Cryptanalysis of GSM Encryption in 2G/3G Networks without Rainbow Tables

Bin Zhang[1,2,3,4]

[1]TCA, SKLCS, Institute of Software, Chinese Academy of Sciences
[2] State Key Laboratory of Cryptology, P.O.Box 5159, Beijing, 100878, China
[3] University of Chinese Academy of Sciences, Beijing, 100049, China
[4] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences
martin_zhangbin@hotmail.com

Abstract. The GSM standard developed by ETSI for 2G networks adopts the A5/1 stream cipher to protect the over-the-air privacy in cell phone and has become the de-facto global standard in mobile communications, though the emerging of subsequent 3G/4G standards. There are many cryptanalytic results available so far and the most notable ones share the need of a heavy pre-computation with large rainbow tables or distributed cracking network. In this paper, we present a fast near collision attack on GSM encryption in 2G/3G networks, which is completely new and more threatening compared to the previous best results. We adapt the fast near collision attack proposed at Eurocrypt 2018 with the concrete irregular clocking manner in A5/1 to have a state recovery attack with a low complexity. It is shown that if the first 64 bits of one keystream frame are available, the secret key of A5/1 can be reliably found in $2^{31.79}$ cipher ticks, given around 1 MB memory and after the pre-computation of $2^{20.26}$ cipher ticks. Our current implementation clearly certified the validity of the suggested attack. Due to the fact that A5/3 and GPRS share the same key with A5/1, this can be converted into attacks against any GSM network eventually.

Key words: Cryptanalysis, GSM, A5/1, Near collision.

1 Introduction

The GSM standard, developed by ETSI for 2G networks used by mobile phones, specifies the A5/1 stream cipher to protect the over-the-air privacy all over the world. As of today, A5/1 has become the de-facto global standard for mobile communications with more than 4 billion customers and over 90% market share, operating in over 219 countries and territories.

A5/1 is the strong version of the encryption algorithm in GSM standard and A5/2 is the weak version free of export limitations. A5/1 was designed in the 80's of last century as a typical LFSR-based stream cipher with an irregular clocking mechanism. The exact design was reverse-engineered in [6] in 1999 and confirmed by the relevant authorities subsequently. A newly standardized version in GSM networks is A5/3, which is based on the block cipher KASUMI. In practice, the algorithm is frequently re-synchronized and a GSM conversation consists of a series of frames sent every 4.6 milliseconds. From a 64-bit secret key and a 22-bit publicly known frame counter, only 228 bits keystream are generated in each frame, and after that a new frame counter is mixed with the same key again in initialization to generate another frame.

There are lots of attacks published so far against A5/1 and the GSM encryption, to name but a few [1–5, 7–9, 12–14]. Different cryptanalytic strategies, e.g., time/memory/data (TMD) tradeoff attacks, guess-and-determine attacks and (conditional) correlation attacks have been tried, resulting in more and more powerful attacks in terms of complexity cost. But from both the theoretical and practical viewpoints, only a few results [2, 13] have given new theoretical insights and have the direct and important consequence on the GSM encryption itself. At Crypto 2003, a practical ciphertext-only attack on A5/2 was depicted in [2], which could be extended to more complex and more expensive attacks on A5/1, of which the cheapest pre-computation required 35 PCs (that were available around 2003) to work a few years with about 600 GB disks. Given the protocol flaws of the GSM networks and with the assumption that the targeted mobile phone supports A5/2, various active attacks were launched accordingly. In January 2007, the Hacker's Choice started the A5/1 cracking project with plans to use FPGAs that allow A5/1 to be broken with a rainbow table attack. Then in 2010, K. Nohl announced a new attack in [13] without the reveal of many details, using a thick rainbow table with the distinguished points technique. It had an online attack time of about 10 seconds on a general-purpose GPU with the success probability of 87%, given 8 known keystream frames and 30 pre-computation tables of about 1.7 Terabytes. The most recent attack on A5/1 was presented in [11], where a unified rainbow table cryptanalytic method was introduced and applied to A5/1. To have a comparable success probability and online attack time to the relevant attacks, two pre-computation tables of 984 GB were needed, each has to be prepared in $55 \cdot 2 = 110$ days.

In this paper, we take an entirely different cryptanalytic approach to break A5/1 used in the GSM networks, free of the extremely heavy rainbow tables and long time pre-computation. While all the previous TMD tradeoff attacks regard the internal state of A5/1 as a whole and try to restore it in one shot immediately, we adopt the fast near collision attack (FNCA) strategy in [16] and make a divide-and-conquer partition of the full internal state. Precisely, we regard the internal state as the union of the crucial part (CP) and the rest part (RP), which could be retrieved easily given the corresponding CP and the keystream prefix. Thus, the security of the primitive mainly depends on the intractability of restoring the CP and the efficiency of restoring the RP accordingly. We first launch a fast near collision attack to recover the CP part of the internal state in A5/1 according to the irregular clocking mechanism, based on which the RP part could be recovered later by a dynamic guess-and-determine attack similar to [9]. It is surprising that the irregular clocking mechanism in A5/1 actually facilitates the list merging procedure in a fast near collision attack. Due to the event that not all the three registers moves simultaneously happens with a probability of 0.75, two more overlapping bits are gained for free for each such step. Further in A5/1, it is found that the parameter configuration in a fast near collision attack can be easily tuned to have the desirable non-random behaviour between the resultant list size and a good existence probability of each correct restricted internal state in FNCA. As a result of these findings, it is shown that if the first 64 bits of one keystream frame are intercepted, the internal state, thus the secret key of A5/1 can be reliably found in $2^{31.79}$ cipher ticks, given around 1 MB memory and after a pre-computation of $2^{20.26}$ cipher ticks. Our current implementation in C language on a single core of a PC clearly certified the correctness of this new attack. It takes tens of seconds on average to find the targeted internal state of A5/1, and we feel that further optimization of the code will reduce the time to several seconds. This is the best known attack on A5/1 so far and it is worthy noting that due to the fact that A5/3 and GPRS share the same key with A5/1 in GSM, our attack can be leveraged into attacks against any GSM 2G/3G network eventually.

This paper is organized as follows. A brief description of the A5/1 stream cipher and the GSM encryption scheme is presented in Section 2. Some basic definitions and preliminaries of the fast near collision attack relevant to our analysis are provided in Section 3. Then a high-level description of our attack against A5/1 and the technical details are presented in Section 4, followed by the experimental results in Section 5. We leverage our attack to any GSM network in Section 6 and finally, some conclusions are drawn in Section 7.

2 Description of A5/1 and the GSM Encryption

Let us first present the algorithmic details of the A5/1 stream cipher and the GSM encryption scheme that are relevant to our analysis. A5/1 consists of 3

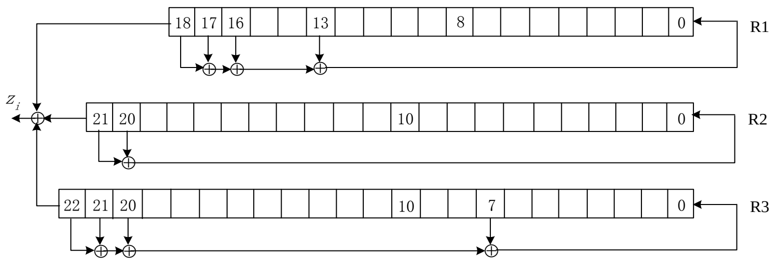

Fig. 1. The A5/1 stream cipher.

short linear feedback shift registers (LFSR), denoted by R1, R2 and R3, which are of length 19, 22 and 23 bits respectively. Each register has a primitive feedback polynomial and thus generates a maximum-length binary sequence. The rightmost bit in each register is labelled as bit 0. The taps of R1 are at the 13th, 16th, 17th and 18th bit positions; the taps of R2 are at the 20th, 21st bits and the taps of R3 are at the 7th, 20th, 21st and 22nd bits, as depicted in Figure 1.

Besides, each register has another clocking tap, i.e., bit 8 for R1, bit 10 for R2 and bit 10 for for R3, to feed a majority function defined as $maj(ct_1, ct_2, ct_3) = ct_1 \cdot ct_2 \oplus ct_2 \cdot ct_3 \oplus ct_3 \cdot ct_1$, where $ct_1 =$R1[8], $ct_2 =$R2[10] and $ct_3 =$R3[10]. The three registers are clocked in a stop/go fashion using the majority rule: at each clock, take the majority function of the clocking taps and only run those registers whose clocking taps agree with the computed majority bit. It is easy to see that at each step, either two or three registers are clocked, and that each register moves with a probability of 3/4 and stops with a probability of 1/4.

In the initialization phase, the three registers are all first set to zero. The secret session key K and a publicly known frame number FN are first injected and then mixed, after that 228 keystream bits are generated in each frame.

One Session of GSM Encryption

Input Parameters:

```
1: $K = (K_{63}, \cdots, K_1, K_0)$ is the 64-bit secret key
2: $FN = (FN_{21}, \cdots, FN_0)$ is a 22-bit frame counter
3: for $i = 0$ to 63 do
4:    regularly clock R1, R2 and R3
5:    R1[0] =R1[0] $\oplus K_i$
6:    R2[0] =R2[0] $\oplus K_i$
7:    R3[0] =R3[0] $\oplus K_i$
8: for $i = 0$ to 21 do
9:    regularly clock R1, R2 and R3
10:    R1[0] =R1[0] $\oplus FN_i$
11:    R2[0] =R2[0] $\oplus FN_i$
12:    R3[0] =R3[0] $\oplus FN_i$
13: for $i = 0$ to 99 do
14:    clock R1, R2 and R3 in the specified stop/go fashion
       without producing any output
15: for $i = 0$ to 227 do
16:    clock R1, R2 and R3 in the specified stop/go fashion
17:    generate $z_i =$R1[18] $\oplus$ R2[21] $\oplus$ R3[22]
```

Output: the keystream segment $\{z_i\}_{i=0}^{227}$

In the initialization phase (Step 3 to Step 14), the content of the three registers after step 12 is called the initial state, which is denoted by $S(0)$. The internal state before Step 15 is denoted by $S(100)$. In one session of a GSM conversation, one 114-bit keystream segment is generated for one direction communication and the other 114-bit segment for the opposite direction. Thus, only 228 keystream bits are available in each frame.

3 Preliminaries

In this section, some notations and basic definitions relevant to the fast near collision attack and our work are presented.

We start with the notions of keystream prefix, keystream segment difference (KSD) and internal state difference (ISD).

Definition 1. *For a specified cipher such as A5/1, a keystream prefix* $\mathbf{z} = (z_0, z_1, \cdots, z_{l-1})$ *of* l*-bit length is the keystream vector generated consecutively and directly from the corresponding internal state* $\mathbf{x}$.

Definition 2. *For a specified cipher such as A5/1 and two keystream prefixes* $\mathbf{z}$ *and* $\mathbf{z}'$*, if* $\mathbf{z}$ *is generated from the internal state* $\mathbf{x}$ *and* $\mathbf{z}'$ *is generated from the internal state* $\mathbf{x}'$*, the keystream segment difference (KSD) is* $\Delta\mathbf{z} = \mathbf{z} \oplus \mathbf{z}'$ *and the internal difference (ISD) is defined as* $\Delta\mathbf{x} = \mathbf{x} \oplus \mathbf{x}'$.

In a fast near collision attack, the full internal state which usually contains a large number of variables is not targeted directly; instead the adversary only considers the subset of the full internal state which is directly associated with a specified keystream prefix, as shown in the following definition of restricted internal state.

Definition 3. *For a specified cipher such as A5/1, the subset* $\mathbf{x} = (x_{i_0}, x_{i_1}, \cdots, x_{i_{n-1}})$ *of the full internal state which is directly associated with the keystream prefix* $\mathbf{z} = (z_0, z_1, \cdots, z_{l-1})$ *is called the restricted internal state of* $\mathbf{z}$.

Note that Definition 3 has a subtle difference with the corresponding Definition 1 in [16]. Here we only consider the keystream prefix which is generated consecutively from a given internal state, while Definition 1 in [16] covers the general cases that the keystream bits under consideration are non-consecutive, which is called the keystream vector.

We also need the notions of two kinds of sampling resistance of the primitive, which characterizes the enumeration procedure in a fast near collision attack.

Definition 4. *For a specified cipher such as A5/1, if there exists some efficient method to enumerate directly some subset of the full internal state which produces a special keystream prefix of* l *bits, e.g., a string of* 0*s, without trying and discarding the other states, then it has a BSW sampling resistance of* l *bits, or equivalently its BSW sampling resistance is* 2^{-l}.

It was shown in [5] that when targeting the full 64-bit internal state, A5/1 has a BSW sampling resisitance of 2^{-16} due to the poor choice of the clocking taps, which makes the register bits that affect the clock control and those affecting the keystream bits unrelated for about 16 clock cycles, so the adversary can independently choose them. This technique can be transformed as follows when only considering the subset of the full internal state.

Definition 5. *For a specified cipher such as A5/1, let* $\mathbf{z} = (z_0, z_1, \cdots, z_{l-1})$ *be the keystream prefix whose restricted internal state is* $\mathbf{x} = (x_{i_0}, x_{i_1}, \cdots, x_{i_{n-1}})$*, if* l *bits in* $\mathbf{x}$ *could be derived explicitly by* $\mathbf{z}$ *and the other bits in* $\mathbf{x}$*, then* l *is the restricted BSW sampling resistance corresponding to* $(\mathbf{x}, \mathbf{z})$.

Definition 4 is generalized in [16] to deal with the restricted internal states, as described in Definition 5. Now we only use very short keystream prefix once a time, e.g., $l = 2$ bits in a fast near collision attack, the time complexity to fulfill the enumeration step becomes negligible compared to the other procedures of the attack.

Let $\varnothing$ be the empty set, the partition of the full internal state into the CP and RP parts is formally introduced in the following definition.

Definition 6. *For a specified cipher such as A5/1, the subset* $\mathbf{x}^*$ *of the full internal state* $\mathbf{x_{full}}$ *that is crucial for the security of the primitive under our framework is the CP part, the rest of the internal state* $\bar{\mathbf{x}^*}$*, where* $\mathbf{x}^* \cap \bar{\mathbf{x}^*} = \varnothing$ *and* $\mathbf{x}^* \cup \bar{\mathbf{x}^*} = \mathbf{x_{full}}$ *is the RP part.*

Note that it is the freedom of the adversary to determine how to choose the CP part of the internal state. Usually, the CP part is selected in such a way that once retrieved, the rest internal state named RP could be relatively easy to restore, provided the corresponding keystream segment. The partition of CP and RP may be non-unique in a fast near collision attack, we do not exclude other possible choices.

Now we come to the theoretical foundation of a fast near collision attack. Let GF(2) be the binary field and its n-dimensional vector space is denoted by GF$(2)^n$. First comes the definition of d-near-collision in [15, 16].

Definition 7. *Two bit strings* $s_1 \in GF(2)^n$ *and* $s_2 \in GF(2)^n$ *are d-near-collision, if* $w_H(s_1 \oplus s_2) \leq d$*, where* $w_H(\cdot)$ *is the Hamming weight of the input.*

The basic near collision lemma is as follows, as stated in [16].

Lemma 1. *Let A and B be two random subsets of* $GF(2)^n$ *and D is a condition set, then there exist a pair* $(a, b) \in A \times B$ *satisfying one of the conditions in D if*

$$|A| \cdot |B| \geq c \cdot \frac{2^n}{|D|}$$

holds, where $|A|$*,* $|B|$ *and* $|D|$ *are the cardinalities of sets A, B and D respectively; c is the constant that determines the actual existence probability of the good pair* (a, b)*. In particular, if* $D = \{\Delta x \in GF(2)^n | w_H(\Delta x) \leq d\}$*, then* $|D| = v(n, d) = \sum_{i=0}^{d} \binom{n}{i}$ *is the total number of ISDs with* $w_H(\Delta x) \leq d$ *and* $(a, b) \in A \times B$ *is a d-near-collision pair.*

Lemma 1 has a very large connotation in the sense that it does not restrict what kind of condition is defined in D and has nothing to do with any secret information. The only premise is the randomness of the two involved sets. As in [16], we have the following statements as the corollary of Lemma 1.

Corollary 1. *For a specified cipher such as A5/1 and a constant c, if we choose* $|A| = 1$ *and* $|B| = c \cdot \frac{2^n}{|D|}$ *with A and B being the n-bit restricted internal states associated with a l-bit keystream prefix, and* $D = \{\Delta x \in GF(2)^n | w_H(\Delta x) \leq d\}$*, then there exists an element* $b_i \in B$ *such that the pair* (a, b_i) *with the unique element* $a \in A$ *forms a d-near collision pair with a probability dependent on c.*

Corollary 1 implies that we can carefully choose the constant c and the parameter d to control the existence probability of the desirably good near collision pair.

3.1 The Fast Near Collision Attack

At Eurocrypt 2018, a new cryptanalytic method called fast near collision attack was introduced in [16] to analyze modern stream ciphers with a large internal s-tate. The idea is to combine a near collision property with the divide-and-conquer strategy to restore several partial internal states first, merge the recovered partial states together according to the concrete internal structure and finally recover the full large internal state based on the merged part.

While the original near collision attack in [15] tried to collect two keystream vector sets and to identify a near collision in the whole internal state at different time instants, a fast near collision attack only targets several well-chosen partial restricted internal states based on the refined self-contained method first, whose idea is depicted in Figure.2. The observation here is that the adversary could

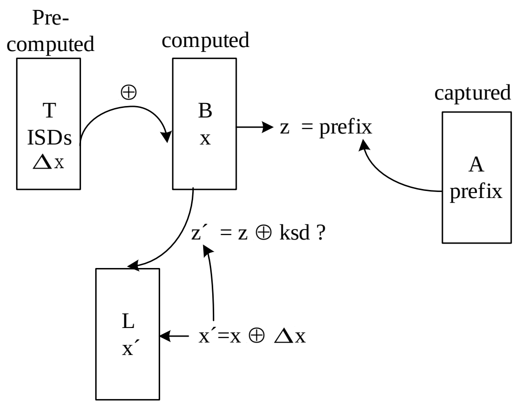

Fig. 2. The idea of the refined self_contained method.

simply collect only one keystream prefix set A and virtualize the other set B by directly computing it by himself/herself, i.e., B is not captured. Since we are playing with the internal states directly, the adversary could randomly produce the internal state $\mathbf{x}$ so that it will have the keystream prefix $\mathbf{z} = \texttt{prefix}$. As stat-ed before, the premise in Lemma 1 is just the randomness of the two sets A and B, it does not restrict the way how the adversary gets them. Thus, the adversary could generate one keystream prefix set by himself/herself *without* knowing any secret key. Given the pre-computed table T mapping from a specified $\texttt{ksd}$ to all the possible ISDs, each partner state $\mathbf{x}' = \mathbf{x} \oplus \Delta\mathbf{x}$ will be checked and stored in the list L only if it would generate $\mathbf{z}' = \mathbf{z} \oplus \texttt{ksd}$. The following Algorithm a shows this technique in the general form, i.e., it could deal with both the full and the restricted internal state cases. This method will be integrated into the

A5/1 setting with the corresponding chosen parameters and formally introduced in section 4.4.

Algorithm a The refined self-contained method in general [10, 16]

```
1: for each prefix in $A$ do
2:   Let $i = 0$
3:   while $i \leq c \cdot 2^n/|T|$ do
4:     randomly generate a new (full or restricted) internal state $\mathbf{x}$
5:     such that $\mathbf{x}$ produces $\mathbf{z} = \texttt{prefix}$
6:       for each $\Delta x$ in |T| do
7:         if $\mathbf{x}' = \mathbf{x} \oplus \Delta x$ generates $\mathbf{z}' = \texttt{prefix} \oplus \texttt{ksd}$ then
8:           store $\mathbf{x}'$ into the list L indexed by prefix
9:     $i = i + 1$
10: Output: The list L
```

In general, there are two phases in a fast near collision attack: the offline and online phases. The pre-computation phase is quite efficient due to the divide-and-conquer strategy, in which the adversary tries to construct some relatively small tables, instead of one large table, mapping from a fixed `ksd` to all the possible ISDs of the corresponding restricted internal state. Now the adversary does not need to exhaustively search through all the possible ISDs over the full internal state; instead, he/she just search through all the possible ISDs over a specified restricted internal state associated with a given keystream `prefix`, which could be much smaller than the whole internal state. In the online phase, the adversary uses the above self-contained method to get a list of the candidates of the targeted restricted internal state. There is a distilling procedure afterwards to get a smaller list of the candidates with a higher existence probability of the correct partial internal state, which consists of some set intersection and union operations with carefully selected parameters. Then merge the restored restricted internal states together to cover the chosen CP portion of the full internal state, which is used later as the starting point for the full recovery of the internal state.

We will go into the details of the refined self-contained method, the distilling and merge procedures, and the final retrieval of the targeted internal state in the context of A5/1 in the following sections.

4 Our New Attack

In this section, we will present our new attack against A5/1 in a step-by-step manner following the above cryptanalytic principles.

4.1 A General Description of the Attack

We first present a high-level overview of our attack. As stated in section 3, the goal is to first recover the CP part of the internal state, and then the RP part at a fixed time instance which is consistent with the captured keystream.

The main idea is as follows. In A5/1, the size of the internal state is only 64 bits, thus in the assumed fast near collision attack, if we target a well-chosen CP part of the internal state, the pre-computation and the memory complexities will be considerably reduced compared to the previous best attacks. What we need to do is just to prepare the partial differential tables which record the mappings from the l-bit KSDs to all the possible ISDs of the restricted internal state with the sorted occurring probabilities. In the online phase, we exploit the special internal structure of A5/1 to partition the 64-bit internal state into the CP and the RP part, and its irregular clocking mechanism to launch the concrete fast near collision attack against it. Formally, a high-level description of our attack is depicted in Algorithm 1.

Algorithm 1 Fast near collision attack on A5/1

```
Parameters: $l, \alpha, \gamma$
Offline: Prepare the tables T[ksd, prefix]
 1: for each possible value of (ksd, prefix) do
 2:     use the method in section 4.3 to construct T[ksd, prefix]
Input:  A keystream segment $\mathbf{z} = (z_0, z_1, \ldots, z_{\gamma-1})$
Online: Recover the full internal state matching with $\mathbf{z}$
 3: Divide $\mathbf{z}$ into $\alpha$ overlapping prefixes $\mathbf{z}_i$ $(0 \leq i \leq \alpha - 1)$ and a suffix $\mathbf{z}_\mu$
 4: for $i = 0$ to $\alpha - 1$ do
 5:     derive the partial state list $L_i$ for $\mathbf{z}_i$ in section 4.4 and 4.5
 6: Merge $L_i$s to get a candidate list for the CP part in section 4.6
 7: Recover the RP in section 4.7 and check the consistency with $\mathbf{z}_\mu$
```

In the following, we will embed Algorithm 1 into the concrete attack scenario of A5/1 to demonstrate the attack in details.

4.2 Basic Facts of A5/1

As described in section 2, A5/1 adopts the stop/go clocking fashion according to a majority function defined over the 3 taps from R1, R2 and R3, respectively. Thus, for one keystream bit z_i, it actually depends on 9 internal state bits: the 2 leftmost bits from each register, i.e., R1[18], R1[17], R2[21], R2[20], R3[22] and R3[21], and the three clock control bits R1[8], R2[10] and R3[10].

Further, as discussed in section 3.1, the adversary needs to randomly generate the keystream prefix without knowing the secret key in a fast near collision attack. In this case, we could use either the oracle which generates the corresponding keystream prefix directly or the following Algebraic Norm Form (ANF) of z_i, which could be verified by enumerating all the possible values of $x = (x_0, x_1, x_2, \ldots, x_8)$:

$$\begin{aligned} f(x) = {} & x_3 + x_4 + x_5 + x_0x_6 + x_3x_6 + x_1x_7 + x_4x_7 + x_2x_8 + x_5x_8 \\ & + x_0x_6x_7 + x_1x_6x_7 + x_2x_6x_7 + x_3x_6x_7 + x_4x_6x_7 + x_5x_6x_7 \\ & + x_0x_6x_8 + x_1x_6x_8 + x_2x_6x_8 + x_3x_6x_8 + x_4x_6x_8 + x_5x_6x_8 \\ & + x_0x_7x_8 + x_1x_7x_8 + x_2x_7x_8 + x_3x_7x_8 + x_4x_7x_8 + x_5x_7x_8, \end{aligned}$$

where x_0=R1[18], x_1=R2[21], x_2=R3[22], x_3=R1[17], x_4=R2[20], x_5=R3[21], x_6=R1[8], x_7=R2[10] and x_8=R3[10]. As in [16], in order to have an efficient

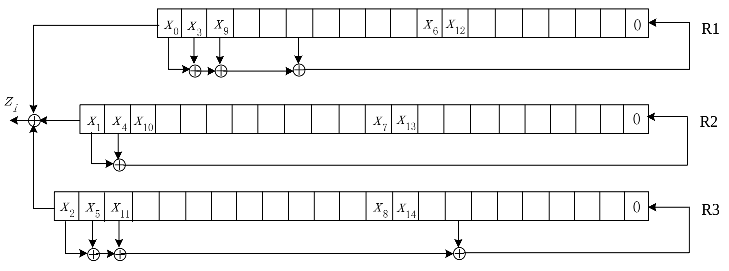

Fig. 3. The 15-bit restricted internal state that a 2-bit keystream prefix depends on.

fast near collision attack, we have also tried to use the 2-bit keystream prefix to recover the corresponding restricted internal state, which contains 6 more variables than the single bit case, as depicted in Figure 3. Due to the action of the majority function, there are now 15 input variables involved for a 2-bit keystream prefix. Precisely, in addition to the above 9 state bits in the ANF of z_i, we have x_9=R1[16], x_{10}=R2[19], x_{11}=R3[20], x_{12}=R1[7], x_{13}=R2[9] and x_{14}=R3[9]. For a 2-bit keystream prefix, we just directly exploit the keystream generation oracle to generate it in the refined self-contained method.

4.3 Pre-computing the Partial Differential Tables

Now we come to the offline phase of Algorithm 1. Since there are only $n = 15$ or $n = 9$ variables involved for the 2-bit keystream prefix or 1-bit keystream, given a small value of d, we could fully enumerate all the possible values of the corresponding restricted internal state and all the $v(n, d)$ ISDs to *accurately* compute the occurring probabilities of each ISD.

Algorithm 2 The offline algorithm

```
Parameters: $n, d$ and ksd, prefix
 1: for each possible value of (ksd, prefix) do
 2:   Initialize the table T[ksd, prefix]
 3:   for each of the $v(n,d)$ $\Delta x$ do
 4:     Initialize $t = 0$ and $cc = 0$
 5:     for each $i \in \{0, 1, \cdots, 2^n - 1\}$ do
 6:       check if $\mathbf{x} = i$ generates the prefix
 7:       if yes then
 8:         $t = t + 1$
 9:         generate the partial internal state $\mathbf{x}' = \mathbf{x} \oplus \Delta x$
10:         compute the prefix $\mathbf{z}'$ generated by $\mathbf{x}'$
11:         if $\mathbf{z}'$ = prefix $\oplus$ ksd then store $\Delta x$ and $cc = cc + 1$
12:   Sort the ISDs according to the occurring rates $cc/t$
```

Algorithm 2 describes how to generate a series of pre-computed tables, which record the mappings from a specified `ksd` to all the possible ISDs Δx, each with a sorted occurring probability. The core point here is that the only premise in Lemma 1, i.e., the randomness, enables the adversary to freely compute the second set B by himself/herself. Thus, the adversary would know the matching between the partial internal state $\mathbf{x}'$ and the output prefix $\mathbf{z}' = \texttt{prefix} \oplus \texttt{ksd}$.

Table 1. The complete differential table when $ksd = 0x3$ and $d = 2$

ISD	Prob.	ISD	Prob.	ISD	Prob.	ISD	Prob.
(x_3, x_9)	0.75	(x_6, x_{11})	0.25	$(x_6, -)$	0.25	(x_0, x_5)	0.125
(x_5, x_{11})	0.75	(x_2, x_6)	0.25	(x_1, x_7)	0.25	(x_0, x_4)	0.125
(x_4, x_{10})	0.75	(x_6, x_{13})	0.25	$(x_7, -)$	0.25	(x_1, x_{14})	0.125
(x_0, x_3)	0.4375	(x_6, x_{14})	0.25	$(x_8, -)$	0.25	(x_1, x_{11})	0.125
(x_2, x_5)	0.4375	(x_2, x_7)	0.25	(x_6, x_{12})	0.234375	(x_2, x_4)	0.125
(x_1, x_4)	0.4375	(x_7, x_9)	0.25	(x_7, x_8)	0.234375	(x_2, x_{12})	0.125
(x_3, x_{12})	0.375	(x_2, x_8)	0.25	(x_6, x_8)	0.234375	(x_0, x_{11})	0.125
(x_5, x_{14})	0.375	(x_7, x_{10})	0.25	(x_8, x_{14})	0.234375	(x_2, x_{13})	0.125
(x_4, x_{13})	0.375	(x_0, x_7)	0.25	(x_7, x_{13})	0.234375	(x_1, x_{12})	0.125
(x_3, x_{14})	0.3125	(x_7, x_{11})	0.25	(x_6, x_7)	0.234375	(x_1, x_5)	0.125
(x_4, x_9)	0.3125	(x_1, x_6)	0.25	(x_4, x_8)	0.21875	(x_2, x_3)	0.125
(x_4, x_{12})	0.3125	(x_7, x_{12})	0.25	(x_3, x_7)	0.21875	(x_0, x_{13})	0.125
(x_4, x_{14})	0.3125	(x_1, x_8)	0.25	(x_5, x_6)	0.21875	(x_1, x_2)	0.125
(x_5, x_9)	0.3125	(x_7, x_{14})	0.25	(x_5, x_7)	0.21875	(x_0, x_{14})	0.125
(x_3, x_{11})	0.3125	(x_6, x_{10})	0.25	(x_3, x_8)	0.21875	(x_0, x_1)	0.125
(x_5, x_{10})	0.3125	(x_8, x_9)	0.25	(x_4, x_6)	0.21875	(x_1, x_{10})	0.0625
(x_3, x_{13})	0.3125	(x_4, x_5)	0.25	$(x_3, -)$	0.1875	(x_0, x_{12})	0.0625
(x_5, x_{12})	0.3125	(x_8, x_{10})	0.25	$(x_4, -)$	0.1875	(x_2, x_{11})	0.0625
(x_3, x_{10})	0.3125	(x_0, x_6)	0.25	$(x_5, -)$	0.1875	$(x_1, -)$	0.0625
(x_5, x_{13})	0.3125	(x_8, x_{11})	0.25	(x_0, x_{10})	0.125	(x_2, x_{14})	0.0625
(x_4, x_{11})	0.3125	(x_3, x_5)	0.25	(x_2, x_9)	0.125	(x_1, x_{13})	0.0625
(x_3, x_6)	0.28125	(x_8, x_{12})	0.25	(x_1, x_9)	0.125	(x_0, x_9)	0.0625
(x_4, x_7)	0.28125	(x_6, x_9)	0.25	(x_2, x_{10})	0.125	$(x_2, -)$	0.0625
(x_5, x_8)	0.28125	(x_8, x_{13})	0.25	(x_1, x_3)	0.125	$(x_0, -)$	0.0625
(x_0, x_8)	0.25	(x_3, x_4)	0.25	(x_0, x_2)	0.125		

Now we choose $d = 2$ in Algorithm 2, i.e., we consider the ISDs of Hamming weight less than or equal to 2-bit, there are $v(15, 2) = \sum_{i=0}^{2} \binom{15}{i} = 121$ ISDs for the 15-bit restricted internal state under consideration. For each ISD Δx satisfying $w_H(\Delta x) \leq d$, we enumerate all the possible values of the 15-bit restricted internal states $\mathbf{x}$. If $\mathbf{x}$ can generate the considered `prefix`, we xor it with the Δx and save Δx into the pre-computed table if and only if the xored state $\mathbf{x}'$ could generate $\texttt{prefix} \oplus \texttt{ksd}$. In Algorithm 2, t is the number of times that $\mathbf{x}$ generates the `prefix`, and cc is the number of occurrences of the event that $\mathbf{z}' = \texttt{prefix} \oplus \texttt{ksd}$ under the condition that $\mathbf{x}$ generates the `prefix`. Finally, the table is sorted according to the occurring probabilities of each ISD. We have computed T[`ksd`, `prefix`] for each combination of (`ksd`, `prefix`) when $l = 1$ and 2 with a low complexity of $2^{2l} \cdot v(n,d) \cdot 2^n \cdot l$. We list the complete table when `ksd` = `0x3` with an arbitrary 2-bit keystream `prefix` when $d = 2$ in Table 1.

Note that in Table 1, (x_i, x_j) for $0 \leq i, j \leq 14$ means the 2-bit differences are at the positions x_i and x_j in Figure 3, respectively. For $(x_i, -)$ with $0 \leq i \leq 14$, it means the 1-bit difference is at the position x_i only. Given all the pre-computed tables T[`ksd`, `prefix`], it is easy to see that for a fixed `ksd`, the average reduction effect of each table indexed by `prefix` is almost the same, which is measured by the diversified probability defined below and in [16].

Definition 8. *The diversified probability is defined as* $P_{divs} = \frac{\sum_{\Delta x \in T} Pr_{\Delta x}}{|T|}$, *where* Δx *ranges over all the* $|T|$ *possible ISDs in the table T[`ksd`,`prefix`].*

This probability measures the average reducing effect of T[`ksd`, `prefix`] so that for a random restricted internal state **x** generating `prefix`, if the bits in **x** are flipped according to a $\Delta x \in T$ to get $\mathbf{x}'$, then with the probability P_{divs}, $\mathbf{x}'$ could produce `prefix` $\oplus$ `ksd`. The observation that P_{divs} is mainly determined by `ksd` further reduces the memory requirement of the pre-computation. Thus, given the fixed `ksd=0x3`, Table 1 actually works for an arbitrary keystream `prefix`. This property is crucial for the partial restricted state recovery, since the recovery procedure will be the same even if the keystream `prefix` under investigation has varied along the captured keystream, which is shown in Algorithm 3 in section 4.4. It is interesting to see that this small differential table only has $99 < 121$ possible ISDs when `ksd` = `0x3` and $d = 2$. Note that in the previous TMD trade-off attacks, the adversary is expected to search through all the possible ISDs over the full internal state, which will result in a huge pre-computation complexity and memory consumption. Now we can just try the 99 possible ISDs over the 15-bit restricted internal state in the online phase. Further, the occurring probabilities of the ISDs in Table 1 ranges from 0.75 to 0.0625 and their distribution is heavily biased. We have tested all the cases for $l = 1$ and $l = 2$ with $2 \leq d \leq 4$ to find the optimal choice of the pre-computed table T[`ksd`, `prefix`]. The results are listed in Table 2.

Table 2 clearly shows that for the case $l = 2$ and $n = 15$, we have $P_{divs} = 0.234848$, which is the minimum value among all the empirical results. Besides, there are some more reasons for choosing this configuration, which are briefly discussed below. The cost of merging the candidates list in the following section 4.6 accounts for some proportion of our attack, thus we expect to have less candidates for the involved restricted internal state. Note that the candidates are generated by xoring all the possible ISDs in the table T[`ksd`, `prefix`] in the refined self-contained method, thus we prefer a small number of possible ISDs in the selected pre-computed table. Further, we expect the average reduction effect, measured by P_{divs}, to be as low as possible for the efficiency reasons.

In Table 2, given $2 \leq d \leq 4$ and $0 \leq$ `ksd` ≤ 3, we have enumerated all the ISDs Δx such that $w_H(\Delta x) \leq d$. We have tested the 2^9 restricted internal states for the 1-bit keystream case and the 2^{15} restricted internal states for the 2-bit keystream prefix case, respectively. The experimental results in Table 2

Table 2. The empirical results for $l = 1$ and $l = 2$ with $2 \leq d \leq 4$

l	d	ksd	|T|	P_{divs}	l	d	ksd	|T|	P_{divs}
1	2	0x0	43	0.511628	2	2	0x0	118	0.313559
		0x1	45	0.533333			0x1	96	0.289063
		–	–	–			0x2	111	0.297297
		–	–	–			0x3	99	0.234848
	3	0x0	127	0.507874		3	0x0	555	0.269876
		0x1	128	0.511719			0x1	547	0.272795
		–	–	–			0x2	521	0.269494
		–	–	–			0x3	533	0.256273
	4	0x0	253	0.509881		4	0x0	1875	0.258133
		0x1	251	0.505976			0x1	1864	0.261266
		–	–	–			0x2	1853	0.261603
		–	–	–			0x3	1874	0.258938

imply that it seems to be a good choice to restore the involved 15-bit restricted internal state for $l = 2$ with $(\texttt{ksd}, d) = (\texttt{0x3}, 2)$. There are only 99 possible ISDs involved in the corresponding pre-computed table and the diversified probability is 0.234848. Note that each ISD in the table has 15 bits and can be stored in 2 bytes, which means that this table only requires about 198 bytes memory complexity.

4.4 Determining the Candidates List of the Involved Restricted Internal State

Now we are ready to enter the online phase of Algorithm 1, whose aim is to recover the restricted internal states for the targeted 2-bit keystream prefixes by using the pre-computed tables prepared in section 4.3.

Algorithm 3 The refined self-contained method in online phase

```
Parameters: $l = 2$, $n = 15$, $d = 2$ and $\texttt{ksd} = \texttt{0x3}$
            $\texttt{iter\_num} = 4 \cdot 2^{15}/|\mathrm{T}[\texttt{ksd}, \texttt{prefix}]| = 4 \cdot 2^{15}/99$
1: Initialize $i = 0$
2: while $i \leq \texttt{iter\_num}$ do
3:    randomly generate a new restricted internal state $\mathbf{x}$
4:    such that $\mathbf{x}$ produces $\mathbf{z} = \texttt{prefix}$
5:       for each of the |T| possible ISDs $\Delta x$ do
6:          if $\mathbf{x}' = \mathbf{x} \oplus \Delta x$ generates $\mathbf{z}' = \texttt{prefix} \oplus \texttt{ksd}$ then
7:             store $\mathbf{x}'$ into the list L
8:    $i = i + 1$
9: Output: The list L
```

According to Corollary 1, given the pre-computed table $\mathrm{T}[\texttt{ksd}, \texttt{prefix}]$, let A with $|A| = 1$ be the 15-bit restricted internal state associated with a 2-bit keystream prefix and B with $|B| = c \cdot \frac{2^{15}}{|T|}$ be the virtualized 15-bit state in the refined self-contained method, then there should exist an element $b_i \in B$

such that the pair (a, b_i) with the unique element $a \in A$ forms a d-near collision pair with a probability dependent on c. This is the theoretical basis of Algorithm 3, where $c = 4$.

Recall that in the previous near collision attack in [15], the adversary needs to collect two random keystream vector sets, A and B, and tries to identify a d-near-collision state pair at two different time instants from the corresponding keystream segments. In this process, a strong wrong-candidate filter with a low complexity is needed, while in its form in [15], the reduction effect is not as good as expected. In [10], the self-contained method is introduced as a generic approach to obtain the candidates of the involved internal state. As briefly mentioned in section 3.1, this method only requires one keystream prefix set, and allows the adversary to freely compute the other one based on the pre-computed table. Precisely, let A be the collected keystream prefix set, the adversary just randomly generates the other set B by himself/herself. For $\mathbf{x}' \in A$ and $\mathbf{x} \in B$, if the corresponding keystream prefixes satisfying $\mathbf{z}' \oplus \mathbf{z} = \mathtt{ksd}$ with $\mathtt{ksd}$ being the concrete value of the considered KSD, since the adversary knows the matching between the internal state $\mathbf{x}$ and its corresponding output $\mathbf{z}$, he/she could restore the targeted internal state $\mathbf{x}'$ by trying $\mathbf{x}' = \mathbf{x} \oplus \Delta x$. Thus the adversary could generate the candidates list of the targeted restricted internal state $\mathbf{x}'$ in this way. Taking into account the divide-and-conquer strategy on the partition of the full internal state, we have Algorithm 3.

Note that Steps 3 and 4 in Algorithm 3 can be fulfilled by a method similar to the BSW sampling enumeration technique in [5], but with a very small $l = 2$. Besides, Corollary 1 further implies that there is an inherent relationship between c, d and the existence probability of one good pair. For A5/1, we list this correspondence when 2-bit keystream prefix is considered with $d = 2$ in Table 3, which is achieved from 10^6 times repetition of empirical simulations. We have tried all the meaningful combinations of parameter configuration from the practical implementation point of view, and determined to choose $c = 4$ in our attack. Note that in this case, when $l = 2$ and $d = 2$, we have $n = 15$ and $v(n, d) = 121$ and the existence probability in one invoking of the refined self-contained method is 0.9835. As can be seen in the following sections, this choice provides a very good balance between the complexity aspects and the success probability.

Table 3. The correspondence between the constant c, the list size and the existence probability of one good pair for A5/1 when $d = 2$

c	List size r	Prob.
2	6903	0.8475
3	7654	0.9510
4	7963	0.9835

Further, Let us have a closer look at Algorithm 3 and Table 3 together. There is a magic fact here that though the adversary has iterated $4 \cdot 2^{15} / |\mathrm{T}[\mathtt{ksd}, \mathtt{prefix}]|$

$= 4 \cdot 2^{15}/99 \doteq 1324$ times and there are 99 ISDs to be xored and checked the consistency with $\mathbf{z}'$ in each iteration in Algorithm 3, the number of hit values stored in the list L is much less than $\frac{2^{15}}{2^2} = 8192$ with a high existence probability of the correct candidate being in L, as can be seen from Table 3. All these facts can be well explained through the following Theorem 1 and are illustrated in *Example* 1.

Theorem 1. *([16]) Let b be the number of all the values that can be hit and $a = c \cdot \frac{2^n}{|D|} \cdot |T| \cdot P_{divs}$, then after one invocation of the refined self-contained method, the expectation of the final number r of hitting values in the candidates list is*

$$E[r] = \sum_{r=1}^{a} \frac{\binom{b}{r} \cdot r! \cdot \left\{ {a \atop r} \right\} \cdot r}{b^a}, \tag{1}$$

where $\left\{ {a \atop r} \right\}$ is the Stirling number of the second kind, $\binom{b}{r}$ is the binomial coefficient and $r!$ is the factorial.

Example 1. Let $c = 4$, from Eq.(1), the mathematical expectation of the list size can be calculated as $E[r] = \sum_{r=1}^{a} \frac{\binom{b}{r} \cdot r! \cdot \left\{ {a \atop r} \right\} \cdot r}{b^a} \doteq 2^{12.96} \doteq 7963$, where $l = 2$, $n = 15$, $b = \frac{2^{15}}{2^2} = 8192$, $a = 4 \cdot \frac{2^{15}}{|T|} \cdot |T| \cdot P_{divs} = 4 \cdot 2^{15} \cdot P_{divs} = 2^{17} \cdot 0.234848 = 30782$. Here $P_{divs} = 0.234848$ is the diversified probability of the selected precomputation table T[`ksd`, `prefix`] defined in Definition 8 in section 4.3. □

4.5 Distilling Phase: Enhancing the Existence Probability of the Correct Candidate

In the real online attack, the adversary usually wants to have a candidate list of smaller size, while at the same time still containing the correct restricted internal state with a reasonably good probability. That is, we want to efficiently filter out the wrong candidates of the involved restricted internal state got from Algorithm 3. Note that due to the partition of the full internal state, now we cannot run A5/1 from the involved partial state forwards and backwards to check the consistency with the available keystream. This is the motivation of the distilling phase in this section, i.e., we need an efficient wrong-candidate filter without knowing the full internal state. For consistency with the work in [16], we follow the conventional notations and descriptions in this domain hereafter.

Algorithm 4 The Distilling Procedure 1: Intersection [16]

```
Parameters: $L = (L_1, \ldots, L_\beta)$: $\beta$ candidate lists of
            the targeted partial state from Algorithm 3
1: for $i = 1$ to $\beta - 1$ do
2:     $L_{i+1} \leftarrow L_i \cap L_{i+1}$
```

To improve the hitting rate of the correct restricted internal state while keeping a smaller list size, we continuously modify the involved candidate lists got from Algorithm 3 by set intersection and union, depicted in Algorithm 4 and

5, respectively. Note that the candidates in list L are generated randomly in Algorithm 3 with the following property: the correct one has an existence probability determined by the constant c, which is usually higher than the random wrong ones, which appear in the list randomly. Some candidate of the restricted internal state may be hit several times by xoring different ISD Δx with different values of $\mathbf{x}$ in Algorithm 3.

Example 2. Let $\mathbf{x} =$ `111000010111110` be the correct restricted internal state, where the leftmost bit is x_0 and the rightmost bit x_{14}. It is observed that $\mathbf{x}$ is hit 30 times in one invocation of Algorithm 3 with the following ISDs: $(x_5, -)$, (x_3, x_4), $(x_5, -)$, (x_5, x_{13}), (x_3, x_4), (x_1, x_7), (x_7, x_{11}), (x_1, x_7), (x_1, x_9), (x_2, x_7), (x_2, x_5), (x_2, x_7), (x_2, x_6), (x_6, x_8), (x_5, x_{14}), (x_4, x_8), (x_6, x_8), (x_7, x_{12}), (x_1, x_9), (x_7, x_{11}), (x_5, x_{14}), (x_5, x_{13}), $(x_7, -)$, (x_0, x_5), (x_5, x_{10}), (x_1, x_{14}), (x_1, x_9), (x_2, x_6), (x_4, x_8), (x_3, x_{14}). □

Thus, it is natural to intersect the resultant lists from different independent invokes of Algorithm 3 to reduce the number of candidates, while still containing the correct one with a reasonable probability. During the intersection process, many wrong candidates can be removed from the list. Algorithm 4 depicts this distilling process. As proved in [16], the expected number of candidates after $\beta - 1$ steps of intersection can be estimated by $|L_1| \cdot (\frac{E[r]}{b})^{\beta-1}$, where $|L_1|$ is the number of candidates in the first generated list L_1, b and $E[r]$ are defined and listed in Theorem 1. We have found in practical experiments that $\beta = 6$ is a good choice for the reduction purpose when attacking A5/1, with an appropriate choice of the following parameter γ in Algorithm 5 at the same time.

On the other hand, we have also observed in the experiments that for a single invoking of Algorithm 3, there is some missing probability of the event that the correct partial state is not in the candidate list. If such an event happens, we may miss the correct partial state through the intersection process. This fact indicates that we should take some action to remedy such a situation and the existence probability of the correct restricted internal state in this case. The following Algorithm 5 resolves this problem by the set union operation. Precisely, after getting some intersected lists, we adopt Algorithm 5 to generate a candidate list for the considered restricted internal state. The union operation can well mitigate the influence of the missing event in Algorithm 3, i.e., as long as the correct restricted internal state survives in one of the γ lists obtained from Algorithm 4, it will be in the final list after Algorithm 5. Once Algorithm 5 is executed, we can get a candidate list for the n-bit restricted internal state associated with the l-bit keystream prefix.

Algorithm 5 The distilling procedure 2: Union [16]

```
Parameters: $U = (U_1, \dots, U_\gamma)$: $\gamma$ lists
            obtained from Algorithm 4
1: for $i = 1$ to $\gamma - 1$ do
2:     $U_{i+1} \leftarrow U_i \cup U_{i+1}$
```

Further, it is also proved in [16] that after the preparation of γ resultant lists from Algorithm 4, the expected number of candidates $|F_{i+1}|$ after i steps of union can be recursively derived as

$$|F_{i+1}| = |F_i| + |U_{i+1}| - \sum_{j=0}^{|U_{i+1}|} \frac{\binom{|F_i|}{j} \cdot \binom{|F_{i+1}|-|F_i|}{|U_{i+1}|-j}}{\binom{|F_{i+1}|}{|U_{i+1}|}} \cdot j \ , \ 1 \le i \le \gamma - 1, \qquad (2)$$

where F_i is the resultant list after $i-1$ steps of union. Eq.(2) is derived from the fact that $|F_{i+1}| = |F_i| + |U_{i+1}| - |F_i \cap U_{i+1}|$ and the $|F_i \cap U_{i+1}|$ term follows the hypergeometric distribution. Finally, the following corollary provides the existence probability of the correct restricted internal state after Algorithm 5.

Corollary 2. *Let Pr_{Alg5} be the probability that the correct restricted internal state will exist in the final list generated by Algorithm 5, we have $Pr_{Alg5} = 1-(1-(p_1)^\beta)^\gamma$, where p_1 is the existence probability of the correct restricted internal state in Table 3.*

Proof. It suffices to consider the opposite event that the correct restricted internal state does not exist after Algorithm 5, which means that it does not exist in any of the γ lists, each going through $\beta - 1$ intersections. This completes the proof. $\square$

In order to have a good existence probability of the correct restricted internal state with a smaller size of candidate list, we have tried $\beta = \{2,3,4,5,6,7,8,9\}$ and $\gamma = \{2,3,4,5\}$ in the experiments to identify the optimal parameter configuration for attacking A5/1. Here we only present some of the simulation results in Table 4.

Table 4. Some results for different parameter configurations with $c = 4$

γ	β	$\|U\|$	`Prob.`
2	3	8065	0.9940
2	4	7989	0.9927
2	5	7934	0.9912
2	6	7835	0.9903

From this table, the configuration that $\beta = 6$ and $\gamma = 2$ is selected to be used in our attack against A5/1. The success probability can be calculated as $Pr_{Alg5} = 1-(1-0.9835^\beta)^\gamma = 0.9909$ according to Corollary 2, which is quite close to the result 0.9903 obtained from practical experiments, while the expected list size should be $2^{12.95}$ in theory according to the Eq.(2), which is also very close to the simulation result $7835 \doteq 2^{12.94}$. The following Lemma is the basis of our non-randomness observation after Algorithm 5.

Lemma 2. *(Chebyshev's Inequality) Let X be a random variable with the finite expected value μ and finite non-zero variance σ^2. Then for any real number $k > 0$, we have $Pr(|X-\mu| \ge k\sigma) \le \frac{1}{k^2}$ and only the case $k > 1$ is useful.*

Based on Chebyshev's Inequality, we have the following statements on the non-randomness observation of the resultant candidate list after Algorithm 5.

Theorem 2. *Let $\beta = 6$ and $\gamma = 2$ in Algorithm 4 and 5, if $c = 4$, $\texttt{ksd} = \texttt{0x3}$, and $d = 2$ for a 2-bit keystream prefix, then the candidates list generated after Algorithm 5 has an averaged size of* 7835 *with the existence probability* 0.9903 *for the correct candidate being in the list, which is a non-random case.*

Proof. Here we follow the classic way of distinguishing two distributions in theory to show the non-randomness. Below we will investigate the distribution in the pure random case and that obtained in our attack respectively.

For a specified 2-bit keystream prefix, the candidate space has a size of $\frac{2^{15}}{2^2} = 2^{13} = 8192$ in the pure random case. The probability that a candidate 15-bit restricted internal state will generate the specified keystream prefix is $p = \frac{1}{4}$ and $q = 1 - p = \frac{3}{4}$ otherwise. We regard the list size as a sum of random variables which follow the binomial distribution with the corresponding parameters in each case and approximate it with the normal distribution. Then the standard deviation in the pure random case is $\sigma = \sqrt{2^{13} \cdot p \cdot q} \approx 39.19$. Further, the expectation of the list size in the pure random case with the existence probability 0.9903 should be $\mu_{union} = 2^{13} \cdot 0.9903 \approx 8113$.

On the other side, in our attack against A5/1 after Algorithm 5, the averaged list size is $\mu'_{union} = 7835$ with the existence probability 0.9903. Then according to the Chebyshev's inequality in Lemma 2, we can compute the coefficient k as

$$k = \frac{|\mu_{union} - \mu'_{union}|}{\sigma} \approx \frac{8113 - 7835}{39.19} \approx 7.09.$$

Thus, we can conclude that the resultant list size in our attack is non-random with a probability greater than 99%. This completes the proof. □

This non-random phenomenon is the basis of our new attack against A5/1. Note that the probability in theory 0.9909 and empirical rate 0.9903 from Table 4 are very close to each other, we have chosen to use the 'worse' value 0.9903 in the proof of Theorem 2 to demonstrate the strong validity of our attack.

4.6 Merging Phase: Restoring the CP Part of the Internal State

Now we show how to restore the CP part of the full internal state through the merging phase in Algorithm 1. The main difference between A5/1 and the target Grain v1 in [16] is that A5/1 executes according to the specified irregular clocking mechanism in a stop/go manner, while Grain v1 runs regularly. Though it is commonly believed that the irregular clocking mechanism improves the security of the primitive by introducing certain implicit non-linearity, we have found in the mounted fast near collision attack that the stop/go irregular clocking in A5/1 actually facilitates the list merging procedure in the attack. Let us first introduce a notion that is used in our attack.

Definition 9. *According to the stop/go clocking rule in A5/1, the intersection set of the two partial states at the time instants* t *and* $t+1$ *is called the check-state in the merging phase.*

From the irregular clocking rule in A5/1, the check-state in the merging phase has the following property.

Proposition 1. *The check-state in the merging phase has a cardinality of* 9 *if all the three registers are clocked once and has a cardinality of* 11 *if two registers are clocked once and one register stops at the corresponding time instant.*

Proof. There are 8 possible values for the three clock control bits, R1[8], R2[10] and R3[10]. It suffices to check the 8 cases one-by-one to conclude that only when the three bits take the value pattern $(0, 0, 0)$ or $(1, 1, 1)$, there will be 9 bits in the check-state as if all the registers are clocked regularly; otherwise there is one register unchanged which will offer 5 bits, while each of the two clocked registers will offer 3 bits. This completes the proof. □

Example 3. Let `111011011010010` be the starting restricted internal state, where the leftmost bit is x_0 and the rightmost bit is x_{14}. From Figure 3, consider the following state transmission chain `111011011010010` $\rightarrow$ `111010010000000` $\rightarrow$ `010010010100101` $\rightarrow$ `010110111100101` $\rightarrow$ `110100101010001` to see the validity of the statements in Proposition 1. For example, when `111010010000000` $\rightarrow$ `010010010100101`, R2 stops with the clock control pattern `010`, thus the 5-bit state `11010` in R2 remains unchanged. □

Note that the event that not all the three registers move simultaneously happens with a probability of 0.75, in which case two more overlapping bits are gained for free from Proposition 1. This implies that in most cases, the check-state in the merging phase has an reduction effect which is unexpected from the cryptanalyst's point of view. Based on the check-state, we have Algorithm 6 for the merging phase, whose basic idea is to combine the candidates which are coincident on the identified check-state according to the clock control bits, and finally derive the candidates for the CP part.

Algorithm 6 Merging the lists from Algorithm 5 to restore CP

```
Parameters: $L_{z_i z_{i+1} \cdots z_{i+m-1}}$: one list from Algorithm 5
            $L_{z_{i+1} z_{i+2} \cdots z_{i+m}}$: the other list to be merged
            $L_{z_i z_{i+1} \cdots z_{i+m-1} z_{i+m}}$: the resultant list
1: Find the clock control bits set $J$ between the two lists
2: for each value pattern in $J$ do
3:     Form the subgroup of $L_{z_i \cdots z_{i+m-1}}$ according to the pattern
4:     Sort the subgroup according to the check-state values
5: for each value pattern in $J$ do
6:     Form the subgroup of $L_{z_{i+1} \cdots z_{i+m}}$ according to the pattern
7:     Sort the subgroup according to the check-state values
8: for each pattern in $J$ do
9:     for each value pattern of the corresponding check-state do
10:        Merge each pair of elements and save the result into $L_{z_i \cdots z_{i+m}}$
11: Output: $L_{z_i z_{i+1} \cdots z_{i+m-1} z_{i+m}}$
```

Algorithm 6 actually split the first list $L_{z_i \cdots z_{i+m-1}}$ into many sublists according to the value pattern of the clock control bits and the value of the corresponding check-state under each clock control pattern. Similarly, the second list $L_{z_{i+1} \cdots z_{i+m}}$ will be regrouped into sublists according to the value of the overlapping bits determined by each pattern of clock control bits. Then, select an element from each sublist with the same check-state pattern under the same clock control pattern to form a candidate of the merged state.

In our attack, we first generate 4 candidate lists[1], $L_{z_0 z_1}, L_{z_1 z_2}, L_{z_2 z_3}, L_{z_3 z_4}$, for the corresponding restricted internal states of the 5-bit keystream prefix $\mathbf{z} = (z_0, z_1, z_2, z_3, z_4)$, after that we run Algorithm 6 to merge the involved restricted internal states. The merged candidate list for the CP part of the internal state is used as the starting point for the following retrieval of the RP part in a guess-and-determine like manner. In our experiments, each list $L_{z_i z_{i+1}}$ for $0 \le i \le 3$ contains around 7835 candidates on average, thus storing the four lists costs about $\frac{7835 \cdot 15 \cdot 4}{2^{10} \cdot 8} \approx 58$ KB. The whole list merging process is depicted in Figure 4.

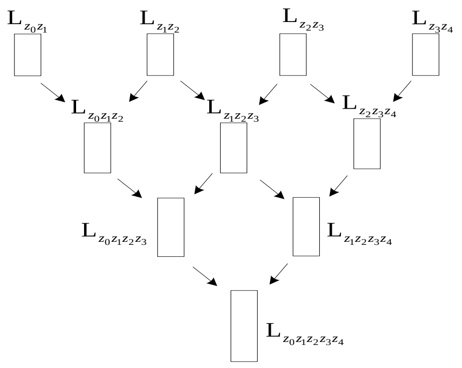

Fig. 4. The list merging process.

Precisely, in order to take the full advantage of the list size reduction during the merging process in Figure 4, we adopt the merging routine as follows: let the input lists be denoted by $L_{z_0 z_1}, L_{z_1 z_2}, L_{z_2 z_3}, L_{z_3 z_4}$. We first merge every two consecutive candidate lists, and get the new candidate lists $L_{z_0 z_1 z_2}, L_{z_1 z_2 z_3}, L_{z_2 z_3 z_4}$. Now the consecutive lists will have more overlapping state bits, and can further reduce the memory complexity to a large extent. We repeat the similar procedure to get $L_{z_0 z_1 z_2 z_3}$ and $L_{z_1 z_2 z_3 z_4}$. Finally, we merge the last two candidate lists to get the final list for the CP part of the internal state.

Before running the merging procedure, we have already generated 4 candidate lists in the online phase from the 5-bit keystream prefix. The expected size of

[1] Other choices are also possible, e.g., we can get the first 6 keystream bits to launch the attack. For simplicity of description, we take the first 5 keystream bits here.

the check-state between two consecutive lists at the most upper level in Figure 4 is $9 \cdot \frac{1}{4} + 11 \cdot \frac{3}{4} = 10.5$ bits. The averaged size of the input list is 7835, so it takes about

$$\frac{4 \cdot 8 \cdot 7835 + 3 \cdot 2^{12.3733} \cdot 2^{21}}{\Omega} \approx 2^{28.3}$$

cipher ticks to fulfill the merging procedure, where $\Omega = 2^{6.66}$ is the number of CPU-cycles to generate 1 bit keystream in A5/1. The detailed computation process of Ω is presented in section 5. The expected number of merged candidates is $2^{16.6}$, each candidate can be stored in 5 bytes at most, so the memory requirement is $2^{18.92}$ bytes, approximately 486 KB. Note that we usually need to store 2 lists in memory when merging, thus the total memory cost is around 1 MB.

Note that we can recover the correct merged partial internal state if and only if the two candidate lists contain the correct candidate associated with their keystream prefixes. Taking into account the tree-like merging procedure in Figure 4, the probability that the correct CP part will survive in the resultant list is $Pr_{merge} = (0.9903)^{\eta} = 0.9618$ where $\eta = 4$ is the number of the candidate lists $L_{z_i z_{i+1}}$. In other words, if we carry out the attack $\lceil \frac{1}{Pr_{merge}} \rceil \approx 2$ times, we are expected to find the actual CP part of the internal state from the resultant list. In our experiments, we have found that the probability p_1 defined in Corollary 2 is not stable sometimes, thus we multiply $\lceil \frac{1}{Pr_{merge}} \rceil$ by a small constant $\lambda = 4$ to recover the actual CP with a high probability.

Remarks. From the A5/1 specification and Figure 3, for an i-bit keystream prefix with $i \geq 2$, the associated restricted internal state is of size $15 + 6(i - 2)$ bits. That is, for a 2-bit keystream prefix, the restricted internal state has 15 bits; for a 3-bit keystream prefix, the restricted internal state has $15 + 6 = 21$ bits; for a 4-bit keystream prefix, the restricted internal state has $21 + 6 = 27$ bits and for a 5-bit keystream prefix, the restricted internal state has $27+6 = 33$ bits. So far, we have already obtained a candidate list of the restricted internal state of 33 bits associated with the first 5 keystream bits $(z_0, z_1, z_2, z_3, z_4)$ in a probabilistic way. This is accomplished by independently treat with the 4 overlapping 2-bit keystream prefixes (z_i, z_{i+1}) for $i = 0, 1, 2, 3$. Precisely, for each keystream prefix, we derive the corresponding candidate list by the method in section 4.4 and 4.5. By carefully choosing the attack parameters β and γ, we can guarantee that the corresponding correct restricted internal state for (z_i, z_{i+1}) is indeed in the candidate list with a reasonably good probability. Then we combine these 4 candidate lists together by the merging procedure in section 4.6 based on the *randomness and independence* assumptions to have the larger partial internal state corresponding to the first 5 keystream bits. Once the 4 correct restricted internal states, corresponding to the 4 keystream prefixes (z_i, z_{i+1}) for $i = 0, 1, 2, 3$, are restored successfully in each case, the merging procedure will definitely retrieve the correct union larger state corresponding to the first 5 keystream bits with probability 1, in which process the merging operation will also massacre lots of candidates when the two adjacent restricted internal states have a number of overlapping bits.

4.7 Restoring the RP Part of the Internal State

We have already derived the CP part of the internal state in A5/1 after the merging phase, the RP part will be retrieved in this section. We decided to take a dynamic guess-and-determine like method to restore the RP part, similar to the approach in [9]. The difference is that in [9], all the possibilities of the guessed part of the internal state are tried, while now we have already obtained some subset of the CP part in the internal state *without* trying all the possibilities, which will reduce the overall complexity to a large extent.

Let $S(t)$ be the internal state of A5/1 at time t, then the state recovered after the merging phase is a subset of $S(100)$, while $S(101)$ is the targeted state. Since the clock control taps have already been recovered, it is easy to get $S(101)$ by clocking $S(100)$ one step forwards. As analyzed in [9][2], the state-transition function of A5/1 is not one-to-one and there are less than or equal to $5 \cdot 2^{61} \approx 2^{63.32}$ reachable internal states for $S(101)$. Thus, we take 63.32 instead of 64 to analyze the complexity after one step forwards, and the unreachable states can be easily distinguished by a set of linear equations.

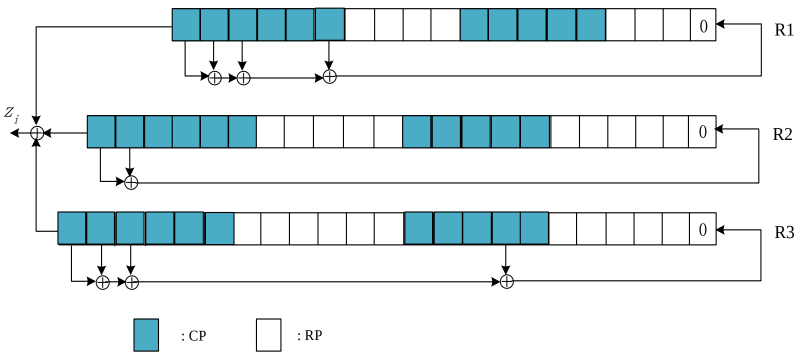

Fig. 5. The CP and RP parts of the internal state.

Our attack have exploited 5 keystream bits with $\eta = 4$ so far, as shown in section 4.6. Due to the irregular clocking rule in A5/1, each register moves $\frac{3}{4}$ of the time, which means that each register is expected to clock $5 \times \frac{3}{4} \approx 4$ times after the merging phase. From each 2-bit keystream prefix, we can recover 15 bits of the corresponding restricted internal state, of which 11 bits are overlapped by the next restricted internal state with a probability of $\frac{3}{4}$, and 9 bits are overlapped with a probability of $\frac{1}{4}$. Hence, there are around $15 + 3 \cdot 6 = 33$ known bits after the merging phase, and the total entropy of the full internal state will be reduced to $63.32 - 33 = 30.32$ bits *on average*, which is depicted in Figure 5.

[2] Though the clock control taps defined in [9] are different from here, this issue does not have any effect on the analysis of the state-transition properties under the condition that $\min(ct_1, ct_2, ct_3) \geq 2$.

The dynamic guess-and-determine attack is as follows. Note that in Fig.1 and Fig.5, the three registers are shifted towards the left direction and the newest feedback bits are injected into the state at the rightmost cells. The adversary could first guess y bits in each register from the first unknown clock control tap on to the right direction. One can thus obtain $3y$ linearly independent equations for the unknown bits in $S(101)$ if y is a small integer, e.g., $y = 4$ as shown in the register R1 in Fig.5 whose rightmost 4 cells are unknown. As in [9], the adversary can get one linear equation for free due to the fact that $S(101)$ is the targeted state and A5/1 first makes the stop/go clocking when producing the keystream bit. Corresponding to the directly guessed $3y$ clock control tap bits, there will be some number of linear equations obtained from the generated keystream bits for the adversary. Since on average each register clocks 3 times in 4 steps, the y guessed bits in each register can determine about $\frac{4y}{3}$ values of the clocking control taps in the register under consideration. Thus, the attacker can obtain around $1 + \frac{4y}{3}$ additional linear equations derived from the keystream bits on $S(101)$. When $y = 4$, each equation contains at least two new unknown bits that never appeared before, and these additional equations will be linearly independent to each other accordingly. The additional equations are linearly independent to the above $3y$ equations if and only if there is at least one bit not guessed in each of them. This is true when y is a small integer, e.g., $y = 4$. On the other side, if there are indeed some linear equations linearly dependent on the $3y$ single variable equations, these equations can be used as the linear consistency test of the guesses. Now there are around $3y + 1 + \frac{4y}{3} = 18.33$ linearly independent equations on average, and there are around $\tau = 30.32 - y = 12$ unknown bits which can be uniformly averaged among the three registers. Since there are some number of candidates left for the CP part when we only exploit the first 5 keystream bits, we need some very efficient procedure for the recovery of RP part. To fulfill this task, we then build a tree structure in the same manner as in [9] to derive the unknown bits dynamically and sequentially. Precisely, this tree structure sequentially record all the possibilities for the next coming bits consistent with the linear equations derived from the keystream bits. In each node of the tree, we store the next 3 clock taps of the three registers, representing the clocking which are consistent with the $1 + \frac{4y}{3}$ linear equations. As a result of the action of the majority function, there are 8 types of nodes, two of which have 8 possible successors and six of which have 4 possible successors. Thus, each node has $\frac{3}{4} \cdot 4 + \frac{1}{4} \cdot 8 = 5$ children in the tree on average. When checked with the corresponding linear equations, this number can be reduced to $5 \cdot \frac{1}{2} = 2.5$. Hence, to retrieve the τ unknown bits, the length of the tree should be $\frac{4}{3} \cdot \frac{\tau}{3}$ on average, and we have to check $2.5^{\frac{4}{3} \cdot \frac{\tau}{3}} \approx 2^{7.22}$ possibilities on average, instead of $2^{\tau} \approx 2^{12}$ possibilities under the independence assumption of the supercritical branching process, analyzed in the Appendix of [9]. In summary, the initial state $S(100)$ can be restored with a low complexity as shown above, the same as the state $S(101)$. Once we get the candidates of the full state, we can run A5/1 forwards for some ticks to check the consistency with the available keystream to decide whether the recovered state is correct or not.

5 Experimental Results

To check the validity and the actual performance of our attack, we have implemented the crucial steps of the suggested attack on a single core of a PC, running with Windows 7, Intel 3.4 GHz CPU and 16 GB RAM. In general, the experimental results verified the correctness of the crucial steps in the new method, and matched the theoretical prediction of each procedure quite well.

Since we take cipher ticks as the complexity unit in complexity analysis, we have first determined the constant Ω used in our attack. The source code of A5/1 for testing is a modification of a pedagogical implementation in [6]. We found that the average time to generate 1-bit keystream for A5/1 is $2.97 \cdot 10^{-8}$ seconds. Thus, one cipher tick of the A5/1 stream cipher accounts for $\Omega = 2.97 \cdot 10^{-8} \cdot 3.4 \cdot 10^{9} \doteq 2^{6.66}$ CPU cycles.

Since the basis of a fast near collision attack is the randomness, we use the RC4 stream cipher with a 128-bit secret key by discarding the first 8192 keystream words as the random source in our experiments. We adopted a random seed key dependent on the system time when invoking RC4, thus could assure that we have used random different sources among different calls[3].

To see the correctness of the suggested attack, there are essentially two points to check and verify. The first one is to assure that the correct restricted internal state associated with a 2-bit keystream prefix is indeed included in the candidates list generated by Algorithm 3 in section 4.4. The other is to assure that with the suggested attack parameters, the attack will indeed behave as predicted in theory, which actually means that the 4 correct restricted internal states are *simultaneously* included in $L_{z_0z_1}$, $L_{z_1z_2}$, $L_{z_2z_3}$ and $L_{z_3z_4}$, respectively.

To check the first statement, following the analysis in section 4.3, we have prepared the small differential tables indexed by $(\texttt{ksd}, d) = (\texttt{0x3}, 2)$ with all the possible ISDs in the pre-processing phase, requiring 198 bytes memory. Each pre-computed table is sorted according to the occurring probabilities of each involved ISD, which takes about $P = \frac{(2\cdot 2^4\cdot 2^{15}\cdot 121+2^4\cdot 99\cdot \log_2 99)}{\Omega} \approx 2^{20.26}$ cipher ticks.

Then in the online phase, we have tried all the 99 ISDs in the pre-computed table $c\cdot \frac{2^n}{|T|} = 4\cdot \frac{2^{15}}{99} \doteq 1324$ times to make sure that the generated list contains the correct restricted internal state under consideration, so it takes $\frac{4\cdot \frac{2^{15}}{99}\cdot 99}{\Omega} \approx 2^{10.34}$ cipher ticks for $n = 2^{15}$ and $|\texttt{T}| = 99$. Then we enhance the existence probability of the correct restricted internal state by intersecting $\beta = 6$ candidate lists and unifying $\gamma = 2$ such resultant lists, as suggested in Algorithm 4 and 5. The complexity of the intersection operation in Algorithm 4 is $2^{12.93}$ cipher ticks, and the union operation takes about $2^{13.93}$ cipher ticks. Next, we could check whether the correct restricted internal states are indeed contained in the corresponding candidate lists generated by Algorithm 3. Precisely, we just set a flag value 1

[3] The seed key in our c implementation is derived from the system time via some arithmetic operations such as modulo addition. We have also tried AES as the random source and obtained almost the same results as the RC4 case.

when the check is successful, and expect to see 4 consecutive `1,1,1,1` when running the above steps a few times. In the experiments, we indeed saw the expected pattern `1,1,1,1` after running the routine a few times. Thus, we have verified the essential point for verifying the correctness of the suggested attack. This fact, together with the following merging step, also explained the surprising phenomenon that we actually restored around 33 internal state bits from only 5 keystream bits in a probabilistic way.

Note that with the correct restricted internal states being in the candidate lists, the merging procedure will preserve them, thus the correct union partial state with probability 1. Precisely, we could merge the restored restricted internal states to get the CP part of the internal state from the 4 unified lists from $L_{z_0 z_1}$ to $L_{z_3 z_4}$. The preparation for the merging procedure takes about $T_1 = 2^2 \cdot 2^{13.93} \doteq 2^{15.93}$ cipher ticks. Further, we adopt the list merging process depicted in Figure 4 to restore the CP part of the internal state, whose time complexity is reduced to $T_2 = 2^{28.3}$ cipher ticks with approximately 1 MB memory, as discussed in section 4.6.

Next, the RP part of the internal state is restored by the dynamic guess-and-determine attack in section 4.7, which needs $2^{29.16}$ cipher ticks on average, which can be seen from the following deductions. First, the complexity of solving the $3y$ linear equations is at most 2^{12} when $y = 4$, and the subsequent steps take $2^{8.22}$ cipher ticks. The total complexity of the guess-and-determine attack is $\frac{2^{12+8.22}}{\Omega} = 2^{13.56}$ cipher ticks. The averaged number of trials to find the correct $S(101)$ is reduced to $2^{12.56}$ cipher ticks. Note that there are about $2^{16.6}$ candidates resultant from the merged CP state, so the complexity of the RP recovery is finally increased to $T_3 = 2^{16.6} \cdot 2^{12.56} = 2^{29.16}$ cipher ticks. The complexity of our attack is $T_1 + T_2 + T_3 = 2^{15.93} + 2^{28.3} + 2^{29.16} \approx 2^{29.79}$ cipher ticks. Since we choose $\lambda = 4$ to stabilize the success probability, the total complexity of our attack increase to $2^{31.79}$ cipher ticks.

We have run the above attack routines many times with different keys and frame numbers generated randomly, and the experimental results are always consistent with the theoretical analysis. One run of our attack is as follows. We use RC4 to generate the key `0xeab64598b32b32c4` and the frame number `0x3efd4c`, respectively. The produced keystream frame is `0x4a01e459770cdf81af52e70706a4c0` for one direction communication and `0xdefc02c7d0697294be821ae0f7adc0` for the other direction. The first 64 keystream bits are `0x4a01e459770cdf81`, where the first keystream bit z_0 is at the most significant bit position of a byte and z_7 is at the least significant bit position. Then we have $z_0 z_1 z_2 z_3 z_4 =$ `01001`. When one $l = 2$-bit keystream prefix $z_i z_{i+1}$ $(0 \leq i \leq 3)$ is used in our attack, it takes around more than one second to generate the pre-computed tables so far, stored in 198 bytes in RAM. Once constructed, these pre-computation tables can be used when the key or the frame numbers have changed. In the online phase, with the determined parameters `ksd=0x3`, $c = 4$, $\beta = 6$ and $\gamma = 2$, we could mount our attack step-by-step as presented above. The average size of the generated candidate list is 7835, while the theoretical size is $7880 \doteq 2^{12.95}$, which seems to imply that the practical implementation is more efficient to restore the correct restricted internal state than predicted in theory. We then ran the merging routine to restore the CP, which takes tens of seconds for the current non-optimized C implementation. The 4 correct restricted internal state 01 $\leftharpoonup$ 111000111010000, 10 $\leftharpoonup$ 000010000011010, 00 $\leftharpoonup$ 010011010001111 and 01 $\leftharpoonup$ 011011111101010 were simultaneously restored at the first invocation, which means that after the merging phase, the correct CP state 1110001110100000110100011111111010 would be recovered with probability 1. Finally, we retrieve the full internal state following the steps in Section 4.7. To improve the success probability, we set $\lambda = 4$ to retrieve the correct target internal state with a high probability. Note that the current implementation results are from the non-optimized code, and we think the attack can actually find the correct internal state, thus the secret key in a few seconds after future optimizations. The C language codes for verifying the validity of our attack are available via `https://github.com/martinzhangbin`.

We end this section by the comparison with other best known TMD attacks on A5/1, shown in Table 5. This table clearly shows the advantage of the new attack. Note that the (conditional) correlation attacks in [1, 7, 12] need much more keystream frames than our attack, though no long-term storage and no preprocessing are required.

Table 5. Comparison with the previous best known TMD attacks on A5/1

Attack	Data	Memory	Pre-comp.	Online time	Success rate
Nohl's Attack [13]	8 `frames`	1.7 `TB`	−`weeks`	10 `seconds`	87%
TMD Attack [11]	8 `frames`	1.968 `TB`	110`days`	9 `seconds`	81%
BB Attack [5]	$2^{14.67}$ `frames`	146 `GB`	$>$ 5 `years`	1 `second`	−
KP Attack [2]	4 `frames`	17.6 `TB`	$>$ 2300 `years`	5 `minutes`	$>$ 60%
Our Attack	$<$ 1 `frame`	1 `MB`	$2^{20.26}$ `ticks`	$2^{31.79}$ `ticks`	$\approx$ 99%

Note that the pre-computation phase of the Biased Birthday (BB) attack in [5] was extensively sampled rather than completely executed, i.e., there are non-trivial cases that the online time needed to find the correct internal state is much more than the claimed time.
KP means known plaintext.

Remarks. Since our attack is a known plaintext attack, we need first capture the first 64 keystream bits in 1 frame, which is always feasible in general in GSM networks. As shown in section 7 of [1], each traffic channel between the handset and the network is accompanied by a slower control channel, which is referred to as the Slow Associated Control CHannel (SACCH). The mobile uses the SACCH channel (on the uplink) to report its reception of adjacent cells. The network uses this channel (on the downlink) to send (general) system messages to the mobile, as well as to control the power and timing of the current conversation. The contents of the downlink SACCH can be inferred by passive eavesdropping. Besides, an attacker would still need to cope with the Frequency Hoping (FH) used by GSM, of which the hopping sequence can be determined through a quick exhaustive search due to the fact that given n, GSM defines only $64n$ hopping sequences (n cannot be large since the total number of frequencies in GSM is only about 1000, of which only 124 belong to GSM 900).

6 Leveraging the Attack to Any GSM Network

For completeness, we leverage our attack on A5/1 to other algorithms used in the GSM network in this section. It exploits the already detected flaw of the GSM protocols that the same key is used in different encryption algorithms. The key only depends on `RAND` in GSM, thus once we have restored the secret key of A5/1 by our method with ease, we are able to use this key to encrypt or decrypt for A5/3 or GPRS.

Precisely, the adversary can carry out a man-in-the-middle attack by impersonating the base station to the victim and the customer to the base station. In the initialization of a conversation, there is an authentication phase, which is a basic challenge-response scheme between the mobile phone and the network. The network sends a `RAND` to the attacker, who will transmit it to the victim then. The victim computes the `SRES` and return it to the attacker. In GSM protocols, the customer phone only reports the list of ciphers that it supports, while the network chooses which encryption algorithm is to be used. Hence, the attacker asks the victim to encrypt with A5/1 so that he could retrieve the key of A5/1 by our attack to encrypt in the conversation of the base station. Then the attacker returns the `SRES` computed by the victim to the base station, and the authentication is finished. When the base station asks the attacker to encrypt with A5/3 or GPRS, the attacker has already recovered the same secret key as used by A5/1, and has the ability to encrypt or decrypt using the restored key. Due to the fact that the authentication phase may not be initialized frequently, the key recovered from the previous conversation may be used in future conversations as well.

There is also a similar attack against GPRS, whose security is based on the same mechanisms as of GSM except that it uses `GPRS-RAND/GPRS-SRES` in the authentication and key agreement, and the GPRS cipher, referred to as GPRS-A5 or GPRS Encryption Algorithm (GEA), is different from A5/1 or A5/2 and is never made public so far. The attacker can take advantage of the symmetry in the key agreement of GPRS and GSM by performing an active attack on the customer phone via a fake base station. That is, he impersonates the network and starts a radio session with the mobile phone victim that is protected by A5/1. Accordingly, the resulting key is identical to the one that is used in GPRS and the attacker can recover it using our attack on A5/1. This means that the attacker can encrypt or decrypt the corresponding GPRS traffic successfully.

7 Conclusion

In this paper, we have proposed an entirely different cryptanalytic approach to break the A5/1 stream cipher without the need of large rainbow tables and a huge pre-computation phase. We have taken the new viewpoint of the fast near collision attack to restore the internal state of A5/1 in a divide-and-conquer manner. Based on the refined self-contained method, we could efficiently recover the restricted internal state of a given keystream prefix and merge several restored partial states together according to the irregular clocking mechanism in A5/1. Now the pre-computation becomes quite lightweight and cheap for individual cryptanalysts. It is shown that the irregular stop/go mechanism in A5/1 does not frustrate our attack as expected in the merging phase and we can reliably find the initial internal state, thus the secret key of A5/1 from only 1 keystream frame in $2^{31.79}$ cipher ticks. The total storage of our attack is about 1 MB, which is much less than all the previous best known TMD attacks. Due to the fact that A5/3 and GPRS share the same key as A5/1 in GSM, our attack can be converted into attacks against any GSM network eventually. It is well known that the analysis of stream ciphers based on irregularly clocked shift registers is a long standing problem in theory, our results shed some light on this issue and open new possibilities to deal with irregularly clocked shift registers.

Acknowledgements. The author would like to thank the anonymous reviewers for very helpful comments and Yanyi Liu, Hui Peng and Di Zhai for the discussions on the topic. This work is supported by the National Key R&D Research programm (Grant No. 2017YFB0802504), the program of the National Natural Science Foundation of China (Grant No. 61572482), National Cryptography Development Fund (Grant No. MMJJ20170107) and National Grand Fundamental Research 973 Programs of China (Grant No. 2013CB338002).

References

1. Barkan, E., Biham, E.: Conditional Estimators: An Effective Attack on A5/1. In: Preneel B., Tavares S. (eds) *Selected Areas in Cryptography–SAC 2005*. LNCS, vol. 3897, pp. 1-19, Springer, Berlin, Heidelberg, 2005.
2. Barkan, E., Biham, E., Keller, N.: Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. In: Boneh D. (eds) *Advances in Cryptology–CRYPTO 2003*. LNCS, vol. 2729, pp. 600-616, Springer, Berlin, Heidelberg, 2003.
3. Biham, E., Dunkelman, O.: Cryptanalysis of the A5/1 GSM Stream Cipher. *Progress in Cryptology-INDOCRYPT 2000*, LNCS vol. 1977, pp. 43-51, Springer-Verlag, 2000.
4. Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. in T. Okamoto, ed., *Advances in Cryptology–ASIACRYPT 2000*, LNCS Vol. 1976, pp. 1-13, Springer-Verlag, 2000.
5. Biryukov, A., Shamir, A., Wagner, D.: Real Time Cryptanalysis of A5/1 on a PC. *Fast Software Encryption–FSE 2000*, LNCS, vol. 1978, pp. 1-18, Springer, Heidelberg, 2001.
6. Briceno, M., Goldberg, I., Wagner, D.: A pedagogical implementation of A5/1. `http://www.scard.org`, May 1999.
7. Ekdahl, P., Johansson, T.: Another attack on A5/1. *IEEE Transactions on Information Theory*, vol. 49(1), pp. 284-289, Jan 2003.

8. Gendrullis, T., Novotný, M.: A Real-World Attack Breaking A5/1 within Hours. In: Oswald E., Rohatgi P. (eds) *Cryptographic Hardware and Embedded Systems–CHES 2008* LNCS, vol. 5154, pp. 266-282, Springer, Berlin, Heidelberg 2008.
9. Golić, J.D.: Cryptanalysis of Alleged A5 Stream Cipher, *Advances in Cryptology–EUROCRYPT'97*, LNCS vol. 1233, pp. 239-255, Springer-Verlag 1997.
10. Koch,P. C.: Cryptanalysis of Stream Ciphers-Analysis and application of the Near Collision Attack for stream ciphers. Technical University of Denmark, *Master Thesis–Supervisor: Christian Rechberger*, pp. 111-122, November 2013.
11. Lu, J., Li, Z., Henricksen, M.: Time-memory tradeoff attack on the GSM A5/1 stream cipher using commodity GPGPU. T. Malkin et al. (Eds.), *Applied Crptography and Network Security–ACNS'2015*, LNCS vol. 9092, pp. 350-369, 2015.
12. Maximov, A., Johansson, T., Babbage, S.: An Improved Correlation Attack on A5/1. In: Handschuh H., Hasan M.A. (eds) *Selected Areas in Cryptography–SAC 2004*. LNCS, vol. 3357, pp. 1-18, Springer, Berlin, Heidelberg, 2005.
13. Nohl, K.: Attacking phone privacy. In: Black Hat USA 2010 Lecture Notes (2010). `https://srlabs.de/decrypting-gsm/`
14. Pornin, Th. and Stern, J.: Software-Hardware Trade-offs: Application to A5/1 Cryptanalysis. In: Koç ç., Paar C. (eds) *Cryptographic Hardware and Embedded Systems–CHES 2000*. LNCS, vol. 1965, pp. 318-327, Springer, Berlin, Heidelberg, 2000.
15. Zhang, B., Li, Z., Feng, D., Lin, D.: Near Collision Attack on the Grain v1 Stream Cipher. In: S. Moriai (Ed.): *Fast Software Encryption-FSE'2013*, LNCS vol. 8424, pp. 518-538, 2014.
16. Zhang, B., Xu, C. and Meier, W.: Fast Near Collision Attack on the Grain v1 Stream Cipher. In: J. B. Nielsen and V. Rijmen (Eds.): *Advances in Cryptology – EUROCRYPT'2018*, LNCS vol. 10821, pp. 771–802, 2018.