A New Cryptanalytic Time/Memory/Data Trade-off Algorithm

Sourav Mukhopadhyay and Palash Sarkar
Cryptology Research Group
Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road, Kolkata
India 700108
e-mail:{sourav_t,palash}@isical.ac.in

Abstract

In 1980, Hellman introduced a time/memory trade-off (TMTO) algorithm satisfying the TMTO curve $TM^2 = N^2$, where T is the online time, M is the memory and N is the size of the search space. Later work by Biryukov-Shamir incorporated multiple data to obtain the curve $TM^2D^2 = N^2$, where D is the number of data points. In this paper, we describe a new table structure obtained by combining Hellman's structure with a structure proposed by Oechslin. Using the new table structure, we design a new multiple data TMTO algorithm both with and without the DP method. The TMTO curve for the new algorithm is obtained to be $T^3M^7D^8 = N^7$. This curve is based on a conjecture on the number of distinct points covered by the new table. Support for the conjecture has been obtained through some emperical observations. For $D > N^{1/4}$, we show that the trade-offs obtained by our method are better than the trade-offs obtained by the BS method.

Keywords: one-way function, time/memory trade-off, cryptanalysis.

1 Introduction

One-way functions are fundamental primitives in the design of cryptographic algorithms. Consequently, from a cryptanalytic point of view, it is of fundamental importance to study methods for inverting one-way functions.

Hellman [7] in 1980, introduced a generic method of inverting one-way functions. Let $f : \{0,1\}^n \rightarrow \{0,1\}^n$ be the one-way function to be inverted. (Hellman originally considered f to be obtained from a block cipher by mapping the key space to the cipher space for a fixed message.) Using pre-computation time of N, Hellman showed that the online time T and memory M satisfy the relation $TM^2 = N^2$, where $N = 2^n$. Consequently, the attack is called a time/memory trade-off (TMTO) algorithm and the last equation is called a TMTO curve. Note that in the TMTO set-up, the pre-computation time of N is not considered since this is an offline one-time activity "which can be performed at the cryptanalyst's leisure"(quotation from [7]).

In the context of stream ciphers, the function from state to a substring of the keystream can be considered to be a one-way function. The difference from the block cipher scenario is that in this case, the obtained keystream provides multiple data points, inverting any of which yields a state of the stream cipher and constitutes an attack. Independent works by Babbage [1] and Golić [6], investigated this situation and obtained the so-called birthday attack satisfying the relations $TM = N$ and $T = D$, where D is the number of available data points.

Later work by Biryukov and Shamir [2], incorporated multiple data into the Hellman attack and obtained the TMTO curve $TM^2D^2 = N^2$ and $1 \leq D^2 \leq T$. This curve cannot be directly compared to the BG curve, since the applicability ranges are different.

Hellman's method requires a number of look-ups into a relatively large table. The cost of these look-ups can be significantly high. The number of table look-ups can be brought down by using the technique of distinguished points (DPs) attributed to Rivest. This idea has also been used in the context of stream ciphers. The attack on A5/1 uses this idea along with a different kind of sampling called the BSW sampling [3].

OUR CONTRIBUTIONS: We combine the table structure used by Hellman and the one used by Oechslin [10] to propose a new table structure. This table structure is then used to design a new time/memory trade-off algorithm. We provide the search algorithms both for the cases when DP method is used and when it is not used. The number of distinct points covered in the new table is difficult to analyse[1]. At present we make a conjecture on the coverage in the table. Support for the conjecture has been obtained by us through certain computer experiments which makes us believe the conjecture to be true. Further work is required to settle it. Based on the conjecture, we obtain a TMTO curve for the new table. The curve that we obtain is $T^3M^7D^8 = N^7$. This curve is better than the BS curve when $D > N^{1/4}$.

RELATED WORK: In an earlier work, Fiat-Naor [5] described a TMTO algorithm (for $D = 1$) satisfying the TMTO curve $TM^3 = N^3$. This is worse than the Hellman curve, but can be *proved* to hold for any one-way function. On the other hand, Hellman's (also BS and our) method requires certain randomness assumptions, which are reasonable to expect from standard cryptographic functions. Thus, the Fiat-Naor technique is mainly of theoretical interest. A more recent work by Oechslin [10], describes a construction which has the same asymptotic behavior as the Hellman method but improves the runtime by a factor of one-half. But the rainbow method is inferior to the Hellman method in the presence of multiple data points (see [4]).

2 New Algorithm

An idea of the table construction methods of Hellman [7] and Oechslin [10] is helpful in understanding the new method and its success probability. We present brief descriptions of these two methods in the Appendix.

In this section, we describe the new method. This has two parts – offline table preparation and online search algorithm. These two algorithms are described in Sections 2.1 and 2.2. The

[1]Our analysis in an earlier version of the paper submitted to Asiacrypt 2005 was incorrect as was pointed out by a reviewer for that conference. The current conjecture on the coverage follows from certain suggestions made by the reviewer.

combination of the online search with the DP method of Rivest substantially brings down the number of table look-ups. This variation of the online search algorithm is described in Section 2.3.

2.1 Table Preparation

Let $f : \{0,1\}^n \to \{0,1\}^n$ be the one-way function to be inverted and $N = 2^n$ be the size of the search space. Elements of $\{0,1\}^n$ will be called points. We will be given a point y and will have to find a point x such that $f(x) = y$.

We follow the convention $(f_1 \circ f_0)(x) = f_1(f_0(x))$ and similarly for extension to composition of more than one functions. Let $f_0, \ldots, f_{r-1}$ be r functions obtained from the one-way function f to be inverted, i.e.,

$$f_i = g_i \circ f \tag{1}$$

where g_i is the ith output modification (bijective) function. The standard assumption in the area (originally considered by Hellman [7]) is that if f behaves like a random function (as a proper cryptographic function should), then for simple choices of the g_i's, the functions f_i's can be assumed to be pairwise independent random functions.

In the pre-computation phase, we construct one table $\mathcal{M}$ of size $m \times (rt+1)$. Let the entries of the table be $x_{i,j}$ with $0 \leq i \leq m-1$ and $0 \leq j \leq rt$. For $0 \leq i \leq m-1$, the elements $x_{i,0}$ are chosen randomly. For $1 \leq j \leq rt$, we define,

$$x_{i,j+1} \quad = \quad f_{j \bmod r}(x_{i,j}). \tag{2}$$

The set of pairs of points $(x_{i,0}, x_{i,rt})$ for $0 \leq i \leq m-1$ are stored in a list $\mathcal{L}$ sorted on the second components. The first component is called a start-point and the second component is called an end-point. Thus, the list of pairs is sorted on the end points, as is usual in TMTO algorithms. (If $r = 1$, then we obtain a single Hellman table, while if $t = 1$, then we obtain a single rainbow table.)

Suppose we are given $D \geq 1$ many data points and it is required to invert any of these points. Our aim during the table preparation stage is to cover a constant fraction of N/D many data points, so that by the birthday bound, with a constant probability of success we will be able to invert one of the D data points. For this, we choose m, r and t to be such that $mrt = N/D$. If a single $m \times (rt+1)$ table covered a constant fraction of N/D points, then we would be done. Unfortunately, this is not the case and there are repetitions in a table which reduce the coverage. We have the following conjecture on the coverage of a single table.

Conjecture 1: Let m and t be chosen such that, $mt^2 \leq N$ in the above description. Also note that $mrt = N/D$. Then the number of distinct points in the first rt columns of $\mathcal{M}$ is at least a constant fraction of $mr^{\frac{1}{2}}t$.

Note:

1. Due to the fall in coverage, we have to use $r^{\frac{1}{2}}$ random tables each of size $m \times (rt+1)$. By the above conjecture, the total coverage in all the tables becomes a constant fraction of mrt as desired.

2. If $mt^2 \gg N$, then the above conjecture does not hold. In usual TMTO method, $mt^2 \leq N$ is usually taken to be the matrix stopping rule based on the birthday bound. This is not the intrepretation of this constraint in the above conjecture. We would like to emphasize that the condition $mt^2 \leq N$ is important for the conjecture to be true.

3. If $mt^2 \ll N$, then the coverage can be better. In fact, in our simulations, we have observed the coverage to be mrt/r^{α}, where α varies between 1/3 and 1/2. In this paper, we will be working only with $\alpha = 1/2$.

4. An earlier version of this paper had been submitted to Asiacrypt 2005, where we had mistakenly assumed the coverage of the table to be mrt, which gives a much better result compared to the Hellman method. Anonymous reviewers had pointed out that the coverage assumption was incorrect and one reviewer had mentioned the fall in coverage by the factor $r^{1/2}$ based on the experiments done by him/her. Comments by this reviewer was the starting point for arriving at Conjecture 1.

Emperical Observations: For $0 \leq j \leq rt$, define C_j to be jth column of the table $\mathcal{M}$. For $0 \leq k \leq r-1$, the function f_k is applied to the points in the columns $C_k, C_{r+k}, \ldots, C_{(t-1)r+k}$. Let $\mathcal{M}_k$ be the $m \times t$ sub-matrix formed by the columns $C_k, C_{r+k}, \ldots, C_{(t-1)r+k}$. By the table construction procedure, the column C_{lr+k} is obtained from $C_{(l-1)r+k}$ as

$$C_{lr+k} = (f_{k+(r-1) \bmod r} \circ \cdots \circ f_k)(C_{(l-1)r+k}).$$

If we define $\phi_k = (f_{k+(r-1) \bmod r} \circ \cdots \circ f_k)$, then $C_{lr+k} = \phi_k(C_{(l-1)r+k})$. In other words, in $\mathcal{M}_k$, the next column is obtained from the previous column by applying ϕ_k. By the table construction procedure, we have

$$\mathcal{M}_0, \mathcal{M}_1 = f_0(\mathcal{M}_0), \mathcal{M}_2 = f_1(\mathcal{M}_1), \ldots, \mathcal{M}_{r-1} = f_{r-2}(\mathcal{M}_{r-2}). \tag{3}$$

The columns of $\mathcal{M}_0$ are $C_0, C_r, \ldots, C_{(t-1)r}$ where $C_{lr} = \phi_0(C_{(l-1)r})$.

We have observed the following through computer experiments.

1. The number of distinct points covered by $\mathcal{M}_0$ is $\frac{mt}{r^{\alpha}}$ for some α in the range $\frac{1}{3} \leq \alpha \leq \frac{1}{2}$. The actual value of α depends on the values of m, t and r. The value $\alpha = \frac{1}{2}$ is attained for relatively small values of m.

2. If $\mathcal{M}_0$ is chosen to be a random collection of mt points (with $mt^2 \leq N$), then the union of $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_{r-1}$ covers a constant fraction of mrt points.

3. When $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_{r-1}$ are constructed using the suggested method, the total number of distinct points in the union of $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_{r-1}$ is $\frac{mrt}{r^{\alpha}}$ for some α in the range $\frac{1}{3} \leq \alpha \leq \frac{1}{2}$.

From emperical observations, the coverage drop is explained by the first point above, i.e., $\mathcal{M}_0$ covers $\frac{mt}{r^{\alpha}}$ distinct points. The function applied to the columns of $\mathcal{M}_0$ is ϕ_0 which is a composition of r many pairwise independent random functions $f_0, f_1, \ldots, f_{r-1}$. At this point, we do not have a good understanding of the behaviour of α, except for the emperical observations of its range. Explaining the value of α and settling the conjecture requires further study.

2.2 Search Algorithm

We describe the search in a single table for the pre-image of a single element y in the range of f. The search algorithm requires an online memory of r elements of $\{0,1\}^n$. We define

$$\psi \quad = \quad \phi_0 \quad = \quad f_{r-1} \circ f_{r-2} \circ \cdots \circ f_0. \tag{4}$$

Note that the cost of applying ψ once is equal to r invocations of f. The functions g_k which are used to define f_k from f are easy to compute and do not add to the cost.

Algorithm Search(y)
1. for $i = 0$ to $r-1$ do
2. $\quad v_i = g_i(y)$;
3. $\quad$ for $j = i+1$ to $r-1$ do
4. $\quad\quad v_i = f_j(v_i)$;
5. $\quad$ end do;
6. $\quad$ Process($r-1-i, v_i$);
7. end do;
8. for $i = 1$ to $t-1$ do
9. $\quad$ for $j = 0$ to $r-1$ do
10. $\quad\quad v_j = \psi(v_j)$;
11. $\quad\quad$ Process($r-1-j+ir, v_j$);
12. $\quad$ end do;
13. end do;
14. return "failure".

end Search.

For $0 \leq k \leq r-1$, let $w_k = g_k(y)$, i.e., w_k is the initial value of v_k (set by Search in Line 2). If a pre-image for y is present in the table, then one of the w_k's must be present in the table. To see this, suppose $x_{i,j}$ is such that $f(x_{i,j}) = y$. Let $k = j \bmod r$. Then

$$x_{i,j+1} = f_k(x_{i,j}) = g_k(f(x_{i,j})) = g_k(y) = w_k.$$

Thus, if none of the $w_0, \ldots, w_{r-1}$ is present in the table, then no pre-image of y is present in the table and the search is unsuccessful.

The function Process(k,u) performs the following task. The first input to Process(,) is the number of applications of the f_i's which have already been applied to obtain u. The algorithm first looks up u in the end points of $\mathcal{L}$. If no match is found, then it returns nothing and the Search continues as usual. If a match is found, then it goes to the corresponding start point and starts generating the relevant row. The parameter k denotes that a pre-image of y is (possibly) at the $(rt-1-k)$th position in the row. Thus, the row is generated upto this point and then f is applied to the last generated point to see if we obtain y. If this verification is correct, then Process(,) outputs the pre-image it has found and the whole algorithm (including Search) stops. If the verification fails, then we have a false alarm. Again Process(,) returns nothing and Search continues as usual. The details of the algorithm is given below.

Algorithm Process(k, u)
1. Look-up u among the end-points of the list $\mathcal{L}$;
2. if not found then return;
3. else
4. let $(x_{i,0}, x_{i,rt})$ be such that $u = x_{i,rt}$;
5. set $w = x_{i,0}$;
6. for $j = 0$ to $rt - k - 2$ do
7. $w = f_{j \bmod r}(w)$;
8. end do;
9. if $f(w) = y$ then "output w" and stop;
10. else return;
11. end if;

end Process.

The description of Process(,) ensures that it does not terminate on a false alarm. This is usually implicit in previous descriptions of search algorithms appearing in the literature.

The online memory requirement consists in storing the points $v_0, \ldots, v_{r-1}$. The total number of invocations of f in Process(,) is at most rt in the case a match is found else it is zero. We now count the total number of invocations of f made in Search(). The total number of invocations made in Lines 1 to 7 is $(r-1) + (r-2) + \cdots + 1 = r(r-1)/2$ and the total number of invocations made in Lines 8 to 13 is $(t-1)r^2$. Hence, overall the maximum number of invocations made by Search() is $rt + r(r-1)/2 + (t-1)r^2 \approx tr^2$.

Note that false alarms (as tested in Line 9 of Process(,)) increases the number of invocations of f. Each false alarm incurs a cost of rt invocations. It was shown by Hellman [7], that the expected number of false alarms per table is bounded above by $mt(t+1)/2N$. By the matrix stopping rule $mt^2 \leq N$ and hence the number of false alarms per table is upper bounded by half. Since there are r tables and each false alarm incurs a cost of at most rt invocations of f, the total cost due to the false alarms is at most $r^2t/2$. Hence, as in the case of Hellman's original method, the false alarms increase the expected computation by at most 50%.

Table look-ups are only made inside Process(,). Each call to Process(,) results in exactly one table look-up. Since at most $r + (t-1)r = rt$ calls to Process() are made, the number of table look-ups is also at most rt.

The idea of the search algorithm is to determine whether one of the w_k is present in the table. For each w_k, Search checks if it is in one of columns

$$C_{rt-(r-1-k)}, C_{r(t-1)-(r-1-k)}, \ldots, C_{k+1}.$$

The checking in $C_{rt-(r-1-k)}$ is done by successively applying $(f_{r-1} \circ \cdots \circ f_{k+1})$ to w_k (the initial value of v_k) to obtain the new value of v_k. This v_k is searched among the set of end-points (column C_{rt}) of the table. If a match is found, then w_k is in column $C_{rt-(r-1-k)}$. If no match is found, then $\psi = f_{r-1} \circ \cdots \circ f_0$ is applied to v_k to get the new value of v_k and again this is searched in column C_{rk}. If a match is found, then v_k is in column $C_{r(t-1)-(r-1-k)}$ and if a match is not found, the procedure is repeated by applying ψ successively to v_k to check if w_k is in one of the columns

$C_{r(t-2)-(r-1-k)}, \ldots, C_{r-1-k}$. More precisely, the test of whether w_k is in $C_{rj-(r-1-k)}$ $(1 \leq j \leq t)$ is done by checking if the v_k obtained after applying $(\psi^{(t-j)} \circ (f_{r-1} \circ \cdots \circ f_{k+1}))$ to w_k is in C_{rt}.

Suppose it has been determined that $(\psi^{(t-j)} \circ (f_{r-1} \circ \cdots \circ f_{k+1}))(w_k) \in C_{rt}$ (i.e., $w_k \in C_{rj-(r-k-1)}$). Let $p = rj - (r-k-1)$. Then, for some $0 \leq i \leq m-1$, we have

$$f_{rt-1 \bmod r}(\cdots(f_{p+1 \bmod r}(f_{p \bmod r}(w_k))\cdots) = x_{i,rt}.$$

Also, by table construction we have,

$$f_{rt-1 \bmod r}(\cdots(f_1(f_0(x_{i,0}))\cdots) = x_{i,rt}.$$

The two equalities suggest that

$$w_k = f_{p-1 \bmod r}(f_{p-2 \bmod r}(\cdots(f_1(f_0(x_{i,0}))\cdots)).$$

Note that this might not always hold, giving rise to a false alarm. (If f is a bijection, then this will always hold.) If the equation holds, then we have

$$\begin{aligned}
g_k(y) &= w_k \\
&= f_{p-1 \bmod r}(f_{p-2 \bmod r}(\cdots(f_1(f_0(x_{i,0}))\cdots)) \\
&= f_{p-1 \bmod r}(x_{i,p-1}) \\
&= f_k(x_{i,p-1}) \\
&= g_k(f(x_{i,p-1})).
\end{aligned}$$

Since g_k is invertible, this implies $y = f(x_{i,p-1})$, i.e., $x_{i,p-1}$ is a pre-image of y under f.

We now argue that if a pre-image of y is indeed present in the table, then it will be found by the search algorithm. So suppose that $x_{i,j}$ is a pre-image of y, i.e., $y = f(x_{i,j})$. Let $k = j \bmod r$. Then by definition

$$x_{i,j+1} = f_{j \bmod r}(x_{i,j}) = f_k(x_{i,j}) = g_k(f(x_{i,j})) = g_k(y).$$

Note that $\mathsf{Search}()$ sets $v_k = g_k(y)$ in Line 2. Also, by definition,

$$\begin{aligned}
x_{i,rt} &= (f_{rt-1 \bmod r} \circ f_{rt-2 \bmod r} \circ \cdots \circ f_{j+1 \bmod r})(x_{i,j+1}) \\
&= (f_{rt-1 \bmod r} \circ f_{rt-2 \bmod r} \circ \cdots \circ f_{j+1 \bmod r})(g_k(y)) \\
&= \underbrace{(\psi \circ \cdots \circ \psi)}_{l}(f_{r-1} \circ \cdots \circ f_{k+1})(g_k(y)).
\end{aligned}$$

Here $l \in \{0, \ldots, t-2\}$. If $l = 0$, the value of v_k passed to $\mathsf{Process}(,)$ on Line 6 is equal to $x_{i,rt}$. If $l > 0$, then after l iterations of the loop starting on Line 8, the value of v_k passed to $\mathsf{Process}()$ on Line 11 is equal to $x_{i,rt}$. Hence, the look-up with v_k in $\mathsf{Process}(,)$ on Line 1 will be successful and hence the value of w on Line 7 of $\mathsf{Process}(,)$ will equal $x_{i,j}$. Since w is returned, the search is successful.

2.3 Distinguished Point Method

The idea of using distinguished points is attributed to Rivest. A point is called distinguished if it satisfies some simple property, say the first q bits are zero. Rivest's idea was to generate a row of a Hellman table only upto a distinguished point. Thus, the rows are of variable lengths. The online search technique is suitably modified to tackle the DPs. The net effect is that the number of table look-ups reduces significantly.

We consider the use of the DP method to bring down the number of table look-ups in the new algorithm. During table preparation, a row is generated for at most rt steps or until a DP is reached. Thus, the end points of the table are DPs and the rows are of varying lengths. The triples (start-point, end-point, length) are stored sorted on end-points in the list $\mathcal{L}$. The third component (length) denotes the number of applications of the f_i's before reaching the end-point which is a DP. We require a bit vector $\beta[\,]$. The new Search() algorithm is as follows.

```
Algorithm DPSearch(y)
for i = 0 to r - 1 do β_i = 0;
for i = 0 to r - 1 do
    v_i = g_i(y);
    if (v_i is DP) then
        β_i = 1; DPProcess(0, v_i);
    end if;
    j = i + 1;
    while (β_i == 0) and (j ≤ r - 1) do
        v_i = f_j(v_i);
        if (v_i is DP) then
            β_i = 1; DPProcess(j - i, v_i);
        end if;
        j = j + 1;
    end do;
end do;
for i = 0 to t - 2 do
    for j = 0 to r - 1 do
        k = 0;
        while (β_j == 0) and (k ≤ r - 1) do
            v_j = f_k(v_j);
            if (v_j is DP) then
                β_j = 1; DPProcess(r - j + ir + k, v_j);
            end if;
            k = k + 1;
        end do;
    end do;
end do;
return "failure".
```

end DPSearch.

The basic idea of the search technique is the following. As before let $w_k = g_k(y)$, i.e., the initial value of v_k. The algorithm starts r (parallel) searches starting from the points $w_0, \ldots, w_{r-1}$. On w_k, we successively apply $f_{k+1 \bmod r}, f_{k+2 \bmod r}, \ldots$. After each application, we check whether a DP has been obtained. If a DP has been obtained, then we discontinue the chain starting at w_k and process v_k.

The new DPProcess(k, u) algorithm is the following. The changes are lesser. We only have to tackle the variable length rows. As before, the first input to DPProcess$(,)$ is the number of applications of the f_i's that have already been applied to obtain u.

Algorithm DPProcess(k, u)
1. Look-up u in the list $\mathcal{L}$;
2. if not found then return;
3. else
4.   let $(x_{i,0}, x_{i,l-1}, l)$ be such that $u = x_{i,l-1}$;
    here l is the length of the row starting at $x_{i,0}$;
5.   set $w = x_{i,0}$;
6.   for $j = 0$ to $l - k - 1$ do
7. $w = f_{j \bmod r}(w)$;
8.   end do;
9.   if $f(w) = y$ then "output w" and stop;
10.   else return;
11. end if;

end DPProcess.

The online memory requirement is to store the elements $v_0, \ldots, v_{r-1}$, which is r elements as before. The maximum number of invocations of f (ignoring false alarms) is $\approx tr^2$ as before. The advantage is that the number of table look-ups reduce substantially from rt to r. This is because, we perform at most one table look-up for each of $v_0, \ldots, v_{r-1}$. For each v_i, we apply the f_i's iteratively until a DP is reached. Only then is a table look-up performed. Also the bit β_i is set to one and v_i is not processed thereafter by DPSearch().

The search algorithm using DP is similar to the search algorithm without DP. There are two main differences. First, the rows are of variable lengths and each row ends in exactly one DP. Second, there are r parallel searches starting at points $w_0, \ldots, w_{r-1}$. If any of these searches end in a DP which is not among the end-points of the table, then we discontinue the corresponding search. The correctness of the search algorithm will follow, if we can only argue that none of the discontinued searches could have led to a pre-image.

Let $w_0, \ldots, w_{r-1}$ be the intial values of $v_0, \ldots, v_{r-1}$ respectively (set by DPSearch in Line 3). Suppose the search starting at w_k ends in a DP which is not among the set of end-points of the table. Then there cannot be any $x_{i,j}$ in the table such that $k = j \bmod r$ and $y = f(x_{i,j})$. To see this first note that by table construction, the row containing $x_{i,j}$ ends in a DP. Also,

$$w_k = g_k(y) = g_k(f(x_{i,j})) = f_k(x_{i,j}) = f_{j \bmod r}(x_{i,j}) = x_{i,j+1}$$

i.e., the next element after $x_{i,j}$ in the ith row is w_k and hence the search starting at w_k will end in the DP of the ith row. By an extension of this argument, if all the searches starting at $w_0, \ldots, w_{r-1}$ end in DPs which are not in the end-points of the table, then there is no pre-image of y in the table.

2.4 Parallelism

Practical implementations of TMTO will require some amount of parallel processing. Hence, it is important to identify the inherent parallelism present in the algorithms.

The table preparation stage can be fully parallelised in the sense that the generation of two distinct rows require no interaction and can be done in parallel.

During the online search, we consider the r different searches starting at the points $w_0, \ldots, w_{r-1}$. These searches are independent of each other and can be carried out in parallel for both the methods with and without DP. Thus, if we have r processors, then we can keep these busy for the entire search computation algorithm.

3 TMTO Curve

Suppose we have to find the pre-image of one of the D distinct points $y_1, \ldots, y_D$. This constitutes a set of size at least D of pre-images (we say at least, since each y_i may have more than one pre-image). Based on Conjecture 1 in Section 2, we assume the set of domain points covered by a single table with size $m \times (rt+1)$ to be $\frac{mrt}{r^{1/2}}$. To cover mrt points, we construct $r^{1/2}$ many random tables with size $m \times (rt+1)$ each. If $mrt = N/D$, then by the birthday bound, with constant probability of success, we have an intersection between the set of pre-images covered by the table and the set of possible pre-images of $y_1, \ldots, y_D$. Hence, the method will find a pre-image for at least one of the y_i's.

The memory required for storing the pairs of points does not depend on D. Also the runtime memory of r points can be reused for each data point. Hence, this also does not increase with D. On the other hand, the online time (the number of invocations of f and the number of table look-ups) increases linearly with an increase in the number of data points. We now define the following parameters and constraints.

$$
\begin{array}{lcll}
N & = & 2^n & \text{(size of search space);} \\
P & = & r^{1+\frac{1}{2}}mt & \text{(pre-computation time);} \\
M & = & r^{\frac{1}{2}}m & \text{(fixed memory size);} \\
\mathsf{Mr} & = & r & \text{(runtime memory);} \\
T & = & tr^{2+\frac{1}{2}}D & \text{(number of invocations of } f\text{);} \\
\mathsf{Tt} & = & tr^{1+\frac{1}{2}}D & \text{(number of table look-ups without DP);} \\
\mathsf{TDP} & = & r^{1+\frac{1}{2}}D & \text{(number of table look-ups with DP);} \\
mt^2 & = & N & \text{(constraint in Conjecture 1).}
\end{array}
$$

Note that for a feasible attack, we should have $M, \mathsf{Mr}, T, \mathsf{Tt}, \mathsf{TDP} < N$.

In previous TMTO algorithms, the parameter Mr was not present. In this algorithm, we have to take this into consideration. In general, the total coverage of distinct points in all the tables is $\frac{1}{r^{1/2}}$ times mrt, i.e., the coverage drops by a fraction of $r^{\frac{1}{2}}$. Hence, we will not choose r to be too large and certainly assume that $m > r$. Hence, the online memory will be less than the memory required to store the tables. Thus, if we perform the analysis only using M, then the actual memory requirement is at most twice M. Since we ignore constants (and logarithmic factors) in the analysis, this does not affect the asymptotic nature of the analysis.

Solving from $mrt = N/D$, $mr^{\frac{1}{2}} = M$ and $T = tr^{\frac{5}{2}}D$ we get,

$$m = \frac{N^{\frac{1}{4}}M^{\frac{3}{4}}}{T^{\frac{1}{4}}}; \quad r = \left(\frac{MT}{N}\right)^{1/2}; \quad t = \frac{N^{\frac{5}{4}}}{DM^{\frac{5}{4}}T^{\frac{1}{4}}}. \tag{5}$$

Substituting the values of m and t in $mt^2 = N$ we have $T^3M^7D^8 = N^7$. The condition $r \geq 1$ must hold, i.e., $N \leq MT$. We can now write the following relations the first one of which is usually called the TMTO curve.

$$\left.\begin{array}{rcl} T^3M^7D^8 & = & N^7; \\ N & \leq & MT. \end{array}\right\} \tag{6}$$

Note: Assuming the coverage to be $\frac{mrt}{r^{\alpha}}$, the tradeoff curve is $T^{2-\alpha}M^{\alpha+3}D^{2(1+2\alpha)} = N^{\alpha+3}$

3.1 Comparison

The original TMTO curve $TM^2 = N^2$ is due to Hellman [7] for the case $D = 1$. We will call this the Hellman curve. Later Oechslin [10], described a method for reducing the online runtime by one-half. TMTO was applied to stream ciphers by Babbage [1] and Golić [6]. This is actually the birthday attack and the curve obtained was $MT = N$ and $T = D$. We will call this the BG curve. (BS [2] writes the last condition as $1 \leq T \leq D$, which is achieved by ignoring some of the data during the online phase. However, this is misleading, since the table contains N/D points and if we ignore some of the online data, then the birthday bound no longer applies and the success probability goes down.)

Hellman attack in the presence of multiple data was analysed by Biryukov and Shamir [2]. They obtained the curve $TM^2D^2 = N^2$ with $1 \leq D^2 \leq T$. We will call this the BS curve. (For the BS curve, $r = t/D$, $T = t^2$, $M = mr = mt/D$ and $mt^2 = N$. Using the last two equations, $t = N/(MD)$ and $r = t/D = N/(MD^2)$. Since $r \geq 1$ must hold, we have $MD^2 \leq N$. Also, since $TM^2D^2 = N^2$, we have $N/(MT) = (MD^2)/N \leq 1$, i.e., $MD^2 \leq N \leq MT$. The last inequality is not explicitly mentioned in the literature.) In the presence of multiple data the rainbow method is inferior than the Hellman method (see [4]). For $D = 1$, the BS curve reduces to the Hellman curve. Also, a direct comparison of the BG and BS curves is not possible since in the BG curve, $T = D$ and in the BS curve $T \geq D^2$.

We do not perform a direct comparison to the BG method, since (as for the BS method) the two methods hold for different ranges of T and D. We next compare to the BS curves. For comparison, we fix the values of N and D, i.e., the size of the search space and the amount of available data are fixed.

Substituting $T = M$ and $D = N^a$ where $0 < a < 1$ in the new curve, we get $T = M = N^{\frac{7-8a}{10}} = T_{new}$ (say) whereas from BS curve we get $T = M = N^{\frac{2(1-a)}{3}} = T_{BS}$ (say). If $a > \frac{1}{4}$, then $T_{new} < T_{BS}$. Hence, we conclude that the trade-offs obtained from the new curve is better than the BS curve when the number of data points $D > N^{\frac{1}{4}}$.

4 Conclusion

In this paper, we have described a new time/memory trade-off algorithm to invert one-way functions. Our algorithm can use multiple data and satisfies the TMTO curve $T^3M^7D^8 = N^7$ based on a conjecture. We show that the trade-offs of the new algorithm is better than the curve obtainable from the Biryukov-Shamir [2] trade-off $TM^2D^2 = N^2$ for $D > N^{1/4}$. We also consider the number of table look-ups and show that the use of the DP method of Rivest can be combined with the new search algorithm to considerably bring down the required number of table look-ups.

References

[1] S. Babbage. A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers, European Convention on Security and Detection, IEE Conference Publication No. 408, May 1995.

[2] A. Biryukov and A. Shamir. Cyptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers, in the proceedings of Asiacrypt 2000, LNCS, vol 1976, pp 1-13, 2000.

[3] A. Biryukov, A. Shamir and D. Wagner. Real Time Cryptanalysis of A5/1 on a PC, Proceedings of FSE 2000, LNCS, vol 1978, pp 1-18, 2000.

[4] A. Biryukov, S. Mukhopadhyay and P. Sarkar. Improved Time-Memory Trade-offs with Multiple Data, in the proceedings of SAC 2005, LNCS, to appear.

[5] A. Fiat and M. Naor. Rigorous time/space tradeoffs for inverting functions, In STOC 1991, pp 534-541, 1991.

[6] J. Golić. Cryptanalysis of Alleged A5 Stream Cipher, Proceedings of Eurocrypt 1997, LNCS, vol 1233, pp. 239-255, 1997.

[7] M. Hellman. A cryptanalytic Time-Memory Trade-off, IEEE Transactions on Information Theory, vol 26, pp 401-406, 1980.

[8] J. Hong and P. Sarkar. New Applications of Time Memory Data Tradeoffs, Proceedings of Asiacrypt 2005, to appear.

[9] I.J. Kim and T. Matsumoto Achieving Higher Success Probability in Time-Memory Trade-Off Cryptanalysis without Increasing Memory Size, TIEICE: IEICE Transactions on Communications/Electronics/Information and System, pp 123-129, 1999.

[10] P. Oechslin. Making a faster Cryptanalytic Time-Memory Trade-Off, in the proceedings of Crypto 2003, LNCS, vol 2729, pp 617-630, 2003.

[11] M. J. Wiener. The Full Cost of Cryptanalytic Attacks, Journal of Cryptology 17(2): 105-124 (2004)

A Previous Constructions

In this section, we provide brief descriptions of the Hellman [7] and the rainbow construction of Oechslin [10]. This will help in understanding our construction. As before, let $f : \{0,1\}^n \to \{0,1\}^n$ be the one-way function to be inverted and $N = 2^n$. Elements of $\{0,1\}^n$ are called points. This f is assumed to behave like a random function and for simple output modification functions $g_0, \ldots, g_{r-1}$, we define $f_k = g_k \circ f$.

A.1 Hellman Method

Each of the functions $f_0, \ldots, f_{r-1}$ is used to construct one table. Thus, there are r tables $\mathcal{M}_0, \ldots, \mathcal{M}_{r-1}$, where each $\mathcal{M}_k$ is an $m \times (t+1)$ matrix.

We describe the construction of $\mathcal{M}_k$. Let $x^k_{0,0}, x^k_{1,0}, \ldots, x^k_{m-1,0}$ be a set of points chosen independently and uniformly at random from $\{0,1\}^n$. These m points form the first column of $\mathcal{M}_k$. The other entries of $\mathcal{M}_k$ are obtained using the rule

$$x^k_{i,j+1} \quad = \quad f_k(x^k_{i,j}) \tag{7}$$

where $0 \leq i \leq m-1$ and $0 \leq j \leq t-1$. Thus, each row is a chain of the form

$$x^k_{i,0};\ x^k_{i,1} = f_k(x^k_{i,0});\ x^k_{i,2} = f_k(x^k_{i,0});\ \cdots;\ x_{i,t} = f_k(x^k_{i,t-1}).$$

For the table $\mathcal{M}_k$, the pairs of points $(x^k_{i,0}, x^k_{i,t})$ are stored sorted on the second components. The first component is called a start-point and the second component is called an end-point.

During the online phase, a point y is given and we have to find an x such that $f(x) = y$. The search algorithm is as follows. It successively searches in the tables $\mathcal{M}_0, \ldots, \mathcal{M}_{r-1}$. The search in the table proceeds as follows. First apply g_k to y to obtain w_k. Then apply f_k repeatedly a maximum of t times to w_k. After each application of f_k, perform a look-up among the end-points of $\mathcal{M}_k$. If a match is found, then go to the corresponding start point and keep on applying f_k until w_k is reached. The previous point visited is a possible pre-image of y which can be easily verified.

A.2 Rainbow Method

In 2003, Oechslin [10] described a different construction method. In Oechslin's method, a single $m\times(t+1)$ table covers N points in the following manner. In this case, we use t functions $f_0, \ldots, f_{t-1}$, where $f_k = g_k \circ f$ and $g_0, \ldots, g_{t-1}$ are output modification functions as described above for the

Hellman method. The first column is a set of random points $x_{0,0}, \ldots, x_{m-1,0}$. The ith row of the table is formed as follows.

$$x_{i,0};\ x_{i,1} = f_0(x_{i,0});\ x_{i,2} = f_1(x_{i,1});\ \cdots;\ x_{i,t} = f_{t-1}(x_{i,t-1}).$$

Each such chain is called a rainbow chain and the method the rainbow method. The set of pairs $(x_{i,0}, x_{i,t})$ is stored sorted on the second component.

During the online phase, the search for a pre-image of y is carried out in the following manner. Define $w_k = g_k(y)$. First, w_{t-1} is searched among the end-points of the table; then f_{t-1} is applied to w_{t-2} and a look-up is performed among the end-points of the table; next f_{t-2}, f_{t-1} is applied to w_{t-3} and a look-up is performed. If the look-up for w_k is successful, then we go to the corresponding start-point and generate the chain until w_k is reached. The point visited prior to w_k is a possible pre-image, which is again easily verified by applying f to it.