A comparison of time-memory trade-off attacks on stream ciphers

Fabian van den Broek and Erik Poll

Institute for Computing and Information Sciences,
Radboud University Nijmegen, The Netherlands.

Abstract. Introduced by Hellman, Time-Memory Trade-Off (TMTO) attacks offer a generic technique to reverse one-way functions, where one can trade off time and memory costs and which are especially effective against stream ciphers. Hellman's original idea has seen many different improvements, notably the Distinguished Points attack and the Rainbow Table attack. The trade-off curves of these approaches have been compared in literature, but never leading to a satisfying conclusion. A new TMTO attack was devised for the A5/1 cipher used in GSM, which combines both distinguished points and rainbow tables, which we refer to as the Kraken attack.[1]. This paper compares these four approaches by looking at concrete costs of these attacks instead of comparing their trade-off curves. We found that when multiple samples are available the Distinguished Points attack has the lowest costs. The Kraken attack is an alternative to save more disk space at the expense of attack time.

1 Introduction

An attacker trying to break a cryptographic function can always try to either brute force the function, or precompute all possible values beforehand and store them in a large table, so every subsequent attack is a simple look-up. Most cryptographic functions are protected from these attacks by having a large enough key size or state size, which makes the time complexity or the storage requirements of such attacks too large in practice.

In 1980 Hellman caused a breakthrough by suggesting a Time-Memory Trade-Off attack which is probabilistic and falls somewhere in between a brute force attack and a precomputation attack. Hellman showed that using his attack he could reverse an n-bit key cipher, in $2^{2n/3}$ time complexity, by precomputing 2^n values and storing these in $2^{2n/3}$ values [1]. This made ciphers using keys that until then were thought large enough to prevent a brute-force attack suddenly susceptible to this new Time-Memory Trade-Off attack.

Later research into TMTO attacks led to many improvements on Hellman's attack. First came the Distinguished Points method, which reduced the number

[1] Correction after publication, Professor Jin Hong pointed out to us that the attack we call the Kraken attack was already suggested in 2006, and named a fuzzy rainbow table attack [10]. Hong et al. also found different results when comparing the different TMTO attacks [19,20,18]

of disk seeks and is referenced to Rivest [2]. Later Oechslin [3] devised a competing method with a slight speed-up, called Rainbow Table. The Rainbow Table attack seems to be better known, presumably due to its colorful name. Biryukov and Shamir [4] found that TMTO attacks were especially useful against stream ciphers, since an attacker can then make generic TMTO tables for a cipher which can be matched against any large enough sample of keystream, increasing the success chance for every sample. This new understanding directly led to new proposed attacks against one of the most widely deployed stream ciphers in the world: GSM's A5/1 cipher [5,6].

In 2010 researchers demonstrated a TMTO attack to break the A5/1 cipher of GSM [7]. This attack uses a new, unresearched, TMTO method which combines two important, but very different TMTO improvements; namely Distinguished Points and Rainbow Tables [8]. This new attack is called Kraken in this paper, after the name of the tool used to perform the actual attack.

It seems rather strange for these researchers to have chosen a new approach for their attack, so the question arises whether this new Kraken attack improves on the already existing attacks. This paper aims to research how much, if any, of an improvement this Kraken attack brings to the area of TMTO attacks.

Section 2 introduces the general idea of TMTO attacks. Section 3 introduces and analyzes the four TMTO attacks: Hellman's original attack [1] with Biryukov and Shamir's improvement for stream ciphers [4], Rivest's Distinguished Points approach [2], Oechslin's Rainbow Tables [3], and the first theoretical analysis of the Kraken attack (Section 3.4). We compare the TMTO attacks in Section 4, including an informal analysis on the chances of chain merges. Finally some ideas for future research are given and conclusions are drawn.

Related work Some of the discussed TMTO methods have previously been compared with each other. Most of these publications compare the trade-off curves for these attacks [4,9], which give the rate at which extra memory can be traded in for a reduced attack time. Such as $M^2T = N^2$ for both the Hellman and Distinguished Point attack, with M the memory cost, T the time cost of the online phase, and N the size of the state space. Our comparisons are not based on trade-off curves, because we feel that these curves hide too much of the real costs such attacks have, such as the seek times in the online attack, or the precomputation effort. Biryukov and Shamir compare Hellman's attack with Distinguished Points [4] and Erguler et al. compare Hellman's attack with Rainbow Tables in [9]. Barkan et al. [10] make the most complete comparison; within a new theoretic framework they find the Distinguished Points attack better than the Rainbow Table attack, mainly based on the possibility to shorten the stored values of a Distinguished Points attack. However, this comparison is still very broad and the question on which TMTO attack has the lowest costs, in terms of time and memory, seems to still be open to debate.

In 2008 Hong et al. [11] already combined Distinguished Points with Rainbow Tables, but in a different way than in the Kraken approach. Their combination does not improve on just Distinguished Points or Rainbow Table attacks.

To the best of our knowledge there has been no analysis of the Kraken approach. The attack is considered in work by Krhovjak et al. [12], where they use it as a practical example for an attack against A5/1, but they make no comparison with the earlier attacks.

2 Typical TMTO

Assume a scenario in which an attacker tries to break a known cryptographic function f for which he has obtained at least one sample of ciphertext y. His goal is to reverse the function f, i.e. to find an input x for which $y = f(x)$. This model covers different scenarios:

- Finding the preimage x of a hash function f for the hash value y.
- Finding the key x used to encrypt a known plaintext p to produce y, i.e. a key x such that $y = f(x) = encrypt_x(p)$, with *encrypt* e.g. a DES encryption.
- Finding the internal state used to encrypt a known plaintext p with a stream cipher. Here x is the internal state of cipher f and y is the corresponding keystream. So $f(x) = y$ and y is obtained by XORing cipherstream and known plaintext.

This paper is concerned with the third scenario, finding the "internal state" x of a stream cipher and not the key. Note that in many stream ciphers it is possible to retrieve the key that was used from a given internal state. The essential difference is that when reversing a stream cipher an attacker can construct tables which are more generic, so they can accept multiple samples from different plaintexts as explained in Section 3.1.

A typical TMTO attack consists of two phases: the first is the precomputation phase, often called the *offline phase*, while the second is referred to as the real-time, or *online phase*. In the offline phase, the attacker precomputes a large table (or sets of tables) using the function f he is trying to break, while in the online phase the attacker captures a sample of keystream and checks if this happens to be in his tables. If this attack is successful the attacker can learn the internal state x for which $y = f(x)$. We can evaluate these kinds of attacks by looking at different parameters and costs:

- N: the size of the state space.
- T (Attack time): This can be subdivided between the time for the offline phase, T_{pre}, in orders of magnitude, and the time for the online phase, which in turn can be subdivided into computation time T_c, measured in computation steps of f and seek time T_s, measured in number of disk seeks.
- M (Memory): memory cost of the attack.
- C (Coverage): the number of points from N covered by the tables.
- D (Data): number of usable data samples (y's) during the online phase.
- $\mathbb{P}$ (Chance of success): the chances of a collision between the observed ciphertext and the precomputed tables.
- ρ (Precomputation ratio): the ratio between the number of precomputed points from N and the total number of points N.

Intuitively, the chance of success $\mathbb{P}$ seems equal to the precomputation ratio, $\rho = C/N$, i.e. the number of points covered by the tables divided by the number of points in the search space. However, this is not exactly true for a number of reasons. Firstly, the tables can contain duplicate values. A certain number of duplicate values is to be expected when the coverage increases, however duplicates within the same table can lead to so called *chain merges*, which cause large parts of table rows to overlap. These chain merges will be discussed in more detail in the next section, but will for the most part be ignored in the analysis until Section 4, which details why it is hard to give an estimate on the occurrences of these chain merges. To stress this difference we introduce $\bar{C}$ and $\bar{\rho}$ as variants of the respective variables that do take chain merges into account.

Secondly, a definition of $\mathbb{P} = \rho$ assumes that all outputs of the cryptographic function f are equally likely, so all points in N have the same chance of occurring. This difference between $\mathbb{P}$ and ρ does not matter for our comparisons, but we will see in the practical example of Section 3.4 that this assumption does not always hold in practice.

Lastly, if an attacker has multiple samples, as he might have for a stream cipher, then the chance of success increases by a factor D, the number of samples.

3 The TMTO attacks

This section compares the costs of the four attacks: Hellman's original attack, Distinguished Points, Rainbow Tables, and Kraken. For each we give a theorem that states the cost for the general case with an arbitrary number of tables, followed by a corollary where we align some of the parameters to allow for an easy comparison. For these corollaries we assume an attacker abides by the $mt^2 = N$ rule, which will be introduced in Section 3.1, and precomputes enough points so that $D\rho = 1$. So, the corollaries normalize the attack costs, for easier comparison.

3.1 Hellman's original TMTO attack on stream ciphers

TMTO attacks were introduced by Hellman for attacking block ciphers [1]. In Hellman's attack the precomputation tables were created using a single piece of known plaintext. During the online phase an attacker needs to retrieve an encryption of that exact same piece of known plaintext in order to match it against his precomputed tables and have a chance on a successful attack. These precomputed tables are useless for other known plaintext/ciphertext pairs.

In 2000 Biryukov and Shamir [4] found that TMTO attacks against stream ciphers have an extra benefit: an attacker can create tables which are more generic, so any piece of known key stream can be matched to them. These samples can even be overlapping. If an attacker has created TMTO tables to look for keystream occurrences of n bits and he obtains e.g. $n + 6$ consecutive bits, this gives him 7 different keystream samples of length n to match with the results in his tables. Since every sample of known keystream has its independent chance

$$
\begin{array}{ccccccc}
x_0 & \rightarrow & f_i(x_0) & \rightarrow & f_i(f_i(x_0)) & \rightarrow \ldots \rightarrow & f_i^t(x_0) \\
x_1 & \rightarrow & f_i(x_1) & \rightarrow & f_i(f_i(x_1)) & \rightarrow \ldots \rightarrow & f_i^t(x_1) \\
x_2 & \rightarrow & f_i(x_2) & \rightarrow & f_i(f_i(x_2)) & \rightarrow \ldots \rightarrow & f_i^t(x_2) \\
\vdots & & \vdots & & \vdots & \ddots & \vdots \\
x_m & \rightarrow & f_i(x_m) & \rightarrow & f_i(f_i(x_m)) & \rightarrow \ldots \rightarrow & f_i^t(x_m)
\end{array}
$$

Fig. 1. A single $m \times t$ matrix of function f_i. Only the first and last points of each chain are stored.

of matching with the precomputed values, every sample increases the success chance of the attack. Alternatively, an attacker can make an estimate, D, on the expected number of samples he will be able to obtain in the real-time phase, this enables him to save a factor D on precomputation (both time and storage) to achieve the same success probability as that obtained with an attack on a cipher with $D = 1$. This effectively transforms the time-memory trade-off into a time-memory-'number of data samples' trade-off.

Hellman's attack on stream ciphers then goes as follows. In order to reverse the function f, a table is precomputed in the offline phase, for a single known plaintext. In order to cover as much of the N points of the search space as possible, an $m \times t$ matrix is computed, where the m rows consist of chains of length t and where each point in the chain is a new iteration of f on the result of the previous point (see Figure 1). Finally, only the begin point and end point of each chain are stored (ordered by the endpoints) as the precomputation table. In the rest of this article we will talk about precomputation *matrices* and *tables*, where matrices denote the temporary $m \times t$ precomputation chains and tables refer to the end product, essentially the compressed storage of the matrices. During the online phase, the attacker obtains keystream samples (e.g. by sniffing a known plaintext encryption, or because he can perform a chosen-plaintext attack). He then makes another chain of at most t iterations of applying the function f and for each iteration checks if the result matches one of the endpoints stored in his table. If this happens, he recomputes the chain starting from the corresponding begin point until the preimage of the ciphertext, thereby reversing function f in an attack time of order t in the online phase.

Adding more rows to the matrix computed in the offline phase will eventually cause duplicates, two duplicate points in different chains will cause the rest of these chains to cover the exact same points: the chains merge. Merging chains waste storage and precomputation effort on duplicate points. Hellman shows [1] that the probability of success is bounded by:

$$
(1/N) \sum_{i=1}^{m} \sum_{j=0}^{t-1} [(N - it)/N)]^{j+1} \leq \mathbb{P} \leq (mt/N). \tag{1}
$$

Hellman proves that this lower bound can be approximated to 3/4 for tables for which $mt^2 = N$. He argues that increasing m and t beyond $mt^2 = N$ is ineffective, since the chance of overlap only increases as m and t increase. Therefore

Hellman continues his analysis of using $m \times t$ matrices satisfying $mt^2 = N$. Most of the subsequent work on time-memory trade-offs copies this choice, although there is no real reason for this.

A single $m \times t$ matrix satisfying $mt^2 = N$ covers only $1/t$-th of the search space N. So, in order to cover a larger part of the search space, Hellman proposed to construct l different $m \times t$ matrices each using a variant of the f function, f_i. The function f_i is defined as $f_i(x) = h_i(f(x))$ where h_i is a simple output modification that is different for each i. In this way, all l tables only have a small chance of duplicate chains (only within a single table). Naturally there are still chances of duplicate points between different tables, but these will not cause chain merges and are thus not so costly.

Theorem 1. *The general costs for Hellman's attack adapted for stream ciphers are:*
$M = 2ml$ *entries,*
$T_c = tlD$ f_i*-computations,*
$T_s = tlD$ *seeks in tables of* m *entries.*

Proof. The memory costs equals the costs of one table, $2m$ *since it only stores the starting and endpoints, times the number of tables,* l*. Having* l *different tables also carries additional costs in terms of attack time during the online phase, since the attacker will now have to create* l *different chains of length* t *for every sample, so both* T_c *and* T_s *are in the order of* tlD f_i*-computations or seeks, respectively.*

In this general case, it might seem that the factor D only has a negative impact on the costs, however, the value for l, the number of tables, can be reduced with a factor D when attacking stream ciphers while the success chance remains the same.

Corollary 1. *When reversing a stream cipher, using* D *samples and the* $m \times t$ *matrices satisfy* $mt^2 = N$ *and precomputing enough points to satisfy* $D\rho = 1$*, the costs are:*
$T_{pre} = \mathcal{O}(N/D)$, $T_c = t^2$ f_i*-computations,*
$M = 2mt/D$ *entries,* $T_s = t^2$ *seeks in tables of* m *entries.*

Proof. The attacker makes l *tables, each with a different* f_i*. Since each table covers* $1/t$*-th of the search space (*$mt^2 = N$*) and the attacker expects* D *samples, he needs* t/D *different tables to cover an area of equal size to the search space. So, there are* $l = t/D$ *tables each covering* mt *points, which means the precomputation time* T_{pre} *is in the order of* N/D*, since* $mt^2 = N$ *(assuming that* $D \leq t$*). The costs for* M*,* T_c *and* T_s *follow by simply substituting* l *with* t/D *in Theorem 1*

The memory costs M are measured in entries. We are assuming two entries are needed per chain, which is an overestimate, since some bits can be spared by clever storage methods. The seek time T_s is measured in the number of disk seeks necessary for the attack. In his original analysis Hellman ignores the effect that the size of the tables might have on the time of an individual disk seek.

In order to achieve a more accurate measure we take the size of the tables into account, but we ignore the way the tables are organized on disk in our analysis.

Hellman's attack provides a time-memory trade-off controlled by choice of the chain length t. The table only stores two points for each chain, the begin and end point. As Theorem 1 shows, increasing t reduces the memory cost, but increases the time needed in the online phase, as more time is needed for computing the chain. Conversely, reducing t reduces the time in the online phase at the expense of higher memory cost. Note that if we choose $t = 1$ we have a dictionary attack, while if we choose $t = N$ we have a part of a brute-force attack.

3.2 Distinguished Points

The use of distinguished points was the first improvement on Hellman's approach. Hellman's analysis has a practical problem: there is a huge time difference between computing f_i and a disk seek to see if any $f_i(x)$ is stored in the precomputation table. In fact, Hellman's t^2 seeks in the precompution tables are extremely more expensive than the t^2 f_i-computations [2]. Since Hellman's analysis counted only the computation steps ($T = t^2$) the difference between theory and practice is very big.

In 1982, a solution was proposed referenced to Ron Rivest [2, page 100], namely to identify a subset of special points, called distinguished points. These points should be easily recognized, usually by a fixed prefix, such as the first k bits being '0'. In the offline phase, chains are computed until such a distinguished point is reached, and that point is then stored as the endpoint. If no distinguished point is reached for a certain number of maximum computation steps, the entire chain is dropped and a new one is computed. In the online phase, the attacker starts developing a chain from captured ciphertext until he reaches a distinguished point, and only then does he need to perform an expensive disk seek. If no distinguished point is encountered in the development of this chain after a predetermined number of steps, than this captured piece of ciphertext is not covered by the tables.

Rivest's approach reduces the number of disk seeks, since now only a single disk seek is needed for every chain that is computed during the online attack, instead of one disk seek for every link. This leads to matrices with chains of varying length. However on average the chain length will be $t = 2^k$.

Using distinguished points has one other benefit. When the precomputation tables are finished it is possible to remove all chain merges from the tables, simply by looking for identical end points. After all, if two chains within a table merge, they will end in the same distinguished point. There is not really an easy way to decide which chain to drop from the table, although an attacker could record the number of points in each chain, while precomputing, in order to keep the longest one. Alternatively, keeping both chains will increase the coverage of the search space (assuming different start points where chosen, then a least a single unique point is added to the coverage by keeping merging chains), at the cost of using storage for duplicate points.

Theorem 2. *The general costs for a Distinguished Points attack are:*
$M = 2ml$ *entries,*
$T_c = tlD$ f_i*-computations,*
$T_s = lD$ *seeks in tables of* m *entries.*

Proof. The memory costs remain exactly the same as in the previous theorem. The computation costs will also remain the same since a distinguished point will on average be encountered after t *steps. The disk-seek cost is now lowered to one disk seek per chain. Since the attacker needs to make* l *chains —one for each table— for every data sample, the seek time is* $T_s = lD$*.*

Corollary 2. *For the Distinguished Points attack, where the* $m \times t$ *matrices satisfy* $mt^2 = N$ *and precomputing enough points to satisfy* $D\rho = 1$*, the costs are:*
$T_{pre} = \mathcal{O}(N/D)$, $T_c = t^2$ f_i*-computations,*
$M = 2mt/D$ *entries,* $T_s = t$ *seeks in tables of* m *entries.*

Proof. The attacker again needs to create $l = t/D$ *tables, so both the precomputational work and memory storage remain the same. The costs for* T_c *and* T_s *are determined by substituting* t/D *for* l *in the preceding theorem.*

This approach can actually save some memory in practice, since k bits of every endpoint are constant and need not be stored. This makes the entries smaller, but the number of entries remains $2mt$. The time cost in the online phase also remains t^2 evaluations of an f_i, but now only t disk seeks are expected, instead of t^2 for Hellman's original attack: a disk seek is only needed when a distinguished point is encountered, which happens once for each chain (on average after per $t = 2^k$ computations), whereas in Hellman's original attack it has to be done for all points in the chain.

3.3 Rainbow Table

A different improvement on Hellman's approach, called Rainbow Table, was proposed by Oechslin in 2003 [3], with a factor-2 speed-up in the online phase, for an attack with single samples. Additionally, it has none of the overhead that Distinguished Points causes with its variable length, sometimes even unending, chains. However, the Rainbow Table attack is mostly known for its smaller chance of chain merge when less than N points are precomputed.

Oechslin suggested to precompute one large matrix (instead of t different ones) with a different f_i for every link in the chain. The name Rainbow Table stems from the idea of calling each simple output modification h_i a different color; each column has its own color, so the entire table looks like a rainbow. This prevents some chain merges, since now two chains can only merge if they reach the same value in the same column (i.e. while applying the same f_i). Duplicate points can, of course, still occur, but the penalty for these is not as severe since the chains will not merge if a duplicate happens in a different column.

$$
\begin{array}{ccccccccc}
x_0 & \to_1 & f_0(x_0) & \to_2 & f_1(f_0(x_0)) & \to_3 \ldots \to_t & f_t(f_{t-1}(\ldots f_0(x_0)\ldots)) \\
x_1 & \to_1 & f_0(x_1) & \to_2 & f_1(f_0(x_1)) & \to_3 \ldots \to_t & f_t(f_{t-1}(\ldots f_0(x_1)\ldots)) \\
x_2 & \to_1 & f_0(x_2) & \to_2 & f_1(f_0(x_2)) & \to_3 \ldots \to_t & f_t(f_{t-1}(\ldots f_0(x_2)\ldots)) \\
\vdots & & \vdots & & \vdots & \ddots & \vdots \\
x_{ml} & \to_1 & f_0(x_{mt}) & \to_2 & f_1(f_0(x_{mt})) & \to_3 \ldots \to_t & f_t(f_{t-1}(\ldots f_0(x_{mt})\ldots))
\end{array}
$$

Fig. 2. A $ml \times t$ rainbow matrix using t different f_i functions. Only the first and last points of each chain are stored.

Theorem 3. *The general costs for a Rainbow Table attack are:*
$M = 2ml$ *entries,*
$T_c = \frac{t(t+1)}{2} D$ f_i*-computations,*
$T_s = tD$ *seeks in a table of* lm *entries.*

Proof. In a Rainbow Table attack there is a single rainbow table which has ml *chains, of which only the first and last point are stored, so* $2ml$ *entries. These* ml *chains are for comparisons sake, so* ml *chains of length* t *have the same coverage as the* l $m \times t$ *matrices of other attacks. The online attack time becomes* $\frac{t(t+1)}{2} D$ *instead of* tlD*, because a different* f_i *is used for every link in the chain. So instead of computing a single chain* $(y, f(y), f^2(y), ..)$ *for every data sample an attacker now needs to evaluate* t *chains of a length ascending from* 1 *to* t f_i *calculations, with a different* f_i *for every link:*

$$
\begin{array}{ccccccc}
 & & & y & \to_{f_t} f_t(y) & & \uparrow \\
 & & y & \to_{f_{t-1}} f_{t-1}(y) & \to_{f_t} f_t(f_{t-1}(y)) & & t \\
 & \iddots & \vdots & \vdots & \vdots & & \\
y & \to_{f_0} \ldots & \to_{f_{t-1}} f_{t-1}(\ldots) & \to_{f_t} f_t(f_{t-1}(\ldots(f_0(y)\ldots)) & & & \downarrow
\end{array}
$$

For each of these t *chains, the end point needs to be looked up in the table, for each of the* D *data samples, which results in* tD *disk seeks.*

In order to compare this attack to the other approaches we need the matrix to cover an equal number of points. The other approaches use t $m \times t$ matrices. With a rainbow table there is only a single table, so this needs to cover mt^2 points. Keeping the chain length t, means the attacker will need mt entries in his table to cover mt^2 points. So we assume an $mt \times t$ matrix, with t different f_i's, as Figure 2 shows.

Corollary 3. *For the Rainbow Table attack, where the* $ml \times t$ *matrices satisfy* $mt^2 = N$ *and precomputing enough points to satisfy* $D\rho = 1$*, the costs are:*
$T_{pre} = \mathcal{O}(N/D)$, $\quad T_c = \frac{t(t+1)}{2} D$ f_i*-computations,*
$M = 2mt/D$ *entries,* $\quad T_s = tD$ *seeks in a table with* mt/D *entries.*

Proof. In the Rainbow Table case there is little difference in costs between the general case and the case where an attacker chooses m *and* t *to satisfy* $mt^2 = N$*, since there is only a single table and only the chain size determines the attack*

time. For comparison's sake, we use a rainbow table of dimensions $(mt/D) \times t$*, which covers an equal number of points as the previous stream cipher TMTO attacks, and keeps the values for* T_{pre} *and* M *equal. By substituting* l *with* t/D*, the memory costs are also fixed.*

Since D will generally be smaller than t, the number of disk seeks is an improvement when compared to a Hellman style attack, though not as much as the use of distinguished points. Also keep in mind that every table seek could be more costly when using a rainbow table, because of its larger size than the l tables used in the other approaches.

The Rainbow Table attack is most known for a smaller chance of chain merges, but the table defined in Corollary 3 will have a similar chance of chain merges than the previous attacks for $D = 1$. Because the same amount of points are precomputed in every f_i (all the points in a single rainbow table column, or all the points covered in 1 Hellman or distinguished point table) and a duplicate between those points causes a chain merge. When assuming more samples, or when precomputing fewer points, i.e. $D\rho < 1$, then the Rainbow Table attack will probably have fewer chain merges than the other TMTO attacks.

The online attack time of the Rainbow Table attack is only dependent on the chain length, ignoring the number of entries in the table, which causes the slight speed up for attacks where $D = 1$.

3.4 Generalized Kraken approach

In 2009, researchers started a project to break GSM's standard encryption cipher A5/1 in practice, using a combination of time-memory trade-off techniques. They proposed the joint creation of a set of TMTO tables to which everyone could contribute [13]. The idea was to share the intense computing burden of a TMTO's precomputation step by having everyone willing to participate perform a part of the computation on modern GPUs, and share their results over the Internet. In the end however, the project ended up using a set of tables being computed on a single computer. This set was dubbed "The Berlin Set" and its parameters are discussed in detail later in this section. First we focus on the general approach that was used in this attack.

In order to find the internal state of a generic stream cipher, the Kraken approach combines both distinguished points and rainbow tables in the table layout. This is done by first choosing distinguished points as bit strings starting with k zeros. Then, normal TMTO chains are computed by repeatedly applying f_i to random start points until the output is a distinguished point. The chain is then continued but now with a different f_i; in essence changing the rainbow color. This is repeated for a predetermined number, s, of rainbow colors ($f_0 \dots f_s$ functions), until a distinguished point is found while using the final f_s of this chain. This point is the endpoint of a chain and is stored together with the corresponding start point in the TMTO table. Figure 3 shows such a precomputation matrix. In order to match a sample y against a table during the online phase, s different chains need to be developed ranging in size from

$$
\begin{array}{l}
x_0 \rightarrow f_0(x_0) \rightarrow f_0(f_0(x_0)) \rightarrow^* k||y_{00} \rightarrow f_1(k||y_{00}) \rightarrow^* \ldots \rightarrow f_s(\ldots) \rightarrow k||y_{0s} \\
x_1 \rightarrow f_0(x_1) \rightarrow f_0(f_0(x_1)) \rightarrow^* k||y_{10} \rightarrow f_1(k||y_{10}) \rightarrow^* \ldots \rightarrow f_s(\ldots) \rightarrow k||y_{1s} \\
x_2 \rightarrow f_0(x_2) \rightarrow f_0(f_0(x_2)) \rightarrow^* k||y_{20} \rightarrow f_1(k||y_{20}) \rightarrow^* \ldots \rightarrow f_s(\ldots) \rightarrow k||y_{2s} \\
\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
x_m \rightarrow f_0(x_m) \rightarrow f_0(f_0(x_m)) \rightarrow^* k||y_{m0} \rightarrow f_1(k||y_{m0}) \rightarrow^* \ldots \rightarrow f_s(\ldots) \rightarrow k||y_{ms}
\end{array}
$$

Fig. 3. A Kraken matrix, where $k||y$ denotes a distinguished point with the first k bits '0'. Only the first and last points of each chain are stored. Note that each Kraken chain consists of s Distinguished Points chains.

t to st f_i-computations, analogously to the Rainbow Table online attack. On average this will lead to one distinguished point per f_i subchain, assuming that each possible output of f_i is equally likely. While applying f_s, the attacker can see if the resulting distinguished point matches the stored endpoint. The distinguished point found in the chain while applying f_{s-1}, needs to be developed further by applying f_s until the last distinguished point is found. This continues to the distinguished point found using the first f_0, which should require a chain of around st computation steps to match the final distinguished point with the stored endpoint, as Figure 4 shows. This approach can boil down to compressing s different Distinguished Points tables into one: Each chain basically consists of s subchains, depending on the choice of s and t. Intuitively, this means that the memory costs will be lowered by a factor s, but the attack time will increase by a factor s. This attack should keep all other advantages of a Distinguished Points attack, such as the easily identifiable chain merges. When compared with a Rainbow Table attack the number of chain merges should rise with a factor $t = t'/s$, where t' is the new total chain length, because the same f_i is used for each subchain.

The average length of each subchain, t, can be adjusted by choosing a different length of k for the k-bit distinguished point. The length of one full chain is equal to $t' = st$.

$$
\begin{array}{rcccccl}
 & & & & y & \rightarrow^*_{f_s} k||x_{0s} & \uparrow \\
 & & y & \rightarrow^*_{f_{s-1}} k||x_{1s-1} & \rightarrow^*_{f_s} & k||x_{1s} & \\
 & y & \rightarrow^*_{f_{s-2}} k||x_{2s-2} & \rightarrow^*_{f_{s-1}} k||x_{2s-1} & \rightarrow^*_{f_s} & k||x_{2s} & \text{s} \\
\iddots & \vdots & & \vdots & & \vdots & \\
y \rightarrow^*_{f_0} \ldots & \rightarrow^*_{f_{s-2}} k||x_{ss-2} & \rightarrow^*_{f_{s-1}} k||x_{ss-1} & & \rightarrow^*_{f_s} & k||x_{ss} & \downarrow
\end{array}
$$

Fig. 4. The online phase of a Kraken attack. Here $k||X$ denotes a distinguished point with the first k bits '0'. Only the last point of every chain is matched against the precomputation table.

Theorem 4. *The general costs for the Generalized Kraken attack are:*
$M = 2ml$ *entries,*
$T_c = \frac{s(s+1)}{2} tlD$ f_i*-computations,*
$T_s = slD$ *seeks in tables of* m *entries.*

Proof. The costs for memory use and disk seeks remain the same as in the Distinguished Points case. The computation costs are still based on the costs for matching a single sample against a single table multiplied with the number of tables and the number of samples. The attacker needs to make s *chains of sizes increasing from* t *to* st*, so in total* $\frac{s(s+1)}{2}t$ f_i*-computations, to match a single sample against a single table.*

In the Kraken attack we are faced with an additional variable s, which introduces a new Time-Memory Trade-Off within a TMTO attack. It also complicates matters when creating the accompanying corollary by increasing the possible choices. Here we choose two of the most obvious scenario's:

- the full chain length of the Kraken tables is as large as in the previous attacks, which leads to s more tables (Corollary 4, more tables),
- the sub chain length of the Kraken tables is equal to the chain length of the previous attacks, the full chains are s times larger than the previous attacks (Corollary 5, bigger tables).

Of course these only show two possible choices for s, t and m, of which the first scenario coincides with $m(st)^2 = N$ and the second with the familiar $mt^2 = N$.

Corollary 4 (more tables). *When reversing a stream cipher with the Kraken approach, using* D *samples and the* $m \times st$ *matrices satisfy* $m(st)^2 = N$ *and precomputing enough points to satisfy* $D\rho = 1$*, the costs are:*
$T_{pre} = \mathcal{O}(N/D)$, $T_c = \frac{(s+1)}{2}(st)^2$ f_i*-computations,*
$M = \frac{2mst}{D}$ *entries,* $T_s = s^2t$ *seeks in a tables of* m *entries.*

Proof. A single table will cover $m \times st$ *points, or* $1/st$ *of the key space* N *(since* $m(st)^2 = N$*), so* st *tables are needed to achieve enough coverage for* $D\rho = 1$*. Given that* f *is a stream cipher, an attacker can reduce the number of required tables to* $l = \frac{st}{D}$*. This means the memory costs will be* $M = 2m \times \frac{st}{D} = \frac{2mst}{D}$ *entries.*

The attacker needs a total of $\frac{s(s+1)}{2}t$ f_i*-computations, to match a single sample against a single table. Since there are* D *samples and* $\frac{st}{D}$ *tables, the total attack time* T_c *equals:* $\frac{(s+1)}{2}(st)^2$*. The attacker must create* s *separate chains for each table. During the online attack this comes down to* s *disk seeks per table, on* $\frac{st}{D}$ *tables and* D *samples gives* $T_s = s^2t$ *disk seeks within tables of* m *value pairs.*

Corollary 5 (bigger tables). *When reversing a stream cipher with the Kraken approach, using* D *samples and the* $m \times st$ *matrices satisfy* $mt^2 = N$ *and precomputing enough points to satisfy* $D\rho = 1$*, the costs are:*
$T_{pre} = \mathcal{O}(N/D)$, $T_c = \frac{(s+1)}{2}t^2$ f_i*-computations,*
$M = \frac{2mt}{sD}$ *entries,* $T_s = t$ *seeks in a tables of* m *entries.*

Proof. A single table will cover $m \times st$ points, or s/t of the key space N (assuming $s < t$), so t/s tables are needed to achieve enough coverage for $D\rho = 1$. Given that f is a stream cipher, an attacker can reduce the number of required tables to $l = \frac{t}{sD}$. This means the memory costs will be $M = 2m \times \frac{t}{sD} = \frac{2mt}{sD}$ entries.

The attacker needs a total of $\frac{s(s+1)}{2}t$ f_i-computations, to match a single sample against a single table. Since there are D samples and $\frac{t}{sD}$ tables, the total attack time T_c equals: $\frac{(s+1)}{2}t^2$. The attacker must create s separate chains for each table. During the online attack this comes down to s disk seeks per table, on $\frac{t}{sD}$ tables and D samples gives $T_s = t$ disk seeks, but again disk seeks within tables of M value pairs.

The scenario of Corollary 4 for Kraken uses the same sized matrices as that of Corollary 2 for the Distinguished Points. It needs s^2 more tables than the scenario of Corollary 5, which is reflected in all the costs. However, keep in mind that the value of t in Corollary 4 is s times higher than the value of t in Corollary 5.

The costs in Corollary 5 confirm our intuition of Kraken using s compressed distinguished points tables. So it can reduce memory costs a factor s at the price of increasing the computation cost in the online phase by a factor $\frac{s+1}{2}$ in comparison with Distinguished Points approach, which is better than our initial intuition.

Kraken in practice The Kraken approach was devised by researchers from the hacker community to demonstrate the weakness of the encryption used in GSM; the stream cipher A5/1. This stream cipher has an internal state of 64 bits, which is initiated with a 64 bit session key and a 22-bit, publicly known, frame number. The cipher then produces 328 bits of keystream of which the first 100 are discarded. Of the remaining 228 bits, the first half are used for the encryption of a packet on the uplink (mobile phone to cell tower) and the second half is used for encryption on the downlink (cell tower to mobile phone).

The natural assumption here is that the state space has size 2^{64}, but careful examination of the clocking function shows that a large part of the possible internal states are unreachable from any valid state. Several studies have measured the decline of possible states in the A5/1 cipher [14,8], and all of these find that only around 15% of all possible states are still viable after the initial 100 clockings. This means in practice that an attacker only needs to cover around 15% of the state space: $N \approx 2^{61.26}$.

In the attack against A5/1 the $f_i(x)$ is setting x in the internal state of A5/1, clocking it a 100 steps forward and then producing 64 bits of keystream, combined with some trivial output modification (the rainbow colors). These 64 output bits are then used to set the new internal state for the next round.

In the precomputation phase 40 independent tables ($l \approx 2^{5.3}$) were created, dubbed the Berlin set. As distinguished points were chosen those points starting with 12 zeros ($k = 12$). Eight rainbow colors were used per table ($s = 8$) and they differ for each table, so in total there are 320 different colors.

Table 1. Comparison of the different attacks, for $D\rho = 1$ and $mt^2 = n$.

TMTO technique	M	T_c	T_s
Hellman's attack	$2mt/D$	t^2	t^2 in m entries
Distinguished Points	$2mt/D$	t^2	t in m entries
Rainbow Table	$2mt/D$	$\frac{t(t+1)}{2}D$	tD in mt/D entries
Kraken (more tables, $t' = st$)	$2mt'/D$	$\frac{(s+1)}{2}t'^2$	st' in m entries
Kraken (bigger tables)	$2mt/sD$	$\frac{(s+1)}{2}t^2$	t in m entries

Every chain consists of eight subchains of average length t. Assuming each possible outcome of an f_i is equally likely: $t = 2^{12}$. So $t' = 8 \times 2^{12} = 2^{15}$.

Initially, every table was computed with 8,662,000,000, approximately 2^{33}, rows ($m = 2^{33}$). After which one of every two chains with duplicate end points was removed and the current set contains around 6,000,000,000 entries per table.

This means that for the Berlin set around $2^{15} \times 2^{33} \times 2^{5.3} = 2^{53.3}$ points were precomputed. The set ended up covering around $2^{15} \times 2^{32.5} \times 2^{5.3} = 2^{52.8}$ distinct points, so over 29% of the chains ended up merging with an earlier chain. This surprisingly high percentage can, in part, be explained by the state-space collapse of A5/1. However, it still shows that chain merges can indeed have a significant impact on a TMTO attack performance in practice.

With $N \approx 2^{61.26}$, the Berlin set has its parameters between the two discussed options in Corollaries 4 and 5: $mt^2 < N < m(st)^2$. The tables in the Berlin set take up around 1.6TB on disk and one attack with 51 samples (one packet in GSM is 114 bits, so 51 samples of 64 bits) can be performed within several seconds on high-end, but off-the-shelf hardware. Experiments with self-generated bursts put the success chance of the attack with 51 samples to around 20%.

4 Comparison

The main idea behind Kraken is to combine the benefits of both Distinguished Points (i.e. low number of disk seeks) and Rainbow Tables (i.e. fewer duplicates). The question is whether this really turns out beneficial. The cost of the different attacks are compared in Table 1, which lists the costs given in Corollaries 1 to 5. The two possible Kraken approaches are both shown, but Corollary 4 has st substituted for t', so its t' is comparable to the value of t for the other attacks.

The three classic attacks Hellman's attack (adapted for stream ciphers) is added in this table as a baseline, since the other attacks all improve on almost all costs. Between Distinguished Points and Rainbow Table it seems that a Rainbow Table is the best choice for $D = 1$, with only the disk seeks being more expensive due to the larger table. This makes Rainbow Tables the best choice for attacking block ciphers.

However, when attacking stream ciphers with multiple samples, $D > 1$, the comparison is not so simple. The online attack time and the number of disk seeks

for Rainbow Table, T_c and T_s, both increase beyond those of the Distinguished Points attack. Of course increasing D also decreases the size of the rainbow table used, making each table search cheaper, but generally seek time will be in the order of the logarithm of the table size, so this benefit is smaller than the increase in the number of disk seeks. Based on these calculations, the more samples are expected during the online attack, the more attractive the Distinguished Points approach becomes, compared to Rainbow Tables.

The Kraken attacks Our initial intuition that the Kraken approach is comparable to s Distinguished Points tables stored as one, seems validated when looking at the respective costs of both Kraken attacks and the Distinguished Points attack. If we take $s = 1$, then the Kraken approaches are the same as Distinguished Points, and their costs are identical. Another way to look at the Kraken approach is as a "bloated" rainbow table, with every rainbow color expanded from one column to t columns (on average). Looking at the costs for the online attack time for the Kraken approach, if we choose $t = 1$ and $s = t'$ (essentially a rainbow table), it almost compares to the online attack costs of a Rainbow Table attack, where the difference can be explained by a Rainbow Table attack having a single table instead of the $\frac{st}{D}$ or $\frac{t}{sD}$ tables of the respective Kraken attacks.

It is clear from this comparison that having Kraken tables of the more-tables variant is not the best choice. It has more disk seeks than the bigger-tables approach and the Distinguished Points attack, without the benefit of smaller memory costs.

Kraken vs the classics The Kraken attack can be tuned further by changing the s parameter. Basically, the Kraken attack moves in between the Distinguished Points and Rainbow Table attacks, guided by the value of s, where a higher choice of s will save memory costs, but increase the online attack costs.

From the two realistic Kraken approaches shown in the comparison table, only the bigger-tables approach is competitive in this analysis. If the memory costs are the single limiting factor for using the Distinguished Points or Rainbow Table attack, then this Kraken attack seems a good choice.

Although, with the continuous drop in the prices of memory such a scenario seems unlikely, so depending on the number of expected samples a Rainbow Table or a Distinguished Points attack is probably the better choice.

Comparing chain merges The comparison above is based on all attacks satisfying $D\rho = 1$, in other words the costs are compared when all attacks precompute the same amount of points. However, due to duplicates and chain merges not all precomputed points will be unique. It would be more fair if we compared the costs of the attacks when they all satisfy $D\bar{\rho} = 1$, so the number of precomputed unique values would be in the order of N/D.

However, it is hard to estimate the chances on chain merges in a general case for the different approaches. In essence this problem boils down to the expected

overlap between two paths (of length t for most approaches) within a digraph consisting of the N points of the search space as nodes and the current f_i as the edges between these nodes. This digraph is a directed pseudo forest, so every node has out-degree 1, meaning there can exist source nodes, but no sinks and from every node there exists a path leading to a cycle. An analysis of the number of expected duplicates, or analogously the number of unique values for a certain TMTO attack seems hard [15,16,17,10] and to our knowledge this is in fact an open problem for most TMTO attacks.

We can however make some assumptions over the chain merges for the different approaches, when they all precompute the same amount of points. We ignore single duplicate points and only look at chain merges, so only duplicates under the same f_i. Then by looking at the number of precomputed points per different f_i function, although ignoring many of the subtle differences between the TMTO attacks, can give an indication on the chances of chain merges.

When we assume that the distinguished points from a Distinguished Points attack are uniformly spread over the iterative function graph, then in general the number of duplicates in the Distinguished Points tables will be about the same as those in the Hellman attack. The number of duplicates in the Rainbow Table attack will in general be smaller than for the Hellman and the Distinguished Points attack, as long as the number of records in a rainbow table is smaller than the $m \times t$ points in a Hellman or distinguished point table.

When we look at the two possible approaches for the Kraken attack, then they are most easily compared to a Distinguished Points attack. The first Kraken approach ($m(st)^2 = N$, Corollary 4) uses s different colors inside an almost standard Distinguished Points matrix. So, only $1/s$ th of the points in a single table have the chance of leading to a chain merge, and we would therefore expect s less chain merges in this Kraken approach than in a Distinguished Points approach. The second Kraken approach ($mt^2 = N$, Corollary 5) compresses s different Distinguished Points tables into one, but because the end point of one of those Distinguished Points tables is the start point for the next, chain merges in one of these subchains will carry through the rest of the chain. Therefore, we can roughly estimate that this Kraken attack has around $s/2$ more duplicates due to chain merges than a Distinguished Points attack.

When we keep chain merges in mind, the comparison from Table 1 becomes more subtle, since it seems that the factor s extra costs in memory and disk seeks of the first Kraken approach compared to the second, is somewhat mitigated by having less duplicates, and thus more unique points in its tables and a higher success chance. However, since we have no hard way of quantifying the number of chain merges in these attacks, this analysis remains very tentative.

Both Distinguished Points and Kraken have one extra benefit when comparing chain merges. Both approaches have identifiable chain merges, which means that every chain merge will automatically end in the same end point. So by simply comparing end points all chain merges can be identified. With extra precomputation effort both approaches are able to replace one chain of every merging chain pair, with a new one, thereby increasing their coverage $\bar{C}$. Nat-

urally, both the increase in coverage and the amount of extra precomputation work are dependent on the number of chain merges.

5 Conclusions and directions for future research

We have presented the first analysis of the cost of the generalized form of the TMTO attack used to break the A5/1 cipher, which we have called Kraken. We have also given a first comparison of the costs of Kraken and three older TMTO attacks: Hellman's original attack, Distinguished Points, and Rainbow Tables.

Our comparison is more detailed than earlier work comparing these three older forms of attack. Most [4,9] earlier work compared the trade-off curves of these well known attacks. This tells us the rate at which extra memory can be traded in for a reduced time, but completely ignores some important costs, namely the precomputation, seek times, and the number of unique points covered by an attack. We do consider these costs in our comparison: for each attack we give the memory and time costs, split into precomputation time, online computation time, and number of disk seeks.

In our comparison in Section 4 the new Kraken attack performed fine, with the lowest memory cost of all attacks and the ability to identify chain merges as its major benefits. Only Distinguished Points seems a better choice in comparison, having a higher memory cost, but the lowest online attack costs. The more well-known Rainbow Tables are only interesting for attacks with only a single sample of plaintext-ciphertext known, as it is outperformed by Distinguished Points for multiple samples.

Another limitation of comparisons of trade-off curves for the different approaches is that these curves are invariably made under the assumption that the table sizes are always chosen so that $mt^2 = N$. We see no convincing reason to constraint the choice in parameters in this way. Hellman used the constraint $mt^2 = N$ to compute a nice bound for the chance of success of his attack, but other choices for m and t that do not satisfy this constraint might perform better in concrete instances.

One factor that we still have not been able to quantify precisely in our comparison is the chance of duplicates during the precomputation of the tables.

We conjecture that the effectiveness of the Kraken attack is in fact lower than our current results suggest when this number of duplicates values is taken into account. The informal analysis of the expected number of chain merges in Section 4 shows that Kraken has a higher chance of chain merges than the other attacks when the number of rainbow colors in the Kraken approach is chosen to achieve lower memory cost.

Estimating the chance of duplicates during precomputation is the most difficult aspect in achieving a fair comparison. Over 29% of all chains created in the Kraken tables ended up merging with existing chains, showing that chain merges can indeed be a significant factor when comparing TMTO attacks. We know no way to compute the expected number of chain merges for the general case, or indeed for any non-trivial practical cipher. Since theoretical analysis

of the chance of duplicates seems very difficult, we think that further research which collects empirical data of practical experiments in constructing TMTO tables may be the best way to shed light on this.

References

1. M. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401 – 406, Jul 1980.
2. D. Denning. *Cryptography and Data Security*. Addison-Wesley, 1992.
3. P. Oechslin. Making a faster cryptanalytic time memory trade-off. volume 2729 of *LNCS*, pages 617,630. Spinger, 2003.
4. A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 1–13. Springer, 2000.
5. Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In *FSE 2000*, pages 1–18. 2000.
6. E. Barkan, E. Biham, and N. Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. In *CRYPTO'03*, volume 2729 of *LNCS*. Springer, 2003.
7. K. Nohl and S. Munaut. Wideband GSM sniffing. Presentation at 26C3, `http://events.ccc.de/congress/2010/Fahrplan/events/4208.en.html`, 2010.
8. K. Nohl. A5/1 decrypt website, November 2012. `http://opensource.srlabs.de/projects/a51-decrypt/`.
9. I. Erguler and E. Anarim. A new cryptanalytic time-memory trade-off for stream ciphers. In *ISCIS 2005*, volume 3733 of *LNCS*, pages 215–223. Springer, 2005.
10. E. Barkan, E. Biham, and A. Shamir. Rigorous bounds on cryptanalytic time/memory tradeoffs. In *CRYPTO'06*, volume 4117 of *LNCS*. Springer, 2006.
11. J. Hong, K. Jeong, E. Kwon, I. Lee, and D. Ma. Variants of the distinguished point method for cryptanalytic time memory trade-offs. In *Information Security Practice and Experience*, volume 4991 of *LNCS*. Springer, 2008.
12. J. Krhovjak, O. Siler, P. Leyland, and J. Kur. TMTO attacks on stream ciphers theory and practice. *Security and Protection of Information 2011*, 2011.
13. K. Nohl. Cracking A5 GSM encryption. Presentation at HAR 2009, `https://har2009.org/program/events/187.en.html`, 2009.
14. J. Golic. *Cryptanalysis of Alleged A5 Stream Cipher*. 1997. `http://jya.com/a5-hack.htm`.
15. J. Hong. The cost of false alarms in Hellman and rainbow tradeoffs. *Designs, Codes and Cryptography*, 57:293–327, 2010.
16. P. Flajolet and A. Odlyzko. Random mapping statistics. In *EUROCRYPT 89*, volume 434 of *LNCS*, pages 329–354. Springer, 1990.
17. A. Fiat and M. Naor. Rigorous time/space tradeoffs for inverting functions. STOC '91, pages 534–541. ACM, 1991.
18. J. Hong and S. Moon. A comparison of cryptanalytic tradeoff algorithms. *Journal of Cryptology*, 26(4):559–637, 2013.
19. B.-I. Kim and J. Hong. Analysis of the non-perfect table fuzzy rainbow tradeoff. *Report 2012/612*, 2012. `http://eprint.iacr.org/`.
20. B.-I. Kim and J. Hong. Analysis of the non-perfect table fuzzy rainbow tradeoff. In C. Boyd and L. Simpson, editors, *Information Security and Privacy*, volume 7959 of *Lecture Notes in Computer Science*, pages 347–362. Springer Berlin Heidelberg, 2013.