A Colorful Ascent: Painting a New Rainbow Tables Variant

Gildas Avoine, Xavier Carpent, Diane Leblanc-Albarel

A Colorful Ascent: Painting a New Rainbow Tables Variant

Gildas Avoine[1†], Xavier Carpent[2†], Diane Leblanc-Albarel[3*†]

[1]SPICY group, INSA Rennes, IRISA, France.
[2] University of Nottingham, United Kingdom.
[3]COSIC group, KU Leuven, Belgium.

*Corresponding author(s). E-mail(s): diane.leblanc-albarel@kuleuven.be;
†These authors contributed equally to this work.

Abstract

The concept of a time-memory trade-off was first introduced by M. Hellman in 1980, providing an algorithm for conducting brute-force attacks more efficiently. This method consists of an intensive precomputation phase, the results of which are stored in tables and subsequently used to significantly reduce the time required for brute-force attacks. A notable advancement is the introduction of rainbow tables by Oechslin in 2003. However, the process of precomputing rainbow tables is marked by inefficiency, primarily due to the large proportion of computed values that are ultimately discarded. In 2023, descending stepped rainbow tables were introduced by Avoine et al., which consists of recycling chains during the precomputation phase. This paper introduces and evaluates ascending stepped rainbow tables, unlike traditional rainbow tables, which use uniform chains, this variant adds new, distinct chains during the precomputation phase. The paper presents a detailed analysis of the ascending stepped rainbow tables, including formulas to predict attack time, precomputation time, memory requirements, and coverage. Through theoretical results and implementation, the analysis demonstrates that this new variant offers significant improvements over both descending stepped rainbow tables and traditional rainbow tables for high coverage. Specifically, for the typical 99.5% coverage, ascending stepped rainbow tables achieve up to 30% faster precomputation time compared to descending stepped tables, and up to 45% compared to traditional rainbow tables, while also reducing attack times by up to 15% and 11% respectively. For lower coverages, although the precomputation times are higher, the attacks remain faster.

Keywords: Applied Cryptogrphy, Password Cracking, Time-Memory Trade-Off, Rainbow Tables, Descending Stepped Rainbow Tables

1 Introduction

Time-Memory Trade-Off (TMTO) algorithms aim to retrieve the preimage of an one-way function image more efficiently than exhaustive searches while requiring less memory than dictionary attacks. Rainbow Tables (RT) are a type of TMTO algorithm introduced by Oechslin in 2003 [22]. They are considered as one of the most effective TMTO algorithms [1, 2] and are widely used today [3–9].

Given a one-way function $h : A \to B$ (where A is an input space of $N = |A|$ elements), the goal of RTs is to retrieve an element $x \in A$ from its image Y with $Y = h(x)$. Although an exhaustive search can solve this problem, it is slow, requiring an average of $N/2$ operations to retrieve x. Dictionary attacks are faster, but they require too much memory (in order of N) for practical use cases. RTs allow the retrieval of x using a given memory M, with a cost of $T \propto N^2/M^2$ operations, which is much faster than an exhaustive search while requiring less memory than a dictionary attack.

RTs consist of two phases: a precomputation phase and an attack phase. During the precomputation phase, a "matrix" (a collection of hash chains) is computed and truncated. This phase is performed only once but is computationally expensive (with a lower bound of about $6N$ operations per table – see section 2). In practice, several tables (typically between 2 and 5) are generated to ensure a high success probability, increasing the precomputation time. Once the precomputation phase is completed, the attack phase can be performed quickly and repeatedly as desired using the generated tables. RTs are effective when (1) a large number of attacks must be performed, (2) the attack phase must be very fast but the attacker has time to prepare, or (3), the attacker can buy or download tables and only perform the attack.

Many improvements and variants of RTs have been proposed over the years [10–20]. Recently, a variant called *Stepped Rainbow Tables* has been introduced [20]. This variant performs better than vanilla RTs in both the precomputation and attack phases.

In this paper, we introduce a new variant called *Ascending Stepped Rainbow Tables* (ASRT) and show that in some cases, ASRT performs better than the original Stepped Rainbow Tables (which we refer to as *Descending Stepped Rainbow Tables* (DSRT) for clarity). Despite the similarity of their name, and to some extent of their structure, ASRT and DSRT vary quite dramatically in their analysis. In this paper, we explain in which cases ASRT performs better than DSRT, why, and discuss when it is more appropriate to use each variant. The paper presents the background on RTs and DSRT, introduces ASRT, compares ASRT and DSRT, and propose recommendations for their use.

2 Background on Rainbow Tables

2.1 Precomputation phase

The precomputation phase consists in computing a series of chains of hashes, called (rainbow) *matrices*. Once computed, only the first and last columns of each matrix are saved into *tables* for the attack phase.

2.1.1 Matrix Construction

In the precomputation phase, a matrix of $t+1$ columns and m_t rows is computed using elements from the search space A. The matrix elements are denoted as $x_{i,j}$ with $0 \leqslant i \leqslant t$ representing the column and $1 \leqslant j \leqslant m_t$ representing the row. Two types of functions are used to construct the matrix: the one-way function $h : A \to B$, and so-called *reduction functions* $r_i : B \to A$. The reduction functions are fast, mapping elements from the hash space B to A with uniform distribution. In contrast, the function h with $h : A \to B$ is considered slow and is the function that the algorithm aims to invert. The matrix element $x_{i+1,j}$ is obtained from $x_{i,j}$ (element in the same row, previous column) using $x_{i+1,j} = r_i(h(x_{i,j}))$. A *chain* depicts the collection of elements of the same row. Functions f_i with $f_i : A \to A$ are the composition functions of r_i and h such as $x_{i+1,j} = f_i(x_{i,j})$ and are called *hash-reduction* functions. Elements in the first column $x_{0,j}$ with j as $1 \leqslant j \leqslant m_0$ are chosen arbitrarily but must be different. They are called *start points* (SP). The elements in the last column of the matrix are called *end points* (EP). The combination of SP and EP is called a *table*.

2.1.2 Clean Rainbow Table

The word *collision* is used to describe a situation where $x_{i,j} = x_{i',j'}$. Many collisions occur between chains during matrix computations. When two different chains collide in different columns, they will remain distinct in subsequent columns because different reduction functions are applied. However, when two chains merge in the same column, they will be identical in all subsequent columns, constituting a so-called *merge*. A merge in column i between two chains in rows $j \neq j'$ occurs when $x_{i-1,j} \neq x_{i-1,j'}$ but $f(x_{i-1,j}) = f(x_{i-1,j'})$. As discussed in section 2.4, these merges significantly slow down the attack phase. As a result, only one among a set of merged chains is kept at the end of the precomputation phase. Discarding chains that have merged together to keep only one instance is called *cleaning*. A RT without merged chains is referred to as a *clean* RT[1]. In the following, RTs are considered clean.

2.1.3 Maximality

When using clean RTs, the number of elements in the last column of the cleaned matrix is less than the number of elements with which the precomputation begins. The number of elements considered at the start of the precomputation phase is denoted by m_0, while the number of elements in the last column after cleaning is denoted by m_t. The *surviving chains* in column i are the number of distinct elements in column i, i.e., the number of chains remaining if only the chains of the distinct elements in column i are kept. The average number of surviving chains in a column i is represented by m_i and can be calculated using the equation 1 from [21]:

$$m_i = \frac{2N}{i + \frac{2N}{m_0}}. \tag{1}$$

[1] Also known as *perfect* in some papers [1, 11, 21, 22]

To achieve the highest success probability, one can choose $m_0 = N$ elements at the beginning of the precomputation phase. In this case, a maximum of m_t^{max} elements will remain at the end of the phase, where $m_t^{\max}$ is given by equation 2 from [22]:

$$m_t^{\max} = \frac{2N}{t+2}. \tag{2}$$

A table generated using $m_0 = N$ elements is known as a *maximal* table. However, selecting $m_0 = N$ elements is impractical due to a very high precomputation time. Instead, fewer start points are usually considered, and it is useful to express this number as $m_0 = rm_t^{max}$, resulting in αm_t^{max} elements remaining at the end of the precomputation phase, where α is given by equation 3 introduced in [18]:

$$\alpha = \frac{r}{r+1}. \tag{3}$$

The parameter α is referred to as the *maximality factor*, and characterizes how far a table is from being maximal. For instance, taking $r = 20$ (corresponding to $\alpha \simeq 0.95$) allows to drastically reduce the precomputation time [18] while not significantly impacting the attack time or the success probability.

2.1.4 Precomputation Time

In 2021, [18] proposed the *filtration* method, which consists in cleaning the matrix several times during precomputation, instead of cleaning a single time in the final column. While cleaning in every columns reduces the total number of required hashes, it increases the overhead in the form of interruption of row computations during cleaning. In [18], the authors show that only several dozen of cleaning closely approximates the theoretical minimum precomputation time, while minimizing the interuptions and overhead due to the cleaning itself.

In order to compare the precomputation time of vanilla RTs with those of DSRT and ASRT, we choose to instead use precomputation lower bound as our comparison criterion. The rationale for this choice is explained in section 5. Throughout the paper, unless otherwise stated, we will use the term *time* to refer to the number of hash (h) operations for a particular action. The minimum precomputation time of a RT has been established and demonstrated in [18] and is presented in equation 4:

$$P_{\min} = \sum_{i=0}^{t-1} m_i \approx 2N \ln(1+r). \tag{4}$$

2.2 Success Probability

In a RT attack, the success probability depends on the number of distinct elements in the matrix. The success probability of a single RT is provided in [22] and is given by Equation 5:

$$p = 1 - \left(1 - \frac{m_t}{N}\right)^t. \tag{5}$$

When coupled with Eq. (2), this shows that the maximal coverage of a clean RT is $1 - e^{-2} \approx 86\%$. As mentioned in section 1, the use of multiple tables increases the success probability beyond the limit of 86%. Specifically, when ℓ tables are used, the success probability of the attack is given by Equation 6, also introduced in [22]:

$$p_\ell = 1 - (1 - p)^\ell. \tag{6}$$

2.3 Memory Used

In [17], a method called *compressed delta encoding* is introduced for storing RTs. This method achieves a memory usage very close to the theoretical lower bound, with a difference of only approximately 0.66%. Consequently, for simplicity, we approximate the memory used to store a single RT, denoted as M^{RT}, using the lower bound introduced in [17]. This lower bound is provided in equation 7.

Since only the SP and EP are required for the attack, only the memory required to store the SP (M_{sp}^{RT}) and the memory required to store the EP (M_{ep}^{RT}) needs to be considered. The total memory used to store a RT, M^{RT}, is the sum of M_{sp}^{RT} and M_{ep}^{RT}.

$$\begin{aligned} M^{RT} &= M_{sp}^{RT} + M_{ep}^{RT} \\ &= \ell \left[m_t \lceil \log_2(m_0) \rceil + \log_2 \binom{N}{m_t} \right]. \end{aligned} \tag{7}$$

2.4 Attack Phase

The aim of the attack phase is to retrieve x from its hash value $Y = h(x)$. The attacker begins by assuming that the target x is in the penultimate column of the rainbow matrix, and tests this hypothesis. The process of assuming x is in a particular column and testing the assumption is referred to as a *search*. If x is not found in the penultimate column, the attacker iteratively searches previous columns until x is found or all columns have been searched.

To perform a search in a column c_i, the attacker computes the *attack chain* which is a chain starting by $r_i(Y)$ and finishing in column t. More formally the attack chain is $Z = f_t(f_{t-1}(...(f_{i+1}(r_i(Y)))))$. Once this chain computed, the attacker checks if Z matches any of EPs stored in the RT. If there is a match (with j the row of the matched EP), the attacker computes a chain starting from $x_{0,j}$ and ending at $x_{c_i,j}$. The attacker then computes $h(x_{c_i-1,j})$, if $h(x_{c_i-1,j}) = Y$, then $x_{c_i-1,j}$ is the desired value.

Because of collisions, a *false alarm* can occur if there is a match between Z and an EP despite the fact that $h(x_{c_i-1,j}) \neq Y$. In this case, the attacker continues the search in other columns until either all columns have been searched or the correct x is found.

It is worth noting that when multiple RTs are used, it is faster to search through the columns of each table in parallel rather than searching through each table sequentially, because the cost of a searches increases as it gets deeper.

The average number of hash operations required to search through a single column is given by Proposition 1 from [20]:

Proposition 1. *For a given column c, the average number of hash operations C_c needed to perform a search is given by:*

$$C_c = t - c \prod_{i=c}^{t} \left(1 - \frac{m_i}{N}\right).$$

Given the cost of searching through a single column, the average total time required to perform an attack using ℓ tables is given by Theorem 2. This theorem is introduced and proven in [21]. Intuitively, it corresponds to the sum of the cost of a search in column i weighted by the probability that the search stops there, plus the cost of the fail case.

Theorem 2. *Given a search space of size N, the average number of hash operations T required to perform an attack using ℓ RTs with $t+1$ columns, is:*

$$T = \ell \sum_{c=1}^{t} \left(\frac{m_t}{N} \left(1 - \frac{m_t}{N}\right)^{\ell(c-1)} \sum_{j=1}^{c} C_{t-j+1} \right) + \ell \left(1 - \frac{m_t}{N}\right)^{t} \sum_{c=1}^{t} C_c.$$

3 Background on Descending Stepped Rainbow Tables

Stepped Rainbow Tables have been introduced in 2023 in [20]. But they are referred to as Descending Stepped Rainbow Tables (DSRTs) in this paper for distinction with ASRTs. This choice is made to easily distinguish DSRTs and ASRTs. In this section we first provide an overview of DSRTs, followed by an explanation of the precomputation phase, its cost, the memory required to store DSRTs, their success probability, and finally, the attack phase and formulas to evaluate their running time.

3.1 Overview

During the cleaning of RTs, a large proportion of computed chains are discarded as a result of merges. DSRTs instead recycle some merged chains that would otherwise be discarded. These recycled chains are shorter than regular chains because they exclude the merged portions. As a result, tables generated using this approach are composed of several *steps*, giving the technique its name. A step $s_i < t$, is a column in which chains are cut. The collection of chains ending in step s_i have a length of s_i.

The parameters of DSRTs are the total number of steps τ, the steps themselves $\boldsymbol{s}$ with $s = \{s_1, s_2, ..s_\tau\}$, the maximality factor α, the total number of columns t and the number of tables ℓ. The first two are unique for DSRTs, while the others are inherited from RTs.

3.2 Precomputation

The precomputation begins by computing the m_0 chains from column 0 to the first step s_1, then a filtration is performed in s_1, a copy of the m_{s_1} remaining chains is put aside and the precomputation continues until reaching step s_2. A new filtration

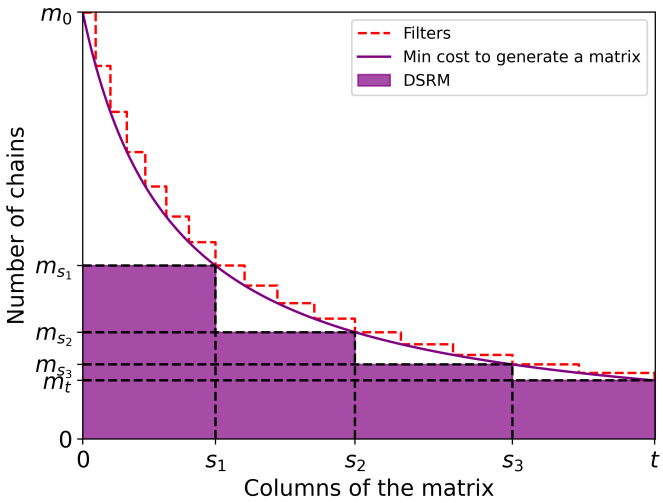


Figure 1: Construction of a DSRT with 3 steps.

is performed and instead of discarding chains that have merged between s_1 and s_2, those chains are kept with a size s_1. A copy of the m_{s_2} remaining chains is put aside then the computation continues until reaching s_3 and so on until reaching column t.

The minimum DSRT precomputation time is the same as for RT, see equation 4 from [18, 20].

Figure 1 illustrates the construction of a DSRT with 3 steps . The area under the dashed red line represents the precomputation cost for building the DSRT using the filtration method (each landing of the curve corresponds to the application of a filter). The solid purple line is the number of chains that remain after a cleaning for each column (the area under which corresponds to the minimum precomputation time) . The matrix obtained at the end of precomputation phase is the solid purple area.

3.3 Success Probability

The success probability of a DSRT is computed similarly to that of RTs, with the difference that some chains are shorter, which must be taken into account when computing the success probability.

The success probability for a single DSRT is given in theorem 3 from [20].

Theorem 3. *Given τ steps noted s_i with $0 < i \leqslant \tau$, $s_0 = 0$ and $s_{\tau+1} = t$, and considering m_{s_i} the number of surviving chains in column s_i, the success probability p of a single clean DSRT is:*

$$p = 1 - \prod_{i=1}^{\tau+1} \left(1 - \frac{m_{s_i}}{N}\right)^{s_i - s_{i-1}}.$$

The success probability using ℓ tables is then obtained the same way as RT, using equation 6.

3.4 Memory Used

The total memory lower bound used to store DSRT, M^{DSRT}, is computed as the sum of the memory used to store the SPs, M_{sp}^{DSRT} (as defined in equation 8) with the memory used to store the EPs, M_{ep}^{DSRT}, (as defined in equation 9). The difference with RT is that EPs compression cannot be performed on all EPs due to varying chain lengths[2]. Therefore, the total EPs memory is computed as the sum of memory used to store each step. The total memory required for storing SPs and EPs is given in equation 10 and is proved in [20].

$$M_{sp}^{DSRT} = M_{sp}^{RT} = \ell m_{s_1} \lceil \log_2(m_0) \rceil. \tag{8}$$

$$M_{ep}^{DSRT} = \ell \left(\log_2 \binom{N}{m_t} + \sum_{i=1}^{\tau} \log_2 \binom{N}{m_{s_i} - m_{s_{i+1}}} \right). \tag{9}$$

$$M^{DSRT} = M_{sp}^{DSRT} + M_{ep}^{DSRT}. \tag{10}$$

3.5 Attack Phase

In this section, we first describe the attack process and introduce definitions to characterize the values involved in attack time evaluation. We then provide an explanation of how to evaluate the attack time, along with the corresponding formulas. Proofs of propositions and theorems included therein are given in [20].

3.5.1 Attack Process

In contrast to vanilla RT, the DSRT attack is not carried out by searching linearly from right to left in the matrix. Indeed, while the cost does increase with the size of the attack chain (just as it does in RTs), the probability of successful search does not remain constant (contrarily to RTs). As a result, a metric μ is used to evaluate the success probability of a search in a given column per average number of operations require to perform this search. This success probability over cost is computed for all columns of the table, and the resulting values are sorted in decreasing order in a vector $\mathbf{v}$. Hence, vector $\mathbf{v}$ represents the optimal order in which the columns should be searched during the attack. The values of μ and $\mathbf{v}$ are formally defined in definitions 2 and 3, respectively.

Once the order of search has been computed, the attack begins by searching in the column c that maximizes μ. If $c \geqslant s_\tau$, the chain starting by $r_c(Y)$ is built from column $c+1$ to column t, and the attacker searches for a match with the stored EPs. If an alarm is raised, the attacker builds the attack chain from column 0 to column c and checks if the element in column c matches the search element (i.e., if $Y = h(x_{c,j})$, with j the row of the matched EP). If not, the attack proceeds to the next columns

[2] In addition, information on the steps in which chains are needs to be stored

c chosen usin μ. If $c < s_\tau$, the chain starting by $r_c(Y)$ is first built until reaching the leftmost step that is to the right of c. We note this step by $s_k(c)$, as defined in Definition **??**. A check for a match with the EP in column $s_{k(c)}$ is then performed.

If a match is found, the attack chain is rebuilt as usual. In case of a false alarm, no EP match is searched in the other steps or in column t since, by construction, it would lead to another false alarm.

If no match in column $s_{k(c)}$ is found, the chain is computed until $s_{k(c)+1}$ or until reaching column t, and the same process is repeated.

3.5.2 Evaluation of the Attack Time

To evaluate the average attack time, three different events are considered (1) *no alarm* (i.e., no match with any EPs), (2) *true alarm* (i.e., success of the search), and (3) *false alarm* (the attack chain match an EP but does not allow to retrieve the searched element). Each has a distinct probability of occurrence and associated cost. The time for a search is the weighted average of these.

Firstly, definitions and lemmas that are used to characterize DSRTs are introduced. Then the probability and cost of each event is given. The cost of a search C_c, in column c is obtained by summing up the cost of each event multiplied by its probability, which is given by Theorem 9. Finally, this cost is used to compute the average time of the attack in the whole DSRT, as presented in Theorem 10.

3.5.3 DSRT Characterization

The index of the leftmost step to the right of a given column c is noted $k(c)$, with thus $s_{k(c)-1} \leqslant c < s_{k(c)}$.

Given a column c, the proportion of chains ending in a particular step s_i with $c < s_i$ must to be known. This proportion is called $\rho_{s_i,k(c)}$ and is defined in Definition 1.

Definition 1. *Given a column c and a step s_i, the proportion of chains with length s_i in column c is given by $\rho_{i,k(c)}$ with $\rho_{i,j}$ such that:*

$$\rho_{i,j} = \begin{cases} \frac{m_{s_i} - m_{s_{i+1}}}{m_{s_j}} & i \leqslant \tau \\ \frac{m_t}{m_{s_j}} & i = \tau + 1 \end{cases}$$

where m_{s_i} and $m_{s_{i+1}}$ are the number of chains with length s_i and s_{i+1}, respectively.

The metric $\mu(c)$ is used to evaluate the ratio of the probability of finding the searched element x in column c over the cost of a search in this column c. It is defined in Definition 2.

Definition 2. *Given N and a column c with $0 \leqslant c \leqslant t$ and a cost of the in column c, C_c:*

$$\mu(c) = \frac{m_{s_{k(c)}}}{NC_c}.$$

C_c (the average cost of a search in column c) for DSRT is formally defined in theorem 9.

The vector $\mathbf{v}$, defined in Definition 3, is used to determine the order of search in the table. It is such that $\mu(v_i) > \mu(v_j)$ for $0 \leq i < j \leq t$.

Definition 3. *Given a DSRT table with t columns, the vector* $\mathbf{v}$ *with* $\mathbf{v}= [v_1, v_2, \ldots, v_t]$ *is obtained by sorting the columns* $[1, \ldots, t]$ *in decreasing order of* $\mu(c)$.

The probability $p_{\text{nomrg}}(c, c')$ represents the probability that an attack chain does not merge with any chains of a DSRT between column c and c'. It is given in Lemma 1.

Lemma 1. *Given two columns c and c' with $c < c' \leqslant t$, the probability that the attack chain does not merge with any chains of the rainbow matrix by c', given it had not merged in or before c, is:*

$$p_{nomrg}(c, c') = \prod_{i=c+1}^{c'} \left(1 - \frac{m_i}{N}\right). \tag{11}$$

3.5.4 No Alarm

When searching in a column c that does not raise any alarm, the cost of a search is $t - c$. The attacker computes the chain from column c to t even when $c < s_\tau$.

The probability of no alarm occurring between column c and column t is given in Proposition 4.

Proposition 4. *The probability that no alarm occurs between a column c and column t is*

$$p_{noalarm}(c) = p_{nomrg}(c, t).$$

3.5.5 True Alarm

When searching in a column c, the success probability, i.e., the probability of raising a true alarm, is given by Proposition 5. Its cost is given in Proposition 6. This cost corresponds to the sum of the probability of raising a true alarm in each step to the right of c, multiplied by the cost of dealing with these true alarms.

Proposition 5. *The probability that a true alarm occurs when starting the attack chain in c is:*

$$p_{find}(c) = \frac{m_{s_{k(c)}}}{N}.$$

Proposition 6. *Given a search performed in a column c and $k(c)$ the index of the leftmost step that is to the right of c, the number of hash operations needed to rule out an alarm is:*

$$\sum_{i=k(c)}^{\tau+1} \rho_{i,k(c)} s_i.$$

3.5.6 False Alarm

The probability of raising a false alarm during the search process depends on the column in which the merge that leads to the false alarm occurs. Proposition 7 gives the probability of raising a false alarm when searching in column c, when the merge occurred between columns c and $k(c)$.

Proposition 7. *The probability to raise a false alarm due to a merge between columns c and $s_{k(c)}$ is*

$$p_{fa\text{-}pre}(c) = 1 - p_{find}(c) - p_{nomrg}(c, s_{k(c)}).$$

Additionally, Proposition 8 gives the probability of raising a false alarm when searching in column c, when the merge occurred between columns s_i and s_j, with $c \leqslant s_{k(c)} < s_i < s_j$.

Proposition 8. *The probability to raise a false alarm due to a merge between columns s_i an s_j, with $c \leqslant s_{k(c)} < s_i < s_j$, is:*

$$p_{fa\text{-}post}(c, s_i, s_j) = p_{nomrg}(c, s_i) - p_{nomrg}(c, s_j).$$

The cost of a false alarm is the same as a true alarm and is therefore given by Proposition 6.

3.5.7 Cost of a Single Search

The average cost of a search in a column c is noted C_c, and is the sum of the cost of each event described bellow weighted by its probability. It should be noted that if $s_\tau < c \leqslant t$, then the cost is the same as with RTs.

The cost required to perform a search in a column c is given by Theorem 9.

Theorem 9. *For a given column c and the index $k(c)$, the average number of cryptographic operations C_c needed to perform a search is:*
For $s_\tau < c \leqslant t$:

$$C_c = t - cp_{noalarm}(c).$$

For $c \leqslant s_\tau$:

$$\begin{aligned} C_c = \left(1 - p_{nomrg}(c, s_{k(c)})\right) & \sum_{i=k(c)}^{\tau+1} \rho_{i,k(c)} s_i \\ + & \sum_{j=k(c)}^{\tau} p_{fa\text{-}post}(c, s_j, s_{j+1}) \sum_{i=j+1}^{\tau+1} \rho_{i,j+1} s_i \\ + & \left(t - c\right) p_{noalarm}(c). \end{aligned}$$

3.5.8 Average Attack Time

The average attack time using ℓ DSRTs is given in Theorem 10. This attack time is computed in two parts.

The first part is the sum of the probability of finding the searched element in a given column, given that it has not been found yet. This sum is multiplied by the total cost of searching in all columns.

The second part is the probability of not finding the searched element in any of the tables. This probability is multiplied by the total cost of searching in all tables.

Theorem 10. *Given N, ℓ DSRT with τ steps, and considering its vector $v = [v_1, v_2, ...v_t]$ ordering the columns of tables, the average number of hash operations T required to perform an attack is:*

$$T = \ell \sum_{c=1}^{t} \left(\frac{m_{v_c}}{N} \prod_{i=1}^{c-1} \left(1 - \frac{m_{v_i}}{N}\right) \sum_{j=1}^{c} C_{v_{t-j+1}} \right) + \ell \prod_{i=1}^{t} \left(1 - \frac{m_{v_i}}{N}\right) \sum_{c=1}^{t} C_{v_c}.$$

with m_{v_c} the number of points in column c, and with $s_{v_{c-1}} \leqslant c < s_{v_c}$.

4 Ascending Stepped Rainbow Tables

This section introduces the Ascending Stepped Rainbow Tables (ASRT) variant, which differs in behavior and efficiency from the DSRT. Although the ASRT shares some similarities with the DSRT and is in many ways its dual. ASRTs are introduced in this section.

4.1 Overview

The ASRT algorithm involves a gradual *addition* of chains to the matrix during the precomputation phase. In well chosen columns of the matrix called *steps*, a cleaning is performed, and new chains are added to the matrix. These chains do not begin in column 0, but rather in some columns s_i where $0 < s_i < t$. As a result, the matrix consists of chains that *start* in different columns but all *end* in the last column. The Ascending Stepped Rainbow Matrix (ASRM) can be cleaned in the usual way since the added chains are computed using the same hash-reduction functions as those already present in the matrix.

The addition of chains to the matrix in given columns results in an increase in both the precomputation and memory cost. However, this leads to a matrix with more chains present in the right part of the table, including in particular, more shorter chains. The purpose is thus to reduce the attack time by utilizing these shorter chains. Moreover, a higher number of chains also means a higher success probability.

In DSRT and RT, m_i (defined in Equation 1) represents the number of unique elements in column i. However, this notation is not suitable for ASRT since not all chains start in the same column. Therefore, we introduce $m_{i,j}$, which represents the *number of unique elements in column j that are part of chains starting in column i or before.*

4.2 Precomputation

4.2.1 Generation

Differing from RT and DSRT, the initial number of SP used in ASRT precomputation is referred to as $m_{0,0}$. Accordingly, $m_{0,0}$ chains are computed from column 0 up to the first step, denoted by s_1, and then filtered to retain m_{0,s_1} chains.

Next, a value of m_{s_1,s_1}, greater than m_{0,s_1}, is chosen (refer to section 4.2.3 for the formal definition of the m_{s_1,s_1} value). Then, $m_{s_1,s_1} - m_{0,s_1}$ elements are chosen to be as small as possible[3] and not equal to any of the m_{0,s_1} elements that remained in column s_1.

At this point, m_{0,s_1} chains with length s_1 remain from the first part of the precomputation, while $m_{s_1,s_1} - m_{0,s_1}$ additional elements have been selected. In total, m_{s_1,s_1} elements are present in column s_1. The $m_{s_1,s_1} - m_{0,s_1}$ elements are the SP of chains starting in s_1.

[3] This choice is justified to minimize the total memory.

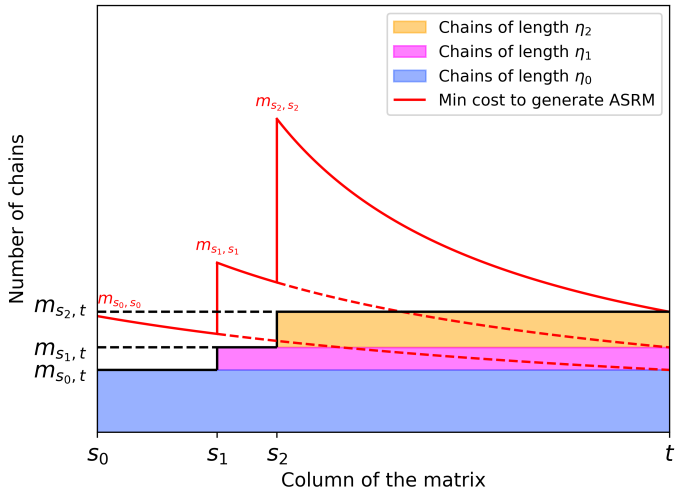


Figure 2: ASRM with 2 steps.

The computation of chains then continues by computing the next columns of the m_{s_1,s_1} chains, regardless of where they started. This computation continues until reaching column s_2, where another cleaning is performed.

Whenever chains starting in different columns merge, the longer chain is always retained. Intuitively, keeping a shorter chain decreases the success probability of the table too much compare to the gain in attack time and is, therefore, not worth it.

After cleaning in column s_2, m_{s_1,s_2} chains remain. As previously, $m_{s_2,s_2} - m_{s_1,s_2}$ elements are added to the m_{s_1,s_2} that remain from the previous part. The computation of chains starts again with m_{s_2,s_2} chains.

This continue for the τ steps chosen and until reaching the last column of the ASRT t.

As for DSRT, the total number of steps is denoted by τ, these steps are denoted $\{s_1, s_2, ..., s_\tau\}$, with by convention $s_0 = 0$ and $s_{\tau+1} = t$. The choice of the steps placement is explained in section 5.

Figure 2 represents an ASRM with 2 steps and noteworthy points represented.

4.2.2 Maximality

Similarly to RT and DSRT, one could set $m_{0,0} = N$, $m_{s_1,s_1} = N$,.., $m_{s_\tau,s_\tau} = N$ to achieve a maximal ASRT. However, this choice would result in a significant increase in precomputation cost for comparatively little benefit. Therefore, only $m_{s_i,s_i} < N$ with $i \in \{0, 1, ..., \tau\}$ are selected.

To determine m_{s_i,s_i}, a maximality factor α_i is chosen for each step, with $0 < \alpha_i < 1$ and $i \in \{0, 1, ..., \tau\}$.

Unlike RT and DSRT, ASRT has multiple maximality factors (one per step).

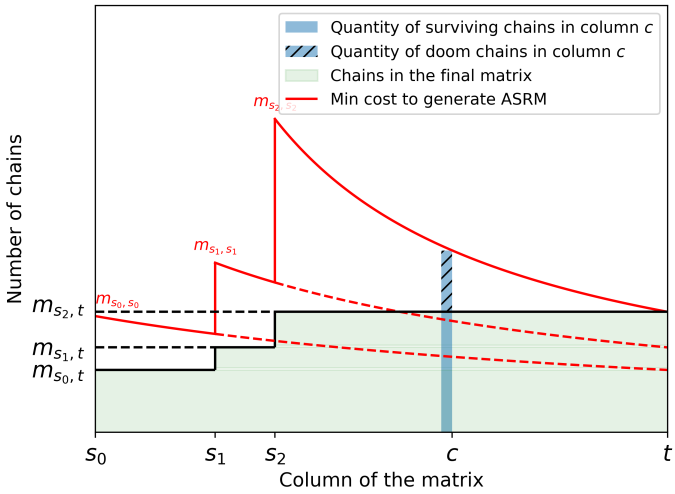


Figure 3: ASRM with 2 steps.

Given an α_i and $m^{max}_{t-s_i}$, the value of m_{s_i,s_i} is obtained from Equation 12:

$$m_{s_i,s_i} = \frac{\alpha_i}{1-\alpha_i} m^{max}_{t-s_i} \tag{12}$$

where $m^{max}_{t-s_i}$, obtained from Equation 2, is the maximum number of chains that started in column s_i and remain in column t.

4.2.3 Characterization

To facilitate the characterization and visualization of ASRT, and to simplify equations, this section provides notations and notions utilized for characterizing ASRT. Figure 3 illustrates these notions on an ASRM. The introduced notions are then used throughout the rest of this paper.

Firstly, η_i denotes the number of elements contained in a chain starting at s_i. As $t - s_i + 1$ columns are present between s_i and t, $\eta_i = t - s_i + 1$. We note ψ_{c,s_i} the number of columns between s_i and column c if $c > s_i$, and 0 if $c \leqslant s_i$, resulting in $\psi_{c,s_i} = \max(c - s_i + 1, 0)$.

With ASRT, it is essential to differentiate chains that are part of the final matrix (and thus present in the final table) from those that were present at a given column during precomputation but not in the final matrix (due to merges). Given a column c and a starting column s_i, the *unique chains* are chains that begin in column s_i or earlier and, during the precomputation phase, remain after cleaning in column c. These chains may or may not be present in the final matrix. The number of unique chains in column c starting at s_i or earlier is denoted by $m_{s_i,c}$ and is given by Equation 13, which is derived from Equation 1.

$$m_{s_i,c} = \begin{cases} \frac{2N}{\psi_{c,s_i} + \frac{2N}{m_{s_i,s_i}}} & c \geqslant s_i \\ 0 & c < s_i \end{cases} \tag{13}$$

Among the unique chains in column c, some will merge and, consequently, will not be present in the final matrix. Given a column c and a step s_i, the *faulty chains* are chains that begin in column s_i or earlier and, during the precomputation phase, remain after cleaning in column c but do not belong to the final matrix.

The number of faulty chains in column c starting at s_i or earlier is denoted by $m^f_{s_i,c}$ and is given by Equation 14, which is derived from Equation 13.

$$m^f_{s_i,c} = \begin{cases} m_{s_i,c} - m_{s_i,t} & c \geqslant s_i \\ 0 & c < s_i \end{cases} \tag{14}$$

Moreover, to effectively utilize ASRT, it is often necessary to determine the closest step to the left of a given column c. The index of this closest step is denoted by $k^A(c)$ and is defined in Definition 4.

Definition 4. *We note $k^A(c)$ the index of the rightmost step that is to the left of column c, i.e., $s_{k^A(c)} \leqslant c < s_{k^A(c)-1}$ and with $k^A(c) \in \{0, .., \tau\}$.*

4.2.4 Precomputation Time

If the cleaning method is not used , the precomputation time of an ASRM is given by multiplying for each step, the m_{s_i,s_i} SPs considered, by the number of columns until the next step. The maximum precomputation time P_{max} can then be obtained by summing up the product of m_{s_i,s_i} and $(s_{i+1} - s_i)$, as shown in Equation 15:

$$P_{max} = \sum_{i=0}^{\tau} m_{s_i,s_i}(s_{i+1} - s_i) \tag{15}$$

However, as with RT and DSRT, filtering can significantly reduce the precomputation time by reducing the number of hash computations required. The minimum theoretical precomputation time for a given ASRM can be achieved when a filter is used in every column. The minimum number of operations needed to generate the matrix is obtained by summing every $m_{s_{k^A(c)},c}$ values for $c \in 1, .., t$, as shown in Equation 16.

$$P_{min} = \sum_{c=0}^{t-1} m_{s_{k^A(c)},c} \tag{16}$$

4.3 Success Probability

The success probability for a single ASRT is given by Theorem 11. Similarly to DSRT and RT, the success probability for ℓ ASRTs is obtained by applying Equation 6. As for RT and DSRT, the intuitive success probability is the number of distinct elements in the ASRM. Therefore, the success probability is obtained by counting the number of unique chains in each column of the final matrix, which is divided by the total number of possible elements N.

Theorem 11. *Given a single ASRT wih τ steps and $t+1$ columns, the success probability for this single ASRT is given by:*

$$P = 1 - \prod_{i=0}^{\tau} \left(1 - \frac{m_{s_i,t}}{N}\right)^{s_{i+1}-si}$$

Proof. In each column c, there are $m_{s_i,t}$ different elements present. The probability of finding the target element in a given column is $\frac{m_{s_i,t}}{N}$. The probability of not finding the target element in a column c is therefore $1 - \frac{m_{s_i,t}}{N}$.

Between each step, there are $s_{i+1} - s_i$ columns. Thus, the probability of not finding the target element in any column between s_i and s_{i+1} is

$$\left(1 - \frac{m_{s_i,t}}{N}\right)^{s_{i+1}-si}.$$

As there are τ steps plus the main table, the probability of not finding the target element in the entire ASRT is

$$\prod_{i=0}^{\tau} \left(1 - \frac{m_{s_i,t}}{N}\right)^{s_{i+1}-si}.$$

□

4.4 Memory

The storage approach for ASRT is similar to RT and DSRT with a slight difference in the computation of the SP lower bound. Like RT and DSRT, we use the memory lower bound to compare each variant since the available storage methods [17] are very close to this lower bound and simplifies the analysis. The total memory used by ASRT, denoted by M^{ASRT}, is the sum of the memory used for storing the SP and the memory required to store the EP, denoted respectively as M_{sp}^{ASRT} and M_{ep}^{DSRT}.

The method for storing EPs is the same as for DSRT, where each collection of chains starting in a given step is compressed and stored separately from chains starting in other steps. The memory used to store EPs is given by Equation 17, which is adapted from [17] and is the same as for DSRT introduced in [20].

$$M_{ep}^{ASRT} = \ell \left(\log_2 \binom{N}{m_{0,t}} + \sum_{i=1}^{\tau} \log_2 \binom{N}{m_{s_i,t} - m_{s_{i-1},t}} \right). \tag{17}$$

For SPs, unlike RT and DSRT, the number of SP varies depending to in which step the corresponding chains start. Hence, the formula used for SP needs to be adapted to take this into account.

At each step s_i, there are m_{s_i,s_i} possible SPs to considered. Thus, the minimal naive way to store SP is to store $(m_{s_i,t} - m_{s_{i-1},t}) \log_2(m_{s_i,s_i})$ for each step. This is because there are $m_{s_i,t} - m_{s_{i-1},t}$ elements to store for each step, and for each step, there are m_{s_i,s_i} possible SPs.

However, when considering ASRT, among the m_{s_i,s_i} possible elements, m_{s_{i-1},s_i} are already part of the chains computed previously and cannot be chosen. Each chain among these m_{s_{i-1},s_i} only have very low probability to be among the m_{s_i,s_i} elements considered in step s_i, but there will nonetheless be a small amount of redundancy. To model this, we can compute the value $m^u(i, z_p)$, which depicts the maximum number of possible elements among the m_{s_{i-1},s_i} elements that are among the m_{s_i,s_i} with a probability of $1-p$ that more elements than $m^u(i, z_p)$ will be among the m_{s_i,s_i} elements.

Here, z_p is the quantile function of the standard normal distribution. In our experiments, we have chosen $p = 0.99994$, i.e., $z_p = 4$ to ensure that the probability that more elements than $m^u(i, z_p)$ (defined in Proposition 12) will be among the m_{s_i,s_i} elements is less than 0.0007.

Proposition 12. *Given a step s_i with m_{s_i,s_i} SPs, given m_{s_{i-1},s_i} elements of chains starting in s_{i-1} and remaining in s_i after cleaning. $m^u(i, z_p)$ is the maximum number of elements from m_{s_{i-1},s_i} that are in the m_{s_i,s_i} elements with a probability $1-p$ with:*

$$m^u(i, z_p) = m_{s_{i-1},s_i} - \left(m_{s_{i-1},s_i} \frac{m_{s_i,s_i}}{N} + z_p \times \sqrt{m_{s_{i-1},s_i} \frac{m_{s_i,s_i}(1 - m_{s_i,s_i})}{N^2}} \right)$$

Proof. The probability that an element from the m_{s_{i-1},s_i} elements of the previous chains is part of one of the m_{s_i,s_i} elements to choose is $\frac{m_{s_i,s_i}}{N}$. This is because the hash-reduction function is considered random, and thus the m_{s_{i-1},s_i} elements are considered as m_{s_{i-1},s_i} random elements in N. The number of m_{s_{i-1},s_i} elements that are part of the m_{s_i,s_i} elements to choose thus follows a binomial distribution with parameters $p = \frac{m_{s_i,s_i}}{N}$ and $n = m_{s_{i-1},s_i}$. The expected value E of this distribution is thus given by $E = np = m_{s_{i-1},s_i} \frac{m_{s_i,s_i}}{N}$, and the standard deviation σ is given by

$$\sigma = \sqrt{p(1-p)n} = \sqrt{m_{s_{i-1},s_i} \frac{m_{s_i,s_i}(1 - m_{s_i,s_i})}{N^2}}$$

From this, we obtain $m^u(i, z_p)$ by adding E with $z_p\sigma$ and subtracting this sum from $n = m_{s_{i-1},s_i}$. □

According to Proposition 12, the maximum number of possible SPs to store for step s_i with $1 \leqslant i \leqslant \tau$ is given by $m_{s_i,s_i} - m^u(i, z_p)$. The total memory required to store all the SP can be computed using Equation 18 which is derived directly from [17].

$$M_{sp}^{ASRT} = \ell m_{s_0,t} \lceil \log_2(m_{s_0,s_0}) \rceil + \ell \left(\sum_{i=1}^{\tau} m_{s_i,t} \lceil \log_2(m_{s_i,s_i} - m^u(i, z_p)) \rceil \right). \quad (18)$$

By taking $z_p = 4$, the probability that the memory for SPs is larger than the one given by Equation 18 is less than 0.0007%.

The total memory used by ASRT, M^{ASRT}, is then obtained by adding M_{sp}^{ASRT} and M_{ep}^{ASRT}, and is given in Equation 19.

$$M^{ASRT} = M_{sp}^{ASRT} + M_{ep}^{ASRT} \tag{19}$$

4.5 Attack Phase

4.5.1 Attack Process

The attack using ASRTs is similar to the attack using vanilla RT. Contrary to DSRT, it is not required to define any metric for choosing the column of the search. By construction, the most advantageous columns in which to perform a search are the rightmost ones, as they are the columns for which the price to build a chain until column t is the lowest. In addition, they are the columns with the higher success probability since they are the column with the higher number of chains. They are also the columns that contain the shortest chains. All these reasons make the search to have a monotonically increasing cost as the online chains get longer.

Thus, the attack begins by assuming that the target element is in the second to last column. The attacker computes $R_t(Y)$ and checks if the result is one of the EPs stored.

If this is the case, the attacker builds the attack chain from the corresponding SP (regardless of the column in which the chain starts) to column $t-1$ and thus obtains $x_{t-1,j}$ with j the rows of the matched EP. The attacker computes $h(x_{t-1,j})$ and if $h(x_{t-1,j}) = Y$ the attack ends and the target element is x_{t-1}. If $h(x_{t-1,j}) \neq Y$ it is a false alarm and the attack continues in the next left columns until finding the target element or reaching the end of the table.

4.5.2 Roadmap

In the next sections, we introduce propositions used to characterize the average cost of the attack. The average attack time is the sum of the average cost of the search in each column multiplied by each column's probability that a search in that column occurs. Thus, we first need to define the cost and probability of a search in a given column c.

The search in a given column c is equal to the sum of the cost of each possible event multiplied by their respective probability of occurrence.

We thus, firstly define each event, namely *no alarm*, *false alarm*, and *true alarm*. We define the cost of each event and their probability of occurrence. We then multiply each cost of event by its probability of occurrence and sum the whole to obtain the cost of search in any column of the table.

4.5.3 No Alarm

Cost

In the process of searching for a match in column c, if the event of *no alarm* occurs, it indicates that there is no match with any EP in the table. In such a scenario, the cost associated with this event is equivalent to the construction of a chain from column c to t, which is equal to $t - c$.

Probability

If no match occurs with any EP in the table, it implies that the attack chain does not merge with any chain of the matrix. More formally, for all columns i between column c and t, the elements of the attack chain are not present in any of the $m_{s_{k^A(i)},i}$ elements of the matrix in column i. This corresponds to the fundamental results of Equation (3) in [22], which is generalized in Lemma 2. Instead of m_i elements, $m_{s_{k^A(i)},i}$ surviving chains are present in column i, in the ASRT variant.

Lemma 2. *Given a column c with $c < t$, the probability that the attack chain does not merge with any chain of the rainbow matrix by t, given it had not merged in or before c, is:*

$$p^A_{noalarm}(c) = \prod_{i=c+1}^{t} \left(1 - \frac{m_{s_{k^A(i)},i}}{N}\right).$$

4.5.4 True Alarm

Cost

When searching for an element in column c, a *true alarm* occurs when the searched element is found in that column.

The cost associated with this search depends on the column in which the searched element is found. Since the attack chain needs to be built from column c to t and then from the corresponding SP to column c, the cost of the search is the length of the chains in which the corresponding SP is found.

To evaluate the cost of the search, Definition 5 introduces $\rho^A_{i,j}$, which represents the proportion of chains with a length of s_i in column j. For instance, consider an ASRT with two steps in columns s_1 and s_2 with $s_1 < s_2 < t$. In column c with $s_2 < c < t$, chains can be of length s_1, s_2, or t. $\rho^A_{1,k^A(c)}$ gives the proportion of chains in column c that have a length of s_1. Definition 5 generalizes this concept for all steps and columns.

Definition 5. *Given a step s_i and a step s_j, the proportion of chains with length s_i in a column c with $s_j < c < s_{j+1}$ is given by $\rho^A_{i,j}$ such that:*

$$\rho^A_{i,j} = \begin{cases} \frac{m_{s_i,t} - m_{s_{i-1},t}}{m_{s_j,t}} & i > 0 \\ \frac{m_{s_0,t}}{m_{s_j,t}} & i = 0 \end{cases}$$

Given that a true alarm occurs, its cost is the sum, for each s_i such that $s_i \leqslant s_{k^A(c)}$, of the probability that the true alarm is caused by a chain of length s_i with $0 \leqslant i \leqslant k^A(c)$, denoted by $\rho^A_{i,k^A(c)}$, multiplied by its length s_i. This cost is given by Proposition 13.

Proposition 13. *Given a search performed in a column c and $k(c)$ the index of the leftmost step that is to the right of column c, the number of hash operations needed to rule out a true alarm is:*

$$\sum_{i=0}^{k^A(c)} \rho^A_{i,k^A(c)} \eta_i.$$

Proof. $\rho^A_{i,k^A(c)}$ is the proportion of chains with length η_i in column c. In column c, chains that remain in the final matrix have a length between $\eta_{k^A(c)}$ and t.

If a true alarm is raised when searching in column c, this means that the attack chain merged with one of the $m_{s_{k^A(c),t}}$ chains present in the final matrix in column c. Each of these $m_{s_{k^A(c),t}}$ chains has a different length, the cost for ruling out the alarm, is thus the sum of the probability of merging with a chain of a given length, multiplied by its length.

Given a merge with one of the chains present in the final matrix in column c, the probability of merge with a chain of length η_i, with $0 \leqslant i \leqslant k^A(c)$ is $\rho^A_{i,k^A(c)}$.

Thus $\forall i \in \{0, ...k^A(c)\}$, the probability of matching a chain of length η_i multiplied by the cost of going through all the chain is $\rho^A_{i,k^A(c)}\eta_i$. □

Probability

Equation 13 asserts that a given column c contains $m_{s^A_{k(c)},t}$ elements. Consequently, for each column c, the probability of finding the searched element in any of the column's c elements can be computed straightforwardly. This probability is provided by Proposition 14.

Proposition 14. *The probability that a true alarm occurs when starting the attack chain in c is:*

$$p_{find}(c) = \frac{m_{s_{k^A(c)},t}}{N}.$$

Proof. As the search space consists of N elements, and since the column c contains $m_{s_{k^A(c)},t}$ elements, the probability of finding the searched element among the $m_{s_{k^A(c)},t}$ elements of column c is simply $\frac{m_{s_{k^A(c)},t}}{N}$. □

4.5.5 False Alarm

The cost and probability of a false alarm depend on various factors. To characterize it effectively, we first need to introduce several propositions.

Intermediary Results

We first introduce in Proposition 15 the probability that the starting element of the attack chain is not part of a surviving chain (Equation 13) in a given column.

Proposition 15. *Given a column c in which a search is performed, the probability that the starting element of the attack chain is not among the elements of surviving chains starting in $s_{k^A(c)}$ in column c is:*

$$p_{notunique}(c) = 1 - \frac{m_{s_{k^A(c)},c}}{N}.$$

Proof. The number of surviving chains in column c is, by construction, $m_{s_{k^A(c)},c}$, thus the probability that a random element in N is not one of the $m_{s_{k^A(c)},c}$ elements of those chains is straightforward. □

Following, we introduce in Proposition 16, the probability that the starting element of the attack chain is an element of a doom chain (the number of doom chains in a column c, $m^d_{s_i,c}$ has been defined in Equation 14).

Proposition 16. *Given a column c in which a search is performed, the probability that the starting element of the attack chain is among a doom chain is:*

$$p_{faulty}(c) = \frac{m^d_{s_{k^A(c)},c}}{N}.$$

Proof. The probability that the starting element of the attack chains is among the elements of surviving chains in column c is by definition, the average number of doom chains in column c, $m^d_{s_i,c}$, divided by the number N, of elements in the searched space. □

Lemma 3 defines the probability that the attack chain does not merge with a chain starting in a precise step between two columns c and c'. Lemma 3 is obtained by derivation of Lemma 1.

Lemma 3. *Given two columns c and c' and a step of index j with $c < c' \leqslant t$, the probability that the attack chain does not merge with any chain starting in step s_j or before by c', given it had not merged in or before c, is:*

$$p^A_{subnomrg}(c, c', j) = \prod_{i=c+1}^{c'} \left(1 - \frac{m_{s_j,i}}{N}\right).$$

Cost

The cost of the false alarm depends on the column in which the alarm is detected.

Thus, the cost to rule out a false alarm is η_i, with s_i the step at which the matrix chain that merged with the attack chain starts.

Probability

When performing a search in column c, the probability of a false alarm depends, among other factors, on the value of the starting element of the attack chain.

There are two possibilities:

(a) If the starting element is among the elements of doom chains, a false alarm will occur, but the probability of occurrence will vary for different steps.

(b) If the starting element is not one of the $m_{s_{k^A(c)},c}$ elements of surviving chains, either no alarm will be raised or a false alarm will occur.

These two possibilities ((a) and (b)) have different probabilities of false alarm, and thus require separate propositions. Proposition 17 provides the probability of a false alarm in the first case (a), while Proposition 18 provides the probability of a false alarm in the second case (b).

Proposition 17. *Given an attack chain starting in c, the probability $p_{fa}(c, i)$, to raise a false alarm due to merge with chains of length η_i and given that the starting element*

of the attack chain is not an element of a surviving chain in column c is:

$$p_{fa}(c,i) = \begin{cases} p^A_{subnomrg}(c,t,i-1) - p^A_{subnomrg}(c,t,i) & i > 0 \\ 1 - p^A_{subnomrg}(c,t,i) & i = 0 \end{cases}$$

Proof. The probability of the attack chain merging with any chain of length at least η_i is the complementary event of Lemma 3, for parameters (c,t,i), thus this probability is $1 - p^A_{\text{subnomrg}}(c,t,i)$. For the special case of $i = 0$, the attack chain can only merge with a chain of length η_i.

For $i > 0$, we define events E_1 and E_2 as "no merge occurs between c and t with a chain of length at least η_{i-1}" and "no merge occurs between c and t with a chain of length at least η_i", respectively. Since $E_2 \subset E_1$, we deduce that $\Pr(E_1 \wedge E_2) = P(E_2)$. Therefore, we have:

$$\begin{aligned} p_{\text{fa}}(c,i) &= \Pr(E_1 \wedge \bar{E_2}) \\ &= \Pr(E_1) - \Pr(E_1 \wedge E_2) \\ &= \Pr(E_1) - \Pr(E_2) \\ &= p^A_{\text{subnomrg}}(c,t,i-1) - p^A_{\text{subnomrg}}(c,t,i) \end{aligned}$$

□

Proposition 18. *Given an attack chain starting in c, the probability of raising a false alarm due to merge with chains of length η_i and given that the starting element of the attack chain is among the elements of doom chains in column c:*

$$p'_{fa}(c,i) = \begin{cases} p^A_{subnomrg}(c,t,k^A(c)-1) & i = k^A(c) \wedge i \neq 0 \\ p^A_{subnomrg}(c,t,i-1) - p^A_{subnomrg}(c,t,i) & i \neq k^A(c) \wedge i \neq 0 \\ 1 - p^A_{subnomrg}(c,t,i) & i = 0 \end{cases}$$

Proof. In the cases of $i \neq k^A(c)$ or $i = 0$ (second and third cases), since c is among the doom chains present in column c, it follows that $\eta_{k_A(c)} < \eta_i$. If the starting element of the attack chain is among a doom chain in column c, this implies that the starting element is not among a surviving chain of any step starting to the left of column $s_{k_A(c)}$. Given that $\eta_{k_A(c)} < \eta_i$ when $i \neq k^A(c)$ or $i = 0$, the second and third case are obtained exactly as demonstrated in Proposition 17.

For the specific case of $i = k^A(c)$ and $i \neq 0$, we again define events E_1 and E_2 as "no merge occurs between c and t with a chain of length at least $\eta_{k^A(c)-1}$" and "no merge occurs between c and t with a chain of length at least $\eta_{k^A(c)}$". By the definition of a doom chain, a chain starting with an element among a doom chain in column c will merge with a chain of length at least $\eta_{k^A(c)}$, therefore $\Pr(E_2) = 0$. Consequently, we have $\Pr(E_1 \wedge E_2) = 0$. From this, we deduce that when $i = k^A(c)$ and $i \neq 0$:

$$p_{\text{fa}}(c,i) = \Pr(E_1 \wedge \bar{E_2})$$

$$= \Pr(E_1)$$
$$= p^A_{\text{subnomrg}}(c, t, i-1)$$

□

4.5.6 Cost of the Search in One Column

The number of operations needed to perform a search in a column c is given by Theorem 19. It is obtained by summing for each event, its probability with its cost.

Theorem 19. *For a given column c, the average number of cryptographic operations C^A_c needed to perform a search is:*

$$\begin{aligned} C^A_c = {} & p_{find}(c) \sum_{i=0}^{k^A(c)} \rho_{i,k^A(c)} \eta_i \\ & + p_{notunique}(c) \sum_{j=0}^{k^A(c)} p_{fa}(c,j) \eta_j \\ & + p_{faulty}(c) \sum_{j=0}^{k^A(c)} p'_{fa}(c,j) \eta_j \\ & + (t-c)\, p_{noalarm}(c). \end{aligned}$$

Proof. To compute the total cost of a search in any column c, one must multiply the probabilities of the three possible events (true alarm, false alarm, no alarm) by their respective costs and sum these results.

(a) The probability of a true alarm, denoted as p_{find}, is given by Proposition 14. Its corresponding cost is $\sum_{i=0}^{k^A(c)} \rho_{i,k^A(c)} \eta_i$ as stated in Proposition 13. Hence, the cost of a true alarm in column c can be expressed as $p_{\text{find}}(c) \sum_{i=0}^{k^A(c)} \rho_{i,k^A(c)} \eta_i$.

(b) A false alarm can occur under two scenarios: (E_1) "The starting element of the attack chain does not equal an element of a surviving chain in column c". (E_2) "The starting element of the attack chain matches an element of a doom chain in column c".

Hence, the cost of a search in case of a false alarm is the sum of the probabilities of these two events multiplied by their respective costs.

- (E_1): The probability of event E_1 is provided by Proposition 15. The cost of a false alarm in this case, analogous to the true alarm, is, for all j with $0 \leqslant j \leqslant k^A(c)$, the probability of a merge with a chain of length exactly η_j multiplied by its cost

(η_j). The probability of merging with a chain of length exactly η_j under event E_1 is given by Proposition 17 as $p_{\text{fa}}(c,j)$. Hence, the cost of event E_1 is $\sum_{j=0}^{k^A(c)} p_{\text{fa}}(c,j)\eta_j$.

- (E_2): The probability of event E_2 is provided by Proposition 16. Analogous to E_1, the probability of a merge with a chain of length exactly η_j under event E_2 is given by Proposition 18 as $p'_{\text{fa}}(c,j)$. Hence, the cost of event E_2 is $\sum_{j=0}^{k^A(c)} p'_{\text{fa}}(c,j)\eta_j$.

In sum, the cost of a false alarm in column c is:

$$p_{\text{notunique}}(c) \sum_{j=0}^{k^A(c)} p_{\text{fa}}(c,j)\eta_j + p_{\text{faulty}}(c) \sum_{j=0}^{k^A(c)} p'_{\text{fa}}(c,j)\eta_j.$$

(c) The probability of no alarm, denoted as $p_{\text{noalarm}}(c)$, is given by Proposition 2. Its cost is $t-c$ as seen in Section 4.5.3. Hence, the cost of no alarm in column c can be expressed as $(t-c)\,p_{\text{noalarm}}$.

By summing the cost of each event ((a), (b), and (c)), C_c^A is obtained.

□

4.5.7 Average Attack Time

The average attack time using ℓ ASRTs is given in Theorem 20. As for RTs and DSRTs, this attack time is computed in two parts. The first part is the cost of the attack if the searched element is in the table, multiplied by the probability that the searched element is in the table. The second part is the cost of the attack if the searched element is not in the table multiplied by the probability that the searched element is not in the table.

Theorem 20. *Given N, ℓ ASRT with τ steps, the average number of hash operations T required to perform an attack is:*

$$T = \ell \sum_{c=1}^{t} \left(\frac{m_{s_{k^A(c)},t}}{N} \prod_{i=1}^{c-1} \left(1 - \frac{m_{s_{k^A(c)},t}}{N}\right) \sum_{j=1}^{c} C_{t-j+1}^A \right) + \ell \prod_{i=1}^{t} \left(1 - \frac{m_{s_{k^A(c)},t}}{N}\right) \sum_{c=1}^{t} C_c^A.$$

Proof. This expression is a generalization of Theorem 2 as it has been done for DSRT in Theorem 10. T is obtained by adding on the one hand, the success probability of the attack using ℓ tables, multiplied by its average cost, and on the other hand, the failure probability of the attack using ℓ tables, multiplied by the cost of a failed search.

The first term is obtained by multiplying for each column c, the probability of a true alarm in the column, with the probability of no true alarm in all earlier iterations:

$$\frac{m_{s_{k^A(c)},t}}{N} \prod_{i=1}^{c-1} \left(1 - \frac{m_{s_{k^A(c)},t}}{N}\right).$$

This is multiplied by the cost of all searches performed until reaching this column, which is $\sum_{j=1}^{c} C^A_{t-j+1}$.

The second term is obtained by multiplying the failure probability using ℓ tables, namely:

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

with the cost of performing a search in all columns of a table, namely: $\sum_{c=1}^{t} C^A_c$. □

5 Comparison

In this Section, we compare ASRT, DSRT and RT. We first present in Section 5.1 the experimental validation of the analysis of ASRT provided in Section 4.2.3. We then provide in Section 5.2 the methodology used for the comparison, and we finally present our results in Section 5.3.

In what follows, we call *configuration* a list of parameters describing a set of either RTs, DSRTs, or ASRTs. For RTs, a configuration is composed of one maximality factor α, a number of columns t, and a number of tables ℓ. For DSRT, in addition to these three parameters a configuration is also composed of the steps positions $\{s_1, s_2, ..., s_\tau\}$. Finally, the ASRT configuration is composed of the same parameters as DSRT plus the quasi-maximality factors considered at each step, defined by $\{\alpha_0, \alpha_1, ..., \alpha_\tau\}$.

5.1 Experimental Validation

In order to validate the formulas characterizing the different variants, RT, DSRT, and ASRT were implemented. A series of experiments were conducted to verify the close alignment between the theoretical results and the practical outcomes observed in concrete examples. The experiments were carried out on small-sized problems ($N = 2^{24}$ and $N = 2^{32}$) to facilitate a large number of attacks and generate multiple sets of tables for the different variants. Some larger simulations have then been made on an input space of size $N = 2^{42}$.

This section provides an overview of the tests performed to assess the success probability, memory requirement, precomputation time, and attack time of the implemented variants.

5.1.1 Success Probability

To evaluate the success probability of each variant, we generated multiple sets of tables for various success probabilities and configurations. For each variant, a large number of attacks were carried out using these tables, typically we conducted 1 000 000 attacks per configuration in order to obtain accurate results. The observed success probabilities were consistent with the theoretical predictions given by Equation 5 and Theorems 3, and 11, with differences below 0.1%.

5.1.2 Precomputation Time

The precomputation time is assessed by generating tables for the three variants with a fixed number of filters (typically around 20). Additionally, tests were conduced on smaller spaces ($N = 2^{32}$) using one filter per column. We did not conduct tests using one filter per column on bigger spaces because as the computations are distributed, the overhead becomes prohibitively large.

Our experimental results demonstrate that the precomputation time with one filter per column ($P_{\min}$) closely aligns with the theoretical predictions, with a maximum difference of less than 0.1%. Moreover, the experiments demonstrate that employing approximately 20 filters (including those used for steps) results in precomputation times that closely approach the theoretical lower bound, given by Equation 16, for all variants.

5.1.3 Memory Requirements

We did not implement table compression for memory testing since compression is independent of the variants chosen. Instead, we adapted the original formula provided in [17] for each variant. We tested the values involved in these formulas and verified that we obtained the expected numbers of EPs and SPs according to the theory. When applying the formulas to our experimental results versus the theoretical number of SPs and EPs to store, the differences were less than 0.05% across all tested configurations.

5.1.4 Attack Phase

The attack phase was tested in a manner analogous to the success probability evaluation. Tables were generated for various configurations and target success probabilities, followed by conducting a substantial number of attacks for each variant and configuration (typically between 100 000 and 1 000 000 attacks to ensure the accuracy of the average attack time measured – the attack times of DSRT, and especially ASRT, are more variable than for RT, so more attacks were needed to obtain reliable measurements). The average attack time closely adhered to the theoretical predictions, with a difference of less than 0.8% for all variants. The results were well distributed around the theoretical mean, with no significant difference based on the configuration used. The variability of the attack time is further discussed in Section 6.

5.2 Comparison Methodologies

In Section 5.3, the precomputation and attack times of the RT, DSRT, and ASRT variants are compared, using the same targeted memory and targeted success probability. This approach allows for an evaluation of RT, DSRT and ASRT in a manner consistent with that presented in the DSRT paper [20]. Furthermore, comparing the variants at fixed success probability and memory settings highlights the trade-off between precomputation and attack times for each case.

This Section outlines the evaluation process for the precomputation time, attack time, memory, and success probability of each variant, and provides justifications for the chosen methodology.

5.2.1 Precomputation Time

$P_{\min}$ was chosen for comparison instead of P. The reasons explaining this choice is firstly that our tests demonstrated that filter usage gives results near the theoretical lower bound for RT, DSRT, and ASRT, thus offering a suitable comparison basis. Secondly, to perform the evaluation of tens of thousands of configurations, we favored $P_{\min}$ over a more time-consuming filter optimization evaluation. Finally, the use of $P_{\min}$ avoids any bias that could arise from selecting filter-based comparisons and, as discussed in Section 5.1.2, $P_{\min}$ approximates filter results for all variants.

5.2.2 Memory

The memory lower bound is used to evaluate the memory used by each variant. This lower bound closely approximates (within 1%) the compressed delta encoding, but offers formulas that are easier to work with. To ensure no bias towards DSRT or ASRT over RT, each step of DSRT and ASRT was treated as a separate table for memory computation which tends to slightly favor RTs over DSRTs and ASRTs. This is fair as DSRT and ASRT share similar step-by-step storage methodologies. We also checked that the difference between compressed delta encoding applied to ASRT and DSRT and their minimal memory lower bound remains under 0.7%, giving us the confidence to use the memory lower bound for comparison.

5.2.3 Success Probability and Attack Time

To compare the success probability and attack time of each variant, we simply applied Equation 6, and Theorems 3, 11, 9, 10, and 19. As mentioned in Section 5.1, the success probabilities and attack times estimated using these formulas closely align with the success probabilities and attack time obtained in practice for each variant.

5.3 Results

5.3.1 Parameters

The search space considered in the comparison is $N = 2^{42}$, which was chosen to allow easy comparison with the results from papers [14, 17, 18, 20]. The memory considered is $M = 32$GB, which represents a practical use case. The variants are then compared for various success probabilities, ranging from 80% to 99.95%. To maintain brevity, only results for some of the tested success probabilities are presented; however, the conclusions drawn are valid for all success probabilities.

For each success probability, possible configurations for RTs, DSRTs, and ASRTs are computed. For RTs, the number of possible configurations for fixed memory and fixed success probability is limited, as the only variable left free is the number of tables. For DSRT, in addition to the number of tables, the positions of the steps can vary according to the configurations, leading to many possible configurations as extensively explained in [20]. When considering ASRT, the position of steps that remain free, and the number of elements to add in each step (determined by $\{\alpha_0, \alpha_1, ...\alpha_\tau\}$) are additional parameters to set.

In total, the number of possible configurations for a given number of tables, given probability, and given memory is 1 for RT, bounded by $(t-1)^\tau$ for DSRT, and bounded by $(t-1)^\tau \times N^\tau$ for ASRT.

Given the number of possible configurations for DSRTs and ASRTs, the number of steps is set to 4 for DSRTs and to 2 for ASRTs. This choice is justified by the fact that using more than 4 steps for DSRTs does not significantly increase their performance, as stated in [20] and using more than 2 steps for ASRT does not allow a significant gain compared to the computational cost needed to find possible configurations with 3 steps.

The algorithm presented in [20], is used to determine the DSRT configurations . For ASRT, we performed an exhaustive search , adjusting the steps for each α and t. We varied both t and the columns $\{s_1, s_2\}$ in steps of 100 columns, α_0 in steps of 0.003, α_1 in steps of 0.002, and α_2 in steps of 0.001. This strategy offers a balance between precision and computational efficiency in discovering configurations. The step size for α_0 is larger than that for α_1, and the step size for α_1 is larger than that for α_2. This is due to the fact that α_0, the maximality factor of the leftmost step, is associated with longer chains. Therefore, the impact of changes of α_0 is less pronounced in the resulting number of chains. The same conclusion holds for choosing the step variation of α_1 larger than those for α_2.

5.3.2 Figures Interpretation

Figures 4a 4b, 4c depict, for various success probabilities, a series of points in the attack time / precomputation time space corresponding to RTs, DSRTs and ASRTs in a multitude of configurations. Table 1, presents some noteworthy results. The attack time and precomputation time are expressed in the number of hashes to perform.

For each variant the *best configurations* are the configurations for which there is no existing configuration that is better both in precomputation and in attack.

In each plot, ASRT configurations are represented by red dots, DSRT configurations are represented by green dots, and RT configurations are indicated by orange dots. The black line represents the *optimal configurations* among all the configurations of all variants (the Pareto frontier).

5.3.3 ASRTs Versus RTs

In the following Sections, the focus will be on comparing ASRT solely to DSRT, since as illustrated in Figures 4a, 4b, and 4c, there always exists an ASRT configuration superior to the RT configurations. Furthermore, similar to DSRT, the number of possible configurations for a given success probability and specified memory is higher when using ASRT than when using DSRT or RT. As a result, employing ASRTs allows to reach more trade-off between precomputation and attack compared to RTs.

The explanation regarding why ASRTs outperform RT is provided in Section 6.

5.3.4 ASRTs Versus DSRTS

The subsequent paragraphs highlight noteworthy results from Table 1 and Figure 4, which help illustrate the differences between the use of ASRTs and DSRTs. We discuss and provide interpretations for these results in Section 6.

Case 1: DSRTs more Efficient than ASRTs

There are instances, especially when the targeted coverage is low enough to necessitate only a single DSRT or ASRT, where DSRT configurations are more advantageous than ASRT configurations. This is demonstrated in Figure 4a and the first sub-Table of Table 1. For a given attack time achievable with DSRT, the corresponding ASRT needs considerably more precomputation time, rendering the variant configuration less interesting. The configurations on the left of Figure 4a may be worthwhile in some cases, as they permit a reduction in attack time by 4% compared to DSRT, albeit at the cost of a 31% increase in precomputation time.

Table 1 Expected gain illustrated on several examples with ASRT and DSRTs. Precomputation and attack phase numbers are quantity of cryptographic operations.

Success probability: 90%		
	Precomputation	Attack
1 ASRT	1.48×10^{13}	1.82×10^{6}
1 DSRT	1.21×10^{13}	
Gain	+22%	
1 ASRT	2.9×10^{13}	1.3×10^{6}
1 DSRT	2.2×10^{13}	1.35×10^{6}
Gain	+31%	−4%
Success probability: 99%		
	Precomputation	Attack
ASRT	5×10^{13}	4.11×10^{6}
DSRT	2.89×10^{13}	4.98×10^{6}
Gain	+73%	−17%
ASRT	3.23×10^{13}	4.7×10^{6}
DSRT	2.89×10^{13}	4.98×10^{6}
Gain	+12%	−6%
Success probability: 99.95%		
	Precomputation	Attack
ASRT	4.8×10^{13}	8.71×10^{6}
DSRT	7.2×10^{13}	
Gain	−33%	
ASRT	7.2×10^{13}	7.57×10^{6}
DSRT		8.66×10^{6}
Gain		−13%

Case 2: DSRTs and ASRTs Efficient

Figure 4b illustrates a typical scenario where ASRT may be preferred over DSRT if attack time is the most important factor for the attacker. ASRT configurations achieve nearly identical trade-offs as the fastest DSRT configurations, and additionally offer a range of faster attack configurations at the cost of increased precomputation time. For instance, compared to the fastest DSRT configurations, it is possible to reach trade-offs 6% quicker in attack but requiring 12% additional precomputation time, or trade-offs that are 17% faster in attack at the expense of a 73% increase in precomputation time compared to DSRT (only 24% slower than the fastest RT configuration).

These results are observable for different coverage values greater than 97%. When targeting coverage higher than 99.5%, ASRT configurations outperform DSRT configurations.

5.3.5 Case 3: ASRTs more Efficient than DSRTs

For high targeted coverage, typically coverage requiring three or more tables, ASRTs outperform DSRTs. Figure 4c presents results for a common case discussed in the literature: the use of four quasi-maximal vanilla RTs, which allows to achieve a coverage of 99.95%.

As depicted in Figure 4c and Table 1, the optimal configurations are nearly all ASRT configurations. Compared to the fastest DSRT configuration, an ASRT configuration can achieve the same attack time with 33% less precomputation time, or can reach a configuration 13% faster in attack for the same precomputation time.

6 Discussion

We initiate the discussion by comparing ASRT with RT exclusively in Section 6.1. This comparison facilitates the comprehension of the critical factors that render ASRT effective in a straightforward manner. Subsequently, in Section 6.2, we use the arguments developed in comparison with RT to contrast DSRT and ASRT, explaining why ASRT outperforms DSRT under certain circumstances and not in others.

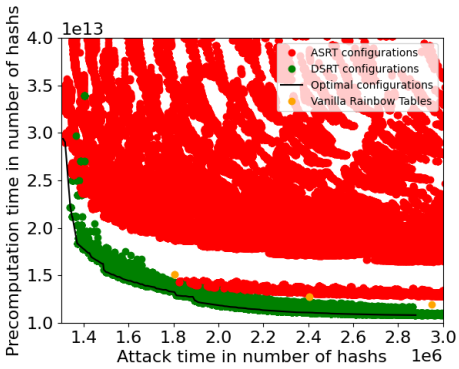
6.1 Comparison with RT

To simplify the explanation for the reader, we opted to compare RT and ASRT using a representative example. Although the comparison uses a specific example, the insights presented can be generalized to a broad range of cases.

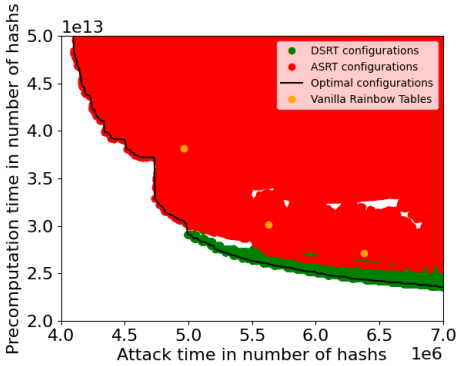
Figure 5 illustrates a Rainbow matrix (depicted in green) and its corresponding ASRM. For the sake of clarity, this figure is constructed for a small space ($N = 2^{24}$), but the proportions remain consistent for larger spaces. Both configurations aim for a coverage of 98% and with the same memory.

Precomputation

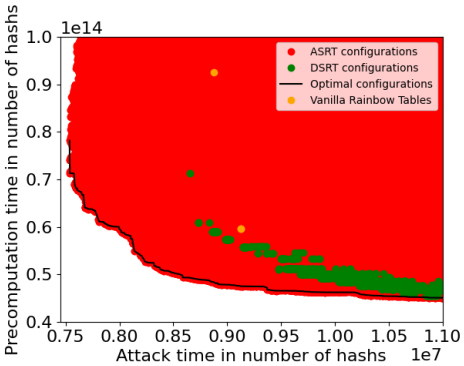
For the same coverage and memory, the initial m_0 considered for the RT is substantially larger than the m_{s_0,s_0} of the ASRT (almost four times larger). This observation holds for all significant configurations and constitutes a key factor exploited by both



(a) Trade off between precomputation time and attack time **90%** of success, $N = 2^{42}$ and a 31.99 GB memory.



(b) Trade off between precomputation time and attack time **99%** of success, $N = 2^{42}$ and a 31.99 GB memory.



(c) Trade off between precomputation time and attack time **99.95%** of success, $N = 2^{42}$ and a 31.99 GB memory.

Figure 4: Trade-off between precomputation and attack.

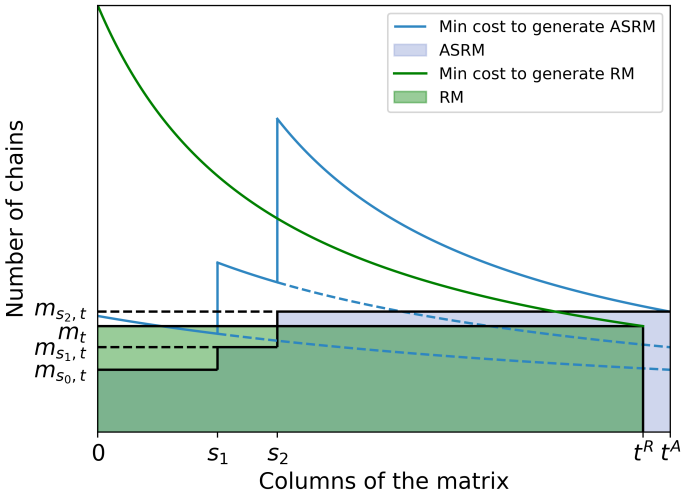


Figure 5: ASRM with 2 steps versus the corresponding RT for the same memory and the same coverage (98%).

ASRT and DSRT. Selecting four times fewer elements at the beginning of precomputation does not lead to a reduction by a factor of four in the final number of elements obtained at the end of the precomputation, but slightly less than twice as many. It is important to note that when employing filtration, as is our case, choosing four times more elements at the beginning of the precomputation does not quadruple the precomputation time, but increases it slightly less than three-fold. The saved time can be effectively employed to add more elements later in the precomputation phase.

In column s_1, adding more elements to the ASRM still keeps the number of hashes required during the precomputation phase much lower for the ASRM than for the rainbow matrix, and allows increasing the number of elements in the final table to about 75% of the number of elements in the final rainbow matrix.

The number of SPs considered in column s_2 provides crucial insight into how the ASRT outperforms vanilla RT. The number of SPs considered in column s_2 is much larger than those considered in s_0 and s_1 and, at the same time, is significantly shifted to the right of the table. This enables to keep more elements in column t than the vanilla RT. Although the precomputation of the elements between s_2 and t requires more time than the precomputation of the RT chains in the corresponding columns, the time gained at the beginning of the precomputation by computing fewer chains significantly compensates for this extra time.

Memory

A crucial point to understanding why the matrices depicted in Figure 5 occupy the same memory is based on acknowledging the impact of the $\log(m_0)$ and $\log(m_{s_0,s_0})$ factors in the memories formulas. For instance, with RTs, the initial m_0 in the case of Figure 5 is four times larger than the number of elements obtained at the end. Since,

in this example $N = 2^{24}$, this implies that each SP of the RT consumes about 15% more memory per element than the m_{s_0,s_0} elements considered in the ASRT variant. The RT SPs then take about 25% more memory per SP than the SP of the chains beginning in s_1 and about 10% more memory per element than the SPs of chains starting in s_2.

In the final analysis, even though more SPs must be stored with the ASRT than with RT, and the fact that storing three batches of SPs slightly mitigates the decrease in memory required to store each SP, the memory used to store the ASRT SPs is roughly 12% lower than the memory required to store the RT SPs.

This 12% memory saving is then used to "compensate" for storing the EPs, which is more optimized for the RTs due to: (a) greater efficiency in compressing a single "large" batch of elements (as in RT EPs) as opposed to three "small" batches of elements (as in ASRT); and (b) The fact that slightly fewer EPs need to be stored when using RT than when using ASRT.

Attack

Even though the ASRT chains are longer than the RT chains, the attack phase is faster when using this particular ASRT configuration than the RT configuration.

Firstly, the chains of the ASRT starting in s_1 and s_2 are substantially shorter than those of the RT, and these chains account for about half of the ASRT chains. Thus, when performing a search in columns between s_2 and t, about half the time, the search will cost significantly less than the search in the RT. For the remaining half of the time, where a match occurs with chains starting in s_0 rather than s_1 or s_2, the cost is higher than when using RT, but less significantly. This is due to the fact that the difference between the lengths of the RT chains and the ASRT chains starting in s_0 is considerably less high than the difference in length between the RT chains and the ASRT chains starting in s_1 or s_2.

Lastly, there are more chains between s_2 and t in the ASRM than in the rainbow matrix . This shifts the average column in which the searched element is found, pushing it further to the right. Consequently, this decreases the number of searches before finding the searched element and increases the chances of matching with chains of length η_i and η_2 instead of t, thus increasing the chance of performing a search costing less operations.

6.2 ASRTs versus DSRTs

The comparison between ASRT and DSRT is more complex than between ASRTs and RTs. To facilitate this comparison, we will separately address the comparison of ASRT and DSRT in terms of coverage, precomputation time, attack time, and memory.

It is worth noting that the following points are true when ASRT and DSRT are compared for the same number of tables. When fewer ASRT tables are used than DSRT, their shape can be changed . For brevity and simplicity, we chose an example where DSRT and ASRT use the same number of tables.

6.2.1 Coverage

Quasi-Maximality Factor

A key point when using the same number of ASRT tables than DSRT or RT is the need for continuously increasing quasi-maximality factors. In other words, for ASRT to be effective, the quasi-maximality factors should satisfy $\alpha_0 < \alpha_1 < ... < \alpha_s$. The intuition behind this is that the operational principle of ASRT is based on maintaining a large number of chains, ideally shorter ones, towards the right of the matrix.

One can perceive the right part of an ASRM as the segment that ensures the speed of the attack phase, and the left part as the segment that guarantees to reach the targeted coverage.

To perform well in comparison, ASRT must therefore start with an initial number of chains m_{s_0,s_0} significantly lower than that of DSRT. As the coverage increases, the quasi-maximality factor used tends to increase (to maintain an acceptable attack time by not adding an additional table), thus enhancing the use of ASRT.

However, at lower coverage, the maximality factor of RTs and DSRTs tends to be lower. This is particularly true for DSRTs, where the central idea is to generate matrices with lower quasi-maximality factors and to "compensate" for the waste of chains caused by the decrease in the quasi-maximality factor, with the steps.

Number of Tables

The DSRT variant tends to be less interesting, particularly regarding the attack time, as the number of tables increases. One of the factors contributing to DSRT better attack performance over RT is its general reliance on one fewer table than RT, which thereby reduces attack time. However, at higher coverage, the requirement for tables increases, and thus the benefit of using one less table decreases, since each table has less impact when more tables are used.

Conversely, the most effective ASRT configurations can use the same number of tables as the best-performing RT configurations in the attack phase. This is particularly true when fewer RT tables are used, in this case, the attack performance of ASRT is not based mainly on the difference in the number of tables used. In highter coverage, e.g. 99.95%, ASRT configurations can require only 3 tables against 4 DSRT tables and 5 RTs tables.

Conclusion on Coverage

ASRT outperforms DSRT in both attack and precomputation scenarios when the same number or less tables is used, or if the quasi-maximality factors are sufficiently high. Under different circumstances, either DSRT performs better in both attack and precomputation, or ASRT is faster in attack but slower in precomputation. These latter cases are further detailed in Sections 6.2.2 and 6.2.3.

6.2.2 Precomputation

DSRT was designed with fast precomputation in mind. The critical element that facilitates the speed of DSRT precomputation is the choice of a lower maximality factor than RT, and compensate the resulting waste of chains through the use of steps.

Despite that, compared to ASRT, DSRT generally uses a higher maximality factor than ASRT initial maximality factor (while still being lower than RT). Consequently, the initial phase of precomputation is more costly when generating DSRT than ASRT. However, ASRT subsequent maximality factors will increase during the precomputation phase. In some cases, as soon as the first ASRT step is reached, the number of chains to compute becomes higher for ASRT than for DSRT. Ultimately, in a significant number of cases, the rising quasi-maximality factors of ASRT lead to a slower precomputation phase, as more chains need to be computed in the ASRT case after the first step.

When DSRT maximality factor is sufficiently high, the number of chains considered up until the final step of ASRT remains lower than the number of chains considered in the DSRT matrix. In the final step, the number of chains considered when using ASRT exceeds that of DSRT in all examples we have encountered. However, when the difference does not offset ASRT initial precomputation speed advantage, the precomputation time for the ASRT variant ends up being less than that of DSRT. On the other hand, if this is not the case, the ASRT precomputation time exceeds that of DSRT.

6.2.3 Attack

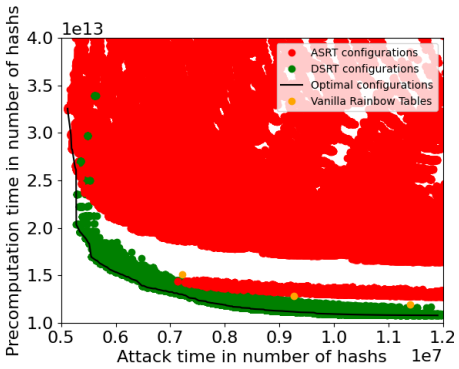
Inherently, ASRTs tend to outperform DSRTs in terms of attack due to their key concept of maximizing the number of short chains on the right side of the matrix. As discussed in Section 6.2.2, this may come to the cost of a higher precomputation time.

Compared to DSRTs, ASRTs have more chains on the right of the matrix, with some parts of these chains being shorter than even the shortest DSRT chains. When ASRT shorter chains are not shorter than DSRT shortest chains, ASRT typically still comes with superior efficiency in attack due to the slightly higher number of chains per table, and the lower cost of building the attack chain when using ASRT.

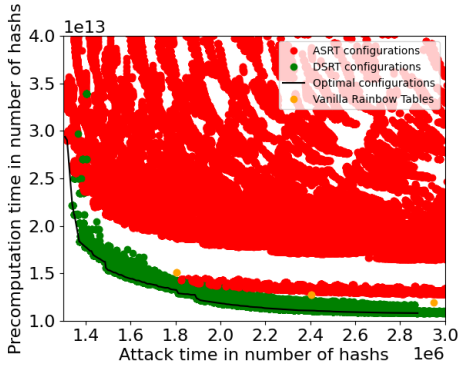
However, when ASRT shorter chains are not short enough, DSRT surpasses ASRT in attack speed mainly due to the fact that when a false alarm is detected in a DSRT step, the attack chain is not rebuilt until the final table column. It acts as a sort of partial checkpoints [11] and is one of the key points for maintaining the efficiency of DSRTs in attack.

In scenarios where ASRT does not have a sufficiently high last maximality-factor α_τ to guarantee a sufficient number of chains on the right side of the matrix, and where the chains of steps are not short enough, ASRT configurations are inferior to DSRT configurations in attack.

Nonetheless, at sufficiently high coverage (typically greater or equal to 90%), there always exists an ASRT configuration that outperforms the fastest DSRT in terms of attack time. However, this may come at the cost of a longer precomputation phase, particularly when 3 or fewer tables are used.



(a)



(b)

Figure 6: Trade-off between precomputation and attack for 16GB and 32GB available memory, and 90% coverage.

6.3 Memory

6.3.1 Memory Variation

It is essential to note that the relative performances of ASRT, DSRT, and RT do not depend on the memory available. By relative performances, we refer to the difference in performance between the variants, irrespective of the memory. Figures 6a and 6b illustrate the configurations of ASRTs, DSRTs, and RTs for a space $N = 2^{42}$, a coverage of 90%, and memory availabilities of 16GB and 32GB, respectively.

For a given coverage, it is clear that the precomputation time does not vary with the available memory, provided this memory allocation remains "reasonable". However, noticeable side effects might appear in the precomputation time when t is exceedingly low (high available memory), or when t is overly high (low available memory), resulting in an insufficient number of chains. Excluding these exceptional cases where

precomputation time could marginally fluctuate with memory changes, the precomputation time does not depend on the available memory, since the number of element in the matrix to compute remains the same irrespective of the memory available. The only variation lies in the shape of the computed matrices, which may be more or less wide or tall, depending on the memory available.

Regarding the attack phase, the results, though intuitive, are less clear-cut. Initially, when considering the RT, the results shown in Figures 6a and 6b align perfectly with expectations. As the memory is doubled while N remains the same and following the relation $T = N^2/M^2$, we can expect that by doubling the memory, we quarter the attack time, which is indeed the case.

For the DSRT and ASRT variants, we do not provide proof, but we hypothesize that the behavior of these variants is equivalent to that of RTs.

We give some arguments to justify our intuition: (a) numerous experiments performed on various search spaces, memory availabilities, and coverage consistently confirm this supposition. For instance, as presented in Figures 6a and 6b, DSRTs and ASRTs follow this postulate and their attack time quarter when the memory is divided by two.; (b) the fact that RTs are, in essence, a special case of DSRT and ASRT (ASRT and DSRT formulas perfectly align with RT formulas when all steps are in the same column t); (c) ASRT and DSRT can also be considered as multiple RTs sharing the same reduction functions. If each RT follows the relation $T = N^2/M^2$, we can expect the combined DSRT and ASRT to also adhere to this relation.

6.3.2 Memory Accesses

On average, the number of memory accesses required for the ASRT attack is less than those needed for the DSRT and RT variants. This is primarily due to the greater number of chains in the right part of the table, which tends to be higher in ASRT than in the other two variants, thereby reducing the number of searches (and consequently the number of memory accesses).

Nevertheless, the decrease in the number of memory accesses is not significant; it amounts to only a few percents, depending on the coverage and memory used.

6.4 Worst-Case Attack Time

Like DSRTs, a drawback of ASRTs is that their worst-case attack time is longer than that of the worst-case RT attack. This is due to the fact that in a significant number of cases (almost all), the longest ASRT chain exceeds the length of the longest RT chain.

Consequently, the attack time increases when the entire table must be searched through. This disadvantage can be mitigated since ASRTs are typically of interest in situations with high coverage, and thus, the worst-case scenario occurs very infrequently.

The most significant implication of this is an increase in the variability of the attack time. Similar to DSRTs, while the average attack time of ASRTs is shorter than that of RTs, it is more variable.

7 Conclusion

This paper introduces ASRTs, which demonstrate superior performance than vanilla RTs and, under certain conditions, DSRTs. The concept of ASRTs is the addition of chains in specified columns, referred to as steps. The variant involves incrementally adding more chains at each step, with the goal of having more chains on the right side of the final matrix at the end of the precomputation phase. This approach implies the concept of *ascending stepped* Rainbow Tables and offers a two-fold benefit: improved matrix coverage and faster attack phase.

Owing to the larger parameter space, ASRTs afford a greater number of possible configurations compared to DSRTs. When targeting sufficiently high coverages, ASRTs outperform both DSRTs and RTs in terms of both attack and precomputation times, with the extent of the gain primarily dependent on the targeted coverage.

In our practical experiments, when a coverage of 99.95% (frequently referenced in literature) is targeted, ASRTs can reduce precomputation time by 33% when compared to DSRTs, for the same attack time, coverage, and memory. Alternatively, ASRTs can trim attack time by 13% for the same coverage, memory, and precomputation time. Compared to RTs, this represents a precomputation time reduction of 48% for the same attack time, coverage, and memory, or an attack time reduction of 15% with a precomputation time that remains 24% lower.

When targeting lower coverage, ASRTs can reduce attack time compared to DSRTs (and thus RTs) at the cost of an increase in precomputation time. For instance, for a 99% coverage, ASRTs can decrease attack time by 17% at the cost of an additional 73% in precomputation time compare to DSRT, or reduce attack time by 6% with a 12% increase in precomputation time.

Using both DSRTs and ASRTs instead of RTs can enable an attacker to substantially decrease both the attack and precomputation times of the used TMTOs, if the suitable variant is selected based on their targeted coverage and requirements. The potential for combining DSRTs and ASRTs in one variant remains a prospect for future work.

References

[1] J. Hong, S. Moon, A comparison of cryptanalytic tradeoff algorithms, J. Cryptol. 26 (4) (2013) 559–637.

[2] G. Lee, J. Hong, Comparison of perfect table cryptanalytic tradeoff algorithms, Des. Codes Cryptogr. 80 (3) (2016) 473–523.

[3] A. Biryukov, A. Shamir, D. Wagner, Real time cryptanalysis of a5/1 on a pc, in: International Workshop on Fast Software Encryption, Springer, 2000, pp. 1–18.

[4] E. Biham, O. Dunkelman, Cryptanalysis of the A5/1 GSM stream cipher, in: B. K. Roy, E. Okamoto (Eds.), Progress in Cryptology - INDOCRYPT 2000, First International Conference in Cryptology in India, Calcutta, India, December 10-13, 2000, Proceedings, Vol. 1977 of Lecture Notes in Computer Science, Springer,

2000, pp. 43–51. doi:10.1007/3-540-44495-5_5.
URL https://doi.org/10.1007/3-540-44495-5_5

[5] J. W. Kim, J. Seo, J. Hong, K. Park, S. Kim, High-speed parallel implementations of the rainbow method based on perfect tables in a heterogeneous system, Softw. Pract. Exp. 45 (6) (2015) 837–855. doi:10.1002/spe.2257.
URL https://doi.org/10.1002/spe.2257

[6] J. D. Golić, Cryptanalysis of alleged a5 stream cipher, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 1997, pp. 239–255.

[7] G. Avoine, X. Carpent, D. Leblanc-Albarel, Rainbow Tables: How Far Can CPU Go?, The Computer JournalBxac147 (10 2022). arXiv:https://academic.oup.com/comjnl/advance-article-pdf/doi/10.1093/comjnl/bxac147/46671690/bxac147.pdf, doi:10.1093/comjnl/bxac147.
URL https://doi.org/10.1093/comjnl/bxac147

[8] M. Vanhoef, A time-memory trade-off attack on WPA3's SAE-PK, in: J. P. Cruz, N. Yanai (Eds.), APKC '22: Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop, APKC@AsiaCCS 2022, Nagasaki, Japan, 30 May 2022, ACM, 2022, pp. 27–37. doi:10.1145/3494105.3526235.
URL https://doi.org/10.1145/3494105.3526235

[9] N. Mentens, L. Batina, B. Preneel, I. Verbauwhede, Time-memory trade-off attack on FPGA platforms: UNIX password cracking, in: K. Bertels, J. M. P. Cardoso, S. Vassiliadis (Eds.), Reconfigurable Computing: Architectures and Applications, Second International Workshop, ARC 2006, Delft, The Netherlands, March 1-3, 2006, Revised Selected Papers, Vol. 3985 of Lecture Notes in Computer Science, Springer, 2006, pp. 323–334.

[10] W. Wang, D. Lin, Analysis of multiple checkpoints in non-perfect and perfect rainbow tradeoff revisited, in: S. Qing, J. Zhou, D. Liu (Eds.), Information and Communications Security - 15th International Conference, ICICS 2013, Beijing, China, November 20-22, 2013. Proceedings, Vol. 8233 of Lecture Notes in Computer Science, Springer, 2013, pp. 288–301. doi:10.1007/978-3-319-02726-5_21.
URL https://doi.org/10.1007/978-3-319-02726-5_21

[11] G. Avoine, P. Junod, P. Oechslin, Time-memory trade-offs: False alarm detection using checkpoints, in: S. Maitra, C. E. V. Madhavan, R. Venkatesan (Eds.), Progress in Cryptology - INDOCRYPT 2005, Vol. 3797 of Lecture Notes in Computer Science, Springer, 2005, pp. 183–196. doi:10.1007/11596219_15.
URL https://doi.org/10.1007/11596219_15

[12] G. Avoine, A. Bourgeois, X. Carpent, Analysis of rainbow tables with fingerprints, in: E. Foo, D. Stebila (Eds.), Information Security and Privacy - 20th

Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings, Vol. 9144 of Lecture Notes in Computer Science, Springer, 2015, pp. 356–374. doi:10.1007/978-3-319-19962-7_21.
URL https://doi.org/10.1007/978-3-319-19962-7_21

[13] B. Kim, J. Hong, Analysis of the non-perfect table fuzzy rainbow tradeoff, in: C. Boyd, L. Simpson (Eds.), Information Security and Privacy - 18th Australasian Conference, ACISP 2013, Brisbane, Australia, July 1-3, 2013. Proceedings, Vol. 7959 of Lecture Notes in Computer Science, Springer, 2013, pp. 347–362. doi:10.1007/978-3-642-39059-3_24.
URL https://doi.org/10.1007/978-3-642-39059-3_24

[14] G. Avoine, X. Carpent, Heterogeneous rainbow table widths provide faster cryptanalyses, in: R. Karri, O. Sinanoglu, A. Sadeghi, X. Yi (Eds.), Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017, ACM, 2017, pp. 815–822.

[15] A. Biryukov, S. Mukhopadhyay, P. Sarkar, Improved time-memory trade-offs with multiple data, in: B. Preneel, S. E. Tavares (Eds.), Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers, Vol. 3897 of Lecture Notes in Computer Science, Springer, 2005, pp. 110–127. doi:10.1007/11693383_8.
URL https://doi.org/10.1007/11693383_8

[16] A. Biryukov, S. Mukhopadhyay, P. Sarkar, Improved time-memory trade-offs with multiple data, in: B. Preneel, S. E. Tavares (Eds.), Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers, Vol. 3897 of Lecture Notes in Computer Science, Springer, 2005, pp. 110–127.

[17] G. Avoine, X. Carpent, Optimal storage for rainbow tables, in: H. Lee, D. Han (Eds.), Information Security and Cryptology - ICISC 2013 - 16th International Conference, Seoul, Korea, November 27-29, 2013, Revised Selected Papers, Vol. 8565 of Lecture Notes in Computer Science, Springer, 2013, pp. 144–157.

[18] G. Avoine, X. Carpent, D. Leblanc-Albarel, Precomputation for rainbow tables has never been so fast, in: E. Bertino, H. Shulman, M. Waidner (Eds.), Computer Security – ESORICS 2021, Springer International Publishing, Cham, 2021, pp. 215–234.

[19] G. Avoine, X. Carpent, B. Kordy, F. Tardif, How to handle rainbow tables with external memory, in: J. Pieprzyk, S. Suriadi (Eds.), Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part I, Vol. 10342 of Lecture Notes in Computer Science, Springer, 2017, pp. 306–323.

[20] X. C. Gildas Avoine, D. Leblanc-Albarel, a completer.

[21] 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) (2008) 17:1–17:22.

[22] P. Oechslin, Making a faster cryptanalytic time-memory trade-off, in: D. Boneh (Ed.), Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings, Vol. 2729 of Lecture Notes in Computer Science, Springer, 2003, pp. 617–630.