

Inversion Problem. Let us first clarify the inversion problem we are considering, as there are two versions that need to be distinguished in any accurate analysis of the tradeoff algorithms. Given the inversion target $\mathbf{y} = F(\mathbf{x})$, the first version asks the algorithm to return any single $x \in \mathcal{N}$ satisfying $F(x) = \mathbf{y}$. In the second version, the tradeoff algorithm is allowed to return multiple $x \in \mathcal{N}$ satisfying $F(x) = \mathbf{y}$ and is declared successful as soon as the specific $\mathbf{x}$ that was used to create $\mathbf{y}$ is returned. The two inversion objectives clearly require different amounts of resources to achieve. If applications of the tradeoff technique to

password recovery from sufficiently long password hashes are to be considered, the second version is the correct inversion problem to study [11]. In the interest of practical applicability, the current paper deals with the second inversion problem.

Terminology. We assume that the reader has basic knowledge of the DP and rainbow tradeoff algorithms. Below, we quickly review the terminology and fix notation, mainly focusing on the DP tradeoff, but analogous terminology and notation will be used with the rainbow tradeoff. All tradeoff algorithms considered in this paper are the non-perfect table versions.

The correct answer to be recovered and the inversion target will always be denoted by $\mathbf{x}$ and $\mathbf{y} = F(\mathbf{x})$, respectively. Each *DP table*, consisting of starting and ending point pairs of pre-computed *DP chains*, will contain approximately m entries. What is meant by the term approximately will be explained below. The probability of DP occurrence is set to $\frac{1}{t}$, so that the average chain length is roughly t. We distinguish between a DP table and a *DP matrix*, which is the complete set of m pre-computed chains. An online chain *merging* into a pre-computation chain will bring about a *false alarm*. We omit any mentioning of reduction functions, as our arguments will mostly focus on a single table. Rather than using exactly t tables, we allow more flexibility and use ℓ tables, where $\ell \approx t$. Flexibility is also given to the matrix stopping rule, so that $mt^2 = \mathtt{D}_{msc}N$, with a *matrix stopping constant* $\mathtt{D}_{msc} \approx 1$. The condition $\mathtt{D}_{msc} \approx 1$ can be justified as was originally done by Hellman [8] or through a birthday paradox argument as given in [5, 6]. In all cases, we assume that the parameters are reasonable in the sense that $m, t \gg 0$, which, in particular, implies $t \ll \sqrt{N}$ through the matrix stopping rule.

Any implementation of the DP tradeoff will place a *chain length bound* $\hat{t}$ to detect chains falling into loops that do not contain DPs. If one generates m_0 chains with the chain length bound $\hat{t}$, one can expect to collect $m_0\{1-\left(1-\frac{1}{t}\right)^{\hat{t}}\}$ DP chains. Rather than requiring a DP table to contain the information of exactly m DP chains, we take the approach that the chains are always generated from $m_0 = m/\{1-\left(1-\frac{1}{t}\right)^{\hat{t}}\}$ starting points and that the resulting approximately m DP chains are accounted for by each DP table.

Note that the expected number of chains that do not reach a DP within the $\hat{t}$ iterations is $m_0\left(1-\frac{1}{t}\right)^{\hat{t}}$ so that the ratio of wasted iterations $m_0\hat{t}\left(1-\frac{1}{t}\right)^{\hat{t}}$ over the approximate total pre-computation effort $m_0 t$ is $\frac{\hat{t}}{t}\left(1-\frac{1}{t}\right)^{\hat{t}}$. This quickly approaches zero as $\frac{\hat{t}}{t}$ is increased. In the interest of practical applications of the tradeoff technique, we will focus on the situation where $\frac{\hat{t}}{t}$ is sufficiently large, so that most of the pre-computation effort is put to use. However, since the mentioned approach to zero is extremely fast, we treat $\hat{t}$ and t as being of somewhat similar order, even when assuming $\frac{\hat{t}}{t}$ to be sufficiently large. This allows us to assume $\hat{t} \ll \sqrt{N}$ and freely use the approximation $\left(1-\frac{1}{t}\right)^{\hat{t}} \approx e^{-\hat{t}/t}$. In practice, setting $\hat{t} = 10\,t$ should be large enough for most purposes.

Extensions to the DP Tradeoff. There are a few tricks that can be used with the DP tradeoff to increase its efficiency.

1. Starting points that require less storage than random ones are used [2, 6]. In particular, sequential starting points [1] allow each starting point to be recorded in $\log m_0 \approx \log m$ bits rather than $\log N$ bits.
2. Information concerning the ending points that can be recovered from the DP definition is not recorded [6]. This reduces $\log t$ bits of storage per ending point.
3. The index file (or hash table) technique [6] is used to remove almost $\log m$ bits per ending point without any loss of information.
4. The ending points are simply truncated before storage [4, 6]. Information is lost through this process and since a partial match between the end of an online chain and a DP table entry will falsely be interpreted as a chain collision, a new type of false alarm appears. However, these can be resolved in exactly the same way as with the exiting false alarms. Using the analysis given in [11], one can maintain the side effects of additional false alarms to a manageable level by controlling the degree of truncation.
5. Suppose that the online chain has become a DP chain of length i and has produced an alarm. When regenerating the associated pre-computation chain to resolve the alarm, one need not continue any further than $\hat{t}-i$ iterations [11].

From now on, the term (*original*) *DP tradeoff* will refer to the algorithm variant that utilizes all five techniques described above.

When the first four tricks described above, which reduce storage, are applied, each DP table entry can be stored in slightly more than $\log m$ bits. The 1-st, 3-rd, and 4-th items can also be applied to the rainbow tradeoff, after which each rainbow table entry consumes slightly over $\log m$ bits. However, the m for the rainbow tradeoff should be taken to be of order similar to mt of the DP tradeoff, so the rainbow table entries take up larger space than the DP table entries. If the DP tradeoff parameters m and t are roughly of the same order and corresponding rainbow tradeoff parameters are used, each rainbow table entry will occupy twice the number of bits required of a DP table entry. This discussion concerning the difference in bits required per table entry was first made in [4] and was theoretically confirmed in [11].

Previous Results. The original DP tradeoff, as defined above, was analyzed in [11]. In the remainder of this section, we review results from [11] that are required in this paper and introduce some more notation.

Given a DP matrix, its *coverage rate* is defined to be the number of distinct nodes that appear among the DP chains as inputs to the one-way function F, divided by mt. Note that the DPs ending each pre-computation chain are not counted in this definition. The expected coverage rate can be computed through the formula

$$\mathtt{D}_{cr} = \frac{2}{\sqrt{1 + 2\mathtt{D}_{msc}} + 1}, \tag{1}$$

when $\frac{\hat{t}}{t}$ is sufficiently large. In particular, the coverage rate can be seen as a function of the single variable $\mathtt{D}_{msc} = \frac{mt^2}{N}$, rather than the separate parameters m, t, and N.

Let $\mathtt{D}_{pc} = \frac{mt\ell}{N}$ be the pre-computation coefficient so that $\mathtt{D}_{pc}N$ is the pre-computation cost. It is not difficult to show that the probability of success for the DP tradeoff can be expressed as $\mathtt{D}_{ps} = 1 - e^{-\mathtt{D}_{pc}\mathtt{D}_{cr}}$. If we rewrite this equation in the form

$$\mathtt{D}_{pc} = -\frac{\ln(1-\mathtt{D}_{ps})}{\mathtt{D}_{cr}} = -\frac{\ln(1-\mathtt{D}_{ps})}{2}\left(\sqrt{1+2\mathtt{D}_{msc}}+1\right), \tag{2}$$

we can see that, under a fixed requirement on the success rate of the DP tradeoff, the pre-computation coefficient $\mathtt{D}_{pc}$ is a function of the matrix stopping constant $\mathtt{D}_{msc}$.

Finally, when $\frac{\hat{t}}{t}$ is sufficiently large, the time memory tradeoff curve for the original DP tradeoff is given by $TM^2 = \mathtt{D}_{tc}N^2$, where the tradeoff coefficient is

$$\mathtt{D}_{tc} = \left(2 + \frac{1}{\mathtt{D}_{msc}}\right)\frac{1}{\mathtt{D}_{cr}^3}\,\mathtt{D}_{ps}\big\{\ln(1-\mathtt{D}_{ps})\big\}^2. \tag{3}$$

When placed under a fixed success rate requirement $\mathtt{D}_{ps}$, this can also be seen as a function of $\mathtt{D}_{msc}$ through a substitution of (1).

3 Parallel DP

The details of the DP tradeoff algorithm that processes its tables in parallel are made explicit in this section. The following two further extensions to the DP tradeoff appear in [9, 14].

6. A full record of the online chain is maintained during the online phase. When required to resolve an alarm, one compares the current end of the regenerated pre-computation against the complete online chain, rather than against just $\mathbf{y}$. This allows one to stop at the exact position of chain merge, rather than at the end of the pre-computation chain.
7. The ℓ DP tables are processed in parallel, rather than serially. This causes relatively more time to be spent in dealing with short online chains and brings about a reduction in the number of alarms.

The DP tradeoff that incorporates all seven DP extension techniques discussed so far will be referred to as the *parallel DP* tradeoff, or pD in short. The purpose of this paper is to analyze the pD tradeoff. Analogous to the $\mathtt{D}_{msc}$ notation introduced for the DP tradeoff, we use $\mathtt{pD}_{msc}$ to denote the matrix stopping constant associated with the pD tradeoff.

Since the pD and the original DP tradeoffs share the same pre-computation algorithm, their respective coverage rates and pre-computation costs are equal, when identical parameters m, t, and ℓ are used. The online phase algorithms of the two tradeoffs are different, but the differences have no effect on the success

probability. Thus, equations (1) and (2) are also valid for the pD tradeoff, when the variable $\mathtt{D}_{msc}$ is replaced by $\mathtt{pD}_{msc}$. Hence, the notation $\mathtt{D}_{cr}$, $\mathtt{D}_{ps}$, and $\mathtt{D}_{pc}$ will also be used to denote coverage rate, probability of success, and pre-computation coefficient for the pD tradeoff. However, equation (3) is not applicable to pD and obtaining its analogue for the pD tradeoff is the main objective of the next section.

The 6-th item will surely increase the efficiency of the tradeoff algorithm, but the effect of the 7-th item is not as clear. Working with shorter chains will reduce collisions, but each collision is likely to require a longer chain regeneration before it can be ruled out as a false alarm. Predicting its positive effect with certainty does not seem possible without a rigorous analysis, as will be done in this paper.

One can easily argue that a straightforward application of the final two additional DP extensions would require online memory that is of $t\ell = \Theta(T)$ order size, which cannot be acceptable. The number of expected accesses to the online memory is also of the same order and can become a problem. The work [9] explains that an application of a secondary DP definition for selective recording of the online chain can overcome these problems. However, even the resulting vastly reduced requirements for online memory and its accesses will still be somewhat larger than those of the original DP tradeoff. Discussions of how practical or impractical such modified approaches will be on specific implementation environments are outside the scope of this paper. In this work, we simply treat the online memory issue as accesses to acceptably sized fast memory.

The 7-th item requires more explanation. The number of DP tables is roughly of $O(N^{\frac{1}{3}})$ order, which is likely to be larger than the number of available processors for N of interest, implying that each processor will be assigned to multiple tables. In such a situation, we require each processor to work with its share of assigned tables in a round-robin fashion. A processor should process a single iteration for a table and then move onto the next table it was assigned, rather than take the approach of fully processing one table and then fully processing its next assigned table.

We have partially clarified how DP should be parallelized, but there still is an issue concerning the resolving of false alarms. Consider, for the moment, a fully parallel system, where all the DP tables are distributed to different processors. When a processor encounters an alarm, it will regenerate a pre-computation chain, during which time period other processors will continue with their respective online chain iterations. By the time the alarm is resolved, many of the other processors would have reached the end of the online chain creation. This shows that the approach of the 7-th trick in trying to have more time spent on short online chains fails in the fully parallel environment.

Fortunately, each processor is likely to be assigned multiple tables in practice. We assume this situation and, in implementing the 7-th DP extension, each processor is made to resolve any alarm that it encounters, before processing any more online chain iterations. Then, since each processor will be struggling to resolve its share of alarms, further iterations of the online chains are effectively postponed until many of the alarms are resolved. If a set of tables allocated to

a certain processor rarely produces alarms, online chain iterations for this set of tables will proceed faster than those of other sets of tables, but the overall behavior will be as if the online chain iterations were delayed until current alarms are resolved.

During our analysis, when counting the total function iterations, we shall take the simplified view that the i-th online chain iterations for all tables are executed simultaneously and that the $(i+1)$-th simultaneous iterations are executed only after all alarms encountered at the i-th iterations are resolved. This view correctly reflects the parallelization details discussed so far.

4 Complexity of the pD Tradeoff

To analyze the tradeoff efficiency of the pD tradeoff, we need to compute the expected time complexity of its online phase. This will be computed as a sum of two parts. The first part is the time taken for the online chain creation. An extremely rough approximation for this would be t times the number of tables ℓ, but we want to be much more precise. The second part is the extra cost of resolving alarms, which has been ignored in many existing analyses of tradeoff algorithms. Three lemmas will be prepared before we state the tradeoff coefficient of pD.

Since the pD tradeoff processes all the tables in parallel, the online phase is likely to terminate with a correct answer before any of the tables are processed in full. Hence, to compute the online time complexity, we need to understand the success probability associated with the processing of each column of the DP matrix rather than with the complete processing of a DP table.

Lemma 1. *Visualize a DP matrix as having been aligned at the ending points. The number of distinct points found in a column of distance i from the ending points is expected to be*

$$\overleftarrow{m}_i = \mathtt{D}_{cr}\, m\Big(1 - \frac{1}{t}\Big)^{i-1},$$

when $\frac{\hat{t}}{t}$ is sufficiently large.

To roughly verify the correctness of this lemma, first notice that, by the definition of $\mathtt{D}_{cr}$, we can expect there to be $\mathtt{D}_{cr}mt$ distinct points in a single DP matrix. Among these, a ratio of $\frac{1}{t}$ points are expected to reach DPs at their next iterations. Hence, there are $\mathtt{D}_{cr}m$ points that are 1-iteration away from the ending point DPs. This count is what is claimed by this lemma as $\overleftarrow{m}_1$. We can generalize this approach to obtain the number of points that lie further iterations away from the DPs. A full proof of this lemma is given in page 20.

The sum $\sum_{j=1}^{i} \overleftarrow{m}_j$, which may be computed from this lemma, allows us to express the probability for the answer $\mathbf{x}$ not to be found in any of the DP tables within the first i iterations. By suitably combining this with the probability $\big(1 - \frac{1}{t}\big)^{i-1}\frac{1}{t}$ for an online chain to reach a DP at the i-th iteration, it should be possible to obtain the probability for the online chain creation for a specific

table to terminate at the i-th iteration. This leads to the expected online chain creation time of pD and is summarized below. We remind the readers that, as was discussed in the previous section, the online chain creation is to be stopped as soon as the alarm for the correct answer is encountered (and resolved).

Lemma 2. *The online chain creation of the pD tradeoff is expected to require*

$$t^2 \frac{\mathtt{D}_{ps}}{\mathtt{pD}_{msc}\mathtt{D}_{cr}}$$

invocations of F, when $\frac{\hat{t}}{t}$ is sufficiently large.

A detailed proof of this lemma is given in page 21.

Our first goal, i.e., expressing the online chain creation effort, has been reached. The second part, which is the cost of resolving alarms, is given next.

Lemma 3. *The number of iterations required by the pD tradeoff in dealing with alarms is expected to be*

$$t^2 \frac{\ln(1-\mathtt{D}_{ps})}{\mathtt{D}_{cr}} \int_0^1 (1-\mathtt{D}_{ps})^{1-u} \ln u \, du,$$

when $\frac{\hat{t}}{t}$ is sufficiently large.

A full technical proof of this lemma is given in page 23, but let us briefly explain how it can be approached. As was seen while obtaining the previous lemma, we already have access to the probability for the i-th iteration of a table to be processed. This i-th iteration generates work related to an alarm if and only if the online chain becomes a DP chain at precisely the i-th iteration and it merges with a pre-computation chain. The probability to encounter an online DP chain of length i is easily written as $(1-\frac{1}{t})^{i-1}\frac{1}{t}$. For the merging part, we turn things around and view the pre-computation chain as colliding into the online chain of length i. This allows us to keep track of the length of the pre-computation chain up to the point of collision while computing the collision probability. Note that the length up to collision is equal to the work factor when the 6-th DP extension is used. To arrive at the final statement, some computation is required after combining the three parts that we have explained.

We have gathered enough material to compute the efficiency of pD in balancing storage against online time.

Theorem 1. *The time memory tradeoff curve for the pD tradeoff is $TM^2 = \mathtt{pD}_{tc}N^2$, where the tradeoff coefficient is given by*

$$\mathtt{pD}_{tc} = \Big(\frac{\ln(1-\mathtt{D}_{ps})}{\mathtt{D}_{ps}} \int_0^1 (1-\mathtt{D}_{ps})^{1-u} \ln u \, du + \frac{1}{\mathtt{pD}_{msc}}\Big) \frac{1}{\mathtt{D}_{cr}^3} \mathtt{D}_{ps} \big\{ \ln(1-\mathtt{D}_{ps})\big\}^2,$$

when $\frac{\hat{t}}{t}$ is sufficiently large.

Proof. The expected online time complexity of the pD tradeoff is the sum of its online chain creation and alarm treatment costs, which are given by Lemma 2 and Lemma 3, respectively. Thus, the time complexity may explicitly be written as

$$T = t^2\Big(\frac{\mathtt{D}_{ps}}{\mathtt{pD}_{msc}\mathtt{D}_{cr}} + \frac{\ln(1-\mathtt{D}_{ps})}{\mathtt{D}_{cr}}\int_0^1 (1-\mathtt{D}_{ps})^{1-u}\ln u\,du\Big).$$

Since the storage size is $M = m\ell$, we find

$$TM^2 = (mt\ell)^2\Big(\frac{\mathtt{D}_{ps}}{\mathtt{pD}_{msc}\mathtt{D}_{cr}} + \frac{\ln(1-\mathtt{D}_{ps})}{\mathtt{D}_{cr}}\int_0^1 (1-\mathtt{D}_{ps})^{1-u}\ln u\,du\Big).$$

To arrive at the claimed statement, it suffices to substitute $mt\ell = \mathtt{D}_{pc}N$ and suitably combine the result with $\mathtt{D}_{ps} = 1 - e^{-\mathtt{D}_{pc}\mathtt{D}_{cr}}$, which was mentioned above (2) and stated as also being correct for pD in Section 3. □

Note that even though the definite integral appearing in this theorem cannot be simplified any further, it can be treated as an explicit constant, as soon as the requirement for inversion success rate $\mathtt{D}_{ps}$ is fixed.

5 Experiment Results

This section presents two sets of experiments that were conducted to test some of our theoretic arguments made in the previous section. In all tests, the one-way function was taken to be the key to ciphertext mapping computed with AES-128. Different fixed plaintexts were used to create multiple one-way functions. Zero padding of keys and truncation of ciphertexts were used to control the size of the space the one-way function acted on.

During our theoretic developments we dealt with two DP extension techniques that were not treated in existing analyses of tradeoff algorithms. The first concerns the 6-th DP extension that shortens the regeneration of precomputation chains, and we had to compute the reduced expected cost of dealing with alarms. The second hurdle concerns the 7-th DP extension that changed the order of online chain iteration executions, and we had to work out the probability of inversion success associated with each column of a DP matrix, as opposed to that associated with a whole DP matrix. We tested the correctness of our arguments concerning these two issues with experiments.

Our theory surrounding the online chain record is hidden from view behind Lemma 3. Using arguments made during its proof we can explicitly write out the expected cost of resolving alarms associated with a *single* table as

$$\mathtt{D}_{msc}\,t\int_0^{\hat{t}/t} x\,e^{-x} - \frac{\hat{t}/t}{e^{\hat{t}/t}-1}\big(1-e^{-x}\big)\,dx = \mathtt{D}_{msc}\,t\,\Big\{1 - \frac{1}{e^{\hat{t}/t}} - \frac{(\hat{t}/t)^2}{e^{\hat{t}/t}-1}\Big\}. \quad (4)$$

More precisely, this ignores whether or not the correct answer was found among other tables, and is the expected cost of resolving false alarms during the complete processing of a single table, when all DP extensions up to the 6-th trick are

applied. Rather than testing Lemma 3 directly, which could hide small details through the averaging effect over multiple tables, we tested our theoretic treatment of the 6-th DP extension through (4) that predicts behavior seen while processing a single table. Testing this equation is also more appropriate in that it still retains the dependence on the chain length bound parameter $\hat{t}$.

Table 1. Cost of resolving alarms when fully processing a single DP table with an online chain record

N	m	t	$\hat{t}/t$	$\mathtt{D}_{msc}$	theory	test	N	m	t	$\hat{t}/t$	$\mathtt{D}_{msc}$	theory	test
2^{28}	512	512	1	0.5	12.84	13.02	2^{28}	768	512	1	0.75	19.26	19.47
2^{28}	512	512	5	0.5	210.86	210.19	2^{28}	768	512	5	0.75	316.29	315.67
2^{28}	512	512	10	0.5	254.83	254.70	2^{28}	768	512	10	0.75	382.24	382.12
2^{29}	1536	512	1	0.75	19.26	19.47	2^{28}	1536	512	1	1.5	38.51	38.88
2^{29}	1536	512	5	0.75	316.29	316.28	2^{28}	1536	512	5	1.5	632.58	631.08
2^{29}	1536	512	10	0.75	382.24	381.68	2^{28}	1536	512	10	1.5	764.48	763.41

Experimental verification of (4) is summarized in Table 1. Each table entry corresponds to 2,000 randomly generated DP tables and 5,000 online chain creations per table. All iterations spent in dealing with alarms that were generated during this whole process were counted and divided by $2,000 \times 5,000$. The tests were conducted with different values of $\frac{\hat{t}}{t}$ and $\mathtt{D}_{msc}$, so as to verify (4) at various inputs. We also tried different m and t pairs that give the same $\mathtt{D}_{msc}$ value to verify that the formula is indeed a function of $\mathtt{D}_{msc}$. All the experiment results are very close to our theory.

The second experiment validates our treatment of the parallel processing of tables. At the core of our associated argument is an equation obtained during the proof of Lemma 2, that expresses the probability for the processing of a table to stop at its i-th iteration.[4] The theoretically obtained probability was compared with values obtained through experiments. Working with the i-th termination probability, rather than Lemma 2, which gives the number of one-way function applications summed over all i, allows us more direct verification of finer details.

After fixing various parameters, we first executed a complete pre-computation phase, thus preparing a set of ℓ tables. A simple XOR with a random fixed constant was used as the reduction function for each table. Then, we ran the pD online phase algorithm with a randomly generated inversion target and recorded the iteration count at which the processing of each table was terminated. The online phase was repeated with a certain number of random inversion targets. The process described so far, starting from the pre-computation phase to the multiple target inversions, was repeated a small number of times. Let us denote the number of inversion targets tried per pre-computation set by *keys* and the

[4] This is precisely (5) and (6) of page 22.

number of complete pre-computation phases that were executed as *rpt*. The number of times each specific i was recorded, summed over all tables and test trials, was divided by $\ell \times keys \times rpt$, and was taken as the probability obtained through the experiment.

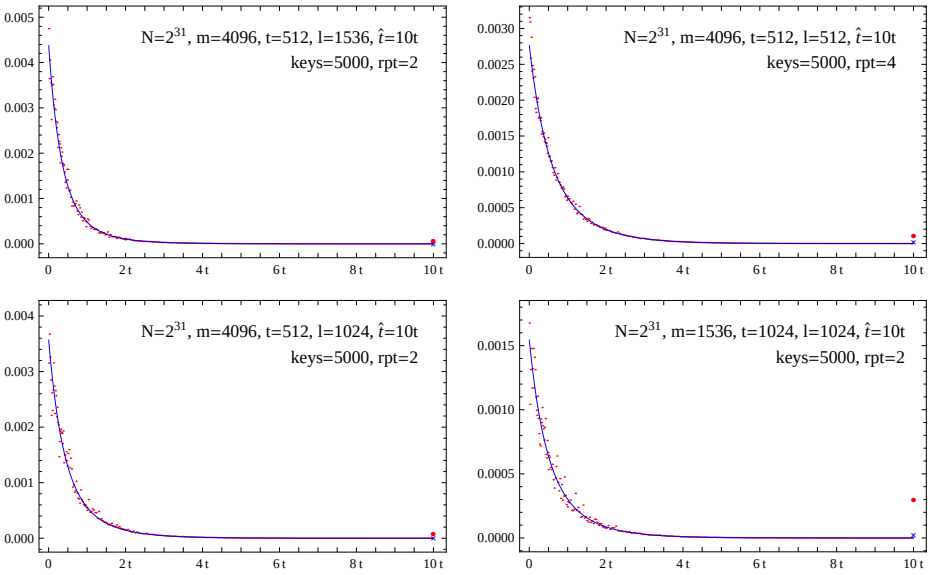

Fig. 1. Probability for pD to stop processing of a specific table at the i-th iteration; Only approximately 240 out of the $\hat{t}$ test values are plotted as dots in each graph (x-axis: i; y-axis: probability)

Test results are depicted as graphs in Figure 1. In each framed graph box, the tiny irregular (red) dots represent the experiment results and the smooth (blue) curve represents the theory. Because plotting all test results made the dots too densely packed and hard to see, we only plotted approximately 240 points among the $\hat{t}$ points of each graph, with the intervals between plotted points shorter at smaller i values. The theoretic value for the $i = \hat{t}$ position, which is expressed as a separate equation, is marked with an $\times$-sign and the corresponding experiment result is marked with a dot that is slightly larger than others. It is clear that the test results and theory are mostly in good agreement. However, the experiment data and theoretic value are visibly different at $i = \hat{t}$ and this requires explanation.

The disagreement at $i = \hat{t}$ is the largest for the right bottom graph. In this case the chain length bound $\hat{t} = 10t = 10240 \approx 2^{13.3}$ is rather close to $\sqrt{N} = 2^{15.5}$, and since such a choice does not satisfy the assumptions made in Section 2, this does not indicate a problem with our analysis. We were forced to use such small parameters by lack of computational resources, but this should

not be an issue in practical applications of the tradeoff technique, where N would be much larger.

Let us discuss this matter slightly further. Recall that we had repeatedly used $\left(1-\frac{1}{t}\right)^i$ as the probability for a chain not to reach a DP until the i-th iteration. This value disregards the possibility of the chain looping back onto itself and a more exact expression is

$$\prod_{j=1}^{i}\left(1-\frac{1}{t}-\frac{j}{N}\right)+\sum_{k=1}^{i}\frac{k}{N}\prod_{j=1}^{k-1}\left(1-\frac{1}{t}-\frac{j}{N}\right).$$

The value $\left(1-\frac{1}{t}\right)^i$ is a good approximation of this, as long as $i \ll \sqrt{N}$, but one knows from the birthday paradox that $\left(1-\frac{1}{t}\right)^i$ will slowly deviate from the value given by the more complicated expression as i approaches $\sqrt{N}$. We conducted a supplementary test, which we do not explain here due to page limits, to verify that our use of $\left(1-\frac{1}{t}\right)^i$ as the ratio of non-DP chains remaining after the i-th iteration was indeed the cause of the discrepancy between theory and experiment at $i=\hat{t}$, when $\hat{t}$ is close to $\sqrt{N}$.

6 Comparison of Tradeoff Algorithms

The analysis given in Section 4 contains enough information for us to make comparisons between algorithms. We will first present a direct comparison between the pD and the original DP tradeoffs. Then, we will present the range of design options made available by the two DP variants and the rainbow tradeoffs compactly as graphs. These graphs will be of more practical value than the initial direct comparisons.

6.1 pD versus DP

As discussed in Section 3, the pD and the original DP tradeoffs will display identical coverage rate, require identical pre-computation cost, and succeed in recovering $\mathbf{x}$ with identical probability, when they are executed under the same set of parameters m, t, and ℓ. The two algorithms also require the same amount of physical storage for DP tables and they only differ in the online execution time. Since the tradeoff coefficients are given as their respective $\frac{TM^2}{N^2}$ values, the online time complexities may directly be compared through $\mathtt{pD}_{tc}$ and $\mathtt{D}_{tc}$. Here, a smaller coefficient implies a more efficient tradeoff algorithm.

After reviewing the tradeoff coefficients as given by (3) and Theorem 1, one can see that they are expressed in a very similar form. In fact, the only difference is in the first constant that sits inside the first set of parentheses. To compare the pD tradeoff against the original DP tradeoff, it suffices to evaluate $\frac{\ln(1-\mathtt{D}_{ps})}{\mathtt{D}_{ps}}\int_0^1(1-\mathtt{D}_{ps})^{1-u}\ln u\,du$ at various success rates. Some explicit values are 0.9293, 0.8345, 0.6879, 0.5283, 0.4332, and 0.2830 at respective success rates 25%, 50%, 75%, 90%, 95%, and 99%. Since all of these are strictly less than 2,

the corresponding constant of the original DP tradeoff, we have a sure indication that the pD tradeoff will outperform the original DP tradeoff.

We have clearly ranked pD over DP, but there is one issue that could affect this conclusion. The online memory access bottleneck of the pD tradeoff could remain a non-negligible overhead, even when the secondary DP technique mentioned in Section 3 is applied, and one iteration of the pD tradeoff could take longer, on average, than that of the DP tradeoff. Should the circumstances be such that a marked difference in this iteration time is inevitable, the difference must be taken into account when interpreting a tradeoff coefficient ratio as a performance ratio. For example, if the pD tradeoff requires α time units per iteration and the DP tradeoff requires β time units per iteration, then one must compare the values $\alpha \cdot \mathtt{pD}_{tc}$ and $\beta \cdot \mathtt{D}_{tc}$ against each other rather than compare $\mathtt{pD}_{tc}$ against $\mathtt{D}_{tc}$.

After seeing that pD outperforms DP, one naturally questions whether the improvement comes from the 6-th or the 7-th DP extension, i.e., whether keeping a record of the online chain or the parallel processing of tables was the main cause for the improvement. Let us refer to the DP variant that uses the first six of the seven DP extensions as the DP-OCR tradeoff. Details are not provided in this paper due to the page limit, but we have also analyzed DP-OCR in full. Results are that the tradeoff coefficient for DP-OCR falls between those of pD and DP, if all three are subject to the same parameters m, t, and ℓ. When parameters are such that only low inversion success rates (50%) are expected, DP-OCR performs quite close to pD, implying that the online chain record accounts for most of the improvement. However, when parameter achieving high success rates (99%) are chosen, DP-OCR stands approximately halfway between original DP and pD, implying that the parallelization does play a significant role in increasing efficiency.

6.2 pD versus Rainbow

Our next goal is to include the rainbow tradeoff in the comparison. This is not as straightforward as the comparisons between the two DP variants, mainly because of the large structural differences between the DP and rainbow tradeoffs. Below, we quickly review the approach of [11] before following it to present a fair comparison. The information required to draw the graphs for the rainbow tradeoff was also taken from [11]. Notation $\mathtt{R}_{pc}$ and $\mathtt{R}_{tc}$ will be used to denote the pre-computation coefficient and the tradeoff coefficient of the rainbow tradeoff, and the notation $\mathtt{X}_{pc}$ and $\mathtt{X}_{tc}$ will be used when referencing the pre-computation and tradeoff coefficients that are not specific to a tradeoff algorithm.

The tradeoff coefficient $\mathtt{X}_{tc} = \frac{TM^2}{N^2}$ is a measure of how efficiently the algorithm balances online time against storage requirements. A smaller $\mathtt{X}_{tc}$ implies a more efficient tradeoff between online time and storage. However, better tradeoff efficiency usually requires a higher pre-computation cost and is not always desirable in practice. The optimal balance point between the tradeoff efficiency $\mathtt{X}_{tc}$ and pre-computation cost $\mathtt{X}_{pc}$ is a subjective matter that cannot be arbitrarily set in this paper. What can be done objectively is to present the range of

> "tradeoff efficiency $\mathtt{X}_{tc}$ can be utilized, if pre-computation effort $\mathtt{X}_{pc}$ is invested"

options that are made available by each algorithm, as a graph. Then, the implementer can make subjective decisions based on this compact display of possible choices and the physical resources that are available to him.

The above discussion would be sufficient for anyone interested in choosing parameters for one fixed tradeoff algorithm. However, a little more care is required when comparing different algorithms. The range of possible options for each algorithm must be presented in a consistent manner. A common requirement on the success rate must be taken and the unit used to express the tradeoff coefficients must be unified. We have already seen that a direct comparison between $\mathtt{pD}_{tc}$ and $\mathtt{D}_{tc}$ is justifiable. As for comparisons of these two against the rainbow tradeoff, let us simply state that $\mathtt{pD}_{tc}$ and $\mathtt{D}_{tc}$ should be compared against $4\mathtt{R}_{tc}$ (as opposed to $\mathtt{R}_{tc}$) to be fair. The main reason is that, as discussed in Section 2, the number of bits per table entry required by a rainbow tradeoff is roughly twice that of a DP tradeoff. The decision to compare $\mathtt{pD}_{tc}$ against $4\mathtt{R}_{tc}$ involves certain assumptions, such as that m and t are of similar order and that the bookkeeping of online chain records causes negligible overhead, but they are assumptions that are typically made during theoretic analysis of tradeoff algorithms and the details should be clear to anyone with a full understanding of [11]. This concludes our short review of the approach to fair tradeoff comparison given by [11].

The range of $(\mathtt{X}_{pc}, \mathtt{X}_{tc})$ options each tradeoff algorithm is capable of providing is given in Figure 2. To draw the graph, for example, corresponding to the DP tradeoff, one refers to (2) for the x-coordinate (pre-computation cost), substitutes (1) into (3) for the y-coordinate (tradeoff coefficient), and then plots the curve parameterized by $\mathtt{D}_{msc}$.

In each framed graph box, the two curves and the sequence of dots should be seen as extending infinitely upwards. However, the right ends of the three graphs are either clearly marked or clearly visible. The curves start to go back up beyond the marked right ends, so that these marks correspond to the minimum tradeoff coefficient achievable by each algorithm. As going beyond this minimum implies using larger pre-computation while obtaining worse tradeoff efficiency, parameters corresponding to the parts that are not drawn should not be used.

We are now ready to discuss the implications of the graphs given in Figure 2. The graphs for $\mathtt{D}_{tc}$ and $\mathtt{pD}_{tc}$ are given by the dashed and solid lines, respectively. The possible $(\mathtt{R}_{pc}, 4\mathtt{R}_{tc})$ choices are given by the discrete sequence of dots. Each dot corresponds to the use of a certain number of rainbow tables and since these table counts tend to be small, especially at low success rate requirements, the possible options appear spaced apart from each other. If one is required to fill in the space between the dots, one may extend horizontal lines to the right of each dot, until the line reaches over the dot to its right. Each box corresponds to a certain requirement on the probability of successful inversions.

In all the graph boxes, the graphs for the pD tradeoff sit further away from the bottom left corner than the dots for the rainbow tradeoff. Being closer to the bottom left corner implies that the same tradeoff efficiency can be obtained

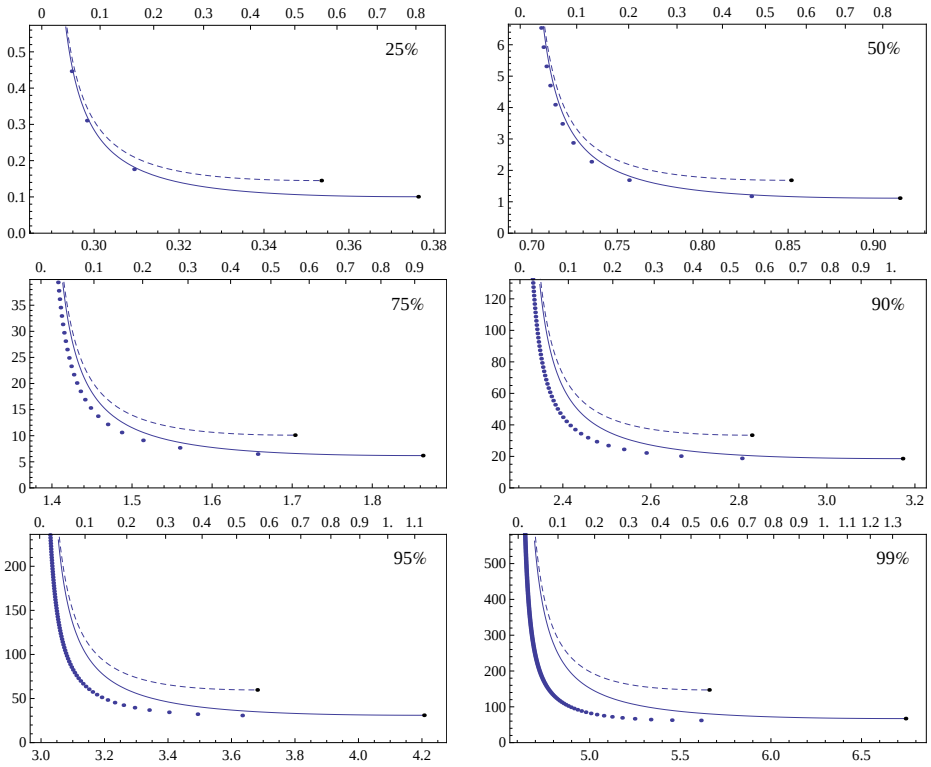

Fig. 2. Tradeoff coefficients for the DP (dashed), pD (solid), and rainbow (large dots) tradeoffs in relation to pre-computation cost, at various success rates; Numeric values on each frame represent pre-computation iterations in units of N (bottom), tradeoff coefficient values $\mathtt{D}_{tc}$, $\mathtt{pD}_{tc}$, $4\mathtt{R}_{tc}$ (left), and the matrix stopping constants $\mathtt{D}_{msc}$, $\mathtt{pD}_{msc}$ that served as parameters for drawing the curves (top)

at a smaller pre-computation cost and that a better tradeoff efficiency can be obtained at equal pre-computation cost. Hence a very rough conclusion would be that the rainbow tradeoff is the best among the three algorithms.

Let us discuss this in more detail, starting with the case of success rate set to 25%. The optimal tradeoff coefficient reachable by the pD tradeoff is $\mathtt{pD}_{tc} = 0.10$. This tradeoff efficiency can be used if the available resources permit $0.376N$ iterations of pre-computation. In comparison, the rainbow tradeoff achieves optimal tradeoff coefficient $4\mathtt{R}_{tc} = 0.18$ at $0.309N$ pre-computation iterations. Even though $\mathtt{pD}_{tc} = 0.10$ and $4\mathtt{R}_{tc} = 0.18$ represent online time ratio of 1.8 at equal physical storage size, depending on the resources available to the tradeoff implementer, the advantage of pD in tradeoff efficiency may or may not be worth its disadvantage in the pre-computation cost one must accept. So far, neither of the two tradeoffs can be said to be clearly superior over the other.

However, another issue that is evident in the 25% case is that the rainbow tradeoff provides much less flexibility in options than the pD tradeoff. For example, the option of using $\mathtt{pD}_{tc} = 0.11$ at $\mathtt{D}_{pc} = 0.336$ is available with the pD algorithm. Compared to the optimal efficiency of $\mathtt{pD}_{tc} = 0.10$ at $\mathtt{D}_{pc} = 0.376$, this gives a valuable reduction in pre-computation cost at a small degradation of tradeoff efficiency. Unless the cost of pre-computation is extremely cheap, most implementers of the tradeoff algorithm will prefer to use $\mathtt{pD}_{tc} = 0.11$ over the optimal $\mathtt{pD}_{tc} = 0.10$. The rainbow tradeoff does not allow such a freedom of choice at the 25% probability of success.

Since the dots for the rainbow tradeoff are very close to the curve for the pD tradeoff, one can say that every option provided by the rainbow tradeoff can (nearly) be provided by the pD tradeoff. Since the pD tradeoff provides higher flexibility and even the possibility of a lower tradeoff coefficient, it seems safe to conclude that the pD tradeoff is preferable over the rainbow tradeoff at 25% success rate.

Even though we have explained at length that the pD tradeoff could be preferable over the rainbow tradeoff, the observations made for the 25% success rate are not very applicable to any of the other graph boxes. In the 50% success rate case, the optimal pD option of $\mathtt{pD}_{tc} = 1.12$ at $\mathtt{D}_{pc} = 0.915$ is not a very attractive choice over the rainbow option of $4\mathtt{R}_{tc} = 1.17$ at $\mathtt{R}_{pc} = 0.828$, which achieves similar tradeoff efficiency at a visibly lower pre-computation cost. Similarly, we have $\mathtt{pD}_{tc} = 6.19$ at $\mathtt{D}_{pc} = 1.86$ versus $4\mathtt{R}_{tc} = 6.48$ at $\mathtt{R}_{pc} = 1.66$ for the 75% success rate, and $\mathtt{pD}_{tc} = 18.5$ at $\mathtt{D}_{pc} = 3.17$ versus $4\mathtt{R}_{tc} = 18.7$ at $\mathtt{R}_{pc} = 2.81$ for the 90% success rate. At these moderate success rates, the minimum $\mathtt{pD}_{tc}$ is slightly better than the minimum $4\mathtt{R}_{tc}$, but its use cannot be justified when pre-computation cost is taken into account. In fact, as discussed in the 25% success rate, implementers are likely to choose parameters somewhat away from the optimal tradeoff efficiency points, where the rainbow tradeoff is clearly advantageous over the pD tradeoff. As for higher success rates 95% and 99%, even the minimum $4\mathtt{R}_{tc}$ is smaller than the minimum $\mathtt{pD}_{tc}$, so that there is no reason to prefer the pD tradeoff over the rainbow tradeoff.

7 Conclusion

The parallel DP tradeoff studied in this work is a cryptanalytic time memory tradeoff algorithm that adds two extra techniques to the more widely known DP tradeoff. The first is to keep a full record of the online chain so that alarms can be resolved earlier during the pre-computation chain regeneration. The second idea is to process the multiple DP tables in parallel. This allows for more time to be spent in dealing with relatively shorter chains so that false alarms are hopefully reduced. We have confirmed that both of these ideas have positive effects on the efficiency of the DP tradeoff.

Our analysis of the pD tradeoff did not ignore the time taken to resolve false alarms and is accurate enough to provide multiple significant digits. Results of the analysis were used to provide a comparison between the pD and rainbow

tradeoffs. The comparison of tradeoff efficiency was done in a fair manner in the sense that factors such as the success probability of inversion, the storage size in number of bits, and pre-computation cost were all taken into account. Hence, the comparison results have practical implications on the choice of which tradeoff algorithm to use.

Comparisons show that, even with the extra enhancements, the pD tradeoff is not likely to be preferable over the rainbow tradeoff under most situations. The only exception is when the success rate requirement is very low. For example, when dealing with multi-target time memory tradeoffs [5], where the rainbow tradeoff is known to be much less efficient than both the original Hellman and DP tradeoffs, our analysis is an indication that the use of the pD tradeoff could be advantageous over the original DP tradeoff. At moderate success rate requirements, the pD tradeoff can be slightly more efficient than the rainbow tradeoff, but the choice to use the pD tradeoff cannot be justified when the pre-computation cost is taken into account.

In short, when reduction in wall-clock running time is very important and one is willing to parallelize the online phase to a very high degree, depending on the degree of parallelization available, variants of the DP tradeoff could be a reasonable choice. However, if total CPU time is more important than wall-clock time, one should work with the rainbow tradeoff. Still, the pD tradeoff is more efficient than the usual DP tradeoff, in that it requires a smaller total number of function iterations, when the two are provided with the same pre-computation table.

The theoretic analysis and the resulting concrete graphs of this paper can easily be adjusted to cope with various specific situations and allow for educated decisions. For example, when run on resource constrained environments such as GPUs, iterations of pD may take longer than those of DP due to pD's higher demands for online memory. In this situation, it suffices to scale the tradeoff coefficients of pD and DP according to their respective average iteration timings before comparing their graphs to conclude whether the online memory requirement undermines the small advantage of pD over DP.

References

1. G. Avoine, P. Junod, P. Oechslin, Characterization and improvement of time-memory trade-off based on perfect tables. *ACM Trans. Inform. Syst. Secur.*, **11**(4), 17:1–17:22 (2008). Preliminary version presented at INDOCRYPT 2005.
2. J. Borst, *Block Ciphers: Design, Analysis, and Side-Channel Analysis*. Ph.D. Thesis, Katholieke Universiteit Leuven, September 2001
3. J. Borst, B. Preneel, J. Vandewalle, On the time-memory tradeoff betweeen exhaustive key search and table precomputation. In *Proceedings of the 19th Symposium on Information Theory in the Benelux*, WIC, 1998
4. E. Barkan, E. Biham, A. Shamir, Rigorous bounds on cryptanalytic time/memory tradeoffs. In *Advances in Cryptology—CRYPTO 2006*, LNCS, vol. 4117, (Springer, 2006), pp .1–21.

5. A. Biryukov, A. Shamir, Cryptanalytic time/memory/data tradeoffs for stream ciphers, In *Advances in Cryptology—ASIACRYPT 2000*. LNCS, vol. 1976, (Springer, 2000), pp. 1–13.
6. A. Biryukov, A. Shamir, D. Wagner, Real time cryptanalysis of A5/1 on a PC. In *FSE 2000*, LNCS **1978**, (Springer, 2001), pp. 1–18
7. D. E. Denning, *Cryptography and Data Security* (Addison-Wesley, 1982)
8. M. E. Hellman, A cryptanalytic time-memory trade-off. *IEEE Trans. on Infor. Theory*, **26**, (1980), pp. 401–406.
9. Y. Z. Hoch, Security analysis of generic iterated hash functions. Ph.D. Thesis, Weizmann Institute of Science, August 2009.
10. J. Hong, *Des. Codes Cryptogr.*, The cost of false alarms in Hellman and rainbow tradeoffs. **57**(3), (Springer, 2010), pp.293–327.
11. J. Hong, S. Moon, A comparison of cryptanalytic tradeoff algorithms. Cryptology ePrint Archive. Report 2010/176.
12. A. Narayanan, V. Shmatikov, Fast dictionary attacks on passwords using time-space tradeoff. *Proceedings of the 12th ACM CCS*. (ACM, 2005), pp. 364–372.
13. P. Oechslin, Making a faster cryptanalytic time-memory trade-off. in *Advances in Cryptology–CRYPTO 2003*, LNCS, vol. 2729, (Springer, 2003) pp .617–630.
14. A. Shamir, Random Graphs in Security and Privacy. Invited talk at ICITS 2009.
15. F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, J.-D. Legat, A time-memory trade-off using distinguished points: New analysis & FPGA results. In *Cryptographic Hardware and Embedded Systems—CHES 2002*, LNCS **2523**, (Springer, 2003), pp. 593–609

A Technical Proofs of Lemmas

Full proofs to the three lemmas that were introduced in Section 4 are provided here. The proofs mostly consist of careful applications of the random function arguments followed by some technical computations.

A.1 Lemma 1

We want to compute the number $\overleftarrow{m}_i$ of distinct points situated at distance i from the ending points in a DP matrix.

Consider a DP matrix constructed with a sufficiently large $\frac{\hat{t}}{t}$. We may assume that the fraction of wasted pre-computation iterations $\frac{\hat{t}}{t}\left(1-\frac{1}{t}\right)^{\hat{t}} \approx \frac{\hat{t}}{t}\exp(-\frac{\hat{t}}{t})$ is very small. This implies that only a negligible fraction of the points on which the random function F was defined is discarded during the pre-computation table creation.

Let us fix a specific method for counting the number of distinct non-ending points in the DP matrix. These points are the inputs on which the random function images were defined and their total number is expected to be $\mathtt{D}_{cr}mt$. An example of counting method would be to count by rows. One starts by counting the number of points on the first row, excluding the ending point, and successively adds the number of points found on the next row, being careful to exclude any chain segment that merges into a previously counted chain. One

could also count by columns or choose a more complicated walk through the DP matrix.

Regardless of the counting method that is taken, one can view the points that are counted as those inputs on which the random function was randomly defined without any restriction. Chain iterations computed on all other F-input points of the DP matrix can be seen as having followed the random function definitions made on the counted points.

Since the F-image definitions were made randomly on the $\mathtt{D}_{cr}mt$ fresh points, we can expect $\mathtt{D}_{cr}mt\frac{1}{t}$ many of these points to have been mapped to DPs by F. The discussion at the beginning parts of this proof assures us that the ratio $\frac{1}{t}$ of points mapped to DPs has not been altered inadvertently by the discarded pre-computation. These points that were mapped to DPs are clearly the points situated one iteration away from the DP ending points and these contain no duplicates. A straightforward extension of this idea is that $\mathtt{D}_{cr}mt\big(1-\frac{1}{t}\big)^{i-1}\frac{1}{t}$ many distinct points are found i iterations away from the DPs, as claimed.

A.2 Lemma 2

The cost of creating the online chain is given by this lemma. Even though pD processes all ℓ tables in parallel, let us focus on a single fixed table and compute the expected work associated with its processing. The total cost will then be ℓ times the value we compute.

The initial searching of the inversion target $\mathbf{y} = F(\mathbf{x})$ among the DPs, which requires no F invocation, will be referred to as the 1-st iteration. As explained at the end of Section 3, we take the convention that the outcome from the i-th iteration of one table does not affect the i-th iteration of another table. Only strictly previous iterations will have the possibility of affecting the current iteration.

The pD algorithm will terminate the online chain creation for the table under our focus right after processing the i-th iterations for all tables if and only if one of the following events occur.

1. The online chain for the table under focus became a DP chain of length[5] i, while none of the other $\ell - 1$ tables produced the correct answer $\mathbf{x}$ to be recovered up to the i-th iteration.
2. The online chain for the table under focus did not reach a DP up to the i-th iteration, and the correct $\mathbf{x}$ was found for the first time in one (or more) of the other $\ell - 1$ tables at the i-th iteration.
3. The online chain for the table under focus became a DP chain of length i, and the correct inverse $\mathbf{x}$ was found for the first time in one (or more) of the other $\ell - 1$ tables at the i-th iteration.

[5] We measure the length of an online chain starting from the unknown answer $\mathbf{x}$. The inversion target $\mathbf{y}$ is already an online chain of length 1 and the i-th iteration produces an online chain of length i.

Note that the above three events are mutually exclusive and that the two sub-events that constitute each of the above three events are independent from each other.

Let us compute the probability for the first event to occur. The online chain requirement is satisfied with probability $\left(1-\frac{1}{t}\right)^{i-1}\frac{1}{t}$. As for the inversion failure part, observe that no two distinct columns of a DP matrix, aligned at its ending points, can contain a common element. Hence the number of distinct points, used as inputs to F during table creation, that lie within i iterations away from the ending points, is given by $\sum_{j=1}^{i}\overleftarrow{m}_j$. The probability for the other tables to fail in producing the correct inverse up until the i-th iteration is thus $\left(1-\frac{\sum_{j=1}^{i}\overleftarrow{m}_j}{N}\right)^{\ell-1}$.

The second event is discussed next. Its online chain part occurs with probability $\left(1-\frac{1}{t}\right)^{i}$. As for the part concerning the recovery of $\mathbf{x}$, one must succeed in recovering the inverse within the first i iterations, but not succeed within the first $i-1$ iterations. This probability is given by the difference $\left\{1-\left(1-\frac{\sum_{j=1}^{i}\overleftarrow{m}_j}{N}\right)^{\ell-1}\right\}-\left\{1-\left(1-\frac{\sum_{j=1}^{i-1}\overleftarrow{m}_j}{N}\right)^{\ell-1}\right\}$. We emphasize that successful inversion before the i-th iteration must be ruled out, so that the simpler expression $1-\left(1-\frac{\overleftarrow{m}_i}{N}\right)^{\ell-1}$ does not serve our need.

After the probability for the third event is similarly computed, we can write the probability for the processing of the table under focus to stop at the i-th iteration to be

$$\begin{aligned}
&\left(1-\frac{1}{t}\right)^{i-1}\frac{1}{t}\left(1-\frac{\sum_{j=1}^{i}\overleftarrow{m}_j}{N}\right)^{\ell-1}\\
&+\left(1-\frac{1}{t}\right)^{i}\left\{\left(1-\frac{\sum_{j=1}^{i-1}\overleftarrow{m}_j}{N}\right)^{\ell-1}-\left(1-\frac{\sum_{j=1}^{i}\overleftarrow{m}_j}{N}\right)^{\ell-1}\right\}\\
&+\left(1-\frac{1}{t}\right)^{i-1}\frac{1}{t}\left\{\left(1-\frac{\sum_{j=1}^{i-1}\overleftarrow{m}_j}{N}\right)^{\ell-1}-\left(1-\frac{\sum_{j=1}^{i}\overleftarrow{m}_j}{N}\right)^{\ell-1}\right\}.
\end{aligned}$$

If we apply Lemma 1, the approximation $(1-1/a)^b \approx e^{-b/a}$, and $\mathtt{D}_{pc}=\frac{mt\ell}{N}$, we arrive at

$$\begin{aligned}
&\left(1-\frac{1}{t}\right)^{i-1}\exp\Big(-\mathtt{D}_{pc}\mathtt{D}_{cr}\Big\{1-\left(1-\frac{1}{t}\right)^{i-1}\Big\}\Big)\\
&\qquad-\left(1-\frac{1}{t}\right)^{i}\exp\Big(-\mathtt{D}_{pc}\mathtt{D}_{cr}\Big\{1-\left(1-\frac{1}{t}\right)^{i}\Big\}\Big),
\end{aligned}\tag{5}$$

which is correct for $1\le i<\hat{t}$. In order for all the probabilities to add up to 1, the final probability at $i=\hat{t}$ should clearly be

$$\left(1-\frac{1}{t}\right)^{\hat{t}-1}\exp\Big(-\mathtt{D}_{pc}\mathtt{D}_{cr}\Big\{1-\left(1-\frac{1}{t}\right)^{\hat{t}-1}\Big\}\Big).\tag{6}$$

Since stopping at the i-th iteration implies $i-1$ iterations of F, the cost of online chain creation for the single table under our focus can be written as

$$\Big\{\sum_{i=1}^{\hat{t}-1}\big(\text{Equation (5)}\big)\cdot(i-1)\Big\}+\big(\text{Equation (6)}\big)\cdot(\hat{t}-1)$$
$$=\sum_{i=1}^{\hat{t}-1}\Big(1-\frac{1}{t}\Big)^{i}\exp\Big(-\mathtt{D}_{pc}\mathtt{D}_{cr}\Big\{1-\Big(1-\frac{1}{t}\Big)^{i}\Big\}\Big)$$
$$\approx t\int_{0}^{\hat{t}/t}e^{-x}\exp\big(-\mathtt{D}_{pc}\mathtt{D}_{cr}\big(1-e^{-x}\big)\big)\,dx.$$

One can compute this explicitly and find it to be

$$\frac{t}{\mathtt{D}_{pc}\mathtt{D}_{cr}}\big\{1-\exp\big(-\mathtt{D}_{pc}\mathtt{D}_{cr}(1-e^{-\hat{t}/t})\big)\big\}$$
$$\approx\frac{t}{\mathtt{D}_{pc}\mathtt{D}_{cr}}\big\{1-\exp(-\mathtt{D}_{pc}\mathtt{D}_{cr})\big\}=\frac{t}{\mathtt{D}_{pc}\mathtt{D}_{cr}}\mathtt{D}_{ps},$$

where the approximation is valid for sufficiently large $\frac{\hat{t}}{t}$. To arrive at the formula claimed by Lemma 2, it now suffices to multiply ℓ to the above and then apply $\frac{1}{\ell}\mathtt{D}_{pc}=\frac{1}{t}\mathtt{pD}_{msc}$.

A.3 Lemma 3

The number of one-way function iterations required to resolve alarms is given by this lemma. We will continue to use the convention that was explained in the previous subsection concerning the labeling of iterations and how only strictly previous iterations can affect the current iteration. As before, we focus our attention on a single table.

Let us assume that the i-th iteration of the online chain for this table resulted in a DP and compute the work expected to deal with the alarm which may or may not result from this DP. Even though the pre-computation chains were generated long before the current online chain, we treat each pre-computation chain as if it were being freshly generated and study how it might collide with the current online DP chain of length i. That is, we generate a pre-computation chain with a random function that has only been defined on the online chain so far.

The probability for a randomly created chain to collide with a given online chain of length i at the j-th iteration is

$$\Big(1-\frac{1}{t}-\frac{i}{N}\Big)^{j-1}\frac{i+1}{N}\approx\exp\big(-\frac{j}{t}\big)\frac{i+1}{N}.$$

This is since the first $(j-1)$ iterations must be chosen among non-DPs that do not belong to the length-i online chain and the j-th iteration must land on one

of the $(i+1)$ online chain points. The approximation ignores the $\frac{i}{N}$ term, since it is of much smaller order than $\frac{1}{t}$.

Note that the length of a pre-computation chain is at most $\hat{t}$. This implies that, as the j values approach $\hat{t}$, we know beforehand that the next iteration images will not land on the online chain points that are far from the ending point. More precisely, we can write the probability for a random pre-computation chain to collide with the online chain at distance $j \leq \hat{t}$ from the starting point of the pre-computation chain as

$$\begin{aligned}
&\prod_{k=0}^{j-2} \Big(1 - \frac{1}{t} - \frac{\min\{i+1, \hat{t}-k\}}{N - (i+1-\min\{i+1, \hat{t}-k\})}\Big) \\
&\times \frac{\min\{i+1, \hat{t}-j+1\}}{N - (i+1-\min\{i+1, \hat{t}-j+1\})} \\
&\qquad\qquad \approx \exp\Big(-\frac{j}{t}\Big) \frac{\min\{i+1, \hat{t}-j+1\}}{N}.
\end{aligned}$$

Since pD keeps a record of the complete online chain (the 6-th extension to DP), if the online chain merges with the pre-computation chain at distance j from the starting point of the pre-computation chain, regeneration of the pre-computation to resolve alarms can be stopped at the j-th iteration. Also recall that the regeneration of a pre-computation chain need not exceed $(\hat{t}-i+1)$ iterations (the 5-th extension to DP). The expected cost of resolving alarms which may or may not occur from an online DP chain of length i can be written as

$$\begin{aligned}
&\sum_{j=1}^{\hat{t}} \min\{j, \hat{t}-i+1\} \frac{m}{1-e^{-\hat{t}/t}} \exp\Big(-\frac{j}{t}\Big) \frac{\min\{i+1, \hat{t}-j+1\}}{N} \\
&\qquad \approx \frac{m}{1-e^{-\hat{t}/t}} \frac{t^3}{N} \int_0^{\hat{t}/t} \min\Big\{x, \frac{\hat{t}}{t} - \frac{i}{t}\Big\} \exp(-x) \min\Big\{\frac{i}{t}, \frac{\hat{t}}{t} - x\Big\} dx \\
&\qquad = \frac{\mathtt{D}_{msc}}{1-e^{-\hat{t}/t}} t \Big\{\frac{i}{t}\big(1-e^{-\hat{t}/t}\big) - \frac{\hat{t}}{t}\big(e^{i/t}-1\big)e^{-\hat{t}/t}\Big\}. \qquad (7)
\end{aligned}$$

We are finally ready to write down the cost of dealing with alarms. It suffices to combine the above expected work with the probability for other tables not to produce the correct answer up to the $(i-1)$-st iteration and the probability for the online chain we are focusing on to become a DP chain of length i. Recalling that the cost must be added over all ℓ tables, the cost of dealing with alarms can be written as

$$\ell \sum_{i=1}^{\hat{t}} \Big(1 - \frac{\sum_{j=1}^{i-1} \overleftarrow{m}_j}{N}\Big)^{\ell-1} \cdot \Big(1-\frac{1}{t}\Big)^{i-1} \frac{1}{t} \cdot \frac{\mathtt{D}_{msc}}{1-e^{-\hat{t}/t}} t \Big\{\frac{i}{t}\big(1-e^{-\hat{t}/t}\big) - \frac{\hat{t}}{t}\big(e^{i/t}-1\big)e^{-\hat{t}/t}\Big\}.$$

After using Lemma 1 to replace the $\sum_{j=1}^{i-1} \overleftarrow{m}_j$, this can be approximated by the integral expression

$$\mathtt{D}_{msc}\,\ell t \int_0^{\hat{t}/t} \exp\big(-\mathtt{D}_{pc}\mathtt{D}_{cr}(1-e^{-x})\big)\, e^{-x}\left(x - \frac{\hat{t}}{t}\,\frac{e^x-1}{e^{\hat{t}/t}-1}\right) dx.$$

By applying change of variables to the first term $\mathtt{D}_{msc}\,\ell t \int_0^{\hat{t}/t} \exp\big(-\mathtt{D}_{pc}\mathtt{D}_{cr}(1-e^{-x})\big)\, e^{-x}\, x\, dx$ of this expression, we can rewrite it as

$$-\mathtt{D}_{msc}\,\ell t \int_{e^{-\hat{t}/t}}^{1} \exp\big(-\mathtt{D}_{pc}\mathtt{D}_{cr}(1-u)\big)(-\ln u)\,(-du).$$

The formula stated in the lemma is a tweaked version of this equation that can be obtained by applying the relations $\mathtt{D}_{msc}l = \mathtt{D}_{pc}t$, $\mathtt{D}_{ps} = 1 - e^{-\mathtt{D}_{pc}\mathtt{D}_{cr}}$, and $e^{-\hat{t}/t} \approx 0$.

It only remains to deal with the second term. One can easily check that

$$\begin{aligned} 0 \;&\le \mathtt{D}_{msc}\,\ell t \int_0^{\hat{t}/t} \exp\big(-\mathtt{D}_{pc}\mathtt{D}_{cr}(1-e^{-x})\big)\, e^{-x}\left(\frac{\hat{t}}{t}\,\frac{e^x-1}{e^{\hat{t}/t}-1}\right) dx \\ &= \mathtt{D}_{msc}\,\ell t\,\frac{\hat{t}/t}{e^{\hat{t}/t}-1}\int_0^{\hat{t}/t} \frac{1-e^{-x}}{\exp\big(\mathtt{D}_{pc}\mathtt{D}_{cr}(1-e^{-x})\big)}\, dx \;\le\; \mathtt{D}_{msc}\,\ell t\,\frac{(\hat{t}/t)^2}{e^{\hat{t}/t}-1}, \end{aligned}$$

where the final inequality follows from the observation that the integrand is less than 1. The upper bound we have obtained for the second term is clearly negligible in comparison to the previously computed first term, when $\frac{\hat{t}}{t}$ is sufficiently large.