Time–Memory Trade-Off Attack on the GSM A5/1 Stream Cipher Using Commodity GPGPU (Extended Abstract)

Jiqiang Lu(✉), Zhen Li, and Matt Henricksen

Infocomm Security Department, Institute for Infocomm Research,
Agency for Science, Technology and Research,
1 Fusionopolis Way, #11-01 Connexis, Singapore 138632, Singapore
{jlu,liz,mhenricksen}@i2r.a-star.edu.sg

Abstract. Time–memory trade-off (TMTO) cryptanalysis is a powerful technique for practically breaking a variety of security systems in reality. There are mainly four general TMTO cryptanalysis methods, namely Hellman table cryptanalysis, rainbow table cryptanalysis, thin rainbow table cryptanalysis and thick rainbow table cryptanalysis, plus a few supplementary techniques that can be combined with a general method to produce possibly distinct TMTOs, like distinguished points. In this paper, we present a unified TMTO cryptanalysis, which we call unified rainbow table cryptanalysis, basing it on a unified rainbow table, then we describe its general combination with distinguished points, and finally we apply unified rainbow table cryptanalysis to the A5/1 stream cipher being used in the Global System for Mobile Communications (GSM). On a general-purpose graphics processing unit (GPGPU) computer with 3 NVIDIA GeForce GTX690 cards that cost about 15,000 United States dollars in total, we made a unified rainbow table of 984 GB in about 55 days, and implemented a unified rainbow table attack that had an online attack time of 9 s with a success probability of 34 % (or 56 %) when using 4 (respectively, 8) known keystreams (of 114 bits long each). If two such tables of 984 GB were generated, the attack would have an online attack time of 9 s with a success probability of 81 % when using 8 known keystreams. The practical results show again that nowadays A5/1 is rather insecure in reality and GSM should no longer use it.

Keywords: Time–memory trade-off · Hellman table cryptanalysis · Rainbow table cryptanalysis · Stream cipher · A5/1 · GPGPU

1 Introduction

Generally, there are two elementary cryptanalysis techniques that can be applicable to any cryptosystem from a theoretical perspective, namely exhaustive key search and the dictionary attack, which require a negligible number of data but represent two extreme cases in terms of time and memory complexities: exhaustive key search requires a negligible memory, but has a time complexity of the

T. Malkin et al. (Eds.): ACNS 2015, LNCS 9092, pp. 350–369, 2015.
DOI: 10.1007/978-3-319-28166-7_17

same order as the key size of the concerned cryptosystem, while the dictionary attack has a negligible time complexity, but requires a memory of the same order as the key size of the concerned cryptosystem. As the two elementary cryptanalysis techniques have a time or memory complexity of the same order as the key size of the concerned cryptosystem, it is usually impossible to apply them in reality to break a real-world cryptosystem during its lifetime, since the key size of a cryptosystem is usually set to be larger than that required for the designed lifetime of the cryptosystem.

In 1980, Hellman [24] introduced a cryptanalytic time–memory trade-off (TMTO), known as Hellman table cryptanalysis; it requires a memory less than that for the dictionary attack to store some precomputation tables, called Hellman tables, and has an online attack time smaller than that for exhaustive key search. If the required memory is small enough to be realistic, Hellman table cryptanalysis may allow an attacker to break a cryptosystem within a reasonable time. In 2003, Oechslin [31] described a refinement of Hellman table cryptanalysis, called rainbow table cryptanalysis, which is based on a rainbow table that has a different structure with a Hellman table; in a straightforward way, rainbow table cryptanalysis can be twice as fast as Hellman table cryptanalysis under the same precomputation complexity. By combining rainbow table cryptanalysis with Hellman table cryptanalysis, in 2006 Barkan, Biham and Shamir [10] gave two variants of rainbow table cryptanalysis, called thin and thick rainbow table cryptanalyses, that are based on thin and thick rainbow tables respectively. Except the four general TMTO cryptanalysis methods, there are several complicated variants [17,21] with merely theoretical interest, plus a few supplementary techniques that aim to improve a general TMTO method from certain aspect(s) but usually at the sacrifice of other aspects more or less, for example, distinguished points [18] and checkpoints [6]. A general TMTO method can be used alone, however, a supplementary technique has to be used together with a general TMTO method. When applied to particular areas, TMTO has been extended to time–memory–data trade-off (TMDTO) [14], time–memory–processor trade-off [3], time–memory–key trade-off [11,13], etc.

The A5/1 stream cipher was designed in 1987 for use in the Global System for Mobile Communications (GSM) to provide data confidentiality in Europe and the U.S.A. It was reported that by 2011, about 4 billion GSM users relied on A5/1 to protect their communications [1]. Since A5/1 was reverse-engineered partially in 1994 and completely in 1999 [4,16], many cryptanalytic results have been published on it, including guess-and-determine attacks [12,19,22], correlation attacks [8,20,27] and TMTO attacks [9,15,19,30]. From a realistic viewpoint, almost all these attacks are somewhat academic in the sense of their impact on the real-world security of GSM, for they either require a large data complexity, ranging usually from 1,500 to 70,000 known keystreams, or have a long attack time, typically from several minutes to hours. The only exception is Nohl's recent attack [30] using thick rainbow table cryptanalysis with distinguished points (i.e., fuzzy-rainbow table cryptanalysis [10]), which uses 8 known keystreams and has an online attack time of about 10 s on a general-purpose graphics processing unit (GPGPU) and a

success probability of 87 %, plus 30 precomputation tables with a total of about 1.7 Terabytes, (Note that there were predecessors to this work with an inferior performance, say much larger precomputational workload). Nohl described mainly the details related to the structure of the precomputation tables, but did not disclose the online attack details (maybe because of his company's need for confidentiality). After an investigation, we find that Nohl's online attack on A5/1 is actually not so straightforward as a general TMTO attack from a theoretical viewpoint, and it seems that a crucial technique associated with the attack's precomputational workload and success probability is not disclosed.

In this paper, we make a few theoretical contributions to TMTO cryptanalysis and a practical contribution to the cryptanalysis of the GSM A5/1 stream cipher. First, inspired by thin and thick rainbow table cryptanalyses, we present a unified TMTO cryptanalysis, which we call unified rainbow table cryptanalysis, that is built on a unified rainbow table. Then, we discuss its general combination with distinguished points, as well as the resulting TMDTOs. On one hand the unified rainbow table itself represents a novel type of rainbow tables, and on the other hand it can be regarded as a unified model for rainbow-type tables, from which Hellman table, rainbow table, thin rainbow table and thick rainbow table can be easily obtained as special cases. Unified rainbow table cryptanalysis offers a trivially more comprehensive TMTO curve than the above four general methods, and its combination with distinguished points can offer a better TMDTO curve than any other general method except fuzzy-rainbow table cryptanalysis, but nevertheless it does not provide a better TMTO (or TMDTO) curve than the best previously known, thus it is merely of theoretical significance as a unified TMTO cryptanalysis. In particular, from its success probability formula, we can have success probability formulas for rainbow table, thin rainbow table and thick rainbow table cryptanalyses, which take into account some possible redundancy among different columns and the distinctions among different function variants. At last, we apply unified rainbow table cryptanalysis to A5/1, by working out a crucial theoretical technique. On a GPGPU computer with 3 NVIDIA GeForce GTX690 cards that cost about 15,000 United States dollars in total, we generated a unified rainbow table of 984 GB in about 55 days, and finally implemented a unified rainbow table attack that had an online attack time of 9 s with a success probability of 34 % (or 56 %) when using 4 (respectively, 8) known keystreams (of 114 bits long each). If two such tables of 984 GB were generated, the attack would have an online attack time of 9 s with a success probability of 81 % when using 8 known keystreams. Our attack on A5/1 is the first rainbow-type TMTO attack on A5/1 that reveals crucial theoretical techniques and implementation details, and the practical experimental results sufficiently show again that A5/1 is rather weak in terms of its realistic security.

The remainder of the paper is organised as follows. In the next section, we give the abbreviations and notation used throughout this paper and describe the A5/1 stream cipher. We present our unified rainbow table cryptanalysis in Sect. 3, describe its combination with distinguished points in Sect. 4, and discuss the resulting TMDTOs in Sect. 5. In Sect. 6, we describe our application of

unified rainbow table cryptanalysis to A5/1 and give our experimental results. Section 7 concludes this paper. Because of page constraints, we leave many other theoretical and implementation materials in the full version of this paper.

2 Preliminaries

In this section we describe the abbreviations and notation and the A5/1 cipher.

2.1 Abbreviations and Notation

The bits of a value are numbered from left to right, starting with 1. In the description below and throughout this paper, we assume $\mathtt{f} : \{1, 2, \ldots, N\} \rightarrow \{1, 2, \ldots, N\}$ is a one-way function, where N is a positive integer. We use the following abbreviations and notation.

CPU	Central Processing Unit
(GP)GPU	(General-Purpose) Graphics Processing Unit
GSM	the Global System for Mobile Communications
LFSR	Linear Feedback Shift Register
TM(D)TO	Time–Memory(–Data) Trade-Off
XOR/$\oplus$	bitwise logical exclusive OR
XNOR	the inverse of bitwise logical exclusive OR (XOR)
$\circ$	functional composition. When composing functions X and Y, Y$\circ$ X denotes the function obtained by first applying X and then applying Y. Sometimes we simply write X^i to denote $\overbrace{\mathrm{X} \circ \cdots \circ \mathrm{X}}^{i\ times\ \mathrm{X}}$, where i is a non-negative integer
$\mathtt{e}$	the base of the natural logarithm, ($\mathtt{e} = 2.71828\ldots$)
$\lfloor X \rfloor$	the largest integer that is less than or equal to X
$\mathcal{O}(X)$	a value that is of the same order as a value X
$X[i_1, \ldots, i_j]$	the j-bit string of bits $(i_1, \ldots, i_j)$ of a bit string X

2.2 The A5/1 Stream Cipher

A5/1 is a synchronous stream cipher, specifically a binary additive stream cipher. Its core part is a keystream generator, which is depicted in Fig. 1. The keystream generator is built on three Fibonacci linear feedback shift registers [5] (LFSRs) of 19, 22 and 23 bits long, which we denote by $R1, R2, R3$, respectively. As an LFSR with a primitive feedback polynomial can generate a maximum-length binary sequence, the three LFSRs have been set to have a primitive feedback polynomial each: The taps for $R1$ are at the 19th, 18th, 17th and 14th bits; the taps for $R2$ are at the 22nd and 21th bits; and the taps for $R3$ are at the 23th, 22th, 21th and 8th bits. The three LFSRs are mutually clocked in a stop/go fashion under a majority function `Maj` of three bits from the three LFSRs, respectively. The majority function `Maj` takes as input the 9-th bit of $R1$, the 11-th bit of $R2$ and

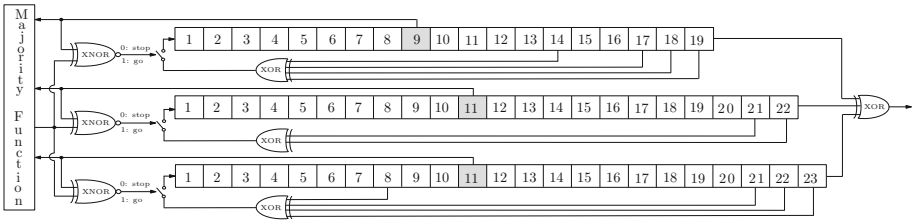

Fig. 1. The A5/1 keystream generator

the 11-th bit of $R3$, that is, $R1[9], R2[11], R3[11]$, and outputs the majority of the three input bits, more formally, it is defined to be $\texttt{Maj}(R1[9], R2[11], R3[11]) = (R1[9] \times R2[11]) \oplus (R1[9] \times R3[11]) \oplus (R2[11] \times R3[11])$.

In a clocking cycle, for each LFSR, if the input bit to the `Maj` function is equal to the output of the `Maj` function, then the LFSR is clocked, outputting the rightmost bit as the output bit, and the feedback value is given by its feedback polynomial and is fed into the leftmost bit of the LFSR; otherwise, the LFSR remains invariant, but still outputs the rightmost bit as the output bit. Then, the keystream generator outputs the XOR of the output bits of the three LFSRs. The `Maj` function guarantees that at least two of the three LFSRs are clocked every clocking cycle.

A GSM conversation is represented as a series of frames of 114 bits long each, and it is classified into two categories: downlink (base station to mobile) conversation and uplink (mobile to base station) conversation; the number of frames can be at most 22 bits long, that is from 0 to $2^{22} - 1$. GSM sends a frame of conversation every 4.615 ms, and thus a GSM conversation can usually be at most several hours unceasingly. For a conversation, GSM generates a 64-bit secret session key K from a master key, and uses the A5/1 stream cipher to encrypt every frame, with the frame number IV starting with a particular value generated by the system and being increased sequentially ($0 < IV < 2^{22}$).

When encrypting a frame of conversation, the A5/1 stream cipher first generates a 228-bit pseudorandom keystream through the keystream generator. The keystream generator takes as input the 64-bit secret session key K and the 22-bit publicly known frame number IV, and generates a 228-bit pseudorandom keystream in the following steps:

1. Initialization phase that involves: (a) setting the 64 bits of the three LFSRs to zero; (b) key setup that loads the 64-bit key into the LFSRs, with the `Maj` function ineffective; and (c) IV setup that loads the 22-bit IV into the LFSRs, with the `Maj` function ineffective.
2. 100 clockings on the three LFSRs with the `Maj` function effective and without outputting the output of the keystream generator.
3. 228 clockings on the three LFSRs with the `Maj` function effective and outputting the output of the keystream generator.

A 228-bit pseudorandom keystream is generated after Step 3. The first 114 bits of a 228-bit keystream are used for downlink conversation and the last 114 bits are used for uplink conversation. The A5/1 stream cipher encrypts a frame of conversation by XORing it with a 114-bit keystream, and produces a frame of ciphertext. Decryption is identical to encryption.

We refer to the internal state of the three LFSRs immediately after Step 1 (i.e., at the end of the initialization phase) as the initial state of a keystream.

3 The Unified Rainbow Table Cryptanalysis

In this section we present unified rainbow table cryptanalysis. Like the dictionary attack and the four general TMTO methods, unified rainbow table cryptanalysis also consists of two phases: offline precomputation phase to build one or more tables, and online attack phase to search the correct solution. We start with generating a unified rainbow table.

3.1 A Unified Rainbow Table

We define five parameters m, s, v, r, t, and each parameter is a positive integer.

- m: the number of starting points in a unified rainbow table.
- s: the number of the $\mathtt{f}$ variants used, namely $\mathtt{f}_1, \mathtt{f}_2, \ldots, \mathtt{f}_s$, each constructed usually by a functional composition of the $\mathtt{f}$ function with a simple operation (called a color sometimes [10]), say an XOR with a unique number [32] or a rotation of a unique number of bits [24].
- v: the number of the $\mathtt{f}_i$ variant in each sector of the adjacent columns $(\mathtt{f}_i, \mathtt{f}_i, \ldots, \mathtt{f}_i)$ with the same $\mathtt{f}_i$ variant.
- r: the number of the sector $(\mathtt{f}_s)^v \circ \cdots \circ (\mathtt{f}_2)^v \circ (\mathtt{f}_1)^v$.
- t: the total number of appearances of the $\mathtt{f}$ variants, that is equal to rsv.

The attacker builds a unified rainbow table of size $m \times t$ by applying the following procedure:

1. Choose m starting points uniformly at random from the domain $\{1, 2, \ldots, N\}$, and we denote them by SP_1, SP_2, …, SP_m.
2. For $1 \le i \le m$:
   (a) Let $x_{i,0} = \mathrm{SP}_i$.
   (b) For $q = 1, 2, \ldots, r$:
   For $j = 1, 2, \ldots, s$:
   For $l = 1, 2, \ldots, v$:
   – Compute $x_{i,(q-1)\cdot sv+(j-1)\cdot v+l} = \mathtt{f}_j(x_{i,(q-1)\cdot sv+(j-1)\cdot v+l-1})$.
   – If $(q = r)$ and $(j = s)$ and $(l = v)$, the value $x_{i,rsv}$ is termed the endpoint corresponding to SP_i; and we denote it by EP_i, that is $\mathrm{EP}_i = x_{i,rsv}$.

3. Store the m pairs $(\mathrm{SP}_i, \mathrm{EP}_i)$ into a table sorted by the values of the endpoints, (and discard the intermediate values $x_{i,1}, x_{i,2}, \ldots, x_{i,rsv-1}$). The resulting table is called a unified rainbow table. This completes the construction of a unified rainbow table.

In short, the series of $\mathtt{f}$ variants used to generate a chain is $[(\mathtt{f}_s)^v \circ \cdots \circ (\mathtt{f}_2)^v \circ (\mathtt{f}_1)^v]^r$, or more intuitively

$$\overbrace{\underbrace{\mathtt{f}_1 \ldots \mathtt{f}_1}_{v\ times}\underbrace{\mathtt{f}_2 \ldots \mathtt{f}_2}_{v\ times} \ldots \underbrace{\mathtt{f}_s \ldots \mathtt{f}_s}_{v\ times}}^{1}\overbrace{\underbrace{\mathtt{f}_1 \ldots \mathtt{f}_1}_{v\ times}\underbrace{\mathtt{f}_2 \ldots \mathtt{f}_2}_{v\ times} \ldots \underbrace{\mathtt{f}_s \ldots \mathtt{f}_s}_{v\ times}}^{2} \ldots \overbrace{\underbrace{\mathtt{f}_1 \ldots \mathtt{f}_1}_{v\ times}\underbrace{\mathtt{f}_2 \ldots \mathtt{f}_2}_{v\ times} \ldots \underbrace{\mathtt{f}_s \ldots \mathtt{f}_s}_{v\ times}}^{r}.$$

The unified rainbow table requires a memory of m units, and has a time complexity of $m \times r \times s \times v = mt$ computations of the $\mathtt{f}$ function, where we assume a computation of a variant $\mathtt{f}_j$ of the $\mathtt{f}$ function is approximately equal to a computation of the $\mathtt{f}$ function in terms of computational complexity, as in previous work [10,24,31].

3.2 Online Attack Procedure

Given an inversion target $y = \mathtt{f}(x)$, below we will describe an online attack procedure based on a unified rainbow table of Sect. 3.1. The online attack procedure is devised according to the structure of the unified rainbow table, so that certain previously computed values can be reused to save time.

The attacker tries to find the correct solution x by checking the unified rainbow table column by column in the following procedure. Recall that $t = rsv$.

1. For $j = s, s-1, \ldots, 1$:
   (a) Apply the simple operation concatenated with $\mathtt{f}$ for constructing $\mathtt{f}_j$ to the given output y, and we denote the resulting value by $\widehat{y}_0$.
   (b) For $l = 1, 2, \ldots, v$:
      i. Set $y_0 = \widehat{y}_0$ if $j = s$; otherwise, compute $y_0 = \overbrace{(\mathtt{f}_s)^v \circ \cdots \circ (\mathtt{f}_{j+1})^v}^{from\ (\mathtt{f}_{j+1})^v\ until\ (\mathtt{f}_s)^v}(\widehat{y}_0)$.
      ii. For $q = 1, 2, \ldots, r$:
         A. Check whether y_{q-1} is an endpoint in the unified rainbow table. If y_{q-1} does not match any endpoint, we execute the next sub-step. However, if y_{q-1} atches an endpoint, EP_i say, then re-generate $x_{i,(r-q)\cdot sv+jv-l}$ from the corresponding starting point SP_i as $(\mathtt{f}_1)^{v-l} \circ [(\mathtt{f}_s)^v \circ \cdots \circ (\mathtt{f}_2)^v \circ (\mathtt{f}_1)^v]^{r-q}(\mathrm{SP}_i)$ if $j = 1$, or $(\mathtt{f}_j)^{v-l} \circ (\mathtt{f}_{j-1})^v \circ \cdots \circ (\mathtt{f}_2)^v \circ (\mathtt{f}_1)^v \circ [(\mathtt{f}_s)^v \circ \cdots \circ (\mathtt{f}_2)^v \circ (\mathtt{f}_1)^v]^{r-q}(\mathrm{SP}_i)$ if $j \neq 1$. Finally, $x_{i,(r-q)\cdot sv+jv-l}$ is likely to be the correct solution x, (Note that $x_{i,(r-q)\cdot sv+jv-l}$ may be a false alarm, and under this situation we need do extra work to test whether the recovered $x_{i,(r-q)\cdot sv+jv-l}$ is a false solution). If $x_{i,(r-q)\cdot sv+jv-l}$ is the correct solution, terminate the procedure.
         B. If $q \neq r$, compute $y_q = (\mathtt{f}_s)^v \circ \cdots \circ (\mathtt{f}_2)^v \circ (\mathtt{f}_1)^v(y_{q-1})$.
      iii. If $l \neq v$, compute $\widehat{y}_0 = \mathtt{f}_j(\widehat{y}_0)$.
2. The attack fails if the correct solution is not found after the above steps.

3.3 Online Time Complexity

Before further proceeding, we emphasize two points that are throughout this paper. As in previously published TMTO or TMDTO methods [7,10,13,14,24,31]: first, we consider the online time complexity of the unified rainbow table attack as well as other TMTO attack methods in the worst scenario where the correct solution to the inversion target is found from the very first column of a precomputation table and no false alarm is taken into account; second, we consider the memory complexity in the unit of the number of starting-endpoint pairs, rather than the optimised memory complexity with certain sophisticated techniques like endpoint truncation [10,15]. Anyway, a recently emerging direction is to study the expected online time complexities and the optimised memory complexity, as done in [6,25,26]; we leave such studies as future work.

From the equations $y_q = (\mathtt{f}_s)^v \circ \cdots \circ (\mathtt{f}_2)^v \circ (\mathtt{f}_1)^v(y_{q-1})$ and $\widehat{y}_0 = \mathtt{f}_j(\widehat{y}_0)$ in Steps 1(b)(ii)(B) and 1(b)(iii), we can see that some previously computed values (i.e. y_{q-1} and $\widehat{y}_0$) are re-used during the next iteration, which enables us to save online attack time.

A detailed analysis reveals that the unified rainbow table attack has a total time complexity of approximately $\sum_{j=1}^{s-1}\sum_{l=1}^{v}[(s-j)\cdot v]+\sum_{j=1}^{s}\sum_{l=1}^{v}\sum_{q=1}^{r-1} sv+\sum_{j=1}^{s}\sum_{l=1}^{v-1} 1 = \frac{t^2}{r} - \frac{t^2}{2r^2} - \frac{t^2}{2sr^2} + \frac{t}{r} - s$ computations of the $\mathtt{f}$ function and $\sum_{j=1}^{s}\sum_{l=1}^{v}\sum_{q=1}^{r} 1 = vrs = t$ table look-ups.

3.4 Success Probability

The success probability of the unified rainbow table attack is determined by the coverage rate of the unified rainbow table, that is defined to be the proportion of the number of distinct values for both starting points and intermediate values associated with the generation of the table to the size N of the domain.

As in previous work, we assume that the $\mathtt{f}$ function as well as its variants $\mathtt{f}_i$'s is a random function. Besides, in order to obtain an as accurate as possible formula, we need to consider the difference between two different function variants $\mathtt{f}_i$ and $\mathtt{f}_j$, which depends on how these function variants are constructed from the $\mathtt{f}$ function. If the function composed with $\mathtt{f}$ to construct a function variant is like an XOR operation with a unique integer, then we definitely have $\mathtt{f}_i(x) \neq \mathtt{f}_j(x)$ for any x when $1 \leq j \neq i \leq s$; however, if it is like a rotation of a unique number of bits, we may have $\mathtt{f}_i(x) = \mathtt{f}_j(x)$ for an x, and if there are many such x's then the function variants are unwanted. Thus, below we assume the first case for constructing the function variants. As a result, we obtain the following approximate formula on the expected success probability of the unified rainbow table attack.

Theorem 1. *If $\mathtt{f}$, as well as its variants $\mathtt{f}_1, \mathtt{f}_2, \ldots, \mathtt{f}_s$ constructed each by composing $\mathtt{f}$ with a simple function such that $\mathtt{f}_i(x) \neq \mathtt{f}_j(x)$ for any x when $1 \leq j \neq i \leq s$, is modeled as a random function mapping the set $\{1, 2, \ldots, N\}$ into itself, and the correct solution to the inversion target is chosen uniformly*

from the same set, then a unified rainbow table attack based on one unified rainbow table with parameters m, s, v, t has a success probability of approximately $\frac{\sum_{k=0}^{t-1} m_k}{N}$, where m, s, v, t are defined in Sect. 3.1 and m_k is computed as follows:

$$\overline{m}_0 = \widehat{m}_0 = m_0 = N \cdot (1 - e^{-\frac{m}{N}}); \overline{m}_{k+1} = N \cdot (1 - e^{-\frac{\overline{m}_k}{N}});$$

$$\alpha = \lfloor \frac{k+1}{s \cdot v} \rfloor, \beta = \lfloor \frac{k+1-\alpha \cdot s \cdot v}{v} \rfloor, \delta = k+1-\alpha \cdot s \cdot v - \beta \cdot v = (k+1) \mod v,$$

$$\widehat{m}_{k+1} = \begin{cases} \overline{m}_{k+1} \cdot (1 - \frac{\sum_{i=0}^{\alpha-1} \sum_{j=0}^{v-1} \widehat{m}_{i \cdot s \cdot v + \beta \cdot v + j}}{N}), & \text{if } \delta = 0; \\ N \cdot (1 - e^{-\frac{\widehat{m}_k}{N}}) \cdot (1 - \frac{\sum_{i=0}^{\alpha-1} \sum_{j=0}^{v-1} \widehat{m}_{i \cdot s \cdot v + \beta \cdot v + j} + \sum_{j=0}^{\delta-1} \widehat{m}_{\alpha \cdot s \cdot v + \beta \cdot v + j}}{N}), & \text{if } \delta \neq 0. \end{cases}$$

$$m_{k+1} = N \cdot (1 - e^{-\frac{\widehat{m}_k}{N}}) \cdot (1 - \frac{\sum_{j=0}^{k} m_j}{N}).$$

3.5 TMTO Curve

Above we have described a unified rainbow table attack based on a unified rainbow table as well as its success probability. However, because of possible duplication of values associated with a precomputation table, the gain on its coverage rate is not as much as before when the number of starting points goes beyond a threshold value specified by a matrix stopping rule [14] theoretically. Alternatively, an attack may need more than one precomputation tables to reach a large success probability. This results in the following TMTO curve of unified rainbow table cryptanalysis.

Theorem 2. *Suppose P represents computational complexity for (offline) precomputation, T represents online time complexity, M represents memory complexity (for offline precomputation, since the memory complexity of online attack is negligible generally), s and r are those parameters used for unified rainbow table cryptanalysis. Then, unified rainbow table cryptanalysis approximately meets a time–memory trade-off $T \cdot M^3 = \frac{N^3}{s \cdot r} - \frac{N^3}{2s \cdot r^2} - \frac{N^3}{2s^2 \cdot r^2} + \frac{M \cdot N^2}{s \cdot r} - M^2 \cdot N$, where $P = N$, $M \leq N$ and $1 \leq s \cdot r \leq \frac{N}{M}$, plus $\frac{N^2}{s \cdot M^2}$ table look-ups.*

Furthermore, a simple analysis reveals the following result:

Corollary 1. *Hellman table, rainbow table, thin rainbow table, thick rainbow table and unified rainbow table cryptanalyses (can) have a TMTO curve $T \cdot M^2 = \mathcal{O}(N^2)$. Generally, rainbow table cryptanalysis has the best TMTO curve $2T \cdot M^2 \approx N^2$ among the five general TMTO methods.*

4 Unified Rainbow Table Cryptanalysis with Distinguished Points

When attacking a practical cryptosystem in reality, a huge precomputation table is usually required and stored in a hard disk, which makes a table look-up much

more costly than a computation of the $\mathtt{f}$ function in terms of time, thus the supplementary technique of distinguished points is often used to speed up an online attack. Considering this, we briefly discuss a combination of unified rainbow table cryptanalysis with distinguished points in this section.

4.1 A Combination

Unified rainbow table cryptanalysis can be combined with distinguished points in the following way, where DP represents a distinguished point and two values represented by DP may be different, ($1 \leq i \leq m$, here m is the number of starting points).

$$\mathrm{SP}_i \overbrace{\underbrace{\xrightarrow{\mathtt{f}_1\cdots\mathtt{f}_1} \mathrm{DP} \cdots \xrightarrow{\mathtt{f}_1\cdots\mathtt{f}_1} \mathrm{DP}}_{v\,\mathrm{DP}s} \underbrace{\xrightarrow{\mathtt{f}_2\cdots\mathtt{f}_2} \mathrm{DP} \cdots \xrightarrow{\mathtt{f}_2\cdots\mathtt{f}_2} \mathrm{DP}}_{v\,\mathrm{DP}s} \cdots \underbrace{\xrightarrow{\mathtt{f}_s\cdots\mathtt{f}_s} \mathrm{DP} \cdots \xrightarrow{\mathtt{f}_s\cdots\mathtt{f}_s} \mathrm{DP}}_{v\,\mathrm{DP}s}}^{r\,times} = \mathrm{EP}_i.$$

Its online attack procedure is similar to the one for unified rainbow table cryptanalysis given in Sect. 3.2, in that we treat the series of functions $\mathtt{f}_j \circ \cdots \mathtt{f}_j$ for generating a distinguished point at here as a single function $\mathtt{f}_j$ in the attack procedure from Sect. 3.2. Suppose q is the expected average length of each sector with the same $\mathtt{f}_j$ variant, (i.e. the chain length is expected to be $t = r \cdot s \cdot v \cdot q$), then we can similarly obtain that such unified rainbow table cryptanalysis with distinguished points has an online time complexity of $\sum_{j=1}^{s}[q + (j-1) \cdot q \cdot v + (r-1) \cdot q \cdot s \cdot v] \cdot v = tsv + \frac{t}{r} - \frac{tsv}{2r} - \frac{tv}{2r}$ computations of the $\mathtt{f}$ function, plus rsv table look-ups.

4.2 TMTO Curve

Below is the TMTO curve of the combination described in Sect. 4.1.

Theorem 3. *Suppose P represents computational complexity for precomputation, T represents online time complexity, M represents memory complexity, plus those parameters v, s, r used for unified rainbow table cryptanalysis with distinguished points. Then, unified rainbow table cryptanalysis with distinguished points approximately meets a time–memory trade-off $T \cdot M^2 = (v - \frac{v}{2r} - \frac{v}{2rs} + \frac{1}{rs}) \cdot N^2$, where $P = N$, $M \leq N$ and $1 \leq r \cdot s \cdot v \leq \frac{N}{M}$, plus $\frac{r \cdot v \cdot N}{M}$ table look-ups.*

5 Time–Memory–Data Trade-Off Curves

In this section, we briefly discuss the time–memory–data trade-off (TMDTO) curves when unified rainbow table cryptanalysis and its combination with distinguished points are applied in certain situations where multiple data can be helpful, for example, cryptanalysis of some stream ciphers. We have the following result.

Theorem 4. *Suppose P represents computational complexity for precomputation, T represents online time complexity, M represents memory complexity, $D(\geq 2)$ represents the number of available data, v, s, r are those parameters used in unified rainbow table cryptanalysis without/with distinguished points. Then, unified rainbow table cryptanalysis approximately meets a time–memory–data trade-off $T \cdot M^2 \cdot D = \frac{N^2}{r} - \frac{N^2}{2r^2} - \frac{M \cdot D^2 \cdot N}{2r^2} + \frac{M \cdot D \cdot N}{r} - M \cdot N$, with $P = \frac{N}{D}$, $M \cdot D^2 \leq N$, $T \geq \frac{D^3}{r} - \frac{D^3}{r^2} + \frac{D^2}{r} - D$ and $1 \leq r \leq D$, plus $\frac{N}{M}$ table look-ups; and unified rainbow table cryptanalysis with distinguished points approximately meets a time–memory–data trade-off $2T \cdot M^2 \cdot D^2 = (2v - \frac{v}{r}) \cdot N^2 - \frac{(v-2) \cdot M \cdot D^2 \cdot N}{2r}$, with $P = \frac{N}{D}$, $M \cdot D^2 \leq N$, $T \geq (v + \frac{1}{r} - \frac{v}{r}) \cdot D^2$ and $1 \leq r \cdot v \leq D$, plus $\frac{r \cdot v \cdot N}{M \cdot D}$ table look-ups.*

At last, a detailed analysis gives the following result:

Corollary 2. *Hellman table, thin rainbow table and unified rainbow table cryptanalyses (can) have a TMDTO curve $T \cdot M^2 \cdot D^2 = \mathcal{O}(N^2)$, while rainbow table and thick rainbow table cryptanalyses have a TMDTO curve $T \cdot M^2 \cdot D = \mathcal{O}(N^2)$. Generally, among all the aforementioned TMDTO methods, fuzzy-rainbow table cryptanalysis is the best TMDTO method, and unified rainbow table cryptanalysis with distinguished points is the second best TMDTO method.*

6 Application to the GSM A5/1 Stream Cipher

In this section, we apply the TMDTO version of unified rainbow table cryptanalysis with distinguishing points to the A5/1 stream cipher, since its design allows us to conduct a TMDTO attack. We first describe the `f` function with respect to A5/1 and the structure of a unified rainbow table, then describe two crucial theoretical techniques and some implementation issues, and finally describe an experimental attack and its results on A5/1.

6.1 The `f` Function with Respect to A5/1

Since our goal is to recover a secret session key, clearly we should choose a suitable `f` function from the A5/1 keystream generator, so that we can have an efficient attack. A detailed analysis on A5/1 reveals the following result:

Proposition 1. *The state (of the three LFSRs) immediately after the key setup of the A5/1 keystream generator is a system of 64 linear functions of the 64 bits of the secret session key. The 64 secret key bits can be easily obtained by solving the system of 64 linear functions of the 64 secret key bits. Given the 64-bit state immediately after the IV setup of the A5/1 keystream generator, we can recover the state immediately before the IV setup (i.e., the state immediately after the key setup) with a time complexity of 22 clockings.*

Finally, after a further investigation we define the `f` function as follows.

Proposition 2. *With respect to A5/1, the* $\mathtt{f}$ *function starts with the initial state (that is immediately after the IV setup), and ends with the first 64 keystream bits. It takes the initial state as input, and outputs a 64-bit sequence. The 64-bit sequence is then input to the next* $\mathtt{f}$ *function as an initial state, and so on.*

6.2 The Structure of a Unified Rainbow Table

We targeted to build a reasonably large unified rainbow table with distinguished points. We set $m = 2^{37}, t = 2^{16}, r = 2^2, s = 2^6, v = 1, q = 2^8$; see Sect. 4.1 for the meaning of the parameters.

The starting points are of 64 bits long; every time we generate a 64-bit random from a particular sub-space as a starting point; and the used sub-spaces guarantee that the generated starting points are distinct one another. A variant $\mathtt{f}_i$ of the $\mathtt{f}$ function is a functional composition of $\mathtt{f}$ and an XOR operation with an integer i, ($i \in [0, s-1]$, i.e., $i = 0, 1, \ldots, 63$), that is, $\mathtt{f}_i = \mathtt{f} \oplus i$; the XOR operation works on the rightmost 6 bits of an output of $\mathtt{f}$, which correspond to positions 1–6 of the first LFSR (i.e. $R1$) of the keystream generator. The distinguished points used here have zeros in eight bit positions 41–48 of a 64-bit value, which correspond to positions 17–19 of $R1$ and positions 1–5 of $R2$. Thus, on average it takes $q = 2^8$ computations to have a distinguished point. By the definitions of the $\mathtt{f}$ function and distinguished points, the endpoints are distinguished points that have zeros in the above-mentioned eight bits, and thus only 56 bits are effective for each endpoint. That is, a starting-endpoint pair takes a memory of 120 bits.

We do not use any other supplementary technique for the table, such as sequential starting points [24], index tables [15], endpoint truncation [15], or checkpoints [6].

6.3 Two Crucial Theoretical Techniques

We used two crucial theoretical techniques in our online attack on A5/1; the first one aims to reduce an attack's data complexity by using the sliding property [19] of the A5/1 keystream generator, and the second one aims to increase an attack's success probability by using the state convergence property [19,30] of the A5/1 keystream generator. In other words, the two techniques allow us to reach a certain success probability with much smaller precomputational workload. Since the first technique has been described and used in previous work like [19], below we only describe the second technique, which can be similarly applied to any stream cipher with a similar property. Note that Dj. Golić [19] also mentioned the state convergence property of A5/1, however, he used the property in a direct way — simply reducing the concerned searching space from the ideal 2^{64} to the convergent $2^{63.32}$ (upper bound), which (is not sufficient and should be followed by) is quite different from the technique we describe below. It seems that Nohl [30] used the state convergence property in some way comparable to the second technique described below, but he did not describe or even mention it (maybe because of his company's need for confidentiality), thus we are not sure

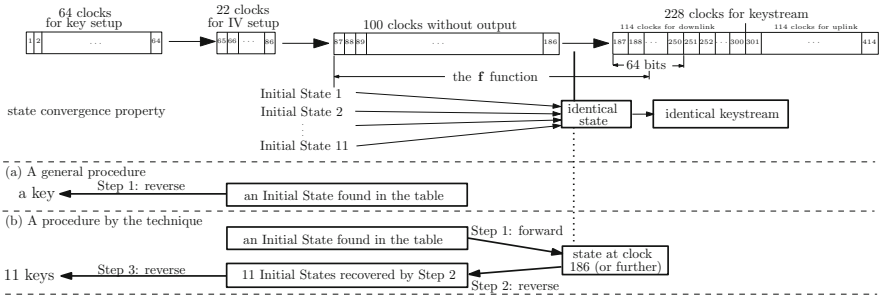

Fig. 2. The second technique compared with a general procedure

whether the second technique was used in [30]. To the best of our knowledge, the straightforward way that Dj. Golić used is the popular way to use the state convergence property, and none has explicitly described the second technique before.

The second technique relies on the fact that usually a few initial states of A5/1 will converge to the same internal state after the 100 clockings without output, known as the state convergence problem of A5/1 [19]. A theoretical analysis from [19] reveals that there are only $2^{63.32}$ possible states after the first clocking, and experimental results [15,30] show that only about 15 % of the 2^{64} states is possible after the 100 clockings without output; and our experiments show that on average about $2^{3.5} \approx 11$ initial states can be obtained from a state just after the 100 clockings without output. Below we describe an approach to make use of this state convergence property, which is illustrated in Fig. 2.

Suppose $y = \mathtt{f}(x)$ is our inversion target. Let's consider the case with the first keystream segment, that is a preimage is also an initial state. When we find a preimage x^* to y from our (unified) rainbow table, we compute the internal state immediately after 100 (or 164, that is immediately after generating the first 64-bit keystream segment) clockings from x^*, and then we reverse this internal state at Clock 100 (respectively 164) to obtain all possible initial state(s) that can produce this internal state at Clock 100 (respectively 164), and finally we recover the corresponding session key for every obtained initial state and check whether one of them is the correct session key. (By contrast, a general procedure after finding a preimage x^* to y is to reverse x^* through the key and IV setups to check whether it could lead to the correct session key)

As mentioned earlier, on average about 11 initial states can be obtained from a state immediately after the 100 clockings without output; for some initial states, there are a large number of other initial states that converge to the same internal state at Clock 100, while for others there are a small number of other initial states that converge to the same internal state at Clock 100. For session keys such that there are a large number of other initial states that converge to the same internal state after 100 (or 164) clocks, there is a larger success rate to find it, because there is a larger probability that one of the many initial states

converging to the same internal state is covered in the (unified) rainbow table, and as long as one of the convergent initial states is found with our rainbow table, then we can definitely recover the correct initial state and then find the correct session key; for session keys such that there are a small number of other initial states that converge to the same internal state after 100 (or 164) clocks, there is a smaller success rate to find it, for there is a smaller probability to cover one of the few convergent initial states in our table. This explains the existence of preferred session keys and non-preferred session keys for our attack.

The second technique also indicates why we did not define the f function to be from the state after the 100 clocks (without output) to the first 64-bit keystream segment. In this case, roughly 85 % of the points covered in a rainbow table would not have a corresponding initial state and thus is wasteful, and we cannot use the above technique to increase success probability.

The approach holds similarly for the cases with other keystream segments, where a preimage x^* to y obtained from our rainbow table is not an initial state, and we first compute forward to get the internal state immediately after 100 or 164 clockings, and then reverse this internal state to obtain all possible initial states that converge to the same internal state at Clock 100 or 164.

Note that we can compute the internal state after more than 100 (or 164) clockings from x^*, and then we reverse this state to obtain all the possible initial state(s). This could increase success probability slightly further, although our experiment shows that the gain is not much.

Thus, given a (unified) rainbow table, the second technique enables us to achieve a success probability of about 11 times as much as that for a naive attack (at the expense of negligible extra work). In other words, it enables us to reduce the precomputational workload by about 90 % to have a certain success probability. Notice that this technique is also owing partially to the fact that it is easy to reverse an internal state to the initial state position.

6.4 Implementation Issues

Below we briefly describe some implementation issues associated with our attack.

6.4.1 Implementations on A5/1

An efficient implementation of A5/1 is essential to our work, for it could save a lot of time during both precomputational and online phases. We have implemented three versions of A5/1 in C language:

1. A basic version that generates a keystream bit by bit under a session key;
2. A bit-slicing version that generates 64 keystreams in parallel under 64 different session keys; and
3. A multiple-bit version that generates a keystream block by block under a session key, by using several small precomputation tables, here a block consists of i bits generated simultaneously ($i \in [2, 8]$). (We did not consider $i > 8$, for the corresponding precomputation tables would take a large memory)

We have checked the performance of the three versions on a HP Z600 workstation with Intel Xeon Processor E5630 (2.53 GHz, 12 MB cache) and Ubuntu 12.04 operating system under the following three cases:

* Case A — the session key is fixed and, each time a large number of keystreams with only the first 64 bits are generated (10^6 in our test).
* Case B — each time the session key is updated and only the first 64 keystream bits are generated.
* Case C — only the part of the `f` function is considered.

For the three cases, the basic version has a throughput of approximately 202, 636 and 406 cycles/byte, respectively; the bit-slicing version has a throughput of approximately 29, 130 and 57 cycles/byte, respectively; and among the multiple-bit versions the four-bit version has a best throughput of approximately 89, 195 and 154 cycles/byte, respectively. We use the bit-slicing version (with further optimisation for using distinguished points) during offline precomputation phase, and use the four-bit version during online attack phase.

6.4.2 Computing a (Unified) Rainbow Table with GPGPU

We used the TMDTO version of unified rainbow table cryptanalysis with distinguishing points in our attack, and used a GPGPU workstation in both the offline precomputation and online attack phases.

The workstation used for our attack consists of a host system based on one dual XEON CPU (2 CPUs of Intel Xeon Processor E5-2620, 2 GHz, 15 MB cache and 32 GB ECC RAM each), one Quadro 600 GPU for display, 3 NVIDIA GeForce GTX690 graphics cards (which are actually 6 GTX680 cards roughly) for parallel computation, and 10 solid state disks (SSDs) of 480G SATA3 each. The total price was about 15,000 United States dollars. A GTX690 contains 8 graphic processing clusters (GPCs) and 16 streaming multiprocessors (SMXs); a streaming multiprocessor contains 192 streaming processors (SPs). In total, a GTX690 has 3072 cores with processor clock of 915MHz, and the on-chip memory is 2×2 GB GDDR5 with 2×256 bit width.

In the offline precomputation phase, the startpoints were processed by the 6 GTX680s in parallel; and 6 CPU threads were needed to control the 6 GPUs, respectively. The startpoints and endpoints were all located in host memory, and all of them can be accessed and modified by GPUs via direct memory access (DMA). Startpoints were randomly generated with no collisions, which were made by dividing the whole 2^{64} space into many small sections of equal length and then selecting a random number in each small section, (thus avoiding collisions). An additional thread was used to store the starting-endpoint pairs into a table. Whenever an endpoint was generated by a GPU, the additional thread put it into the input/output (I/O) buffer, and a fast I/O was achieved by writing out the buffer at one time. The additional thread was in parallel with the 6 threads controlling the 6 GPUs.

In the online attack phase, only one GPU (i.e. GTX680) was used, (which makes the attack feasible on laptops). Distinguishing points were computed on

the GPUs, in a procedure similar to the offline phase; and the only difference was that the intermediate distinguishing points before reaching the endpoint were also recorded. We look up the 256 distinguishing points generated typically in an online chain by the optimized binary search, which took less than 1 s normally. Since GPU can modify the host memory via DMA, the host can check the generated distinguishing points in real time before the whole task on computing distinguishing points in GPU is finished. It is advisable to use an additional thread to check the calculated distinguishing point and perform table lookup of these distinguishing points. Therefore, table lookups are actually parallel with online computations on distinguishing points. Since a table lookup takes a shorter time than a computation on distinguishing points, virtually it does not need any time. The total online attack time is approximately the sum of the online computation time of distinguishing points and the online regeneration time of false alarms.

6.4.3 Configuration on SSDs

We used SSDs instead of general hard disks (HDDs), because SSD access is faster than CPU RAM access. Redundant Array of Independent Disks (RAID) is a data storage virtualization technology that combines multiple disk drive components into a logical unit for the purposes of data redundancy and performance improvement. The most common RAID configurations are RAID 0 (striping), RAID 1 and variants (mirroring), RAID 5 (distributed parity) and RAID 6 (dual parity). We have tested and compared RAID0 and RAID5. The stripe size is defaulted to be 128 KB, and the 10 SSDs of about 480 GB each used in our attack are viewed as a single virtual huge disk to users by being connected to an Intel SSD controller RS2WG160 which can control up to 16 SSDs. A RAID 0 splits data evenly across two or more disks (striped) without parity information for speed. RAID 0 is not resistant to any error or failure in disk, since it does not provide data redundancy. In most situations RAID 0 yielded the highest read and write performance, and its read and write speeds were tested to be approximately 3.2 GB/s and 1.8 GB/s, respectively.

6.4.4 Table Sorting for Fast Lookup

A unified rainbow table is sorted by the endpoints. In practice, there is a constraint on the maximum size of a file, so a dramatically large table has to be divided into a number of reasonably large parts to store. In our attack, we divide a unified rainbow table into 512 parts by dividing the space of the endpoints into 512 intervals equally, with each part of the table corresponding to a unique interval. During offline precomputation phase, after generating a starting-endpoint pair we simply store it into the part corresponding to the interval that the endpoint locates in; and during online attack phase, we can readily identify which part of the table the endpoint of an online chain is from, without searching it over all the parts of the table; then, binary search is performed in the identified part of the table.

6.4.5 Software Speedup of Table Access

A straightforward binary search (i.e. lookup) on an endpoint in the unified rainbow table with the 512 parts described in Sect. 6.4.4 takes 0.0073 s on SSD on average, (and 0.1400 s on HDD, thus the speedup is $\frac{0.1400}{0.0073} \approx 19$ times). Generally, given an A5/1 keystream in our attack, for an online chain we need to compute $r \times s \times v = 256$ distinguishing points and thus need to search 256 times in the unified rainbow table. Thus, the table lookups on all the 51 segments of a 114-bit keystream generally take $0.0073 \times 256 \times 51 \approx 96$ s on SSD (and $0.1400 \times 256 \times 51 \approx 1828$ s on HDD), regardless of the online computational time on `f` and its variants. We optimised the straightforward binary search with several software techniques on disk access: (1) Cached binary search with multithreading, which is $\frac{96}{31.1} \approx 3.09$ times of the standard binary search; (2) Endpoint indexing, which achieves a speedup of $\frac{31.1}{4.0} \approx 7.78$ times on the cached binary search; and (3) Multithreading with thread pool, which achieves a speedup of $\frac{4.0}{0.43} \approx 9.31$ times on the indexed and cached binary search. Taking all the three techniques into consideration, the final speedup is $3.09 \times 7.78 \times 9.31 \approx 223$ times compared to the straightforward binary search, that is, the table lookup time on all the 51 segments of a 114-bit keystream is reduced from the original 96 s to $\frac{96}{223} \approx 0.43$ s finally for our unified rainbow table. Our table lookup speed (on our SSD) is now $\frac{256 \times 51}{0.43} \approx 30,363$ searches per second, which is higher than the table lookup speed of 20,000 searches per second (i.e. 100,000 searches in 5 s) reported in Nohl's work.

6.4.6 Design Criteria on Compute Unified Device Architecture (CUDA)

NVIDIA's Compute Unified Device Architecture (CUDA) [28] is a programming model that integrates host code and GPU code in the same C/C++ source files. CUDA provides convenient lightweight programming abstractions of the actual parallelism implemented by the hardware architecture. Our attack used several key criteria introduced in [2,23,29] for optimising GPU, namely thread packing, arithmetic intensity and resource usage, warp divergence, the communication between host and device, the on-chip memory for storing lookup tables, and GPU error rate estimation.

6.5 Experimental Results

With the `f` function and parameters defined in Sect. 6.1, we first generated a unified rainbow table of about 1.6 TB on the workstation under the computing framework described in Sect. 6.4, which took about 54 days and involved about $2^{52.68}$ computations of the `f` function. The table became 984 GB (≈ 0.96 TB) after we sorted and removed starting-endpoint pairs with already existing endpoints, which took less than one day. That is, it took a total of about 55 days to make the 984 GB table on our GPGPU. The 984 GB table covered a total of about $2^{51.94}$ initial states (with possible duplication).

Then, we implemented and optimised a unified rainbow table attack based on the 984 GB table and downlink keystreams (of 114 bits long each). Based on four thousand tests, on average the attack had an online attack time of 9 s (of which: 5 s for computing online chains with one GPU, in parallel with table look-ups; and 4 s on false alarms with this GPU) with a success probability of 34 % when using 4 keystreams (204 segments); or an online attack time of 9 s with a success probability of 56 % when using 8 keystreams (408 segments), where two GPUs were used in parallel (i.e. one GTX690), and each of the two GPUs dealt with 4 keystreams for online chains (in parallel with table lookups) with 5 s and spent 4 s on false alarms.

At last, we targeted to generate three more tables of 984 GB with three different sets of the `f` variants, but at the moment the three other precomputation tables only had a size of 0.140 TB, 0.093 TB and 0.093 TB, respectively, (after sorting and removing degenerate starting-endpoint pairs). Using the four tables with a total size of $0.96+0.140+0.093+0.093 \approx 1.29$ TB, we got an experimental attack that used 4 keystreams (204 segments) and had an online attack time of 9 s (of which: 5 s for computing online chains with one GPU, in parallel with table look-ups of about 2 s; and 4 s on false alarms) with a success probability of 43 %. The attacks indicate that unified rainbow table cryptanalysis has a good performance as well.

Note that more cryptanalytic results can be extrapolated from the above experimental results, for example: If one more table of 984 GB was generated (i.e., a total of 1.92 TB), we could expect an attack that had an online attack time of $5+4 = 9$ s with a success probability $1-(1-34\,\%)^2 \approx 56\,\%$ when using 4 keystreams, where two of the six GPUs were used in parallel (i.e. one GTX690), and each of the two GPUs dealt with a table for online chains (in parallel with table lookups) and false alarms; or an online attack time of $5 + 4 = 9$ s with a success probability $1 - (1 - 56\,\%)^2 \approx 81\,\%$ when using 8 keystreams, where two GTX690s (i.e. four GPUs) were used in parallel, and each of the two GTX690s dealt with a table for online chains (in parallel with table lookups) and false alarms.

7 Conclusions

In this paper, we have presented a unified TMTO cryptanalysis, called unified rainbow table cryptanalysis, have described its general combination with distinguished points, and have discussed their TMTO as well as TMDTO curves under the worst scenario. Finally, we applied unified rainbow table cryptanalysis to the A5/1 stream cipher, by working out a crucial technique that made the precomputational workload feasible to have an acceptable success probability. We made a unified rainbow table of 984 GB in about 55 days on a GPGPU computer with 3 NVIDIA GeForce GTX690 cards at a total cost of about 15,000 United States dollars, and implemented a unified rainbow table attack on A5/1, that had an online attack time of 9 s with a success probability of 34 % (or 56 %) when using 4 (respectively 8) keystreams.

Unified rainbow table cryptanalysis is of theoretical significance as a unified TMTO cryptanalysis framework, although it offers a trivially more comprehensive TMTO curve than the previously published four general TMTO methods and can offer the second best TMDTO curve when combined with distinguished points. From its success probability formula, we can have success probability formulas for rainbow table, thin rainbow table and thick rainbow table cryptanalyses, which take into account some possible redundancy among different columns and the distinctions among different function variants. The presented unified rainbow table attack on A5/1 is the first rainbow-type TMTO attack on A5/1 that reveals crucial techniques and implementation details, and is of practical significance, for the practical experimental results show again that A5/1 is rather insecure in reality.

Acknowledgments. This work resulted from an industry project on rainbow table cryptanalysis of the A5/1 stream cipher. The authors are very grateful to Wun-She Yap and Chee Hoo Yian for their participation in the early stage of the project, in particular, in the implementation of the A5/1 stream cipher.

References

1. http://en.wikipedia.org/wiki/A5/1
2. http://www.cs.virginia.edu/~mwb7w/cuda_support/optimization_techniques.html
3. Amirazizi, H.R., Hellman, M.E.: Time-memory-processor trade-offs. IEEE Trans. Inf. Theory **34**(3), 505–512 (1988)
4. Anderson, R.: A5, Newgroup Communication (1994)
5. Anderson, R.: On Fibonacci keystream generators. In: Preneel, B. (ed.) FSE 1994. LNCS, vol. 1008, pp. 346–352. Springer, Heidelberg (1995)
6. Avoine, G., Junod, P., Oechslin, P.: Characterization and improvement of time-memory trade-off based on perfect tables. ACM Trans. Inf. Syst. Secur. **11**(4), 17:1–17:22 (2008)
7. Barkan, E.: Cryptanalysis of ciphers and protocols. Ph.D. thesis, Technion – Israel Institute of Technology, Israel (2006)
8. Barkan, E., Biham, E.: Conditional estimators: an effective attack on A5/1. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 1–19. Springer, Heidelberg (2006)
9. Barkan, E., Biham, E., Keller, N.: Instant ciphertext-only cryptanalysis of GSM encrypted communication. J. Cryptology **21**(3), 392–429 (2008)
10. Barkan, E., Biham, E., Shamir, A.: Rigorous bounds on cryptanalytic time/memory tradeoffs. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 1–21. Springer, Heidelberg (2006)
11. Biham, E.: How to decrypt or even substitute DES-encrypted messages in 2^{28} steps. Inf. Process. Lett. **84**(3), 117–124 (2002)
12. Biham, E., Dunkelman, O.: Cryptanalysis of the A5/1 GSM stream cipher. In: Roy, B., Okamoto, E. (eds.) INDOCRYPT 2000. LNCS, vol. 1977, pp. 43–51. Springer, Heidelberg (2000)

13. Biryukov, A., Mukhopadhyay, S., Sarkar, P.: Improved time-memory trade-offs with multiple data. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 110–127. Springer, Heidelberg (2006)
14. Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000)
15. Biryukov, A., Shamir, A., Wagner, D.: Real time cryptanalysis of A5/1 on a PC. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 1–18. Springer, Heidelberg (2001)
16. Briceno, M., Goldberg, I., Wagner, D.: A pedagogical implementation of the GSM A5/1 (1999)
17. De, A., Trevisan, L., Tulsiani, M.: Time space tradeoffs for attacks against one-way functions and PRGs. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 649–665. Springer, Heidelberg (2010)
18. Denning, D.E.: Cryptography and Data Security. Addison-Wesley, Boston (1982)
19. Golić, J.D.: Cryptanalysis of alleged A5 stream cipher. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 239–255. Springer, Heidelberg (1997)
20. Ekdahl, P., Johansson, T.: Another attack on A5/1. IEEE Trans. Inf. Theory **49**(1), 284–289 (2003)
21. Fiat, A., Naor, M.: Rigorous time/space trade-offs for inverting functions. SIAM J. Comput. **29**(3), 790–803 (1999)
22. Gendrullis, T., Novotný, M., Rupp, A.: A real-world attack breaking A5/1 within hours. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 266–282. Springer, Heidelberg (2008)
23. Harris, M.: Optimizing cuda. SC07: High Performance Computing With CUDA (2007)
24. Hellman, M.E.: A cryptanalytic time-memory trade-off. IEEE Trans. Inf. Theory **26**(4), 401–406 (1980)
25. Hong, J.: The cost of false alarms in Hellman and rainbow tradeoffs. Des. Codes Crypt. **57**(3), 293–327 (2010)
26. Hong, J., Moon, S.: A comparison of cryptanalytic tradeoff algorithms. J. Crypt. **26**(4), 559–637 (2013)
27. Maximov, A., Johansson, T., Babbage, S.: An improved correlation attack on A5/1. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 1–18. Springer, Heidelberg (2004)
28. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with cuda. Queue **6**(2), 40–53 (2008)
29. Nvidia, C.: Compute unified device architecture programming guide (2007)
30. Nohl, K.: Attacking phone privacy. In: Black Hat USA 2010 Lecture Notes (2010). https://srlabs.de/decrypting-gsm/
31. Oechslin, P.: Making a faster cryptanalytic time-memory trade-off. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 617–630. Springer, Heidelberg (2003)
32. Standaert, F.X., Rouvroy, G., Quisquater, J.J., Legat, J.D.: A time-memory tradeoff using distinguished points: new analysis & FPGA results. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 593–609. Springer, Heidelberg (2003)