Rigorous Bounds on Cryptanalytic Time/Memory Tradeoffs

Elad Barkan[1] Eli Biham[1] Adi Shamir[2]

[1] Computer Science Department
Technion – Israel Institute of Technology
Haifa 32000, Israel
[2] Department of Computer Science and Applied Mathematics
The Weizmann Institute
Rehovot 76100, Israel

Abstract. In this paper we formalize a general model of cryptanalytic time/memory tradeoffs for the inversion of a random function $f : \{0, 1, \ldots, N-1\} \mapsto \{0, 1, \ldots, N-1\}$. The model contains all the known tradeoff techniques as special cases. It is based on a new notion of *stateful random graphs*. The evolution of a path in the stateful random graph depends on a *hidden state* such as the color in the Rainbow scheme or the table number in the classical Hellman scheme. We prove an upper bound on the number of images $y = f(x)$ for which f can be inverted, and derive from it a lower bound on the number of hidden states. These bounds hold for an overwhelming majority of the functions f, and their proofs are based on a rigorous combinatorial analysis. With some additional natural assumptions on the behavior of the *online phase* of the scheme, we prove a lower bound on its worst-case time complexity $T = \Omega(\frac{N^2}{M^2 \ln N})$, where M is the memory complexity. Finally, we describe new rainbow-based time/memory/data tradeoffs, and a new method for improving the time complexity of the online phase (by a small factor) by performing a deeper analysis during preprocessing.
Keywords: Time/memory tradeoff, time/memory/data tradeoff, rigorous, lower bound, hidden state, stateful random graph, Hellman, Rainbow, Cryptanalysis.

1 Introduction

In this paper we are interested in generic ("black-box") schemes for the inversion of one-way functions such as $f(x) = E_x(0)$, where E is any encryption algorithm, x is the key, and 0 is the fixed plaintext zero. For the sake of simplicity, we assume that both x and $f(x)$ are chosen from the set of N values $\{0, 1, \ldots, N-1\}$.

The simplest example of a generic scheme is exhaustive search, in which a pre-image of $f(x)$ is found by trying all the possible pre-images x', and checking whether $f(x') = f(x)$. The worst-case time complexity T (measured by the number of applications of f) of exhaustive search is N, and the space complexity M is negligible. Another extreme scheme is holding a huge table with all the images (in increasing order), and for each image storing one of its pre-images. This method requires a *preprocessing phase* whose time and space complexities are about N, followed by an *online inversion phase* whose running time T is negligible and space complexity M is about N. Cryptanalytic time/memory tradeoffs deal with finding a compromise between these extreme schemes, in the form of a tradeoff between the time and memory complexities of the online phase (assuming that the preprocessing phase comes for free). Cryptanalytic time/memory/data tradeoffs are a variant which accepts D inversion problems and has to be successful in at least one of them. This scenario typically arises in stream ciphers, when it suffices to invert the function that maps an internal state to the output at one point to break the cipher. However, the scenario also arises in block ciphers when the attacker needs to recover one key out of D different encryptions with different keys of the same message [4, 5]. Note that for $D = 1$ the problem degenerates to the time/memory tradeoff discussed above.

1.1 Previous Work

The first and most famous cryptanalytic time/memory tradeoff was suggested by Hellman in 1980 [11]. His tradeoff requires a preprocessing phase with a time complexity of about N and allows a tradeoff curve of $M\sqrt{T} = N$. An interesting point on this curve is $M = T = N^{2/3}$. Since only values of $T \leq N$ are interesting, this curve is restricted to $M \geq \sqrt{N}$. Hellman's scheme consists of several tables, where each table covers only a small fraction of the possible values of $f(x)$ using chains of repeated applications of f. Hellman rigorously calculated a lower bound on the expected coverage of images by a single table in his scheme. However, Hellman's analysis of the coverage of images by the full scheme was highly heuristic, and in particular it made the formally unjustifiable assumption that many simple variants of f are independent of each other. Under this analysis, the success rate of Hellman's tradeoff for a random f is about 55%, which was verified using computer simulations. Shamir and Spencer proved in a rigorous way (in an unpublished manuscript from 1981) that for an overwhelming majority of the functions f, even the best Hellman table (with chains of unbounded length created from the best collection of start points, which are chosen using an unlimited preprocessing phase) has essentially the same coverage of images as a random Hellman table (up to a multiplicative logarithmic factor). However, they could not rigorously deal with the full (multi-table) Hellman scheme.

In 1982, Rivest noted that in practice, the time complexity is dominated by the number of disk accesses (random access to disk can be many orders of magnitude slower than the evaluation of f). He suggested to use distinguished points to reduce the number of disk accesses to about $\sqrt{T}$. The idea of distinguished points was described in detail and analyzed in 1998 by Borst, Preneel, and Vandewalle [8], and by Standaert, Rouvroy, Quisquater, and Legat in 2002 [15].

In 1996, Kusuda and Matsumoto [13] described how to find an optimal choice of the tradeoff parameters in order to find the optimal cost of an inversion machine. Kim and Matsumoto [12] showed in 1999 how to increase the precomputation time to allow a slightly higher success probability. In 2000, Biryukov and Shamir [6] generalized time/memory tradeoffs to time/memory/data tradeoffs, and discussed specific applications of these tradeoffs to stream ciphers.

A new time/memory tradeoff scheme was suggested by Oechslin [14] in 2003. It claims to save a factor 2 in the worst-case time complexity compared to

Hellman's original scheme (see Section 6.1 for a discussion of this point). Another interesting work on time/memory tradeoffs was performed by Fiat and Naor [9, 10] in 1991. They introduce a rigorous time/memory tradeoff for inverting *any* function. Their tradeoff curve is less favorable compared to Hellman's tradeoff, but it can be used to invert any function rather than a random function.

A question which naturally arises is what is the best tradeoff curve possible for cryptanalytic time/memory tradeoffs? Yao [16] showed that $T = \Omega(\frac{N \log N}{M})$ is a lower bound on the time complexity, regardless of the structure of the algorithm, where M is measured in bits. This bound is essentially tight in case f is a single-cycle permutation.[3] However, the question remains open for functions which are not single-cycle permutations. Can there be a better cryptanalytic time/memory tradeoff than what is known today?

1.2 The Contribution of This Paper

In this paper we formalize a general model of cryptanalytic time/memory tradeoffs, which includes all the known schemes (and many new schemes). In this model, the preprocessing phase is used to create a *matrix* whose rows are long chains (where each link of a chain includes one oracle access to f), but only the *start points* and *end points* of the chains are stored in a table, which is passed to the online phase (the chains in the matrix need not be of the same length).

The main new concept in our model is that of a *hidden state*, which can affect the evolution of a chain. Typical examples of hidden states are the table number in Hellman's scheme and the color in a Rainbow scheme. The hidden state is an important ingredient of time/memory tradeoffs. Without it, the chains are paths in a single random graph, and the number of images that these chains can cover is extremely small (as shown heuristically in [11] and rigorously by Shamir and Spencer). We observe that in existing schemes, almost all of the online running time is spent on discovering the value of the hidden state (hence the name *hidden* state), which means that it is advisable to keep the number of hidden states minimal. Once the correct hidden state is found, the online phase needs to spend only about a square root of the running time to complete the inversion.

The main effect of the hidden state is that it increases the number of nodes in the graph from N to NS, where S is the number of values that the hidden state can assume. The new larger graph is called the *stateful random graph*, and the chains we create are paths in this graph. Two nodes in the stateful random graph defined by a particular f are connected by an edge:

$$\boxed{\; y_i \;\big|\; s_i \;} \longrightarrow \boxed{\; y_{i+1} \;\big|\; s_{i+1} \;},$$

if (y_{i+1}, s_{i+1}) is the (unique) successor of (y_i, s_i) defined by a deterministic transition function, where y_i and y_{i+1} are the outputs of the f function in the two consecutive steps, and s_i, s_{i+1} are the respective values of the hidden state during the creation of y_i and y_{i+1} (see Figure 1). The evolution of the y values along a path in the stateful random graph is "somewhat random" since it is

[3] In [11] and in the rest of this paper, M represents the number of start points, rather than the number of bits used to represent them.

determined by the random function f applied to a possibly non-random input. However, the evolution of the hidden state (s_i and s_{i+1}) can be totally controlled by the designer of the scheme.

The larger number of nodes is what allows chains to cover a larger number of images y, by reducing the probability of collisions. We rigorously prove that for any time/memory scheme and for an overwhelming majority of the functions f, the number of images that can be covered by any collection of M chains is bounded from above by $2\sqrt{SNM \ln(SN)}$, where $M = N^\alpha$ for any $0 < \alpha < 1$. Intuitively it might seem that making S larger at the expense of N should cause the coverage to be larger (as S can be made to behave more like a permutation). Surprisingly, S and N play the same role in the bound. The product SN (which is the number of nodes in the stateful random graph) remains unchanged if we enlarge S at the expense of N or vice versa. Note that $\sqrt{SNM}$ is about the coverage that is expected with the Hellman or Rainbow schemes, and thus even for the best choice of start points and path lengths (found with unlimited preprocessing time), there is only a small factor of at most $2\sqrt{\ln SN}$ that can be gained in the coverage. We use the above upper bound to derive a lower bound on the number S of hidden states required to cover at least half of the images by the matrix.

Under some additional natural assumptions on the behavior of the online phase, we give a lower bound on the worst-case time complexity:

$$T \geq \frac{1}{1024 \ln N} \frac{N^2}{M^2},$$

where the success probability is at least $1/2$.[4] Therefore, either there are no fundamentally better schemes, or their structure will have to violate our assumptions. Finally we show a similar lower bound for time/memory/data tradeoffs:

$$T \geq \frac{1}{1024 \ln N} \frac{N^2}{D^2 M^2}.$$

1.3 Structure of the Paper

The model is formally defined in Section 2, and in Section 3 we prove the rigorous upper bound on the coverage of M chains in a stateful random graph. Section 4 uses this bound to derive a lower bound on the number of hidden states. The lower bound on the time complexity (under additional assumptions) is given in Section 5. Additional observations and notes appear in Section 6, and the paper is summarized in Section 7. Appendix A contains an extension of the bound of Section 3 to a special case needed in Section 5. We refer the reader to [11, 14] for the details of previous tradeoffs schemes, or see a summary in [3, Appendix A.3].

2 The Stateful Random Graph Model

The class of time/memory tradeoffs that we consider in this paper can be seen as the following game: An adversary commits to a generic scheme with oracle

[4] We use constants rather than big O notation to demonstrate that no huge constants are involved; however, we do not claim that these constants are tight.

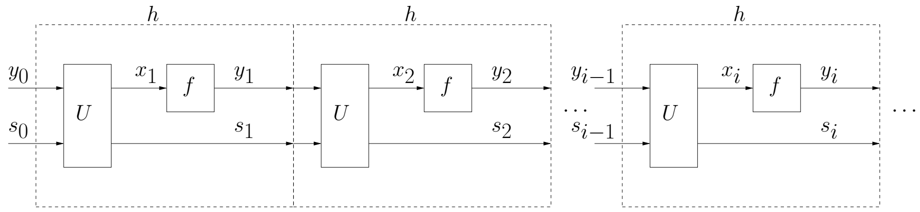

Fig. 1. A Typical Chain — A Path in the Stateful Random Graph

accesses to a function f, which is supposed to invert f on most images y. Then, the actual choice of f is revealed to the adversary, who is allowed to perform an unbounded *precomputation* phase to construct the best collection of M chains. The chains are not necessarily of the same length, and the collection of the M chains is called the *matrix*. Then, during the *online phase*, a value y is given to the adversary, who should find x such that $f(x) = y$ using the scheme it committed to. We are interested in the time/memory complexities of schemes for which the algorithm succeeds to invert y with probability of at least $1/2$ for an overwhelming majority of random functions f.

In the model that we consider, we are generous to the adversary by not counting the size of the memory that is needed to represent the scheme that it has committed to. Having been generous, we cannot allow the adversary to choose the scheme after f is revealed, as the adversary can use his knowledge to avoid collisions during the chain creation processes, and thus cover almost all the images using a single Hellman table.[5]

We do not impose any restrictions on the behavior of the preprocessing algorithm, but we require that it performs all oracle accesses to f through a *sub-algorithm*. When the preprocessing algorithm performs a series of oracle accesses to f, in which each oracle access can depend on the result of previous oracle accesses in the series, it is required to use the sub-algorithm. We call such a series of oracle accesses a *chain*. The *hidden state* is the internal state of the sub-algorithm (without the input/output of f).

A typical chain of the sub-algorithm is depicted in Figure 1, where by U we denote the function that updates the internal state of the sub-algorithm and prepares the next input for f, and by h we denote the entire complex of U together with the oracle access to f (i.e., an application of h corresponds to a single computation step of the sub-algorithm). We denote by s_i the hidden state which accompanies the output y_i of f in the sub-algorithm. The choice of U by the adversary together with f defines the stateful random graph, and h can be seen as the function that takes us from one node (y_{i-1}, s_{i-1}) in the stateful random graph

[5] In the *auxiliary memory* variant of the model, we can allow the scheme to depend on an additional collection of $M \log_2 N$ bits, which the adversary chooses during the preprocessing. Thus, the adversary can customize his scheme to the specific function f by giving it a free *advice* of limited size. This variant includes schemes such as the one presented in [10]. Analysis (briefly given in Appendix A) shows that this variant is stronger than the regular model only by a small constant factor, and thus, we restrict our discussion to the regular model without loss of generality.

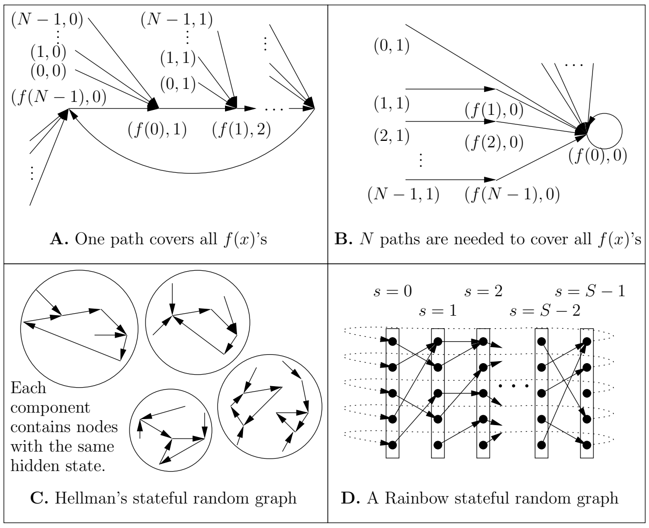

Fig. 2. Four Examples of Stateful Random Graphs

to the next node (y_i, s_i). U is assumed to be deterministic (if a non-deterministic U is desired, then the randomness can be given as part of the first hidden state s_0), and thus each node in the stateful random graph has an out-degree of 1.

Choosing U such that $s_i = s_{i-1} + 1 \pmod N$ and $x_i = s_{i-1}$ creates a stateful random graph that goes over all the possible images of f in a single-cycle (depicted in Figure 2.A), and thus represents exhaustive search (note that the y_{i-1} is ignored by U and thus all its N values with the same hidden state s_{i-1} converge to the same node $(f(s_{i-1}), s_{i-1}+1)$). Such a cycle is very easy to cover even with a single path, but at the heavy price of using N hidden states. At the other extreme, we can construct a stateful random graph (see Figure 2.B) that requires a full lookup table to cover all images of f by choosing U as: *if* $s_{i-1} = 1$ *then* $x_i = y_{i-1}$ and $s_i = 0$, *else* $x_i = s_i = 0$. In this function, each $(y_{i-1}, 1)$ is mapped by h to $(f(y_{i-1}), 0)$, and all these values are mapped to the same node $(f(0), 0)$.

As another example consider the mapping $x_i = y_{i-1}$ and $s_i = g(s_{i-1})$, where g is some function. This mapping creates a stateful random graph which is the direct product of the random graph induced by f, and the graph induced by g (this graph is not shown in Figure 2). We can implement Hellman's scheme by setting $x_i = y_{i-1} + s_i \pmod N$ and $s_i = s_{i-1}$, where s_i represents the table number to which the chain belongs. This stateful random graph (see Figure 2.C) consists of S disconnected components, where each component is defined by h and a single hidden state. Finally, we can implement a Rainbow scheme by setting $x_i = y_{i-1} + s_{i-1} \pmod N$ and $s_i = s_{i-1} + 1 \pmod S$, where S is the number of colors in the scheme. This stateful random graph (see Figure 2.D) looks like a layered graph with S columns and random connections between adjacent columns (including wrap-around links).

The preprocessing algorithm can stop the sub-algorithm at any point, using any strategy that may or may not depend on the value of the hidden states and the results of the oracle accesses, and it can use unbounded amount of additional space during its execution. For example, in Hellman's original method, the chain is stopped after t applications of f. Therefore, the internal state of the preprocessing algorithm must contain a counter that counts the length of the chain. However, the length of the chain does not affect the way the next link is computed, and therefore this counter can be part of the internal state of the preprocessing algorithm rather than the hidden state of the sub-algorithm. As a result, only the table number has to be included in the hidden state of Hellman's scheme. In the Rainbow scheme, however, the current location in the chain determines the way the next link is computed, and thus the index of the link in the chain must be part of the hidden state. The two kinds of states affect the development of chains in completely different ways: the hidden state can actually affect the values in the chain (as U depends on the hidden state), while any state which is not included in the hidden state can only stop the development of the chain, but not affect its values. As shown later, the number of hidden states strongly affects the success probability and the running time of the online phase.

The preprocessing algorithm can store in a *table* only the start points and end points of up to M chains, which are used by the *online algorithm*. Note that the requirement of passing information from the preprocessing phase to the online phase only in the form of chains does not restrict our model in any way, as the sub-algorithm that creates the chains can be designed to perform any computation. Moreover, the preprocessing algorithm can encode any information as a collection of start points, which the online algorithm can decode to receive the information. Also note that this model of a single table can accommodate multiple tables (for example, Hellman's multiple tables) by including with each start point and end point the respective value of the hidden-state.

The input of the online algorithm is y that is to be inverted, and the table generated by the preprocessing algorithm. We require that the online algorithm performs all oracle accesses to f (including chain creation) through the same sub-algorithm used during the preprocessing. In the variant of time/memory/data tradeoffs, the input of the online algorithm consists of D values $y_1, y_2, \ldots, y_D$ and the table, and it suffices that the algorithm succeeds in inverting one image. This concludes the definition of our model.

In existing time/memory tradeoffs, the online algorithm assumes that the given $y = f(x)$ is covered by the chains in the table. Therefore, y appears with some hidden state s_i, which is unfortunately unknown. The algorithm sequentially tries all the values that s_i can assume, and for each one of them it initializes the sub-algorithm on (y, s_i). The sub-algorithm is executed a certain number of steps (for example, until an end point condition has been reached). Once an end point that is stored in the table has been found, the start point is fetched, and the chain is reconstructed to reveal the x_i such that $y = f(x_i)$.[6] Existing

[6] Note that the fact that an end point is found does not guarantee a successful inversion of y. Such a failure in inversion is called a false alarm, and it can be caused, for example, when the chain that is recreated from y merges with a chain (of the matrix) that does not contain y.

time/memory/data tradeoffs work in a similar way, and the process is repeated for each one of the D given images.

2.1 Coverage Types and Collision of Paths in the Stateful Random Graph

A Table with M rows induces a certain coverage of the stateful random graph. Each row in the table contains a start point and an end point. For each such pair, the matrix associated with the table contains the chain of points spanned between the start point and the end point in the stateful random graph. The set of all the points (y_i, s_i) on all these chains is called the *gross coverage* of the stateful random graph that is induced by the table.

The gross coverage of the M paths is strongly affected by collisions of paths. Two paths in a graph collide once they reach a common node in the graph, i.e., two links in two different chains have the same y_i value and the same hidden state s_i. From this point on, the evolution of the paths is identical (but, the end points, which can be chosen arbitrarily on the path, might be different). As a result, the joint coverage of the two paths might be greatly reduced (compared to paths that do not collide). It is important to note that during the evolution of the paths, it is possible that the same value y_i repeats under different hidden states. However, such a repetition does not cause a collision of the paths.

To analyze the behavior of the online algorithm, we are interested in the *net coverage*, which is the number of different y_i values that appear during the evolution of the M paths, regardless of the hidden state they appear with, as this number represents the total number of images that can be inverted. Clearly, the gross coverage of the M paths is larger than or equal to the net coverage of the paths.

When we ask what is the maximum gross or net coverage that can be gained from a given start point, we can ignore the end point and allow the path to be of unbounded length, since eventually the path loops (as the graph is finite). Once the path loops, the coverage cannot grow further. An equivalent way of achieving the maximum coverage of M paths is by choosing the end point of each path to be the point (y_i, s_i) along the path whose successor is the first point seen for the second time along this path.

3 A Rigorous Upper Bound on the Net Coverage of M Chains in a Stateful Random Graph

In this section we formally prove the following upper bound on the net coverage:

Theorem 1. *Let $A = \sqrt{SNM \ln(SN)}$, where $M = N^{\alpha}$, for any $0 < \alpha < 1$. For any U with S hidden states, with overwhelming probability over the choice of $f : \{0, 1, \ldots, N-1\} \mapsto \{0, 1, \ldots, N-1\}$, the net coverage of images ($y = f(x)$) values) on any collection of M paths of any length in the resulting stateful random graph is bounded from above by $2A$.*

This theorem shows that even though stateful random graphs can have many possible shapes, the images of f they contain can only be significantly covered

	M_1	M_2	$\cdots$	$M_{\binom{NS}{M}}$
f_1	0	0		
f_2	1	0		
$\vdots$			$\ddots$	
f_{N^N}				

Fig. 3. A Table W denoting for each function f_i whether the net coverage obtained from the set of start points M_j is larger (1) or smaller (0) than $2A$

by using many paths or many hidden states (or both), as defined by the implied tradeoff formula above. Without loss of generality, we can assume that $S < N$, since otherwise the claimed bound is larger than N, and clearly, the net coverage can never exceed N.

3.1 Reducing the Best Choice of Start Points to the Average Case

In the first phase of the proof, we reduce the problem of bounding the best coverage (gained by the best collection of M start points) to the problem of bounding the coverage defined by a random set of start points and a random f. We do it by constructing a huge table W (as shown in Figure 3) which contains a row for each possible function f, and a column for each possible set of M start points. In entry $W_{i,j}$ of the table we write 1 if the net coverage obtained by the set M_j of start points for function f_i (extended into paths of unbounded length) is larger than our bound ($2A$), and we write 0 otherwise. Therefore, a row i with all zeros means that there is no set of start points for the function f_i that can achieve a net coverage larger than $2A$.

To prove the theorem, it suffices to show that the number of 1's in the table, which we denote by $\#1$, is much smaller than the number of rows, which we denote by $\#r$ (i.e., $\#1 \ll \#r$). From counting considerations, the vast majority of rows contain only zeros, and the correctness of the theorem follows.

We can express the number of 1's in the table by the number of entries multiplied by the probability that a random entry in the table contains 1, and require that the product is much smaller than $\#r$, i.e., $\#1 = \text{Prob}(W_{i,j} = 1) \cdot \#c \cdot \#r \ll \#r$, where $\#c$ is the number of columns in the table. Therefore, it suffices to show that for a random choice of the function f and a random set of start points, $\text{Prob}(W_{i,j} = 1) \cdot \#c$ is very close to zero. We have thus reduced the problem of proving that the coverage in the best case is smaller than $2A$, to bounding the number of columns multiplied by the probability that the average case is larger than $2A$. This is proven in the next few subsections.

3.2 Bounding $\text{Prob}(W_{i,j} = 1)$

We bound $\text{Prob}(W_{i,j} = 1)$ by constructing an algorithm that counts the net coverage of a given function f and a given set of M start points, and analyzing the probability that the coverage is larger than $2A$. During this analysis, we would like to consider each output of f as a new and independent coin flip, as $\text{Prob}(W_{i,j} = 1)$ is taken over a uniform choice of the function f. However, this assumption is justified only when x_i does not appear as an input to f on

1. For $i \in \{1, \ldots, S\}$ $Bucket_i = LowerFreshBucket_i = UpperFreshBucket_i = \phi$.
2. $NetCoverage = SeenX = \phi$.
3. Apply h to the first start point to generate the first event $\stackrel{x_i}{\rightarrow}(y_i, s_i)$.
4. If y_i appears in $Bucket_{s_i}$ Jump to Step 7 (Collision is detected). Otherwise:
5. Add y_i to $Bucket_{s_i}$.
6. If x_i does not appear in $SeenX$ (i.e., x_i is fresh):
   (a) If y_i does not appear in $NetCoverage$, add it to $NetCoverage$.
   (b) If $|LowerFreshBucket_{s_i}| < A/S$, add y_i to $LowerFreshBucket_{s_i}$, otherwise, add y_i to $UpperFreshBucket_{s_i}$.
7. Move to the next event:
   - Add x_i to $SeenX$ (i.e., mark that x_i is no longer fresh)
   - If a collision was detected in Step 4, apply h to the next start point (stop if there are no unprocessed start points). Otherwise: generate the next event by applying h to (y_i, s_i).
8. Jump to Step 4.

Legend:

- $SeenX$ is used to determine freshness by storing all the values of x that have been seen by now. This is the only set that stores input values of f. All the other sets store output values of f.
- $Bucket_i$ stores the all the y's that have been seen along with hidden state i (used for collision detection).
- $NetCoverage$ stores all the y's that have been seen from all chains considered so far, but without repetitions caused by different hidden states.
- For fresh values of x, $LowerFreshBucket_i$ stores the first A/S values of $y = f(x)$ seen with hidden state i (note that the x is fresh, but the y could have already appeared in other Buckets).
- For fresh values of x, $UpperFreshBucket_i$ stores the values of y after the first A/S values were seen with hidden state i (again, such a y could have already appeared in other Buckets).

Fig. 4. A Particular Algorithm for Counting the Net Coverage

any previously considered point. In this case we say that x_i is *fresh*, and this freshness is a sufficient condition for f's output to be random and independent of any previous event.

Denote by $\stackrel{x_i}{\rightarrow}(y_i, s_i)$ the event of reaching the point (y_i, s_i), where x_i is the input of f during the application of h, i.e., $y_i = f(x_i)$. When we view the points $(y_i = f(x_i), s_i)$ as nodes in the stateful random graph, the value x_i is a property of the edge that enters (y_i, s_i), rather than a property of the node itself, since the same (y_i, s_i) might be reached from several preimages. The freshness of x_i (at a certain point in time) depends on the order in which we evolve the paths (the x_i is fresh the first time it is seen, and later occurrences of x_i are not fresh), but it should be clear that the net coverage of a set of paths is independent of the order in which the paths are considered.

The algorithm is described in Figure 4. It refers to the ratio A/S, which for the sake of simplicity we treat as an integer. Note that $A/S \geq 2\sqrt{M \ln(NS)}$ (as $S < N$), and $A/S \gg 1$ (as N grows to infinity) since $M = N^{\alpha}$. Thus, the rounding of A/S to the nearest integer causes only a negligible effect.

Lemma 1. *At the end of the algorithm $|NetCoverage|$ is the size of the net coverage.*

Proof We observe that the algorithm processes all the points (y_i, s_i) that are in the coverage of the chains originating from the M start points, since it only stops a path when it encounters a collision.

A necessary condition for a $y_i = f(x_i)$ to be counted in the net coverage is that y_i appears in an event $\stackrel{x_i}{\to}(y_i, s_i)$ that is not a collision and in which x_i is fresh. If this condition holds, the algorithm reaches Step 6a, and adds y_i to $NetCoverage$ only if the sufficient condition $y_i \notin NetCoverage$ holds. ■

At the end of the algorithm $NetCoverage = \cup_{i=1}^{S}(LowerFreshBucket_i \cup UpperFreshBucket_i)$, and thus

$$|NetCoverage| \leq \sum_{i=1}^{S}(|LowerFreshBucket_i| + |UpperFreshBucket_i|),$$

since each time a y_i value is added to $NetCoverage$ (in Step 6a) it is also added to either $LowerFreshBucket$ or $UpperFreshBucket$ in Step 6b. We use this inequality to upper bound $|NetCoverage|$.

Bounding $\sum_{i=1}^{S}|LowerFreshBucket_i|$ is easy, as the condition in Step 6b assures that for each i, $|LowerFreshBucket_i| \leq A/S$, and thus their sum is at most A. Bounding $\sum_{i=1}^{S}|UpperFreshBucket_i|$ requires more effort, and we do it with a series of observations and lemmas.

Our main observation on the algorithm is that during the processing of an event $\stackrel{x_i}{\to}(y_i, s_i)$, the value y_i is added to $UpperFreshBucket_{s_i}$ if and only if:

1. x_i is fresh (Step 6); and
2. $LowerFreshBucket_{s_i}$ contains exactly A/S values (Step 6b); and
3. (y_i, s_i) does not collide with a previous point placed in the same bucket (Step 4).

Definition 1. *An event $\stackrel{x_i}{\to}(y_i, s_i)$ is called a coin toss if the first two conditions hold for the event.*

Therefore, a y_i is added to $UpperFreshBucket_{s_i}$ only if $\stackrel{x_i}{\to}(y_i, s_i)$ is a coin toss (but not vice versa), and thus the number of coin tosses serves as an upper bound on $\sum_{i=1}^{S}|UpperFreshBucket_i|$.

Our aim is to upper bound the net coverage (number of images in the coverage) by the number of different x values in the coverage (which is equal to the number of fresh x's), and to bound the number of fresh x's by A (for lower fresh buckets) plus the number of coin tosses (upper fresh buckets).

Definition 2. *A coin toss $\stackrel{x_i}{\to}(y_i, s_i)$ is called successful if before the coin toss $y_i \in LowerFreshBucket_{s_i}$.*

Observe that a successful coin toss causes a collision, as $LowerFreshBucket_{s_i} \subseteq Bucket_{s_i}$ at any point in time, i.e., a successful coin toss means that the node (y_i, s_i) in the graph was already visited at some previous time (the collision is detected at Step 4). Note that a collision can also be caused by events other than a successful coin toss (and these events are not interesting in the context of the proof): For example, a coin toss might cause a collision in case $y_i \in Bucket_{s_i}$ (but $y_i \notin UpperFreshBucket_{s_i} \bigcup LowerFreshBucket_{s_i}$) before the coin toss.

Another example is when x_i is not fresh, and therefore, $\stackrel{x_i}{\rightarrow}(y_i, s_i)$ is not a coin toss, but $y_i \in Bucket_{s_i}$ before the event (x_i was marked as seen in an event of a hidden state different than s_i).

Since each chain ends with the first collision that is seen, the algorithm stops after encountering exactly M collisions, one per path. As a successful coin toss causes a collision, there can be at most M successful coin tosses in the coverage.

Note that the choice of some of the probabilistic events as coin tosses can depend on the outcome of previous events (for example, $LowerFreshBucket_s$ must contain A/S points before a coin toss can occur for hidden state s), but not on the current outcome. Therefore, once an event is designated as a coin toss we have:

Lemma 2. *A coin toss is successful with probability of exactly $A/(SN)$, and the success (or failure) is independent of any earlier probabilistic event.*

Proof As x_i is fresh, $y_i = f(x_i)$ is truly random (i.e., chosen with uniform distribution and independently of previous probabilistic events). There are exactly A/S different values in $LowerFreshBucket_{s_i}$, and thus the probability that y_i collides with one of them is exactly $\frac{A/S}{N} = \frac{A}{SN}$. As all the other coin tosses have an x_i value different from this one, the value of $f(x_i)$ is independent of theirs.■

It is important to note that the independence of the outcomes of the coin tosses is *crucial* to the correctness of the proof.

What is the probability that the number of coin tosses in the M paths is larger than A? It is smaller than or equal to the probability that among the first A coin tosses there were fewer than M successful tosses, i.e., it is bounded by

$$\operatorname{Prob}\left(\mathrm{B}\left(A,q\right) < M\right),$$

where $q = A/(SN)$ and $\mathrm{B}(A,q)$ is a random variable distributed according to the binomial distribution, namely, the number of successful coin tosses out of A independent coin tosses with success probability q for each coin toss.

Note that choosing A too large would result in a looser bound. On the other hand, choosing A too small might increase our bound for $\operatorname{Prob}(W_{i,j} = 1)$ too much. We choose A such that the expected number of successes Aq in A coin tosses with probability of success q satisfies $Aq = M\ln(SN)$. This explains our choice of $A = \sqrt{SNM\ln(SN)}$.

It follows that:

$$\begin{aligned}
\operatorname{Prob}(W_{i,j} = 1) &= \operatorname{Prob}(|NetCoverage| > 2A) \\
&\le \operatorname{Prob}\left(\sum_{i=1}^{S}(|LowerFreshBucket_i| + |UpperFreshBucket_i|) > 2A\right) \\
&\le \operatorname{Prob}\left(A + \sum_{i=1}^{S}(|UpperFreshBucket_i|) > 2A\right) \\
&= \operatorname{Prob}\left(\sum_{i=1}^{S}(|UpperFreshBucket_i|) > A\right) \le \operatorname{Prob}\left(\mathrm{B}\left(A,q\right) < M\right).
\end{aligned}$$

The first inequality holds due to the fact that $\sum_{i=1}^{S}(|LowerFreshBucket_i| + |UpperFreshBucket_i|) > |NetCoverage|$. The last inequality holds as the number of coin tosses upper bounds $\sum_{i=1}^{S}(|UpperFreshBucket_i|)$.

We bound $\text{Prob}\left(\text{B}\left(A,q\right) < M\right)$ by $M \cdot \text{Prob}\left(\text{B}\left(A,q\right) = M\right)$ because the binomial distribution satisfies $\text{Prob}(\text{B}(A,q) = b) \geq \text{Prob}(\text{B}(A,q) = b-1)$ as long as $b < (A+1)q$, and in our case $b \leq M$ while $(A+1)q = Aq+q = M\ln(NS)+q > M$ (as $Aq = M\ln(NS)$). Therefore, we conclude that

$$\text{Prob}\left(W_{i,j} = 1\right) \leq \text{Prob}\left(\text{B}\left(A,q\right) < M\right) \leq M \cdot \text{Prob}\left(\text{B}\left(A,q\right) = M\right).$$

3.3 Concluding the Proof

To complete the proof we show that $\text{Prob}(W_{i,j} = 1) \cdot \#c$ is very close to zero by bounding $\#c \cdot M \cdot \text{Prob}\left(\text{B}\left(A,q\right) = M\right)$.

In the following equations, we use the bound $\binom{x}{y} \leq x^y/y! \leq (xe/y)^y$, since from Stirling's approximation $y! \geq (y/e)^y$. We bound $(1-q)^{-M}$ by estimating that $q = \frac{A}{SN} = \sqrt{\frac{M\ln(SN)}{SN}} = \sqrt{\frac{\ln(SN)}{SN^{1-\alpha}}}$ is very close to 0, certainly lower than 0.5 (recall that $M = N^\alpha$, and $\alpha < 1$). Thus, $1-q > 0.5$, and $(1-q)^{-M} < 2^M$. Moreover, as $q > 0$ is very close to 0, we approximate $(1-q)^A$ as e^{-Aq}.

Since each column in W is defined by a subset of M out of the NS start points, $\#c = \binom{NS}{M}$, and thus

$$\#c \cdot M \cdot \text{Prob}\left(\text{B}\left(A,q\right) = M\right)$$

$$= \binom{NS}{M} M \binom{A}{M} (q)^M \cdot (1-q)^{A-M} \leq M e^{-Aq} \left(\frac{2e^2 AqNS}{M^2}\right)^M$$

and substitute $Aq = M\ln\left(SN\right)$

$$= M e^{-M\ln(NS)} \left(\frac{2e^2 NSM\ln(NS)}{M^2}\right)^M = M(NS)^{-M}\left(\frac{2e^2 NS\ln(NS)}{M}\right)^M$$

$$= M\left(\frac{2e^2\ln(NS)}{M}\right)^M = N^\alpha\left(\frac{2e^2\ln(NS)}{N^\alpha}\right)^{N^\alpha}.$$

When $N \to \infty$ the expression converges to zero, which concludes the proof.

4 A Lower Bound for S

We now analyze the minimum S required by the scheme. By Section 3, the net coverage of even the best set of M chains contains at most $2\sqrt{SNM\ln\left(SN\right)}$ distinct y_i values. To make the success probability at least one half, we need a net coverage of at least $N/2$. Therefore (recalling that $S \leq N$),

$$N/2 \leq 2\sqrt{SNM\ln\left(SN\right)} \leq 2\sqrt{SNM\ln(N^2)}.$$

From this, we derive a rigorous lower bound on the number of hidden states in any tradeoff scheme which covers at least half the images for almost all f:

$$S \geq \frac{N}{32M\ln N}.$$

5 A Lower Bound on the Time Complexity

So far we bounded the net coverage of the matrix produced by the preprocessing phase. In order to use this result to bound the running time of the online phase, we have to make an assumption on the behavior of the online phase.

As a motivation for the assumption consider the following simplistic "proof" for the lower bound on the time to invert an image y: As shown in the previous section, the preprocessing phase uses at least $S \geq \frac{N}{32M \ln N}$ hidden states for the overwhelming majority of the functions. How much time is spent per hidden state? The online algorithm assumes that y is covered by the table with some hidden state, but it does not know with which. Therefore, for each hidden state s_i, the algorithm tries y with s_i by repeatedly applying h on (y, s_i) until it can rule out s_i as the correct hidden state. The expected number of applications of h is at least the width of the matrix (i.e., $\frac{N}{2M}$). Multiply the number of hidden states by the expected running time per hidden state to receive the "bound":

$$T \geq \frac{N^2}{64M^2 \ln N}.$$

However, it should be clear that this proof is incorrect, since there can be a correlation between the hidden state and the length of the path we have to explore. One example of such a correlation is the Rainbow scheme, in which some hidden states appear only near the end points. Moreover, there can be more hidden states close to the end points than hidden states far from the end points, which shifts the average run per hidden state towards the end points.

In the rest of the section we rigorously lower bound the running time in the worst case, based only on the following assumption:

- Given y, the online algorithm works by sequentially trying the hidden states (in any order). For each hidden state s, it applies h on (y, s) at least t_s times in case (y, s) does not appear in a chain in the matrix, where t_s is the largest distance from any point with hidden state s in the matrix to its corresponding end point. In some cases (e.g., the Rainbow scheme) each t_s is a known constant. In other cases, the t_s values can depend on the specific matrix that results from the precomputation (and thus depend on the function f). The algorithm might not know the exact value of t_s, but can use any upper bound on t_s to limit the length of the chains it traverses, and in this case, its running time will be longer than our bound.

The assumption can be seen as a combination of three smaller principles: First, all the points in the precomputed matrix must be reachable by the online algorithm (therefore, if a point (y, s) appears in the matrix in a column which is t_s steps away from the end point, the algorithm must develop the chain for image y and hidden state s for at least t_s links). Second, the algorithm cannot tell one image from another (therefore, the algorithm must develop the chain for at least t_s links not only for that y, but also for all images tried with hidden state s). Third, the algorithm cannot know if an image y is covered by the matrix of the preprocessing, and assuming that it is covered, with which hidden state (therefore, all hidden states must be tried).

As a preparation for the proof, shift the chains in the matrix until their end points are aligned in the rightmost column. Consider the $l = \frac{N}{4M}$ columns which are adjacent to the end points. The sub-matrix which constitutes these l columns contains at most $N/4$ different images $f(x)$. We call this sub-matrix the *right sub-matrix*, and the rest of the matrix the *left sub-matrix*. As $M = N^{\alpha}$, l is large enough so we can round it to the nearest integer (with negligible effect).

The worst case (with regards to the time complexity) is when the input y to the algorithm is not an image under f, or y is an image under f but is not covered by the matrix. Then, the time complexity is at least the sum of all the lengths t_s. We divide the hidden states into two categories: *short hidden states* for which $t_s \leq l$, and *long hidden states* for which $t_s > l$. We would like to show that the number of long hidden states S_L is large, and use the time complexity spent just on the long hidden states as a lower bound on the total time complexity.

The net coverage of $f(x)$ images in the left sub-matrix must be at least $N/4$ images which do not appear in the right sub-matrix (since the total net coverage is at least $N/2$). Note that all the $N/4$ images in the left sub-matrix must be covered only by the S_L long hidden states, as all the appearances of short hidden states are concentrated in the right sub-matrix. In other words, the left sub-matrix can be viewed as a particular coverage of at least $N/4$ images by M continuous paths that contain only the S_L long hidden states.

It is not difficult to adapt the coverage theorem to bound the coverage of the left sub-matrix (using only long hidden states). The combinatorial heart of the proof remains the same, but the definitions of the events are slightly changed. For more details see Appendix A. The adapted coverage theorem implies that in order to have a net coverage of at least $N/4$ images, the number of long hidden states must satisfy

$$S_L \geq \frac{N}{64M \ln((SN)^2)} \geq \frac{N}{256 \ln N}$$

for an overwhelming majority of the functions. Since for each long hidden state $t_s \geq l$, the total time complexity in the worst case is at least

$$T \geq l \cdot S_L \geq \frac{N}{4M} \frac{N}{256 M \ln N} \geq \frac{1}{1024 \ln N} \frac{N^2}{M^2}.$$

Note that we had to restrict the length of t_s such that it includes all occurrences of the hidden state s in the matrix, as otherwise (and using the unlimited preprocessing), each chain could start with a prefix consisting of all the values of $f(x)$, and thus any image in the rest of the chain (the suffix) cannot be a fresh occurrence. The algorithm can potentially encode in the hidden state information about the x_i and $f(x_i)$ values seen in the prefix, in such a way that it can change the probability of collision (and in particular, avoid collisions). Note that the preprocessed chains in this case are very long, but the online phase can be very fast if it covers only short suffixes of each path. As a result, we cannot use the methods of our proof without making the assumption.

See [3, Appendix 5.9] for an algorithm that violates the assumption by spending less time on each one of the many wrong guesses of the hidden state compared to the correct guess of the hidden state. The key idea is to use a new variant of Hellman's method with distinguished points: Let p be the probability of a point to be distinguished. As a result, the expected chain length is about p^{-1}, and the standard deviation is also about p^{-1}. The algorithm takes advantage of the large variation in length by trying k times more start points than we need, and storing in the table only the longest chains. The resulting matrices are called *stretched* matrices. The gain in time is achieved due to two facts: First, we have to search fewer tables (as each table covers more values due to the longer chains). Second, we spend on average only p^{-1} applications of h on each wrong guess of the hidden state (which is several times shorter than the average chain length in the matrix). The marginal gain in time decreases fast as k grows larger (as we gain from the tail of the distribution). A few experimental results show that by trying four times as many start points ($k = 4$) during the preprocessing, we can save a factor of about 4 in the time complexity of the online algorithm, and with $k = 8$ we can save a factor of about 4.8. Note that these results are based only on the expected size of the coverage, and ignore other issues such as the time spent on false-alarms.

5.1 A Lower Bound on the Time Complexity of Cryptanalytic Time/Memory/Data Tradeoffs

The common approach to construct a time/memory/data tradeoff is to use an existing time/memory tradeoff, but reduce the coverage (as well as the preprocessing) of the table by a factor of D. Thus, out of the D images, one is likely to be covered by the table. The decreased coverage reduces the number of hidden states, and thus the time complexity per image is reduced by a factor of D^3. However, the online algorithm is applied D times in the worst case (for the D images), which results in an overall decrease in the time complexity by a factor of D^2 (note that the D time/memory tradeoffs can be executed in parallel, which can reduce the average time complexity in some cases). Using similar arguments and assumptions to the ones in the case of time/memory tradeoff, and assuming that the net coverage of the table is at least $N/(2D)$, it follows that the worst-case time complexity can be lower bounded by

$$T' \geq D \frac{1}{1024 D^3 \ln N} \frac{N^2}{M^2} = \frac{1}{1024 D^2 \ln N} \frac{N^2}{M^2}.$$

Note that this analysis works for cases where D is not too large (and thus S is larger than 1). Otherwise, a tighter bound can be reached as S cannot be lower than 1 (but the bound itself is correct for all choices of D).

6 Notes on Rainbow-Like Schemes

6.1 A Note on the Rainbow Scheme

The worst-case time complexity of the original Rainbow scheme was claimed to be half that of Hellman's scheme. However, the reasoning behind the claim estimates M by considering only the number of start points and end points, and completely disregards the actual number of bits that are needed to represent these points. What [14] ignores is that the start points and end points in Hellman's scheme can be represented by half the number of bits required in the Rainbow scheme. If we double M in Hellman's scheme to get a fair comparison, we can reduce T by a factor of four via the time/memory tradeoff, which actually outweights the claimed improvement by a factor of two in the Rainbow scheme (ignoring issues such as the number of false alarms, which are difficult to analyze). The basic idea is that the starts points need not be chosen at random. For example in Hellman's scheme (with $T = M = N^{2/3}$), the first 1/3 of the bits in a start point can be chosen to be zero. The second 1/3 of the bits can be chosen to be identical to the table number, and the last 1/3 of the bits can be chosen to be an index of the row. Only the last 1/3 of the bits have to be actually stored in memory (i.e., we need only $(\log N)/3$ bits for each start point). In Rainbow tables by contrast, we can choose the first 1/3 of the bits to be zero, but as there is only one table, the remaining 2/3 of the bits must be the index. Thus, we have to store twice as many bits for each start point compared to Hellman's scheme. In both methods the number of bits that are required to store end points is considerably smaller than the number of bits that are required for the start points (and we therefore ignore it): The end points in Hellman (and Rainbow) can be compressed by storing only a little more than the last $(\log N)/3$ bits (respectively, $2(\log N)/3)$ bits). Moreover, as the end points are sorted (and thus the difference between subsequent end points is expected to be small), we can further compress the end points by storing only the differences between subsequent end points.

6.2 Notes on Rainbow Time/Memory/Data Tradeoffs

The original Rainbow scheme does not provide a time/memory/data tradeoff, but only a time/memory tradeoff. A possible adaptation of the Rainbow scheme to time/memory/data tradeoffs is presented in [7], but the resulting tradeoff curve of $TM^2D = N^2$ is far inferior to the $TM^2D^2 = N^2$ curve presented in [6] for the Hellman method. We suggest two new ways of implementing a Rainbow-based time/memory/data tradeoff, with a curve similar to [6]. In both ways, we reduce the number of colors in the table, but the colors are organized in different ways along the chains. This section describes the key ideas of the new methods; the full analysis (which was verified by computer simulations) can be found in [3].

The first method is to reduce the number of colors to S by repeating the series of colors t times:

$$f_0 f_1 f_2 \cdots f_{S-1} f_0 f_1 f_2 \cdots f_{S-1} f_0 f_1 f_2 \cdots f_{S-1} \cdots f_0 f_1 f_2 \cdots f_{S-1},$$

we call the resulting matrix a *thin-Rainbow* matrix. The stateful random graph can be described by $x_i = y_{i-1} + s_{i-1} \pmod N$ and $s_i = s_{i-1} + 1 \pmod S$.

The resulting tradeoff[7] is $TM^2D^2 = N^2$, which is similar to the tradeoff in [6], i.e., we lose the claimed improvement (by a factor of 2) of the original Rainbow time/memory tradeoff. However, like the Rainbow scheme, this method still requires twice as many bits to represent its start points, and thus it is slightly inferior to [6]. In the online phase, each color is sequentially tried by continuing the chain for at most tS links.

As a preliminary to the second method, consider a scheme in which we group the colors together in groups of t, and thus a typical row looks like:

$$\underbrace{f_0f_0f_0\cdots f_0}_{t\text{ times}}\underbrace{f_1f_1f_1\cdots f_1}_{t\text{ times}}\underbrace{f_2f_2f_2\cdots f_2}_{t\text{ times}}\cdots\underbrace{f_{S-1}f_{S-1}f_{S-1}\cdots f_{S-1}}_{t\text{ times}},$$

we call the resulting matrix a *thick-Rainbow* matrix. Note, however, that during the online phase the algorithm needs to guess not only the "flavor" i of f_i, but also the phase of f_i among the other f_i's (except for the last f_i). In fact, the hidden state is larger than S and includes the phase, as the phase affects the development of the chain. Therefore, the number of hidden states is $t(S-1)+1$ (which is almost identical to the number of hidden states in the original Rainbow scheme), and we get an inferior tradeoff of $TM^2D = N^2$. On the other hand, in some cases we retain the claimed savings of 2 in the time complexity. This example demonstrates the difference between "flavors" of f and the concept of a hidden state.

We propose to implement a Rainbow-based time/memory/data tradeoff by using the notion of distinguished points not only to determine the end of the chain, but also to determine the points in which we switch from one flavor of f to the next. In this case, the number of hidden states is equal to the number of flavors, and does not have to include any additional information. We can specify U as: $x_i = y_{i-1} + s_{i-1} \pmod N$, and *if* y_{i-1} is special, *then* $s_i = s_{i-1} + 1 \pmod S$ *else* $s_i = s_{i-1}$, where y_{i-1} is special if its $\log_2 t$ bits are zeros. We call the resulting matrix a *fuzzy-Rainbow* matrix, as each hidden state appears in slightly different locations in different rows of the matrix. In the online phase, the colors are tried in the same order as in the Rainbow scheme. Analysis shows that the tradeoff curve is $2TM^2D^2 = N^2 + ND^2M$, with $T \geq D^2$. The factor two savings is gained when $N^2 \gg ND^2M \Rightarrow D^2M \ll N$ (which happens when $T \gg D^2$). The number of disk accesses is about $\sqrt{2T}$, when $D^2M \ll N$, but is never more than in thin-Rainbow scheme for the same memory complexity.

7 Summary

In this paper, we proved that in our very general model, and under the natural assumption on the behavior of the online phase, there are no cryptanalytic time/memory tradeoffs which are better than existing time/memory tradeoffs, up to a logarithmic factor.

[7] When we write a time/memory/data tradeoff curve, the relations between the parameters relate to the expected worst-case behavior when the algorithm fails to invert y, and neglecting false-alarms.

Acknowledgements

We would like to thank Joel Spencer for his contribution to the proof of the single table coverage bound in 1981, and Eran Tromer for his careful review and helpful comments on earlier versions of the paper. The second author was supported in part by the Israel MOD Research and Technology Unit.

References

1. Gildas Avoine, Pascal Junod, Philippe Oechslin, *Time-Memory Trade-Offs: False Alarm Detection Using Checkpoints (Extended Version)*, Available online on `http://lasecwww.epfl.ch/pub/lasec/doc/AJO05a.pdf`, 2005.
2. Steve Babbage, *A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers*, European Convention on Security and Detection, IEE Conference Publication No. 408, 1995. Also presented at the rump session of Eurocrypt '96. Available online on `http://www.iacr.org/conferences/ec96/rump/`.
3. Elad Barkan, *Cryptanalysis 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.
4. Eli Biham, *How to decrypt or even substitute DES-encrypted messages in* 2^{28} *steps*, Information Processing Letters, Volume 84, Issue 3, pp. 117–124, 2002.
5. Alex Biryukov, *Some Thoughts on Time-Memory-Data Tradeoffs*, IACR ePrint Report 2005/207, `http://eprint.iacr.org/2005/207.pdf`, 2005.
6. Alex Biryukov, Adi Shamir, *Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers*, Advances in Cryptology, proceedings of Asiacrypt 2000, Lecture Notes in Computer Science 1976, Springer-Verlag, pp. 1–13, 2000.
7. Alex Biryukov, Sourav Mukhopadhyay, Palash Sarkar, *Improved Time-Memory Trade-Offs with Multiple Data*, proceedings of SAC 2005, LNCS 3897, pp. 110–127, Springer-Verlag, 2006.
8. Johan Borst, Bart Preneel, Joos Vandewalle, *On the Time-Memory Tradeoff Between Exhaustive Key Search and Table Precomputation*, Proceedings of 19th Symposium on Information Theory in the Benelux, Veldhoven (NL), pp. 111–118, 1998.
9. Amos Fiat, Moni Naor, *Rigorous Time/Space Tradeoffs for Inverting Functions*, STOC 1991, ACM Press, pp. 534-541, 1991.
10. Amos Fiat, Moni Naor, *Rigorous Time/Space Tradeoffs for Inverting Functions*, SIAM Journal on Computing, 29(3): pp. 790-803, 1999.
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. Il-Jun Kim, 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.
13. Koji Kusuda, Tsutomu Matsumoto, *Optimization of Time-Memory Trade-Off Cryptanalysis and Its Application to DES, FEAL-32, and Skipjack*, IEICE Transactions on Fundamentals, Vol. E79-A, No. 1, pp. 35–48, 1996.
14. Philippe Oechslin, *Making a Faster Cryptanalytic Time-Memory Trade-Off*, Advances in Cryptology, proceedings of Crypto 2003, Lecture Notes in Computer Science 2729, Springer-Verlag, pp. 617–630, 2003.

15. Francois-Xavier Standaert, Gael Rouvroy, Jean-Jacques Quisquater, Jean-Didier Legat, *A Time-Memory Tradeoff Using Distinguished Points: New Analysis & FPGA Results*, proceedings of CHESS 2002, Lecture Notes in Computer Science 2523, Springer-Verlag, pp. 593–609, 2003.
16. Andrew Chi-Chih Yao, *Coherent Functions and Program Checkers (Extended Abstract)*, STOC 1990, ACM Press, pp. 84-94, 1990.

A The Extended Coverage Theorem

We can extend the coverage theorem to bound the net coverage that can be obtained by M paths, where the paths contain only a subset of size S' out of the $S \geq S'$ hidden states that U can use. In Section 5, the S' hidden states we are interested in (long hidden states) appear in the left sub-matrix, and the rest of the hidden states (short hidden states) appear *only* in the right sub-matrix.

One way of viewing the proof of Theorem 1 is as an algorithm that constructs the best possible coverage given an *advice* chosen from a set of $\#c = \binom{NS}{M}$ possible advices. The proof shows that for an overwhelming majority of the functions f, even the best advice (which can depend on the specific choice of f) cannot lead to a coverage which is larger than the bound. In the proof we did not use any properties of the advice, except for the number of possible advices. We can model the coverage of the left sub-matrix using a similar algorithm, only now the advice is larger. It contains not only the set of M start points, but also a termination point for each start point, and the number S' of long hidden states ($1 \leq S' \leq S$) that the coverage includes. When the algorithm reaches a termination point, it stops the development of the chain. Clearly, there are $\#c' = S\binom{(NS)^2}{M}$ possible advices. To accommodate for the larger number of advices, we update the values of q and A to $q' = A'/(S'N)$, where $A' = \sqrt{S'NM \ln{(SN)^2}}$ (A' is chosen such that $A'q' = M\ln((SN)^2)$ to deal with the effect of the larger advice). The only remaining change compared to the original proof is that if the algorithm encounters more than S' hidden states it halts and sets its net coverage to zero, as the advice is inconsistent.

Note that we can allow $S > N$, but the model would not be fair if we allow S to be arbitrarily large, as too much information on f can be encoded by every choice of the hidden state (and we do not count the memory complexity of representing U). For example, if $S = N^N$, then with $S' = 1$ we can encode all the information on f by the specific choice of single hidden state. However, a huge amount of $N \log_2 N$ bits are required just to represent that single hidden state. Therefore, we are only interested in $S \leq N^k$ for some constant k. Then, $A' \leq \sqrt{S'NM \ln{(N^k N)^2}} = \sqrt{S'NM 2(k+1) \ln N}$.

In the auxiliary memory model (described in a footnote in Section 2), the adversary is allowed to customize U to the specific f using $M \log_2 N$ bits of memory. The proof of the coverage theorem in this case is very similar to the above proof, only now there are $\#c'' = N^M\binom{(NS)}{M}$ possible advices. As a result, the coverage theorem remains correct if we replace the original A by $A'' = \sqrt{SNM \ln{(SN^2)}}$, which increase the bound by a small constant factor.