J. Cryptol. (2013) 26: 559–637
DOI: 10.1007/s00145-012-9128-3

Journal of
CRYPTOLOGY

A Comparison of Cryptanalytic Tradeoff Algorithms

Jin Hong*
Department of Mathematical Sciences and ISaC, Seoul National University, Seoul 151-747, Korea
jinhong@snu.ac.kr

Sunghwan Moon
Department of Mathematics, Texas A&M University, College Station, TX 77843-3368, USA
shmoon@math.tamu.edu

Communicated by Antoine Joux

Received 19 July 2010
Online publication 24 July 2012

Abstract. Three time-memory tradeoff algorithms are compared in this paper. Specifically, the classical tradeoff algorithm by Hellman, the distinguished point tradeoff method, and the rainbow table method, in their non-perfect table versions, are treated.

We show that, under parameters and assumptions that are typically considered in theoretic discussions of the tradeoff algorithms, the Hellman and distinguished point tradeoffs perform very close to each other and the rainbow table method performs somewhat better than the other two algorithms. Our method of comparison can easily be applied to other situations, where the conclusions could be different.

The analysis of tradeoff efficiency presented in this paper does not ignore the effects of false alarms and also covers techniques for reducing storage, such as ending point truncations and index tables. Our comparison of algorithms fully takes into account success probabilities and precomputation efforts.

Key words. Time-memory tradeoff, Hellman, Distinguished point, Rainbow table.

1. Introduction

There are numerous security systems in use today that rely on passwords. Access to much of the content on a network requires one to log in with a password, and many file formats today have security features that restrict access to the file until the correct password is supplied. These systems are usually based on a *password hash* technique, which operates by storing a one-way function image of the password in the file or on the system. Indeed, storing the password in its raw form within the file to which one wishes to set access control would be meaningless. Authentication of a user is performed by

* J. Hong was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012003379).

recomputing the one-way function image from a freshly supplied password and comparing the result with the stored password hash.

A time-memory tradeoff algorithm attempts to recover the password from the knowledge of the one-way function image, with the help of a table created through precomputation. The massive precomputation that is required before the actual attack can be mounted is the largest barrier in applying the time-memory tradeoff technique to any specific security system. However, the precomputation cost is roughly proportional to the size of the password space and, since many users do not use strong passwords, the tradeoff attacker is free to choose a manageable set consisting of short or more likely passwords and decide to be satisfied with recovering only those passwords belonging to this set. Then the precomputation requirement does not stand as an impenetrable barrier to the tradeoff attack.

It has long been known that properly *salting* a password can remove any realistic threats of the time-memory tradeoff attacks. The security system concatenates a randomly generated string (salt) of sufficient length to the user-supplied password before computing the one-way function image. The salt value that was used is stored alongside the computed password hash so that it is available to the system for the one-way function recomputation whenever a user needs to be authenticated. The effective number of passwords is increased by the use of salts and this can increase the precomputation requirement of a tradeoff attack to an unrealistic degree.

Nevertheless, the salting countermeasure is still not being used in many proprietary systems, and some systems are known to be using both the newer salted and the older non-salted versions of the security system simultaneously to remain compatible with older systems. Hence, the time-memory tradeoff technique still remains a powerful tool against these vulnerable password hash systems. Since human-generated passwords will continue to be used for the foreseeable future, one would like to fully understand the powers and limitations of the tradeoff techniques.

There are a large number of tradeoff algorithm variants, and we will restrict ourselves to the three major tradeoff algorithms in this work. The first algorithm we study is the original tradeoff algorithm [14] devised by *Hellman*. The second algorithm is the *distinguished point* method, which is attributed to Rivest in [10]. The number of table lookups that are required by a Hellman tradeoff is significantly reduced in this slightly modified method. The final algorithm we consider is the *rainbow* table method [23], announced by Oechslin. The precomputation table for this method is structurally different from the previous two versions.

Let us briefly mention some of the more notable tradeoff variants or techniques that we are not treating in this work. The first is the *perfect table* version of the distinguished point method [8]. This is a variant of the distinguished point method where some of the redundancies contained in the precomputed tables are removed and replaced with nonoverlapping data generated through additional precomputation. The more efficient usage of storage leads to better performance during the actual attack, at the expense of higher precomputation cost. The removal of redundancies is facilitated by the distinguished point technique and cannot be done as easily with the classical Hellman algorithm, but the rainbow table method also admits a perfect table version [23] naturally. The perfect table versions of tradeoff algorithms are of interest due to their better efficiency during the attack phase. However, analyzing them at the accuracy level aimed for by the current paper is quite delicate, and is left as a subject of future study.

Another class of tradeoff variants that we do not consider is the multi-target versions of the tradeoff algorithms [2,5,6,13], which are usually referred to as the time-memory-*data* tradeoffs. The objective of these algorithms is to recover at least one of the many original inputs that were used to create the multiple one-way function images that are supplied as inversion targets. This class of algorithms attracted attention as realistic attacks on stream ciphers, but present-day stream ciphers are designed to withstand these attacks. The most practical application of the tradeoff technique today is with the password hash systems, and we will present the current work with this application in mind.

Even though a considerable portion of this paper is devoted to the performance analyses of the three major tradeoff algorithms, the main motivation for this work was to determine *which time-memory tradeoff algorithm is the best*. Providing a fair and acceptable answer to this seemingly simple question is the ultimate goal of this paper.

It has been shown [3,4] that, if we restrict ourselves to a certain class of algorithms, the explicit tradeoff algorithms that are known today already achieve the best tradeoff efficiency one can hope for, at least asymptotically. However, the measure of efficiency considered by this theory is only accurate up to a small multiplicative factor. In practice, experience seems to be a critical factor in deciding which algorithm to use, and researchers have varied opinions on which algorithm performs better.

Comparison of tradeoff algorithms has been a controversial subject. There are claims of superiority of one algorithm over another, but, in many cases, these arguments are either heuristic or based on complexity analyses that are not accurate up to small constant factors. There are at least two obstacles to providing a fair comparison of tradeoff algorithms. The first is that the online time of each algorithm is hard to predict accurately, due to certain events called false alarms. Some answers to this problem may be found in [1,15] for the Hellman and rainbow cases. The current paper relies heavily on these results. The second obstacle concerns the minimal number of bits required to store each precomputation table entry. In particular, a technique for storage optimization called ending point truncation has not yet been fully analyzed.

There is a naturally occurring measure of how efficiently a tradeoff algorithm balances time against storage in achieving its goal, and the accurate value of this efficiency measure becomes accessible once the first obstacle mentioned above is resolved. As was first noted in [3,4], the measure of tradeoff efficiency has been expressed in different units for different algorithms. In this work, by extending the approach of [3,4], we carefully convert the tradeoff efficiency measures for the three algorithms to a common unit so that they may be directly compared. The unification of units is intimately connected to the second obstacle mentioned above. We also carefully treat the time taken for table lookups during our initial transition of units.

The above two obstacles that are due to our lack of accuracy in presenting the tradeoff efficiency figures can be overcome through rigorous algorithm analyses, but there is yet another problem which is related to the precomputation cost. Currently there is no widely accepted way of comparing two algorithms that can achieve different tradeoff performances only after the investment of different precomputation efforts. Due to this difficulty, many comparisons of tradeoff algorithms have focused on the above-mentioned measure of balancing capability and have ignored the cost of precomputation.

In this work, we clear all the obstacles mentioned so far and provide a fair comparison between tradeoff algorithms. More precisely, we present a method to visualize what can be achieved by each algorithm in terms of precomputation cost and tradeoff efficiency. This will be done in a unified way so that the range of choices made possible by each algorithm can directly be compared against each other. A tradeoff implementer can use this information to decide on which algorithm to use and which set of parameters to use with the algorithm. The judgement of which algorithm is more suitable depends on how the user values the precomputation cost and tradeoff efficiency relative to each other, and, in most cases, the judgement cannot be done in an objective manner.

While presenting the above comparison method, we will mainly focus on a certain set of parameters and environmental assumptions that are typically considered during theoretic analyses of tradeoff algorithms. Under the circumstances under focus, the performances of the classical Hellman and the distinguished point methods are shown to be very close to each other. When placed under the additional requirement that the success rates of the tradeoff algorithms must be high, the rainbow table method is shown to outperform the other two algorithms. These comparison conclusions will stand true for any relative valuing of the precomputation cost and tradeoff efficiency, as long as we are working with a typical situation. Comparisons for other situations can easily be done by following through our methods, and the resulting conclusions could be different.

The remainder of this paper is organized as follows. In the next section, we fix the notation and terminologies while reviewing previous results related to this work. Section 3 clarifies the connection between the theory of tradeoff algorithms and the use of the algorithms in attacking password hash systems. In Sects. 4, 5, and 6, we study the distinguished point, Hellman, and rainbow table tradeoff algorithms, in turn. For each algorithm, we present an accurate tradeoff efficiency figure that does not ignore small multiplicative factors and also analyze the applicable storage reduction techniques. These sections overcome the first and second obstacles that were mentioned previously. Comparisons of tradeoff efficiencies under different parameter sets for the same algorithm are made in Sect. 7. Finally, our goal of algorithm comparison is reached in Sect. 8, and the work is summarized in Sect. 9. The experimental data supporting the arguments of this paper are given in Appendix E. We acknowledge that a small part of this work was previously made public through [21].

2. Time-Memory Tradeoff Algorithms

In this section we review the basic theory of time-memory tradeoffs and fix the notation that is used throughout the paper. We introduce previous results that are related to the results here, but make no attempt at providing a complete history or survey of the time-memory tradeoff technique. In particular, the perfect table tradeoffs algorithms are explained, but advancements concerning their analyses or comparisons are not introduced.

Below, after stating some simple technical facts, we describe the three major tradeoff algorithms, and then explain some auxiliary techniques that can enhance their tradeoff efficiency. The descriptions are condensed, and readers that are new to the time-memory tradeoff technique should consult the original papers for more detail.

Throughout this paper, the function $F : \mathcal{N} \rightarrow \mathcal{N}$ will always act on a set $\mathcal{N}$ of size N and the k-times iterated composition $F \circ \cdots \circ F$ of F is written as F^k.

2.1. *Technical Preliminaries*

Many of the results given in this paper are expected values for random functions. In very rough terms, a random function F is a function that assigns independent and random values $F(x) \in \mathcal{N}$ to each of its arguments $x \in \mathcal{N}$. As briefly discussed in [12,16,24], working with a random function is equivalent to choosing a function uniformly at random from the set of all functions of a certain domain and codomain. In other words, any expected value expressed for a random function is an average computed over all functions.

For large positive integers a and b such that $\mathsf{a} = O(\mathsf{b})$, we can use the approximation

$$\left(1 - \frac{1}{\mathsf{b}}\right)^{\mathsf{a}} \approx e^{-\mathsf{a}/\mathsf{b}},$$

which is very accurate. For example, when $\mathsf{a} = \mathsf{b}$, the error in the approximation is bounded by $\frac{e}{\mathsf{b}}$. This approximation is frequently used in the tradeoff literature without any explanation and is also used very frequently in this paper. Its use can be justified through easy computation, which is explicitly carried out in Appendix A.

The final technical fact we present concerns the image size of a random function. Let $F : \mathcal{N} \to \mathcal{N}$ be a random function. If $\mathcal{M} \subset \mathcal{N}$ is of size m_0, then the size of $F(\mathcal{M})$ is expected to be

$$m_1 = \mathsf{N}\left\{1 - \left(1 - \frac{1}{\mathsf{N}}\right)^{m_0}\right\} \approx \mathsf{N}\bigl(1 - e^{-m_0/\mathsf{N}}\bigr). \tag{1}$$

An elementary proof of this statement can be given by treating it as a classical occupancy problem.

More generally, the expected kth iterated image size $m_k = E(|F^k(\mathcal{M})|)$ can be iteratively computed through

$$m_j = \mathsf{N}\bigl(1 - e^{-m_{j-1}/\mathsf{N}}\bigr) \quad (j = 1, \dots, k), \tag{2}$$

starting from $m_0 = |\mathcal{M}|$. This is stated in [11,20] to hold asymptotically. The explicit statements given there are only for the case when the input set $\mathcal{M}$ is the complete domain $\mathcal{N}$, but the case where $\mathcal{M}$ is strictly smaller than the complete domain is used in [23] to state the success probability of a non-perfect rainbow table. The relation between (1) and (2) is discussed in detail in Appendix B.

2.2. *Overview of the Tradeoff Technique*

Let F be fixed to a publicly known one-way function. The goal of any tradeoff algorithm is to recover the input $\mathbf{x}$, when it is given the function image $\mathbf{y} = F(\mathbf{x})$. The *correct answer* $\mathbf{x}$ and the *inversion target* $\mathbf{y}$ may occasionally be referred to as the *password* and *password hash*, respectively.

Any tradeoff algorithm consists of a *precomputation phase* and an *online phase*. The precomputation phase algorithm gathers information about the one-way function F through extensive computation and stores a condensed digest of the gathered information in a *precomputation table*. When an inversion target $\mathbf{y} = F(\mathbf{x})$ is given, the online phase algorithm is executed to recover $\mathbf{x}$ from $\mathbf{y}$, using the precomputation table as reference.

To be meaningful as an attack, the size M of the precomputation table must be smaller than N and the online phase algorithm should return the answer in a time T that is shorter than N. Note that N is the size of the complete dictionary $\{(x, F(x))\}_{x\in\mathcal{N}}$ and is also the time required for an exhaustive search. A tradeoff algorithm should allow tradeoffs between storage and online time in the sense that online attack time T can be reduced by using a larger storage M and, conversely, smaller M can be used if a longer T is acceptable. Tradeoff algorithms are usually implemented with the intention of running a large number of online phases after a single precomputation phase. This gives one justification for a precomputation effort that is larger than exhaustive search.

Even though every implementation of the tradeoff technique works with a specific one-way function F, analyses of the tradeoff techniques are always done with the assumption that F is a random function.

2.3. *Hellman Tradeoff*

The first algorithm we explain is the classical tradeoff algorithm of Hellman [14].

2.3.1. *Parameter Setup*

Certain parameters need to be fixed before the precomputation phase can be started. Positive integers m and t that satisfy the relation $mt^2 \approx \mathsf{N}$ are fixed. This equation is referred to as the *matrix stopping rule*. Another positive integer $\ell \approx t$, which will become the number of tables, is also fixed.

In this paper, we let the parameters m and t satisfy $mt^2 = \mathrm{H}_{\mathrm{msc}}\mathsf{N}$, with a *matrix stopping constant* $\mathrm{H}_{\mathrm{msc}}$ that is neither very large nor too close to zero. Much of the tradeoff literature sets $\mathrm{H}_{\mathrm{msc}} = 1$. The conditions we have given to $\mathrm{H}_{\mathrm{msc}}$ and ℓ may (inaccurately) be expressed as $\mathrm{H}_{\mathrm{msc}} = \Theta(1)$ and $\ell = \Theta(t)$, respectively. The parameters are always assumed to be reasonable in the sense that $1 \ll m, t \ll \mathsf{N}$. The tradeoff algorithms behave somewhat differently when instantiated with extreme parameters.

The *reduction functions* $R_k : \mathcal{N} \to \mathcal{N}$, one for each $k = 1, \dots, \ell$, are fixed. These may be any family of simple bijections that are very easy to compute. When N is a power of 2 and $\mathcal{N}$ consists of non-negative integers less than N, bit permutations or XOR-ing by constants are practical choices for reduction functions. The *colored* iterating functions $F_k : \mathcal{N} \to \mathcal{N}$ are defined through $F_k = R_k \circ F$.

2.3.2. *Precomputation Phase*

In the precomputation phase, the process explained below is repeated ℓ times, once for each $1 \le k \le \ell$, to build ℓ tables.

We start by choosing m random *starting points* $\mathbf{sp}_1^k, \mathbf{sp}_1^k, \dots, \mathbf{sp}_m^k \in \mathcal{N}$. Hellman required each of these starting points to be chosen independently at random, but most researchers today see the starting points as being distinct. For each $1 \le i \le m$, we initially set $\mathbf{x}_{i,0}^k = \mathbf{sp}_i^k$ and recursively compute $\mathbf{x}_{i,j}^k = F_k(\mathbf{x}_{i,j-1}^k)$ for $0 < j \le t$. The final point reached by each chain of iterative computations is said to be an *ending point* $\mathbf{ep}_i^k = \mathbf{x}_{i,t}^k = F_k^t(\mathbf{sp}_i^k)$. The ordered pairs $\{(\mathbf{sp}_i^k, \mathbf{ep}_i^k)\}_{i=1}^m$ are stored as the kth *Hellman table*, after being sorted with respect to the ending points.

The collection of all points $\{\mathbf{x}^k_{i,j}\}_{i,j}$, associated with an iterating function F_k of one *color* k, is said to be a *Hellman matrix* of size $m \times t$. One usually visualizes a Hellman matrix as follows:

$$\begin{array}{l} \mathbf{sp}^k_1 = \mathbf{x}^k_{1,0} \xrightarrow{F_k} \mathbf{x}^k_{1,1} \xrightarrow{F_k} \mathbf{x}^k_{1,2} \xrightarrow{F_k} \cdots \xrightarrow{F_k} \mathbf{x}^k_{1,t-1} \xrightarrow{F_k} \mathbf{x}^k_{1,t} = \mathbf{ep}^k_0, \\ \mathbf{sp}^k_2 = \mathbf{x}^k_{2,0} \xrightarrow{F_k} \mathbf{x}^k_{2,1} \xrightarrow{F_k} \mathbf{x}^k_{2,2} \xrightarrow{F_k} \cdots \xrightarrow{F_k} \mathbf{x}^k_{2,t-1} \xrightarrow{F_k} \mathbf{x}^k_{2,t} = \mathbf{ep}^k_1, \\ \quad \vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ \mathbf{sp}^k_m = \mathbf{x}^k_{m,0} \xrightarrow{F_k} \mathbf{x}^k_{m,1} \xrightarrow{F_k} \mathbf{x}^k_{m,2} \xrightarrow{F_k} \cdots \xrightarrow{F_k} \mathbf{x}^k_{m,t-1} \xrightarrow{F_k} \mathbf{x}^k_{m,t} = \mathbf{ep}^k_m. \end{array}$$

It consists of m *rows* and $t+1$ *columns*. We number the columns so that the starting point column is the 0th column and the ending point column is the tth column. Each row of a Hellman matrix is a *precomputation chain*. Any chain of points from $\mathcal{N}$ that has been formed by iteratively applying an F_k of the same color k is a *Hellman chain*.

2.3.3. *Online Phase*

Once the inversion target $\mathbf{y} = F(\mathbf{x})$ is given, the process explained below is repeated for each $1 \le k \le \ell$, until the correct answer $\mathbf{x}$ is found. Occasionally, the algorithm will report failure in returning the answer after processing all ℓ indices k.

We first compute $\mathbf{y}^k_1 = R_k(\mathbf{y}) = F_k(\mathbf{x})$ and check if this appears as one of the ending points in the kth Hellman table. The table lookup is repeatedly done for each recursively computed $\mathbf{y}^k_j = F_k(\mathbf{y}^k_{j-1})$, until $\mathbf{y}^k_t = F^t_k(\mathbf{x})$ has been searched for in the table. The Hellman chain

$$(\mathbf{x} \xrightarrow{F_k}) \, \mathbf{y}^k_1 \xrightarrow{F_k} \mathbf{y}^k_2 \xrightarrow{F_k} \mathbf{y}^k_3 \xrightarrow{F_k} \cdots \xrightarrow{F_k} \mathbf{y}^k_j$$

that is computed through this process is referred to as the *online chain* for the kth Hellman table.

Whenever a match $\mathbf{y}^k_j = \mathbf{ep}^k_i$ is found, the corresponding starting point $\mathbf{sp}^k_i$ is retrieved from the kth Hellman table, and the associated precomputation chain is (partially) regenerated to obtain $\mathbf{x}_{\text{tmp}} = \mathbf{x}^k_{i,t-j} = F^{t-j}_k(\mathbf{sp}^k_i)$. Since

$$F^j_k(\mathbf{x}_{\text{tmp}}) = F^j_k\big(F^{t-j}_k(\mathbf{sp}^k_i)\big) = \mathbf{ep}^k_i = \mathbf{y}^k_j = F^{j-1}_k(\mathbf{y}_1) = F^j_k(\mathbf{x}),$$

there is a chance that $\mathbf{x}_{\text{tmp}} = \mathbf{x}$. This is why the jth iteration of the online phase for a specific table is sometimes referred to as searching for the answer $\mathbf{x}$ in the $(t-j)$th column of the Hellman matrix. If multiple ending points match the current end of the online chain, one must not forget to regenerate all the corresponding precomputation chains.

Even though the existence of $\mathbf{x}$ in the $(t-j)$th column of a Hellman matrix will surely imply the *collision* of $\mathbf{y}^k_j$ with an ending point, the converse is not true unless F_k is injective. An ending point collision could be caused by a *merge* between the online chain and a precomputation chain. Hence, the online phase algorithm must check whether the candidate answer $\mathbf{x}_{\text{tmp}}$ is the correct answer $\mathbf{x}$. The candidate is clearly incorrect if $F(\mathbf{x}_{\text{tmp}}) \ne \mathbf{y}$, but a full verification requires more information than is contained in $\mathbf{y}$; this is explained in more detail in Sect. 3. If the candidate $\mathbf{x}_{\text{tmp}}$ is found to be incorrect, the event is referred to as a *false alarm*, in which case the online phase resumes the iterative computations of the online chain.

2.3.4. *Success Probability*

The algorithm description for the Hellman tradeoff is complete, and we now give some rough analyses.

The success of inversion is intimately related to how many distinct points are covered by the Hellman matrices. Assume that there are not too many duplicates in an $m \times t$ Hellman matrix and consider the addition of one more precomputation chain to this matrix. The existing Hellman matrix and the new chain contain approximately mt and t points, respectively. Since the matrix stopping rule gives $mt \cdot t \approx \mathsf{N}$, we know from the birthday paradox that there is a high chance that the new chain and the existing Hellman matrix will contain a common element. Hence, the new chain is likely to merge into an existing precomputation chain, and much of the computation that was done to create this additional chain goes to waste. Hence, it makes little sense to continue enlarging a Hellman matrix beyond the $m \times t$ bound set through the matrix stopping rule. This is the reason for using multiple small tables, rather than a very large table. The discussion given so far also indicates that there will not be too many duplicates within the matrix until one comes close to the $m \times t$ bound.

Let us use $|\mathtt{HM}|$ to denote the expected number of distinct nodes contained in a Hellman matrix. The probability of successful inversion after the processing of a single Hellman table is $\frac{|\mathtt{HM}|}{\mathsf{N}}$. Hellman [14] provided the lower bound

$$\frac{|\mathtt{HM}|}{\mathsf{N}} \geq \frac{1}{\mathsf{N}} \sum_{i=1}^{m} \sum_{j=1}^{t} \left(1 - \frac{it}{\mathsf{N}}\right)^{j} \tag{3}$$

and used it to explain the appropriateness of the matrix stopping rule. The arguments given above involving the birthday paradox are from [5,6], and are not found in [14].

When all $\ell \approx t$ tables are processed, assuming that the reduction functions provide independence between tables, the probability of success becomes

$$1 - \left(1 - \frac{|\mathtt{HM}|}{\mathsf{N}}\right)^{\ell} \approx 1 - \exp\left(-\frac{\ell|\mathtt{HM}|}{\mathsf{N}}\right). \tag{4}$$

Since the number of duplicates within each Hellman matrix is kept low by the matrix stopping rule, we have $|\mathtt{HM}| \approx mt$. Recalling that $\ell \approx t$ and applying the matrix stopping rule, we can state that the probability of the Hellman tradeoff in successfully recovering the correct answer $\mathbf{x}$ is approximately $1 - \frac{1}{e} \approx 63.2\ \%$. This is sufficiently large for the Hellman algorithm to be meaningful as an attack.

Interestingly, the original paper [14] does not explicitly express the success probability (4) of the complete algorithm. It is only stated that the inverse of the right-hand side of (3) should be taken as the approximate number of precomputation tables to be created. However, statements similar to (4) may be found in works as far back as [17,18].

In [18], the right-hand side of (3) was carefully approximated, so that the bound could be rewritten as

$$\frac{|\mathtt{HM}|}{\mathsf{N}} \geq \frac{mt}{\mathsf{N}} \frac{1}{\mathrm{H}_{\mathrm{msc}}} \int_0^{\mathrm{H}_{\mathrm{msc}}} \frac{1-e^{-x}}{x}\, dx. \tag{5}$$

The experimental data provided in the work supported the correctness of this bound but also showed that this bound was far from being tight. For example, at $\mathtt{H}_{\mathrm{msc}} = 1$, the test data provided was $\frac{|\mathtt{HM}|}{\mathsf{N}} = 0.85\frac{mt}{\mathsf{N}}$, while the right-hand side of (5) was $0.80\frac{mt}{\mathsf{N}}$.

This discrepancy was resolved in [9,19], which computed the expected value $|\mathtt{HM}|$ itself, rather than its lower bound. This result is copied as Proposition 21 in the main body of the current paper.

The success probability of the Hellman tradeoff was also studied in [27]. However, the inversion problem considered there is different from that considered here. Their analysis is applicable if one wishes to recover *any* pre-image corresponding to a *random image*. The inversion problem considered in [27] is neither of the two inversion problems that are discussed here in Sect. 3.4, in that the inversion target is directly chosen without the involvement of an input.

2.3.5. *Cost of Resolving Alarms*

An upper bound for the number of false alarms per table was given as $\frac{\mathrm{H}_{\mathrm{msc}}}{2}$ in [14]. This was combined with the fact that resolving each alarm requires at most t iterations to argue that the side effects of false alarms on the online time complexity were limited.

A much better bound on the effects of false alarms is given in [18] as

$$\text{(cost of resolving alarms for all tables)} \leq \frac{\mathrm{H}_{\mathrm{msc}}}{6}\ell t. \tag{6}$$

Almost the same content reappears in [15], expressed in the form

$$\text{(expected cost of resolving alarms per table)} = \frac{\mathrm{H}_{\mathrm{msc}}}{6} t. \tag{7}$$

The proofs given by the two papers for the above two statements are essentially identical.

2.3.6. *Tradeoff Curve*

We have $\ell \approx t$ tables, each containing m entries, so that the total storage size is $M = m\ell \approx mt$. Disregarding the time taken to treat false alarms, it takes t iterations of the one-way function to process each of the $\ell \approx t$ tables, so the online time complexity is at most $T \approx t\ell \approx t^2$. Applying the matrix stopping rule to T and M, one can arrive at the *tradeoff curve*

$$TM^2 \approx \mathsf{N}^2 \tag{8}$$

for the Hellman tradeoff.

Conversely, suppose that certain values T and M satisfy the tradeoff curve (8). Then the parameters $t = \sqrt{T}$ and $m = M/\sqrt{T}$ satisfy the matrix stopping rule. When the

Hellman tradeoff is implemented with these t, m, and $\ell \approx t$, it will require storage M and run in online time T.

The tradeoff curve (8) did not appear in the original publication [14]. The above presentation has been adopted from [5,6].

2.4. *DP Tradeoff*

The distinguished point method, which we shall refer to simply as the *DP tradeoff*, is a simple modification of the Hellman tradeoff. The introduction of the DP technique is attributed to Rivest in the book [10], but no corresponding publication can be found. The *perfect table* version of the DP tradeoff was first studied in [7,8], followed by some further analyses in [1,26,28], but literature analyzing the non-perfect DP tradeoffs, which we deal with in this work, is hard to find.

2.4.1. *Parameter Setup*

As in the Hellman tradeoff, one fixes positive integers m and t satisfying the matrix stopping rule $mt^2 \approx \mathsf{N}$. Reduction functions $R_k : \mathcal{N} \to \mathcal{N}$ are chosen and colored iterating functions $F_k = R_k \circ F$ are defined as before. Our work will use the notation $mt^2 = \mathtt{D}_{\mathrm{msc}}\mathsf{N}$ with a *matrix stopping constant* $\mathtt{D}_{\mathrm{msc}} = \Theta(1)$. As in the Hellman tradeoff, $\ell = \Theta(t)$ will be the number of tables. The parameters are always assumed to be reasonable in the sense that $1 \ll m, t \ll \mathsf{N}$.

One fixes a property which is satisfied by a random element of $\mathcal{N}$ with probability $\frac{1}{t}$. This *distinguishing property* should be very easy to check. For example, suppose that t and N are powers of 2 and that the set $\mathcal{N}$ consists of non-negative integers less than N. Then, one usually defines an element of $\mathcal{N}$ to be a *distinguished point*, or a DP, if the first $\log t$ bits of its binary representation are zero.

2.4.2. *Precomputation Phase*

Rather than fixing the length of each precomputation chain to t, the precomputation iterations $\mathbf{x}^k_{i,j} = F_k(\mathbf{x}^k_{i,j-1})$ are continued until the current chain end $\mathbf{x}^k_{i,j}$ is found to be a DP. The resulting m precomputation chains will be of varying lengths, but their average length will be t. As in the Hellman tradeoff, the m starting point and ending point pairs are stored as a *DP table* and ℓ tables are constructed, each corresponding to a different color $1 \leq k \leq \ell$.

Any chain computed through iterative applications of a single F_k that ends at a DP is a *DP chain*. The collection of all precomputed DP chains associated with one DP table is referred to as a *DP matrix*, even though the collection can no longer be visualized as a rectangular shaped matrix.

2.4.3. *Online Phase*

Given the inversion target $\mathbf{y} = F(\mathbf{x})$, the online phase of the DP tradeoff proceeds quite similarly to the Hellman tradeoff online phase. However, since only DPs can be found among the ending points, table lookups are done only when the iteratively computed $\mathbf{y}^k_j$ is found to be a DP. Since no precomputation chain contains a DP in the middle part of the chain, the online chain iterations for any single DP table is terminated at its first DP occurrence.

Resolving alarms is slightly tricky with the DP tradeoffs. Because the length of each precomputation chain is not known, one regenerates the precomputation chain until either $\mathbf{y}_1^k$ is reached or a DP, which sits at the end of the precomputation chain, is reached. One can store the length of each precomputation chain in the DP table [7,8] to remove this problem, but this has the side effect of increasing the precomputation table size, and is not considered here. If multiple ending points match the current end of the online chain, all corresponding precomputation chains need to be regenerated.

2.4.4. *Preliminary Analysis*

The success probability (4) is also valid for the DP tradeoff, when $|\mathtt{HM}|$ is replaced with the number of distinct entries in a DP matrix. Since the average length of the precomputed DP chains is t, each DP matrix covers approximately mt points and the previous rough approximation $1 - \frac{1}{e}$ for the success rate remains valid for the DP tradeoffs. The online chain is likely to reach a DP in approximately t iterations, so that the number of online iterations is $T \approx \ell t \approx t^2$, when the efforts made to resolve alarms are ignored. Combining this with the precomputation table size, which is $M = \ell m \approx mt$, we find that the tradeoff curve (8) is also valid for the DP tradeoff.

2.4.5. *Chain Length Bound*

In practice, a chain may fall into a loop that does not contain a DP and thus may never reach a DP. Hence, any implementation of the DP tradeoff sets a chain length bound [7, 8], which we denote by $\hat{t}$, and any chain that fails to reach a DP within this bound, during either the precomputation phase or the online phase, is discarded. The precomputation phase of a DP tradeoff must generate additional chains to fill in the discarded chains.

Even though some of our results are stated in a way that displays its dependence on $\hat{t}$, we are mainly interested in the case where $\hat{t}$ is sufficiently larger than t. The number of discarded chains is minimized by such a choice, and most of the precomputation is put to good use. Since the precomputation cost is the main barrier to any large-scale implementation of the tradeoff technique, such a choice is natural in practice.

If a chain is generated with the random function, the probability for it to become a DP chain within the chain length bound $\hat{t}$ is

$$1 - \left(1 - \frac{1}{t}\right)^{\hat{t}} \approx 1 - e^{-\hat{t}/t}. \tag{9}$$

This easy statement may be found in [7].

2.5. *Rainbow Tradeoff*

The rainbow table method was introduced by Oechslin [23]. From this point on, we will refer to the rainbow table method simply as the *rainbow tradeoff*.

2.5.1. *Parameter Setup*

One starts with positive integers m and t satisfying the matrix stopping rule $mt \approx \mathsf{N}$. Notice that this equation is different from the matrix stopping rules for the previous two algorithms. In this work, we use the notation $mt = \mathrm{R}_{\mathrm{msc}}\mathsf{N}$ with the *matrix stopping constant* $\mathrm{R}_{\mathrm{msc}} = \Theta(1)$. Unlike the previous two algorithms, a small number of tables $\ell = \Theta(1)$ is used with the rainbow tradeoff. The parameters are always assumed to be reasonable in the sense that $1 \ll m, t \ll \mathsf{N}$. Reduction functions $R^k_j : \mathcal{N} \to \mathcal{N}$ are fixed as before, but these have double indices that are made to run over $j = 1, \dots, t$ and $k = 1, \dots, \ell$. The doubly colored iterating functions are defined through $F_{j,k} = R^k_j \circ F$.

2.5.2. *Precomputation Phase*

Instead of using a single reduction function for each table, t different reduction functions are sequentially applied to create a *precomputation chain* of length t. Each *precomputation table* stores the information from m chains. More explicitly, the ith precomputation chain for the kth rainbow table takes the form

$$\mathbf{sp}^k_i = \mathbf{x}^k_{i,0} \xrightarrow{F_{1,k}} \mathbf{x}^k_{i,1} \xrightarrow{F_{2,k}} \mathbf{x}^k_{i,2} \xrightarrow{F_{3,k}} \cdots \xrightarrow{F_{t-1,k}} \mathbf{x}^k_{1,t-1} \xrightarrow{F_{t,k}} \mathbf{x}^k_{i,t} = \mathbf{ep}^k_i,$$

where $1 \le i \le m$ and $1 \le k \le \ell$. Each of these is a *rainbow chain.*

The complete set of m chains for any fixed k is an $m \times t$ *rainbow matrix,* and the set of pairs $\{(\mathbf{sp}^k_i, \mathbf{ep}^k_i)\}_i$ is stored as the kth *rainbow table* after being sorted on the ending points. Columns of a rainbow matrix are numbered from the 0th, containing the starting points, to the tth, containing the ending points.

2.5.3. *Online Phase*

Let the inversion target $\mathbf{y} = F(\mathbf{x})$ be given for the online phase. For each $j = 1, \dots, t$ and $k = 1, \dots, \ell$, we compute the jth online chain for the kth table

$$(\mathbf{x} \xrightarrow{F_{t-j+1,k}}) \, \mathbf{y}^{k,j}_{t-j+1} \xrightarrow{F_{t-j+2,k}} \mathbf{y}^{k,j}_{t-j+2} \xrightarrow{F_{t-j+3,k}} \cdots \xrightarrow{F_{t-1,k}} \mathbf{y}^{k,j}_{t-1} \xrightarrow{F_{t,k}} \mathbf{y}^{k,j}_{t},$$

through iterative computation, starting from the point

$$\mathbf{y}^{k,j}_{t-j+1} = R^k_{t-j+1}(\mathbf{y}) = F_{t-j+1,k}(\mathbf{x}).$$

After each chain computation, the chain end $\mathbf{y}^{k,j}_t$ is searched for among the ending points of the kth rainbow table. The absence of a collision indicates that the correct answer $\mathbf{x}$ does not belong to the $(t-j)$th column of the rainbow matrix. The appropriate precomputation chain is regenerated whenever a collision is found. Many of these regenerations will lead to the announcement of a *false alarm*.

The order of incrementing the double indices during the online phase requires clarification. One should take the chain length j-index to be the outer loop and the table number k-index to be the inner loop. In other words, for any index j, one computes the jth online chains for all ℓ tables, before computing any of the $(j+1)$th online chains. This is referred to as the parallel processing of rainbow tables. The opposite nesting of the loops is called the sequential processing of tables. As was already noted in [23], the parallel approach is more efficient in terms of the expected number of one-way function invocations. Parallel processing of tables is more commonly considered, and this is the approach we assume throughout this work.

2.5.4. *Success Probability*

In [23], one can find the success probability of a rainbow tradeoff that uses a single table written as

$$1 - \prod_{j=0}^{t-1}\left(1 - \frac{m_j}{\mathsf{N}}\right), \tag{10}$$

where $m_0 = m$ and m_j are recursively computed through (2). However, this was not simplified into a closed form formula there.

While studying the perfect table version of the rainbow tradeoff, the work [1] restricts to the $m = \mathsf{N}$ case and gives the approximation

$$\prod_{j=t-i}^{t-1}\left(1 - \frac{m_j}{\mathsf{N}}\right) \approx \frac{t-i}{t}\frac{t-i+1}{t+1}. \tag{11}$$

Notice that the range of indices in the left-hand side product is shorter than that appearing in (10). The left-hand side product of i terms expresses the probability for the first i online chain computations for a single table (non-perfect) rainbow tradeoff to fail in returning the correct answer $\mathbf{x}$. This expression is valid for any m, even though the right-hand side approximation is appropriate only for $m = \mathsf{N}$.

After almost repeating the computations done by [1], the work [15] obtains a generalization of (11) that is valid for any m. The result is restated as Lemma 28 in the main body of this paper. Neither (11) nor Lemma 28 were explicitly stated as separate results in the referenced papers, but they can be inferred from parts of their proofs.

2.5.5. *Preliminary Analysis*

A collision of points from two rainbow chains will result in merging chains only if the collision occurred at a matching color index. When a new rainbow chain is added to an existing $m \times t$ rainbow matrix that contains no collisions within each column, the probability of not experiencing a merge can be expressed as $(1 - \frac{m}{\mathsf{N}})^t \approx e^{-\frac{mt}{\mathsf{N}}}$. Hence, the matrix stopping rule $mt \approx \mathsf{N}$ is the correct boundary at which collisions among precomputation chains start to become problematic.

Let us assume the use of a single table for the rest of this rough analysis. Ignoring collisions within each rainbow matrix column, the success probability (10) may roughly be approximated as $1 - (1 - \frac{m}{\mathsf{N}})^t \approx 1 - e^{-mt/\mathsf{N}} \approx 1 - \frac{1}{e}$. This is equal to what we saw during the rough analyses for both the Hellman and DP tradeoffs.

Notice that the computations for the jth online chain cannot reuse any of the information computed for previous online chains. Hence, the number of one-way function iterations required for the computation of all online chains is $T = 0 + 1 + \cdots + (t-1) \approx \frac{t^2}{2}$. The storage size for the single rainbow table is $M = m$. Recalling the matrix stopping rule $mt \approx N$, the tradeoff curve can be written as

$$TM^2 \approx \frac{1}{2}\mathsf{N}^2. \tag{12}$$

The above time complexity analysis appears in [23], from which the tradeoff curve directly follows.

2.5.6. *Further Analysis*

The preliminary analysis given above corresponds to the worst case where the complete table is processed. In practice, the online phase is likely to terminate before computing the tth online chain. On the other hand, the cost of resolving alarms has been ignored. Hence, the rough analysis does not give the true worst-case complexity.

The work [15] provides an accurate analysis of the time complexity for rainbow tradeoffs. The expected number of one-way function iterations required to process a single rainbow table was expressed as an explicit rational function of $\mathrm{R}_{\mathrm{msc}}$ times t^2. A similar result for the additional number of one-way function iterations required to process alarms was also stated. However, the results were restricted to the single table case. We do not state their results here, but their results are reobtained if we substitute $\ell = 1$ into (22), appearing in the main body of this paper.

2.6. *Perfect Table Tradeoffs*

The main objective of introducing the DP technique was to reduce the number of table lookups that occur in the Hellman tradeoff. However, it was soon noticed that DPs allow easy detection of merging chains. During the precomputation phase of a *perfect table* version of the DP tradeoff [7,8], one removes chain collisions by keeping only the longest of the merging chains. Chains are additionally generated until m non-merging DP chains have been collected. The resulting perfect DP matrix contains no overlapping points. The online phase of the perfect DP tradeoff is identical to the non-perfect version. The work [7] gives credit to the unpublished work [25] for independently introducing the same algorithm.

Detection of merging chains is also easily done with the rainbow tradeoff. The *perfect table* version of the rainbow tradeoff [23] stores information for just one chain from each set of merging chains. Unlike the DP case, a perfect rainbow matrix may contain overlapping points if they belong to different columns.

The perfect table version of the Hellman tradeoff refers to the case where the Hellman matrix contains no overlapping points. Some discussions may be found in [1,27]. However, generating a perfect Hellman table is costly, and its use is not considered to be practical.

Since there are fewer or no overlaps in a perfect table, these provide better coverage of the search space than their corresponding non-perfect versions for the same amount of storage. Hence, perfect tradeoffs are likely to be more efficient than non-perfect tradeoffs. However, this gain in tradeoff efficiency is paid for with the precomputation that was wasted in generating the discarded chains.

The extra precomputation required for the use of perfect tradeoffs may not seem to be of importance. However, the precomputation cost can be critical when implementing tradeoffs at the limit of one's resources. Consider a large-scale implementation for which the precomputation may take several months on a large cluster of computers. In such a situation, extending the precomputation period by another few months or doubling the number of computers allocated to the precomputation task will not be a viable option, even if it promised a significant advantage in the online tradeoff efficiency.

Even though there are analyses of perfect tradeoffs [1,7,8,15,23,28], dealing with them at the accuracy level aimed for by the current paper is considerably more complicated than for the non-perfect tradeoffs. This is especially true with the perfect DP tradeoffs. In view of relative practicality and theoretic accessibility, we deal only with the non-perfect versions of tradeoff algorithms in this work. Inclusion of the perfect tradeoffs into the comparison results obtained in this paper is left as a subject for future study.

2.7. *Storage Optimization*

The storage size M appearing in the tradeoff curves (8) and (12) refers to the total number of starting point and ending point pairs that need to be stored in the tradeoff tables. In practice, it is important to know the physical size, or the number of bits, required for the table. Each starting point and ending point pair can surely be stored in $2\log \mathsf{N}$ bits, but there are techniques that allow more efficient use of storage.

Below, we assume that a suitable method of enumerating the elements of $\mathcal{N}$ has been fixed and treat elements of $\mathcal{N}$ as $\log \mathsf{N}$-bit integers. This enumeration is trivial when $\mathcal{N}$ is the set of all bit strings of certain length, but may require a small amount of work when $\mathcal{N}$ is given as the set of passwords satisfying certain complicated linguistic structures.

2.7.1. *Consecutive Starting Points*

The first storage reduction technique we review is the use of starting points that require less storage. The work [6] does this while implementing an attack on a specific system, and [7] mentions this as a well-known trick without giving any reference. A clear understanding of random functions shows that the starting points may be chosen in any manner, as long as it has no relation to the graph structure of the specific one-way function under attack.

A practical method of choosing starting points is to use consecutive integers [1]. The integers 0 through $m-1$ will work for any (non-perfect) table. Inter-table collisions among the starting points can be removed by concatenating the table index to the consecutive integers [4]. Note that the table index need only be recorded once for each table. However, the effect of joining table numbers is almost nonexistent on even the second columns of the precomputation matrices, so this detail is not very important. In any case, the starting points can be stored in $\log m$ bits, rather than $\log \mathsf{N}$ bits.

The experiment provided by Hellman [14], supporting the arguments concerning the success probability, was executed with starting points set to small numbers, rather than random points. However, it is not clear if this was intended to reduce the storage size.

2.7.2. *Taking Advantage of the DP Definition*

In the case of DP tradeoffs, any information that can be recovered from the definition of a DP may be removed from the ending point before storage. For example, if a prefix consisting of $\log t$ zero bits defines a DP, the $\log t$ bits of zeros can be removed from each ending point without any loss of information. This method was actively used in [6] and clearly stated in [28], but seems trivial enough to have been widely known before these works.

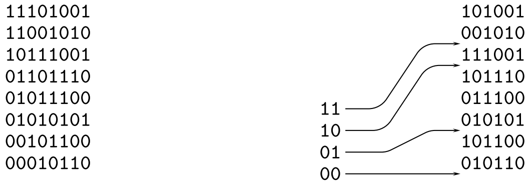


Fig. 1. Index table technique. (The sorted list on the left-hand side is transformed to the right-hand side list, which contains two less bits per entry.)

2.7.3. *Index Table*

The work [6] introduces the *index table* method. This is a degenerate form of a widely known technique called hash tables, which is explained in Appendix D.

To facilitate fast table lookups, the precomputation tables are usually sorted on the ending points before being written to storage. Let us focus on the $\{(\log m) - \varepsilon\}$ most significant bits of each ending point in the sorted table, where ε is any small positive integer. Assuming that the ending points are randomly distributed, for each integer $0 \le i < \frac{m}{2^\varepsilon}$, we can expect to find approximately 2^ε consecutive entries in the sorted table that have the $\{(\log m) - \varepsilon\}$ bit prefix of the ending point equal to integer i. Hence, one can remove $\{(\log m) - \varepsilon\}$ bits from each ending point and replace it with an index table that points to the starting positions for each i value without losing any information. The number of entries contained in the index table is only $\frac{m}{2^\varepsilon}$; hence the additional storage required by its introduction can be ignored. An example is illustrated by Fig. 1.

In practice, the index table could store the number of entries corresponding to each index value rather than the full physical addresses. With such an approach, since only very small number of bits is required to store each count, even the use of $\varepsilon = 0$ could be considered.

2.7.4. *Ending Point Truncation*

The methods described so far reduce the storage size without losing any information concerning each starting point and ending point pair. However, this is not so with the final storage reduction method we describe, which is to simply truncate a part of the ending point before storage.

The truncation of ending points was done in [6] for a specific tradeoff implementation, where it was simply stated that the number of bits they allocate is sufficient for identification purposes. In [4],[1] under the assumption that $m \approx \mathsf{N}^{1/3}$, it is claimed that the ending points of a DP table can be *compressed* to slightly more than $\frac{1}{3}\log \mathsf{N}$ bits. It is also claimed that the ending points for the rainbow tradeoff can be compressed to slightly more than $\frac{2}{3}\log \mathsf{N}$ bits. The paper does not provide any justification for these claims.

During the online phase, when a table lookup is required, the object to be searched for in the table is truncated to the same length and compared with the truncated ending points of the table. The table lookups may now falsely return a match even when a

[1] The paper refers to the Hellman tradeoff, but it seems that the DP tradeoff was implied. Many researchers view the Hellman tradeoff as always incorporating the DP technique.

merge between the online chain and a precomputation chain did not occur. Still, since we were already expecting false alarms, no new measure needs to be devised to deal with the new type of false alarms. Aggressive ending point truncation will cause more frequent false alarms; hence the degree of truncation should be carefully controlled.

The word truncation may give the impression that such a method is applicable only when the space $\mathcal{N}$ consists of bit strings. On spaces that look different, any surjective map that is pre-image uniform, in the sense that the number of pre-images for each element in the range is identical, can serve as the truncation operation. In practice, password hashes are usually bit strings and one does not apply the reduction function at the end of a chain, so truncations can easily be done.

2.8. *Parameter Optimization*

Choosing the parameters m, t, and ℓ for a concrete tradeoff implementation is not an easy task.

The work [18] starts with the assumption that the cost, in dollars, of a tradeoff attack implementation is proportional to the storage size and the number of one-way function computations the online phase machine can perform per unit time. This allows one to consider the lowest possible monetary cost of an attack machine that must succeed with a given probability and finish within a preset real-world time. Expressions giving lower and upper bounds for the optimal cost are presented, and parameters t, m, and ℓ that can achieve the optimal cost are also found. The optimal parameters that are stated depend on the relative cost of storage versus one-way function computations at unit speed.

This analysis is one of the few that takes false alarms into account when computing the time complexity of the online phase. However, the analysis relied on the bounds (5) and (6), which are not very tight, and the upper bound for the optimal cost was simply taken to be an approximation for the optimal cost. Also, while defining the optimal cost, the amount of precomputation was fixed to what is required for a single exhaustive search.

The measure of efficiency used in the current work is different from the monetary cost discussed in [18]. Our interest is in how efficient each tradeoff algorithm is in balancing storage against online time. This balancing ability changes with the amount of precomputation that is invested and the required success rate. The optimal monetary cost for implementation can easily be computed whenever this balancing ability is accurately fixed.

In [17], an attempt was made to optimize the success probability of Hellman tradeoff, while keeping both the time and storage complexities constant. The gain in success probability was paid for with larger precomputation.

There are two parts of their argument that introduce inaccuracy into their results. Since they did not have access to a good expression for the time complexity, it was not possible for them to keep the time complexity exactly constant. They had to be satisfied with keeping ℓt, which is an upper bound for the time complexity in the absence of false alarms, constant. The second point was that they lacked knowledge of the exact success probability and had resorted to using its lower bound given by (5).

The general conclusions of [17] may still be correct, but the details, in particular, the explicit optimal parameters and values, will need to be recomputed with the information given in the current paper. A little more light was shed on the attempt by [23], but the discussion there still relied on rough estimates of time complexity and success probability.

2.9. *Comparison of Tradeoff Algorithms*

Let us attempt a comparison of the three tradeoff algorithms we have explained, based on their tradeoff curves that are already available. Both the Hellman and DP tradeoff curves are given by (8) and the rainbow tradeoff curve is given by (12). Considering the case where the same storage M is given to the three tradeoff algorithms, the tradeoff curves imply that the rainbow tradeoff will require only half the number of one-way function invocations compared to the other two algorithms during the online phase. In addition to giving an argument that is equivalent to what we have just described, the work [23] argues heuristically that the rainbow tradeoff is at an advantage over the DP tradeoff concerning false alarm issues.

The claimed efficiency of the rainbow tradeoff over the DP tradeoff is refuted in [3,4][2] with the observation that the number of physical bits required to store each entry of the tradeoff table has been ignored in [23].

Assume the use of typical parameters $m = t = \ell = \mathsf{N}^{1/3}$ for the DP tradeoff. Recalling the contents of Sect. 2.7, one finds that the starting points for the DP tradeoff can be stored in $\frac{1}{3}\log\mathsf{N}$ bits. It is claimed in [4] that the ending points can first be compressed to slightly more than $\frac{1}{3}\log\mathsf{N}$ bits and then further compressed to a very small number of bits by applying the index table method. Hence each entry of a DP table requires slightly more than $\frac{1}{3}\log\mathsf{N}$ bits to record. In the case of rainbow tradeoffs, one assumes the typical parameters $m = \mathsf{N}^{2/3}$, $t = \mathsf{N}^{1/3}$, and $\ell = 1$. Then each starting point requires $\frac{2}{3}\log\mathsf{N}$ bits. The ending point is first compressed to $\frac{2}{3}\log\mathsf{N}$ bits, and then most of this is removed through the index table method.

Accepting the above arguments, we see that each entry of a rainbow table requires twice the number of bits required by an entry of a DP table. When given the same physical amount of storage, the DP tradeoff can store twice as many starting point and ending point pairs. This translates to a gain in online time by a factor of four through the tradeoff curve. In conclusion, the DP tradeoff will run two times faster than the rainbow tradeoff for the same physical amount of storage.

The more recent work [1] once again advocates the rainbow tradeoff and tries to explain that the arguments of [4] that we have explained so far are misleading. They emphasize that the advantage of the rainbow tradeoff claimed in [23] was by a factor of *at least* two, rather than just two. This is a reasonable point to make, but their ensuing arguments seem to indicate that they were not aware of the ending point truncation method, which was taken into account in [4]. One could interpret this as showing how uninformative [4] was in treating the ending point truncation method.

As we will verify in this work, the claims of [3,4] were mostly correct, but there are hidden issues that can overturn their conclusion. The first is that the tradeoff curves given by (8) and (12) are not accurate. Both of these correspond to the worst case where the algorithms are executed to the end without the correct answer being found. In fact, this was the point made by [1], although it was used to support only the rainbow tradeoff.

[2] It seems that the DP tradeoff was implied, even though the paper refers to the Hellman tradeoff.

One must also note that the effects of false alarms have been ignored by both tradeoff curves so that neither accurately reflects even the worst-case complexity.

The second issue is that the success probabilities of the two algorithms may not be precisely equal at the typical parameters. We have already noted that both algorithms have an approximate success probability of $1 - \frac{1}{e}$ at the typical parameters, but this is an extremely rough estimate, and the running time of a tradeoff algorithm is very sensitive to the required success rate. The controversy explained here is discussed in more detail in Sect. 8.4, after we have developed the necessary tools.

The comparison claims of [23] and [3,4] were made using parameters that require precomputation equal to a single exhaustive search. Recent comparison claims that deal with the perfect tables, which we do not treat in this paper, have the tendency to completely ignore the precomputation cost. Neither approach reflects what can be done in practice. The difficulty of including the precomputation cost into the comparison of tradeoff algorithms seems to have been one reason why perfect tradeoffs have received more focus recently. They certainly appear more attractive, when precomputation is ignored.

2.10. *Checkpoint*

The *checkpoint* [1] technique allows for the resolving of alarms without the regeneration of the precomputation chain. This technique is applicable to both Hellman and rainbow tradeoffs. Application to the DP tradeoff is also possible but slightly more complicated due to the variations in chain lengths.

A column of the precomputation matrix is designated as the *checkpoint* before precomputation. After generation of each precomputation chain, the least significant bit of the chain element that sits at the checkpoint column is appended to the starting point and ending point pair that is to be recorded in the precomputation table. During the online phase, we proceed as usual until an alarm is encountered. At each collision, the online chain is aligned with the colliding precomputation chain at the ending points. If the online chain is long enough, the least significant bits of the two points that belong to the checkpoint column are compared. If the two checkpoint bits do not match, the ending point collision must have resulted from a merge of chains, and the collision is declared a false alarm. If the checkpoint bits do match, the precomputation chain is regenerated as usual to resolve the alarm.

The use of checkpoints filters out some of the efforts spent on precomputation chain regeneration. One can generalize what has been explained to multiple checkpoint columns, consider other methods of extracting a checkpoint bit, or collect more than one bit of information from each checkpoint column.

An analysis of the effects of checkpoints in reducing online time was given by [1] for the perfect rainbow tables. Analyses for Hellman tradeoffs and single table (non-perfect) rainbow tradeoffs were performed in [15]. With a single checkpoint at the optimal position, the Hellman tradeoff online time decreases by 3.17 % at $\mathtt{H}_{\mathrm{msc}} = 1$, and the online time of a single table non-perfect rainbow tradeoff decreases by 5.91 % at $\mathtt{R}_{\mathrm{msc}} = 1$. The effects of checkpoints are more visible at higher $\mathtt{H}_{\mathrm{msc}}$ and $\mathtt{R}_{\mathrm{msc}}$ values.

The advantage of checkpoints must be compared with its side effect on the storage size. After the techniques of Sect. 2.7 have been applied, even a single bit difference in table entry size could translate to a meaningful size ratio change. For example, at

50 bits per table entry, if the increase of a single bit per table entry caused by the use of checkpoints was instead allocated to enlarge the number of table entries, the online time would have reduced by $1-(\frac{50}{51})^2 = 3.88~\%$. This is better than the above-mentioned $3.17~\%$ reduction effect of checkpoints on the Hellman tradeoff, and the $5.91~\%$ reduction effect on rainbow tradeoff should be interpreted as achieving only approximately $2.0~\%$ extra reduction.

Since the effects of checkpoints are small and selective applications of checkpoints will affect all algorithms in the positive direction, its effect on the final comparison of algorithms will be minimal. On the other hand, consideration of the checkpoint technique would add another layer of complication to our analysis. Hence, the analysis given in the current paper does not consider the use of checkpoints. However, we are not claiming that the use of checkpoints should not be considered in practice.

3. Applying Time-Memory Tradeoff to Password Hashes

One usually states the objective of a tradeoff algorithm as the inversion of a one-way function. A closer look reveals that there are two versions of the inversion problem, and we will explain how one of these corresponds to the applications of the tradeoff technique to password hash systems. Issues concerning the use of random functions in the theoretic analysis of tradeoff algorithms are also discussed in this section.

In this section, we refer to the one-way function image as the *password hash* and the input as the *password*.

3.1. *Password Hash*

Let us briefly explain how the security features of many file formats that rely on passwords for access control work in their very basic form.

The designer of the system chooses and fixes a one-way function H. This one-way function is a part of the file format specification and is usually considered to be public. In fact, the one-way function definition can be extracted from the related software even if it was not originally made public. When the owner of a file following this format wants access control to be applied to the file, the user supplies a password $\mathbf{x}$. An encryption key is derived from the password, and the main content of the file is replaced by its encryption under this key. Then the image $\mathbf{y} = H(\mathbf{x})$ of the user password, under the one-way function specified for the file format, is added to the file. Finally, any record of the encryption key and the raw password supplied by the user is destroyed.

Later, when authentication is required for file access, the supporting software asks for a password. The one-way function image $H(\mathbf{x}')$ of the newly supplied password $\mathbf{x}'$ is computed by the software and is compared with the corresponding information $\mathbf{y}$ stored within the file. If a perfect match $\mathbf{y} = H(\mathbf{x}')$ is found, equality $\mathbf{x} = \mathbf{x}'$ is assumed, the main body of the file is decrypted using the key derived from the password $\mathbf{x}'$, and access to the decrypted content is granted. Note that the one-way function image $\mathbf{y}$ of the correct password is stored within the file without any protection and is accessible to anyone that has obtained the file.

The user authentication procedure for computer system logins works in much the same way. At the time of initial user registration to the system, the one-way function image of the password supplied by the user is recorded in a file that is stored within the system. In this case, access to the one-way function images may be harder for the attacker to achieve than in the above case, but this information is often sent over the network in the clear to a group of computers, so that each of these computers may allow authenticated logins to a user that has registered at a central server.

3.2. *Uniqueness of the Pre-image to a Password Hash*

Out of theoretic curiosity, we first ask whether a password hash uniquely determines the password. This should seem obvious in any practical usages of the password hash systems.

Proposition 1. *Let $H : \mathscr{P} \to \mathscr{H}$ be a random function. Given any password $\mathbf{x} \in \mathscr{P}$, the number of inputs that H maps to the password hash $H(\mathbf{x})$ is expected to be $1 + \frac{|\mathscr{P}|-1}{|\mathscr{H}|}$.*

Proof. Since H is a random function, we can first assign a randomly chosen value of $\mathscr{H}$ to $H(\mathbf{x})$ and then define all the other function values. The probability for any one of the later assignments to strike $H(\mathbf{x})$, which is an explicitly fixed value in $\mathscr{P}$, is $\frac{1}{|\mathscr{H}|}$. Each later assignment is independent of all other assignments, and we can expect the number of later assignments to $H(\mathbf{x})$ to be $\frac{|\mathscr{P}|-1}{|\mathscr{H}|}$. □

Readers should not misinterpret the above proposition as giving the pre-image size of a random $\mathbf{y} \in \mathscr{H}$ under a random H. For the random function H, the distribution on $\mathscr{H}$ produced by $H(\mathbf{x})$ is the uniform distribution for each fixed $\mathbf{x} \in \mathscr{P}$, and every $\mathbf{y} \in \mathscr{H}$ is expected to have $\frac{|\mathscr{P}|}{|\mathscr{H}|}$-many pre-images, rather than $1 + \frac{|\mathscr{P}|-1}{|\mathscr{H}|}$. This is not in contradiction with the proposition, as the proposition deals with the distribution on $\mathscr{H}$ produced from random inputs by the specific H that has been constructed, and this is different from the uniform distribution on $\mathscr{H}$. Those points of $\mathscr{H}$ that lie outside $H(\mathscr{P})$, for the specifically constructed H, do not have any chance of appearing.

One can also ask for the pre-image size of a random password hash $\mathbf{y} \in H(\mathscr{P})$. Note that this question can only be asked after the random function H has been fully constructed. The corresponding answer will depend on the size of $H(\mathscr{P})$, but, when $|\mathscr{P}| = |\mathscr{H}|$, this should be close to

$$\frac{|\mathscr{P}|}{E(|H(\mathscr{P})|)} \approx \frac{1}{1 - \frac{1}{e}} \approx 1.582.$$

Once again, this question is not related to the content of the above proposition. It deals with the uniform distribution on $H(\mathscr{P})$, which is different from the distribution on $H(\mathscr{P})$ given by the fully specified H. Those points with larger pre-image sets will have a larger probability of appearing than those with smaller pre-image sets.

Consider an application of the tradeoff technique to a block cipher whose key length is equal to its block length. In such a case, one is working with $|\mathscr{P}| = |\mathscr{H}|$, and Proposition 1 states that there will be approximately two keys, on average, that map to a given target ciphertext. This is probably larger than what many would have naively expected. Of course, in practice, one usually assumes the use of a second ciphertext to almost uniquely identify the key. In fact, if one interprets the key to two ciphertexts mapping as a new one-way function, then Proposition 1 claims that the key is almost uniquely determined from the two ciphertexts.

Let us next discuss what Proposition 1 implies for systems that rely on passwords for access control. These systems are usually designed so that the space $\mathscr{H}$ of potential hash values is significantly larger than the space $\mathscr{P}$ of admissible passwords. A typical password hash would be a bit string of at least 128 bits in length, and the number of alphanumeric passwords consisting of ten characters is only $62^{10} \approx 2^{59.5}$. In such a case, Proposition 1 shows that a password hash $H(\mathbf{x})$, produced from a password $\mathbf{x}$, will almost always identify $\mathbf{x}$ uniquely.

Furthermore, in practice, the set of all passwords admissible by the security system is not of much importance. Since human-generated passwords are not uniformly distributed within the complete admissible password space, the tradeoff attacker first fixes a manageable subset $\mathscr{P}' \subset \mathscr{P}$ from the set of all passwords and decides to be satisfied with recovering only those passwords that lie in $\mathscr{P}'$. The size of this subset is determined by the computational power that the attacker can allocate to the precomputation phase and should preferably cover the passwords that are most likely to be used. In fact, it has been shown [22] that human-memorable passwords can be enumerated efficiently. Under such a setting the password hash set $\mathscr{H}$ is immensely larger than the set of passwords $\mathscr{P}'$ that is being considered, and hence the password hash determines the password uniquely.

For the remainder of this paper, we assume that the target system for the application of the tradeoff technique is such that $|\mathscr{P}| \ll |\mathscr{H}|$, implying that the password hash uniquely determines the password.

3.3. *The Reduction Function*

The tradeoff technique requires the one-way function to be iterated. Since the codomain $\mathscr{H}$ of the one-way function $H : \mathscr{P} \to \mathscr{H}$ is usually larger than the domain $\mathscr{P}$, iteration is achieved by utilizing a *reduction function* $R : \mathscr{H} \to \mathscr{P}$. One role of the reduction function is to let a password hash be interpreted as another password. As any theoretic treatment of the tradeoff technique assumes $R \circ H$ to be a random function, let us check whether this is appropriate.

Proposition 2. *Let $|\mathscr{P}|$ be a divisor of $|\mathscr{H}|$, so that $\frac{|\mathscr{H}|}{|\mathscr{P}|}$ is an integer. Let $R : \mathscr{H} \to \mathscr{P}$ be any fixed function that is pre-image uniform in the sense that it is exactly $\frac{|\mathscr{H}|}{|\mathscr{P}|}$-to-1. If $H : \mathscr{P} \to \mathscr{H}$ is a random function, then $R \circ H : \mathscr{P} \to \mathscr{P}$ is a random function.*

Proof. In more precise terms, we want to show that the distribution on $\mathscr{P}^{\mathscr{P}}$, produced from the uniform distribution on $\mathscr{H}^{\mathscr{P}}$, through the mapping $H \mapsto R \circ H$, is a uniform distribution.

Let $F_0 : \mathscr{P} \to \mathscr{P}$ be any specific function. It suffices to show that, after random construction of a function $H : \mathscr{P} \to \mathscr{H}$, we will find $R \circ H = F_0$ with probability $\frac{1}{|\mathscr{P}|^{|\mathscr{P}|}}$. Note that $\{R^{-1}(\mathbf{z})\}_{\mathbf{z} \in \mathscr{P}}$ is a partition of $\mathscr{H}$ into cells of size $\frac{|\mathscr{H}|}{|\mathscr{P}|}$. The event $F_0 = R \circ H$ will happen if and only if the value assigned as $H(\mathbf{x})$ belongs to the cell $R^{-1}(F_0(\mathbf{x}))$, for every $\mathbf{x} \in \mathscr{P}$. Since the size of $R^{-1}(F_0(\mathbf{x}))$ is always $\frac{|\mathscr{H}|}{|\mathscr{P}|}$, and since the assignment to $H(\mathbf{x})$ is independent and random for every $\mathbf{x}$, the probability of arriving at $F_0 = R \circ H$ is

$$\left(\frac{|\mathscr{H}|/|\mathscr{P}|}{|\mathscr{H}|}\right)^{|\mathscr{P}|} = \frac{1}{|\mathscr{P}|^{|\mathscr{P}|}},$$

as claimed. □

Every application of the time-memory tradeoff technique to a security system involves a specific one-way function $H : \mathscr{P} \to \mathscr{H}$, and there is no strictly logical reason to believe that the specific H will display the properties expected of a random function. Hence we need to discuss if predicting the behavior of an explicit tradeoff implementation with arguments concerning random functions can be justified in practice.

There are two ways to resolve this problem. The first is to appeal to our intuition. When one ignores his knowledge of the inner working of the given specific function, it will seem as if the function is returning independently and randomly generated values to each given input. Hence, viewed from the outside, it looks as if the specific function is a random function in the construction sense. The second argument, which seems slightly more plausible, is that the one-way function used in the security system is in fact a function that has been selected from the pool of all functions. Unless we had chosen the one-way function in an unusual way, any property exhibited by a specific function will be close to the property averaged over all functions. Further discussion related to this second argument may be found in Appendix B.

We have thus partly justified the use of random functions in place of specific one-way functions $H : \mathscr{P} \to \mathscr{H}$ when analyzing the behavior of time-memory tradeoffs. What we have shown through Proposition 2 is that if we may treat the specific one-way function H as a random function, then the same can be done with the function $R \circ H : \mathscr{P} \to \mathscr{P}$. Hence, throughout this paper, while analyzing the behavior of time-memory tradeoffs, we shall work with a random function $F : \mathscr{N} \to \mathscr{N}$ whose domain and codomain coincide.

3.4. *Two Versions of the Inversion Problem*

Discussions of this subsection should be read with the Hellman tradeoff in mind. However, the content can easily be translated to language that is appropriate for any other tradeoff algorithm.

We have already mentioned that we shall work in the situation $H : \mathscr{P} \to \mathscr{H}$ where the sets satisfy $|\mathscr{P}| \ll |\mathscr{H}|$, so that a password hash almost always determines a unique password. We also know that any analysis of time-memory tradeoff behavior is usually done with a random function $F : \mathscr{N} \to \mathscr{N}$, whose image does not uniquely determine the input. In actual implementations, reduction functions $R_k : \mathscr{H} \to \mathscr{P}$ are defined and the online phase algorithm works with the colored iterating functions $H_k = R_k \circ H : \mathscr{P} \to \mathscr{P}$.

The unique password $\mathbf{x}$ corresponding to inversion target $\mathbf{y} = H(\mathbf{x})$ is obtained through the tradeoff algorithm as follows. The online phase algorithm is given $\mathbf{y}$, and $R_k(\mathbf{y}) = H_k(\mathbf{x})$ is passed onto its sub-algorithm that processes the kth table. The best the sub-algorithm can do is return inputs $x \in \mathscr{P}$ satisfying $H_k(x) = H_k(\mathbf{x})$. Since this relation is weaker than $x = \mathbf{x}$, the parent algorithm must verify whether the password candidate x is the correct password $\mathbf{x}$ by testing the relation $H(x) = \mathbf{y}$.

Let us discuss how often during the online phase such candidate checks need to be performed. Assume that the precomputation algorithm required $\varepsilon|\mathscr{P}|$ iterations of H to complete. We will have $\varepsilon = \Theta(1)$ in practice. For exactly the same reason given in the proof of Proposition 1, the expectation for the number of x appearing in the kth Hellman matrix that maps to $H_k(\mathbf{x})$ under H_k, combined over all k, is upper bounded by $\varepsilon + 1$, which is a small number. Hence, the cost of such candidate checks may safely be ignored.

During a tradeoff algorithm analysis, one does not mention anything about H or R, the source of the inversion problem, and simply assumes that the inversion target $\mathbf{y} = F(\mathbf{x})$ is given, for some function $F : \mathscr{N} \to \mathscr{N}$. Note that in this setting, the password hash $\mathbf{y}$ does not uniquely determine the password $\mathbf{x}$. However, the goal of the tradeoff algorithm in this paper will be to find *the* correct password $\mathbf{x}$ that was used to create $\mathbf{y}$, rather than *any* password x that corresponds to the given $\mathbf{y}$ through $F(x) = \mathbf{y}$. The *any* version may be useful when working to find the pre-image of a cryptographic hash function, but the *the* version is suitable when looking for the correct password to an access control mechanism. A clear distinction between these two inversion problems was first made in [15].

Since it is logically impossible to distinguish between the many pre-images with only the $\mathbf{y} \in \mathscr{N}$ information, our analysis will focus on whether $\mathbf{x}$ is among the possibly multiple pre-images to $\mathbf{y}$, returned by the tradeoff algorithm. The determination of whether each returned value is *the* correct password is assumed to be done outside the tradeoff algorithm.

The difference between looking for *the* pre-image versus *any* pre-image implies that the tradeoff algorithm will succeed under different circumstances. The *the* version succeeds if and only if the correct password $\mathbf{x}$ had appeared as an *input* to the one-way function F during the precomputation phase, i.e., if $\mathbf{x}$ is among the precomputation matrix entries excluding the ending points. On the other hand, the *any* version succeeds if and only if the image $\mathbf{y} = F(\mathbf{x})$ had appeared as the function *output* during the precomputation phase, i.e., if $\mathbf{y}$ is among the precomputation matrix entries excluding the starting points. The two approaches will show differences in properties such as success probability and online running time.

Let us add a final word of caution—both inversion problems we have discussed require the target $\mathbf{y} = F(\mathbf{x})$ to be fixed through a random choice of the *input* $\mathbf{x}$. One should distinguish this from the case where the inversion target is directly chosen at random from either the image space or the codomain. These variants do not seem to fit any naturally occurring real-world situation.

4. DP Tradeoff

A complexity analysis of the DP tradeoff is given in this section. We present a formula for computing the probability of success for the non-perfect DP algorithm and provide a tradeoff curve which takes the effects of false alarms into account. We also discuss the number of bits required to efficiently store the starting point and ending point pairs.

In this work, to simplify some of our proofs, we assume that the starting points are always chosen among non-DPs. Hence, in a precomputed DP chain, every point preceding the ending point, including the starting point, is a non-DP. A rigorous treatment that allows starting points to be DPs can be performed, but differences between results from such an analysis and those presented in this work will be negligible.

Recall the probability for a random chain to become a DP chain within the chain length bound $\hat{t}$, given by (9). Rather than requiring each table to contain exactly m entries, we assume that each precomputation DP matrix is always generated from

$$m_0 = \frac{m}{1 - e^{-\hat{t}/t}} \tag{13}$$

distinct starting points. Then we can expect to collect approximately m chains that terminate at DPs.

All of our tradeoff algorithm analyses are done under the assumption that the one-way function is the random function. In particular, many expectations mentioned hereinafter are to be understood as averages made over the choice of all functions. Most of our arguments will be made over a single table, so we remove the display of dependence on the reduction functions from all the notation.

4.1. *Probability of Success*

Let us discuss the probability of success for a DP tradeoff under a given set of parameters. We first present a general formula connecting precomputation and probability of success and then show how to compute these for specific parameters. Our first lemma is quite trivial.

Lemma 3. *The number of one-way function invocations required in either creating a DP chain or stopping at the $\hat{t}$th iteration without having reached a DP is expected to be*

$$t\left(1 - e^{-\hat{t}/t}\right).$$

Proof. It suffices to add the probabilities of having to compute the successive iterations. Since the next iteration is computed if and only if a DP has not yet been reached, the expected one-way function invocation count is

$$\sum_{i=1}^{\hat{t}} \left(1 - \frac{1}{t}\right)^{i-1} = t\left\{1 - \left(1 - \frac{1}{t}\right)^{\hat{t}}\right\},$$

which we can approximate to what is stated. □

In the above proof, we have implicitly assumed the one-way function to be a random function and computed the probability for the first i assignments to be non-DPs. A more exact analysis would additionally consider the possibility for the next assignment to produce a previously assigned value. We have not done so because the above result was good enough as an approximation.

Clearly, the success rate of a tradeoff algorithm is intimately connected to the amount of precomputation, so let us present a way to write down the precomputation.

Proposition 4. *The precomputation phase of the DP tradeoff is expected to require $mt\ell$ one-way function invocations.*

Proof. We know from Lemma 3 that each attempt at a DP chain creation is expected to require $t(1-e^{-\hat{t}/t})$ one-way function invocations. Recall that the creation of a single DP table is to start with $m_0 = \frac{m}{1-e^{-\hat{t}/t}}$ chains. Together, these imply that the creation of a single DP table is expected to consume mt one-way function invocations. Hence, the total precomputation requirement may be written as $mt\ell$. □

This proposition is trivially true when the chain length bound is not set, but what we have shown is that the precomputation cost does not depend on the chain length bound. We define the *precomputation coefficient* for the DP tradeoff to be $\mathtt{D}_{\mathrm{pc}} = \frac{mt\ell}{\mathsf{N}}$, so that the precomputation cost of a DP tradeoff is $\mathtt{D}_{\mathrm{pc}}\mathsf{N}$.

The *coverage rate* $\mathtt{D}_{\mathrm{cr}}$ of a DP table is defined to be the expected number of distinct nodes that appear among the DP chains as inputs to the one-way function, divided by mt. Since our starting points are always non-DPs, all of the nodes that are counted will be non-DPs. The mentioned expectation is an average over the choice of one-way functions. In other words, the coverage rate is a certain expected value for the random function. Our next statement reduces the search for success rate to the computation of the coverage rate.

Proposition 5. *The success probability of the DP tradeoff is*

$$\mathtt{D}_{\mathrm{ps}} = 1 - e^{-\mathtt{D}_{\mathrm{cr}}\mathtt{D}_{\mathrm{pc}}}.$$

Proof. If we are given $\mathbf{y} = F(\mathbf{x})$ as the inversion target, the DP tradeoff will succeed in recovering the correct answer $\mathbf{x}$, if and only if $\mathbf{x}$ had appeared as one of the inputs to the one-way function during the creation of the DP table. As was discussed in Sect. 3.4, this is not equivalent to asking for the appearance of $\mathbf{y}$ among the output values. The objective of recovering *the correct*, rather than *any* inverse, corresponds to finding $\mathbf{x}$ among the one-way function inputs.

By definition of the coverage rate, a single DP matrix is expected to contain $\mathtt{D}_{\mathrm{cr}}mt$ distinct nodes that were used as inputs to the one-way function. Hence the processing of a single table will fail in returning the correct answer with probability $(1-\frac{\mathtt{D}_{\mathrm{cr}}mt}{\mathsf{N}})$. The success probability of the complete DP tradeoff process is given by

$$\mathtt{D}_{\mathrm{ps}} = 1 - \left(1 - \frac{\mathtt{D}_{\mathrm{cr}}mt}{\mathsf{N}}\right)^{\ell} \approx 1 - \exp\left(-\mathtt{D}_{\mathrm{cr}}\frac{mt\ell}{\mathsf{N}}\right) = 1 - e^{-\mathtt{D}_{\mathrm{cr}}\mathtt{D}_{\mathrm{pc}}},$$

assuming that the multiple tables are independent. □

We confide that our treatment in the proof of separate tables as being independent does not strictly conform to the assumption of F being a *single* random function.

This lemma is almost identical to (4), which had already appeared in many works. We wrote out the proof in detail, only because most previous works did not clarify whether the inputs or outputs of the random functions were being counted. In fact, many of them did not even clarify which version of the inversion problem was being considered, as it did not matter for their intended rough analysis.

If the creator of the inversion target $\mathbf{y} = F(\mathbf{x})$ chooses $\mathbf{x}$ to be a DP, the online phase will definitely fail. The success probability would be very low for such challenges even if the starting points were allowed to be DPs. For our analysis to be applicable, the challenge $\mathbf{x}$ needs to be chosen without reference to the structure of the DP tradeoff table. Note that this is not as strong a requirement as asking for the choice of $\mathbf{x}$ to be random. In practice, since distinguishing properties are defined with reference to the password hashes rather than the passwords, such challenges do not cause any problem.

For the remainder of this subsection, all chains belonging to the DP matrix will be seen as having been aligned at the starting points, rather than at the ending points, and the starting point column will be referred to as the 0th column.

The above expression for probability of success can only be put to use if we know how to compute the coverage rate. Our computation of the coverage rate will be done in two steps. Of the m_0 chains generated, only m will be DP chains, but we disregard this in the first step and count the number of new nodes added by each column of the extended matrix. The sum of these values is the total number of all distinct input entries. In the second step, we will count the number of nodes that belonged to chains not ending at DPs and subtract these from the total count.

Let us write m_j for the number of new non-DP nodes added by the jth column. The number m_0 of distinct starting points, stated by (13), conforms to this notation.

Lemma 6. *The number of new non-DP nodes added by each column satisfies the recurrence relation*

$$m_j = \mathsf{N}\left\{1 - \exp\left(-\frac{m_{j-1}}{\mathsf{N}}\right)\right\}\left(1 - \frac{1}{t}\right)\left(1 - \frac{\sum_{i=0}^{j-1} m_i}{\mathsf{N}(1-1/t)}\right).$$

Proof. Suppose a node positioned in the $(j-1)$th column is old, in the sense that it has appeared in one of the 0th through $(j-2)$th columns. Application of the random function to this node will not result in a random element of $\mathcal{N}$, but a node that had appeared in one of the 1-st through $(j-1)$th columns. Hence when counting new nodes of the jth column we need only consider the nodes of the jth column that are assigned as images to new nodes of the $(j-1)$th column. Recalling (1), we write this as the $\mathsf{N}\{1 - \exp(-\frac{m_{j-1}}{\mathsf{N}})\}$ part appearing in the claimed equation.

Of the distinct entries that have appeared in the jth column, that are not automatically old, we want to filter out the DPs. The previous count is made to correspond to the non-DPs by multiplying by a $(1 - \frac{1}{t})$ factor.

Still, not all of these non-DPs are new nodes. Those that have appeared in previous columns are removed by multiplying $(1 - \frac{\sum_i m_i}{\mathsf{N}(1-1/t)})$. Notice that we have $\mathsf{N}(1 - \frac{1}{t})$, rather than N, in the denominator, as we are dealing only with non-DPs at this point. □

The next two lemmas are technical computation results. We first turn the recursive formula for m_j into a difference equation concerning a certain sum of m_j.

Lemma 7. *Let* $\mu_i = \frac{m_i}{\mathsf{N}(1-1/t)}$ *and* $\sigma_j = \sum_{i=0}^{j-1} \mu_i$. *Then,* σ_j *satisfies the recursive formula*

$$\sigma_{j+1} - \sigma_j = \frac{m_0}{\mathsf{N}} - \frac{1}{t}\sigma_j - \frac{1}{2}\sigma_j^2 \quad \textit{with } \sigma_0 = 0,$$

which is accurate up to modulo $O(\frac{1}{t^3})$.

Proof. It is straightforward to rewrite the recursive formula of Lemma 6 in terms of the notation μ_j:

$$\mu_j = \left\{1 - \exp\left(-\left(1 - \frac{1}{t}\right)\mu_{j-1}\right)\right\}\left(1 - \sum_{i=0}^{j-1} \mu_i\right).$$

This may be rewritten once again as

$$\exp\left(-\left(1 - \frac{1}{t}\right)\mu_{j-1}\right) = 1 - \frac{\mu_j}{1 - \sigma_j} = \frac{1 - \sigma_{j+1}}{1 - \sigma_j}.$$

Now, by taking products of both sides over $j = 1, \ldots, k$, we obtain

$$\exp\left(-\left(1 - \frac{1}{t}\right)\sigma_k\right) = \frac{1 - \sigma_{k+1}}{1 - \sigma_1}.$$

We have thus arrived at a relation involving only the σ_k notation.

By expanding the exponential function in its Taylor series, we obtain

$$\sigma_{k+1} = 1 - (1 - \sigma_1)\left\{1 - \left(1 - \frac{1}{t}\right)\sigma_k + \frac{1}{2}\left(1 - \frac{1}{t}\right)^2 \sigma_k^2 - \cdots\right\},$$

and we can modify the above into the difference equation

$$\sigma_{k+1} - \sigma_k = \sigma_1 - \left(\sigma_1 + \frac{1}{t} - \frac{\sigma_1}{t}\right)\sigma_k - \frac{1}{2}(1 - \sigma_1)\left(1 - \frac{1}{t}\right)^2 \sigma_k^2 + \cdots .$$

Noting that the left-hand side $\sigma_{k+1} - \sigma_k = \mu_k$ is of order $O(\frac{m}{\mathsf{N}}) = O(\frac{1}{t^2})$, we remove every term on the right-hand side of $O(\frac{1}{t^3})$ order. This may easily be done after noting that $\sigma_1 = \mu_0$ is $O(\frac{1}{t^2})$ and that σ_k is $O(\frac{mk}{\mathsf{N}})$, which is at most $O(\frac{1}{t})$. The simplified equation is now

$$\sigma_{k+1} - \sigma_k = \mu_0 - \frac{1}{t}\sigma_k - \frac{1}{2}\sigma_k^2 + O\left(\frac{1}{t^3}\right).$$

It is clear that the initial condition $\sigma_1 = \mu_0$ may be replaced by $\sigma_0 = 0$, under this recursive formula. As a final tweak, we subtract $\frac{m_0}{\mathsf{N}(t-1)}$, which is of $O(\frac{1}{t^3})$ order, from the constant term $\mu_0 = \frac{m_0}{\mathsf{N}(1-1/t)} = \frac{m_0}{\mathsf{N}}(1 + \frac{1}{t-1})$, to arrive at the claimed formula. □

Now that we have a difference equation, we can obtain σ_k through an application of the Euler method.

Lemma 8. *For each non-negative integer k, we have*

$$m_k \approx \mathsf{N}\big(\sigma(k+1) - \sigma(k)\big)$$

where

$$\sigma(k) = \frac{\Xi^2 - 1}{t} \frac{\exp(\Xi \frac{k}{t}) - 1}{(\Xi + 1)\exp(\Xi \frac{k}{t}) + (\Xi - 1)} \quad \textit{with } \Xi = \sqrt{1 + \frac{2\mathrm{D}_{\mathrm{msc}}}{1 - e^{-\hat{t}/t}}}.$$

Proof. Let a function $\sigma : \mathbf{R} \to \mathbf{R}$ be the unique solution to the differential equation

$$\frac{d}{dk}\sigma = \frac{m_0}{\mathsf{N}} - \frac{1}{t}\sigma - \frac{1}{2}\sigma^2 \quad \text{and} \quad \sigma(0) = 0. \tag{14}$$

If one defines the sequence $\{\sigma_k\}_{k \geq 0}$ through the corresponding difference equation

$$\sigma_{k+1} - \sigma_k = \frac{m_0}{\mathsf{N}} - \frac{1}{t}\sigma_k - \frac{1}{2}\sigma_k^2 \quad \text{and} \quad \sigma_0 = 0, \tag{15}$$

then the Euler method tells us that $\sigma(k)$, the evaluation of the function σ at the non-negative integer k, may be approximated by the sequence value σ_k. We may turn this the other way around to present approximate values of σ_k through the function evaluations $\sigma(k)$.

The unique solution to differential equation (14) is

$$\sigma(k) = \frac{2m_0 t}{\mathsf{N}} \frac{\exp(\sqrt{1 + \frac{2m_0t^2}{\mathsf{N}}}\frac{k}{t}) - 1}{(\sqrt{1 + \frac{2m_0t^2}{\mathsf{N}}} + 1)\exp(\sqrt{1 + \frac{2m_0t^2}{\mathsf{N}}}\frac{k}{t}) + (\sqrt{1 + \frac{2m_0t^2}{\mathsf{N}}} - 1)}.$$

The form of $\sigma(k)$ stated by this lemma is obtained when (13) and $mt^2 = \mathrm{D}_{\mathrm{msc}}\mathsf{N}$ are substituted.

Since the definition of σ_k given by (15) is identical to the approximate recursive relation of Lemma 7, we have

$$\sigma(k) \approx \sigma_k = \sum_{i=0}^{k-1} \mu_i, \quad \text{where } \mu_i = \frac{m_i}{\mathsf{N}(1 - 1/t)}.$$

This allows us to write

$$m_k \approx \mathsf{N}\left(1 - \frac{1}{t}\right)\big(\sigma(k+1) - \sigma(k)\big) \approx \mathsf{N}\big(\sigma(k+1) - \sigma(k)\big),$$

where the $\frac{1}{t}$ term removal is justifiable, as it is of strictly smaller order. □

This completes the first step of the coverage rate computation. The coverage rate corresponds to the number of distinct non-DP nodes contained in just the DP chains, but the currently computed m_k includes all points contained in even the non-DP chains. We need to account for these nodes belonging to non-DP chain nodes. This is the second step in finding the coverage rate.

Proposition 9. *The coverage rate of a single DP table is expected to be*

$$\mathrm{D}_{\mathrm{cr}} = \frac{2}{e^{\hat{t}/t}-1}\int_0^{\hat{t}/t}\frac{\exp(\Xi u)-1}{(\Xi+1)\exp(\Xi u)+(\Xi-1)}\exp(u)\,du,$$

where $\Xi = \sqrt{1+\frac{2\mathrm{D}_{\mathrm{msc}}}{1-e^{-\hat{t}/t}}}$.

Proof. To count the number of distinct non-DPs belonging to all DP chains, we need to subtract the number of all new points belonging to non-DP chains from $\sum_{i=0}^{\hat{t}-1} m_i$. Before doing this, we first need to consider whether any of these points may not also appear within a DP chain and take the status of being a new point when the non-DP chain is removed.

It is clear that any new node belonging to a non-DP chain cannot have appeared in a column previous to its position, as the node is supposed to be new. Furthermore, such a node cannot appear within the DP chains in the same column or any future columns, since it would then reach a DP before the chain length bound is exceeded. Hence new nodes belonging to non-DP chains do not appear within any DP chains, and we may safely remove all of these new points without worrying about their possible contribution to coverage by DP chains.

Now, let us count how many points belong to non-DP chains, one column at a time. We start with the 0th column. Among all m_0 chains, even though we do not know ahead of time which ones they would turn out to be, there will be $m_0(1-\frac{1}{t})^{\hat{t}}$ chains that do not reach a DP even after $\hat{t}$ more iterations. Hence $m_0(1-\frac{1}{t})^{\hat{t}}$ nodes among the m_0 nodes belonging to the 0th column need to be removed from the count of new nodes. As for the 1st column, we had focused on m_1 chains, but $m_1(1-\frac{1}{t})^{\hat{t}-1}$ nodes among these will not reach a DP before exceeding the chain length bound, and they need to be removed. The general term is now clear.

The coverage rate of a single DP table can thus be stated as

$$\frac{1}{mt}\sum_{k=0}^{\hat{t}-1} m_k\left\{1-\left(1-\frac{1}{t}\right)^{\hat{t}-k}\right\}.$$

Using Lemma 8, we can approximate this to

$$\frac{1}{mt}\sum_{k=0}^{\hat{t}-1}\mathsf{N}\big(\sigma(k+1)-\sigma(k)\big)\left\{1-\left(1-\frac{1}{t}\right)^{\hat{t}-k}\right\}$$

$$=\frac{\mathsf{N}}{mt}\frac{1}{t}\sigma(\hat{t})+\frac{\mathsf{N}}{mt}\sum_{k=0}^{\hat{t}-1}\sigma(k)\left(1-\frac{1}{t}\right)^{\hat{t}-k}\frac{1}{t}$$

$$\approx\frac{\sigma(\hat{t})}{\mathrm{D}_{\mathrm{msc}}}+\frac{t}{\mathrm{D}_{\mathrm{msc}}}\exp\left(-\frac{\hat{t}}{t}\right)\sum_{k=0}^{\hat{t}-1}\sigma(k)\exp\left(\frac{k}{t}\right)\frac{1}{t}.$$

Since the coverage rate is of $O(1)$ order and the first term $\frac{\sigma(\hat{t})}{\mathrm{D}_{\mathrm{msc}}}$ is of $O(\frac{1}{t})$ order, we simply discard the first term, and the summation term can be approximated by the integral

$$\frac{t}{\mathrm{D}_{\mathrm{msc}}} e^{-\hat{t}/t} \int_0^{\hat{t}/t} \sigma(tu) \exp(u)\, du,$$

when $\frac{1}{t}$ is small. The claimed formula follows after substitution of $\sigma(tu)$, as given by Lemma 8, and some simplifications. □

We state the case where $\hat{t}$ is sufficiently large separately for later use.

Proposition 10. *The expected coverage rate of a single DP table is approximately*

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

when the chain length bound $\hat{t}$ is sufficiently large.

Proof. When the chain length bound $\hat{t}$ is sufficiently large, almost all of the $m_0 \approx m$ chains that are generated will terminate with a DP, and hence the coverage rate may be computed as $\frac{1}{mt} \sum_{i=0}^{\hat{t}-1} m_i$.

Based on Lemma 8, we may write

$$\mathrm{D}_{\mathrm{cr}} \approx \frac{\sum_{i=0}^{\hat{t}-1} m_i}{mt} = \frac{\mathsf{N}\sigma(\hat{t})}{mt} = \frac{2}{1 - e^{-\hat{t}/t}} \frac{e^{\Xi \hat{t}/t} - 1}{(\Xi + 1) e^{\Xi \hat{t}/t} + (\Xi - 1)},$$

where $\Xi = \sqrt{1 + \frac{2\mathrm{D}_{\mathrm{msc}}}{1 - e^{-\hat{t}/t}}}$. When $\hat{t}$ is sufficiently larger than t, this is approximate to what is claimed. □

A careful reading of this proof shows that $\frac{\hat{t}}{t}$ does not need to be very large for the final approximation to be accurate. A ratio between $\hat{t}$ and t of such a not-too-large order is all we assume when we use the expression $\hat{t}$ *is sufficiently large*. We are not referring to the limit $\hat{t} \to \infty$. To the contrary, we wish to have $\hat{t}$ and t of somewhat similar order so that the approximation $(1 - \frac{1}{t})^{\hat{t}} \approx e^{-\hat{t}/t}$ remains valid.

4.2. *Time-Memory Tradeoff Curve*

Our next goal is to summarize the ability of the DP tradeoff algorithm in balancing storage against online time into a single tradeoff equation.

This subsection is easier to follow if one visualizes the chains of the DP matrix as having been aligned at the ending points. The online iterations for the processing of a single DP table are counted starting from the 1st iteration. That is, checking whether $\mathbf{y} = F(\mathbf{x})$ is among the DPs in the DP table is referred to as the 1st iteration.

Our first task is to find the probability for merges to occur between DP chains.

Lemma 11. *Fix a random function $F : \mathcal{N} \to \mathcal{N}$ and suppose that we are given a precomputed DP chain of length $j \le \hat{t}$, generated with F from a random non-DP starting point. If a second chain is generated with F from a random starting point, the probability for it to become a DP chain of length i and merge with the given precomputed chain is*

$$\frac{t}{\mathsf{N}}\left\{\exp\left(\frac{\min\{i, j\}}{t}\right) - 1\right\}\exp\left(-\frac{i}{t}\right).$$

Proof. Within this proof, let us refer to the event of the second chain becoming a DP chain of length i and merging with the precomputed chain simply as *the event.*

We first restrict ourselves to the $i \le j$ case and fix the notation for the two chains as follows:

$$\begin{array}{ccccccccccccccc} \mathbf{x}_0 \to \cdots \to & \mathbf{x}_{j-i} & \to & \mathbf{x}_{j-i+1} & \to & \mathbf{x}_{j-i+2} & \to \cdots \to & \mathbf{x}_{j-1} & \to & \mathbf{x}_j \\ & \mathbf{z}_0 & \to & \mathbf{z}_1 & \to & \mathbf{z}_2 & \to \cdots \to & \mathbf{z}_{i-1} & \to & \mathbf{z}_i. \end{array}$$

The nodes $\mathbf{x}_0, \dots, \mathbf{x}_{j-1}$ are non-DPs and $\mathbf{x}_j$ is a DP.

Let us consider all possible scenarios by which the event can occur. If the randomly chosen starting point $\mathbf{z}_0$ happens to be equal to $\mathbf{x}_{j-i}$, then the second chain will follow the first chain and the event surely will occur. On the other hand, if $\mathbf{z}_0$ is either one of the points $\mathbf{x}_0, \dots, \mathbf{x}_{j-i-1}, \mathbf{x}_{j-i+1}, \dots, \mathbf{x}_{j-1}$, or a DP, then the event cannot occur. In the remaining case, i.e., when $\mathbf{z}_0$ is neither a DP nor any one of the points $\mathbf{x}_0, \dots, \mathbf{x}_{j-1}$, then the possibility of the event occurring remains. Furthermore, in this last case, we may freely set $F(\mathbf{z}_0)$ to a randomly chosen point of $\mathcal{N}$.

The above argument may now be repeated. If the randomly chosen $\mathbf{z}_1 = F(\mathbf{z}_0)$ is equal to $\mathbf{x}_{j-i+1}$, then the event occurs. If $\mathbf{z}_1$ is either a DP or one of the points $\mathbf{x}_0, \dots, \mathbf{x}_{j-i}, \mathbf{x}_{j-i+2}, \dots, \mathbf{x}_{j-1}$, then the event cannot occur. And if $\mathbf{z}_1$ is neither a DP nor one of the points $\mathbf{x}_0, \dots, \mathbf{x}_{j-1}$, then the event occurrence is yet undecided and we are free to define $\mathbf{z}_2 = F(\mathbf{z}_1)$ to a random point of $\mathcal{N}$.

Hence, when $i \le j$, the probability for the event to occur may be written as

$$\frac{1}{\mathsf{N}} + \left(1 - \frac{1}{t} - \frac{j}{\mathsf{N}}\right)\frac{1}{\mathsf{N}} + \left(1 - \frac{1}{t} - \frac{j}{\mathsf{N}}\right)^2\frac{1}{\mathsf{N}} + \cdots + \left(1 - \frac{1}{t} - \frac{j}{\mathsf{N}}\right)^i\frac{1}{\mathsf{N}},$$

which is equal to

$$\frac{1}{\mathsf{N}}\frac{1 - (1 - \frac{1}{t} - \frac{j}{\mathsf{N}})^{i+1}}{1 - (1 - \frac{1}{t} - \frac{j}{\mathsf{N}})}.$$

Noting that $\frac{j}{\mathsf{N}} \ll \frac{1}{t}$ and using $(1 - \frac{1}{t})^{i+1} \approx (1 - \frac{1}{t})^i \approx \exp(-\frac{i}{t})$, we can approximate this as

$$\frac{t}{\mathsf{N}}\left\{1 - \exp\left(-\frac{i}{t}\right)\right\}.$$

We can similarly work with the $i \ge j$ case. The event can occur only if the beginning random choices $\mathbf{z}_0, \dots, \mathbf{z}_{i-j-1}$ are made among non-DPs that are different from

$\mathbf{x}_0, \dots, \mathbf{x}_{j-1}$. The probability for the event to occur is

$$\left(1-\frac{1}{t}-\frac{j}{\mathsf{N}}\right)^{i-j}\frac{1}{\mathsf{N}}+\left(1-\frac{1}{t}-\frac{j}{\mathsf{N}}\right)^{i-j+1}\frac{1}{\mathsf{N}}+\cdots+\left(1-\frac{1}{t}-\frac{j}{\mathsf{N}}\right)^{i}\frac{1}{\mathsf{N}},$$

which is approximately

$$\frac{t}{\mathsf{N}}\left\{\exp\left(-\frac{i-j}{t}\right)-\exp\left(-\frac{i}{t}\right)\right\}.$$

The results for the cases $i \leq j$ and $i \geq j$ can be combined and stated as claimed. □

With the probability of alarms in our hands, we can compute the cost induced by false alarms.

Lemma 12. *The number of extra one-way function invocations induced by alarms is expected to be*

$$t\frac{\mathfrak{D}_{\mathrm{msc}}}{1-e^{-\hat{t}/t}}\left\{2-8e^{-\hat{t}/(2t)}+\left(5+3(\hat{t}/t)-\frac{1}{2}(\hat{t}/t)^2\right)e^{-\hat{t}/t}+e^{-2\hat{t}/t}\right\},$$

for each DP table.

Proof. When the chains are generated from m_0 non-DP starting points as given by (13), one can expect to collect

$$\frac{m}{1-e^{-\hat{t}/t}}\left(1-\frac{1}{t}\right)^{j-1}\frac{1}{t}\approx\frac{\frac{m}{t}}{1-e^{-\hat{t}/t}}\exp\left(-\frac{j}{t}\right) \tag{16}$$

DP chains of length j.

The probability of collision between the online chain and any one of these DP chains of length j, at the ith iteration of the online phase, is given by Lemma 11. Here, the 1st iteration deals with an online chain of length one, rather than zero, that starts at the unknown correct answer and ends at the inversion target.

The third component is the work required at each collision. If we take advantage of the fact that there is a chain length bound, in most cases, the number of iterations required to deal with a collision between a precomputed chain of length j and an online chain of length i will be $\min\{\hat{t}-i+1, j\}$. The only exception is when a pre-image to the inversion target is found, which is rare enough to be ignored.

Multiplying the three components and summing over all possible indices i and j, the expected number of iterations can be expressed as

$$\sum_{i=1}^{\hat{t}}\sum_{j=1}^{\hat{t}}\frac{\frac{m}{t}}{1-e^{-\hat{t}/t}}\exp\left(-\frac{j}{t}\right)\cdot\frac{t}{\mathsf{N}}\left\{\exp\left(\frac{\min\{i,j\}}{t}\right)-1\right\}\exp\left(-\frac{i}{t}\right)\cdot\min\{\hat{t}-i+1,j\}.$$

Replacing $\frac{i}{t}$ with u and $\frac{j}{t}$ with v, the above can be approximated by the integral

$$\frac{\frac{mt^2}{\mathsf{N}}t}{1-e^{-\hat{t}/t}}\int_0^{\hat{t}/t}\int_0^{\hat{t}/t}\exp(-u)\exp(-v)\big\{\exp\big(\min\{u,v\}\big)-1\big\}\min\left\{\frac{\hat{t}}{t}-u,v\right\}dv\,du,$$

when $\frac{1}{t}$ is small. The claimed value appears when this definite integral is computed. □

Finally, we write the tradeoff curve for the DP tradeoff in a way that takes into account the extra cost of alarm resolving.

Theorem 13. *The time-memory tradeoff curve for the DP tradeoff is* $TM^2 = \mathrm{D}_{\mathrm{tc}}\mathsf{N}^2$, *where the tradeoff coefficient is*

$$\begin{aligned}\mathrm{D}_{\mathrm{tc}} = &\left\{(2\mathrm{D}_{\mathrm{msc}}+1)-\frac{8\mathrm{D}_{\mathrm{msc}}}{e^{\hat{t}/2t}}+\frac{(5+\frac{3\hat{t}}{t}-\frac{\hat{t}^2}{2t^2})\mathrm{D}_{\mathrm{msc}}-2}{e^{\hat{t}/t}}+\frac{\mathrm{D}_{\mathrm{msc}}+1}{e^{2\hat{t}/t}}\right\}\\ &\times\frac{\mathrm{D}_{\mathrm{ps}}\{\ln(1-\mathrm{D}_{\mathrm{ps}})\}^2}{(1-e^{-\hat{t}/t})\mathrm{D}_{\mathrm{cr}}^3\mathrm{D}_{\mathrm{msc}}}.\end{aligned}$$

Proof. The ith DP table is processed if and only if all previous tables did not return the correct answer. The probability of such a failure is $(1-\frac{\mathrm{D}_{\mathrm{cr}}mt}{\mathsf{N}})^{i-1}$. The time required in processing a single table is the sum of one-way function invocation counts given by Lemma 3 and Lemma 12. Hence the expected total running time of the DP tradeoff may be written as

$$T=\sum_{i=1}^{\ell}\left(1-\frac{\mathrm{D}_{\mathrm{cr}}mt}{\mathsf{N}}\right)^{i-1}\left\{\left(1-e^{-\hat{t}/t}\right)+\frac{\mathrm{D}_{\mathrm{msc}}}{1-e^{-\hat{t}/t}}\left(2-\frac{8}{e^{\hat{t}/2t}}+\frac{5+\frac{3\hat{t}}{t}-\frac{\hat{t}^2}{2t^2}}{e^{\hat{t}/t}}+\frac{1}{e^{2\hat{t}/t}}\right)\right\}t.$$

The summation index i appears only in the first multiplicative factor, and we can easily check that

$$\sum_{i=1}^{\ell}\left(1-\frac{\mathrm{D}_{\mathrm{cr}}mt}{\mathsf{N}}\right)^{i-1}=\frac{\mathsf{N}}{\mathrm{D}_{\mathrm{cr}}mt}\left\{1-\left(1-\frac{\mathrm{D}_{\mathrm{cr}}mt}{\mathsf{N}}\right)^{\ell}\right\}=\frac{\mathrm{D}_{\mathrm{ps}}}{\mathrm{D}_{\mathrm{cr}}\mathrm{D}_{\mathrm{msc}}}t, \tag{17}$$

where the second equality follows from Proposition 5. The running time can now be rewritten as

$$T=\frac{\mathrm{D}_{\mathrm{ps}}}{\mathrm{D}_{\mathrm{cr}}\mathrm{D}_{\mathrm{msc}}}\left\{\left(1-e^{-\hat{t}/t}\right)+\frac{\mathrm{D}_{\mathrm{msc}}}{1-e^{-\hat{t}/t}}\left(2-\frac{8}{e^{\hat{t}/2t}}+\frac{5+\frac{3\hat{t}}{t}-\frac{\hat{t}^2}{2t^2}}{e^{\hat{t}/t}}+\frac{1}{e^{2\hat{t}/t}}\right)\right\}t^2. \tag{18}$$

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

$$\begin{aligned} TM^2 = {} & \frac{\mathtt{D}_{\mathrm{ps}}}{\mathtt{D}_{\mathrm{cr}}\mathtt{D}_{\mathrm{msc}}} \Bigg\{ \big(1 - e^{-\hat{t}/t}\big) \\ & + \frac{\mathtt{D}_{\mathrm{msc}}}{1 - e^{-\hat{t}/t}} \left(2 - \frac{8}{e^{\hat{t}/2t}} + \frac{5 + \frac{3\hat{t}}{t} - \frac{\hat{t}^2}{2t^2}}{e^{\hat{t}/t}} + \frac{1}{e^{2\hat{t}/t}} \right) \Bigg\} (mt\ell)^2 \\ = {} & \left\{ (2\mathtt{D}_{\mathrm{msc}} + 1) - \frac{8\mathtt{D}_{\mathrm{msc}}}{e^{\hat{t}/2t}} + \frac{(5 + \frac{3\hat{t}}{t} - \frac{\hat{t}^2}{2t^2})\mathtt{D}_{\mathrm{msc}} - 2}{e^{\hat{t}/t}} + \frac{\mathtt{D}_{\mathrm{msc}} + 1}{e^{2\hat{t}/t}} \right\} \\ & \times \frac{\mathtt{D}_{\mathrm{ps}}\mathtt{D}_{\mathrm{pc}}^2 \mathsf{N}^2}{(1 - e^{-\hat{t}/t})\mathtt{D}_{\mathrm{cr}}\mathtt{D}_{\mathrm{msc}}}. \end{aligned}$$

The claim is reached by observing that

$$\mathtt{D}_{\mathrm{pc}}^2 = \frac{(\mathtt{D}_{\mathrm{cr}}\mathtt{D}_{\mathrm{pc}})^2}{\mathtt{D}_{\mathrm{cr}}^2} = \frac{\{\ln(1 - \mathtt{D}_{\mathrm{ps}})\}^2}{\mathtt{D}_{\mathrm{cr}}^2},$$

where the second equality is again an application of Proposition 5. □

Let us emphasize that the tradeoff coefficient $\mathtt{D}_{\mathrm{tc}}$ is an expected value rather than a bound. The tradeoff curve was computed without restricting to the worst case, in which the algorithm fails after processing all tables. The following statement is an immediate consequence of the preceding theorem.

Corollary 14. *The time-memory tradeoff curve for the DP tradeoff is $TM^2 = \mathtt{D}_{\mathrm{tc}}\mathsf{N}^2$ with*

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

when the chain length bound $\hat{t}$ is sufficiently large.

We make the number of table lookups explicit for later use.

Lemma 15. *The online processing of the DP tradeoff that uses the parameters m, t, ℓ, and $\hat{t}$ is expected to require $t\frac{\mathtt{D}_{\mathrm{ps}}}{\mathtt{D}_{\mathrm{cr}}\mathtt{D}_{\mathrm{msc}}}$ lookups to the DP tables.*

Proof. The ith DP table is processed if and only if all previous tables have failed in returning the correct answer and the processing of each table requires a single table lookup. Hence, the expected total number of table lookups is given by (17), as claimed. □

The dependence of this result on the chain length bound $\hat{t}$ is hidden inside the $\mathtt{D}_{\mathrm{cr}}$ term.

4.3. *Efficient Use of Storage*

The storage size M appearing in any tradeoff curve refers to the total number of starting point and ending point pairs that need to be stored in the tradeoff tables. As explained in Sect. 2.7, the number of bits required to store a single starting and ending point pair will be different for each tradeoff algorithm. The focus of this section is on analyzing the ending point truncation technique explained in Sect. 2.7.4 for the DP tradeoffs.

It seems that the intention of the works [4,6] while using ending point truncation was to keep slightly more than $\log m$ bits of each ending point, so that each ending point within a DP table could be identified almost uniquely. However, this would also imply that almost every lookup to the precomputation table will generate a match of truncated points.

Let us start with a rough preliminary analysis of the situation where $\log m$ bits are stored for each ending point. The online chain creation during processing of a table requires $\Theta(t)$ iterations of the one-way function and will generate a single lookup to the table. The alarm that is almost surely generated by the lookup will require $\Theta(t)$ additional one-way function iterations to resolve. Hence, the total cost per table processing remains at $\Theta(t)$ even with ending points truncated to $\log m$ bits, and the truncation to $\log m$ bits seem reasonable. Truncation to smaller than $\log m$ bits will result in the return of multiple collisions at the single table lookup and will quickly become problematic.

Although one is guaranteed not to see a radical change in the online time complexity after truncating ending points to $\log m$ bits, the above analysis does not provide implementers with the information on how close to $\log m$ bits one may venture without experiencing visible side effects to the online time complexity. For now, implementers can only repeatedly tweak and make test runs to decide on the appropriate degree of truncation.

Consider an ending point truncation method for which two random points of $\mathcal{N}$, truncated in the specified manner, will have probability $\frac{1}{r}$ of matching with each other. We shall express such a situation as having $\frac{1}{r}$ probability of truncated match. For example, if $\log t$ bits from the ending points were truncated with $\mathrm{D}_{\mathrm{msc}} = 1$, so that $(\log m + \log t)$ bits remain, then the truncated matches would occur with probability $\frac{1}{mt}$. When truncating ending point DPs, one should truncate the random-looking part, rather than the distinguished part. Removal of the distinguished part can always be undone, and does not cause any loss of ending point information.

Lemma 16. *Assume the use of ending point truncation with the truncated match probability set to* $\frac{1}{r}$. *The number of extra one-way function invocations induced by truncation-related alarms is expected to be*

$$t\frac{1-2(\hat{t}/t)e^{-\hat{t}/t}-e^{-2\hat{t}/t}}{1-e^{-\hat{t}/t}}\frac{mt}{r},$$

for each DP table.

Proof. Consider a random function $F : \mathcal{N} \to \mathcal{N}$ and suppose that the first chain, generated with F and a random non-DP starting point, became a DP chain of length $j \leq \hat{t}$. Now, suppose a second chain is generated with F from a random non-DP starting point. Let us compute the probability for the second chain to become a DP chain of length i and not merge with the first chain, but have the same truncated ending point as the first chain.

The first i nodes of the second chain must be chosen among non-DPs that are different from the j pre-ending points of the first chain. The ith node chosen, when truncated, needs to agree with the truncated ending point of the first chain. Note that this agreement already requires the final point to be a DP. Thus the probability we aimed to write can be expressed as

$$\left(1-\frac{1}{t}-\frac{j}{\mathsf{N}}\right)^{i}\left(\frac{1}{r}-\frac{1}{\mathsf{N}}\right)\approx \exp\left(-\frac{i}{t}\right)\frac{1}{r}. \tag{19}$$

Now, we can combine the number of DP chains of length j, as given by (16), together with the probability of non-merging truncated collision with such a chain, as given by (19), to write the cost of truncation-related false alarms as

$$\sum_{i=1}^{\hat{t}}\sum_{j=1}^{\hat{t}}\frac{\frac{m}{t}}{1-e^{-\hat{t}/t}}\exp\left(-\frac{j}{t}\right)\cdot\exp\left(-\frac{i}{t}\right)\frac{1}{r}\cdot\min\{\hat{t}-i+1,j\}.$$

It now suffices to simplify this expression. Replacing $\frac{i}{t}$ with u and $\frac{j}{t}$ with v, the above can be approximated by the definite integral

$$\frac{mt^2}{1-e^{-\hat{t}/t}}\frac{1}{r}\int_0^{\hat{t}/t}\int_0^{\hat{t}/t}\exp(-u)\exp(-v)\min\left\{\frac{\hat{t}}{t}-u,v\right\}dv\,du,$$

when $\frac{1}{t}$ is small. We arrive at the claimed value when this is explicitly computed. □

Combining Lemmas 3, 12, and 16, we know that the online processing of a single DP table requires

$$\begin{aligned}&t\left(1-e^{-\hat{t}/t}\right)\\&\quad+t\frac{\mathrm{D_{msc}}}{1-e^{-\hat{t}/t}}\left\{2-8e^{-\hat{t}/(2t)}+\left(5+3(\hat{t}/t)-\frac{1}{2}(\hat{t}/t)^2\right)e^{-\hat{t}/t}+e^{-2\hat{t}/t}\right\}\\&\quad+t\frac{1}{1-e^{-\hat{t}/t}}\left\{1-2(\hat{t}/t)e^{-\hat{t}/t}-e^{-2\hat{t}/t}\right\}\frac{mt}{r}\end{aligned}$$

invocations of the one-way function. When $\hat{t}$ is sufficiently large, this simplifies to

$$t+t2\mathrm{D_{msc}}+t\frac{mt}{r},$$

with each additive term corresponding to the three terms given before. The ratio of the original number of iterations to the number of extra iterations incurred by truncations is

$$(t+t2\mathrm{D_{msc}}):t\frac{mt}{r}=r:\frac{mt}{1+2\mathrm{D_{msc}}}.$$

The choice of $r = \frac{mt}{1+2\mathrm{D}_{\mathrm{msc}}}$ will give an implementation whose added cost of truncation-related alarms increases the nontruncated original cost by 100 %. Noting that a truncated match probability of $\frac{1}{r}$ is achieved by leaving $\log r$ bits after truncation, we summarize what we have discussed in the following statement.

Proposition 17. *Fix a set of parameters for a DP tradeoff such that the chain length bound $\hat{t}$ is sufficiently large. Suppose that the online phase of the DP tradeoff implementation that stores each ending point in full requires T iterations of the one-way function to complete. Then, an implementation that leaves*

$$\log m + \log t - \log(1 + 2\mathrm{D}_{\mathrm{msc}}) \pm \varepsilon$$

bits per ending point after truncation, where ε is a small non-negative integer, requires $2^{\mp\varepsilon}T$ additional iterations of the one-way function to complete.

Let us recall the contents of Sect. 2.7 and summarize how DP table storage can be optimized. Sequential use of starting points allows each starting point to be recorded in approximately $\log m$ bits. One can truncate and leave slightly more than $\log m + \log t$ bits in each ending point and experience minimal side effect on the online running time. The decision on the exact degree of truncation can be made with the help of Proposition 17. Of the remaining approximately $\log m + \log t$ bits of the ending point, we do not need to store the $\log t$ bits that are fixed through the distinguishing property. Furthermore, the index table technique allows us to remove almost $\log m$ more bits without any loss of information. In all, $\log m$ bits are required to store each starting point, and a very small number of bits are required to store each ending point. We have thus confirmed the claims of [4,6] theoretically.

Example 18. Consider an extremely large tradeoff implementation with $\mathsf{N} = 2^{75}$ and assume the typical parameters $m \approx t \approx \ell \approx \mathsf{N}^{\frac{1}{3}} = 2^{25}$. Each starting point requires 25 bits. The DP definition allows removal of 25 bits from each ending point. We assume removal of 23 further bits through the index table method. Let us approximate $\log(1 + 2\mathrm{D}_{\mathrm{msc}}) \approx 2$. Then, each table entry will require $25 + \varepsilon$ bits.

Let T be the number of one-way function iterations required for the online chain creation and the resolving of alarms in the absence of ending point truncations. When ε is changed from 4 to 3, the storage decreases by $\frac{29-28}{29} \approx 3.45$ % while the iterations increase by 5.88 % from $(1 + \frac{1}{2^4})T$ to $(1 + \frac{1}{2^3})T$. This tradeoff is better than the tradeoff achievable through the changes in m, t, and ℓ. However, when similar calculations are made for the change of ε from 3 to 2, one can confirm that the increase in online time is not worth the decrease in storage.

In summary, for the assumed rough range of parameters, it is advisable to allocate approximately 28 bits per table entry and accept the $\frac{9}{8}T$ online time, even though this is visibly different from T.

5. Hellman Tradeoff

In this section, we gather facts about the complexity of the Hellman tradeoff. As in the previous section, the reduction functions are kept hidden during the analysis.

Our first statement is quite trivial.

Proposition 19. *The precomputation phase of the Hellman tradeoff requires $mt\ell$ one-way function invocations.*

We define the *precomputation coefficient* for the Hellman tradeoff to be $\mathrm{H}_{\mathrm{pc}} = \frac{mt\ell}{\mathsf{N}}$, so that the precomputation cost of a Hellman tradeoff is $\mathrm{H}_{\mathrm{pc}}\mathsf{N}$. The next proposition is a restatement of (4).

Proposition 20. *The success probability of the Hellman tradeoff is*

$$\mathrm{H}_{\mathrm{ps}} = 1 - e^{-\mathrm{H}_{\mathrm{cr}}\mathrm{H}_{\mathrm{pc}}}.$$

We next state the coverage rate, so that the above expression for probability of success can be put to use. This is a trivial modification of statements from [9,19].

Proposition 21. *The coverage rate of a single Hellman table is expected to be*

$$\mathrm{H}_{\mathrm{cr}} = \frac{\sqrt{2}}{\sqrt{\mathrm{H}_{\mathrm{msc}}}} \frac{e^{\sqrt{2\mathrm{H}_{\mathrm{msc}}}} - 1}{e^{\sqrt{2\mathrm{H}_{\mathrm{msc}}}} + 1}.$$

The tradeoff efficiency of the Hellman tradeoff is compactly expressed by the following time-memory tradeoff curve. This result takes into account the cost of resolving alarms, and, unlike (8), which semi-corresponds to an upper bound on the efficiency, expresses the average behavior.

Theorem 22. *The time-memory tradeoff curve for the Hellman tradeoff is $TM^2 = \mathrm{H}_{\mathrm{tc}}\mathsf{N}^2$, where the tradeoff coefficient is*

$$\mathrm{H}_{\mathrm{tc}} = \left(\frac{1}{\mathrm{H}_{\mathrm{msc}}} + \frac{1}{6}\right)\frac{1}{\mathrm{H}_{\mathrm{cr}}^3}\mathrm{H}_{\mathrm{ps}}\bigl\{\ln(1 - \mathrm{H}_{\mathrm{ps}})\bigr\}^2.$$

Proof. The ith Hellman table is processed if and only if all previous tables have failed in returning the correct answer. The probability of such a failure is $(1 - \frac{\mathrm{H}_{\mathrm{cr}}mt}{\mathsf{N}})^{i-1}$. Recalling the number of one-way function invocations required per Hellman table to resolve false alarms (7), the number of all iterations required per table can be written as $(1 + \frac{\mathrm{H}_{\mathrm{msc}}}{6})t$. The expected total running time of the Hellman tradeoff may be written as

$$T = \sum_{i=1}^{\ell}\left(1 - \frac{\mathrm{H}_{\mathrm{cr}}mt}{\mathsf{N}}\right)^{i-1}\left(1 + \frac{\mathrm{H}_{\mathrm{msc}}}{6}\right)t. \tag{20}$$

The summation index i appears only in the first multiplicative factor, and we can easily check that

$$\sum_{i=1}^{\ell}\left(1 - \frac{\mathrm{H}_{\mathrm{cr}}mt}{\mathsf{N}}\right)^{i-1} = \frac{\mathsf{N}}{\mathrm{H}_{\mathrm{cr}}mt}\left\{1 - \left(1 - \frac{\mathrm{H}_{\mathrm{cr}}mt}{\mathsf{N}}\right)^{\ell}\right\} = \frac{\mathrm{H}_{\mathrm{ps}}}{\mathrm{H}_{\mathrm{cr}}\mathrm{H}_{\mathrm{msc}}}t,$$

where the final equality follows from Proposition 20. Returning to (20), the execution time can now be written as

$$T = \left(\frac{1}{\mathrm{H_{msc}}} + \frac{1}{6}\right)\frac{\mathrm{H_{ps}}}{\mathrm{H_{cr}}}t^2. \tag{21}$$

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

$$\begin{aligned}TM^2 &= \left(\frac{1}{\mathrm{H_{msc}}} + \frac{1}{6}\right)\frac{\mathrm{H_{ps}}}{\mathrm{H_{cr}}}(mt\ell)^2 = \left(\frac{1}{\mathrm{H_{msc}}} + \frac{1}{6}\right)\frac{\mathrm{H_{ps}}}{\mathrm{H_{cr}}}\mathrm{H_{pc}^2}\mathsf{N}^2 \\ &= \left(\frac{1}{\mathrm{H_{msc}}} + \frac{1}{6}\right)\frac{\mathrm{H_{ps}}(\mathrm{H_{cr}}\mathrm{H_{pc}})^2}{\mathrm{H_{cr}^3}}\mathsf{N}^2 = \left(\frac{1}{\mathrm{H_{msc}}} + \frac{1}{6}\right)\frac{\mathrm{H_{ps}}\{\ln(1-\mathrm{H_{ps}})\}^2}{\mathrm{H_{cr}^3}}\mathsf{N}^2,\end{aligned}$$

where the final equality again relies on Proposition 20. □

The time T, stated during the above proof as (21), counts the number of one-way function computations, and includes the efforts for resolving alarms. Since the number of table lookups will be smaller, we make this count explicit.

Lemma 23. *The online processing of the Hellman tradeoff that uses the parameters m, t, and ℓ is expected to require $t^2\frac{\mathrm{H_{ps}}}{\mathrm{H_{cr}}\mathrm{H_{msc}}}$ lookups to the Hellman tables.*

The proof to this lemma is almost identical to that of Lemma 15. The only difference is that the processing of each table requires t lookups, rather than one.

After reading the proof to Theorem 22, one can easily write the expected cost of resolving alarms for the Hellman tradeoff as $\frac{\mathrm{H_{ps}}}{6\mathrm{H_{cr}}}t^2$, and by following through the relations

$$\frac{\mathrm{H_{ps}}}{6\mathrm{H_{cr}}}t^2 = \frac{1-e^{-\mathrm{H_{cr}}\mathrm{H_{pc}}}}{6\mathrm{H_{cr}}}t^2 \le \frac{1-(1-\mathrm{H_{cr}}\mathrm{H_{pc}})}{6\mathrm{H_{cr}}}t^2 = \frac{mt\ell}{6\mathsf{N}}t^2 = \frac{\mathrm{H_{msc}}}{6}t\ell,$$

we can recover the old approximation (6). This shows that the bound (6) is far from being tight, unless $\mathrm{H_{cr}}\mathrm{H_{pc}} \ll 1$.

We have so far secured access to the precomputation cost, the success probability, and the tradeoff efficiency of the Hellman tradeoff. It remains to discuss the use of storage. Three of the approaches to storage reduction that were discussed in Sect. 2.7 are applicable to the Hellman tradeoff, and we provide an analysis of the ending point truncation method below.

Let us start with a preliminary analysis. Assume that ending points are truncated so that $\log m$ bits are stored for each ending point. Then the table entries are uniquely identifiable, but each table lookup would return one truncated match on average. The cost of resolving alarms becomes $t + (t-1) + \cdots + 1 \approx \frac{t^2}{2}$ per table. This dominates the online chain creation cost of t, so truncation to $\log m$ bits is not an acceptable method.

A more exact analysis of ending point truncation is given next. We reuse the concept of truncated match probability, previously defined for the DP tradeoff, with the Hellman tradeoff.

Lemma 24. *Assume the use of ending point truncation with the truncated match probability set to $\frac{1}{r}$. The number of extra one-way function invocations induced by truncation-related alarms is expected to be*

$$t\frac{mt}{2r},$$

for each Hellman table.

Proof. Fix a random function $F : \mathcal{N} \to \mathcal{N}$ and suppose that we are given a precomputed chain of length t, generated with F from a random starting point. Now consider a second chain generated with F from a random starting point. The probability for it to produce an alarm related to truncation, i.e., a truncated ending point match without a merge with the first chain, on the ith iteration, is

$$\left(1-\frac{1}{\mathsf{N}}\right)^i\left(\frac{1}{r}-\frac{1}{\mathsf{N}}\right) \approx \left(1-\frac{i}{\mathsf{N}}\right)\left(\frac{1}{r}-\frac{1}{\mathsf{N}}\right) \approx \frac{1}{r}.$$

This is because the first i nodes of the second chain must be chosen among nodes that are different from the t pre-ending points of the first chain.

Taking account of all m precomputed chains, the cost induced by the truncation-related alarms can now be written as

$$\sum_{i=1}^{t} \frac{m}{r}(t-i+1) \approx \frac{mt^2}{r}\sum_{i=1}^{t}\left(1-\frac{i}{t}\right)\frac{1}{t}.$$

When $\frac{1}{t}$ is small, by replacing $\frac{i}{t}$ with u, the above can be approximated with the definite integral

$$\frac{mt^2}{r}\int_0^1 (1-u)\,du,$$

which computes to $\frac{mt^2}{2r}$, as claimed. □

Combining this with what we saw during the proof of Theorem 22, the total online time required to deal with a single Hellman table can be stated as

$$t + t\frac{\mathrm{H}_{\mathrm{msc}}}{6} + t\frac{mt}{2r}.$$

Arguing as we did in the previous section concerning ending point truncations for the DP tradeoffs, we can come to the following conclusion.

Proposition 25. *Fix a set of parameters for the Hellman tradeoff and suppose that its implementation which stores full ending point information requires T iterations of the one-way function to complete the online phase. Then, an implementation that leaves*

$$\log m + \log t - \log\left(2 + \frac{\mathrm{H}_{\mathrm{msc}}}{3}\right) \pm \varepsilon$$

bits per ending point after truncation, where ε is a small non-negative integer, requires $2^{\mp\varepsilon}T$ additional iterations of the one-way function to complete.

We can summarize how Hellman table storage can be optimized after recalling the contents of Sect. 2.7. Each starting point requires $\log m$ bits. Ending points may be truncated so that slightly more than $\log m + \log t$ bits remain without experiencing visible side effects on the online running time. The decision on the exact degree of truncation can be made with the help of Proposition 25. Using the index table technique, almost $\log m$ additional bits can be removed without any loss of information. In all, $\log m$ bits are required for each starting point and slightly more than $\log t$ bits are required for each ending point. This is very different from the conclusions for the DP tradeoff.

Example 26. Let us reuse the parameters of Example 18. Assuming that the index table allows removal of 23 bits and accepting the approximation $\log(2 + \frac{\mathrm{H}_{\mathrm{msc}}}{3}) \approx 1$, each table entry is seen to require $25 + 26 + \varepsilon$ bits.

With T equal to the nontruncated iterations, when ε is changed from 5 to 4, the storage decreases by $\frac{56-55}{56} \approx 1.79\ \%$, while the iterations increase by $\{(1 + \frac{1}{2^4})T - (1 + \frac{1}{2^5})T\}/\{(1 + \frac{1}{2^5})T\} \approx 3.03\ \%$. This is an acceptable tradeoff. However, the change of ε from 4 to 3 results in a 1.82 % decrease in storage, which cannot justify the corresponding 5.88 % increase in online time.

In summary, for the assumed rough range of parameters, it is advisable to allocate approximately 55 bits per table entry and accept the $\frac{17}{16}T$ online time, which is slightly higher than T.

6. Rainbow Tradeoff

In this section, we gather facts about the rainbow tradeoff. Recall that multiple rainbow tables are to be processed in parallel. The 1st iteration of a rainbow tradeoff online phase will refer to the ℓ-many searchings of $\mathbf{y}_t^{k,1} = F_{t,k}(\mathbf{x})$ among the ending points of the kth rainbow table with the index k running from 1 to ℓ. The jth iteration will require $(j-1) \cdot \ell$ invocations of the one-way function and ℓ lookups to different tables.

Our first claim is a direct consequence of the relation $mt = \mathrm{R}_{\mathrm{msc}}\mathsf{N}$ that defines the notation $\mathrm{R}_{\mathrm{msc}}$.

Proposition 27. *The precomputation phase of the rainbow tradeoff requires $\mathrm{R}_{\mathrm{pc}}\mathsf{N}$ one-way function invocations, where the precomputation coefficient is $\mathrm{R}_{\mathrm{pc}} = \mathrm{R}_{\mathrm{msc}}\ell$.*

The contents of the following lemma for the $\ell = 1$ case were already used in certain computations of [15], but let us restate it here in a more readily accessible form. The first statement of this lemma is a trivial extension of the past result (10).

Lemma 28. *The probability for the first k iterations of the online phase to fail is*

$$\prod_{i=1}^{k}\left(1 - \frac{m_{t-i}}{\mathsf{N}}\right)^{\ell},$$

where $m_0 = m$ *and* $\frac{m_{i+1}}{\mathsf{N}} = 1 - \exp(-\frac{m_i}{\mathsf{N}})$. *This product may be approximated by*

$$\left(1 - \frac{\mathrm{R}_{\mathrm{msc}}}{2 + \mathrm{R}_{\mathrm{msc}}} \frac{k+1}{t}\right)^{2\ell}.$$

Proof. The second statement is based on the approximation

$$\frac{m_i}{\mathsf{N}} \approx \frac{1}{\mathsf{N}/m + i/2},$$

which appears in [15]. This is a very small generalization of a result from [1], which treated the $m = \mathsf{N}$ case. After rewriting this as

$$1 - \frac{m_{t-i}}{\mathsf{N}} \approx \frac{2\mathsf{N} + m(t-i-2)}{2\mathsf{N} + m(t-i)},$$

the sequential cancellations within the product become visible, and we arrive at

$$\prod_{i=1}^{k}\left(1 - \frac{m_{t-i}}{\mathsf{N}}\right)^{\ell} \approx \left\{\frac{2\mathsf{N} + m(t-k-1)}{2\mathsf{N} + m(t-1)} \frac{2\mathsf{N} + m(t-k-2)}{2\mathsf{N} + m(t-2)}\right\}^{\ell} \approx \left\{1 - \frac{\mathrm{R}_{\mathrm{msc}} \frac{k+1}{t}}{2 + \mathrm{R}_{\mathrm{msc}}}\right\}^{2\ell},$$

which is the claimed approximation. □

We can arrive at the next claim by substituting $k = t$ into the above lemma and ignoring an insignificant term.

Proposition 29. *The success probability of the rainbow tradeoff is*

$$\mathrm{R}_{\mathrm{ps}} = 1 - \left(\frac{2}{2 + \mathrm{R}_{\mathrm{msc}}}\right)^{2\ell}.$$

The tradeoff efficiency of the rainbow tradeoff is compactly expressed by the following theorem. The average efficiency, rather than the worst-case situation, is expressed by this result, and the effects of false alarms have been taken into account.

Theorem 30. *The time-memory tradeoff curve for the rainbow tradeoff is* $TM^2 = \mathrm{R}_{\mathrm{tc}}\mathsf{N}^2$, *where the tradeoff coefficient is*

$$\mathrm{R}_{\mathrm{tc}} = \frac{\ell^3}{(2\ell+1)(2\ell+2)(2\ell+3)}\Bigg(\big\{(2\ell-1) + (2\ell+1)\mathrm{R}_{\mathrm{msc}}\big\}(2 + \mathrm{R}_{\mathrm{msc}})^2 - 4\big\{(2\ell-1) + \ell(2\ell+3)\mathrm{R}_{\mathrm{msc}}\big\}\left(\frac{2}{2 + \mathrm{R}_{\mathrm{msc}}}\right)^{2\ell}\Bigg).$$

Proof. Substituting $k = i - 1$ into Lemma 28, we know that the ith iteration is processed with probability $(1 - \frac{\mathrm{R}_{\mathrm{msc}}}{2+\mathrm{R}_{\mathrm{msc}}}\frac{i}{t})^{2\ell}$. The probability of alarm occurrence associated with a single chain in a single rainbow matrix at the ith iteration may be inferred from [15] to be $\frac{i+1}{\mathsf{N}}$. The reasoning behind this second statement is identical to the proof that led to the older results (6) and (7).

Hence, the expected total running time of the rainbow tradeoff, taking into account the cost of resolving alarms associated with all m rows, may be written as

$$\begin{aligned}T &= \sum_{i=1}^{t} \ell \left\{ (i-1) + (t-i+1)\frac{m(i+1)}{\mathsf{N}} \right\} \left(1 - \frac{\mathrm{R}_{\mathrm{msc}}}{2+\mathrm{R}_{\mathrm{msc}}}\frac{i}{t}\right)^{2\ell} \\ &\approx t^2 \ell \sum_{i=1}^{t} \left\{ \frac{i}{t} + \left(1-\frac{i}{t}\right)\mathrm{R}_{\mathrm{msc}}\frac{i}{t} \right\} \left(1 - \frac{\mathrm{R}_{\mathrm{msc}}}{2+\mathrm{R}_{\mathrm{msc}}}\frac{i}{t}\right)^{2\ell} \frac{1}{t}.\end{aligned}$$

This may be approximated by the definite integral

$$T = t^2 \ell \int_0^1 u\left\{1 + \mathrm{R}_{\mathrm{msc}}(1-u)\right\} \left(1 - \frac{\mathrm{R}_{\mathrm{msc}}}{2+\mathrm{R}_{\mathrm{msc}}}u\right)^{2\ell} du,$$

which computes to

$$\begin{aligned}T &= t^2 \ell \frac{\{(2\ell-1) + (2\ell+1)\mathrm{R}_{\mathrm{msc}}\}(2+\mathrm{R}_{\mathrm{msc}})^2}{(2\ell+1)(2\ell+2)(2\ell+3)\mathrm{R}_{\mathrm{msc}}^2} \\ &\quad - \frac{4\{(2\ell-1) + \ell(2\ell+3)\mathrm{R}_{\mathrm{msc}}\}(\frac{2}{2+\mathrm{R}_{\mathrm{msc}}})^{2\ell})}{(2\ell+1)(2\ell+2)(2\ell+3)\mathrm{R}_{\mathrm{msc}}^2}.\end{aligned} \tag{22}$$

It now suffices to combine this with the storage size $M = m\ell$ and simplify to arrive at the claim. □

The time T appearing in the above tradeoff curve gives the count of one-way function invocations and ignores table lookups.

Lemma 31. *The online processing of the rainbow tradeoff is expected to require*

$$t\ell \frac{2 + \mathrm{R}_{\mathrm{msc}} - 2(\frac{2}{2+\mathrm{R}_{\mathrm{msc}}})^{2\ell}}{(2\ell+1)\mathrm{R}_{\mathrm{msc}}}$$

lookups to the rainbow tables.

Proof. At the start of the proof of Theorem 30, we saw that the ith iteration is processed with probability $(1 - \frac{\mathrm{R}_{\mathrm{msc}}}{2+\mathrm{R}_{\mathrm{msc}}}\frac{i}{t})^{2\ell}$. Since each iteration requires ℓ table lookups, it suffices to compute

$$\sum_{i=1}^{t} \ell \left(1 - \frac{\mathrm{R}_{\mathrm{msc}}}{2+\mathrm{R}_{\mathrm{msc}}}\frac{i}{t}\right)^{2\ell} \approx t\ell \int_0^1 \left(1 - \frac{\mathrm{R}_{\mathrm{msc}}}{2+\mathrm{R}_{\mathrm{msc}}}u\right)^{2\ell} du,$$

to arrive at the expected number of table lookups. □

We now turn to the issue of efficient storage use. The number of online iterations, which is of $\Theta(t^2\ell)$ order, is much larger than the number of table lookups, given by the above lemma as being of $\Theta(t\ell)$ order. This indicates that truncation to slightly more than $\log m$ bits, which allows unique identification of table entries, should be reasonable. A more accurate analysis is given below. We reuse the concept of truncated match probability, defined for the DP tradeoffs, also in the rainbow tradeoff case.

Lemma 32. *Assume the use of ending point truncation with the truncated match probability set to $\frac{1}{r}$. The number of additional one-way function invocations induced by alarms related to ending point truncations is expected to be*

$$t^2\ell\frac{m}{r}\frac{(-2+(2\ell+1)\mathrm{R}_{\mathrm{msc}})(2+\mathrm{R}_{\mathrm{msc}})+4(\frac{2}{2+\mathrm{R}_{\mathrm{msc}}})^{2\ell}}{(2\ell+1)(2\ell+2)\mathrm{R}_{\mathrm{msc}}^2}.$$

Proof. For exactly the same reason given in the proof of Lemma 24, the probability for a randomly generated second chain to produce a truncation induced alarm without merging with the first chain is

$$\left(1-\frac{1}{\mathsf{N}}\right)^i\left(\frac{1}{r}-\frac{1}{\mathsf{N}}\right)\approx\left(1-\frac{i}{\mathsf{N}}\right)\left(\frac{1}{r}-\frac{1}{\mathsf{N}}\right)\approx\frac{1}{r}.$$

After recalling Lemma 28, the probability for the ith iteration to be processed, and taking into account all the $m\ell$ precomputed chains, the expected online cost can be written as

$$\sum_{i=1}^{t}(t-i+1)\frac{m\ell}{r}\left(1-\frac{\mathrm{R}_{\mathrm{msc}}}{2+\mathrm{R}_{\mathrm{msc}}}\frac{i}{t}\right)^{2\ell}.$$

Replacing $\frac{i}{t}$ with u, the above can be approximated by the definite integral

$$\frac{mt^2\ell}{r}\int_0^1(1-u)\left(1-\frac{\mathrm{R}_{\mathrm{msc}}}{2+\mathrm{R}_{\mathrm{msc}}}u\right)^{2\ell}du,$$

when $\frac{1}{t}$ is small, and the claimed value appears when this is computed. □

After reviewing the arguments concerning ending point truncation made for the DP and Hellman tradeoffs, we can combine (22) and Lemma 32 to write the effects of ending point truncation in terms of the number of bits remaining.

Proposition 33. *Fix a set of parameters for the rainbow tradeoff and suppose that its implementation which stores full ending point information is expected to require T iterations of the one-way function for the online phase. Then, an implementation that leaves*

$$\log m+\log\left(\frac{(2\ell+3)\{\frac{-2+(2\ell+1)\mathrm{R}_{\mathrm{msc}}}{2+\mathrm{R}_{\mathrm{msc}}}+(\frac{2}{2+\mathrm{R}_{\mathrm{msc}}})^{2\ell+2}\}}{\{(2\ell-1)+(2\ell+1)\mathrm{R}_{\mathrm{msc}}\}-\{(2\ell-1)+\ell(2\ell+3)\mathrm{R}_{\mathrm{msc}}\}(\frac{2}{2+\mathrm{R}_{\mathrm{msc}}})^{2\ell+2}}\right)$$

$$\pm\varepsilon$$

bits per ending point after truncation, where ε is a small non-negative integer, requires $2^{\mp\varepsilon}T$ additional iterations of the one-way function to complete.

Referencing Sect. 2.7, let us summarize the number of bits required to store each starting point and ending point pair. Each starting point requires $\log m$ bits. Ending points may be truncated so that slightly more than $\log m$ bits remain without visible side effects on the online running time. The index table method allows most of the remaining $\log m$ bits to be removed from the ending point without any loss of information. In all, $\log m$ bits are required for each starting point and only a very small number of bits are required for each ending point. We have thus confirmed the claims of [4,6].

Example 34. The parameters for a rainbow tradeoff that roughly correspond to those used in Examples 18 and 26 are $m = 2^{50}$, $t = 2^{25}$, and $\ell = 1$. Assume that the index table allows removal of 48 bits. The middle term appearing in the equation of Proposition 33 for the parameters being used is $\log \frac{2^{15}}{2^{28}} \approx 0$. Each table entry will require $50 + 2 + \varepsilon$ bits.

Let T be the number of iterations expected of a nontruncated implementation. When ε is changed from 6 to 5, the storage decreases by $\frac{58-57}{58} \approx 1.72$ %, while the iterations increase by $\{(1 + \frac{1}{2^5})T - (1 + \frac{1}{2^6})T\}/\{(1 + \frac{1}{2^6})T\} \approx 1.54$ %. This is an acceptable tradeoff. However, the change of ε from 5 to 4 results in a 1.75 % decrease in storage, which cannot justify the corresponding 3.03 % increase in online time.

In summary, for the assumed rough range of parameters, it is advisable to allocate approximately 55 bits per table entry and accept the $\frac{33}{32}T$ online time, which is only slightly higher than T.

7. Optimal Tradeoff Parameters

In this section, we find the optimal set of parameters for the three tradeoff algorithms. The notion of optimality in this section ignores the cost of precomputation.

Let us present our initial arguments in terms of the Hellman tradeoff. The balance between time and memory achievable by the Hellman tradeoff is expressed by the tradeoff curve $TM^2 = \mathrm{H}_{\mathrm{tc}}\mathsf{N}^2$. It is clear that the Hellman algorithm at parameters m, t, and ℓ that bring about a smaller tradeoff coefficient H_{tc} will require less resources to run. In other words, tradeoff coefficient H_{tc} is a measure of the tradeoff efficiency, with a smaller value representing a more desirable balancing of storage and online time.

The tradeoff coefficient H_{tc} is fully determined by the parameters m, t, and ℓ. It should first be noticed that a better tradeoff coefficient should always be achievable, if one decides to sacrifice the success probability of finding the correct answer. Hence, any comparison between two Hellman tradeoff coefficients, achievable through two different sets of parameters, should be done under the condition that they produce the same success probability.

Arguments similar to the above may be made for the DP and rainbow tradeoffs. Hence, for each of the three algorithms, we will work to find the smallest tradeoff coefficient achievable under a fixed requirement on the success rate.

The smallest possible tradeoff coefficient value for a tradeoff algorithm is referred to as the *tradeoff characteristic* in [1], where it is used to compare the perfect version of the rainbow table method against other algorithms. However, we wish for the optimal tradeoff coefficients given in this work to be understood separately for each algorithm. Using it to argue the superiority of one algorithm over another may seem plausible, but is of limited value in practice. Parameters achieving better tradeoff efficiency may require more precomputation, and with large-scale implementations of the tradeoff technique, lowering the precomputation cost may be significantly more valuable than achieving better tradeoff efficiency. Our purpose of locating the optimal tradeoff parameters is so that they may be used in the next section to bound the range of parameters, when making fair comparisons between different algorithms.

7.1. *DP Tradeoff*

The parameter set that achieves the optimal DP tradeoff efficiency, under a fixed requirement on the probability of success, is given below.

Proposition 35. *Let* $0 < \mathtt{D}_{\mathrm{ps}} < 1$ *be any fixed value. The DP tradeoff, under any set of parameters* m, t, ℓ, *and* $\hat{t}$, *that are subject to the relations*

$$mt^2 = 1.26453\mathsf{N}, \qquad \ell = 1.28007\bigl\{-\ln(1-\mathtt{D}_{\mathrm{ps}})\bigr\}t, \quad \textit{and} \quad \hat{t} = 2.59169t,$$

attains the given value $\mathtt{D}_{\mathrm{ps}}$ *as its probability of success, and exhibits a tradeoff performance corresponding to*

$$\mathtt{D}_{\mathrm{tc}} = 5.49370\mathtt{D}_{\mathrm{ps}}\bigl\{\ln(1-\mathtt{D}_{\mathrm{ps}})\bigr\}^2,$$

as the four parameters are varied. Under any such choice of parameters, the number of one-way function invocations required for the precomputation phase is

$$\mathtt{D}_{\mathrm{pc}}\mathsf{N} = 1.61869\bigl\{-\ln(1-\mathtt{D}_{\mathrm{ps}})\bigr\}\mathsf{N}.$$

The three relations restricting the parameter choices give optimal parameters in the sense that no choice of m, t, ℓ, *and* $\hat{t}$ *can lead to a tradeoff coefficient smaller than the above while achieving* $\mathtt{D}_{\mathrm{ps}}$ *as its probability of success.*

Proof. The relation of Proposition 5 may equivalently be stated as

$$\ell = \frac{\mathsf{N}}{\mathtt{D}_{\mathrm{cr}}mt}\bigl\{-\ln(1-\mathtt{D}_{\mathrm{ps}})\bigr\} = \frac{1}{\mathtt{D}_{\mathrm{cr}}\mathtt{D}_{\mathrm{msc}}}\bigl\{-\ln(1-\mathtt{D}_{\mathrm{ps}})\bigr\}t. \tag{23}$$

Now, referencing Proposition 9, we know that the DP coverage rate $\mathtt{D}_{\mathrm{cr}} = \mathtt{D}_{\mathrm{cr}}[\mathtt{D}_{\mathrm{msc}}, \hat{t}/t]$ may be treated as a function of the two variables $\mathtt{D}_{\mathrm{msc}}$ and $\frac{\hat{t}}{t}$. Hence, given any m, t, $\hat{t}$, and $\mathtt{D}_{\mathrm{ps}}$, if we set $\mathtt{D}_{\mathrm{msc}} = \frac{mt^2}{\mathsf{N}}$ and $\mathtt{D}_{\mathrm{cr}} = \mathtt{D}_{\mathrm{cr}}[\mathtt{D}_{\mathrm{msc}}, \hat{t}/t]$, and also fix ℓ through the relation (23), then the DP tradeoff with these parameters will always achieve the success probability of $\mathtt{D}_{\mathrm{ps}}$. We remark that ℓ must be set to an integer, but since the right-hand side of (23) is rather large, the error to the success probability, introduced by taking the nearest integer to the right-hand side value, will be very small.

Keeping in mind that we may freely choose m, t, and $\hat{t}$ and still obtain any requested success probability, we now work to minimize the DP tradeoff coefficient $\mathtt{D}_{\mathrm{tc}}$, as given

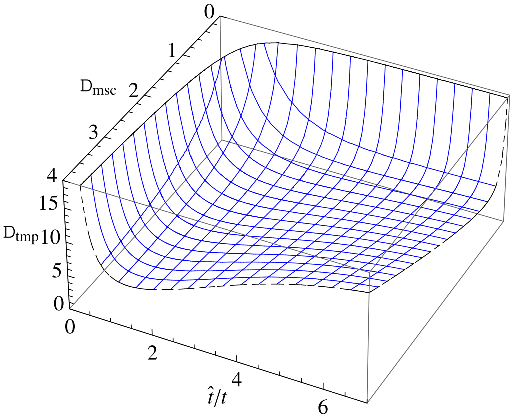


Fig. 2. Tradeoff coefficient for DP tradeoff at fixed probability of success ($\mathrm{D}_{\mathrm{tmp}} = \frac{\mathrm{D}_{\mathrm{tc}}}{\mathrm{D}_{\mathrm{ps}} \cdot \{\ln(1-\mathrm{D}_{\mathrm{ps}})\}^2}$).

by Theorem 13. We drop from the expression for D_{tc} any part that depends only on D_{ps} and consider

$$\mathrm{D}_{\mathrm{tmp}}\left[\mathrm{D}_{\mathrm{msc}}, \frac{\hat{t}}{t}\right] = \frac{(2\mathrm{D}_{\mathrm{msc}} + 1) - \frac{8\mathrm{D}_{\mathrm{msc}}}{e^{\hat{t}/2t}} + \frac{(5 + \frac{3\hat{t}}{t} - \frac{\hat{t}^2}{2t^2})\mathrm{D}_{\mathrm{msc}} - 2}{e^{\hat{t}/t}} + \frac{\mathrm{D}_{\mathrm{msc}}+1}{e^{2\hat{t}/t}}}{(1 - e^{-\hat{t}/t})\mathrm{D}_{\mathrm{cr}}[\mathrm{D}_{\mathrm{msc}}, \frac{\hat{t}}{t}]^3 \mathrm{D}_{\mathrm{msc}}}, \tag{24}$$

which is a function of the two variables $\mathrm{D}_{\mathrm{msc}}$ and $\frac{\hat{t}}{t}$. It is clear that, when the probability of success requirement is fixed, minimizing D_{tc} is equivalent to minimizing $\mathrm{D}_{\mathrm{tmp}}[\mathrm{D}_{\mathrm{msc}}, \hat{t}/t]$. Note that, even though $\mathrm{D}_{\mathrm{msc}} = \frac{mt^2}{\mathsf{N}}$ and $\hat{t}/t$ share the parameter t, since we are free to set m, t, and $\hat{t}$ to any value, there are enough degrees of freedom, and we may treat $\mathrm{D}_{\mathrm{msc}}$ and $\hat{t}/t$ as independent variables when looking for the minimum of $\mathrm{D}_{\mathrm{tmp}}[\mathrm{D}_{\mathrm{msc}}, \hat{t}/t]$.

After $\mathrm{D}_{\mathrm{cr}}[\mathrm{D}_{\mathrm{msc}}, \hat{t}/t]$, as given by Proposition 9, is substituted into the right-hand side of (24), we can use numerical methods to find its minimum. One discovers that the minimum value of $\mathrm{D}_{\mathrm{tmp}} = 5.49370$ is obtained at $\mathrm{D}_{\mathrm{msc}} = 1.25453$ and $\hat{t}/t = 2.59169$. The claimed relation between ℓ and t follows from (23). The final claim concerning the precomputation cost is obtained by combining Proposition 4 with the first two relations stated by the claim. □

The parameter set that achieves the minimum tradeoff coefficient for the DP tradeoff is visible through Fig. 2. It plots $\mathrm{D}_{\mathrm{tmp}} = \frac{\mathrm{D}_{\mathrm{tc}}}{\mathrm{D}_{\mathrm{ps}}\{\ln(1-\mathrm{D}_{\mathrm{ps}})\}^2}$, which is given by (24), as a function of the variables $\mathrm{D}_{\mathrm{msc}}$ and $\hat{t}/t$.

The tradeoff curve reflected by this proposition allows us to say more about the tradeoff than the previously known rough curve (8). Suppose that, for some fixed set of parameters, the success rate of the DP tradeoff is not too small, and suppose that one wishes to increase the success rate, to the extent that the failure rate becomes the square of its current value. Then, for an optimal choice of parameters, the D_{ps} factor will change little and the $\{\ln(1-\mathrm{D_{ps}})\}^2$ factor will increase by a factor of four. Hence, one must allow an increase in the online time by a factor of four or use twice the current storage. The proposition also shows that one must endure twice the precomputation cost to achieve this aim. Of course, the simplest way of doing this would be to double the number of tables, while keeping all other parameters the same.

While the above result gives the parameters that achieve the optimal tradeoff efficiency, in practical applications, precomputation is very costly and one is more likely to choose a sufficiently large $\hat{t}$, so as not to discard any of the precomputed results.

Proposition 36. *Let $0 < \mathrm{D_{ps}} < 1$ be any fixed value. When the use of a sufficiently large $\hat{t}$ is assumed, the DP tradeoff, under any set of parameters m, t, and ℓ, that are subject to the relations*

$$mt^2 = 0.562047\mathsf{N} \quad \textit{and} \quad \ell = 2.18614\bigl\{-\ln(1-\mathrm{D_{ps}})\bigr\}t,$$

attains the given value $\mathrm{D_{ps}}$ as its probability of success, and exhibits a tradeoff performance corresponding to

$$\mathrm{D_{tc}} = 7.01057\mathrm{D_{ps}}\bigl\{\ln(1-\mathrm{D_{ps}})\bigr\}^2,$$

as the three parameters are varied. Under any such choice of parameters, the number of one-way function invocations required for the precomputation phase is

$$\mathrm{D_{pc}}\mathsf{N} = 1.22871\bigl\{-\ln(1-\mathrm{D_{ps}})\bigr\}\mathsf{N}.$$

The two relations restricting the parameter choices give optimal parameters in the sense that, when $\hat{t}$ is sufficiently large, no choice of m, t, and ℓ can lead to a tradeoff coefficient smaller than the above while achieving $\mathrm{D_{ps}}$ as its probability of success.

Proof. The proof is almost identical to that of Proposition 35. The only difference is that we rely on Proposition 10 to view $\mathrm{D_{cr}}$ as a function of $\mathrm{D_{msc}}$ and obtain the tradeoff coefficient from Corollary 14, so that

$$\mathrm{D_{tc}} = \left(2 + \frac{1}{\mathrm{D_{msc}}}\right)\left(\frac{\sqrt{1+2\mathrm{D_{msc}}}+1}{2}\right)^3 \mathrm{D_{ps}}\bigl\{\ln(1-\mathrm{D_{ps}})\bigr\}^2. \tag{25}$$

It suffices to minimize

$$\mathrm{D_{tmp}}[\mathrm{D_{msc}}] = \frac{\mathrm{D_{tc}}}{\mathrm{D_{ps}}\{\ln(1-\mathrm{D_{ps}})\}^2} = \left(2 + \frac{1}{\mathrm{D_{msc}}}\right)\left(\frac{\sqrt{1+2\mathrm{D_{msc}}}+1}{2}\right)^3,$$

which is a function of the single variable $\mathrm{D_{msc}}$. □

In comparison to the previous optimal set of parameters that utilizes $\hat{t}$ as a free variable, this version shows a less efficient tradeoff, but requires less precomputation. The behavior of the DP tradeoff coefficient with sufficiently large $\hat{t}$, under a fixed requirement for success rate, is given as the left-hand graph of Fig. 3. The point of minimum

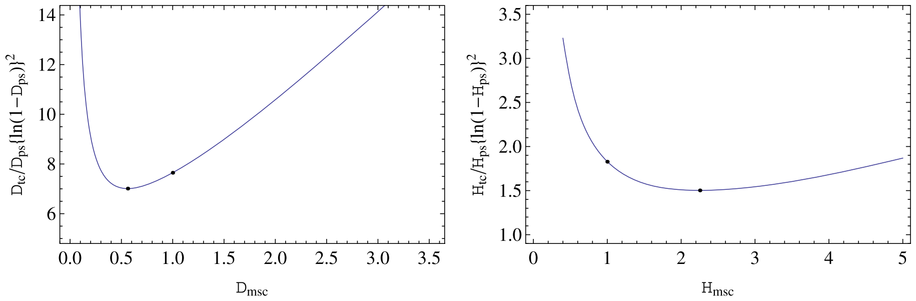


Fig. 3. Tradeoff coefficients at fixed probability of success for the DP tradeoff with a sufficiently large $\hat{t}$ and the Hellman tradeoff.

tradeoff coefficient is marked, together with the position corresponding to the more commonly used matrix stopping rule of $\mathrm{D_{msc}} = 1$. The advantage of using a smaller matrix stopping constant than usual is clearly visible.

7.2. *Hellman Tradeoff*

We now turn to the Hellman tradeoff. This is very similar to the DP tradeoff case that uses a sufficiently large $\hat{t}$.

Proposition 37. *Let $0 < \mathrm{H_{ps}} < 1$ be any fixed value. The Hellman tradeoff, under any set of parameters m, t, and ℓ, that are subject to the relations*

$$mt^2 = 2.25433\mathsf{N} \quad \textit{and} \quad \ell = 0.598941\big\{-\ln(1-\mathrm{H_{ps}})\big\}t,$$

attains the given $\mathrm{H_{ps}}$ as its probability of success, and exhibits a tradeoff performance corresponding to

$$\mathrm{H_{tc}} = 1.50217\mathrm{H_{ps}}\big\{\ln(1-\mathrm{H_{ps}})\big\}^2,$$

as the three parameters are varied. Under any such choice of parameters, the number of one-way function invocations required for the precomputation phase is

$$\mathrm{H_{pc}}\mathsf{N} = 1.35021\big\{-\ln(1-\mathrm{H_{ps}})\big\}\mathsf{N}.$$

The two relations restricting the parameter choices give optimal parameters in the sense that no choice of m, t, and ℓ can lead to a tradeoff coefficient smaller than the above while achieving $\mathrm{H_{ps}}$ as its probability of success.

Proof. The proof given here shall be concise, since it is similar to those of Propositions 35 and 36. Based on Proposition 20, we may fix $\ell = \frac{1}{\mathrm{H_{cr}}\mathrm{H_{msc}}}\{-\ln(1-\mathrm{H_{ps}})\}t$. Reference to Proposition 21 shows that the Hellman coverage rate $\mathrm{H_{cr}} = \mathrm{H_{cr}}[\mathrm{H_{msc}}]$ may be seen as a function of $\mathrm{H_{msc}} = \frac{mt^2}{\mathsf{N}}$. Hence, given any m, t, and $\mathrm{H_{ps}}$, we can set ℓ to an appropriate value with which the Hellman tradeoff achieves a success probability of $\mathrm{H_{ps}}$.

We now work to minimize the Hellman tradeoff coefficient. By combining Theorem 22 and Proposition 21, we obtain

$$\mathrm{H_{tc}} = \left(\frac{1}{\mathrm{H_{msc}}} + \frac{1}{6}\right)\left(\frac{\sqrt{\mathrm{H_{msc}}}}{\sqrt{2}}\,\frac{e^{\sqrt{2\mathrm{H_{msc}}}}+1}{e^{\sqrt{2\mathrm{H_{msc}}}}-1}\right)^{3} \mathrm{H_{ps}}\bigl\{\ln(1-\mathrm{H_{ps}})\bigr\}^{2}. \tag{26}$$

For a fixed success probability, it suffices to minimize the part that depends only on the single variable $\mathrm{H_{msc}}$.

One can use numeric methods to identify the minimum value $\frac{\mathrm{H_{tc}}}{\mathrm{H_{ps}}\{\ln(1-\mathrm{H_{ps}})\}^2} = 1.50217$, which is attained at $\mathrm{H_{msc}} = 2.25433$. The two remaining constants appearing in the proposition may now be obtained through appropriate evaluations. □

The most typical Hellman tradeoff, which is set to use $mt^2 = \mathsf{N}$ and $\ell = t$, attains a success probability of 57.68 % and the tradeoff curve $TM^2 = 0.7797\mathsf{N}^2$, when the cost of resolving alarms is taken into account. In comparison, the choice of $mt^2 = 2.2543\mathsf{N}$ and $\ell = 0.5160t$, suggested by Proposition 37, gives $TM^2 = 0.6409\mathsf{N}^2$, while achieving the same success rate. This improvement in tradeoff efficiency is visible through the right-hand graph of Fig. 3, where the two dots mark the two parameter choices we have just discussed.

The price paid for this better tradeoff efficiency is the increase in precomputation from N to $1.1630\mathsf{N}$. Indeed, after combining Propositions 20 and 21 into

$$\mathrm{H_{pc}} = \frac{\sqrt{\mathrm{H_{msc}}}}{\sqrt{2}}\,\frac{e^{\sqrt{2\mathrm{H_{msc}}}}+1}{e^{\sqrt{2\mathrm{H_{msc}}}}-1}\bigl\{-\ln(1-\mathrm{H_{ps}})\bigr\}, \tag{27}$$

one can check that the precomputation $\mathrm{H_{pc}}[\mathrm{H_{msc}}]$ required under any fixed probability of success is an increasing function of $\mathrm{H_{msc}}$. Hence, while any point that is situated to the left of the minimal point in Fig. 3 may not be optimal in view of tradeoff efficiency, it corresponds to less precomputation. Depending on the available computational resources, one may choose to lower the precomputation cost rather than increase the tradeoff efficiency. On the other hand, increasing $\mathrm{H_{msc}}$ beyond the minimizing value 2.25433 will have bad effects on both the precomputation and the tradeoff efficiency and should be avoided.

Let us briefly return to the DP tradeoff that uses a sufficiently large $\hat{t}$. By combining Propositions 5 and 10, we can write

$$\mathrm{D_{pc}} = \frac{\sqrt{1+2\mathrm{D_{msc}}}+1}{2}\bigl\{-\ln(1-\mathrm{D_{ps}})\bigr\}, \tag{28}$$

and, as with the Hellman tradeoff, confirm that $\mathrm{D_{pc}}$ is an increasing function of $\mathrm{D_{msc}}$. Since we know from Proposition 36 that the best performance is achieved at $\mathrm{D_{msc}} = 0.562047$, the choice of $\mathrm{D_{msc}} \leq 0.562047$ may be reasonable in view of the lower precomputation cost, but using $\mathrm{D_{msc}} > 0.562047$ should be avoided. In particular, the use of $\mathrm{D_{msc}} = 1$ cannot be justified.

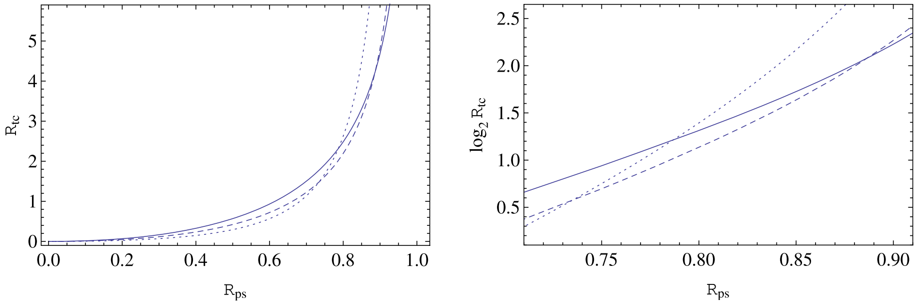


Fig. 4. Tradeoff coefficient of rainbow tradeoff as a function of success rate requirement at small number of tables ($\ell = 1$: *dotted*, $\ell = 2$: *dashed*, $\ell = 3$: *solid*).

7.3. *Rainbow Tradeoff*

The analyses of optimal parameters for the DP and Hellman tradeoffs were very similar. However, the rainbow tradeoff does not allow the same approach, because we have less control over the parameter ℓ. The number of tables ℓ used with the DP and Hellman tradeoffs are quite large, and we had treated ℓ as if it were a continuous variable. In the rainbow tradeoff case, the table count is usually a small integer, and we must keep in mind that it takes only discrete values.

Let us start with a fixed number of tables ℓ. For any given requirement on the success rate, we can rewrite Proposition 29 as

$$\mathrm{R_{msc}} = 2\big\{(1 - \mathrm{R_{ps}})^{-1/2\ell} - 1\big\} \tag{29}$$

and understand this as a lower bound on $\mathrm{R_{msc}}$ that can be used with ℓ to achieve $\mathrm{R_{ps}}$. It is clear that increasing $\mathrm{R_{msc}}$ under a fixed ℓ will increase the precomputation cost $\mathrm{R_{msc}}\ell\mathsf{N}$. One can also work with the tradeoff coefficient $\mathrm{R_{tc}}$, as provided by Theorem 30, to confirm that increasing $\mathrm{R_{msc}}$ under a fixed ℓ will reduce the tradeoff efficiency. Hence, under any fixed ℓ, the exact value of $\mathrm{R_{msc}}$, suggested by (29), should be used to achieve the required success rate.

We can now treat $\mathrm{R_{msc}}$ as a function of the success rate requirement $\mathrm{R_{ps}}$, for any fixed ℓ. After substituting $\mathrm{R_{msc}}$, as given by (29), into the tradeoff coefficient of Theorem 30, one can rewrite it as

$$\begin{aligned}\mathrm{R_{tc}} = {} & \frac{4\ell^3}{(2\ell+1)(2\ell+2)(2\ell+3)} \\ & \times \Big(\big\{-(2\ell+3) + 2(2\ell+1)(1-\mathrm{R_{ps}})^{-1/2\ell}\big\}(1-\mathrm{R_{ps}})^{-1/\ell} \\ & + \big\{(2\ell+1)^2 - 2\ell(2\ell+3)(1-\mathrm{R_{ps}})^{-1/2\ell}\big\}(1-\mathrm{R_{ps}})\Big).\end{aligned} \tag{30}$$

For each fixed ℓ, this is a function of the single variable $\mathrm{R_{ps}}$. A plot of this is given as Fig. 4 for table counts $\ell = 1$, 2, and 3. The right-hand box is a magnified partial view of the left-hand box in logarithmic scale.

Recalling that a smaller tradeoff coefficient implies better tradeoff efficiency, one can clearly read from the figure that the use of $\ell = 1$ is optimal when the requirement

Table 1. Range of success probability requirements for which each table count ℓ is optimal.

ℓ	R_{ps}	$\log_2(1-\mathrm{R}_{\mathrm{ps}})$	$\log_2 \mathrm{R}_{\mathrm{tc}}$	$\mathrm{R}_{\mathrm{msc}}[\mathrm{R}_{\mathrm{ps}}, \ell\uparrow]$	$\mathrm{R}_{\mathrm{msc}}[\mathrm{R}_{\mathrm{ps}}, \ell\downarrow]$
	0	0	$-\infty$		0
1	0.734166	−1.91140	0.565848	1.87905	0.785335
2	0.886651	−3.14116	2.08082	1.44688	0.874929
3	0.946562	−4.22600	2.88968	1.25878	0.884357
4	0.973305	−5.22729	3.41666	1.14577	0.873341
5	0.986146	−6.17353	3.79818	1.06812	0.856920
6	0.992618	−7.08171	4.09387	1.01079	0.839893
7	0.995992	−7.96295	4.33425	0.966542	0.823891
8	0.997795	−8.82486	4.53663	0.931326	0.809415
9	0.998775	−9.67274	4.71157	0.902658	0.796529
10	0.999314	−10.5104	4.86585	0.878902	0.785129
11	0.999614	−11.3404	5.00406	0.858929	0.775059
12	0.999782	−12.1649	5.12941	0.841927	0.766150
13	0.999877	−12.9850	5.24421	0.827299	0.758246
14	0.999930	−13.8020	5.35019	0.814594	0.751208
15	0.999960	−14.6163	5.44869	0.803466	0.744914

for success rate is very low and that the use of successively higher numbers of tables becomes optimal as the success rate requirement is made more stringent. We have numerically solved for the explicit probabilities at which the transition to the next table count should be made and have recorded this in Table 1.

Let us briefly explain the content of the table with examples. Suppose one aims to achieve a success probability of 99.9 % with the rainbow tradeoff. Since 0.999 sits between 0.998775 and 0.999314, it is optimal to use ten tables. If one is requested to set the probability of failure to $\frac{1}{2^7}$, we locate -7 between -6.17353 and -7.08171 and conclude that six tables would be optimal. To understand the other three columns of the table, let us focus on the row that sits between $\ell = 1$ and $\ell = 2$. The use of a single table with $\mathrm{R}_{\mathrm{msc}} = 1.87905$, or the use of two tables at $\mathrm{R}_{\mathrm{msc}} = 0.785335$ will both result in an optimal tradeoff coefficient of $\mathrm{R}_{\mathrm{tc}} = 1.48026 = 2^{0.565848}$ and a success rate of 73.4166 %.

Note that any given success rate requirement R_{ps} makes a certain number of tables ℓ as optimal, and the ℓ value fixes $\mathrm{R}_{\mathrm{msc}}$ through (29). Since the tradeoff coefficient of Theorem 30 is already determined by ℓ and $\mathrm{R}_{\mathrm{msc}}$, and since the relation (29) guarantees the R_{ps} success rate, any parameter set satisfying the mentioned restriction will be optimal in view of the tradeoff coefficient. Let us gather what we have discussed in a proposition.

Proposition 38. *Let $0 < \mathrm{R}_{\mathrm{ps}} < 1$ be any given fixed value. Locate the table count ℓ from Table 1 that corresponds to the given R_{ps} and compute*

$$\mathrm{R}_{\mathrm{msc}} = 2\left\{(1-\mathrm{R}_{\mathrm{ps}})^{-1/(2\ell)} - 1\right\}.$$

Then the rainbow tradeoff that uses the located ℓ and any parameters m and t satisfying the relation

$$mt = \mathrm{R}_{\mathrm{msc}}\mathsf{N}$$

attains the given value R_{ps} as its probability of success. The tradeoff performance corresponding to

$$\mathrm{R}_{\mathrm{tc}} = \frac{\ell^3}{(2\ell+1)(2\ell+2)(2\ell+3)}\Big(\big\{(2\ell-1)+(2\ell+1)\mathrm{R}_{\mathrm{msc}}\big\}(2+\mathrm{R}_{\mathrm{msc}})^2 \\ -4\big\{(2\ell-1)+\ell(2\ell+3)\mathrm{R}_{\mathrm{msc}}\big\}(1-\mathrm{R}_{\mathrm{ps}})\Big)$$

can be observed as m and t are varied under the restriction. With any such choice of parameters, the number of one-way function invocations required for the precomputation phase is

$$\mathrm{R}_{\mathrm{pc}}\mathsf{N} = \mathrm{R}_{\mathrm{msc}}\ell\mathsf{N}.$$

The choice of ℓ through Table 1 and the single relation concerning m and t lead to optimal parameters in the sense that no choice of m, t, and ℓ can result in a tradeoff coefficient smaller than the above while achieving R_{ps} as its probability of success.

To be strictly logical, one must also consider the possibility that allowing the multiple tables to be of different sizes may lead to better tradeoff coefficients. The case of three tables with the most general table sizes is analyzed in [21], and the conclusion is made that optimal tradeoff performance is achieved at equal sized tables. The method used can probably be extended to larger numbers of tables, but the required computations will be much more complicated than the computations done in this work. Since the examination of the three-table case showed that we are not likely to gain anything from the more general analysis, we chose to work with equal sized tables. However, for the case of perfect rainbow tables, we have reasons to believe that this extra flexibility will bring about better tradeoff performance.

Finally, we want to provide an argument that is analogous to what was discussed at the end of Sect. 7.2. One can check that

$$\mathrm{R}_{\mathrm{pc}} = \mathrm{R}_{\mathrm{msc}}\ell = 2\ell\big\{(1-\mathrm{R}_{\mathrm{ps}})^{-1/(2\ell)}-1\big\} \tag{31}$$

is a decreasing function of ℓ, for each fixed R_{ps}. Hence, use of an ℓ count that is larger than what is suggested by Table 1 will decrease the precomputation requirement at the cost of reduced tradeoff efficiency. This may be preferable in some situations. On the other hand, use of an ℓ count that is smaller than the optimal count will have bad effects on both the precomputation cost and tradeoff efficiency, and should be avoided.

8. Comparison of Tradeoff Performances

All the ingredients required for a fair comparison of performances between the tradeoff algorithms are now ready. Any discussion of the DP tradeoff in this section assumes that the chain length bound $\hat{t}$ is sufficiently large.

8.1. *Conversion of the Tradeoff Coefficients to a Common Unit*

It is clear that for any comparison of tradeoff algorithms to be fair, the algorithms must be made to present the same probability of success. One must also consider the precomputation cost required by each algorithm, and this aspect will be considered later on in this section. For now, we focus on the fact that the tradeoff coefficient is a measure of tradeoff efficiency. Let us assume that the DP, Hellman, and rainbow tradeoff algorithms display the respective tradeoff curves

$$T_{\mathrm{D}} M_{\mathrm{D}}^2 = \mathrm{D}_{\mathrm{tc}} \mathsf{N}^2, \qquad T_{\mathrm{H}} M_{\mathrm{H}}^2 = \mathrm{H}_{\mathrm{tc}} \mathsf{N}^2, \quad \text{and} \quad T_{\mathrm{R}} M_{\mathrm{R}}^2 = \mathrm{R}_{\mathrm{tc}} \mathsf{N}^2, \tag{32}$$

at the same success rate. We will discuss how to interpret the ratio $\mathrm{D}_{\mathrm{tc}} : \mathrm{H}_{\mathrm{tc}} : \mathrm{R}_{\mathrm{tc}}$ of the tradeoff coefficients as a ratio of tradeoff efficiencies.

8.1.1. *Unit for Storage*

Let us first consider the storage variable M. For the moment, we will disregard any issues concerning the time unit.

In all three tradeoff algorithms, M represents the number of starting point and ending point pairs that need to be stored, but the actual number of bits required to store each table entry will be different among the tradeoff algorithms. We saw through Propositions 17, 25, and 33 that the number of bits required to store each table entry is as follows for each tradeoff algorithm.

DP:	slightly more than $\log m_{\mathrm{D}}$ bits,
Hellman:	slightly more than $\log m_{\mathrm{H}} + \log t_{\mathrm{H}}$ bits,
rainbow:	slightly more than $\log m_{\mathrm{R}}$ bits.

Let us assume from this point on that the ending point truncations for the three algorithms were done in such a way that their effects on the online time are minimal. In particular, we assume that the contents of Corollary 14, Theorems 22 and 30 remain valid after ending point truncation. We further assume that the *slightly more* bits mentioned above can be ignored.

A fair comparison of tradeoff performances would express storages for the three algorithms in terms of number of bits that are required for the precomputation tables rather than the number of starting point and ending point pairs. Under the two assumptions made, one is led to focus on the ratio

$$(\log m_{\mathrm{D}})^2 \mathrm{D}_{\mathrm{tc}} : (\log m_{\mathrm{H}} + \log t_{\mathrm{H}})^2 \mathrm{H}_{\mathrm{tc}} : (\log m_{\mathrm{R}})^2 \mathrm{R}_{\mathrm{tc}}, \tag{33}$$

rather than the raw tradeoff coefficient ratio $\mathrm{D}_{\mathrm{tc}} : \mathrm{H}_{\mathrm{tc}} : \mathrm{R}_{\mathrm{tc}}$. The bit sizes per entry are multiplied in squares because any change in storage affects the tradeoff efficiency through a square factor.

The implementation environment and tradeoff requirements will place the choice of suitable parameters into a certain range, and it is reasonable to assume that the parameters that would be chosen for each algorithm would be related through

$$\log t_{\mathrm{D}} \approx \log t_{\mathrm{H}} \approx \log t_{\mathrm{R}}, \qquad \log m_{\mathrm{D}} \approx \log m_{\mathrm{H}}, \quad \text{and} \quad \log m_{\mathrm{R}} \approx \log m_{\mathrm{H}} + \log t_{\mathrm{H}}. \tag{34}$$

Some readers may object that our discussion on the number of bits required for each table entry makes $m_{\mathtt{D}} = 2m_{\mathtt{H}}$ more reasonable than $m_{\mathtt{D}} = m_{\mathtt{H}}$, but this difference by a factor of two is lost in the approximations when they are converted bit sizes, as is done in the expression (34).

Assuming the rough correspondence (34) between parameters, the ratio (33) simplifies to

$$\left(\frac{\log m_{\mathtt{D}}}{\log m_{\mathtt{R}}}\right)^2 \mathtt{D}_{\mathrm{tc}} : \mathtt{H}_{\mathrm{tc}} : \mathtt{R}_{\mathrm{tc}}. \tag{35}$$

When issues concerning time units are ignored, this is the correct ratio to focus on when comparing the tradeoff efficiencies of different algorithms.

8.1.2. *Unit for Online Time*

Unification of the time unit T is now considered. Issues concerning the storage unit, which we have already discussed, are ignored for the moment.

Recall that the time variable T used in the tradeoff curves counts the number of one-way function iterations and ignores the table lookups. Hence, parameter sets which lead to identical times $T_{\mathtt{D}} = T_{\mathtt{H}} = T_{\mathtt{R}}$ do not guarantee that the simultaneous executions of the three algorithms will finish at the same time. For a fair interpretation of a tradeoff coefficient ratio as a ratio of tradeoff efficiency, the difference in the time units used by the algorithms must be taken into account.

It is reasonable to expect the time taken for a single one-way function iteration by the three algorithms to be quite similar. Let us fix the notation and express this common time length as |Itr|. We also fix the notation |TL-D|, |TL-H|, and |TL-R| for the time required for lookups to the DP, Hellman, and rainbow tables, respectively. Depending on the implementation platform, it is possible to experience $|\text{TL-}\mathtt{D}| \approx |\text{TL-}\mathtt{H}| \ll |\text{TL-}\mathtt{R}|$, even when equal sized storages are allocated to the three algorithms, since the DP or Hellman tradeoffs utilize a large number of small tables, whereas the rainbow tradeoff uses a small number of large tables.

Referencing Lemma 15, the real-world time required to process the online phase of a DP tradeoff can be written as $T_{\mathtt{D}}|\text{Itr}| + t_{\mathtt{D}} \frac{\mathtt{D}_{\mathrm{ps}}}{\mathtt{D}_{\mathrm{cr}}\mathtt{D}_{\mathrm{msc}}} |\text{TL-}\mathtt{D}|$. Since we know from (18) that $T_{\mathtt{D}} = t_{\mathtt{D}}^2 \frac{\mathtt{D}_{\mathrm{ps}}}{\mathtt{D}_{\mathrm{cr}}\mathtt{D}_{\mathrm{msc}}}(1 + 2\mathtt{D}_{\mathrm{msc}})$, the real-world online time for DP tradeoff can be expressed as

$$\left(1 + \frac{1}{1 + 2\mathtt{D}_{\mathrm{msc}}} \frac{|\text{TL-}\mathtt{D}|}{t_{\mathtt{D}}|\text{Itr}|}\right) T_{\mathtt{D}}|\text{Itr}|. \tag{36}$$

Similarly, gathering information from (21) and Lemma 23, the real-world execution time for the Hellman online phase can be written as

$$\left(1 + \frac{6}{6 + \mathtt{H}_{\mathrm{msc}}} \frac{|\text{TL-}\mathtt{H}|}{|\text{Itr}|}\right) T_{\mathtt{H}}|\text{Itr}|. \tag{37}$$

The corresponding expression for the rainbow tradeoff, relying on (22) and Lemma 31, is given by

$$\left(1 + \mathtt{R}_{\mathrm{tmp}}[\ell, \mathtt{R}_{\mathrm{ps}}] \frac{|\text{TL-}\mathtt{R}|}{t_{\mathtt{R}}|\text{Itr}|}\right) T_{\mathtt{R}}|\text{Itr}|, \tag{38}$$

where

$$\begin{aligned}\mathtt{R}_{\mathrm{tmp}}[\ell, \mathtt{R}_{\mathrm{ps}}] = \Big(&(2\ell+2)(2\ell+3)\big((1-\mathtt{R}_{\mathrm{ps}})^{-1/\ell} - (1-\mathtt{R}_{\mathrm{ps}})^{-1/(2\ell)} \\ &- (1-\mathtt{R}_{\mathrm{ps}})^{1-1/(2\ell)} + (1-\mathtt{R}_{\mathrm{ps}})\big)\Big) \\ &\times \big(2(2\ell+1)(1-\mathtt{R}_{\mathrm{ps}})^{-3/(2\ell)} - (2\ell+3)(1-\mathtt{R}_{\mathrm{ps}})^{-1/\ell} \\ &- 2\ell(2\ell+3)(1-\mathtt{R}_{\mathrm{ps}})^{1-1/(2\ell)} + (2\ell+1)^2(1-\mathtt{R}_{\mathrm{ps}})\big)^{-1}\end{aligned}$$

is of $\Theta(1)$ order. We have used (29) to remove all occurrences of $\mathtt{R}_{\mathrm{msc}}$ in the expression, because our graphs for each fixed $\mathtt{R}_{\mathrm{ps}}$ in the later part of this section are drawn using ℓ as a parameter.

The three equations (36), (37), and (38) can be used to easily find the correct way to compare tradeoff coefficients. For example, consider the simplest case where all table lookups are negligible, i.e., when $|\mathrm{TL}\text{-}\mathtt{D}|, |\mathrm{TL}\text{-}\mathtt{H}|, |\mathrm{TL}\text{-}\mathtt{R}|, \ll |\mathrm{Itr}|$. Then, all the second terms in the three equations are negligible. Hence, the raw coefficient ratio $\mathtt{D}_{\mathrm{tc}} : \mathtt{H}_{\mathrm{tc}} : \mathtt{R}_{\mathrm{tc}}$ reflects the true tradeoff efficiency ratio of the three algorithms.

Let us next consider the case where $|\mathrm{Itr}| \ll |\mathrm{TL}\text{-}\mathtt{D}| \approx |\mathrm{TL}\text{-}\mathtt{H}| \leq |\mathrm{TL}\text{-}\mathtt{R}| \ll t_{\mathtt{D}}|\mathrm{Itr}| \approx t_{\mathtt{R}}|\mathrm{Itr}|$. This might be the situation experienced by a large implementation that requires disk accesses for table lookups. The probable use of large $t_{\mathtt{D}}$ and $t_{\mathtt{R}}$ partially justifies the third inequality. In this case, the second term of (37) dominates all other five terms of the three equations. The Hellman tradeoff clearly cannot compete with the other two algorithms, and the comparison between the DP and rainbow tradeoffs can fairly be done with $\mathtt{D}_{\mathrm{tc}} : \mathtt{R}_{\mathrm{tc}}$.

The final example we consider is when $|\mathrm{Itr}| \approx |\mathrm{TL}\text{-}\mathtt{D}| \approx |\mathrm{TL}\text{-}\mathtt{H}| \leq |\mathrm{TL}\text{-}\mathtt{R}| \ll t_{\mathtt{D}}|\mathrm{Itr}| \approx t_{\mathtt{R}}|\mathrm{Itr}|$. Then neither of the two terms of (37) dominates the other and neither can be ignored. The appropriate ratio to study when comparing tradeoff algorithms would be

$$\mathtt{D}_{\mathrm{tc}} : \left(1 + \frac{6}{6+\mathtt{H}_{\mathrm{msc}}} \frac{|\mathrm{TL}\text{-}\mathtt{H}|}{|\mathrm{Itr}|}\right)\mathtt{H}_{\mathrm{tc}} : \mathtt{R}_{\mathrm{tc}}. \tag{39}$$

There are many other cases to consider, but the correct way to adjust the tradeoff coefficients so that they reflect the tradeoff efficiency ratio of the tradeoff algorithms can easily be found from (36), (37), and (38).

This ends our discussion on the unit of time, but let us briefly digress and discuss the exceptional situation of $|\mathrm{TL}\text{-}\mathtt{R}| \gg t_{\mathtt{R}}|\mathrm{Itr}|$ for the rainbow tradeoff. This could happen when the precomputation tables must be reached over the Internet during the online phase. Then, table lookups dominate the online phase, and we can combine $T_{\mathtt{R}} = \Theta(t\ell)$, $M_{\mathtt{R}} = \Theta(m\ell)$, and $\ell = \Theta(1)$ to conclude that $T_{\mathtt{R}}M_{\mathtt{R}} \propto \mathsf{N}$. At first thought, this might seem to be a much better tradeoff curve than the usual $TM^2 \propto \mathsf{N}^2$ curve.

The counterintuitive conclusion hides the fact that the unit of time $T_{\mathtt{R}}$ is now $|\mathrm{TL}\text{-}\mathtt{R}|$, rather than $|\mathrm{Itr}|$. Furthermore, unless N is small, the assumption $|\mathrm{TL}\text{-}\mathtt{R}| \gg t_{\mathtt{R}}|\mathrm{Itr}|$ cannot continue to hold as $t_{\mathtt{R}}$ is increased, so the tradeoff curve will eventually return to the usual $TM^2 \propto \mathsf{N}^2$ after a certain point. The tradeoff curve $T_{\mathtt{R}}M_{\mathtt{R}} \propto \mathsf{N}$ remains valid when $t_{\mathtt{R}}$ is moved in the decreasing direction, but having $T_{\mathtt{R}}M_{\mathtt{R}}$ constant is worse than having $T_{\mathtt{R}}M_{\mathtt{R}}^2$ constant in that direction.

Similar arguments may be made for the DP tradeoff, but lookups to DP tables over a slow network are even less likely to be seen than with the rainbow tradeoffs. Since each individual DP table is rather small, each could be stored on the node that computes the online chain corresponding to that table.

8.1.3. *Combined Unit Conversion*

The storage unit conversion and the time unit conversion are orthogonal, and the two conversions may simply be multiplied to give modified tradeoff coefficients that are appropriate for comparisons of different tradeoff algorithms. For example, under the reasonable assumption (34), we know that the storage conversion must follow (35). If the one-way function computation and table lookup speeds satisfy $|\mathrm{Itr}| \approx |\text{TL-}\mathtt{D}| \approx |\text{TL-}\mathtt{H}| \leq |\text{TL-}\mathtt{R}| \ll t_{\mathrm{D}}|\mathrm{Itr}| \approx t_{\mathrm{R}}|\mathrm{Itr}|$, the time unit conversion must follow (39). Combing the two, we know that comparisons of tradeoff algorithms must focus on

$$\left(\frac{\log m_{\mathrm{D}}}{\log m_{\mathrm{R}}}\right)^2 \mathrm{D_{tc}} : \left(1 + \frac{6}{6+\mathrm{H_{msc}}}\frac{|\text{TL-}\mathtt{H}|}{|\mathrm{Itr}|}\right)\mathrm{H_{tc}} : \mathrm{R_{tc}},$$

under the stated circumstances.

In our further discussions below, we will mainly restrict ourselves to parameter sets that roughly satisfy

$$\log m_{\mathrm{D}} \approx \log m_{\mathrm{H}} \approx \log t_{\mathrm{D}} \approx \log t_{\mathrm{H}} \approx \log t_{\mathrm{R}} \approx \frac{1}{3}\log \mathsf{N} \quad \text{and} \quad \log m_{\mathrm{R}} \approx \frac{2}{3}\log \mathsf{N}$$

and mostly assume that the time required for a single table lookup is negligible in comparison to that required for a single one-way function computation. Under these assumptions, the ratio that needs to be studied when comparing tradeoff efficiencies is

$$\frac{1}{4}\mathrm{D_{tc}} : \mathrm{H_{tc}} : \mathrm{R_{tc}}. \tag{40}$$

We shall refer to the situation that has just been described as the *typical situation*, as it often appears during theoretic developments of the tradeoff technique. However, we do not claim this to be typical in practical applications of the tradeoff technique.

We emphasize that our further discussions given below concerning tradeoff performance comparisons will only be valid under the typical situation assumption. If the environment and tradeoff performance requirements make parameter choices such that $\log m_{\mathrm{D}} \not\approx \log t_{\mathrm{D}}$ is more appropriate, or if the table lookup delays cannot be ignored, the algorithm comparison conclusions will be different. Still, one will be able to use the information explained in this subsection to easily make the proper adjustments.

Even for the typical situation, the ratio (40) can be made more accurate for each explicit situation. Based on Examples 18, 26, and 34, we can state that

$$28^2\frac{9}{8}\mathrm{D_{tc}} : 55^2\frac{17}{16}\mathrm{H_{tc}} : 55^2\frac{33}{32}\mathrm{R_{tc}} = 1.00\mathrm{D_{tc}} : 3.64\mathrm{H_{tc}} : 3.54\mathrm{R_{tc}},$$

is a more accurate version of (40), for the typical situation with $\mathsf{N} = 2^{75}$. This new ratio does not ignore the extra one-way function invocations caused by ending point truncations and does not ignore the *slightly more* bits discussed at the start of Sect. 8.1.1.

8.2. *DP Tradeoff Versus Hellman Tradeoff*

As discussed in the previous subsection, it suffices to compare $\frac{1}{4}\mathrm{D_{tc}}$ against $\mathrm{H_{tc}}$ for a fair comparison between the DP and Hellman tradeoffs. We are assuming the typical situation explained at the end of the previous subsection; any conclusions we make could be different under different circumstances. The precomputation effort is finally considered during tradeoff comparison in this section.

Propositions 36 and 37 show that the optimal tradeoff efficiencies of the two algorithms are given by

$$\frac{1}{4}\mathrm{D}_{\mathrm{tc}} = 1.75264\mathrm{D}_{\mathrm{ps}}\bigl\{\ln(1-\mathrm{D}_{\mathrm{ps}})\bigr\}^2 \quad \text{and} \quad \mathrm{H}_{\mathrm{tc}} = 1.50217\mathrm{H}_{\mathrm{ps}}\bigl\{\ln(1-\mathrm{H}_{\mathrm{ps}})\bigr\}^2.$$

One may want to conclude that the Hellman tradeoff, with the smaller tradeoff coefficient, is more efficient, but this is acceptable only when the precomputation cost can be totally ignored. In practice, precomputation cost is the largest barrier to any large-scale deployment of tradeoff algorithms and is hard to ignore.

The precomputation costs required to achieve the above tradeoff efficiencies are

$$\mathrm{D}_{\mathrm{pc}} = 1.22871\bigl\{-\ln(1-\mathrm{D}_{\mathrm{ps}})\bigr\} \quad \text{and} \quad \mathrm{H}_{\mathrm{pc}} = 1.35021\bigl\{-\ln(1-\mathrm{H}_{\mathrm{ps}})\bigr\}.$$

The precomputation cost of the DP tradeoff is lower and we are faced with the problem of comparing high efficiency at high cost against low efficiency at low cost.

After a moment of thought, one must admit that such a comparison cannot be done in an objective manner. The comparison must reflect how valuable tradeoff efficiency is to the user and how willing one is to invest more time and resources into the precomputation phase. There is no unit with which to express either of these unquantifiable values. Furthermore, one must also question whether it is reasonable to compare the two tradeoffs at parameters giving their respective optimal tradeoff efficiencies. Non-optimal parameters may be preferable under many situations in view of lower precomputation cost.

We can conclude that all we can do is present the range of choices that can be made with each algorithm and allow the users to make their conclusions based on their explicit circumstances. The crucial information that must be displayed to allow easy judgement of which tradeoff is more suitable is the relation between tradeoff efficiency and precomputation cost. This must be done at each fixed requirement for the inversion success rate.

As was previously noted through (28) and (27), when under a fixed probability of success requirement, both D_{pc} and H_{pc} are functions of their respective $\mathrm{D}_{\mathrm{msc}}$ and $\mathrm{H}_{\mathrm{msc}}$ values. The tradeoff coefficients D_{tc} and H_{tc}, under a fixed success rate requirement, were similarly expressed as functions of the corresponding $\mathrm{D}_{\mathrm{msc}}$ and $\mathrm{H}_{\mathrm{msc}}$ values in (25) and (26).

For a comparison of the DP tradeoff against the Hellman tradeoff, it now suffices to present the graphs

$$\left\{\left(\mathrm{D}_{\mathrm{pc}}[\mathrm{D}_{\mathrm{msc}}], \frac{1}{4}\mathrm{D}_{\mathrm{tc}}[\mathrm{D}_{\mathrm{msc}}]\right) \,\middle|\, \mathrm{D}_{\mathrm{msc}} \le 0.562047\right\} \tag{41}$$

and

$$\bigl\{\bigl(\mathrm{H}_{\mathrm{pc}}[\mathrm{H}_{\mathrm{msc}}], \mathrm{H}_{\mathrm{tc}}[\mathrm{H}_{\mathrm{msc}}]\bigr) \bigm| \mathrm{H}_{\mathrm{msc}} \le 2.25433\bigr\}, \tag{42}$$

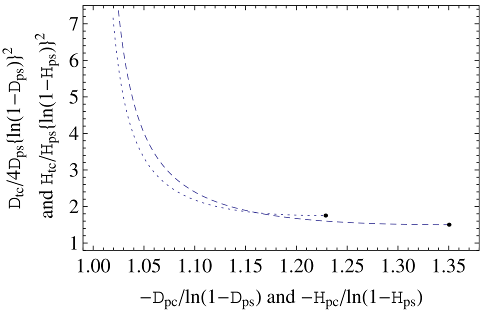


Fig. 5. The tradeoff coefficients $\frac{1}{4}D_{tc}$ (*dotted*) and H_{tc} (*dashed*) in relation to their respective precomputation cost.

where the bounds on D_{msc} and H_{msc} were placed in accordance with the discussion at the end of Sect. 7.2. These graphs are given in Fig. 5. Since the two graphs are to be compared at identical success rate requirements $D_{ps} = H_{ps}$, we have removed the common parts that depend on the success probability from both of the cases before plotting the graphs. Hence, the graphs do not depend on the success rate and are valid for all success rate requirements. Both graphs extend further upward, but the right ends, corresponding to the optimal tradeoff performances, are clearly marked with dots.

The two graphs are very close to each other. Even though slightly better tradeoff efficiency can be obtained with the Hellman tradeoff at higher precomputation cost, in practice, unless parameters far from the typical $m \approx t \approx \mathsf{N}^{1/3}$ region are to be used, the DP tradeoff will be favored in view of fewer table lookups. For example, if the table lookup time makes $\frac{1}{5}D_{tc} : H_{tc}$ a more appropriate measure of tradeoff performance ratio than the current $\frac{1}{4}D_{tc} : H_{tc}$, the dotted curve for the DP tradeoff would move down and present itself as a more advantageous algorithm.

If table lookup time is absolutely negligible in comparison to the one-way function computation time, there is a short range of parameter sets with which the Hellman tradeoff can slightly outperform the DP tradeoff using the same amount of precomputation. If table lookup time is negligible and precomputation is not to be considered, the Hellman tradeoff can be slightly better.

8.3. *Rainbow Tradeoff Versus DP and Hellman Tradeoffs*

We now include the rainbow tradeoff into the comparison graphs. As was discussed in Sect. 8.1, we assume the typical situation concerning the approximate range of parameters and table lookup time, and consider comparisons between $\frac{1}{4}D_{tc}$, H_{tc}, and R_{tc} to be fair.

In addition to the graphs (41) and (42), we need to plot all possible (R_{pc}, R_{tc}) points. We can first check through (31) that R_{pc} can be seen as a function of the table count ℓ, when the success rate requirement R_{ps} is fixed. As for the tradeoff coefficient, equation (30) presents it as a function of just ℓ, when R_{ps} is fixed. Given any requirement on the success rate R_{ps}, it is now possible to draw the graph

$$\big\{ \big(R_{pc}[\ell], R_{tc}[\ell]\big) \;\big|\; \ell \geq \text{optimal table count for } R_{ps} \big\}, \tag{43}$$

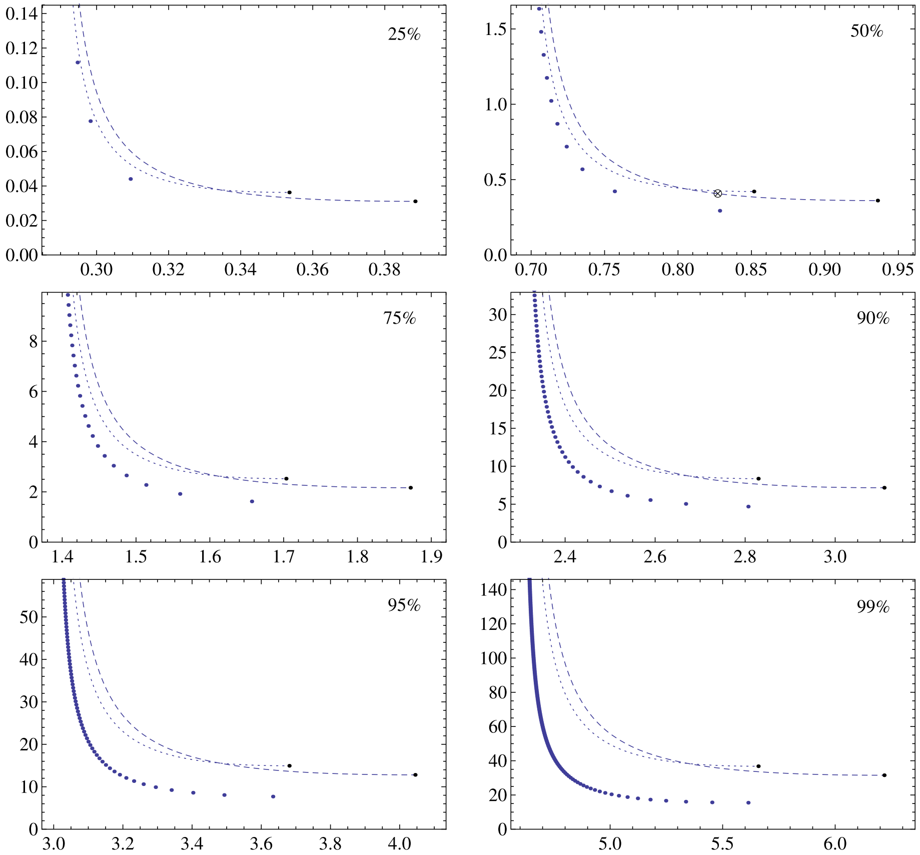


Fig. 6. Tradeoff coefficients $\frac{1}{4}\mathtt{D}_{\mathrm{tc}}$ (*dotted*), $\mathtt{H}_{\mathrm{tc}}$ (*dashed*), and $\mathtt{R}_{\mathrm{tc}}$ (*large dots*) in relation to their respective precomputation cost at success rates 25 %, 50 %, 75 %, 90 %, 95 %, and 99 % (X-axis: $\mathtt{D}_{\mathrm{pc}}$, $\mathtt{H}_{\mathrm{pc}}$, and $\mathtt{R}_{\mathrm{pc}}$; Y-axis: $\frac{1}{4}\mathtt{D}_{\mathrm{tc}}$, $\mathtt{H}_{\mathrm{tc}}$, and $\mathtt{R}_{\mathrm{tc}}$).

where the optimal table count can be obtained from Table 1. Note that this is no longer a continuous graph, but a discrete set of points. In the strict sense, the previous graphs for the DP and Hellman tradeoffs were also discrete sets of points, but unless N is very small, the points are extremely close to each other.

Unlike our comparison between DP and Hellman tradeoffs, the parts that depend on $\mathtt{R}_{\mathrm{ps}}$ appearing in the expressions (31) and (30) are not identical to those appearing in the corresponding expressions (28), (27), (25), and (26). Hence, separate graphs need to be drawn for each success rate. This is given in Fig. 6 for some success rates.

In all of the graphs, one can see that the curve for the rainbow tradeoff sits closer to the origin than the curves for the DP and Hellman tradeoffs. Note that a graph sitting lower shows better tradeoff efficiency and being positioned more to the left implies lower precomputation cost. In all the cases except for the ones corresponding to 25 % and 50 % success rates, given any position on the curve for either the DP or Hellman tradeoff there is a rainbow tradeoff position that presents better tradeoff performance at a smaller precomputation cost. Use of the rainbow tradeoff is definitely advisable in these cases.

The existence of better rainbow position is also mostly true in the 50 % case. The exception is marked with an $\otimes$ on the curve for the Hellman tradeoff. This position is very slightly to the left of the optimal rainbow position and hence corresponds to less precomputation than the optimal rainbow position. At the same time, it is positioned lower than the second best rainbow position and hence shows better tradeoff efficiency than this second best position. Hence, there can be no rainbow tradeoff parameter set that can replace the Hellman position marked with an $\otimes$ without at least very slightly sacrificing either the precomputation cost or the tradeoff efficiency. Still, anybody will agree that this exception is quite unreasonable, and one would normally choose to sacrifice the extremely small amount of either the precomputation cost or the tradeoff performance for a visibly better value of the other factor.

The 25 % case also displays the rainbow tradeoff requiring less precomputation than the other two tradeoffs in achieving equal tradeoff efficiency, but the awkward exceptional position discussed for the 50 % case can be found here as rather large segments. In addition, the best tradeoff efficiency achievable by the rainbow tradeoff falls short of what is reachable by the other two algorithms. Hence there will be situations where the DP or Hellman tradeoff is preferable over the rainbow tradeoff, when required to achieve a 25 % success rate.

The relative advantage of using the rainbow tradeoff is clearly seen to grow with the increase in the success rate requirement. For the 99 % success rate case, it seems almost safe to say that the rainbow tradeoff performs approximately two times better than the other two tradeoff algorithms in any of their reasonable usages.

In conclusion, the use of the rainbow tradeoff is advisable for high success rate requirements, and there may occasionally be low success rate applications with special situations where the other two tradeoffs are preferable. We emphasize once more that this conclusion is only valid under the typical situation assumption explained in Sect. 8.1. For example, if we must work with parameters such that $2\log m_{\mathrm{D}} \approx \log t_{\mathrm{D}}$ and $2\log m_{\mathrm{H}} \approx \log t_{\mathrm{H}}$ and table lookups are negligible, then comparison of the coefficients $\frac{1}{9}\mathrm{D}_{\mathrm{tc}}$, H_{tc}, and R_{tc} would be appropriate. This would bring the curve for the DP tradeoff lower and we would arrive at a different conclusion.

8.4. *Revisit to the Preliminary Tradeoff Comparison*

In Sect. 2.9, we recalled how [23] claimed the rainbow tradeoff to be more efficient than the DP tradeoff by a factor of two. We also explained how [3,4] pointed out that the two algorithms require different numbers of bits to represent each table entry and argued that the DP tradeoff was twice as efficient as the rainbow tradeoff. Since our conclusions of Sect. 8.3 are once again supportive of the rainbow tradeoff, let us explain where in the arguments of [3,4] the inaccuracies were introduced. Details of the current paper, including the proofs, need to be understood if the computations of this section are to be followed.

According to Propositions 5, 10, and Corollary 14, the DP tradeoff performance at parameters $m = t = \ell = \mathsf{N}^{1/3}$ and a sufficiently large chain length bound is given by

$$\mathrm{D}_{\mathrm{ps}} = 51.9\ \%, \qquad \mathrm{D}_{\mathrm{tc}} = 2.13, \qquad \mathrm{D}_{\mathrm{pc}} = 1. \tag{44}$$

In comparison, Proposition 29 and Theorem 30 allow us to state that the rainbow tradeoff at the naturally corresponding parameters $m = \mathsf{N}^{2/3}$, $t = \mathsf{N}^{1/3}$, and $\ell = 1$ shows the performance

$$\mathtt{R}_{\text{ps}} = 55.6\ \%, \qquad \mathtt{R}_{\text{tc}} = 0.422, \qquad \mathtt{R}_{\text{pc}} = 1. \tag{45}$$

As claimed in [3,4] and confirmed in Sect. 8, we must apply an adjustment factor to compensate for the difference in bits required per table entry before comparing these two sets of figures. Comparing $\frac{1}{4}\mathtt{D}_{\text{tc}} = 0.532$ against $\mathtt{R}_{\text{tc}} = 0.422$, we can conclude that, for the same amount of physical storage, the rainbow tradeoff is both faster and succeeds more often than the DP tradeoff. This disagrees with the claim of [3,4] and does not go against our conclusion, which stated that the rainbow tradeoff is slightly better than the DP tradeoff at low success rates.

The main argument of [3,4] that the number of bits required to store each entry of the rainbow tradeoff is twice that required for the DP tradeoff was certainly correct. The primary source of their incorrect conclusion is the inaccurate estimations of running time complexities for the two algorithms. The tradeoff coefficient for the DP was estimated at 1, but in reality, it was a much larger $\mathtt{D}_{\text{tc}} = 2.13$.

After understanding the details of the proof to Theorem 13, one can compute that, out of the value 2.13, the part that corresponds to the online chain computation is only 0.709. This is smaller than 1, the estimate of [3,4], but the remaining 1.42, which is due to the resolving of alarms, was much larger. In the case of the rainbow tradeoff, the tradeoff coefficient was estimated at 0.5 by [3,4], and the actual value $\mathtt{R}_{\text{tc}} = 0.422$ was smaller. Details of the proof to Theorem 30 show that, out of the 0.422, the cost of online chain creation corresponds to a mere 0.306 and the cost of resolving alarms corresponds to an even smaller 0.117.

The true online chain creation efforts for the two algorithms being smaller than the initial rough estimates is a consequence of the algorithms terminating prematurely with the discovery of the correct answer, and the upper bounds for the cost of online chain creation given by the preliminary analysis [3,4] were correct. Since $\frac{1}{4} \times 0.709$ is less than 0.306, a comparison of the two algorithms based only on the online chain creation time would have concluded that the DP tradeoff was superior. In fact, the ratio $\frac{0.709/4}{0.306} \approx 0.579$ is somewhat in agreement with the performance ratio of two that was claimed by [3,4], based on their rough upper bounds. However, when the costs of resolving alarms were taken into account, the conclusions were quite the opposite. This is a clear indication that a careful analysis of the cost associated with resolving of alarms was necessary for a fair comparison of tradeoff algorithms.

Let us now discuss how sensitive a role the success rate plays in making algorithm comparisons. Note that the parameters used in [3,4] achieved success probabilities $\mathtt{D}_{\text{ps}} = 51.9\ \%$ and $\mathtt{R}_{\text{ps}} = 55.6\ \%$. According to Proposition 36, the optimal tradeoff performances of the DP tradeoff at the two success rates are

$$\mathtt{D}_{\text{ps}} = 51.9\ \%, \qquad \mathtt{D}_{\text{tc}} = 1.95, \qquad \mathtt{D}_{\text{pc}} = 0.899, \tag{46}$$

and

$$\mathtt{D}_{\text{ps}} = 55.6\ \%, \qquad \mathtt{D}_{\text{tc}} = 2.56, \qquad \mathtt{D}_{\text{pc}} = 0.996. \tag{47}$$

The figures of (46) show that the typical parameters $m = t = \ell = \mathsf{N}^{1/3}$ considered in [3,4] should not be used. We can obtain the success probability of (44) at a better tradeoff efficiency and with a smaller investment in precomputation.

A comparison of (46) and (47) clearly shows that a small difference in success rate can lead to a large difference in the optimal tradeoff coefficient. It can be seen from Proposition 36 that the optimal tradeoff coefficient will become even more sensitive to the success probability as the demand on success rate is increased.

The figures we gave concerning the success rate difference were not as dramatic as those concerning the alarm resolving cost in that no conclusion was overturned. However, since the performances of the different algorithms are close to each other, it is clear that the ability to accurately predict the success probabilities of tradeoff algorithms is critical in comparing the tradeoff algorithms.

9. Conclusion

In this work, we analyzed the running time complexities of the DP, Hellman, and rainbow tradeoffs, and summarized their abilities to balance storage against online time as tradeoff curves that are correct up to small multiplicative factors. These results were used in the later part of this work to compare the performances of tradeoff algorithms against each other. Our comparison is different from previous attempts in that the efforts for precomputation have been taken into account.

Although we did provide explicit statements comparing the three tradeoff algorithms, our conclusions are only true under certain assumptions concerning the tradeoff environment. We emphasize once more that one should not blindly extend our conclusions to other situations. Rather, researchers should see this work as providing the tools and methodology for fair comparisons of tradeoff algorithms and use them to arrive at their own final judgements specific to their circumstances.

One conclusion we can provide about the relative performances of different tradeoff algorithms is that their differences will be small. The practical inconvenience of having to align each entry of the precomputed table at a byte boundary has not been considered in this work, and the performance differences between algorithms can be so small that such obscure issues may be of equal importance in practice. This fact is disappointing to us as authors of the current work, but should be relieving to practitioners of the tradeoff algorithm that are not concerned with small performance differences. Nevertheless, even if one decides to ignore small performance differences, the comparison graphs of the previous section show that a meaningful reduction in precomputation cost can be achieved with only a small sacrifice in tradeoff efficiency, and being able to take advantage of this knowledge will be of practical importance. Furthermore, with extremely large-scale implementations, having accurate access to the small differences will be of significant value.

Complexity analyses of perfect table versions of the tradeoff algorithms at the accuracy level treated in this paper and their inclusion into the tradeoff performance comparison picture remain to be done. Perfect table tradeoffs are expected to display better tradeoff efficiency and are certainly of interest, even though they require larger amounts of precomputation.

Appendix A. Technical Approximation

The following lemma shows that the approximation $(1-\frac{1}{\mathsf{b}})^{\mathsf{a}} \approx e^{-\frac{\mathsf{a}}{\mathsf{b}}}$, which we have used frequently in this work, is very accurate for large integers a and b such that $\mathsf{a} = O(\mathsf{b})$.

Lemma 39. *For positive integers* a *and* b, *we have*

$$\left|\exp\left(-\frac{\mathsf{a}}{\mathsf{b}}\right) - \left(1-\frac{1}{\mathsf{b}}\right)^{\mathsf{a}}\right| < \left\{\frac{1}{2}\frac{\mathsf{a}}{\mathsf{b}^2} + \frac{1}{(\mathsf{a}+1)!}\left(\frac{\mathsf{a}}{\mathsf{b}}\right)^{\mathsf{a}+1}\right\}\exp\left(\frac{\mathsf{a}}{\mathsf{b}}\right).$$

Proof. We start by writing $\exp(-\frac{\mathsf{a}}{\mathsf{b}})$ in its Taylor series form and fully expanding the term $(1-\frac{1}{\mathsf{b}})^{\mathsf{a}}$.

$$\begin{aligned}
&\left|\exp\left(-\frac{\mathsf{a}}{\mathsf{b}}\right) - \left(1-\frac{1}{\mathsf{b}}\right)^{\mathsf{a}}\right| \\
&\quad = \left|\left\{1 - \frac{\mathsf{a}}{\mathsf{b}} + \frac{1}{2!}\left(\frac{\mathsf{a}}{\mathsf{b}}\right)^2 - \cdots\right\} - \left\{1 - \binom{\mathsf{a}}{1}\frac{1}{\mathsf{b}} + \binom{\mathsf{a}}{2}\frac{1}{\mathsf{b}^2} - \cdots + (-1)^{\mathsf{a}}\binom{\mathsf{a}}{\mathsf{a}}\frac{1}{\mathsf{b}^{\mathsf{a}}}\right\}\right|.
\end{aligned}$$

After noting that the beginning two pairs of terms cancel out, we collect corresponding pairs from the two sequences of terms and bound the above by

$$\left\{\left|\frac{\mathsf{a}^2}{2!} - \binom{\mathsf{a}}{2}\right|\frac{1}{\mathsf{b}^2} + \cdots + \left|\frac{\mathsf{a}^{\mathsf{a}}}{\mathsf{a}!} - \binom{\mathsf{a}}{\mathsf{a}}\right|\frac{1}{\mathsf{b}^{\mathsf{a}}}\right\} + \left\{\frac{1}{(\mathsf{a}+1)!}\left(\frac{\mathsf{a}}{\mathsf{b}}\right)^{\mathsf{a}+1} + \cdots\right\}. \tag{A.1}$$

It is easy to see that

$$\begin{aligned}
0 \le \frac{\mathsf{a}^k}{k!} - \binom{\mathsf{a}}{k} &= \frac{1}{k!}\left\{\mathsf{a}^k - \mathsf{a}(\mathsf{a}-1)\cdots(\mathsf{a}-k+1)\right\} \\
&= \frac{1}{k!}\left\{\frac{k(k-1)}{2}\mathsf{a}^{k-1} - \cdots + (-1)^k(k-1)!\mathsf{a}\right\} \\
&\le \frac{1}{k!}\frac{k(k-1)}{2}\mathsf{a}^{k-1} = \frac{1}{2}\frac{\mathsf{a}^{k-1}}{(k-2)!},
\end{aligned}$$

for every $k \ge 2$, where the last inequality can be checked through induction on k. This shows that the terms of (A.1) that appear inside the first set of braces are bounded by

$$\begin{aligned}
&\frac{1}{2}\left\{\frac{\mathsf{a}}{0!}\frac{1}{\mathsf{b}^2} + \frac{\mathsf{a}^2}{1!}\frac{1}{\mathsf{b}^3} + \frac{\mathsf{a}^3}{2!}\frac{1}{\mathsf{b}^4} + \cdots + \frac{\mathsf{a}^{\mathsf{a}-1}}{(\mathsf{a}-2)!}\frac{1}{\mathsf{b}^{\mathsf{a}}}\right\} \\
&\quad = \frac{1}{2}\frac{\mathsf{a}}{\mathsf{b}^2}\left\{1 + \frac{1}{1!}\frac{\mathsf{a}}{\mathsf{b}} + \frac{1}{2!}\left(\frac{\mathsf{a}}{\mathsf{b}}\right)^2 + \cdots + \frac{1}{(\mathsf{a}-2)!}\left(\frac{\mathsf{a}}{\mathsf{b}}\right)^{\mathsf{a}-2}\right\} \\
&\quad \le \frac{1}{2}\frac{\mathsf{a}}{\mathsf{b}^2}\exp\left(\frac{\mathsf{a}}{\mathsf{b}}\right).
\end{aligned}$$

As for the second set of braces from (A.1), it is easy to see that

$$\frac{1}{(\mathsf{a}+1)!}\left(\frac{\mathsf{a}}{\mathsf{b}}\right)^{\mathsf{a}+1}\exp\left(\frac{\mathsf{a}}{\mathsf{b}}\right)$$

can serve as its very rough bound. It now suffices to gather the two bounds to arrive at the claim. □

Appendix B. Random Function Arguments

Any analysis of a tradeoff algorithm assumes the one-way function F to be a one-way function, and most results given in this work as equations are certain values expected of a random function. In other words, we have been stating values that had been averaged over the choice of all functions $F:\mathcal{N}\to\mathcal{N}$. In this section, we point out that many of the arguments made during these computations are not strictly correct, and we then try to justify heuristically that the existing logical error may safely be ignored.

B.1. *Existence of a Logical Gap*

Recall the expected image size of a random function given by (1) and the expected iterated image sizes given by (2). The claim that (1) implies (2) is acceptable in the realm of cryptology. In this subsection, we clarify that there is a small logical gap in such a claim.

Let us rewrite (1) as an explicit self-contained statement which is precisely correct.

Lemma 40. *Let $F:\mathcal{N}\to\mathcal{N}$ be a random function on a finite set of size* N. *If $\mathcal{M}\subset\mathcal{N}$ is of size m_0, then the size of $F(\mathcal{M})$ is expected to be*

$$m_1=\mathsf{N}\left\{1-\left(1-\frac{1}{\mathsf{N}}\right)^{m_0}\right\}.$$

The proof of this lemma is quite trivial. It suffices to consider the ratio of points among $\mathcal{N}$ that remain untouched throughout the sequential assignments made to elements of $\mathcal{M}$ for the random function construction.

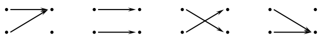
We want to emphasize two things about this lemma. The first is that the value claimed by this lemma is the exact expected value and does not involve any approximation. In fact, the largest reason for rewriting the statement here was to remove the approximate expression. The second point we make is that the statement of this lemma does not contain any averaging over input sets. The expected image size claim holds true for every set $\mathcal{M}\subset\mathcal{N}$ of size m_0.

Discussing just the double iteration case will be sufficient for our purposes. Let us define

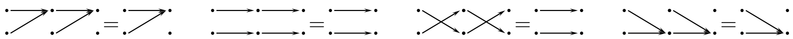
$$m_1=\mathsf{N}\left\{1-\left(1-\frac{1}{\mathsf{N}}\right)^{m_0}\right\}\quad\text{and}\quad m_2=\mathsf{N}\left\{1-\left(1-\frac{1}{\mathsf{N}}\right)^{m_1}\right\},\tag{B.1}$$

for any given m_0. One might believe that m_2 is the expected size of $F^2(\mathcal{M})$, when $F:\mathcal{N}\to\mathcal{N}$ is a random function and $\mathcal{M}\subset\mathcal{N}$ is of size m_0. Since Lemma 40 contains no approximation, one might expect (B.1) to hold exactly. However, this reasonable prediction is not met, at least in the strict sense, by the explicit example given below.

The set of all functions $F : \{0, 1\} \to \{0, 1\}$ can be visualized as follows.



When the input set $\mathscr{M}$ is a single point, the image size expectation is clearly 1. This is in agreement with the value $2\{1 - (1 - \frac{1}{2})^1\} = 1$, computed according to Lemma 40. When the input set is the complete domain $\{0, 1\}$, the image size expectation is $E_F[|F(\{0, 1\})|] = \frac{1}{4} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{4} \cdot 2 + \frac{1}{4} \cdot 1 = \frac{3}{2}$, and this is also identical to the value $E_F[|F(\{0, 1\})|] = 2\{1 - (1 - \frac{1}{2})^2\} = \frac{3}{2}$, computed according to Lemma 40. We have just verified that Lemma 40, which had already been proved, holds *exactly* for the $\mathscr{N} = \{0, 1\}$ case, regardless of the input set size and the choice of the set itself. Now, the four functions $F^2 = F \circ F$ can be visualized as follows.



When the input set $\mathscr{M}$ is taken to be the complete domain, the expected image size of the double iteration is

$$E_F\big[\big|F^2\big(\{0, 1\}\big)\big|\big] = \frac{2}{4} \cdot 1 + \frac{2}{4} \cdot 2 = \frac{3}{2}. \tag{B.2}$$

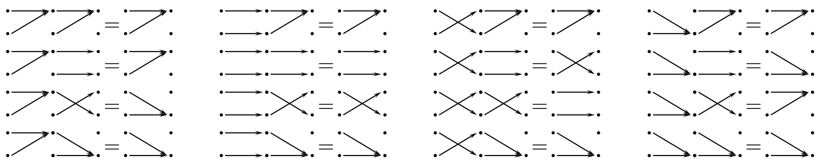
In comparison, the corresponding value computed through (B.1) is

$$2\left\{1 - \left(1 - \frac{1}{2}\right)^{2\{1-(1-1/2)^2\}}\right\} = 2\left\{1 - \left(1 - \frac{1}{2}\right)^{3/2}\right\} \approx 1.293. \tag{B.3}$$

The two values given above are clearly in disagreement.

A cryptographer would naturally attempt to rectify the current situation by relaxing the strict correlation between the two functions that are being composed. Let $F : \mathscr{N} \to \mathscr{N}$ and $G : \mathscr{N} \to \mathscr{N}$ be two independent random functions operating on a finite set of size N. One would like to claim that if $\mathscr{M} \subset \mathscr{N}$ is of size m_0, then the size of $G(F(\mathscr{M}))$ is expected to be the m_2 value given by (B.1). This second version for the doubly iterated image size expectation seems structurally much simpler to analyze than the previous attempt, and one might be tempted to say that the modified claim is a *trivial* consequence of Lemma 40.

We again turn to the example $F, G : \{0, 1\} \to \{0, 1\}$. The complete set of all possible double iterations can be visualized as follows.



When the input set $\mathscr{M}$ is the full domain $\{0, 1\}$, after separately counting the number of functions with image sizes one and two, the expected image size can be computed as

$$E_{F,G}\big[\big|G\big(F\big(\{0,1\}\big)\big)\big|\big] = \frac{12}{16}\cdot 1 + \frac{4}{16}\cdot 2 = \frac{5}{4}. \tag{B.4}$$

Once again, this disagrees with (B.3), which was computed through (B.1).

It is now clear that (2) does not directly follow from (1). The claims to the iterated image sizes are not consequences of the single step image size, at least not without additional arguments. The logical gap persists even when all iterations are allowed to be independent random functions.

B.2. *Narrowing the Logical Gap*

The failed attempt (B.1) at giving a doubly iterated image size expectation had substituted the m_1 value in the place of m_0 in the single step result Lemma 40. This reuse of average value in the computation of another average value was the source of our problem. In reality, as can be seen in the two counterexamples, inputs to the second step function are not all of m_1 size, but of varying sizes that only average to m_1. After this simple observation, we can state that, if $\mathscr{M}_0$ is a set of size m_0 such that the image size $|F(\mathscr{M}_0)|$ is exactly m_1 for every choice of function F and the image size $|F(\mathscr{M}_1)|$ is exactly m_2 for every choice of function F and every input set $\mathscr{M}_1$ of size m_1, then m_2 is the exact expected size of $F^2(\mathscr{M}_0)$. The assumptions included in this statement cannot be met, but it is reasonable to expect the conclusion to hold approximately, when a slight relaxation is given to the assumptions. We are thus justified in stating that, if for the vast majority of the sets $\mathscr{M} \subset \mathscr{N}$ and functions $F : \mathscr{N} \to \mathscr{N}$, the image size $|F(\mathscr{M})|$ is very close to $\mathsf{N}\{1 - (1 - \frac{1}{\mathsf{N}})^{|\mathscr{M}|}\}$, then the m_2 of (B.1) will be a good approximation for the doubly iterated image size expectation.

Therefore, we consider the images of a fixed set $\mathscr{M}$ under different functions F and discuss how their sizes $|F(\mathscr{M})|$ are distributed around its average. Let us use $\mu_{\mathsf{N},m}$ and $\sigma_{\mathsf{N},m}$ to denote the average and standard deviation of the image set size $|F(\mathscr{M})|$. These are to be computed for a fixed input set $\mathscr{M} \subset \mathscr{N}$ of size m and with $F : \mathscr{N} \to \mathscr{N}$ running over all possible function choices. We already know $\mu_{\mathsf{N},m} \approx \mathsf{N}\{1 - \exp(-\frac{m}{\mathsf{N}})\}$. A proof of the following lemma is given in Appendix C.

Lemma 41. *We have* $\frac{\sigma_{\mathsf{N},m}}{\mu_{\mathsf{N},m}} < \frac{2}{\sqrt{\mathsf{N}}}$ *for all* N *and* m.

According to Chebyshev's inequality, at least 99 % of the N^{N} image sizes will fall within the range $\mu_{\mathsf{N},m} \pm 10\sigma_{\mathsf{N},m}$. The above lemma states that this deviation of sizes from the mean is bounded by $\frac{20\mu_{\mathsf{N},m}}{\sqrt{\mathsf{N}}}$. Hence, the distribution or clustering of image sizes around the expected value $\mu_{\mathsf{N},m}$ will tighten, at least in comparison to the expected value, as N is increased.

This observation can be restated in more plain terms as follows. Suppose we take some input set and measure its image size under a single function, chosen at random, and take it to be an estimate of the true average image size. We make it clear that the averaging over multiple measurements made with multiple functions is not being performed here. In such a situation, we can expect each measurement to return a larger number of *significant digits* as N is increased. Let us briefly work with some explicit numbers. For parameters $\mathsf{N} = 2^{64}$ and $m = 2^{50}$ the average image size can be computed to be $\mu_{\mathsf{N},m} \approx 1.13 \times 10^{16}$. For the same parameters, the standard deviation is bounded by $\sigma_{\mathsf{N},m} \leq 5.24 \times 10^5$. Chebyshev's inequality ensures that at least 99 % of the N^{N} image sizes will lie in the range $\mu_{\mathsf{N},m} \pm 10\sigma_{\mathsf{N},m}$, which is $1.13 \times 10^{16} \pm 5.24 \times 10^6$ in the current situation. For any practical purposes, we can believe that close to 10 significant digits from any single measurement are highly likely to be identical to those of the true expected value.

Let us summarize the discussion of this subsection. For any function acting on a large set that was chosen at random and any input set of size m_0, the image size of the first iteration will be very close to the m_1 value given by (B.1). At the second iterated application of the same function, even though the input size was not exactly m_1, we can expect the output size to be very close to the m_2 value given by (B.1). Actually, the output size could be different from m_2 even if the input size was exactly m_1. In any case, the fact that the standard deviation of the image sizes is very small relative to its expected value implies a tight clustering of image sizes, and allows us to believe that the formula (2) will predict doubly iterated image sizes with accuracy, in the sense that a large number of significant digits are returned. The heuristic arguments of this subsection have added further justification to the already acceptable cryptographic argument that (1) implies (2).

B.3. *Other Reuses of Average Values*

The intention of this section was not to test the validity of (2). In fact, although the authors of the current paper are unfit to verify its correctness, a full proof is provided in [11] for at least the case when $\mathscr{M}$ is the full domain. What we have done so far in the current section is to first point out that average values have erroneously been reused in the computation of other average values and then argue heuristically that such methods are still acceptable as long as the distribution of values that are being treated is tightly gathered around the average. This reasoning does not have to be restricted to the discussion of iterated image sizes, or even random function arguments.

There are many occasions in this paper where an average value was used during the computation of another average value. It should now be clear that (10), stating the success probability of a single rainbow matrix, is also slightly problematic, but acceptable. The different reduction functions at each rainbow matrix column do not provide independence of the colored iterating functions, and the existing logical gap would not be closed even if different columns were processed with independent random functions. However, the small standard deviation of image sizes justifies (10) as a good approximation.

The success probability (4) of the DP and Hellman tradeoffs, computed from the average number of points in a tradeoff matrix, is another example of average value reuse. We have not checked if the standard deviation of the coverage rate is small, but we know from experience that (4) predicts the correct value accurately, so this should not be a problem. In fact, this situation is less problematic than the iterated image case, because the arguments become strictly correct when independent random functions are used in different tables.

Readers may have noticed that we were more careful in reusing average values in Sect. 4.2. The distribution of chain lengths in a DP matrix can be inferred from (16), and it is clear that the lengths are not at all centered around the average length t. Hence, we were careful to work with the full range of possible chain lengths, rather than treat t as being the typical precomputation or online chain length. In particular, we did not treat the DP matrix as consisting of m chains of identical length t. This cautious handling of chains should not be confused with our free use of the value (16) itself, which is an expected value, in other computations.

Appendix C. Standard Deviation of Image Sizes

The purpose of the section is to provide a proof to Lemma 41 concerning the standard deviation of image sizes. We first prepare a couple of technical lemmas.

Lemma 42. *Let $F : \mathscr{N} \to \mathscr{N}$ be a random function. Fix a subset $\mathscr{M} \subset \mathscr{N}$ of size m and let $\mathbf{y}_1, \mathbf{y}_2 \in \mathscr{N}$ be any two distinct points. The probability for $F(\mathscr{M})$ to contain both $\mathbf{y}_1$ and $\mathbf{y}_2$ is*

$$\left\{1-\left(1-\frac{1}{\mathsf{N}}\right)^m\right\}^2 - \left(1-\frac{1}{\mathsf{N}}\right)^m\left\{\left(1-\frac{1}{\mathsf{N}}\right)^m - \left(1-\frac{1}{\mathsf{N}-1}\right)^m\right\}.$$

Proof. The probability under consideration may be computed as follows:

$$\begin{aligned}
&\binom{m}{1}\left(\frac{1}{\mathsf{N}}\right)^1\left(1-\frac{1}{\mathsf{N}}\right)^{m-1}\left\{1-\left(1-\frac{1}{\mathsf{N}-1}\right)^{m-1}\right\}\\
&\quad+\binom{m}{2}\left(\frac{1}{\mathsf{N}}\right)^2\left(1-\frac{1}{\mathsf{N}}\right)^{m-2}\left\{1-\left(1-\frac{1}{\mathsf{N}-1}\right)^{m-2}\right\}\\
&\quad+\cdots\\
&\quad+\binom{m}{m-1}\left(\frac{1}{\mathsf{N}}\right)^{m-1}\left(1-\frac{1}{\mathsf{N}}\right)^{1}\left\{1-\left(1-\frac{1}{\mathsf{N}-1}\right)^{1}\right\}.
\end{aligned}$$

In each additive term, the part $\binom{m}{k}(\frac{1}{\mathsf{N}})^k(1-\frac{1}{\mathsf{N}})^{m-k}$ gives the probability for exactly k out of the m inputs to map to $\mathbf{y}_1$. The remaining $\{1-(1-\frac{1}{\mathsf{N}-1})^{m-k}\}$ part is the probability for at least one of the $(m-k)$ inputs that are known not to have reached $\mathbf{y}_1$ to map to $\mathbf{y}_2$. The above sum is equal to the expression

$$\begin{aligned}
&\left\{\frac{1}{\mathsf{N}}+\left(1-\frac{1}{\mathsf{N}}\right)\right\}^m - \left\{\frac{1}{\mathsf{N}}+\left(1-\frac{1}{\mathsf{N}}\right)\left(1-\frac{1}{\mathsf{N}-1}\right)\right\}^m - \left(1-\frac{1}{\mathsf{N}}\right)^m\\
&\quad+\left(1-\frac{1}{\mathsf{N}}\right)^m\left(1-\frac{1}{\mathsf{N}-1}\right)^m.
\end{aligned}$$

To check this claim, it suffices to expand the first two pairs of braces. This expression can be rewritten in the form stated by this lemma. □

Lemma 43. *For positive integers* N *and* m, *we have*

$$\left(1-\frac{1}{\mathsf{N}}\right)^m - \left(1-\frac{1}{\mathsf{N}-1}\right)^m \geq \frac{m}{\mathsf{N}(\mathsf{N}-1)}\left(1-\frac{1}{\mathsf{N}-1}\right)^{m-1}.$$

Proof. It suffices to check the following sequence of equalities and inequality:

$$\begin{aligned}
&\left(1-\frac{1}{\mathsf{N}}\right)^m - \left(1-\frac{1}{\mathsf{N}-1}\right)^m \\
&\quad = \left\{\left(1-\frac{1}{\mathsf{N}}\right) - \left(1-\frac{1}{\mathsf{N}-1}\right)\right\}\left\{\left(1-\frac{1}{\mathsf{N}}\right)^{m-1} + \left(1-\frac{1}{\mathsf{N}}\right)^{m-2}\left(1-\frac{1}{\mathsf{N}-1}\right)\right. \\
&\qquad \left. + \cdots + \left(1-\frac{1}{\mathsf{N}-1}\right)^{m-1}\right\} \\
&\quad = \frac{1}{\mathsf{N}(\mathsf{N}-1)}\left\{\left(1-\frac{1}{\mathsf{N}}\right)^{m-1} + \left(1-\frac{1}{\mathsf{N}}\right)^{m-2}\left(1-\frac{1}{\mathsf{N}-1}\right)\right. \\
&\qquad \left. + \cdots + \left(1-\frac{1}{\mathsf{N}-1}\right)^{m-1}\right\} \\
&\quad \geq \frac{1}{\mathsf{N}(\mathsf{N}-1)} m \left(1-\frac{1}{\mathsf{N}-1}\right)^{m-1}.
\end{aligned}$$

In fact, a similar upper bound is also easy to obtain. □

In the remainder of this section, $\mathscr{M} \subset \mathscr{N}$ will be a fixed set of size m. For each $\mathbf{y} \in \mathscr{N}$, let us define the function $\chi_{\mathbf{y}} : \mathscr{N}^{\mathscr{N}} \to \{0, 1\}$ by

$$\chi_{\mathbf{y}}(F) = \begin{cases} 1 & \text{if } \mathbf{y} \in F(\mathscr{M}), \\ 0 & \text{if } \mathbf{y} \notin F(\mathscr{M}). \end{cases}$$

The dependence of $\chi_{\mathbf{y}}$ on $\mathscr{M}$ was not made explicit in the notation since we will keep $\mathscr{M}$ fixed for the rest of this section. The size of the image of $\mathscr{M}$ under any function $F : \mathscr{N} \to \mathscr{N}$ can be expressed in terms of this indicator function as

$$\left|F(\mathscr{M})\right| = \sum_{\mathbf{y}\in\mathscr{N}} \chi_{\mathbf{y}}(F).$$

Using this observation, one can present

$$\begin{aligned}
E\big[\left|F(\mathscr{M})\right|\big] &= E\Bigg[\sum_{\mathbf{y}\in\mathscr{N}} \chi_{\mathbf{y}}(F)\Bigg] = \sum_{\mathbf{y}\in\mathscr{N}} E\big[\chi_{\mathbf{y}}(F)\big] \\
&= \mathsf{N}E\big[\chi_{\mathbf{y}'}(F)\big] = \mathsf{N}\left\{1-\left(1-\frac{1}{\mathsf{N}}\right)^m\right\}, \qquad \text{(C.1)}
\end{aligned}$$

where $\mathbf{y}'$ is any fixed point of $\mathscr{N}$, as an alternative way of writing the proof to Lemma 40.

Let us fix the notation

$$\chi = \sum_{\mathbf{y}\in\mathscr{N}} \chi_{\mathbf{y}}$$

and view this as a random variable defined on the space $\mathscr{N}^{\mathscr{N}}$, which is given the uniform probability distribution. It maps each element F to the positive integer $|F(\mathscr{M})|$. Equation (C.1) is equivalent to

$$E[\chi] = \mathsf{N}\left\{1 - \left(1 - \frac{1}{\mathsf{N}}\right)^m\right\} \tag{C.2}$$

and we need to work with the standard deviation

$$\operatorname{stdev}(\chi) = \sqrt{E\big[\chi^2\big] - \big(E[\chi]\big)^2}.$$

One can easily check that

$$\begin{aligned} E\big[\chi^2\big] &= E\left[\left(\sum_{\mathbf{y}} \chi_{\mathbf{y}}\right)^2\right] = E\left[\sum_{\mathbf{y}_1,\mathbf{y}_2} \chi_{\mathbf{y}_1}\chi_{\mathbf{y}_2}\right] = E\left[\sum_{\mathbf{y}} \chi_{\mathbf{y}} + \sum_{\mathbf{y}_1\neq\mathbf{y}_2} \chi_{\mathbf{y}_1}\chi_{\mathbf{y}_2}\right] \\ &= E[\chi] + \sum_{\mathbf{y}_1\neq\mathbf{y}_2} E[\chi_{\mathbf{y}_1}\chi_{\mathbf{y}_2}] \\ &= E[\chi] + \mathsf{N}(\mathsf{N}-1)E[\chi_{\mathbf{y}'_1}\chi_{\mathbf{y}'_2}], \end{aligned}$$

where $\mathbf{y}'_1$ and $\mathbf{y}'_2$ are any two distinct points of $\mathscr{N}$. The expectation $E[\chi_{\mathbf{y}'_1}\chi_{\mathbf{y}'_2}]$ is equal to the probability for both $\mathbf{y}'_1$ and $\mathbf{y}'_2$ to belong to the image space, and this is the content of Lemma 42. Referring also to (C.2) and Lemma 43, we can compute a bound for the variance as follows:

$$\begin{aligned} \big\{\operatorname{stdev}(\chi)\big\}^2 &= E\big[\chi^2\big] - \big(E[\chi]\big)^2 = E[\chi] + \mathsf{N}(\mathsf{N}-1)E[\chi_{\mathbf{y}'_1}\chi_{\mathbf{y}'_2}] - \big(E[\chi]\big)^2 \\ &= \mathsf{N}\left(1-\frac{1}{\mathsf{N}}\right)^m \left(\left\{1 - \left(1-\frac{1}{\mathsf{N}}\right)^m\right\}\right. \\ &\qquad \left. - (\mathsf{N}-1)\left\{\left(1-\frac{1}{\mathsf{N}}\right)^m - \left(1 - \frac{1}{\mathsf{N}-1}\right)^m\right\}\right) \\ &\le \mathsf{N}\left(\left\{1 - \left(1-\frac{1}{\mathsf{N}}\right)^m\right\} - \frac{m}{\mathsf{N}}\left(1 - \frac{1}{\mathsf{N}-1}\right)^{m-1}\right) \\ &\le \mathsf{N}\left\{\frac{m}{\mathsf{N}} - \frac{m}{\mathsf{N}}\left(1 - \frac{m-1}{\mathsf{N}-1}\right)\right\} = \frac{m(m-1)}{\mathsf{N}-1} \le \frac{m^2}{\mathsf{N}}. \end{aligned}$$

Here, the second inequality follows from the observation $(1-\frac{1}{\mathsf{N}})^m \ge 1 - \frac{m}{\mathsf{N}}$. The final expression allows us to state that $\operatorname{stdev}(\chi) \le \frac{m}{\sqrt{\mathsf{N}}}$.

On the other hand, from the observation $(1-\frac{1}{\mathsf{N}})^m \le 1 - \frac{m}{\mathsf{N}} + \frac{m(m-1)}{2\mathsf{N}^2}$, which holds for every $m \le \mathsf{N}$, we know that

$$E[\chi] \ge \mathsf{N}\left\{\frac{m}{\mathsf{N}} - \frac{m(m-1)}{2\mathsf{N}^2}\right\} > \mathsf{N}\left(\frac{m}{\mathsf{N}} - \frac{m}{2\mathsf{N}}\right) = \frac{m}{2}.$$

Finally, by combining the two bounds, we can state that

$$\frac{\operatorname{stdev}(\chi)}{E[\chi]} < \frac{2}{\sqrt{\mathsf{N}}}.$$

This concludes the proof of Lemma 41.

Appendix D. Note on the Index Tables Method

The index table method can be seen as a special case of a more general and widely known data structure called *hash tables*. To store m starting point and ending point pairs, one first fixes a *hash function* that maps elements of $\mathcal{N}$ to $\log m$-bit strings. This function need not be a cryptographic hash function, although the same term is used. Instead of sorting the data, each starting point and ending point pair is recorded at the position in the storage addressed by the hash value of the ending point. Collisions of addresses are inevitable, but there are various ways to deal with this problem.

Table lookups to hash tables are performed by first hashing the ending point to be searched for in the table and fetching the data located at the address pointed to by the hash value. Since the address holds $\log m$ bits of information, even if almost $\log m$ bits from each ending point are removed before storage, we can reliably determine whether or not a match has occurred.

One advantage of the hash table method, other than reducing storage and not requiring any sorting, is that it provides constant time table lookups. In comparison, a lookup to a sorted table requires time that is logarithmic in the table size.

If the hash function is set to return the first $\{(\log m) - \varepsilon\}$ bits of its input and buckets to hold approximately 2^ε table entries are placed at the position pointed to by each hash value, then the hash table technique reduces to the index table technique.

Appendix E. Experimental Results

In this section we verify that the main parts of our arguments agree well with the experimental results. Experiments are done to check the validity of our results concerning the coverage rate and the cost of false alarms for the DP tradeoff. Analogous testing for the Hellman and rainbow tradeoffs is not provided, as this testing was done in [15]. We also provide experimental evidence supporting our arguments surrounding the effects of the ending point truncation method.

Since averaging over all functions defined on any reasonably large space is not at all possible, all our tests were conducted with a very small subset of explicitly constructed one-way functions. The one-way function used was always the encryption key to ciphertext mapping, under a fixed plaintext, computed with the block cipher AES-128.

Table 2. Coverage rate of DP tradeoff ($\mathsf{N} = 2^{30}$)

$\log m$	$\log t$	$\hat{t}/t$	$\mathrm{D}_{\mathrm{msc}}$	Test	Theory
11	9	1/2	0.5	0.225357	0.224285
9	10	1/2	0.5	0.225368	0.224285
11	9	1	0.5	0.400071	0.399566
9	10	1	0.5	0.398824	0.399566
11	9	2	0.5	0.628349	0.627405
9	10	2	0.5	0.629802	0.627405
11	9	5	0.5	0.816415	0.814339
9	10	5	0.5	0.811530	0.814339
12	9	1/2	1.0	0.221190	0.219643
10	10	1/2	1.0	0.220655	0.219643
12	9	1	1.0	0.384839	0.383464
10	10	1	1.0	0.385049	0.383464
12	9	2	1.0	0.582370	0.581801
10	10	2	1.0	0.581019	0.581801
12	9	5	1.0	0.722192	0.723263
10	10	5	1.0	0.721465	0.723263
13	9	1/2	2.0	0.212476	0.211204
11	10	1/2	2.0	0.212538	0.211204
13	9	1	2.0	0.357424	0.356587
11	10	1	2.0	0.355287	0.356587
13	9	2	2.0	0.515214	0.515495
11	10	2	2.0	0.514631	0.515495
13	9	5	2.0	0.611834	0.612748
11	10	5	2.0	0.610616	0.612748

Different randomly chosen plaintexts were used to provide multiple one-way functions. The size of the input space was controlled by utilizing only a small number of key bits and padding the remaining key bits with zeros. The output space size was controlled by masking the ciphertext to an appropriate bit length. When working with the DP tradeoff, as discussed at the start of Sect. 4, we constructed $m_0 = \frac{m}{1-e^{-\hat{t}/t}}$ precomputation chains and gathered every resulting DP chain, rather than incrementally generating additional chains until m DP chains were collected.

E.1. *Coverage Rate of DP Tradeoffs*

The experimental results supporting Proposition 9, which presents the coverage rate of a DP table, are given in Table 2. The coverage rate was measured by simply storing all DP matrix entries while constructing the DP chains and later counting the number of distinct matrix entries that were used as inputs to the one-way function. Each test result value given in the table is an average over 100 experiments. Different randomly generated plaintexts for AES were used for each of these experiments. All the tests were done on a space of size $\mathsf{N} = 2^{30}$. One can check that the test figures are very close to what the theory predicted.

E.2. *Cost of Resolving Alarms for the DP Tradeoff*

Our next goal is to check the validity of our arguments concerning the time complexity that incorporates the extra cost of false alarms. We could do this with the expression for time complexity stated during the proof of Theorem 13, but such an approach would hide much of the inner workings. Hence, we decided to verify the following lemma, which allows access to much finer detail.

Lemma 44. *Consider the DP tradeoff. The expected number of chain collisions at the ith iteration of the online phase is*

$$\frac{1}{t}\frac{\mathrm{D_{msc}}}{1-e^{-\hat{t}/t}}\left\{-e^{-\hat{t}/t}+e^{-\hat{t}/t}\exp\left(-\frac{i}{t}\right)+\frac{i}{t}\exp\left(-\frac{i}{t}\right)\right\}.$$

Proof. The expected number of chain collisions is the sum over all rows of the DP matrix of the respective probabilities for the ith iteration to sound an alarm in association with that row. After reading the proof of Lemma 12, it should be clear that the sum of probabilities we are looking for is

$$\sum_{j=1}^{\hat{t}}\frac{\frac{m}{t}}{1-e^{-\hat{t}/t}}\exp\left(-\frac{j}{t}\right)\cdot\frac{t}{\mathsf{N}}\left\{\exp\left(\frac{\min\{i,j\}}{t}\right)-1\right\}\exp\left(-\frac{i}{t}\right).$$

In integral form, this is approximately

$$\frac{1}{t}\frac{\frac{mt^2}{\mathsf{N}}}{1-e^{-\hat{t}/t}}\exp\left(-\frac{i}{t}\right)\int_0^{\hat{t}/t}\exp(-v)\left\{\exp\left(\min\left\{\frac{i}{t},v\right\}\right)-1\right\}dv,$$

which simplifies to what is claimed. □

This lemma contains the core of our arguments given in the main text concerning the cost of alarms, and its verification through experiments should provide good support for the correctness of our theory.

To test this lemma, we first initialized an array of $\hat{t}$ counters to zeros. Next, we fixed a one-way function by randomly choosing a plaintext and constructed a DP table with the fixed function. Then, a random password (= zero-padded encryption key) was generated and the password hash (= masked ciphertext) corresponding to that password was computed. The online chain starting from the password hash was computed until a DP was found or the $\hat{t}$th iteration was reached. If the online chain terminated at a DP and it was found to reside within the DP table, the counter corresponding to the current online iteration count was incremented. The online chain generation was repeated multiple times with the same table, but with newly generated random keys. Note that, since we are not using perfect tables, it is possible for the online chain to collide simultaneously with more than one entry of the DP table. Care was taken to increment the counter corresponding to the current iteration count as many times as the number of collisions found. The whole process described after the counter initialization step was repeated multiple times, with each repetition using a newly generated one-way function and a DP table.

The test results for four different parameter sets are presented in Fig. 7. Each of these experiments was performed with 2000 tables and 5000 random online chains per table. In each of the four boxes, the barely visible thin dashed line represents our theory as

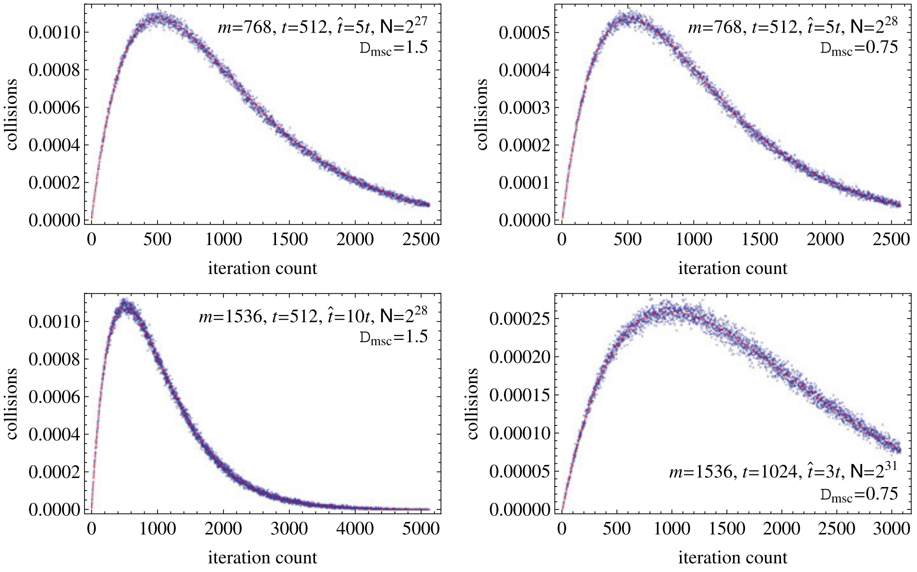


Fig. 7. Expected number of collisions at each iteration of the DP tradeoff (*dots*: experiment; *dashed line*: theory).

given by Lemma 44. There are $\hat{t}$-many tiny dots in each box, and these represent our experimental results. The height of the ith dot, counting from the left, is the value of the ith iteration counter at the end of the experiment divided by 2000 × 5000, the total number of chains that were utilized. All the experiment results match our theory very well.

E.3. *Ending Point Truncation*

Finally, we test the validity of our arguments concerning the ending point truncation method for reducing storage. The straightforward approach would be to simply test Lemma 16, Lemma 24, and Lemma 32, which present the cost of truncation related alarms, but we decided to work with the probability of alarms related to truncations, so as to expose more of our argument details to the tests.

Lemma 45. *Consider the DP tradeoff that uses an ending point truncation of $\frac{1}{r}$ truncated match probability. At the ith iteration of the online processing of a single DP table, the number of* pseudo-collisions *that are due to the ending point truncations, i.e., those that are not associated with any true chain collisions, is expected to be $\frac{m}{r}\exp(-\frac{i}{t})$. The corresponding value for the Hellman tradeoff is $\frac{m}{r}$, and that for the rainbow tradeoff is also $\frac{m}{r}$, if one decides to fully process a single rainbow table without terminating, even when the correct answer is found.*

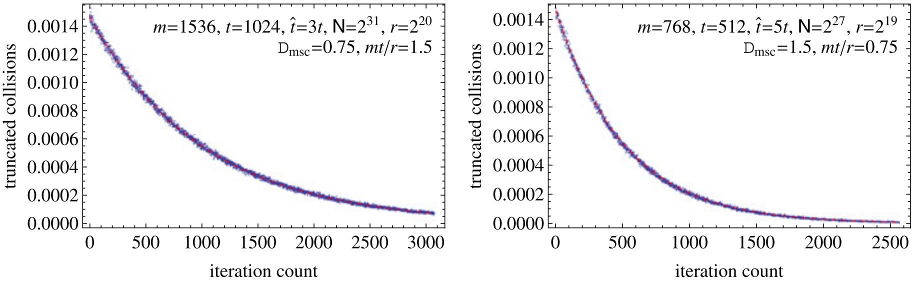


Fig. 8. Expected number of collisions, induced by ending point truncation, at each iteration of the DP tradeoff (*dots*: experiment; *dashed line*: theory).

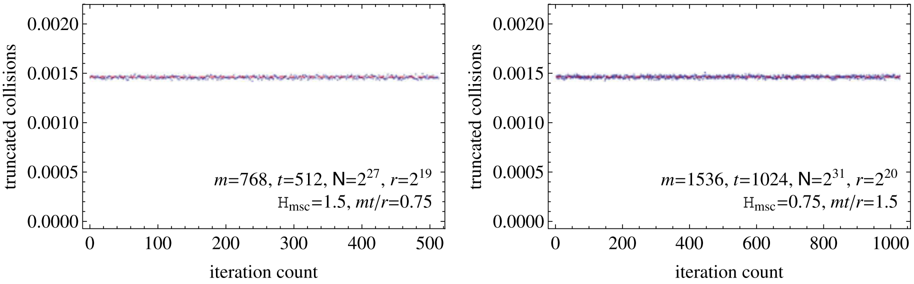


Fig. 9. Expected number of collisions, induced by ending point truncation, at each iteration of the Hellman tradeoff (*dots*: experiment; *dashed line*: theory).

Proof. The proof of Lemma 16 shows that the claimed expected value for the DP tradeoff case can be computed as

$$\sum_{j=1}^{\hat{t}} \frac{\frac{m}{t}}{1-e^{-\hat{t}/t}} \exp\left(-\frac{j}{t}\right) \cdot \exp\left(-\frac{i}{t}\right) \frac{1}{r} \approx \frac{m}{1-e^{-\hat{t}/t}} \int_0^{\hat{t}/t} \exp(-v)\, dv \exp\left(-\frac{i}{t}\right) \frac{1}{r},$$

which simplifies to what is claimed. The statement for the Hellman tradeoff case follows immediately from the proof of Lemma 24, and the rainbow tradeoff case can be inferred from the proof of Lemma 32. □

The three claims given by this lemma are at the core of our arguments concerning the ending point truncation method, and experimental verification of these statements should provide confidence as to the validity of our arguments given in the main text.

As in the previous section, we generated random tradeoff tables and tested with random online chains for the occurrence of alarms induced from truncations. We stored the full ending points, together with the truncated ending points, in the precomputation table. The full ending point information was used to distinguish between alarms that were caused by ending point truncations and those that arose from true chain collisions.

The test results are given in Figs. 8, 9, and 10. As before, the thin dashed lines are the graphs claimed in Lemma 45 and the numerous tiny dots represent the experimental

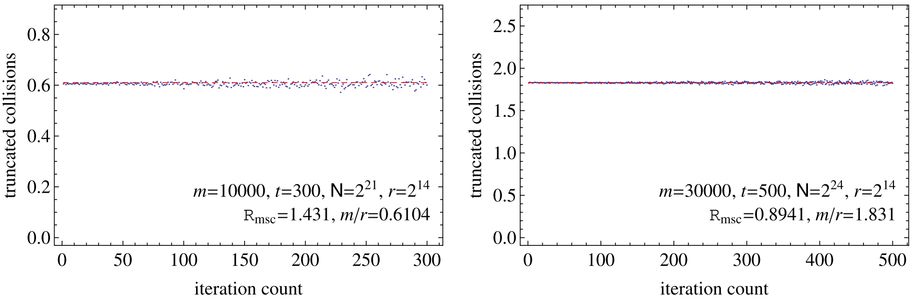


Fig. 10. Expected number of collisions, induced by ending point truncation, at each iteration of the rainbow tradeoff (*dots*: experiment; *dashed line*: theory).

data. All the test results are in good agreement with the theory. Each of the two diagrams for the DP tradeoff was obtained by averaging over 2000 tables and 5000 online chains per table. For the Hellman tradeoff we generated 2000 tables and 5000 inversion targets per table. The online chain was computed to the full length t for each inversion target, and a search was made for the truncated match with the table elements after each one-way function iteration. In the rainbow tradeoff case, each diagram is the result of 100 tables with 5000 inversion targets per table. Recall that the kth iteration for the rainbow tradeoff refers to a process that consists of $(k-1)$ invocations of the one-way function and one table lookup. Full t iterations were attempted for each inversion target; hence each inversion target generated t searches to the table for truncated matches.

References

[1] G. Avoine, P. Junod, P. Oechslin, Characterization and improvement of time-memory trade-off based on perfect tables. *ACM Trans. Inf. Syst. Secur.* **11**(4), 17:1–17:22 (2008). Preliminary version in INDOCRYPT 2005

[2] S.H. Babbage, Improved exhaustive search attacks on stream ciphers, in *European Convention on Security and Detection*. IEE Conference Publication, vol. 408 (IEE, London, 1995), pp. 161–166

[3] E.P. Barkan, Cryptanalysis of ciphers and protocols. Ph.D. Thesis, Israel Institute of Technology, March 2006

[4] E. Barkan, E. Biham, A. Shamir, Rigorous bounds on cryptanalytic time/memory tradeoffs, in *Advances in Cryptology—CRYPTO 2006*. LNCS, vol. 4117 (Springer, Berlin, 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, Berlin, 2000), pp. 1–13

[6] A. Biryukov, A. Shamir, D. Wagner, Real time cryptanalysis of A5/1 on a PC, in *FSE 2000*. LNCS, vol. 1978 (Springer, Berlin, 2001), pp. 1–18

[7] J. Borst, Block ciphers: Design, analysis, and side-channel analysis. Ph.D. Thesis, Katholieke Universiteit Leuven, September 2001

[8] J. Borst, B. Preneel, J. Vandewalle, On the time-memory tradeoff between exhaustive key search and table precomputation, in *Proceedings of the 19th Symposium on Information Theory in the Benelux* (WIC, 1998)

[9] C. Calik, How to invert one-way functions: time-memory trade-off method. M.S. Thesis, Middle East Technical University, January 2007

[10] D.E. Denning, *Cryptography and Data Security* (Addison-Wesley, Reading, 1982)

[11] P. Flajolet, A.M. Odlyzko, Random mapping statistics, in *Advances in Cryptology—EUROCRYPT'89*. LNCS, vol. 434 (Springer, Berlin, 1990), pp. 329–354

[12] S. Goldwasser, M. Bellare, Lecture notes on cryptography. Unpublished manuscript, July 2008. Available at: http://cseweb.ucsd.edu/~mihir/papers/gb.html

[13] J.Dj. Golić, Cryptanalysis of alleged A5 stream cipher, in *Advances in Cryptology—EUROCRYPT'97*. LNCS, vol. 1233 (Springer, Berlin, 1997), pp. 239–255

[14] M.E. Hellman, A cryptanalytic time-memory trade-off. *IEEE Trans. Inf. Theory* **26**, 401–406 (1980)

[15] J. Hong, The cost of false alarms in Hellman and rainbow tradeoffs. *Des. Codes Cryptogr.* **57**, 293–327 (2010)

[16] J. Katz, Y. Lindell, *Introduction to Modern Cryptography* (Chapman & Hall/CRC, London, 2008)

[17] I.-J. Kim, T. Matsumoto, Achieving higher success probability in time-memory trade-off cryptanalysis without increasing memory size. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **E82-A**, 123–129 (1999)

[18] K. Kusuda, T. Matsumoto, Optimization of time-memory trade-off cryptanalysis and its application to DES, FEAL-32, and Skipjack. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **79**(1), 35–48 (1996)

[19] D. Ma, J. Hong, Success probability of the Hellman trade-off. *Inf. Process. Lett.* **109**(7), 347–351 (2009)

[20] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography* (CRC Press, Boca Raton, 1997)

[21] S. Moon, Parameter selection in cryptanalytic time memory tradeoffs. M.S. Thesis, Seoul National University, June 2009

[22] A. Narayanan, V. Shmatikov, Fast dictionary attacks on passwords using time-space tradeoff, in *Proceedings of the 12th ACM CCS* (ACM, New York, 2005), pp. 364–372

[23] P. Oechslin, Making a faster cryptanalytic time-memory trade-off, in *Advances in Cryptology—CRYPTO 2003*. LNCS, vol. 2729 (Springer, Berlin, 2003), pp. 617–630

[24] R. Oppliger, *Contemporary Cryptography* (Artech House, Boston, 2005)

[25] J.-J. Quisquater, J. Stern, Time-memory tradeoff revisited. Unpublished manuscript, December 1998

[26] N. Saran, Time memory trade off attack on symmetric ciphers. Ph.D. Thesis, Middle East Technical University, February 2009

[27] N. Saran, A. Doganaksoy, Choosing parameters to achieve a higher success rate for Hellman time memory trade off attack, in *2009 International Conference on Availability, Reliability and Security* (IEEE, New York, 2009), pp. 504–509

[28] F.-X. Standaert, G. Rouvroy, J.-J. Quisquater, J.-D. Legat, A time-memory tradeoff using distinguished points: New analysis & FPGA results, in *Cryptographic Hardware and Embedded Systems—CHES 2002*. LNCS, vol. 2523 (Springer, Berlin, 2003), pp. 593–609