Variants of the Distinguished Point Method for Cryptanalytic Time Memory Trade-offs
(Full version)

Jin Hong[†], Kyung Chul Jeong[‡], Eun Young Kwon[†],
In-Sok Lee[⋆,‡], and Daegun Ma[⋆,†]

Department of Mathematical Sciences and ISaC-RIM,
Seoul National University, Seoul, 151-747, Korea
{jinhong,white483,madgun7}@snu.ac.kr[†]
{jeongkc,islee}@math.snu.ac.kr[‡]

Abstract. The time memory trade-off (TMTO) algorithm, first introduced by Hellman, is a method for quickly inverting a one-way function, using pre-computed tables. The distinguished point method (DP) is a technique that reduces the number of table lookups performed by Hellman's algorithm.
In this paper we propose a new variant of the DP technique, named variable DP (VDP), having properties very different from DP. It has an effect on the amount of memory required to store the pre-computed tables. We also show how to combine variable chain length techniques like DP and VDP with a more recent trade-off algorithm called the rainbow table method.

Key words: time memory trade-off, Hellman trade-off, distinguished points, rainbow table

1 Introduction

In many cases, cryptanalysis of a cryptographic system can be interpreted as the process of inverting a one-way function. Unlike most approaches that depend on the specific target system, time memory trade-off (TMTO) is a generic approach that can be used on any one-way function.

Let $f : \mathcal{X} \to \mathcal{Y}$ be any one-way function. For example, this could be a map sending a key to the encryption of a specific fixed plaintext. A way to efficiently invert this map would imply total breakdown of the encryption system. There are two trivial ways to invert f that do not involve the inner workings of f. Given a target $y \in \mathcal{Y}$ to invert, one may go about the time consuming process

The short version of this paper will appear under the same title in the proceedings of the 4th Information Security Practice and Experience Conference (ISPEC 2008), LNCS, Springer.

All authors are supported in part by BK 21.
⋆ Partially supported by KRF Grant ♯2005-070-C00004.

of computing $f(x)$ for every $x \in \mathcal{X}$, until a match $f(x) = y$ is found. The other method is to do this exhaustive process in a pre-computation phase and to store the resulting pairs $(x, f(x))$ in a table, sorted according to the second components. Then, when a target $y \in \mathcal{Y}$ is given, it can be searched for in the table among the second components and the corresponding first component is simply read off as the answer. Whereas the exhaustive search method takes a long time, the table lookup method requires a large storage space. TMTO is a method that comes between these two extremes and can invert a one-way function in time shorter than the exhaustive search method using memory smaller than the table lookup method.

Cryptanalytic TMTO was firstly introduced by Hellman [11]. If the one-way function f to be inverted is defined on a set of size N, under typical parameters, the pre-computation phase of his algorithm takes time $\mathcal{O}(N)$ in creating certain one-way chains, after which a *digest* of this exhaustive computation is stored in a table of size $\mathcal{O}(N^{\frac{2}{3}})$. This table is used during the online phase to recover the pre-image x of a given target $f(x)$ in time $\mathcal{O}(N^{\frac{2}{3}})$. Soon after Hellman's work, the idea of distinguished points (DP), attributed to Rivest in [9], was introduced. When applied to Hellman's algorithm, it reduces the number of table lookups required during the online phase. This is useful when the table is so large that table lookups become expensive. More recently, Oechslin [14] suggested a different way of creating the one-way chain. It is called the rainbow table method and a reduction in online time by a factor of two was claimed. It is known that, asymptotically, these algorithms are the best one can hope for if the structure of f is not to be used [5].

The contributions of this paper are two-folds. The first is the introduction of a new technique which we shall call *variable DP* (VDP). As with the original DP idea, VDP is a technique that can be used with the Hellman method and also with the multi-target versions of the trade-off algorithms [3, 6, 10, 12]. Simply put, a DP is a point in the one-way chain that satisfies a preset condition. Whereas the original DP idea had this condition fixed for all chains, VDP allows this condition to depend on the chain's starting point.

While VDP is a variant of the original DP, the two methods show very different characteristics. The simple idea of allowing the chain stopping condition to vary with the chains brings about unexpected consequences. It leads to the removal of the sorting procedure that was needed in the Hellman method's pre-computation phase. Another surprising characteristic is that, whereas all previous trade-off algorithms stored both ends of the one-way chains in the table, our method completely removes the need to store the starting point. This is because the chain end contains information about the chain beginning.

The second contribution of this paper is in successfully applying the DP (and VDP) idea to the rainbow table method. A combination of the DP technique with a variant of the rainbow table method was suggested in [4, 5], but there is a natural barrier to its combination with the original rainbow table method. The one-way chain created during the pre-computation phase is re-traced in the online phase in the opposite direction, and with rainbow tables, this is not possible unless the chain length is known. So techniques like DP that disturb the length of chains were thought to be incompatible with the rainbow table method. We have overcome this difficulty by employing a sorting that takes the chain lengths into account. Our new technique VDP can also be combined with the rainbow table method in a similar way.

The rest of the paper is organized as follows. We start by briefly reviewing some of the previous TMTO works. Then, in Section 3, the new VDP technique is presented. In Section 4, we show how to apply the DP and VDP ideas to rainbow table method. This is followed by Section 5, giving a rough comparison between various TMTO methods and explaining the issues involved in such a comparison. We summarize this paper in Section 6.

2 Previous works

In this section we will quickly review the theory of the time memory trade-offs, recalling the basic concepts and fixing notation. Readers new to the trade-off technique should refer to the original papers. For example, we shall not explain matters related to the success probability of these methods [11, 13, 14].

Throughout this paper, we fix a finite set $\mathbf{Z}_N = \{0, 1, \ldots, N-1\}$ of size N and we shall use $f : \mathbf{Z}_N \to \mathbf{Z}_N$ to denote the target one-way function that is to be inverted. The amount of memory needed to store a digest of the pre-computation is denoted by M and the online attack time is denoted by T.

All trade-off algorithms will involve parameters $t, m \in \mathbf{Z}_N$, satisfying $mt^2 = N$, known as the *matrix stopping rule*. We shall not be concerned with the exact choice of these numbers, which depends on the resources available to the attacker and also on his needs. All trade-off algorithms will involve a family of permutations, $r_i : \mathbf{Z}_N \to \mathbf{Z}_N$, called the *reduction functions*. The range of i will vary with each trade-off algorithm. Each of these defines an *iterating function* $f_i : \mathbf{Z}_N \to \mathbf{Z}_N$ through the equation $f_i(x) = r_i \circ f(x)$.

All trade-off algorithms consist of a pre-computation phase, in which tables are prepared, and an online phase. In the online phase, an inversion target $f(x_0)$ is given, to which the trade-off algorithm will return an X such that $f(X) = f(x_0)$. As f is not injective, there is no guarantee that $X = x_0$, and this has to be checked outside the trade-off algorithm. If X is found to be an unsatisfactory answer, a situation referred to as a *false alarm*, the trade-off algorithm is simply resumed.

2.1 Hellman trade-off

Hellman's original work [11] was presented as an attack on block ciphers, but we shall describe his trade-off algorithm as a generic inversion technique, applicable to any one-way function.

Pre-computation phase What is explained below shall be repeated t times, once for each i in the range $0 \le i < t$, to build t tables. We start by choosing, preferably distinct, m starting points, labeled $\mathrm{SP}^i_0, \mathrm{SP}^i_1, \ldots, \mathrm{SP}^i_{m-1}$. For each $0 \le j < m$, we set $X^i_{j,0} = \mathrm{SP}^i_j$ and compute

$$X^i_{j,k} = f_i(X^i_{j,k-1}) \qquad (1 \le k \le t),$$

recursively. This is said to be the *Hellman chain*. The ending point is written as EP^i_j, so that $\mathrm{EP}^i_j = f^t_i(\mathrm{SP}^i_j)$. All intermediate points of the Hellman chains are discarded and just the ordered pairs $\{(\mathrm{SP}^i_j, \mathrm{EP}^i_j)\}^{m-1}_{j=0}$ are stored as the i-th *Hellman table* $\mathtt{HT}_i$, after they have been sorted with respect to the ending points. A set of Hellman chains that was used to create a single table is referred to as the *Hellman matrix*, and this is depicted in Figure 1. Note that we have t tables, each containing m entries, so that the total storage cost is $M = mt$.

$$\begin{array}{rcccccccr}
\mathrm{SP}^i_0 = X^i_{0,0} & \xrightarrow{f_i} & X^i_{0,1} & \xrightarrow{f_i} \cdots\cdots & \xrightarrow{f_i} & X^i_{0,t-1} & \xrightarrow{f_i} & X^i_{0,t} = \mathrm{EP}^i_0 \\
\mathrm{SP}^i_1 = X^i_{1,0} & \xrightarrow{f_i} & X^i_{1,1} & \xrightarrow{f_i} \cdots\cdots & \xrightarrow{f_i} & X^i_{1,t-1} & \xrightarrow{f_i} & X^i_{1,t} = \mathrm{EP}^i_1 \\
\vdots & & & & & & & \vdots \\
\mathrm{SP}^i_{m-1} = X^i_{m-1,0} & \xrightarrow{f_i} & X^i_{m-1,1} & \xrightarrow{f_i} \cdots\cdots & \xrightarrow{f_i} & X^i_{m-1,t-1} & \xrightarrow{f_i} & X^i_{m-1,t} = \mathrm{EP}^i_{m-1}
\end{array}$$

Fig. 1. The i-th Hellman Matrix

Online phase Given a target point $f(x_0)$, the process below is repeated for each i. We first compute $Y^i_0 = r_i(f(x_0)) = f_i(x_0)$ and check if this appears as an ending point in the i-th Hellman table $\mathtt{HT}_i$. This table lookup is done for each recursively computed $Y^i_k = f_i(Y^i_{k-1})$, where $k = 1, 2, \ldots, t-1$. To distinguish this chain from the Hellman chain, in this paper, we shall refer to this Y^i_k-chain as the i-th *online chain*.

Whenever a match $Y^i_k = \mathrm{EP}^i_j$ is found, we compute $X = X^i_{j,t-k-1} = f^{t-k-1}_i(\mathrm{SP}^i_j)$. Since

$$f^k_i(f_i(X)) = f^{k+1}_i(X) = f^t_i(\mathrm{SP}^i_j) = \mathrm{EP}^i_j = Y^i_k = f^k_i(Y^i_0),$$

there is a large chance that $f_i(X) = Y^i_0$, which is equivalent to $f(X) = f(x_0)$, due to r_i being injective. In such a case, the algorithm returns X. But, as f^k_i is not injective, there could be a merge between the Hellman and online chains, and it is possible to have $f(X) \neq f(x_0)$. This is also referred to as a *false alarm*, in which case the next k is processed. Disregarding the time taken to process false alarms, it takes t iterations of f_i to process each of the t tables, so the online time is $T = t^2$.

Application of the matrix stopping rule to the online time $T = t^2$ and storage size $M = mt$ brings out the *Hellman trade-off curve* $TM^2 = N^2$. Conversely, any T and M satisfying the trade-off curve lead to parameters m and t appropriate for the Hellman trade-off algorithm.

2.2 Distinguished points

The distinguished point method was suggested by Rivest and issues concerning its practical use were investigated in [8, 15]. Rather than fixing the length of each Hellman chain, the iteration $X^i_{j,k} = f_i(X^i_{j,k-1})$ is continued until an $X^i_{j,k}$ satisfying a certain condition is found, and we obtain chains of varying lengths. For example, if one wants the average chain length to be $t = 2^d$, DP are typically defined to be points whose first d bits are all zero.

In practice, some of the chains created in this way could be too long for practical use, and some chains may even fall into a loop and never reach a DP. So we throw away chains longer than a preset $\hat{t} = t_{\max}$. If needed, the shortened average length can be adjusted by discarding chains shorter than a preset $\check{t} = t_{\min}$. The effects of $\hat{t}$ and $\check{t}$ are discussed in more detail in [15]. The length of each chain is usually recorded in the Hellman table so that they can be used when resolving false alarms.

The main advantage of using the DP method is in the reduction of table lookups made during the online phase. The generated point $Y^i_k = f^k_i(Y^i_0)$ can appear as an endpoint in the Hellman table only if it is a DP. So it suffices to do a search of the table only when the online chain reaches a DP, and the number of table lookups is reduced by a factor of 2^d.

As the ending point is the only DP in any Hellman chain, when a certain Y^i_k is found to be a DP, but not in $\mathtt{HT}_i$, the target cannot be in the i-th Hellman matrix, and one can move onto the next table. So the average length of chains generated online is expected to be about 2^d. In this paper, we shall refer to this trade-off method as Hellman+DP.

2.3 Rainbow table

The rainbow table method was introduced by Oechslin [14]. Instead of using a single reduction function for each table, t different reduction functions are sequentially used in each chain of length t to generate a single table. Explicitly, the j-th *rainbow chain* is generated by iterating $X_{j,k+1} = f_{k+1}(X_{j,k})$, and we allow j to run in the range[1] $0 \le j < mt$. As with the Hellman method, $\{(\mathrm{SP}_j, \mathrm{EP}_j)\}$ is stored in the rainbow table $\mathtt{RT}$, after sorting.

In the online phase, for each $0 \le k < t$, the k-th online chain

$$r_{t-k}(f(x_0)) \xrightarrow{f_{t-k+1}} \circ \xrightarrow{f_{t-k+2}} \cdots\cdots \circ \xrightarrow{f_{t-1}} \circ \xrightarrow{f_t} Y^k$$

is computed and Y^k is searched for among the second component of the rainbow table. Thus the online time of the rainbow table method is $T = \frac{1}{2}t^2$, and this is one half of the original Hellman method, when the two are storing the same number of entries. The rainbow table contains $M = mt$ entries, and the rainbow trade-off curve is given by $TM^2 = \frac{1}{2}N^2$.

[1] This is non-restrictive choice that allows a direct comparison between Hellman and rainbow methods.

2.4 Checkpoints

Experiments show that a considerable fraction of the online time is spent in resolving false alarms. The checkpoint method [2] was introduced to solve this problem. It allows recognition of false alarms without the costly regeneration of the Hellman/rainbow chains.

A small number of positions in the Hellman/rainbow chain are designated as checkpoints. During the pre-computation phase, a simple checksum[2] of the points at these checkpoints are additionally stored. During the online phase, when a matching $Y^i_k \in \mathtt{HT}_i$ or $Y^k \in \mathtt{RT}$ is found, checkpoint checksums from the online chain are compared with the stored values, and the match is taken to be a false alarm if there are any discrepancies. The checkpoint method is applicable to all three trade-off algorithms we have described.

3 Variable distinguished points

In this section, we propose a new technique, named the *variable distinguished point* (VDP) method, which is a variant of the DP method, but with very different properties.

3.1 The basic idea

As with the original DP method, our VDP method terminates a Hellman chain when a point satisfying a certain relation is reached. The crucial difference is that, unlike DP, we allow this condition to depend on the starting point of the chain. This results in the ending point containing information about the starting point, so that by using information which is common to the table, one may be able to recover the starting point from the ending point.

While the main objective of the original DP method was to reduce the number of table searches, the VDP method aims to eliminate the need to store the starting points so as to lessen storage requirements.

3.2 Applying VDP to Hellman trade-offs

Let us show how we may apply the VDP technique to the original Hellman trade-off algorithm. To simplify our discussion, we shall restrict to the typical parameters $m = t = N^{\frac{1}{3}}$ and set $d = \frac{1}{3}\log_2 N$. Ways to use more general parameters will be dealt with in Section 3.4.

When creating the j-th Hellman chain of the i-th table, we take our starting point to be

$$\mathrm{SP}^i_j = (0 \,\|\, i \,\|\, j),$$

where each of the three concatenated components are of d bits. The Hellman chain is created as usual through iterated computation of $X^i_{j,k} = f_i(X^i_{j,k-1})$,

[2] In practice, the checksum is a single bit from the point. With k such checkpoints, the probability of having an undetected false alarm could be as low as $\frac{1}{2^k}$.

starting from $X^i_{j,0} = \mathrm{SP}^i_j$, but it is terminated only when the most significant d bits of some $X^i_{j,k}$ is found to be j. Chains longer than a preset $\hat{t} = t_{\max}$ are discarded. The ending point EP^i_j we have reached in the j-th chain is stored at $\mathtt{HT}_i\mathtt{[j]}$, the j-th position of the i-th Hellman table. There is no table sorting involved. Since the chain length is variable, storing chain length information would reduce online time spent dealing with false alarms, but this is not mandatory. We remark that if we take the first d bits of an ending point as its hash[3] value, then the Hellman+VDP table we have created can be seen as a perfect hash table.

Notice that since the storage position index j is equal to the first d bits of EP^i_j and also to the most meaningful part of SP^i_j, neither the starting point nor the first d bits of the ending point need to be stored. The pre-computation phase of Hellman trade-off with VDP, under restricted parameters $m = t = N^{\frac{1}{3}}$, is summarized in Algorithm 1.

Algorithm 1 Pre-computation Phase of Hellman+VDP

Require:
(1) parameters m, t, and $\hat{t} = t_{\max}$.
(2) functions $f_i = r_i \circ f$ $(i = 0, \ldots, t-1)$.
(3) empty tables $\mathtt{HT}_0, \mathtt{HT}_1, \cdots, \mathtt{HT}_{t-1}$.
Ensure:
(1) $m = t = N^{\frac{1}{3}} = 2^d$.

```
1: for i = 0,...,t-1 and j = 0,...,m-1 do
2:     X ← (0 ‖ i ‖ j)                                  ▷ X = SP_j^i
3:     for k = 1 to t̂ do
4:         X ← f_i(X)                                   ▷ iterate Hellman chain
5:         if j == (significant d bits of X) then       ▷ check for DP
6:             HT_i[j] ← (less significant (n − d) bits of X)‖k
                              ▷ HT_i[j] = (lower part of EP_j^i)‖(chain length)
7:             break                  ▷ process next chain if DP is reached
8:         end if
9:     end for                ▷ leave HT_i[j] empty if t̂ is reach before a DP
10: end for
11: return HT_0, HT_1, ···, HT_{t-1}.
```

The online phase of the simplified version of VDP is given in Algorithm 2. When we want to check whether a point from the online chain is an ending point, we can look up the table entry at the position given by the point's first d bits. There is no searching involved. If we find a match, the corresponding starting point can be recovered using the table number and the position index.

[3] We are referring to the data structuring method and not to cryptographic hash functions.

Algorithm 2 Online Phase of Hellman+VDP

Require:
(1) target $f(x_0)$
(2) parameters t, $\hat{t} = t_{\max}$ and functions $f_i = r_i \circ f$ $(i = 0, \ldots, t-1)$
(3) pre-computed Hellman+VDP tables $\mathtt{HT}_0, \mathtt{HT}_1, \cdots, \mathtt{HT}_{t-1}$
Ensure:
(1) $m = t = N^{\frac{1}{3}} = 2^d$.

for $i = 0, \ldots, t-1$ **do**
  $Y \leftarrow r_i(f(x_0))$
  for $k = 1$ to $\hat{t}$ **do**
    $j \leftarrow$ (significant d bits of Y)
    $\mathrm{EP} \,\|\, l \leftarrow \mathtt{HT}_i\mathtt{[j]}$ ▷ $\mathtt{HT}_i\mathtt{[j]} =$ (lower bits of EP^i_j)‖(chain length)
    if $k < l$ and EP `==` (less significant $(n-d)$ bits of Y) **then**
      $X \leftarrow f_i^{l-k-1}(0 \,\|\, i \,\|\, j)$ ▷ $X = f_i^{l-k-1}(\mathrm{SP}^i_j)$
      if $f(X)$ `==` $f(x_0)$ **then** ▷ check for false alarm
        return X ▷ return pre-image of $f(x_0)$
      end if
    end if
    $Y \leftarrow f_i(Y)$ ▷ iterate online chain
  end for
end for
return 'failure'

3.3 Technical details of Hellman+VDP

To maintain the success probability provided by the original Hellman tradeoff, we need to ensure that our Hellman+VDP method results in average chain length of approximately $t = 2^d$ and that not too many of the Hellman table entries are left empty.

When the VDP is defined using d bits, the average chain length would naturally become $t = 2^d$, except that we are throwing away some of the longer chains. The first issue can be approached, as with the original Hellman+DP method, by setting an appropriate lower bound $\check{t} = t_{\min}$ in addition to the upper bound $\hat{t} = t_{\max}$ for chain lengths. But whereas the DP method may simply throw away chains not falling within these bounds and generate more chains from other starting point, with the VDP method, there are no other starting points that can be used, and every discarded chain would imply an empty Hellman table entry.

A solution to the empty table entry problem is to use starting points of the form

$$\mathrm{SP}^i_j = \tau \,\|\, i \,\|\, j\,,$$

where τ is a counter that is incremented every time creation of the j-th chain fails. Now, for reconstruction of SP^i_j during the online phase to be possible, the τ value will need to be stored, so we place a restriction on the size of τ. We allow at most 2^s trials to be done for the j-th chain, and if all trials fail, we use the longest chain among those shorter than $\check{t}$. The entry $\texttt{HT}_i\texttt{[j]}$ is left empty only if all the 2^s chains were longer than $\hat{t}$.

For the parameters $t = 2^d$ and $\hat{t} = c \cdot t$, the probability of generating a chain longer than $\hat{t}$ is

$$\Big(1 - \frac{1}{2^d}\Big)^{\hat{t}} \approx \exp\big(-\frac{\hat{t}}{2^d}\big) = \exp(-c).$$

With the use of s-many extra bits per table entry, the probability of a table position $\texttt{HT}_i\texttt{[j]}$ being left empty would become as small as $\exp(-c \cdot 2^s)$. When $m = N^{\frac{1}{3}}$ chains are used for each Hellman table, by choosing s to satisfy $m \cdot \exp(-c \cdot 2^s) < 1$, or equivalently,

$$\frac{\log\log N - \log 3c}{\log 2} < s,$$

we can expect to find less than a single empty entry from each table, resulting in a minimal perfect hash table. Note that the above bound on s is certainly small and asymptotically negligible when compared to the number of bits needed for the other major parts. Also, the attacker may choose to use an even smaller s according to his needs.

The small number of empty entries can be marked by writing zero as the chain length, or through use of one additional bit, when the chain length is not recorded. One may even choose to fill it with random value and let it generate false alarms at the worst. A typical Hellman+VDP table is depicted in Figure 2.

$$\begin{array}{lcl|l|}
\mathrm{SP}_0^i = (\tau_0 \,\|\, i \,\|\, 0) \xrightarrow{f_i^{l_0}} (0 \,\|\, \mathcal{R}_0) = \mathrm{EP}_0^i & \rightsquigarrow & & \texttt{HT}_i\texttt{[0]} = \{\mathcal{R}_0, \tau_0, l_0\} \\
\mathrm{SP}_1^i = (\tau_1 \,\|\, i \,\|\, 1) \xrightarrow{f_i^{l_1}} (1 \,\|\, \mathcal{R}_1) = \mathrm{EP}_1^i & \rightsquigarrow & & \texttt{HT}_i\texttt{[1]} = \{\mathcal{R}_1, \tau_1, l_1\} \\
\vdots & & & \vdots \\
(\mathrm{EP}_j^i \text{ generation failure}) & \rightsquigarrow & & \texttt{HT}_i\texttt{[j]} = \{\,-\,,\,-\,,\,0\,\} \\
\vdots & & & \vdots \\
\mathrm{SP}_{m-1}^i = (\tau_{m-1} \| i \| m-1) \xrightarrow{f_i^{l_{m-1}}} (m-1 \| \mathcal{R}_{m-1}) = \mathrm{EP}_{m-1}^i & \rightsquigarrow & & \texttt{HT}_i\texttt{[m-1]} = \{\mathcal{R}_{m-1}, \tau_{m-1}, l_{m-1}\}
\end{array}$$

Fig. 2. Typical i-th Hellman Table for Hellman+VDP

Note that the ending point is the only DP within that chain. So if an online chain reaches a DP for the j-th Hellman chain and EP_j^i is found to be a non-match, the inversion target $f(x_0)$ cannot belong to the j-th Hellman chain. There is no reason to refer to the j-th chain any further, even if the online chain reaches another DP for the same chain. So by keeping track of which ending points have been processed, one may reduce the number of table lookups and the chance of false alarms.

One undesirable property of the VDP method concerns its online time. We need to go through t tables with each table costing $\hat{t}$ iterations of f_i, so that the online time is $T = t \cdot \hat{t}$. Hence, one would wish to choose $\hat{t}$ to be as small as possible. But reducing $\hat{t}$ must be paired with an increase in $\check{t}$, and this has the effect of increasing the pre-computation time, if the success probability is to be maintained.

There is one trick that can be used to reduce online time, with no change given to the pre-computation phase, and at a very small cost in success probability. One can simply move onto the next Hellman table a little before the online chain length reaches $\hat{t}$. The effect of this on the inversion success probability will be small due to two reasons. As was mentioned before, after the first lookup of the ending point EP^i_j, the whole j-th chain may be disregarded. The second reason is that all chains shorter than the online chain generated so far may be disregarded. So, only the long chains that have not yet been referred to has any chance of containing the inversion target. Even for these chains, points that are closer to the ending point than the length of currently generated online chain cannot be the target of our inversion. Thus once the online chain reaches a certain length, most of the Hellman matrix has already been searched for and skipping the rest should have only a small effect on the success probability.

The above argument shows that the online phase of Hellman+VDP is much more efficient at the start of each table processing than at the end. This is somewhat similar to a characteristic of the rainbow table method and can be very advantageous when the online phase of Hellman+VDP is carefully scheduled.

3.4 Using general parameters

So far, we have only worked with parameters $m = t = N^{\frac{1}{3}}$. Even though the average length of the Hellman chains can be slightly adjusted by changing the bounds $\check{t} = t_{\max}$ and $\hat{t} = t_{\min}$, other measures are needed when we want parameters m and t to differ by a large factor.

Suppose the parameters m and t obtained from the matrix stopping rule $mt^2 = N$ satisfy $t = 2^r m$. To explain how to use these parameters, let us write $m = 2^d$ and $t = 2^{d+r}$. As before, we can set $\mathrm{SP}^i_j = (0 \,\|\, i \,\|\, j)$ for the i-th table, where 0 and i are of $d + r$ bits and j is of d bits. It only remains to bring the average chain length to $t = 2^{d+r}$. This is easily done by defining the DP for the j-th chain to be those points starting with $j\|0$, where j is of d bits and 0 is of r bits. In a way, this can be seen as a combination of the DP and VDP methods.

We next consider the opposite $m = 2^r t$ case. Let us write $m = 2^{d+r}$ and $t = 2^d$. The starting point for the j-th chain in the i-th table is set to $\mathrm{SP}^i_j = (0 \,\|\, i \,\|\, j)$, where 0 and i are of d bits and j is of $d + r$ bits. A point in the j-th chain is regarded as a DP if its most significant d bits are equal to the most significant d bits of j. Since this *distinguisher* does not contain enough information to fully distinguish between the possible j, ending points corresponding to the same significant j parts are sorted before storage, and r bits of the starting point are also stored. This is not as satisfactory as the previous case, but still reasonable unless r is large. In any case, this can be seen as having placed more of the original Hellman flavor back into Hellman+VDP.

To deal with m and t that are not powers of 2, one can use their closest powers and also utilize $\hat{t}$ and $\check{t}$ for fine adjustments.

4 Applying DP and VDP to the rainbow table method

The rainbow table method applies a different function f_i to every column in its chain creation. During the online phase, creation of the online chain proceeds in a backward direction, and having a fixed chain length is crucial in knowing which f_i to use. In this section, we show that by sorting the table in a slightly different way, it is possible to use rainbow chains of variable lengths.

4.1 Rainbow+DP

As with the Hellman+DP situation, we can use $\hat{t} = t_{\max}$ and $\check{t} = t_{\min}$ to adjust the average chain length. With these numbers fixed, we choose $\hat{t}$ reduction functions defining the iterating functions f_i. The rainbow chains are generated as with the original rainbow table method, setting $X_{j,0} = \mathrm{SP}_j$ and iteratively computing $X_{j,k} = f_k(X_{j,k-1})$. The chain is terminated when a DP is reached, and the starting point, the ending point, and the chain length are stored. So far, we have simply combined the rainbow table method with the DP technique.

Now, the rainbow table is sorted first with respect to chain lengths and then with respect to the ending points within those chains of same length. If collisions are found among those of the same length, one may optionally discarding all but one of them and generating more chains to take their places. Note that collision of ending points between chains of different lengths have minimal effect as they do not correspond to collision within a rainbow matrix column. Also, no collision within a rainbow matrix column is undetected, since any such colliding chains would end at the same length.

The sorted data is stored in separate tables $\mathtt{RT}_{\check{t}}$, $\mathtt{RT}_{\check{t}+1}$, …, $\mathtt{RT}_{\hat{t}}$, indexed by the length of chains they correspond to and the length data within each entry are discarded. It is also possible to store the whole data as one table together with an index file containing the starting positions for each length. The resulting rainbow+DP matrix of sorted rainbow chains is depicted in Figure 3.

During the online phase, we search through this matrix from right to left, and, within each column, from top to bottom. Given a target $f(x_0)$, we first compute $Y^1_{\hat{t}} = r_{\hat{t}}(f(x_0))$. If this is a DP, it is searched for among the ending points of $\mathtt{RT}_{\hat{t}}$. In the next step, we search for $Y^2_{\hat{t}-1} = r_{\hat{t}-1}(f(x_0))$ and $Y^2_{\hat{t}} = f_{\hat{t}}(Y^2_{\hat{t}-1})$ in $\mathtt{RT}_{\hat{t}-1}$ and $\mathtt{RT}_{\hat{t}}$, respectively, if any of them are DP.

In the j-th iteration, starting from $Y^j_{\hat{t}-j+1} = r_{\hat{t}-j+1}(f(x_0))$, the online chain

$$Y^j_{\hat{t}-j+1} \xrightarrow{f_{\hat{t}-j+2}} Y^j_{\hat{t}-j+2} \xrightarrow{f_{\hat{t}-j+3}} \cdots\cdots \xrightarrow{f_{\hat{t}-1}} Y^j_{\hat{t}-1} \xrightarrow{f_{\hat{t}}} Y^j_{\hat{t}}$$

$$
\begin{array}{ll}
\mathtt{RT}_{\check{t}} & \boxed{\begin{array}{l}
\mathrm{SP}_{\check{t},0} \xrightarrow{f_1} \circ \xrightarrow{f_2} \cdots\cdots \circ \xrightarrow{f_{\check{t}}} \mathrm{EP}_{\check{t},0} \\
\mathrm{SP}_{\check{t},1} \xrightarrow{f_1} \circ \xrightarrow{f_2} \cdots\cdots \circ \xrightarrow{f_{\check{t}}} \mathrm{EP}_{\check{t},1} \\
\vdots \qquad\qquad\qquad\qquad\qquad\quad \vdots
\end{array}} \\
\vdots & \qquad\qquad\qquad \vdots \qquad\qquad\qquad\qquad \ddots \\
\mathtt{RT}_{\hat{t}-1} & \boxed{\begin{array}{l}
\mathrm{SP}_{\hat{t}-1,0} \xrightarrow{f_1} \circ \xrightarrow{f_2} \cdots\cdots \circ \xrightarrow{f_{\check{t}}} \cdots\cdots \circ \xrightarrow{f_{\hat{t}-1}} \mathrm{EP}_{\hat{t}-1,0} \\
\mathrm{SP}_{\hat{t}-1,1} \xrightarrow{f_1} \circ \xrightarrow{f_2} \cdots\cdots \circ \xrightarrow{f_{\check{t}}} \cdots\cdots \circ \xrightarrow{f_{\hat{t}-1}} \mathrm{EP}_{\hat{t}-1,1} \\
\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \vdots
\end{array}} \\
\mathtt{RT}_{\hat{t}} & \boxed{\begin{array}{l}
\mathrm{SP}_{\hat{t},0} \xrightarrow{f_1} \circ \xrightarrow{f_2} \cdots\cdots \circ \xrightarrow{f_{\check{t}}} \cdots\cdots \circ \xrightarrow{f_{\hat{t}-1}} \circ \xrightarrow{f_{\hat{t}}} \mathrm{EP}_{\hat{t},0} \\
\mathrm{SP}_{\hat{t},1} \xrightarrow{f_1} \circ \xrightarrow{f_2} \cdots\cdots \circ \xrightarrow{f_{\check{t}}} \cdots\cdots \circ \xrightarrow{f_{\hat{t}-1}} \circ \xrightarrow{f_{\hat{t}}} \mathrm{EP}_{\hat{t},1} \\
\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots
\end{array}}
\end{array}
$$

Fig. 3. Sorted Rainbow Matrix for Rainbow+DP

is computed. While computing, if we come across a $Y^j_{\hat{t}-j+k}$, which is a DP and we have $\hat{t} - j + k \geq \check{t}$, it is searched for among the ending points of $\mathtt{RT}_{\hat{t}-j+k}$. Whenever a DP is reached, the rest of that online chain is skipped and we proceed with the next iteration.

Our j-th iteration requires $j - 1$ function iterations at the worst, and we go through $\hat{t}$ iterations. So the worst case online time is approximately $T = \frac{\hat{t}^2}{2}$ and the number of table searches will be $\hat{t}$ at the most. On average, for $\hat{t} = c{\cdot}t$, we can expect $\hat{t} - (1 - \frac{1}{e^c})t$ table searches and online time of $t \cdot (\hat{t} - (1 - \frac{1}{e^c})t)$. This is explained in Appendix A.

4.2 Rainbow+VDP

To simplify discussion, we take parameters $m = t = 2^d = N^{\frac{1}{3}}$. We fix $\check{t}$ and $\hat{t}$, and choose $\hat{t}$ reduction functions. We take $mt = N^{\frac{2}{3}}$ starting points of the form $\mathrm{SP}_{i,j} = (0 \,\|\, i \,\|\, j)$, where both i and j run over all d-bit values. The distinguished points for the $\mathrm{SP}_{i,j}$-chain are defined to be points with most significant d bits equal to i. So, chains can be split into groups of size 2^d according to their definition of DP.

Sorting is done within each DP group with respect to chain lengths.[4] We write them in 2^d tables indexed by their DP definition, which is also equal to the center d bits of the starting points they contain and also to the first d bits of ending points. Unlike rainbow+DP, the chain length may not be discarded.

The online phase proceeds as with rainbow+DP, except that every computed Y^j_k is now a DP for some definition of DP. So after each computation of Y^j_k, the

[4] It is also possible to sort exactly as with rainbow+DP and obtain similar results.

corresponding table is checked for a chain of correct length and matching ending point. The worst case online time of rainbow+VDP is about $\frac{\hat{t}^2}{2}$ and this is also the number of table searches needed.

5 Trade-off curves and storage issues

In this section, we make a very rough comparison of the various trade-off algorithms and look into some storage optimization techniques, which can complicate any serious attempt at comparison.

5.1 Trade-off curves

Let us make a very rough comparison of the trade-off algorithms we have discussed in this paper. The relevant facts are summarized in Table 1.

Table 1. Comparison of trade-off algorithms ($mt^2 = N$, $\hat{t} = c{\cdot}t$)

	table entries (M)	table lookups	function evaluations (T)	trade-off curve
Hellman	mt	t^2	t^2	$TM^2 = N^2$
Rainbow	mt	t	$\frac{1}{2}t^2$	$TM^2 = \frac{1}{2}N^2$
Hellman+DP	mt	t	t^2	$TM^2 = N^2$
Hellman+VDP	mt	$t \cdot \hat{t}$	$t \cdot \hat{t}$	$TM^2 = cN^2$
Rainbow+DP	mt	$\hat{t}$	$\frac{1}{2}\hat{t}^2$	$TM^2 = \frac{c^2}{2}N^2$
Rainbow+VDP	mt	$\frac{1}{2}\hat{t}^2$	$\frac{1}{2}\hat{t}^2$	$TM^2 = \frac{c^2}{2}N^2$

The table assumes that the parameters m and t satisfy the matrix stopping rule and that we have $\hat{t} = t_{\max} = c \cdot t$. We have listed the total number of pre-computed table entries M and the number of online f_i iterations T. The presented time complexity T disregards false alarms, and corresponds to the worst case, rather than the average case. In any real world use of a trade-off algorithm, there is a practical limit to how much can be loaded onto fast memory, and with bigger tables, access speed of the cheaper and slower storage becomes an important factor of the online time. So we have also given the number of online table searches.

The last column contains the trade-off curves satisfied by T and M. A hasty conclusion from this column alone would be that the rainbow method is the best and that the three algorithms we have introduced are inferior. But this does not seem to be the correct picture. For example, it is argued in [5] that each entry of a rainbow table demands about twice as many bits than that of a Hellman table. So, with equal amount of storage, the Hellman method would be faster by a factor of two, contrary to the naïve interpretation. This shows that finding the optimum number of bits to use for each table entry is crucial in comparing these algorithms.

Another issue in interpreting the above table concerns success probability, which is believed to be somewhat higher than 50% for all of the above algorithms. There are arguments giving lower bounds or expected values for the success rate, but this is not a very well understood subject, and a fair comparison of the algorithms should compare them at the same success rate. A related issue is how much pre-computation is needed to achieve this success rate.

For now, we can only state that the algorithms we have suggested are roughly the same in performance to the previous trade-off algorithms. There will be situations where one of the above algorithms is more suitable than the others, but any difference of performance between them will be by a small multiplicative factor.

Deferring a more exact and fair comparison between trade-off algorithms to a future work, in the next subsection, we will take a closer look into the complexity of finding the optimum number of bits to be allocated to a single entry.

5.2 Table optimization

When using the Hellman trade-off, by setting the starting point to $(0 \| i \| j)$, one may store just j, and not i, which is common to $\mathtt{HT}_i$. There are many other techniques for reducing storage that need to be considered. But, as these have side effects, such as more false alarms, their use is not simple.

We have tested some of the techniques discussed below, using Hellman+VDP, and the result is given in Appendix B. Although the tests are not conclusive about the optimum choice, it shows that the following techniques should be considered.

Hash table Hash table is a way of structuring data in such a way that table searches take constant time. The idea is to use a simple function of the data as the address in which to store a given data. As explained in [7] and [2], this can also make ending point storage more efficient. One can save up to $\log(m)$ bits per entry from a table containing m entries. When seen as a hash table, our VDP method reaches this limit.

Ending point truncation If a table contains m entries, we need at least $\log(m)$ bits to distinguish between the entries. If we expect to do l table lookups, we would want $\log(l)$ additional bits to filter out most of the accidental matches. Hence, simply leaving only $\log(ml)$ bits from the ending point could be an option. Depending on the trade-off algorithm, this method may or may not give additional savings, when used in conjunction with the hash table savings.

Storing chain lengths With trade-off algorithms producing chains of variable length, as indicated in [15], storage of chain lengths reduces effort spent on false alarms. On the other hand, this requires $\log(t_{\max} - t_{\min})$ additional bits per table entry and we believe this may not be as cost effective as believed.

By storing only a few significant bits of the lengths or by simply not storing the lengths, we may increase the time spent on each false alarm, but the saved memory could be used to hold more $(\mathrm{SP}, \mathrm{EP})$ pairs and lead to smaller online f_i iterations. Moreover, proper use of the checkpoint method [2] may resolve the false alarm issue with just few extra bits per entry.

6 Conclusion

In this paper we suggested a new time memory trade-off technique named variable distinguished points (VDP), and showed how to combine the rainbow table method with the DP and VDP ideas.

The original Hellman trade-off terminated a pre-computation chain when it reached a certain fixed length, whereas chain termination in the DP method was taken when a chain element satisfied some preset condition. Our VDP method generalizes DP by allowing the termination condition itself to vary with each chain. The properties of VDP are very different from those of DP, requiring no sorting and aiming to reduce storage, rather than the number of table lookups.

Our combination of the rainbow method and DP or VDP, though simple, is also a nontrivial result. The changing reduction function of the rainbow method and varying chain length of the DP method presented a barrier to this combination, and the only known successful attempt [4, 5] was on a rainbow variant that uses repeating patterns of reduction functions. We have shown that it is possible to overcome this barrier through a different sorting.

The performance of our algorithms is on a par with that of the previous trade-off algorithms, and thus there are more candidates to be considered with time memory trade-offs, than what was known before.

References

1. 3GPP TS 35.202 V7.0.0 (2007-06), Kasumi specification. Available from `http://www.3gpp.org`.
2. Gildas Avoine, Pascal Junod, and Philippe Oechslin. Time-Memory Trade-Offs: False Alarm Detection Using Checkpoints. Progress in Cryptology, proceedings of Indocrypt 2005, LNCS 3797, Springer-Verlag, pp. 183–196, 2005.
3. Steve Babbage. Improved "Exhaustive Search" Attacks on Stream Ciphers. 'European Convention on Security and Detection', Conference publication No. 408, pp. 161–166, IEE, 1995.
4. Elad Barkan. Cryptanlysis of Ciphers and Protocols, Ph. D. Thesis. `http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi?2006/PHD/PHD-2006-04`, 2006.

5. Elad Barkan, Eli Biham, and Adi Shamir. Rigorous Bounds on Cryptanalytic Time/Memory Tradeoffs. Advances in Cryptology, proceedings of Crypto 2006, LNCS 4117, Springer-Verlag, pp. 1–21, 2006.
6. Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. Advances in Cryptology, proceedings of Asiacrypt 2000, LNCS 1976, Springer-Verlag, pp. 1–13, 2000.
7. Alex Biryukov, Adi Shamir, and David Wagner. Real Time Cryptanalysis of A5/1 on a PC. Fast Software Encryption, proceedings of FSE 2000, LNCS 1978, Springer-Verlag, pp. 1–18, 2001.
8. Johan Borst, Bart Preneel, and Joos Vandewalle. On the Time-Memory Tradeoff Between Exhaustive Key Search and Table Precomputation. proceedings of the 19th Symposium in Information Theory in the Benelux, WIC, 1998, pp.111–118.
9. Dorothy E. Denning. Cryptography and Data Security, p.100. Addison-Wesley, 1982.
10. Jovan Dj. Golic. Cryptanalysis of Alleged A5 Stream Cipher. Advances in Cryptology, proceedings of Eurocrypt 1997, LNCS 1233, Springer-Verlag, pp. 239–255, 1997.
11. Martin E. Hellman. A Cryptanalytic Time-Memory Trade-Off. IEEE Transactions on Information Theory, Vol. IT-26, No. 4, pp. 401–406, 1980.
12. Jin Hong and Palash Sarkar. New Applications of Time Memory Data Tradeoffs. Advances in Cryptology, proceedings of Asiacrypt 2005, LNCS 3788, Springer-Verlag, pp. 353–372, 2005.
13. Il-Jun Kim and Tsutomu Matsumoto. Achieving Higher Success Probability in Time-Memory Trade-Off Cryptanalysis without Increasing Memory Size. IEICE Transactions on Fundamentals, Vol. E82-A, No. 1, pp. 123–129, 1999.
14. Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. Advances in Cryptology, proceedings of Crypto 2003, LNCS 2729, Springer-Verlag, pp. 617–630, 2003.
15. François-Xavier Standaert, Gaël Rouvroy, Jean-Jacques Quisquater, and Jean-Didier Legat. A Time-Memory Tradeoff Using Distinguished Points: New Analysis & FPGA Results. In Workshop on Cryptographic Hardware and Embedded Systems, proceedings of CHES 2002, LNCS 2523, Springer-Verlag, pp. 593–609, 2003.

A The online complexity of rainbow+DP

We shall compute the expected number of table lookups and also the number of one-way function iterations needed during the online phase of rainbow+DP. We are referring to the average values, and not the worst case situations.

Let us write $\hat{t} = t_{\max} = c \cdot t$. We know that the approximation

$$\left(1 - \frac{1}{t}\right)^{\hat{t}} \approx \frac{1}{e^c}$$

holds true for any large t. Below, we shall also approximate $(1 - \frac{1}{t})^{\hat{t}-1}$ and $(1 - \frac{1}{t})^{\hat{t}-2}$ by $\frac{1}{e^c}$.

The expected number of table searches for rainbow+DP can be calculated as follows.

$$\begin{aligned}\sum_{j=1}^{\hat{t}}\sum_{k=1}^{j}\Big(1-\frac{1}{t}\Big)^{k-1}\cdot\frac{1}{t} &= \sum_{j=1}^{\hat{t}}\bigg(1-\Big(1-\frac{1}{t}\Big)^{j}\bigg)\\ &\approx \hat{t}-t\Big(1-\frac{1}{t}\Big)\Big(1-\frac{1}{e^c}\Big)=\hat{t}-t\Big(1-\frac{1}{e^c}\Big)+\Big(1-\frac{1}{e^c}\Big).\end{aligned}$$

Neglecting the small constant term, the expected number of table searches can be written as

$$\hat{t}-\Big(1-\frac{1}{e^c}\Big)t. \tag{1}$$

In fact, the above argument does not take into account the fact that DP appearing during the online phase is searched for in the corresponding table only when $\hat{t}-j+k+1>t_{\min}$. So equation (1) is a good upper bound for the expected number of table searches.

The expected number of one-way function iterations taken during the creation of the j-th online chain of rainbow+DP can be computed as

$$0\cdot\frac{1}{t}+1\cdot\Big(1-\frac{1}{t}\Big)\cdot\frac{1}{t}+\cdots+(j-2)\Big(1-\frac{1}{t}\Big)^{j-2}\cdot\frac{1}{t}+(j-1)\Big(1-\frac{1}{t}\Big)^{j-1}.$$

The expected number of total online one-way function iterations is a sum of this over the range $j=1,2,\dots,\hat{t}$. Let us first approximate the sum over the last term.

$$\sum_{j=1}^{\hat{t}}(j-1)\Big(1-\frac{1}{t}\Big)^{j-1}\approx\Big(1-\frac{1}{e^c}\Big)t^2-\frac{1}{e^c}\cdot\hat{t}\cdot t+\Big(\frac{2}{e^c}-1\Big)t. \tag{2}$$

The remaining terms can be approximated as follows.

$$\begin{aligned}\sum_{j=3}^{\hat{t}}\sum_{k=1}^{j-2}k\cdot\Big(1-\frac{1}{t}\Big)^{k}\cdot\frac{1}{t} &= \sum_{j=3}^{\hat{t}}\bigg[(t-1)\Big\{1-\Big(1-\frac{1}{t}\Big)^{j-2}\Big\}-(j-2)\Big(1-\frac{1}{t}\Big)^{j-1}\bigg]\\ &\approx(\hat{t}-2)(t-1)-(t-1)^2\Big(1-\frac{1}{e^c}\Big)-t^2\Big(1-\frac{1}{t}\Big)^2\Big(1-\frac{1}{e^c}\Big)+t(\hat{t}-2)\frac{1}{e^c}.\end{aligned}$$

Recalling $\hat{t}=ct$, we collect the terms of highest order in t, from this and equation (2), to write the expected online time as

$$T=t\cdot\Big\{\hat{t}-\Big(1-\frac{1}{e^c}\Big)t\Big\}. \tag{3}$$

We remark that, as can be seen by comparing (1) and (3), by applying DP to the rainbow table method, we have decreased the number of table searches by a factor of t.

B VDP Test Result

We implemented Hellman+VDP algorithm and analyzed the worst case performance. The worst case is when we carry through the online phase to the end, regardless of whether we have found the correct inverse.

Our one-way function f is built from a reduced version of KASUMI [1], a 64-bit block and 128-bit key block cipher, used in the 3rd generation GSM phones. The 3GPP specification gives the 128-bit key $CK = k\|k$ as a double copy of a 64-bit key. We set our search space size to $N = 2^{36}$ by fixing the first 28 bits of k to zero. Our one-way function maps a 36-bit key to the 36-bit truncation of the KASUMI encryption corresponding to the fixed all-zero plaintext.

The parameters are set to $m = t = 2^{12}$ with $t_{\max} = 2t$ and $t_{\min} = \frac{1}{2}t$. The pre-computation phase for Hellman+VDP required $1.7N$ iterations of our one-way function and the average length of chains was 4147, which is approximately t. Two bits were allocated to the τ value, and out of the 2^{12} entries for each table, 1.38 entries were left empty on average.

We experimented with four different ways of storing the entries in a table. Tests were done with and without the chain length information in the table, and also with different bits allocated to storing ending point information. That is, we compared the performance of algorithm as explained in Section 3.2 and Section 3.3, which allocates 24 bits to ending point information storage, against when 8 bits were simply removed from this ending point information. We did not use checkpoints. The results are summarized in Table 2.

Table 2. Hellman+VDP test ($N = 2^{36}, m = t = 2^{12}$)

	with length		without length	
	24-bit EP	16-bit EP	24-bit EP	16-bit EP
number of FA	3,013	3,232	3,010	3,232
f iterations due to FA	3,341,590	3,779,138	8,995,406	9,771,395
total f iterations	36,891,926	37,329,474	42,545,742	43,321,731
memory	81.78 MB	65.01 MB	56.62 MB	39.84 MB

As was expected, the removal of length information increased total computation through the increased effort in resolving false alarms (FA), and the ending point truncation did increase f iterations by a small amount. But, for both of these actions, the increase in total computation was minimal in comparison to the reduction in storage. This is even more evident when the trade-off curve $TM^2 = N^2$ is considered, as we can see that any small amount of spare memory can return as a large reduction in online time.