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 the new notion of *stateful random graphs*, whose evolution 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 using a tradeoff scheme, and derive from it a lower bound on the number of hidden states. These bounds hold with an overwhelming probability over the random choice of the function 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 algorithm, 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. We describe several new variants of existing schemes, including a method that can improve the time complexity of the online phase (by a small factor) by performing deeper analysis during the preprocessing phase.
Keywords: Time/memory tradeoff, time/memory/data tradeoff, rigorous, lower bound, hidden state, stateful random graph, Hellman, Rainbow.

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 $\{0, 1, \ldots, N-1\}$ of N possible values.

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 of exhaustive search is N, and the space complexity M is negligible. Another extreme scheme is holding a huge table with all the images, and for each image storing one of its pre-images. This method requires a *preprocessing phase* whose time and space complexities T and M 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 [3, 4]. Note that for $D = 1$ the problem degenerates to a 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 [9]. 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 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 with overwhelming probability over the choice of the random function f, even the best Hellman table (with unbounded chains 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 access operations (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 [6], and later by Standaert, Rouvroy, Quisquater, and Legat in 2002 [13].

In 1996, Kusuda and Matsumoto [11] 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 [10] showed in 1999 how to increase the precomputation time to allow a higher success probability. In 2000, Biryukov and Shamir [5] 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 [12] in 2003. It saves a factor 2 in the worst-case time complexity compared to Hellman's original scheme. Another interesting work on time/memory tradeoffs was performed by Fiat and Naor [7, 8] 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.

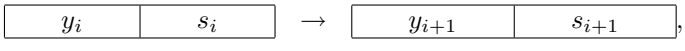
A question which naturally arises is what is the best tradeoff curve possible for cryptanalytic time/memory tradeoffs? Yao [14] showed that $T = \Omega(\frac{N \log N}{M})$ is a lower bound on the time complexity, regardless of the structure of the algorithm. This bound is tight up to a logarithmic factor, in case f is a single-cycle permutation, for which a tradeoff of $TM = N$ is possible [9], but 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 the hidden state, 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 [9] 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 (and hence the name *hidden* state). 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 states during the evolution of the chains in the preprocessing phase from N to NS, where S is the number of values that the hidden state can assume. The chains can be viewed as paths in a new directed graph, which we call the *stateful random graph*. Two nodes in the stateful random graph are connected by an edge:



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 output of the f function, and s_i, s_{i+1} are the respective values of the hidden

state during the creation of y_i and y_{i+1}. The evolution of the y values along a path in the stateful random graph is "somewhat random" since it is controlled by the random function f. 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 states is what allows chains to cover a larger number of images y. We rigorously prove that with an overwhelming probability over the choice of 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 behave more like a permutation). Surprisingly, S and N play the same role in the bound. The product SN 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.

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 (the constant 1024 can be greatly improved by using a tighter analysis). Therefore, either there are no fundamentally better schemes, or their structure will have to violate our assumptions. Finally we show a similar lower bound on time/memory/data tradeoffs of the form:

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

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 best achievable coverage of M chains in a stateful random graph. Section 4 uses the upper 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.

To make the paper self contained without exceeding the page limit, we include a description of the main details of the time/memory tradeoffs of [9, 12] in Appendix A. A new time/memory tradeoff is described in Appendix B. In Appendix C, we describe a time/memory tradeoff scheme that violates our assumptions on the behavior of the online phase, and in Appendix D we compare the time complexity of the Hellman and Rainbow scheme. Finally, Appendix E contains the analysis of some new time/memory/data tradeoffs.

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 accesses to a function f, which is supposed to invert f on a given image 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. 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. The chains are not necessarily of the same length, and the collection of the M chains is called the *matrix*. We are interested in the time/memory complexities of schemes for which the algorithm succeeds 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

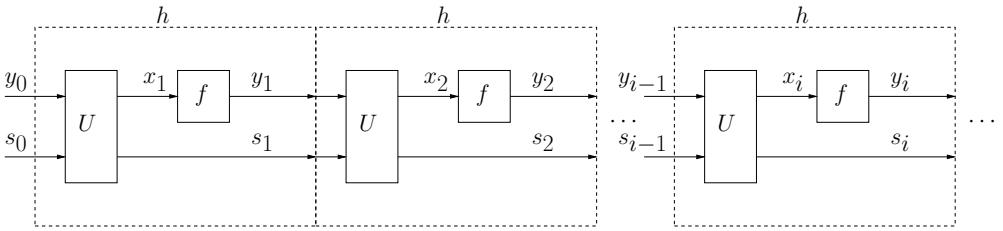


Fig. 1. A Typical Chain — A Path in the Stateful Random Graph

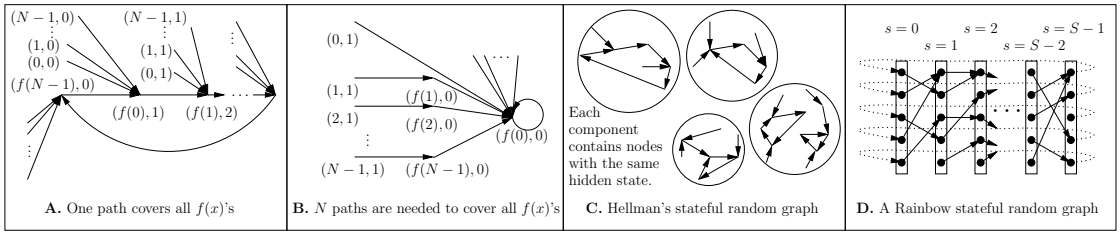


Fig. 2. Four Examples of Stateful Random Graphs

his knowledge to avoid collisions during the chain creation processes, and thus cover almost all the images using a single Hellman table.[1]

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.

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. 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 in the stateful random graph to the next node. 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 the figure). 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,

[1] A variant of the model is the auxiliary-memory model, in which we allow the scheme to depend on an additional collection of $M \ln N$ bits, which the adversary chooses during the preprocessing. Thus, we allow the adversary some customization of his scheme to the specific function f (within the limits of M memory rows). Analysis shows that the auxiliary-memory model is only marginally stronger (by small constant factor) than this model. Therefore, without loss of generality, we can discuss the model without auxiliary memory.

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 perform any preprocessing on a *start point* of the chain before executing the sub-algorithm on that point, and any postprocessing on the *end point* of the chain (for example, before storing it in long-term memory). 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 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 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 $y = f(x_i)$.[2] Existing 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 Collisions 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 can 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

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

	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 $2A$

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* (denoted by C), 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 Maximum Possible 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 maximum net coverage C of images ($y = f(x)$) values) on any collection of M paths of any length in the stateful random graph of U 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 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 the embedded function f_i (extended into paths of unbounded length) is larger than our bound ($2A$), and we write 0 otherwise. Therefore, a row with all zeros means that there is no set of start points for this embedded function 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, it follows that 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

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 that Counts the Net Coverage of M Start Points

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 embedded function and 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 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 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 in the rest of the analysis 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 $\stackrel{x_i}{\rightarrow}(y_i, s_i)$ 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 $NetCoverage$. ■

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

Observe that each 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). $LowerFreshBucket_{s_i}$ contains exactly A/S different values, 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 their values. ∎

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

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

where $q = A/(SN)$ and $\text{B}(A, q)$ 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 $\text{Prob}\left(W_{i,j} = 1\right)$ 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 $A = \sqrt{SNM\ln\left(SN\right)}$.

It follows that:

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

The first inequality holds as $\sum_{i=1}^{S}(|LowerFreshBucket_i|+|UpperFreshBucket_i|) > |NetCoverage|$. 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 $b = M$ 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, $1 - q$ is larger than 0.5, and $(1-q)^{-M}$ must be smaller than $(2)^M$. 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

$$\#c \cdot M \cdot \text{Prob}\left(\text{B}\left(A, q\right) = M\right) = \binom{NS}{M} M \binom{A}{M} (q)^M \cdot (1-q)^{A-M} \leq Me^{-Aq}\left(\frac{2e^2AqNS}{M^2}\right)^M$$

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

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

When N grows to infinity 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 $2\sqrt{SNM\ln(SN)}$ 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 $S \leq N$),

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

From this, we can derive the following rigorous lower bound on the number of hidden states in any time/memory tradeoff which covers at least half the space with high probability:

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

5 A Lower Bound on the Time Complexity

We lower bound the worst-case time complexity of the online phase under the following natural assumption on its behavior:

- 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. Note that the t_s values can depend on the specific matrix that results from the precomputation (and thus depend on the function f).

A simplistic "proof" for the lower bound is to say that with overwhelming probability $S \geq \frac{N}{32M\ln N}$, and for each hidden state we should run on average half the width of the matrix (i.e., $\frac{N}{4M}$). Multiply the two figures to receive the "bound":

$$T \geq \frac{N^2}{128M^2\ln N}.$$

However, it should be clear that this proof is incorrect, as for example, 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 above assumption.

Preparation: align the chains in the matrix such that their end points are aligned in a 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 $t_s > l$.[3] We would like to show

[3] Note that the distinction between short and long hidden states is unrelated to the number of images that appear with these hidden states.

that the number of long hidden states S_L is large, and use the time complexity spent on 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 F. The adapted coverage theorem implies that with an overwhelming probability, the number of long hidden states satisfies

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

Since for each long hidden state $t_s \geq l$, the total time complexity in the worst case is at least

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

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 are very long, but the online phase can be very fast if it covers only the suffixes of each path. As a result, we cannot use the methods of our proof in such a case.

In Appendix C, we present an algorithm that violates the assumption by spending less time on wrong guesses of the hidden state compared to the correct guesses of the hidden states. The resulting matrices are called *stretched* matrices, and allow the algorithm to achieves a time complexity which is better by a small factor compared to the known time/memory tradeoffs (but still far from the lower bound above), at the price of a lengthier preprocessing.

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

The common approach to time/memory/data tradeoffs is to use an existing time/memory tradeoff, but reduce the coverage (as well as the preprocessing) of the tables by a factor of D. Thus, out of the D images, one is likely to be covered by the table. The decrease in coverage reduces the number of hidden states, and thus the time complexity per image is reduced by a factor of D^3. However, the tradeoff might need to be 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, it follows that the worst-case time complexity can be lower bounded by

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

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 considers 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 [12] ignores is that the start points and end points in Hellman's scheme can be

compressed twice as much as 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 outweighs the claimed improvement by a factor of two in the Rainbow scheme (ignoring possible complications such as false alarms). For more details, see Appendix D.

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. The natural way to generalize the Rainbow scheme to a time/memory/data tradeoff is to reduce the number of colors, which can be reduced in several ways. 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[4] is $TM^2D^2 = N^2$, which is similar to the tradeoff in [5], 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 end points, and thus it is far inferior to [5]. Additional details can be found in Appendix E.

The second method is to group the colors together in groups of t, and a typical row looks like:

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

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, 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.

The new strategy we propose to implement a Rainbow-like time/memory/data tradeoff is to use 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. 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. Additional details are given in Appendix E.

7 Summary

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

[4] 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.

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. 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.
4. Alex Biryukov, *Some Thoughts on Time-Memory-Data Tradeoffs*, IACR ePrint Report 2005/207, `http://eprint.iacr.org/2005/207.pdf`, 2005.
5. 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.
6. 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.
7. Amos Fiat, Moni Naor, *Rigorous Time/Space Tradeoffs for Inverting Functions*, STOC 1991, ACM Press, pp. 534-541, 1991.
8. Amos Fiat, Moni Naor, *Rigorous Time/Space Tradeoffs for Inverting Functions*, SIAM Journal on Computing, 29(3): pp. 790-803, 1999.
9. Martin E. Hellman, *A Cryptanalytic Time-Memory Trade-Off*, IEEE Transactions on Information Theory, Vol. IT-26, No. 4, pp. 401–406, 1980.
10. 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.
11. 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.
12. 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.
13. 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.
14. Andrew Chi-Chih Yao, *Coherent Functions and Program Checkers (Extended Abstract)*, STOC 1990, ACM Press, pp. 84-94, 1990.

A Description of Current Approaches to Time/Memory Tradeoffs

A.1 Hellman's Time/Memory Tradeoff — Unicolor Tables

Let $f : \{0, 1, \ldots, N-1\} \mapsto \{0, 1, \ldots, N-1\}$ be a random function to be inverted. Hellman's basic idea is as follows: During a preprocessing phase, m chains of length t are created from m random start points as shown in Figure 5. Only the start points and end points are stored in a memory, indexed by the end points. The other points of the chains are discarded to save space. In the online phase, the algorithm is given $f(x)$. Assume that the value $f(x)$ appears in some chain, then f can be repeatedly applied on $f(x)$ until the end of the chain is reached. It is easy to identify the incident of reaching an end point, as the end points are stored in the database, and the algorithm can just search the database and see if the current value is an end point (only if the value is in the database, it can be an end point). Once the end point is found, the start point is retrieved from the data

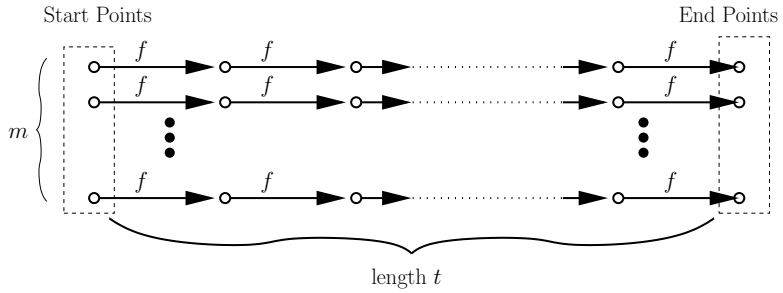


Fig. 5. A Single Hellman Matrix

base, and the chain is reconstructed until we hit $f(x)$ again. The value y before $f(x)$ is its preimage. However, if $f(x)$ is not encountered during the reconstruction of the chain from the start point, the algorithm understands that a *false alarm* occurred, i.e., the chain that was created from $f(x)$ *merged* (due to the collisions in f) with one of the chains that is covered by the chains.

How many applications of f are required? Let i be the number of applications of f on top of $f(x)$, such that $f^{i+1}(x)$ is an end point. Then, $f(x)$ should be encountered exactly after $t-i-1$ applications of f on the start point. Note that if after $t-1$ applications of f on top of $f(x)$ no end point is encountered, it means that no chain covers $f(x)$.

The algorithm succeeds only if $f(x)$ is covered by some chain. Therefore, the success probability of the method is bounded by the number of different $f(x)$ values that are covered by the matrix. Thus, the success probability is bounded by mt/N. Unfortunately, the number of different $f(x)$ values in the matrix cannot grow much beyond N/t: Assume that there are $mt = N/t$ different values in the matrix. When adding the next chain, due to the birthday paradox, one of the t values along the chain is likely to collide with one of the mt values already in the matrix (as $t \cdot (mt) \approx N$). Thus, when additional chains are added to the matrix, the fraction of unique values in the matrix decreases, and as a result, it is assumed that $mt < N/t$ (Hellman calculated that if $mt = N/t$ then the probability of success of a single table is about $0.80mt/N$).

Hellman's solution to this problem is to use t independent matrices from t cycles-independent functions f_i, such that the inversion of f_i would reveal the pre-image of f. Each matrix covers approximately N/t of the points. Thus, the collection of t matrices covers about N points, and the success probability would be about $1-(1-0.8mt/N)^t \approx 1-e^{mt^2/N} = 1-e^{-0.8} \approx 0.55$ (assuming $mt^2 = N$). Hellman suggests to choose $f_i = R_i(f)$, where R_i is a permutation, such that $f_i \neq f_j$ would have a very different cycle structure. However, it should be noted that the structures of f_i and f_j are not independent, and actually they are dependent as both functions are based on the same underlying function f. This dependency could be problematic, and it was not taken into account in the analysis. A similar problem exists with the analysis of the Rainbow scheme.

It should be noted that Hellman considered the case where f's output is larger than its input (e.g., $f(x) = DES_x(0)$ has 56 bits of input and 64 bits of output). In this case, the role of R is to be both a reduction and a re-randomization function. As a reduction function R reduces the size of the output back to 56 bits. As a re-randomization function, R_i can be a different permutation for each value of i. In case the output of f is larger than its input (as happens with $f(x) = DES_x(0)$), R has another degree of freedom as it can choose to remove different bits of the output.

The time/memory curve is as follows: The method requires $M = mt$ rows of memory. In the preprocessing phase the function f evaluated for about N times. In the inversion phase, f is evaluated $t-1$ times per table in the worst case, which gives a total time complexity of about $T = t^2$ (and on average half). By substituting mt by M and t by $\sqrt{T}$ in $mt^2 = N$, it follows that $M\sqrt{T} = N$. We do not analyze the time complexity due to false alarms, but Hellman calculated that it should not exceed $T/2$.

A.2 Oechslin's Time/Memory Tradeoff — Rainbow Tables

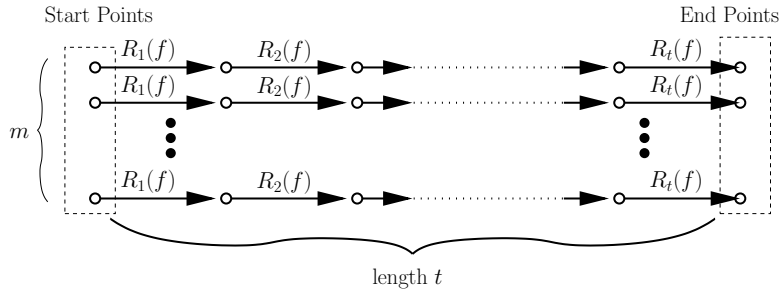


Fig. 6. Oechslin's Rainbow Matrix

Oechslin [12] recently suggested an improved time/memory tradeoff scheme. Oechslin's idea is to use Hellman's original suggestion with the same f_i functions, but during the computation of a chain, every value is calculated using a different f_i, as is depicted in Figure 6. The resulting matrix is called a *Rainbow matrix*. A Rainbow matrix induces a more efficient coverage of the search space by reducing the effect of collisions among the chains of the matrix. While in Hellman's method if the same value appears in two different chains of the same matrix, the chains merge, and one chain does not contribute to the coverage from merger point and on. However, in a Rainbow matrix, the same value must appear in two chains in the same column for a similar effect. Analysis shows that a collision in the same column of the matrix is not likely to occur as long as $mt \approx N$, therefore, a single Rainbow matrix has a coverage comparable to t Hellman matrices of the same length.

However, searching a Rainbow matrix takes about $t/2$ longer than searching in a single Hellman matrix. As only one Rainbow matrix is needed compared to t Hellman matrices, the result is that a Rainbow scheme save a factor $1/2$ of the time complexity in the worst case. In the average case (and assuming $f(x)$ is always covered by the tables), searching a single Rainbow matrix is four times faster than searching t Hellman matrices.

While in Hellman's scheme the effort of searching another column of the matrix is a single application of f, in the Rainbow scheme this is not the case. In the Rainbow scheme, for column i, $f(x)$ is transformed to $f_i(x)$ (by application of R_i on $f(x)$), and the chain is continued until the end point, i.e., $f_t(f_{t-1}(\cdots(f_{i+1}(f_i(x)))\cdots))$ is computed. Therefore, f is evaluated for $t-i$ times for column i. The worst-case time complexity is $T = \sum_{i=1}^{t}(t-i) \approx t^2/2$.

B A Time/Memory Tradeoff with Hidden State that Depends Only on the Previous Values in the Chain

The time/memory tradeoff scheme: We choose $x_i = y_{i-1} + s_{i-1} \pmod N$. We choose $s_i = s_{i-1} + y_{i-1} \pmod S$, where S is the number of hidden states. The hidden state S is chosen to be equal to the chain length. The rest of the details are similar to Hellman's scheme.

Analysis similar to the other tradeoffs results in a $TM^2 = N^2$ tradeoff curve. We have simulated this tradeoff and verified that it gives similar performance compared to Hellman's original time/memory tradeoff.

We can convert the time/memory tradeoff to a time/memory/data tradeoff by reducing the hidden state from the chain length to the chain length divided by D, as well as reducing the number of memory rows by a factor of D. The resulting tradeoff is $TM^2D^2 = N^2$.

C Stretching Distinguished Points — An Algorithm for Time/Memory Tradeoff with Deeper Preprocessing

The main observation behind this algorithm is that most of the time complexity of the algorithm is spent on wrong guesses of the hidden state. Therefore, there are two effective ways to reduce the time complexity: reduce the number of hidden states, and reduce the time that is spent on wrong guesses of the hidden state.

When distinguished points are used, there is variance in the length of the chains. Assuming the chain length is distributed according to the geometric distribution with success probability p (i.e., a point is distinguished with probability p), the expected chain length is $(1-p)/p \approx p^{-1}$. The standard deviation is $\sqrt{(1-p)/p^2} \approx p^{-1}$. Therefore, there is a large variation in the length of chains, and it is not surprising to find chains which are several times longer than their expected length.

Storing the longer chains in the matrices seems to accomplish both of the effective ways of reducing the time complexity: as the chains are longer, each matrix covers more, and less matrices are needed (i.e., the hidden state is reduced). Moreover, the time spent on wrong guesses is the average chain length, which is smaller than the average chain length in the table (as the table stores chains longer than the average). The suggested scheme is essentially Hellman's scheme with distinguished points, but we prefer to store longer chains in the matrices. The scheme performs a longer precomputation, in which many chains are created. Only the longer chains are stored in the matrices, and the shorter chains are discarded. We call the resulting matrices *stretched matrices*, as they contain longer (stretched) chains.

Another possible source of savings in the time complexity is having an idea choice of parameters for the scheme. Consider Hellman's time/memory tradeoff with distinguished points. Hellman suggests to fill a matrix until $mt^2 = N$, where $f : \{0, 1, \ldots, N-1\} \mapsto \{0, 1, \ldots, N-1\}$ is a random function, m is the number of rows in the matrix, and t is the chain length. Adding rows beyond the $mt^2 = N$ matrix stop rule becomes increasingly difficult. However, from the point of view of tradeoff efficiency, it is worthwhile adding rows to the matrix only until the moment that where we gain more by adding a row in a new matrix (rather than adding the row to the existing matrix), i.e., the time savings using the time/memory tradeoff curve is better than adding a row. We the optimal point in the next few paragraphs.

Let S be the number of the hidden states, i.e., S the number of tables, let C be the number of distinct points that are covered by a single matrix, and let T be the time complexity as anticipated by the "regular" tradeoff. For a single matrix, let m be the number of rows, and let

$$\gamma = m/(Np^2) \tag{1}$$

i.e., γ is the fraction of the number of rows compared to a single Hellman table (for Hellman $\gamma = 1$). The total number memory rows is $M = Sm$. Therefore,

$$\gamma = M/(SNp^2). \tag{2}$$

Let

$$\beta(\gamma) = C(\gamma)/(Np), \tag{3}$$

i.e., $\beta(\gamma)$ is the fraction of the coverage gained by a single table with $\gamma(Np^2)$ rows compared to the maximum coverage that is gained from a single Hellman table.

In this paragraph we show that the worst-case time complexity $T = \frac{\gamma^2 N^2}{\beta^3 M^2}$. The number of required tables is $S = N/C(\gamma) = N/(\beta(\gamma)Np) = 1/(\beta(\gamma)p)$ (to reach a constant success probability). Substitute $p = 1/(\beta(\gamma)S)$ in $\gamma = M/(SNp^2)$ and express S as:

$$S = \frac{\gamma N}{\beta^2(\gamma)M}. \tag{4}$$

Substitute S back to

$$p = \frac{1}{\beta(\gamma)S} = \frac{\beta(\gamma)M}{\gamma N}. \tag{5}$$

The worst-case time complexity (ignoring false alarms) is

$$T = S/p, \tag{6}$$

Table 1. Experiments Results

k	Gain Factor	Actual Work Factor
2^0	≈ 2.1	≈ 4.3
2^1	≈ 2.9	≈ 5.9
2^2	≈ 4	≈ 9
2^3	≈ 4.8	≈ 14.5
2^4	≈ 5.6	≈ 24.4
2^5	≈ 6.1	≈ 43
2^6	≈ 6.6	≈ 80.3

as evaluating each matrix takes on average p^{-1} applications of f. In the equation for T, we substitute p and S with their respective expressions from above:

$$T = \frac{S}{p} = \tag{7}$$

$$= \frac{\gamma^2 N^2}{\beta^3 M^2} \tag{8}$$

For Hellman method with distinguished points, the time complexity falls back to $T = \frac{N^2}{M^2}$. However, we are interested in the optimal value of γ that minimize the time complexity. We calculate the minimum (through the derivative of T), and reach the condition that:

$$\frac{d\beta}{d\gamma} = \frac{2\beta}{3\gamma}. \tag{9}$$

Suggested is the following *stretching algorithm* to construct a single *stretched* Hellman matrix with distinguished points:

1. Choose a work factor k.
2. Create kNp^2 rows, if two or more rows have the same end point, keep the longest row.
3. Sort the rows by their length.
4. Add rows to the final matrix, longest-row first.
5. Let L be the total length of the rows added until now, m be the number of the rows that have been added until now, and l be the length of the added row. Do not add the row and stop when $lp < 2Lp/(3m)$, i.e., $l < 2L/(3m)$ (alternatively, add new rows until β^3/γ^2 reaches a maximum).

The stop condition is equivalent to $\frac{d\beta}{d\gamma} = \frac{2\beta}{3\gamma}$.

As the matrix covers L distinct points using m rows, $\beta = L/(Np)$ and $\gamma = m/(Np^2)$. The time savings using this method is the ratio $\beta^3/\gamma^2 = L^3 p/(Nm^2)$, which we call the gain factor. The actual work factor, is the ratio between the time spent during preprocessing compared to the time that is spent during the construction of a regular Hellman matrix to achieve this coverage: $kNp/(L)$.

It is interesting to observe that the above method gains from the fact that the average time spent on wrong guesses of the hidden state is the average chain length p^{-1}. This figure is a several times smaller than the average chain length in the matrix. As most of the time complexity is spent on wrong guesses of the hidden state, the method gains the difference.

Experimental results are shown in Table 1. It should be noted that this method can be adapted to other schemes that are based on distinguished points, such as in Appendix E.

D Time Complexity of Hellman Versus Rainbow

It is surprising that the preprocessing and postprocessing that the algorithm can perform on start and end points is substantially different in the different schemes. For example, the start points in Hellman's scheme (using $M = N^{2/3}$) can be compressed to half of the size of what the start

points in a Rainbow scheme can be compressed to (for $M = N^{2/3}$). This factor two increase in the memory complexity translates to a factor four degredation in the time complexity, which consumes the savings that are introduced by a Rainbow scheme (compared to Hellman's scheme). However, the real advantage of Rainbow over Hellman's scheme is more complicated as it involves other factors such as the false alarm rate. In Hellman's scheme, $(\log_2 N)/3$ bits are enough to store the start points: the y value can be constructed by setting the first $(\log_2 N)/3$ bits to zeros, the next $(\log_2 N)/3$ bits to the hidden state (table number, which can be globally stored), and the last $(\log_2 N)/3$ bits to be an index (only the index bits need to be stored). In a Rainbow scheme, however, the hidden state is identical to all the start points, and therefore, only $2(\log_2 N)/3$ bits per start points are needed (these bits are used to store the index of the start point). We can overcome this disadvantage of the Rainbow scheme by dividing the single matrix to many smaller matrices each starting with another hidden state (and having $y_i = y_{i-1} + 1 \pmod S$), but this modification will also eliminate the factor 2 savings in the worst-case time complexity of the Rainbow scheme (and increase the number of disk accesses).

E Analysis of the New Time/Memory/Data Tradeoffs

E.1 Trivial Rainbow $T/M/D$ Tradeoff: $TM^2D = N^2$

The memory is left the same — M, but each row is shortened to t/D elements. The new Rainbow matrix covers Mt/D points, which represent constant fraction of N/D of the space. This implies $Mt = N$, which when raised to the power of 2 is:

$$M^2t^2 = N^2.$$

The total running time is about

$$T = Dt^2/D^2 = t^2/D,$$

substitute t^2 in the equation $M^2t^2 = N^2$ and get:

$$TM^2D = N^2.$$

As $t/D \geq 1$ it follows that $t \geq D \Rightarrow t^2 \geq D^2 \Rightarrow TD \geq D^2$, and therefore,

$$T \geq D.$$

E.2 Thin-Rainbow $T/M/D$ Tradeoff: $TM^2D^2 = N^2$

The matrix contain M rows of memory. Each row contains t sequences of S colors, i.e., it looks like:

$$f_0f_1f_2...f_{S-1}f_0f_1f_2...f_{S-1}f_0f_1f_2...f_{S-1}...f_0f_1f_2...f_{S-1}.$$

Therefore, the length of each row is St. The matrix stop rule in this case is $Mt^2S = N$ (as analyzed in Appendix E.4). When raised to the power of two:

$$M^2t^4S^2 = N^2.$$

We require that the matrix covers N/D elements, i.e.,

$$MSt = N/D.$$

Therefore, $S = N/(DMt)$. Substituting S in $M^2t^4S^2 = N^2$ gives $t^2 = D^2$, i.e.,

$$t = D.$$

The total time is $T = DStS = DtS^2 = DtN^2/(DMt)^2 = N^2/(DtM^2)$, as for each data point we go over all the S colors, and continue the chain for a length of about St (the length is actually

$St - s$, where s is the current hidden state, but we neglect s compared to St, which is accurate for $D \gg 1$.). Substitute t with D to achieve the tradeoff curve:

$$T = N^2/(D^2M^2).$$

As $S = N/(DMt)$ is at least 1, it follows that $N \geq DMt \Rightarrow N \geq D^2M \Rightarrow N^2/M^2 \geq D^4$. Substitute N^2/M^2 with TD^2 and get that:

$$T \geq D^2.$$

The number of disk accesses is $DtS = DtN/(DMt) = N/(M) = D\sqrt{T}$, as for each hidden state we need to have t disk accesses (whenever the chain reaches hidden state S), and we repeat the search D times. The number of disk accesses can be reduced to $\sqrt{T}$ by using distinguished points to mark the points of hidden state S that can end a chain (a point with hidden state S should be distinguished with probability t^{-1}).

E.3 Fuzzy-Rainbow $T/M/D$ Tradeoff: $2TM^2D^2 = N^2 + ND^2M$

The matrix contain M rows of memory. Each row contains about t repetitions of S colors, i.e., it looks like:

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

where the U function changes the value of the hidden state when a distinguished point occurs (with probability t^{-1}). The chain is terminated when a distinguished point is reached for hidden state S. Therefore, the expected length of each row is St. The matrix stop rule in this case is $Mt^2S = N$ (as analyzed in Appendix E.4). When raised to the power of two:

$$M^2t^4S^2 = N^2.$$

We require that the matrix covers N/D elements, i.e.,

$$MSt = N/D.$$

Therefore, $S = N/(DMt)$. Substituting S in $M^2t^4S^2 = N^2$ gives $t^2 = D^2$, i.e.,

$$t = D.$$

The total time is $T = D(S+1)St/2 = D^2S(S+1)/2 = N^2/(2D^2M^2) + N/(2M)$. It follows that:

$$2TD^2M^2 = N^2 + ND^2M.$$

As $S = N/(DMt)$ is at least 1, it follows that $N \geq DMt \Rightarrow N \geq D^2M \Rightarrow N^2/M^2 \geq D^4$. It follows that $T = N^2/(2D^2M^2) + N/(2M) \geq D^4/(2D^2) + D^2/2 = D^2$, i.e.,

$$T \geq D^2.$$

Note that when $T \gg D^2$, $ND^2M \ll N^2$, and the factor two in time savings is gained.

There is one disk access per hidden state (once we reach the end of the chain), and the search is repeated D times. Therefore, the number of disk accesses is $SD = N/(D^2M)D = N/(DM) \approx \sqrt{2T}$ (this figure is not higher than in the thin-Rainbow scheme).

E.4 Analysis of the Matrix Stop Rule in the Modified Rainbow Scheme

The thin-matrix contains M rows and looks like:

$$f_0f_1f_2\ldots f_{S-1}f_0f_1f_2\ldots f_{S-1}f_0f_1f_2\ldots f_{S-1}\ldots f_0f_1f_2\ldots f_{S-1}$$

$$f_0f_1f_2\ldots f_{S-1}f_0f_1f_2\ldots f_{S-1}f_0f_1f_2\ldots f_{S-1}\ldots f_0f_1f_2\ldots f_{S-1}$$

$$f_0f_1f_2\ldots f_{S-1}f_0f_1f_2\ldots f_{S-1}f_0f_1f_2\ldots f_{S-1}\ldots f_0f_1f_2\ldots f_{S-1}$$

$$f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} ... f_0 f_1 f_2 ... f_{S-1}$$

$$\vdots$$

$$f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} ... f_0 f_1 f_2 ... f_{S-1},$$

where S is the number of hidden states, and each hidden state appears t times in each row (the same analysis follows for the fuzzy-Rainbow scheme). Suppose that the matrix contains M rows and we are in the process of adding the $M+1$ row. Assuming that all the points in the first M rows are distinct, the new row which looks like:

$$f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} f_0 f_1 f_2 ... f_{S-1} ... f_0 f_1 f_2 ... f_{S-1}$$

collides with the matrix if at least one of its points with hidden state k collides with another point in the matrix with the same hidden state k. The probability that it happens is:

$$1 - (\text{prob. no collision}) = 1 - ((N - Mt)/N)^{St} \approx 1 - e^{-(Mt^2 S)/N}.$$

In birthday paradox, and in the matrix stop rule, we stop when the probability is about 0.5 ($1 - e^{-1}$ to be exact), which implies a matrix stop rule of $Mt^2 S = N$.

E.5 Notes

Note that the trivial Rainbow time/memory/data tradeoff scheme is not better than the other tradeoffs at any point. Consider the most extreme point that the tradeoff allows, i.e., when $T = D$. When using the other tradeoffs, what should be the data D', such that the memory complexity and the time complexity is identical?

Substitute $T = D$ in the original tradeoff to obtain $M = N/D$. Substitute the expression for T and M in the other tradeoff curves ($TM^2 D'^2 = N^2$):

$$D(N/D)^2 D'^2 = N^2.$$

It follows that

$$D' = \sqrt{D},$$

which is within the limits of the other tradeoff curves. Moreover, fewer data points are needed to achieve the same memory and time complexity.

We have verified the above tradeoffs through computer simulations.

F 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 S' hidden states out of the $S \geq S'$ hidden states of U. We call the hidden states in the set of S' hidden states *insider hidden states* and call the rest of the hidden states *outsider hidden states*.[5] Therefore, we are interested in the coverage of the sub-chains that begin in the start points and end before the first occurrence of an outsider hidden state. We call this set of sub-chains the *insider matrix*.

The tricky point is that the specific choice of the insider hidden states can depend on the choice of f, which is not a priori known to the algorithm that counts the coverage in the main proof. The crucial observation that solves the tricky point is that the only affect of the specific choice of the insider hidden states is the location in which the chains are terminated. In particular, the choice of the insider hidden states cannot affect the development of the chains, as the development of the chains is part of the definitions of U.

We can model the specific choice of insider hidden states by $M \ln(SN)$ bits that contain a list of M chain-terminating points. The set of terminating points contain the last points in the M chains

[5] Insider hidden states correspond to long hidden states, and outsider hidden states correspond to short hidden states. We make the distinction in the names to avoid confusion.

(after being chopped such that they do not contain outsider hidden states), i.e., the last points of the chains in the insider matrix. In case there are two chains such that one chain contains the terminating point of the other chain in its middle (i.e., not as a terminating point), we lengthen the the other chain such that it is terminated by the same termination point of the first chain.[6] The coverage is uniquely given by U, f_i and the set of start points and termination points. Therefore, it suffices to prove that given any U, it holds that for the overwhelming majority of functions f, there is no set of M start points and M termination points such that the resulting coverage in the insider matrix is larger than $2A$.

We have the following upper bound on the coverage of the insider matrix:

Theorem 2. *Let $A' = \sqrt{S'NM\ln(SN)}$, where $M = N^\alpha$, for any $0 < \alpha < 1$. Let U be any update function with S hidden states. For any $S' \leq S \leq N$, for any choice of f, and for any set of M start points, let the adversary choose a set of S' insider hidden states. Then, with overwhelming probability over the choice of $f : \{0, 1, \ldots, N-1\} \mapsto \{0, 1, \ldots, N-1\}$, there is no choice of start points and S' insider hidden states such that the net coverage in the resulting insider matrix is larger than $2A$.*

The proof begins with a reduction using a huge table W, similar to the one in the main proof (the number of functions remains N^N and the number of possible start points remains $\binom{NS}{M}$). However, we should also take into account the termination points. As there are M chains, there are NS termination points, and therefore, the number of possibilities to choose terminations points is $(NS)^M$. We have to multiply the number of columns by $(NS)^M$, since the net coverage of the insider matrix is given by a choice of a function f_i (determined by the row), and a choice of a set of M start points and M termination points (determined by the column). In each entry of the table we write one if and only if: the coverage of f_i using a set M_j of start points and a set M_k of termination points contains at most S' hidden states, and the net coverage is larger than $2A'$, where $A' = \sqrt{S'NM\ln((SN)^2)}$. Otherwise (if the there are more than S' hidden states in the coverage, or the net coverage is smaller than $2A$), we write zero.

It suffices to prove that the number of ones in the table is considerably smaller than the number of rows, as from counting arguments it would follow that most of the rows are zeros (and a row of zeros for a function f_i means that there is no choice of M_j start points and M_k termination points such that the resulting coverage indeed contains only S' hidden states and is larger than $2A$). Therefore, like in the main proof, it suffices to prove that the product of the number of columns and the probability that $W_{i,(j,k)}$ is 1 is very close to zero.

We now wish to upper bound the probability that $W_{i,(j,k)}$ is one. We use a similar method to the one in the main proof, i.e., an algorithm that counts the coverage. However, this time, the algorithm receives not only the set of M start points as input, but also the set of M termination points so it can count the resulting coverage. The exploration of each chain is stopped once either a collision occurs or a termination point is reached.[7] The only remaining differences in the algorithm compared to the original one is that the threshold of the lower fresh bucket is changed from A/S to A'/S', and once the algorithm encounters more than S' different hidden states, it sets the net coverage to zero and halts.

The analysis of the algorithm is similar. $W_{i,(j,k)}$ is one only if the coverage net coverage counted by the algorithm is larger than $2A'$. The algorithm can count a net coverage larger than $2A'$ only if it encounters more than $2A'$ fresh x's. The fresh x's are stored in the lower and upper fresh buckets. $W_{i,(j,k)}$ can be one only if the coverage contains at most S' hidden states, and in this case, only the buckets for the S' hidden states contain any elements. Therefore, the number of elements in the lower fresh buckets is at most $S'(A'/S') = A'$.

The net coverage is larger than $2A'$ only if the upper fresh buckets contain at least A' elements. That means that there are at least A' coin tosses. The probability of a coin toss of being successful is $q' = A'/(S'N)$ (the independence argument still holds, as a coin toss is performed only for a fresh value of x). We choose A' such that $A'q' = M\ln((SN)^2)$, i.e. $A' = \sqrt{S'NM\ln((SN)^2)}$. As each

[6] Clearly, the resulting coverage can only be larger (as we now need only $(M-1)\ln(SN)$ bits, we can fill the remaining bits with representation of any points that does not appear in the insider matrix).

[7] Like in the main proof, we do not lose any coverage by stopping a chain once it collides with a previously explored chain.

successful coin toss causes a collision that ends a chain, there can be no more than M successful coin tosses. Therefore, the probability that the net coverage is larger than $2A'$ is smaller than

$$\text{Prob}\left(\text{B}\left(A', q'\right) < M\right).$$

In the conclusion of the proof, the number of columns is $\#c = (NS)^M \binom{NS}{M}$. The factor $(NS)^M$ increase in the number of columns compared to the original proof is eliminated by the increase of $\ln(NS)$ (in Aq) from the original proof to $\ln((NS)^2)$ in $A'q'$. This concludes the modified proof.