Characterization and Improvement of Time-Memory Trade-Off Based on Perfect Tables

GILDAS AVOINE
UCL, Louvain-la-Neuve, Belgium
and
PASCAL JUNOD
Nagravision SA (Kudelski Group), Cheseaux, Switzerland
and
PHILIPPE OECHSLIN
Objectif Sécurité, Gland, Switzerland

Cryptanalytic time-memory trade-offs have been studied for twenty five years and have benefited from several improvements since the original work of Hellman. The ensuing variants definitely improve the original trade-off but their real impact has never been evaluated in practice. We fill this lack by analyzing the *perfect* form of classic tables, distinguished point-based tables, and rainbow tables. We especially provide a thorough analysis of the latter variant, whose performances have never been formally calculated yet. Our analysis leads to the concept of a *characteristic* that enables to measure the intrinsic quality of a trade-off. We finally introduce a new technique based on *checkpoints* that still reduces the cryptanalysis time, by ruling out false alarms probabilistically. Our analysis yields the exact gain of this approach and establishes its efficiency when applied on rainbow tables.

Categories and Subject Descriptors: E.3 [**Data Encryption**]: Code breaking

General Terms: Security, Performance, Algorithms

Additional Key Words and Phrases: Cryptography, Hellman's Time-Memory Trade-Off, Password Cracking, Rainbow Tables

1. INTRODUCTION

Many cryptanalytic problems can be solved in theory using an exhaustive search in the key space, but are still hard to solve in practice because each new instance of the problem requires to restart the process from scratch. The basic idea of a time-memory trade-off is to carry out an exhaustive search once for all such that following instances of the problem become easier to solve. Thus, if there are N possible solutions to a given problem, a time-memory trade-off can solve it with

This journal paper is an extended version of *Time-Memory Trade-Offs: False Alarm Detection Using Checkpoints* by Gildas Avoine, Pascal Junod, and Philippe Oechslin, published in the proceedings of *Progress in Cryptology – Indocrypt 2005*, vol. 3797 of Lecture Notes in Computer Science, pages 183–196, Springer-Verlag, December 2005.

T units of time and M units of memory. In the methods we are looking at, T is proportional to N^2/M^2 and a typical setting is $T = M = N^{2/3}$.

Cryptanalytic time-memory trade-offs have been introduced in 1980 by Hellman [Hellman 1980] and applied to DES. Given a plaintext D and a ciphertext C, the problem consists in recovering the key K such that $C = \mathsf{S}_K(D)$ where S is an encryption function assumed to follow the behavior of a random function. Encrypting D under all possible keys and storing each corresponding ciphertext allows for immediate cryptanalysis but needs N elements of memory.

The idea of a time-memory trade-off is to find a trade-off between the exhaustive search and the exhaustive storage. For that, an exhaustive search is carried out once (precomputation) and only a subset of generated values is kept. The precomputation consists in picking m starting elements S_j $(1 \leq j \leq m)$ and iterating the function f, defined by $f(K) := \mathsf{R}(\mathsf{S}_K(D))$, where R is a reduction function which generates an arbitrary key from a ciphertext. Let $X_{j,i+1} := f(X_{j,i})$ be the i-th iteration of f on S_j, and $E_j := X_{j,t}$ where t is the given length of the chains. We have:

$$\begin{array}{rcccccccccc}
S_1 = & X_{1,1} & \xrightarrow{f} & X_{1,2} & \xrightarrow{f} & X_{1,3} & \xrightarrow{f} & \dots & \xrightarrow{f} & X_{1,t} & = E_1 \\
S_2 = & X_{2,1} & \xrightarrow{f} & X_{2,2} & \xrightarrow{f} & X_{2,3} & \xrightarrow{f} & \dots & \xrightarrow{f} & X_{2,t} & = E_2 \\
\vdots & & & & & & & & & & \vdots \\
S_m = & X_{m,1} & \xrightarrow{f} & X_{m,2} & \xrightarrow{f} & X_{m,3} & \xrightarrow{f} & \dots & \xrightarrow{f} & X_{m,t} & = E_m
\end{array}$$

The key point is that only the first and the last elements of each chain are stored, providing so a *table*. In order to increase the success rate, i.e., the probability that $K \in \{X_{j,i},\, 1 \leq j \leq m,\, 1 \leq i \leq t\}$, several tables with different reduction functions are generated.

Given a ciphertext $C = \mathsf{S}_K(D)$, the on-line phase of the cryptanalysis works as follows: R is applied on C in order to obtain a key Y_1, and then the function f is iterated on Y_1 until matching any E_j. We have:

$$C \xrightarrow{\mathsf{R}} Y_1 \xrightarrow{f} Y_2 \xrightarrow{f} \dots \xrightarrow{f} Y_s$$

where s is the length of the generated chain from Y_1. Then the chain ending with $E_j = Y_s$ is regenerated from S_j until yielding the expected key K. However, finding a matching end of chain does not necessarily imply that the key K will be found in the regenerated chain. There exist situations where the chain that has been generated from C merges with a chain that is stored in the tables which does not contain K. This situation is called a *false alarm*.

Since the original work of Hellman, several papers have dealt with time-memory trade-offs. Fiat and Naor [Fiat and Naor 1991; 1999] showed that there exist cryptographically sound one-way functions that cannot be inverted with such a trade-off. Matsumoto, with Kusuda [Kusuda and Matsumoto 1996] and with Kim [Kim and Matsumoto 1999], analyzed the parameters of the trade-off. In 1982, Rivest [Denning 1982] suggested an optimization based on *distinguished points* (DP) which greatly reduces the amount of look-up operations which are needed to detect a matching end point in the table. Distinguished points are keys (or ciphertexts)

that satisfy a given criterion, e.g., the last n bits are all zero. In this variant, chains are not generated with a given length but they stop at the first occurrence of a distinguished point. This greatly simplifies the cryptanalysis. Indeed, during the attack, instead of looking up each Y_i that is generated on the chain from C, keys are generated until a distinguished point is found and only then a look-up is carried out in the table. If the average length of the chains is t, this optimization reduces the amount of look-ups by a factor t. Because merging chains significantly degrades the efficiency of the trade-off, Borst, Preneel, and Vandewalle [Borst et al. 1998] suggested to clean the tables by discarding the merging and cycling chains. This new kind of tables, called *perfect table*, substantially decreases the required memory. Later, Standaert, Rouvroy, Quisquater, and Legat [Standaert et al. 2002] dealt with a more realistic analysis of distinguished points and also proposed an FPGA implementation applied to DES with 40-bit keys. Distinguished points can also be used to detect collisions when a function is iterated, as proposed by Quisquater and Delescaille [Quisquater and Delescaille 1989], and van Oorschot and Wiener [Wiener and van Oorschot 1999]. In 2000, Biryukov and Shamir [Biryukov and Shamir 2001] suggested a new variant of trade-off where a set of values to invert is given, instead of a single value. This variant called *time-memory-data* trade-off has been studied since then by Biryukov, Mukhopadhyay, and Sarkar [Biryukov et al. 2005], Hong and Sarkar [Hong and Sarkar 2005].

In 2003, Oechslin [Oechslin 2003] introduced the trade-off based on *rainbow tables* and demonstrated the efficiency of his technique by recovering Windows passwords. A rainbow table uses a different reduction function for each column of the table. Thus, two different chains can merge only if they have the same key at the same position in the chain. This makes it possible to generate much larger tables. Actually, a rainbow table acts almost as if each column of the table was a separate single classic[1] table. Indeed, collisions within a classic table (or a column of a rainbow table) lead to merges whereas collisions between different classic tables (or different columns of a rainbow table) do not lead to a merge. This analogy can be used to demonstrate that a rainbow table of mt chains of length t has the same success rate as t single classic tables of m chains of length t. Like the trade-off based on distinguished points, rainbow tables reduce the amount of look-ups by a factor of t, compared to the classic trade-off. Recently, an FPGA implementation of rainbow tables has been proposed by Mentens, Batina, Preneel, and Verbauwhede [Mentens et al. 2005] in order to retrieve Unix passwords.

In [Barkan et al. 2006], Barkan, Biham, and Shamir describe a general model of cryptanalytic time-memory tradeoffs that includes all the known schemes. They introduce the concept of *hidden state* (which can be assimilated as the table number in Hellman's scheme or the color in the Rainbow scheme), and they observe that almost all the online running time is spent to discover the value of the hidden state. Once this value is found, the online phase needs only about a square root of the running time to complete the task. Using their very general model and under some natural assumptions about the behavior of the running phase, they formally show that no cryptanalytical time-memory tradeoff which are asymptotically better than existing ones can exist, up to a logarithmic factor.

[1]By *classic* we mean the tables as described in the original Hellman's paper.

This paper formalizes and extends our results that appear in [Avoine et al. 2005], and especially focuses on the analysis of the perfect tables. The resultant contribution is threefold:

—**Perfect Table Analysis.** We provide a thorough analysis of perfect rainbow, classic and DP tables. For each of them, we provide the expected maximum table size and success rate, and we give the expected cryptanalysis time. This leads to simple formula that allow to compute the optimal parameters of the trade-off. Up until now, calculations were always based on non-perfect tables, in the worst case (i.e., the key is not found in any table) and ignoring the amount of work due to false alarms. Optimizations have been proposed with these limitations, but to our knowledge, it is the first time that the average amount of work is used to find optimal parameters.

—**Trade-Off Characteristic.** Using our results on perfect tables, we introduce the concept of *characteristic* that allows to measure the intrinsic quality of a trade-off and thus to compare the different variants of trade-off. Up to our knowledge, that is the first time such a metric is proposed. We then compute the characteristics of classic, DP, and rainbow tables and show that rainbow tables outperform both classic tables and DP tables for high success rates.

—**Checkpoints.** Whether it is the classic, DP, or rainbow trade-offs, they all suffer from a significant quantity of false alarms. As we will develop below, false alarms may increase the time complexity of the cryptanalysis by more than 50%, contrarily to what is claimed in Hellman's original paper. Our technique is based on checkpoints, which are some positions on the chains where a test is applied. Such a test is typically a parity check that allows to rule out false alarms without regenerating the chains from scratch. We establish a formula to compute the exact checkpoint efficiency on rainbow tables, and illustrate our technique by a numerical example.

2. PERFECT TABLE ANALYSIS

The key to an efficient trade-off is to ensure that the available memory is used most efficiently. Consequently, the use of memory to store chains that contain elements which are already part of other chains should be avoided. To do so, more chains than actually needed are generated and then merging chains are removed. The resulting tables are called perfect tables [Borst et al. 1998]. Perfect classic and perfect DP tables are thus made of unique elements. In perfect rainbow tables, no element appears twice in any given column, but it may appear more than once across different columns.

Creating perfect rainbow and perfect DP tables is easy since merging chains can be detected by their identical end points. Since end points need to be sorted to facilitate the look-ups, identifying the merges comes for free. Classic chains do not have this advantage. Every single element of every classic chain that is generated has to be looked up in all elements of all chains of the same table. This requires $mt\ell$ look-ups in total where ℓ is the number of stored tables. In all trade-off variants, there is a limit to the size of the perfect tables. The brute-force way of finding the maximum number of chains of given length t that will not merge is to generate a chain from each of the N possible keys and remove the merges.

Below, we consider a problem of size N and we use a time-memory trade-off with M elements of memory. We denote t, m, and ℓ the parameters of the trade-off, which are respectively the length of the chains constituting the tables, the number of chains per table, and the number of tables. The probability that the trade-off succeeds with a single table is denoted by P and the probability that the trade-off succeeds with ℓ tables is denoted by P^*. For the sake of clarity, we will usually call P^* the *success rate* of the trade-off.

2.1 Rainbow Tables

The fastest cryptanalysis time is reached by using the largest possible perfect tables. This reduces the amount of duplicate information stored in the table and reduces the number of tables that have to be searched. Theorem 1 gives the expected maximum number of chains per table and Theorem 2 gives the expected maximum probability of success of a single table. These theorems lead to Corollary 1, which provides the optimal parameters that yield the largest perfect rainbow tables. Finally, Theorem 3 gives the average cryptanalysis time[2].

THEOREM 1. *Given t and a sufficiently large N, the expected maximum number of chains per perfect rainbow table without merge is:*

$$m_{\max}(t) \approx \frac{2N}{t+1}.$$

PROOF. For a given chain length t, the expected maximum number $m_{\max}(t)$ of rainbow chains that can be generated without merges is obtained by calculating the expected number of distinct keys in column t, if we start the generation with N keys in the first column. Here, the probability space under consideration is the set of all N^N functions on N elements equipped with the uniform distribution. Let m_i denote the number of distinct keys in column i. By definition, we have $m_1 = N$ and $m_{\max}(t) = m_t$. When $0 < i < t$, we can easily find out a recurrence relation on m_i:

$$m_{i+1} = N\left(1 - \left(1 - \frac{1}{N}\right)^{m_i}\right),$$

which can be approximated by

$$m_{i+1} = N\left(1 - e^{-\frac{m_i}{N}}\right).$$

Using the Taylor approximation of the exponential, we get:

$$m_{i+1} \approx N\left(\frac{m_i}{N} - \frac{m_i^2}{2N^2}\right) = m_i - \frac{m_i^2}{2N},$$

which is accurate for small m or non small i. We can transform this expression into a differential equation

$$\frac{dm_i}{di} = -\frac{m_i^2}{2N},$$

[2]Results given in this section appear in an informal way in [Avoine et al. 2005]. Theorem 1 also corrects the erroneous formula provided in [Avoine et al. 2005]

whose solution is:

$$m_i = \frac{2N}{i+c}.$$

Since $m_1 = N$ we get $c = 1$, which finally yields

$$m_{\max}(t) \approx \frac{2N}{t+1}.$$

□

THEOREM 2. *Given t, for any problem of size N, the expected maximum probability of success of a single perfect rainbow table is:*

$$P_{\max}(t) \approx 1 - \left(1 - \frac{2}{t+1}\right)^t$$

which tends toward $1 - e^{-2} \approx 86\%$ when t is large.

PROOF. From [Oechslin 2003], we know that the probability of success of a single un-perfect rainbow table is $1 - \prod_{i=1}^{t} \left(1 - \frac{m_i}{N}\right)$. With perfect rainbow tables, we have $m_i = m$ for all i such that $1 \leq i \leq t$. The probability of success of a single perfect rainbow table is therefore:

$$1 - \left(1 - \frac{m}{N}\right)^t,$$

which is maximum when $m = m_{\max}(t)$. Thus:

$$P_{\max}(t) = 1 - \left(1 - \frac{m_{\max}(t)}{N}\right)^t.$$

From Theorem 1, we deduce the expected maximum probability of success of a single perfect rainbow table containing $m_{\max}(t)$ chains:

$$P_{\max}(t) \approx 1 - \left(1 - \frac{m_{\max}(t)}{N}\right)^t = 1 - \left(1 - \frac{2}{t+1}\right)^t$$

which tends toward $1 - e^{-\frac{2t}{t+1}} \approx 1 - e^{-2} \approx 86\%$ for non small t. □

COROLLARY 1. *Given M, N, and P^*, the optimal parameters of the trade-off that minimize the cryptanalysis time are:*

$$\ell = \left\lceil \frac{-\ln(1-P^*)}{2} \right\rceil,\ m = \frac{M}{\ell},\ \text{and } t = \frac{\ln(1-P^*)}{\ln(1-\frac{M}{\ell N})\ell} \approx \frac{-N}{M}\ln(1-P^*).$$

PROOF. We first search the number of tables ℓ such that the success rate of the trade-off is at least P^*. Since $P_{\max}(t)$ is the expected maximum probability of success of a single table, we have:

$$1 - (1 - P_{\max}(t))^\ell \geq P^*. \tag{1}$$

Using Theorem 2, (1) yields:

$$e^{-2\ell} \leq (1 - P^*)$$

and therefore

$$\ell = \left\lceil \frac{-\ln(1-P^*)}{2} \right\rceil. \tag{2}$$

We deduce m and t from (2):

$$m = \frac{M}{\ell} \text{ and } t = \frac{\ln(1-P^*)}{\ln(1-\frac{M}{\ell N})\ell} \approx \frac{-N}{M}\ln(1-P^*).$$

□

Interestingly, Corollary 1 shows that the smallest number of tables needed for a trade-off only depends on the desired success rate P^*. This makes the selection of optimal parameters very easy.

Given the optimal parameters of the trade-off, we now calculate the exact amount of work required during the on-line phase.

THEOREM 3. *Given N, m, ℓ, and t, the average cryptanalysis time is:*

$$T = \sum_{\substack{k=1 \\ c=t-\lfloor\frac{k-1}{\ell}\rfloor}}^{k=\ell t} p_k \left(\frac{(t-c)(t-c+1)}{2} + \sum_{i=c}^{i=t} q_i i\right)\ell + (1-\frac{m}{N})^{\ell t}\left(\frac{t(t-1)}{2} + \sum_{i=1}^{i=t} q_i i\right)\ell$$

where

$$q_i = 1 - \frac{m}{N} - \frac{i(i-1)}{t(t+1)}.$$

PROOF. Cryptanalysis with a set of rainbow tables is done by searching for the key in the last column of each table and then searching sequentially through previous columns of all tables. There are thus a maximum of ℓt searches. We calculate below the probability of success and the probability of false alarm for each search k. Then we compute the expected cryptanalysis effort.

Computing the probability of success p_k during search k is straightforward: it is simply a geometric law:

$$p_k = \frac{m}{N}\left(1-\frac{m}{N}\right)^{k-1}. \tag{3}$$

The probability of a false alarm during search k does not depend on which table the search is being carried out, but only on which column the search is being carried out. Thus, we denote q_c the probability of a false alarm during search k, where $c = t - \left\lfloor\frac{k-1}{\ell}\right\rfloor$ is the position of the column counted from the end of the chain. To determine q_c, we notice that we generate a chain from a given ciphertext and look-up the end of the chain in the table. Thus, we can (a) either not find a matching end, (b) find the end of the correct chain or (c) find an end that leads to a false alarm. These three events are the only possible outcomes. The probability of finding the end of the correct chain within the m chains is:

$$\frac{m}{N}. \tag{4}$$

The probability of not finding an end point is the probability that all generated keys are not part of the chains that lead to the end points (at column i, these are the m_i chains that we used to build the table), that is:

$$\prod_{i=c}^{i=t}\left(1-\frac{m_i}{N}\right) \tag{5}$$

where $m_t = m$ and $m_{i-1} = -N \ln(1 - \frac{m_i}{N})$. Using (4) and (5), we deduce that the probability of a false alarm during search k is:

$$q_c = 1 - \frac{m}{N} - \prod_{i=c}^{i=t} \left(1 - \frac{m_i}{N}\right) \tag{6}$$

When the tables have exactly the maximum number of chains, (6) can be rewritten as a closed form, by replacing m_i by $m_{\max}(i)$:

$$\begin{aligned} q_c &= 1 - \frac{m}{N} - \prod_{i=c}^{i=t} \left(1 - \frac{m_{\max}(i)}{N}\right) \\ &= 1 - \frac{m}{N} - \frac{c(c-1)}{t(t+1)}. \end{aligned}$$

We now find out the expected cryptanalysis time T. For that, we notice that T corresponds to the work that is being carried out during the $t\ell$ searches and the work that is being carried out every time no key is found in the table. The latter work is simply:

$$(1 - \frac{m}{N})^{\ell t} \left(W(t) + Q(1)\right) \ell \tag{7}$$

where $W(x)$ represents the work needed to generate a chain until matching a end point and $Q(x)$ represents the work to rule out a false alarm. When searching a key at position c of a table, the amount of work to generate a chain that goes from position c to the end of the table is $t - c$. The additional amount of work due to a possible false alarm is $c - 1$ since the chain has to be regenerated from the start to position c in order to rule out the false alarm. Thus

$$W(c) = \sum_{i=c}^{i=t} (t - c) \tag{8}$$

and

$$Q(c) = \sum_{i=c}^{i=t} q_i (i - 1). \tag{9}$$

We compute in the same vein the work carried out during the search from 1 to $t\ell$:

$$\sum_{\substack{k=1 \\ c=t-\lfloor \frac{k-1}{\ell} \rfloor}}^{k=\ell t} p_k \left(W(t - c) + Q(c)\right) \ell. \tag{10}$$

From (7) and (10), we obtain:

$$T = \sum_{\substack{k=1 \\ c=t-\lfloor \frac{k-1}{\ell} \rfloor}}^{k=\ell t} p_k \left(W(c) + Q(c)\right) \ell + (1 - \frac{m}{N})^{\ell t} \left(W(t) + Q(1)\right) \ell. \tag{11}$$

Finally, rewriting (11) using (8) and (9), we obtain:

$$T = \sum_{\substack{k=1 \\ c=t-\lfloor\frac{k-1}{\ell}\rfloor}}^{k=\ell t} p_k \left(\sum_{i=1}^{i=t-c} i + \sum_{i=c}^{i=t} q_i i \right) \ell + (1-\frac{m}{N})^{\ell t} \left(\sum_{i=1}^{i=t} i + \sum_{i=1}^{i=t} q_i i \right) \ell$$

$$= \sum_{\substack{k=1 \\ c=t-\lfloor\frac{k-1}{\ell}\rfloor}}^{k=\ell t} p_k \left(\frac{(t-c)(t-c+1)}{2} + \sum_{i=c}^{i=t} q_i i \right) \ell + (1-\frac{m}{N})^{\ell t} \left(\frac{t(t-1)}{2} + \sum_{i=1}^{i=t} q_i i \right) \ell$$

□

To illustrate Theorem 3, Table I provides the theoretical and practical results when the size of the problem is $N = 7.056 \times 10^{12}$, the length of the chains is $t = 20479$, the number of chains per table is $m = 318422430$, and the number of tables is $\ell = 4$.

$N = 7.056 \times 10^{12}$, $t = 20479$, $m = 318422430$, $\ell = 4$	theory	measured over 1000 experiments
encryptions (average)	1.74×10^8	1.78×10^8
encryptions (worst case)	1.05×10^9	1.05×10^9
number of false alarms (average)	4481	4729
number of false alarms (worst case)	30274	30309

Table I. Calculated and measured performance of rainbow tables

In [Barkan et al. 2006], Barkan, Biham, and Shamir cite the original rainbow table paper [Oechslin 2003] saying that in the worst case, rainbow tables are two times more efficient. Then they explain that distinguished points can be stored with half as many bits than endpoints from rainbow tables, thus making up for the advantage of rainbow tables. The citation is unfortunate because the paper says that the gain must be *at least* a factor of two due to the structure of the tables and explains other aspects of rainbow tables that lead to a factor of 12 in the given example. Also our characteristics graph shows that this factor can be arbitrarily increased by choosing a higher success rate. The argument also fails to take into account that storage of rainbow tables can also be optimized. Indeed for both trade-offs the m starting points can be stored with $\log_2(m)$ bits each, since the starting points can be arbitrarily chosen. With distinguished points the endpoints can be stored without the information that makes them distinguished (e.g. a fixed suffix). With rainbow tables, the endpoints are stored as a long sorted list of random elements of N. A simple way of reducing the memory requirement of the list is to only store the least significant bits of each element and to build an index that points to the ranges of elements corresponding to each value of the remaining bits. In this case, the fact that rainbow tables have many more chains than tables from other trade-offs allows for a more efficient use of this technique. Although the exact values depend on all parameters of the trade-off and on the

complexity of the chosen representation, it is certainly no the case that rainbow tables need twice as many bits for storage as distinguished points.

2.2 Perfect Classic Tables

Although classic tables have been studied extensively, e.g., in [Kim and Matsumoto 1999] and [Kusuda and Matsumoto 1996], *perfect* classic tables have never been in the spotlight. We provide below the expected cryptanalysis time of such tables in terms of encryption operations, missing thus the fact that classic tables need t times more table look-ups than other variants. Theorem 4 and Theorem 5 establish respectively the probability of success of a single perfect classic table and the expected cryptanalysis time.

THEOREM 4. *Given N, m, and t, the expected probability of success of a single perfect classic table is:*

$$P = \frac{mt}{N}.$$

PROOF. The proof if straightforward since the mt keys of such a table are distinct. □

THEOREM 5. *Given N, m, ℓ, and t, the average cryptanalysis time of classic tables is:*

$$T = t \sum_{k=1}^{k=\ell} k \frac{mt}{N} \left(1 - \frac{mt}{N}\right)^{k-1}.$$

PROOF. To find a key in a set of tables we have to search for it in each table in sequence. An interesting fact about perfect classic tables is that it always takes t operations to search a key (a) when we find it, (b) when we do not find it, and even (c) when we have a false alarm. (a) When we find the key, we execute i operations until we find the matching end of chain, and then execute $t - i$ operations from the start of the chain to recover the key. (b) When we do not find the key, we just carry out t operations and never find a matching key. (c) A false alarm triggers the same sequence of operations than when we find the key, the difference is just that the key does not match. Note, however, that when a false alarm occurs, we need not search further in the table. If a key is in the table we can only find the correct end of chain since there can be no other chain that merges into the correct chain. Thus if a false alarm occurs, we know that the key is not in the table. The work we just spent verifying the false alarm is regained by not having to search further in this table. As a result, the number of operations for searching a key in a set of ℓ tables of m chains of length t is t times the average number of tables we have to search. Since each table has the same probability of containing the key and there is a finite number of tables, we have a truncated geometric distribution and thus, the average cryptanalysis time is simply:

$$T = t \sum_{k=1}^{k=\ell} kP \left(1 - P\right)^{k-1}.$$

Using Theorem 4, we find the expected result. □

Note that if the number of tables is such that the success rate is close to one, we can approximate the distribution with an untruncated one and find $T = \frac{N}{m}$.

To find the optimal performance of perfect classic tables we need now to find the maximum size of such tables. Unfortunately the calculation of the maximum number of non-merging chains of length t that can be generated is non-trivial. Therefore, we take an experimental approach and use the following strategy to generate a maximum of non-merging chains: starting from an initial element we generate a sequence of concatenated chains until a merge occurs with a chain which has already been generated. We then simply choose a random starting point and generate a new sequence of chains. The goal of this strategy is to avoid gaps between chains that are not a multiple of the chain length. Indeed, if all chains were to start at random points the space of possible chains would be more fragmented, leaving many short sequences of points that are shorter than a chain length and that can thus not be covered by a chain that does not merge with another one. We have experimented our strategy in a space of 10 million keys with various chain lengths. Experimental results shown in Table II indicate that the number of chains is roughly proportional to the inverse of t^2.

t'	10	20	30	40	50	100	200	400
m	229713	67719	32243	18766	12256	3190	816	195

Table II. The maximum number of chains decreases roughly with the square of the chain length (here $N = 10^7$)

2.3 Perfect DP Tables

THEOREM 6. *Given N and t', where t' is the average chain length or the inverse of the probability to find a distinguished point, the expected maximum number of chains per perfect table without merge is:*

$$m_{\max}(t') = \frac{N}{t'}(1 - e^{-1}).$$

PROOF. From the proof of Theorem 1, we know that the size of the image of a set mapped onto itself is $1-e^{-1}$ times the size of the set, when it is sufficiently large. If we consider that each chain maps a distinguished point into another distinguished point, the number of chains of such a table is equal to $1 - e^{-1}$ times the number of distinguished points. The number of distinguished points being $\frac{N}{t'}$, we have

$$m_{\max}(t') = \frac{N}{t'}(1 - e^{-1}).$$

□

THEOREM 7. *Given N, m, and t', the expected maximum probability of success of a single DP table is*

$$P_{\max}(t') = \frac{t''}{t'}(1 - e^{-1})$$

where t'' is the average chain length after merges have been removed.

PROOF. The proof is straightforward: each table contains $m_{\max}(t')t''$ keys on average, so $P_{\max}(t') = \frac{m_{\max}(t')t''}{N}$. From Theorem 6, we deduce that

$$P_{\max}(t') = \frac{m_{\max}(t')t''}{N} = \frac{t''}{t'}(1 - e^{-1}).$$

□

The non trivial part is to find out the average chain length of the perfect chains after the merges have been removed. Because they have more opportunities to do so, longer chains will more often merge with other chains thus clumping into large trees of long chains. When removing the merges, longer chains are thus removed more often than short ones, resulting in a reduced average length of the perfect chains.

We have computed the expected maximum chain length of DP tables by measuring a sequence of experiments. In a problem of size 10^7 we have chosen various average chain lengths t' and generated chains starting at each of the 10^7 keys. For all chains that merged, we have only kept the longest one (as suggested in [Borst et al. 1998] and [Standaert et al. 2002]) to create perfect tables. Table III illustrates the situation for various initial chain lengths.

t'	m (theory)	m (measured)	t'' (measured)
10	632120	630018	4.92
25	252848	252559	7.32
50	126424	126740	7.31
100	63212	63168	7.31

Table III. Maximum number of non-merging chains m in theory and experiment and average chain length before (t') and after (t'') removing merges in DP tables, for $N = 10^7$

3. TRADE-OFF CHARACTERISTIC

In this section we introduce a generic way of characterizing the different trade-offs and so comparing them.

3.1 Rainbow Tables

From Section 2, we know how to calculate the success rate and the cryptanalysis time of trade-off using rainbow tables. Thus, given a success rate, we plot the cryptanalysis time according to the memory size. Figure 1 shows that cryptanalysis time decreases with the square of the memory size, independently of the success rate. We can thus write the time-memory relation as:

$$T = \frac{N^2}{M^2}\gamma(P^*) \tag{12}$$

where $\gamma(P^*)$ is a factor that depends only on the success probability. It is interesting to note that for $P^* = 86\%$ which is the expected maximum probability of success

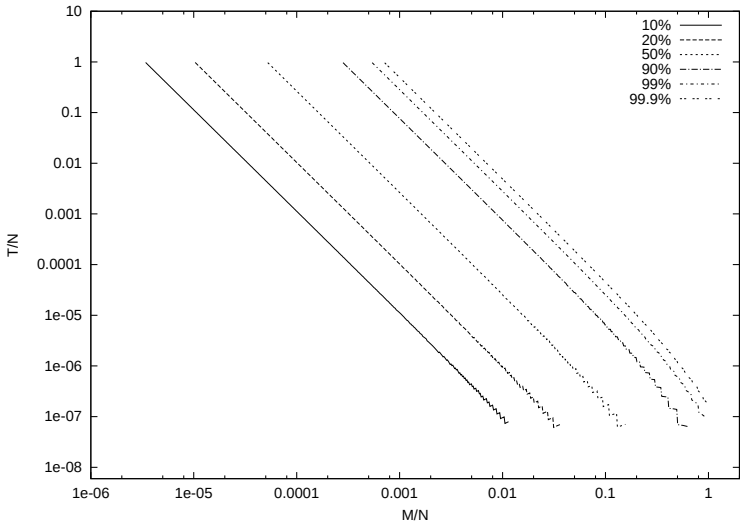

Fig. 1. Cryptanalysis time for rainbow tables, with various success rates

of a single rainbow table, the factor is equal to 1. In that case we find the typical trade-off which was already described by Hellman, that is $M = T = N^{\frac{2}{3}}$.

We now need a criterion to compare the trade-offs. We propose to use $\gamma(P^*)$ as the trade-off characteristic. The evolution of γ over a range of P^* shows how a variant is better than another. Figure 2 shows a plot of $\gamma(P^*)$ for rainbow tables. On the curve, a step occurs every time an additional table has to be used to achieve the given success rate.

In the following section, we compare the characteristic of rainbow tables with those of classic and DP tables.

3.2 Classic and DP Tables

The trade-off using classic or DP tables can also be characterized using the γ factor. Indeed both trade-offs follow the $T \propto N^2/M^2$ relation in a large part of the parameter space up to a factor which depends of the success rate and the type of trade-off. Thus, Figure 3, relying on Table II and Theorem 5, shows that classic tables have the same $T \propto N^2/M^2$ relation as rainbow tables if the success rate is not too small and the memory not too large. Note that in order to plot the graphs, the strategy is to use as many tables as required in order to reach the expected probability P^*. For large memories and small gains in time the trade-off can be implemented with one single perfect table. In that case the trade-off relation becomes $T \approx N/M$. We also notice on this graph that there are no solutions for small M. This is due to the fact that small memory implies long chains and perfect chains tend to loop when they get long. Actually, a well known result from hash table generation is that we can generate an average of $\sqrt{\frac{\pi}{2}\#\mathcal{H}}$ hashes until we get a first collision. Here $\#\mathcal{H}$ denotes the cardinality of the output space of the hash function. This means that we cannot have a configuration where t is larger than this value.

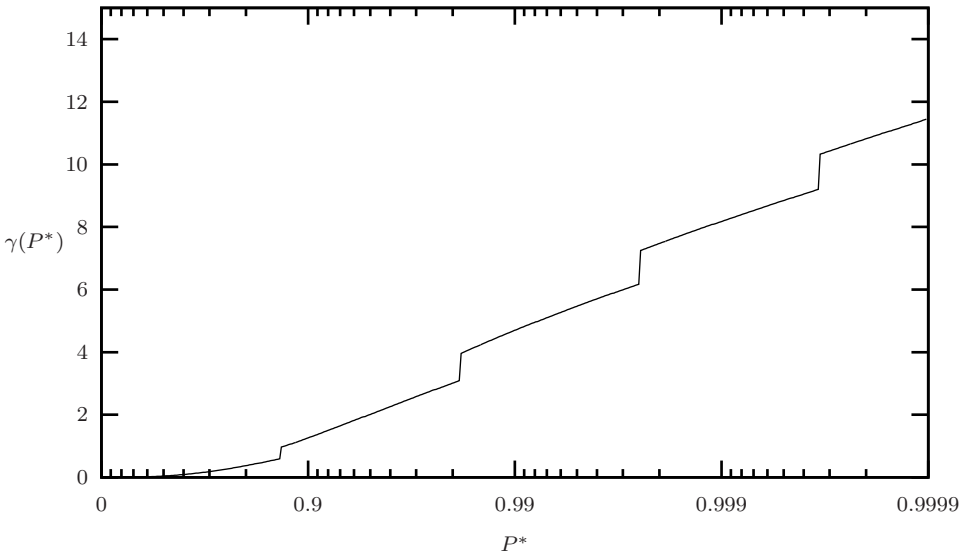

Fig. 2. Rainbow table characteristic.

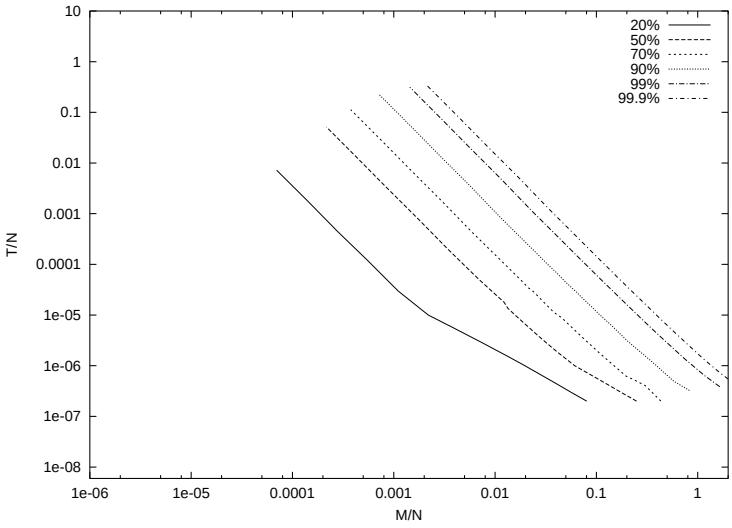

Fig. 3. Cryptanalysis time for classic tables, with various success rate

DP tables also follow the relation $T \propto N^2/M^2$. This is illustrated on Figure 4.

The characteristics of the rainbow, DP, and classic tables are depicted on Figure 5. It shows that perfect DP tables perform much worse than the other two variants, while rainbow tables outperform classic tables and DP tables for success rates above 80%. Below this limit, perfect classic tables are slightly better than perfect rainbow

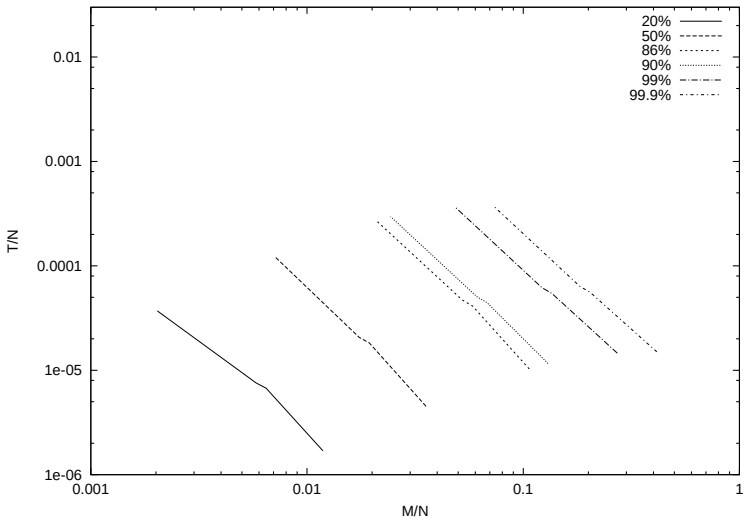

Fig. 4. Cryptanalysis time for DP tables, with various success rate

tables in terms of hash operations needed for cryptanalysis. However, the price of using classic tables is that they need t times more table look-ups. Since these do not come for free in most architectures (content addressable memory could be an exception), rainbow tables seem to be the best option in any case.

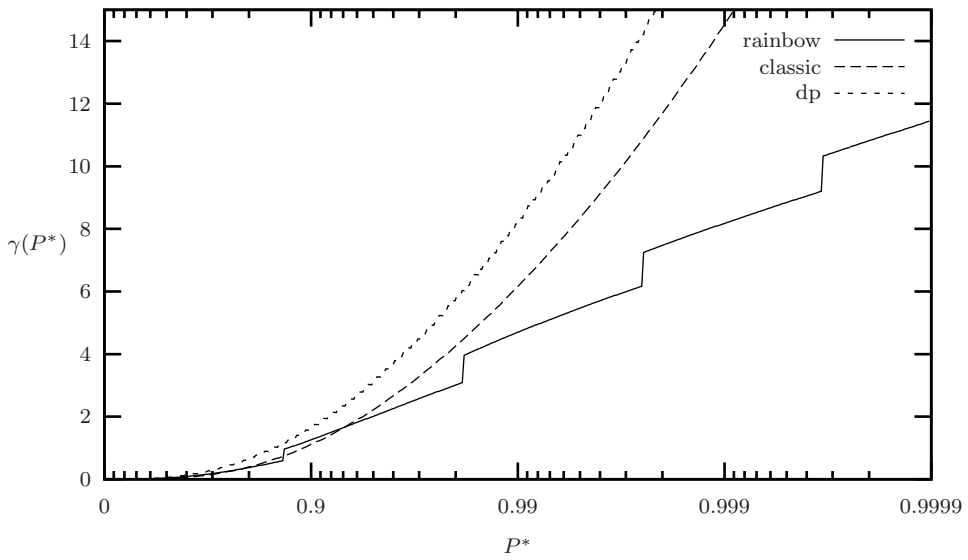

Fig. 5. Characteristics of rainbow, classic and DP tables compared.

4. CHECKPOINTS

4.1 False Alarms

We saw that given a ciphertext $C = \mathsf{S}_K(D)$, the on-line phase of the cryptanalysis works as follows: R is applied on C in order to obtain a key Y_1, and then the function f is iterated on Y_1 until matching any E_j. Let s be the length of the generated chain from Y_1:

$$C \quad \stackrel{\mathsf{R}}{\rightarrow} \quad Y_1 \quad \stackrel{f}{\rightarrow} \quad Y_2 \quad \stackrel{f}{\rightarrow} \quad \ldots \quad \stackrel{f}{\rightarrow} \quad Y_s$$

Then the chain ending with $E_j = Y_s$ is regenerated from S_j until yielding the expected key K. Unfortunately K is not in the explored chain in most of the cases. Such a case occurs when R collides: the chain generated from Y_1 merged with the chain regenerated from S_j after the column where Y_1 is. Such a case, so-called false alarm, requires $(t - s)$ encryptions to be detected.

Hellman [Hellman 1980] points out that the expected computation due to false alarms increases the expected computation by at most 50 percent. This reasoning relies on the fact that, for any i, $f^i(Y_1)$ is computed by iterating f i times. However $f^i(Y_1)$ should be computed from Y_i because $f^i(Y_1) = f(Y_i)$. In this case, the computation time required to reach a chain's end is significantly reduced on average while the computation time required to rule out false alarms remains unchanged. Therefore, false alarms can increase by more than 50 percent the expected computation. For example, Theorem 3 allows to show that the computation wasted during the recovering of Windows passwords [Oechslin 2003] due to false alarms increase by 125% the expected computation.

4.2 Ruling Out False Alarms Using Checkpoints

In order to rule out false alarms, our idea consists in defining a set of positions α_i in the chains to be checkpoints. We calculate the value of a given function G for each checkpoint of each chain j and store these $G(X_{j,\alpha_i})$ with the end of each chain $X_{j,t}$. During the on-line phase, when we generate $Y_1, Y_2, \ldots, Y_s$, we also calculate the values for G at each checkpoint, yielding the values $G(Y_{\alpha_i+s-t})$. If Y_s matches the end of a chain that we have stored, we compare the values of G for each checkpoint that the chain Y has gone through with the values stored in the table. If they differ at least for one checkpoint we definitely know that this is a false alarm. If they are identical, we cannot determine if a false alarm occured without regenerating the chain.

In order to be efficient, G should be easily computable and the storage of its output should require few bits. Below, we consider the function G such that $G(X)$ simply outputs the less significant bit of X. Thus we have:

$$\Pr\{G(X_{j,\alpha}) \neq G(Y_{\alpha+s-t}) \mid X_{j,\alpha} \neq Y_{\alpha+s-t}\} = \frac{1}{2}\left(1 - \frac{1}{2^{|K|}}\right) \approx \frac{1}{2}.$$

Note that the case $X_{j,\alpha} \neq Y_{\alpha+s-t}$ occurs when the merge appears after the column α (Figure 6), while the case $X_{j,\alpha} = Y_{\alpha+s-t}$ occurs when either K appears in the regenerated chain or the merge occurs before the column α (Figure 7).

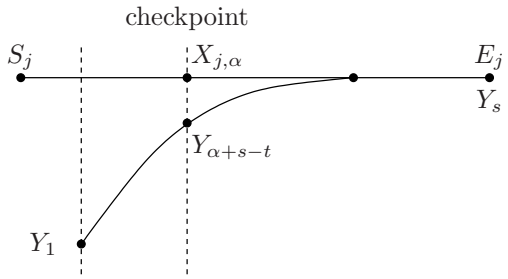

Fig. 6. False alarm detected with probability 1/2

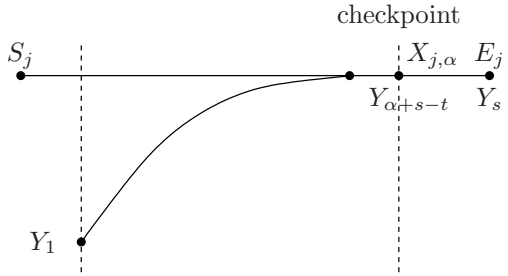

Fig. 7. False alarm not detected

4.3 Checkpoints in Rainbow Tables

Theorem 8 considers only one checkpoint and Corollary 2 extends this result to t checkpoints.

THEOREM 8. *Given N, m, t, and a checkpoint α, the work to rule out a false alarm when searching in column x is:*

$$Q(x) = \sum_{i=x}^{i=t} (i-1)\left(q_i - q_\alpha \cdot g_\alpha(t-i)\right),$$

where

$$q_c = 1 - \frac{m}{N} - \frac{(c-1)c}{t(t+1)}$$

and

$$g_\alpha(s) = \begin{cases} 0 \text{ if there is no checkpoint in column } \alpha, \\ 0 \text{ if } (\alpha+s) \leq t, \text{ i.e. the chain generated from } Y_1 \text{ does not reach column } \alpha, \\ \Pr\{G(X_{j,\alpha}) \neq G(Y_{\alpha+s-t}) \mid X_{j,\alpha} \neq Y_{\alpha+s-t}\} \text{ otherwise.} \end{cases}$$

PROOF. Let $Y_1 \ldots Y_s$ be a chain generated from a given ciphertext C. From (6), we know that the probability that the chain $Y_1 \ldots Y_s$ merges with a stored chain is q_{t-s+1}. The expected work due to a false alarm is therefore $q_{t-s+1}(t-s)$.

We now compute the probability that the checkpoint detects the false alarm. If the merge occurs before the checkpoint (Figure 7) then the false alarm cannot be detected. If the chain is long enough, i.e., $\alpha + s > t$, the merge occurs after the

checkpoint (Figure 6) with probability q_α. In this case, the false alarm is detected with probability $\Pr\{G(X_{j,\alpha}) \neq G(Y_{\alpha+s-t}) \mid X_{j,\alpha} \neq Y_{\alpha+s-t}\}$.

Given $g_\alpha(s)$,

$$g_\alpha(s) = \begin{cases} 0 \text{ if there is no checkpoint in column } \alpha, \\ 0 \text{ if } (\alpha+s) \leq t, \text{ i.e. the chain generated from } Y_1 \text{ does not reach column } \alpha, \\ \Pr\{G(X_{j,\alpha}) \neq G(Y_{\alpha+s-t}) \mid X_{j,\alpha} \neq Y_{\alpha+s-t}\} \text{ otherwise,} \end{cases}$$

we can rewrite $Q(x)$ as follows:

$$Q(x) = \sum_{i=x}^{i=t} (i-1)\left(q_i - q_\alpha \cdot g_\alpha(t-i)\right),$$

where $q_\alpha \cdot g_\alpha(t-i)$ is the work saved by the checkpoints. □

Corollary 2 extends the results to several checkpoints.

Corollary 2. *Given N, m, t, and some checkpoints, the work to rule out a false alarm when searching in column x is:*

$$Q(x) = \sum_{i=x}^{i=t} i \left(q_i - q_i \cdot g_i(t-i) - \sum_{j=i+1}^{j=t} \left(q_j \cdot g_j(t-j) \prod_{k=i}^{k=j-1} (1 - g_k(t-k)) \right) \right).$$

where

$$q_c = 1 - \frac{m}{N} - \frac{c(c-1)}{t(t+1)}$$

and

$$g_\alpha(s) = \begin{cases} 0 \textit{ if there is no checkpoint in column } \alpha, \\ 0 \textit{ if } (\alpha+s) \leq t, \textit{ i.e. the chain generated from } Y_1 \textit{ does not reach column } \alpha, \\ \Pr\{G(X_{j,\alpha}) \neq G(Y_{\alpha+s-t}) \mid X_{j,\alpha} \neq Y_{\alpha+s-t}\} \textit{ otherwise.} \end{cases}$$

Proof. The proof is similar to those of Theorem 8, except that each checkpoint must take into account whether or not the previous checkpoints detected the false alarms. □

4.4 Numerical Results

We use our technique to crack[3] Windows passwords, as proposed in [Oechslin 2003]. In this example the parameters are $N = 8.06 \times 10^{10}$, $t = 10000$, $m = 15408697$, $\ell = 4$, the function to invert is DES, and we use the function G as defined previously. In order to evaluate the performance of the checkpoints, we define the time gain of a trade-off over another trade-off. Let M, T, N and M', T', N' be the parameters of two trade-offs respectively, we define σ_T as follows:

$$T' = \sigma_T \cdot T.$$

[3] `http://lasecwww.epfl.ch/~oechslin/projects/ophcrack/`

The time gain of the second trade-off over the first one is straightforwardly defined by:

$$(1 - \sigma_T) = 1 - \frac{T'}{T}.$$

Figure 8 depicts both theoretical and experimental time gain of the rainbow tables with one checkpoint over the rainbow tables without checkpoints.

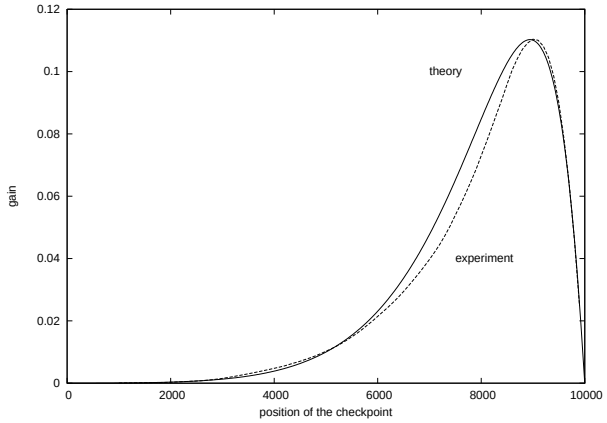

Fig. 8. Theoretical and experimental gain when one checkpoint is used

However, storing checkpoints consumes memory, which is not taken into account in the previous results. Consequently, in order to evaluate the real performance of the checkpoints, we define the memory cost of a trade-off over another trade-off. Let M, T, N and M', T', N' be the parameters of two trade-offs respectively, we define σ_M as follows:

$$M' = \sigma_M \cdot M.$$

The memory cost of the second trade-off over the first one is straightforwardly defined by:

$$(\sigma_M - 1) = \frac{M'}{M} - 1.$$

Thus, given a memory cost, we can compare the time gains when the additional memory is used to store chains and when it is used to store checkpoints. When a trade-off stores more chains, it implies a memory cost but, given that $T \propto N^2/M^2$, it also implies a time gain which is:

$$(1 - \frac{T'}{T}) = 1 - \frac{1}{\sigma_M^2}.$$

Instead of storing additional chains, the memory cost can be used to store checkpoints as we has seen above. The time gain of these two techniques are given in Table IV. Results are amazing: an additional 0.89% of memory saves about 10.99% of cryptanalysis time when the memory is used to store checkpoints, which is six

Number of checkpoints	1	2	3	4	5	6
Memory Cost	0.89%	1.78%	2.67%	3.57%	4.46%	5.35%
Gain (store chains)	1.76%	3.47%	5.14%	6.77%	8.36%	9.91%
Gain (store checkpoints)	10.99%	18.03%	23.01%	26.76%	29.70%	32.04%
Optimal checkpoints	8935	8565 9220	8265 8915 9370	8015 8655 9115 9470	7800 8450 8900 9250 9550	7600 8200 8700 9000 9300 9600
	± 5	± 5	± 5	± 5	± 50	± 100

Table IV. Cost and gain of using checkpoints in password cracking, with $N = 8.06 \times 10^{10}$, $t = 10000$, $m = 15408697$, and $\ell = 4$

times more than the 1.76% of gain that are obtained by using the same amount of memory to store additional chains.

Our checkpoints thus perform much better than the basic trade-off. As we add more and more checkpoints, the gain per checkpoint decreases. In our example it is well worth to use 6 bits of checkpoint values (5.35% of additional memory) per chain to obtain a gain of 32.04%.

The 0.89% of memory per checkpoint are calculated by assuming that the start and the end of the chains are stored in 56 bits each, as our example uses DES keys. However, the memory required to store the chains can be reduced using a few homemade techniques. In our Windows password example, there are about 2^{37} keys of 56 bits. Instead of storing the 56 bits, we store a 37 bit index, as suggested in [Biryukov et al. 2000]. From this index we take 21 bits as prefix and store only the last 16 bits in memory. We also store a table with 2^{21} entries that point to the corresponding suffixes for each possible prefix. Since rainbow tables allow us to choose the starting points at will, we can use keys with increasing value of their index. We use about 300 million starting points. This value can be expressed in 29 bits, so we only need to store the 29 lower bits of the index. The total amount of memory needed to store a chain is thus $29 + 16$ bits for the start and the end. The table that relates the prefixes to the suffixes incurs about 3.5 bits per chain. Altogether we thus need 49 bits per chain[4]. With these improvements, a bit of checkpoint data adds 2% of memory, but it is still well worth using three checkpoints of one bit each to save 23% of work.

Finally, for reasons of efficiency of memory access it may in some implementations be more efficient to store the start and the end of a chain (that is, its suffix) in multiples of 8 bits. If the size of some parameters does not exactly match the size of the memory units, the spare bits can be used to store checkpoints for free. In our case, the 29 bits of the chain start are stored in a 32 bit word, leaving 3 bits

[4]A simple implementation that stores the full 56 bits of the start and end chain would need 2.25 times more memory and be 5 times slower.

available for checkpoints.

5. CONCLUSION

We provided in this paper a thorough analysis of cryptanalytic time-memory trade-offs using perfect tables. For rainbow, DP, and classic variants, we supplied formulas to compute the optimal parameters that allows to reach the expected minimum cryptanalysis time. We also introduced the trade-off characteristic, which evaluates the trade-off efficiency. Finally, we introduced a new technique based on checkpoints that aims at reducing the time wasted to detect false alarms. We show that this technique has a real impact in practice.

We saw that computing formally the maximum table size and the average chain length in classic tables and DP tables respectively, is still an open work. Leading research on these questions may improve our knowledge on the behavior of cryptanalytic trade-offs. Another track to explore is the trade-off evaluation when look-ups cost is taken into account.

REFERENCES

Avoine, G., Junod, P., and Oechslin, P. 2005. Time-memory trade-offs: False alarm detection using checkpoints. In *Progress in Cryptology – Indocrypt 2005*. Lecture Notes in Computer Science, vol. 3797. Cryptology Research Society of India, Springer-Verlag, Bangalore, India, 183–196.

Barkan, E., Biham, E., and Shamir, A. 2006. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *Advances in Cryptology – CRYPTO'06*, C. Dwork, Ed. Lecture Notes in Computer Science. IACR, Springer-Verlag, Santa Barbara, California, USA.

Biryukov, A., Mukhopadhyay, S., and Sarkar, P. 2005. Improved time-memory trade-offs with multiple data. In *Selected Areas in Cryptography – SAC 2005*, B. Preneel and S. Tavares, Eds. Lecture Notes in Computer Science, vol. 3897. Springer-Verlag, Kingston, Canada, 110–127.

Biryukov, A. and Shamir, A. 2001. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In *Advances in Cryptology – ASIACRYPT'01*, C. Boyd, Ed. Lecture Notes in Computer Science, vol. 2248. IACR, Springer-Verlag, Gold Coast, Australia, 1–13.

Biryukov, A., Shamir, A., and Wagner, D. 2000. Real time cryptanalysis of A5/1 on a PC. In *Fast Software Encryption – FSE'00*, B. Schneier, Ed. Lecture Notes in Computer Science, vol. 1978. Springer-Verlag, New York, USA, 1–18.

Borst, J., Preneel, B., and Vandewalle, J. 1998. On the time-memory tradeoff between exhaustive key search and table precomputation. In *Symposium on Information Theory in the Benelux*, P. de With and M. van der Schaar-Mitrea, Eds. Veldhoven, The Netherlands, 111–118.

Denning, D. 1982. *Cryptography and Data Security*. Addison-Wesley, Boston, Massachusetts, USA, 100.

Fiat, A. and Naor, M. 1991. Rigorous time/space tradeoffs for inverting functions. In *ACM Symposium on Theory of Computing – STOC'91*. ACM, ACM Press, New Orleans, Louisiana, USA, 534–541.

Fiat, A. and Naor, M. 1999. Rigorous time/space tradeoffs for inverting functions. *SIAM Journal on Computing 29,* 3 (December), 790–803.

Hellman, M. 1980. A cryptanalytic time-memory trade off. *IEEE Transactions on Information Theory IT-26,* 4 (July), 401–406.

Hong, J. and Sarkar, P. 2005. New applications of time memory data tradeoffs. In *Advances in Cryptology – ASIACRYPT'05*, B. Roy, Ed. Lecture Notes in Computer Science, vol. 3788. IACR, Springer-Verlag, Chennai, India, 353–372.

Kim, I. and Matsumoto, T. 1999. Achieving higher success probability in time-memory trade-off cryptanalysis without increasing memory size. *IEICE Transactions on Communications/Electronics/Information and Systems E82-A,* 1 (January), 123–.

Kusuda, K. and Matsumoto, T. 1996. Optimization of time-memory trade-off cryptanalysis and its application to DES, FEAL-32, and Skipjack. *IEICE Transactions on Fundamentals E79-A,* 1 (January), 35–48.

Mentens, N., Batina, L., Preneel, B., and Verbauwhede, I. 2005. Cracking Unix passwords using FPGA platforms. SHARCS - Special Purpose Hardware for Attacking Cryptographic Systems.

Oechslin, P. 2003. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology – CRYPTO'03*, D. Boneh, Ed. Lecture Notes in Computer Science, vol. 2729. IACR, Springer-Verlag, Santa Barbara, California, USA, 617–630.

Quisquater, J.-J. and Delescaille, J.-P. 1989. How easy is collision search? Application to DES (extended summary). In *Advances in Cryptology – EUROCRYPT'89*, J.-J. Quisquater and V. Joos, Eds. Lecture Notes in Computer Science, vol. 434. IACR, Springer-Verlag, Houthalen, Belgium, 429–434.

Standaert, F. c.-X., Rouvroy, G., Quisquater, J.-J., and Legat, J.-D. 2002. A time-memory tradeoff using distinguished points: New analysis & FPGA results. In *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2002*, B. Kaliski, c. K. Koç, and C. Paar, Eds. Lecture Notes in Computer Science, vol. 2523. Springer-Verlag, Redwood Shores, California, USA, 593–609.

Wiener, M. and van Oorschot, P. 1999. Parallel collision search with cryptanalytic applications. *Journal of Cryptology 12,* 1 (March), 1–28.